Universidad Miguel Hernández de Elche

MÁSTER UNIVERSITARIO EN ROBÓTICA

“Programación de robots ABB para la realización de un proceso automatizado de Pick & Place de piezas del sector textil y cuero con reconocimiento por Visión”

Trabajo de Fin de Máster

Curso 2021-2022

Autor: Francisco José Martínez Peral
Tutor/es: Carlos Pérez Vidal
Jorge Borrell Mendez

AGRADECIMIENTOS

Primero de todo, agradecer principalmente a mis padres, por todo el apoyo incondicional que me dan en el día a día para seguir esforzándome al máximo, y también a mi familia y amigos, por estar siempre a mi lado.

Por supuesto, agradecer a Carlos Pérez Vidal la confianza depositada en mí para haber realizado este proyecto a lo largo del curso junto con una beca de colaboración en el departamento, así como por toda la ayuda e implicación que ha dedicado en el proyecto. De igual manera, agradecer a Jorge Borrell Mendez por ser co-tutor del proyecto, y por toda su ayuda, conocimientos y consejos compartidos.

Agradecer Universidad Miguel Hernández por su colaboración y poner a mi disposición los programas necesarios para realizar este proyecto, así como la licencia y un servidor para la utilización de la licencia del software de RobotStudio.

RESUMEN

El presente Trabajo de Fin de Máster se trata de la continuación del Trabajo de Fin de Grado desarrollado por el alumno. Se basa en la programación de una estación robótica para un proceso de Pick&Place de piezas del sector textil y cuero, realizando la simulación de dicha estación en el programa RobotStudio. Además, el trabajo incluye la creación de un algoritmo capaz de extraer datos de un fichero de AutoCAD, desde el que se obtienen información necesaria para poder conseguir una localización de las piezas que se van a recoger.

Dicha localización incluye el uso de Visión Artificial mediante la librería de OpenCV de Python. A partir de una imagen tomada desde la estación simulada, se identificará el lugar en el que se encuentra la pieza en la cinta transportadora, así como una comprobación de si la pieza se ha girado durante su movimiento al pasar del tapiz rodante de la CNC al de la cinta trasportadora. En el caso de que si se hay producido un desplazamiento o giro, los datos extraídos desde AutoCAD son recalculados para que no haya error a la hora de recoger las piezas por parte del robot.

Una vez hechas todas las comprobaciones, los datos son enviados desde el algoritmo de Python a RobotStudio mediante una comunicación vía Socket. El robot en la simulación de RobotStudio enviará una señal al algoritmo para que los datos de la siguiente pieza sean enviados para recogerla, y así sucesivamente, hasta que todas la plantillas sean recogidas y clasificadas según el modelo cortado.

Palabras clave: ABB, controlador, estación robotizada, proceso industrial, robot industrial, programación RAPID, RobotStudio, simulación, CNC, pieza sector textil, AutoCAD, Python, comunicación, socket, Visión artificial.

ABSTRACT

This Master's Degree Final Project is a continuation of the Bachelor's Degree Final Project developed by the student. It is based on the programming of a robotic station for a Pick&Place process for pieces in the textile and leather sector, carrying out the simulation of this station in the RobotStudio programme. In addition, the work includes the creation of an algorithm capable of extracting data from an AutoCAD file, from which the necessary information is obtained to be able to locate the parts to be picked up.

This localisation includes the use of Artificial Vision using the OpenCV Python library. From an image taken from the simulated station, the location of the part on the conveyor belt will be identified, as well as a check of whether the part has been rotated during its movement from the CNC conveyor belt to the conveyor belt. In the event that there has been a displacement or rotation, the data extracted from AutoCAD are recalculated so that there is no error when the robot picks up the pieces.

Once all the checks have been made, the data is sent from the Python algorithm to RobotStudio via Socket communication. The robot in the RobotStudio simulation will send a signal to the algorithm so that the data of the next piece is sent to pick it up, and so on, until all the templates are picked up and classified according to the cut model.

Keywords: ABB, controller, robotic station, industrial process, industrial robot, RAPID programming, RobotStudio, simulation, CNC, textile and leather sector templates, AutoCAD, Python, communication, socket, Artificial Vision.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Motivación

La automatización de muchos procesos industriales es cada vez mayor en todos los sectores de la industria. Gracias a la automatización de muchas tareas industriales, las empresas han conseguido una mayor optimización del rendimiento y de los tiempos de producción de sus procesos industriales, consiguiendo de esta manera, una reducción en la economía de costes.

La robótica se centra en la automatización y optimización de aquellos procesos de la industria que requieren de una gran producción y precisión, de forma ininterrumpida diaria, incluso, anualmente.

El presente proyecto se ha centrado en la optimización de un proceso industrial que podemos encontrar actualmente en el sector textil y del cuero, tanto en grandes empresas como en talleres de mediana o pequeña envergadura, como es el caso de un proceso de corte mediante una CNC o Centro Numérico Computarizado de distintos modelos de piezas usadas para confeccionar un zapato y su posterior proceso de recogida y clasificación según el modelo que se trate.

Con la automatización de este proceso se quiere conseguir una mejorar en la productividad, así como la reducción en costes. Pero además, se persigue sobre todo liberar a operarios de realizar tareas tan repetitivas y peligrosas, consiguiendo con un robots poder realizarla de forma segura, más rápida y precisa.

1.2. Objetivos

El trabajo tiene como objetivo principal la automatización del proceso de recogida y clasificación de piezas que han sido cortadas previamente por una máquina de control numérico computarizado o CNC.

Para ello, se ha desarrollado un prototipo funcional, el cual consiste en una estación basada en robótica y visión de una línea de corte de plantillas para zapatos mediante una maquina CNC.

Con este diseño se pretende conseguir que un robot sea capaz de realizar una tarea de Pick&Place de las plantillas, al mismo tiempo son clasificadas dependiendo del modelo de corte realizado.

A lo largo del proyecto, se han ido adquiriendo nuevos conocimientos para el diseño y la programación de una estación de Pick&Place mediante un robot industrial con el software de simulación de RobotStudio.

El proyecto se trata de un trabajo de investigación en el que se pretende obtener a partir de un fichero de AutoCAD, el cual es mandado a la CNC para cortar una plancha de cuero en distintas piezas, los datos de la posicion y rotación de cada una de las plantillas, así como el ángulo que está girada la plancha sin cortar y que será comparada con la Visión. Con un algoritmo creado en Python se recalculará los datos de las plantillas tras saber si la plancha se ha girado o movido a lo largo de la cinta. Posteriormente, los datos son enviados al robot mediante un socket entre la estación de RobotStudio y Python.

Con el algoritmo creado conseguimos una mayor optimización del proceso industrial, ya que de esta forma las posiciones son extraídas desde el fichero enviado a la CNC y a partir se comunica al robot donde debe de recoger las piezas de forma autónoma.

1.3. Resumen del trabajo

La idea del proyecto se ha organizado en diferentes fases, comenzando el proyecto de Trabajo Final de Grado llevado a cabo por el alumno, seguido de un proceso de investigación para la introducción de un algoritmo de visión por computador que sea capaz de detectar la pieza sobre la cinta transportadora y poder compararla con los datos del fichero CAD, y finalizando con el diseño de desarrollo de la simulación de la estación en RobotStudio.

En primer lugar, se ha continuado con el proyecto desarrollado por el alumno durante su Trabajo de Fin de Grado, el cual tenía muchos puntos en los que se podía mejorar su diseño, así como el de añadir nuevas aplicaciones para mejorar el proceso industrial. El primer paso era el diseño de la estación, donde se ha comenzado con la colocación de una CNC. A continuación de ella se ha colocado una cinta transportadora donde se encuentra un robot industrial de ABB anclado a un pórtico y donde se recogerán las piezas.

El robot se ha escogido un robot industrial de ABB, de características similares al UR10 de Universal Robotics, ya que se pretende realizar una comparación de ventajas y desventajas de usar uno tipo de robot u otro. El robot escogida es ideal para realizar la tarea, ya que al tratarse de un robot industrial, la velocidad de trabajo alta, así como la precisión a la hora de coger las piezas.

Tras haber realizado el diseño de la estación para el proyecto, el siguiente paso ha sido realizar un algoritmo capaz de extraer datos de cada uno de los datos de las plantillas, además de extraer datos de la plancha donde se cortarán las piezas. Los datos de la plancha serán utilizados en la parte de visión del algoritmo y tras recalcular los datos, estos serán enviado mediante un socket creado entre el programa de Python y RobotStudio.

En cuanto a las posiciones de dejada de las plantillas, estas han sido programadas en RobotStudio, y con la información enviado mediante el socket, se indica donde se debe de depositar la pieza según el modelo que se trate.

La programación de la estación se basa en distintos sensores colocado a lo largo de la cinta transportadora, con los que se indica tanto a los robots, a la cámara o al algoritmo de Python cuando deben de iniciarse.

Para la extracción de los datos de AutoCAD, se ha creado un script en AutoCAD con el que podamos extraer datos de una fichero .DWG. Con el algoritmo de e Python se ha podido iniciar AutoCAD, abrir el archivo y ejecutar el script que guardará los datos en dos ficheros de texto.

En uno de los ficheros tendremos los datos siguientes de cada una de las plantillas (modelos, coordenada X, coordenada Y y rotación), en el segundo fichero, tenemos los datos de la plancha grande donde son cortadas las plantillas, en dicho fichero nos interesa los datos del centro de gravedad de la plancha y además, se obtiene también los vectores que indican los ejes de los momentos principales de la pieza, los cuales se pueden comparar con los que se calculan con la visión.

Todos estos datos serán mandados al robot mediante una comunicación con un socket entre ambos programas, previamente se habrá hecho alguna modificación si se ha detectado que las pieza ha sido girada después de salir de la CNC, así como hacer una correspondencia entre el sistema de coordenadas de AutoCAD y el del robot en RobotStudio.

Finalmente, una vez diseñada la estación, programado el proceso tanto en RAPID, como el algoritmo en Python, se ha pasado a simular toda de la estación. Una vez simulada, realizaremos un estudio del resultado obtenido, así como compararlo con una estación de características similares pero haciendo uso de un robot colaborativo en vez de industrial.

Por último, veremos el resultado que se obtiene al simular el proceso completo.
Los programa utilizados durante el proyecto son los siguientes:

- ABB – RobotStudio 2020, versión de RobotWare 6.10.01.
- Autodesk AutoCAD 2020 versión para estudiantes.
- PyCharm Community Edition 2021.2.2 para la programación en Python.

1.4. Estructura de la Memoria del TFM

La memoria del presente Trabajo Fin de Máster está distribuida en ocho capítulos, en los que se explica en detalle la ejecución, desarrollo y programación de cada uno de los programas del proyecto y donde se va a dar una breve explicación en las siguientes líneas:

- **Capítulo 1. Introducción:** Se expone la motivación y el contexto del presente trabajo, los objetivos que sigue, un resumen y la estructura de la memoria para una mejor visión general de sus contenidos.

- **Capítulo 2. Estado del arte:** Se analiza el estado del arte, con una revisión de las herramientas y métodos aplicados en la industria, y los diferentes modelos de robots industriales y colaborativos que hay en el mercado.

- **Capítulo 3. Metodología:** Se realiza una pequeña introducción al programa de software RobotStudio ABB (versión 6.10.01), se aprenderá y creará una estación en un entorno de simulación. Donde se incorpora un robot y su controlador, creando un sólido y usando una herramienta de la biblioteca del programa. Se realizará la simulación de una sencilla tarea de pick and place, cogiendo la pieza en un lugar depositándola tras una pared que actuará como separador. Se explicará el uso de los SmartComponents, las señales para poder realizar la tarea y también, el apartado de aviso de colisiones que incorpora el programa de Robotstudio.

- **Capítulo 4. Diseño y simulación de una estación robótica con Visión para proceso automatizado de Pick & Place de piezas del sector textil y cuero:** En este apartado se va a explicar todo lo desarrollado, diseñado y programado en el presente proyecto. Se comenzará identificando cada una de las partes que se han introducido en la estación de RobotStudio, a continuación, se pasará a explicar el algoritmo creado en Python, tanto la parte de extracción de datos de AutoCAD, como la parte de Visión para poder calcular el ángulo que se ha girado la pieza durante su movimiento en la cinta. Finalmente, se explicará la programación en RAPID, la conexión de ambos programas y en envío de los datos desde el algoritmo hasta el robot.

- **Capítulo 5. Resultados:** Al finalizar la programación, se ha realizado la de simulación de esta, donde se puede observar por pantalla el correcto funcionamiento del sistema automatizado creado y se ha grabado para poder reproducirlo en cualquier momento.

- **Capítulo 6. Conclusiones y Líneas futuras de trabajos.** Se exponen las conclusiones del proyecto, comparándolas con los objetivos marcados. Además, se ha realizado una comparación en términos de tiempo de ejecución del proceso haciendo uso de un robot industrial o un robot colaborativo.

- **Capítulo 7. Bibliografía:** En este capítulo se indican todas las referencias bibliográficas utilizadas en este trabajo, ordenadas con código para su cita en la memoria, así como las páginas web y videotutoriales usados durante el proyecto.

- **Capítulo 8. Anexos:** En este último capítulo se añade información complementaria, como lo son los planos de los objetos diseñados en AutoCAD, las señales de entrada/salida utilizadas en las estaciones, los códigos de los programas creados y las hojas de características del robot utilizado.

2. ESTADO DEL ARTE

2.1. Introducción

Hoy en día, el entorno industrial es muy variado, por lo que un cambio constante en él es clave para alcanzar una producción más autónoma, rápida, económica y versátil. Como lleva siendo normal en los últimos años, la robótica es una de las principales soluciones utilizadas por las empresas para alcanzar estos objetivos.

2.2. Historia de la robótica

La palabra robot fue usada por primera vez en el año 1921, cuando el escritor checo Karel Capek estrenó en el teatro nacional de Praga su obra “Rossum’s Universal Robot”. El origen de la palabra robot viene de la palabra eslava “*robota”,* que se refiere al trabajo que es realizado de manera forzada.

Es posible que el término hubiera desaparecido si no hubiese sido por los escritores de ciencia ficción del siglo XX. Fue el escritor americano, de origen ruso, Isaac Asimov (1920-1992) el máximo impulsor de la palabra robot.

En octubre de 1945 fue publicada en la revista “*Galaxy Science Fiction”* su historia en la que por primera vez enunció las tres “*Leyes de la robótica*”:

1. Un robot no puede perjudicar a un ser humano, ni con su inacción permitir que un ser humano sufra daño.
2. Un robot ha de obedecer las órdenes recibidas de un ser humano, excepto si tales órdenes entran en conflicto con la primera ley.
3. Un robot debe proteger su propia existencia mientras tal protección no entre en conflicto con la primera o segunda ley.

La creación del término “robotics” (robótica) finalmente se le atribuyo a él, ya que gracias a su obra literaria, ha contribuido en gran medida a la divulgación y difusión de la robótica a lo largo de los años.

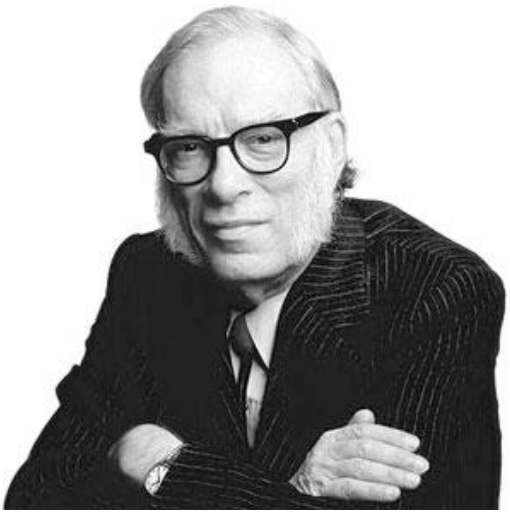


Figura 2.1: Isaac Asimov

Fue en 1956, cuando George C. Devol y Joseph F. Engelberger comenzaron a trabajar en la utilización industrial de las máquinas, fundando la empresa llamada Consolidated Controls Corporation, la cual más tarde acabó convirtiéndose en Unimation (Universal Automation).

Hacia el 1960, instalaron su primera máquina Unimate en la fábrica de General Motors de Trenton, Nueva Jersey, en una aplicación de fundición por inyección [1].

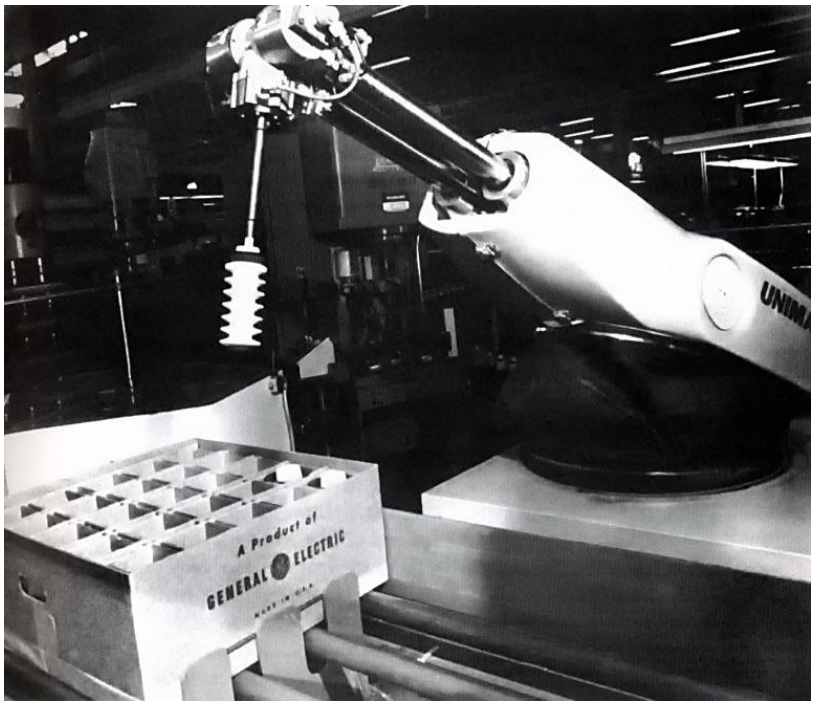


Figura 2.2: Robot Unimate en la planta de General Motors. (1961)

A partir de los años 80, ocurrió una explosión en el desarrollo de la robótica, considerando esta década como el inicio de la Era Robótica, donde su fabricación y venta aumentaron hasta un 80%.

Desde esta década, surgió la que hoy en día conocemos como la Robótica Inteligente, la que gracias al uso de la tecnología adecuada y la Inteligencia Artificial, ha hecho que los robots hayan ganado independencia al realizar tareas y cooperar con el ser humano, tomando decisiones en tiempo real y de forma autónoma.

La robótica inteligente ha supuesto una evolución de la robótica industrial gracias al uso de nuevas tecnológicas como el Big Data, el IoT y los sistemas de visión e inteligencia artificial.

2.3. Tipos de robots industriales

A la hora de hablar de robots industriales se toma como referencia a uno de los organismos internacionales más reconocidos como lo es la Federación Internacional de Robótica (IFR, International Federation of Robotics), el cual define las normas ISO.

Según la norma ISO 8373, nos referimos a un robot industrial como un manipulador multifuncional, controlado automáticamente y que pueden cumplir diferentes tareas, convirtiéndose en una herramienta indispensable a la hora de optimizar los procesos y ahorrar en gastos. También, se definen como reprogramables, ya que pueden ser calibrados constantemente para la tarea que se les asigna.

Lo que caracteriza a un robot industrial son:

- Los grados de libertad, es decir, el número de articulaciones que tiene.
- La zona de trabajo para operar el robot.
- La carga que deben y pueden sostener durante la tarea a realizar.
- La velocidad a la que pueden realizar las acciones.
- Su nivel de programabilidad.

A continuación, se pueden distinguir entre los distintos tipos de robots industriales que hay en la actualidad en la Figura 2.3.

Figura 2.3: Tipos de robots industriales

- **Robot cartesiano:** se trata de un robot que trabaja en coordenadas cartesianas, los cuales funcionan en los tres ejes (x,y,z) moviéndose en línea recta y formando ángulos rectos. Ofrecen una gran precisión, rentabilidad y facilidad en su programación y uso, por lo que son idóneos para movimientos lineales de gran precisión.

- **Robot SCARA:** (Selective Compliant Assembly Robot Arm) son conocidos por sus rápidos tiempos de trabajo, su elevada repetitividad, gran capacidad de carga y su amplio campo de aplicación.

- **Robot angular o antropomórfico:** estos robots tienen 3 articulaciones de posicionamiento y simulan los movimientos de un brazo humano. Podemos encontrar robots de 5, 6, incluso 7 ejes.

- **Robot Colaborativo:** se trata de un robot antropomórfico, pero de dimensiones y peso menor al descrito anteriormente. Son robots más manejables, de fácil programación y no necesitan medidas de portación específicas, ya que tienen sensores que le permiten trabajar en el mismo espacio que un humano.

- **AGV:** (Automatic Guided Vehicle) son robots que se identifican como vehículos autónomos, cuya aplicabilidad va muy ligada a la logística en la empresa.

2.4. Robots Colaborativos

Los robots colaborativos o “Cobots” están diseñados para poder trabajar junto con las personas, sin la necesidad de tener una jaula a su alrededor como es el caso de los robots industriales [2].

Una de las ventajas de estos robots es que son más asequibles que un robot industrial, y además son muy adaptables y sencillos de programas, por lo que en las pequeñas y medianas empresas están comenzando a invertir en ellos, lo que las está ayudando a crecer de forma masiva en los próximos años.

Hay una gran variedad de robots colaborativos y todos ellos vienen con un gran número de sensores integrados, articulaciones de fuerza limitada para una mayor seguridad, sensores de proximidad, lo que les permite trabajar juntamente con las personas en distintas tareas y así poder reaccionar rápidamente antes impactos producidos con el entorno o con las personas [3].

Estos robots no solo son aplicados en la industria y para tareas de fabricación. Son muchas las investigaciones que se están llevando a cabo para implantarlos en hospitales, laboratorios, hogares, almacenes, entre otros lugares.

Algunos de los robots colaborativos más utilizados en la industria son los siguientes:

- **ABB**

  Uno de los robots colaborativos de ABB es el conocido como Yumi. Este robot colaborativo tiene dos brazos a los que se les puede añadir manos intercambiables, sistemas de alimentación de piezas y localización con cámaras [6].

Figura 2.4. Robot colaborativo Yumi de ABB

Otro de los ultimo modelos que ha desarrollado ABB es el GoFa. Este modelos es uno de los últimos robots colaborativo creados por ABB, el cual puede ser usado junto a los trabajadores de forma segura y sin necesidad de crear una barrera o jaula. Tiene una fácil configuración y puede llegar a cargar hasta 5 kg con un alcance máximo de 950mm [7].

Figura 2.5. Robot colaborativo GoFa de ABB

- **Universal Robots**

  La empresa danesa Universal Robots es una de las más punteras a nivel mundial en robots colaborativos. Tiene una gama de diversos modeles de robots dependiendo de la carga y el alcance que se necesite. El más común es el UR10, el cual tiene unas características simulares a algunos d ellos robots industriales más pequeños. Puede cargar hasta 12,5 kg y tiene un alcance de 1300 mm. Se trata de un robot muy sencillo de programar y que cada vez se está implementando en un gran número de aplicaciones industriales [8].

Figura 2.6. Robot UR10e de Universal Robotics

- **OMRON**

La empresa Omron también tiene robots colaborativos, como por ejemplo el que se puede ver en la Figura 2.7. Robot Colaborativo de Omron. Es muy versátil y más sencillo de programar que sus versiones anteriores [9].

Figura 2.7. Robot Colaborativo de Omron

3. METODOLOGÍA

3.1. Introducción al Programa RobotStudio

Tal y como se desarrolla en [10], el programa RobotStudio es un software que permite diseñar, programar y simular estaciones de robots industriales de la marca ABB, el cual utiliza el lenguaje RAPID. El programa está diseñado y patentado por la empresa ABB, líder a nivel mundial en el sector de la robótica.

El programa RobotStudio se basa en un VirtualController de ABB, una copia exacta del software real utilizado por sus robots en producción. Por este motivo, permite programar los robots de manera offline (fuera de línea) en un ordenador, sin necesidad de parar la producción. Se pueden realizar simulaciones muy realistas, gracias que se pueden utilizar programas de robots reales y archivos de configuración idénticos a los utilizados en una instalación.

El programa proporciona una gran variedad de herramientas que ayudan a incrementar la rentabilidad de un sistema robotizado. Se pueden realizar tareas de formación, programación y optimización, sin llegar afectar a la producción en un empresa, lo que proporciona numerosas ventajas como por ejemplo, un arranque más rápido, el incremento en la productividad o la reducción de riesgos.

En la actualidad ya está disponible la versión 2022.1 de RobotStudio, pero para la realización de este Trabajo de Final de Máster se ha utilizado la versión de RobotStudio 2021.4 y la versión 6.10.01 de RobotWare. Para el uso del programa, ABB ofrece una versión de prueba de 30 días que se debe activa para trabajar con el programa, pero en el presente proyecto, la Universidad Miguel Hernández de Elche ha facilitado una licencia de red, conectada mediante un servidor de licencias de red, para poder usar todas las funciones del programa con la Licencia School.

3.1.1. Creación de una estación

Una vez iniciado el programa e instalada la licencia, el primer paso de este tutorial es crear una nueva estación. Para ello, en la pestaña “Archivo”, apartado “Nuevo” y en el apartado “Estaciones”, encontramos varias opciones para crear una nueva estación (o solución):

- **Solución con Estación vacía:** se crea una estructura de archivos de solución con una estación vacía, sin controlador.
- **Solución con Estación y Controlador de robot:** se crea una solución que contiene una estación y un controlador del robot ya iniciado, donde se debe de indicar el robot que se va a utilizar.
- **Estación vacía:** se crea una estación vacía.

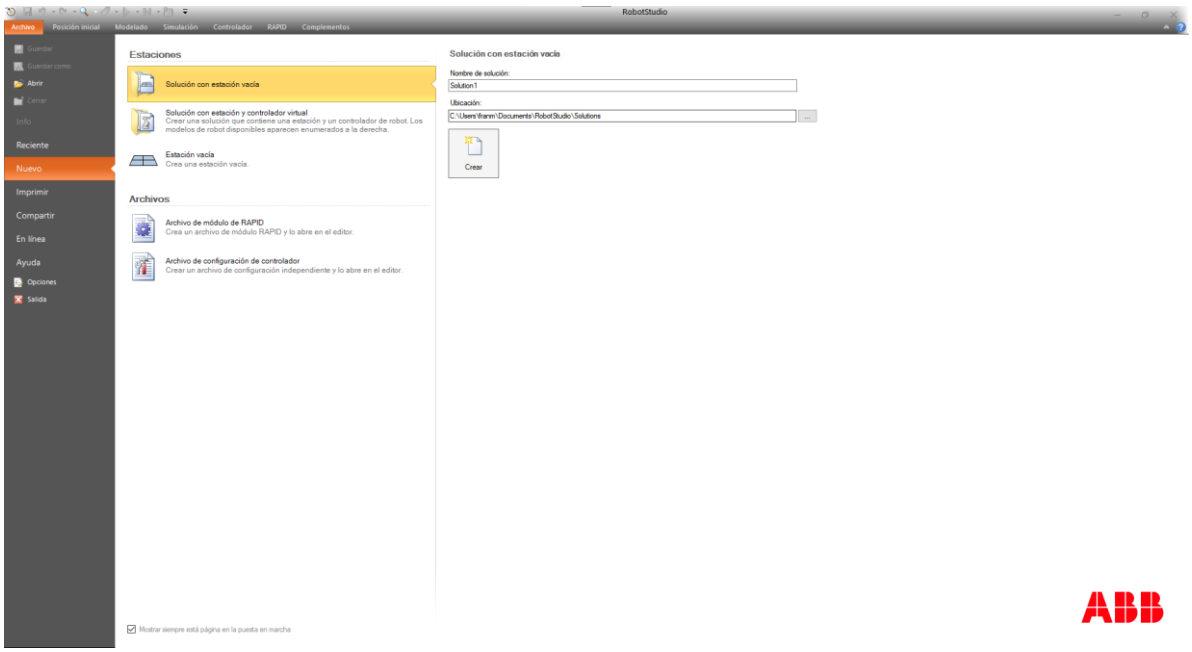


Figura 3.1: Creación de una nueva estación

El programa dispone de siete pestañas en la parte superior de la ventana:

- **Archivo:** encontramos las diferentes opciones básicas como son guardar, abrir, nuevo, compartir, imprimir y opciones del programa.

- **Posición Inicial:** se trata de la pestaña principal del programa, donde se puede añadir robot, controlador, bibliotecas, coordenadas e importar geometrías. Además de realizar la programación de trayectorias, seleccionar la herramienta y el espacio de trabajo, sincronizar, movimientos del robot y herramientas gráficas.
- **Modelado:** En esta pestaña se pueden crear grupos de componentes, componentes inteligentes, importar geometrías, crear sólidos, superficies y curvas, también podemos realizar operaciones de edición de CAD, así como crear mecanismos y herramientas para nuestra estación.
- **Simulación:** en este apartado se pueden crear conjuntos de colisión, editar la lógica de la estación, configuración de simulación, control de la simulación, monitorizar, analizar las señales y realizar la grabación de la simulación de la estación.
- **Controlador:** es la pestaña principal sobre el controlador, donde se puede añadir un controlador a la estación, reiniciarlo, editar señales de entrada/salida, acceder a la configuración del controlador y abrir la paleta FlexPendant.
- **RAPID:** pestaña donde se realiza la programación de los robots de la estación en leguaje RAPID.
- **Complementos:** Desde esta pestaña se pueden descargar algunos complementos e instalar paquetes como Painting, Machinig, Cutting, Palletizing.

Figura 3.2: Pestañas del programa RobotStudio

Para realizar el tutorial de esta introducción vamos a cargar el robot ABB IRB 120. Para ello, desde la pestaña “Posición inicial”, pinchamos en “Biblioteca ABB” y seleccionamos el robot que queremos, como podemos ver en la Figura 3.3.

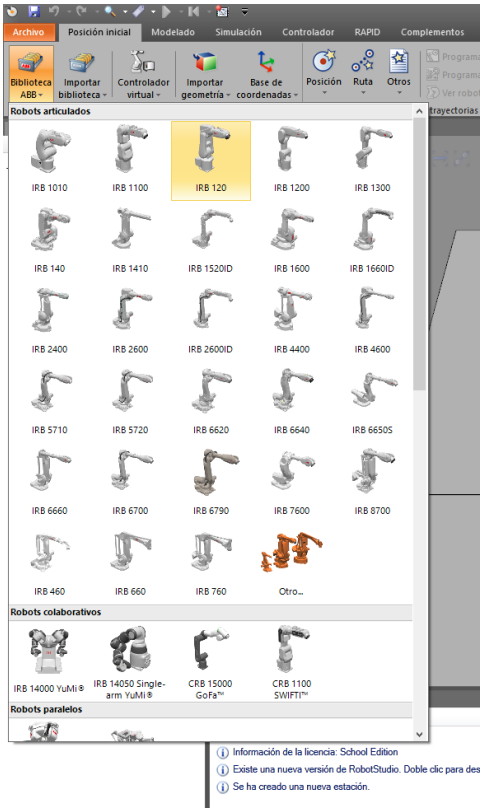


Figura 3.3: Robots de la Biblioteca ABB

Y seleccionamos el robot que queramos utilizar en el estación. En la Figura 3.4, se puede observar el robot situado en la estación.

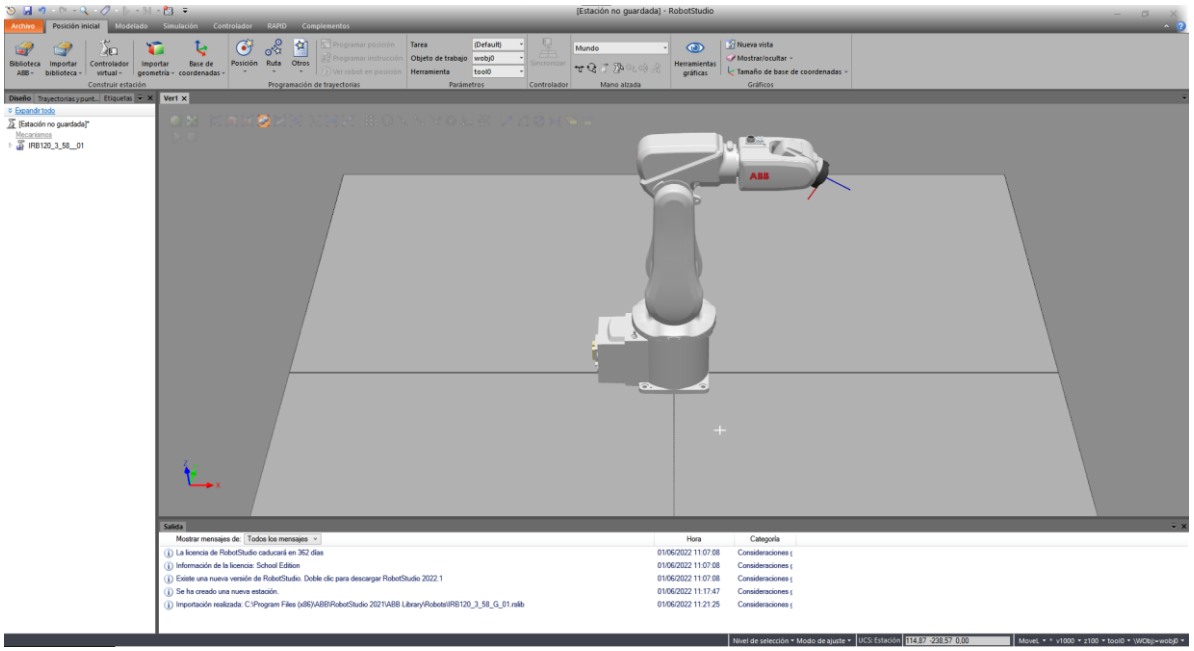


Figura 3.4: Robot ABB IRB 120 en la estación.

3.1.2. Creación de la herramienta de un robot

Para que un robot pueda realizar tareas en una industria, como procesos de pick and place, pintura, soldadura o pulido, necesitan de una herramienta como efector final del robot. A partir de dicha herramienta se crearán todas las posiciones y movimientos que debe de realizar el robot durante la simulación.

En el presente tutorial, vamos a crear una herramienta a través de la pestaña “Modelado”. Entre las diferentes figuras, podemos crear un tetraedro, cilindro, pirámide, esfera o un cono, que va a ser el que se va a utilizar. Pincharemos en “Sólido” y escogeremos “Cono”, como se muestra en la Figura 3.5.

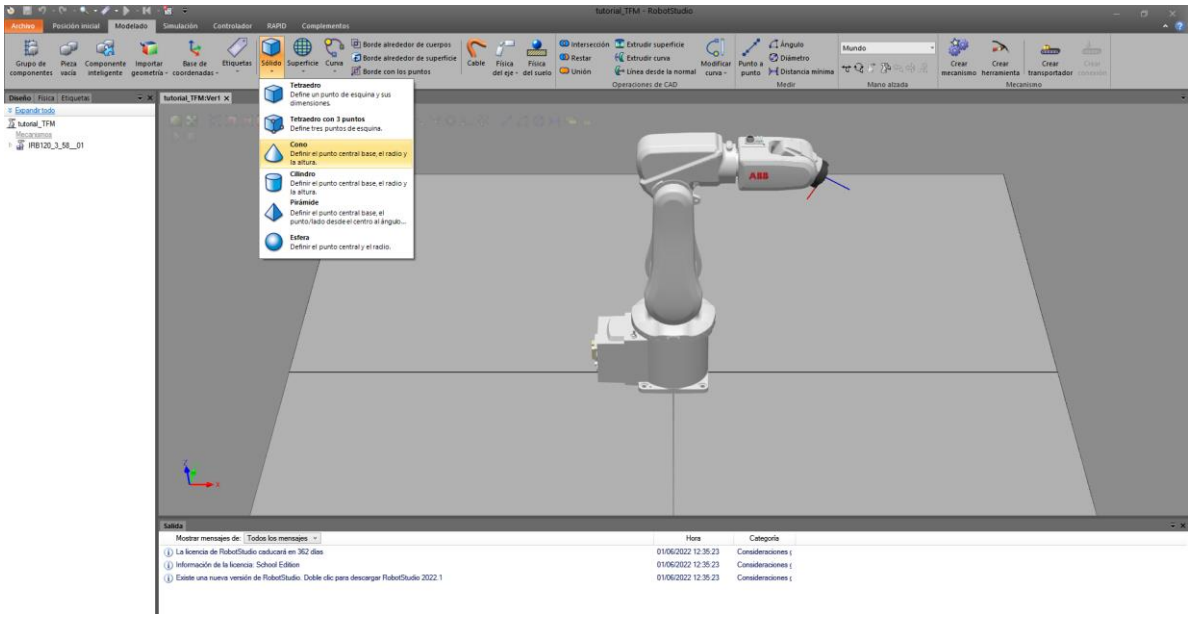


Figura 3.5: Creación de sólido (Cono)

A continuación, configuraremos los parámetros del cono con un radio de 25 mm, diámetro de 50 mm y una altura de 100 mm, y le daremos a “Crear”. Se creará un sólido en forma de cono en la posición (0,0,0), ya que no se ha especificado una posición determinada.

El siguiente paso consiste en transformar la geometría en forma de cono creada en una herramienta para el robot, este paso es necesario para que el programa interprete a la pieza como una herramienta del robot y se pueda conectar al efector final del robot.

Para ello, en la pestaña “Modelado”, seleccionamos la opción “Crear herramienta” de la figura 3.7. Lo primero que se debe de hacer es definir el nombre de la herramienta, en este caso la hemos llamado “Cono_tool”. Se debe de seleccionar la casilla de que se trata de una pieza existente y seleccionar aquella que se llama “Cono”.

A continuación, indicamos una masa de la herramienta y si sabemos el centro de gravedad y los momentos de inercia se pueden añadir a la información de la herramienta. Ahora, se puede añadir la posición y la orientación relativa al extremo de la herramienta, es a partir de esta posicion donde se crearán todas las posiciones robot. Esta posición se denomina TCP (Tool Central Point), y una vez definida su posición sobre la pieza, seleccionamos el boto de la flecha “→” y seleccionamos “Terminado”, como podemos ver en la Figura 3.6.

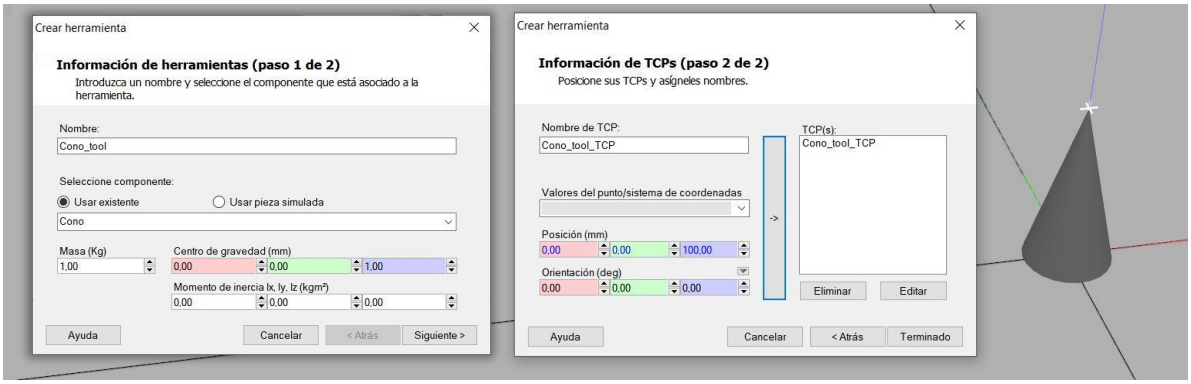


Figura 3.6: Configuración de la herramienta

Para finalizar, se debe de conectar la herramienta creada con el extremo del robot, para realizar esto podemos seleccionar la herramienta “Cono_tool” y arrastrarla al robot IRB120, a continuación, seleccionamos “Si” cuando pregunta si queremos actualizar la posición de la herramienta y automáticamente, la herramienta se encuentra conectada al robot, como podemos ver en la Figura 3.7.

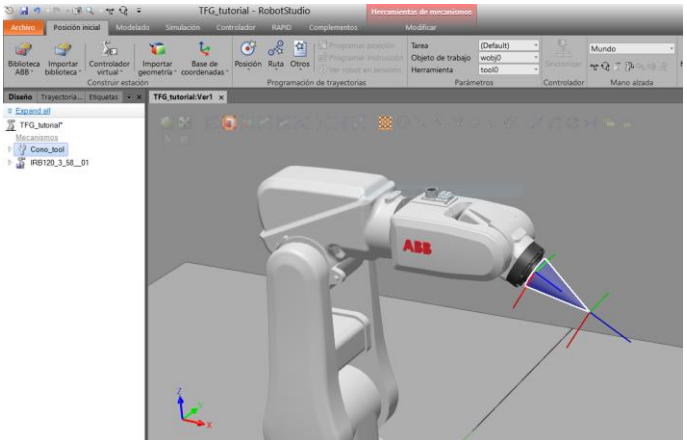


Figura 3.7: Robot IRB120 con herramienta conectada

También, tenemos la opción de “Importar geometría” en la pestaña de “Posición inicial”, con la que podemos añadir una herramienta creada en otro programa, como en la Figura 3.8.

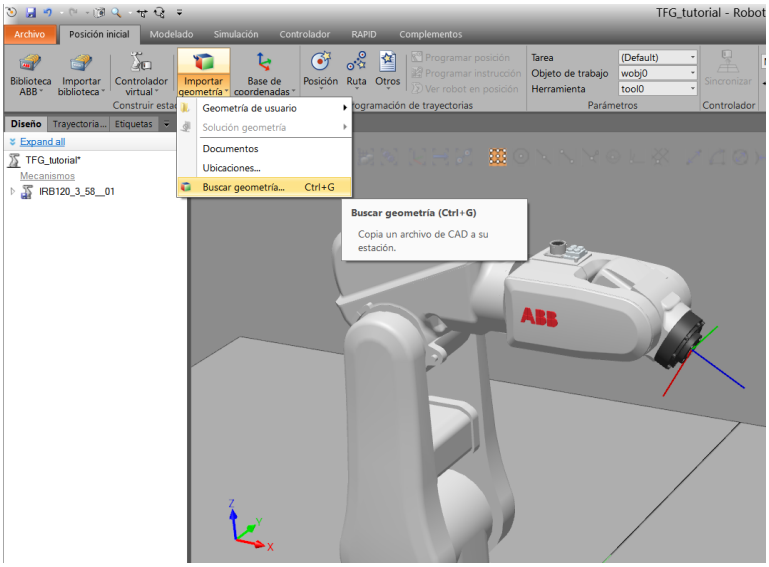


Figura 3.8: Importar geometría

Otra opción es la de añadir una herramienta de la propia biblioteca del programa, donde podemos encontrar algunas herramientas. En la pestaña de “Posición inicial”, haciendo clic en “Importar Biblioteca” y luego abriendo el desplegable de “Equipamiento”, vemos que hay variedad de herramientas y objetos que podemos añadir a nuestra estación. Podemos ver en la Figura 3.9 la biblioteca de RobotStudio.

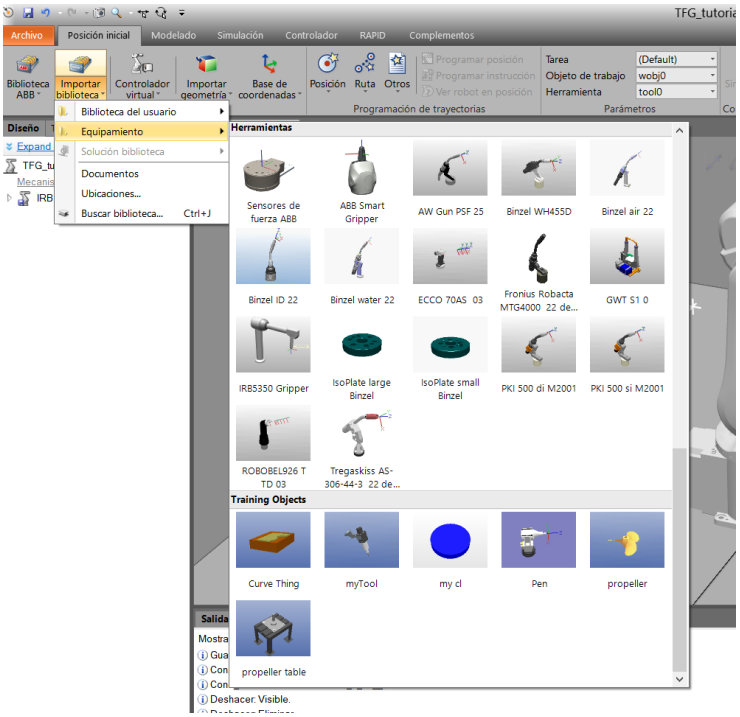


Figura 3.9: Biblioteca RobotStudio (Equipamiento)

Para el presente tutorial, se ha añadido como herramienta para realizar un programa de Pick&Place, la herramienta llamada “My_tool” y se ha conectado al robot como se ha realizado en los párrafos anterior, arrastrando la herramienta al robot. En la Figura 3.10 podemos ver el robot IRB120 con la herramienta conectada en su extremo.

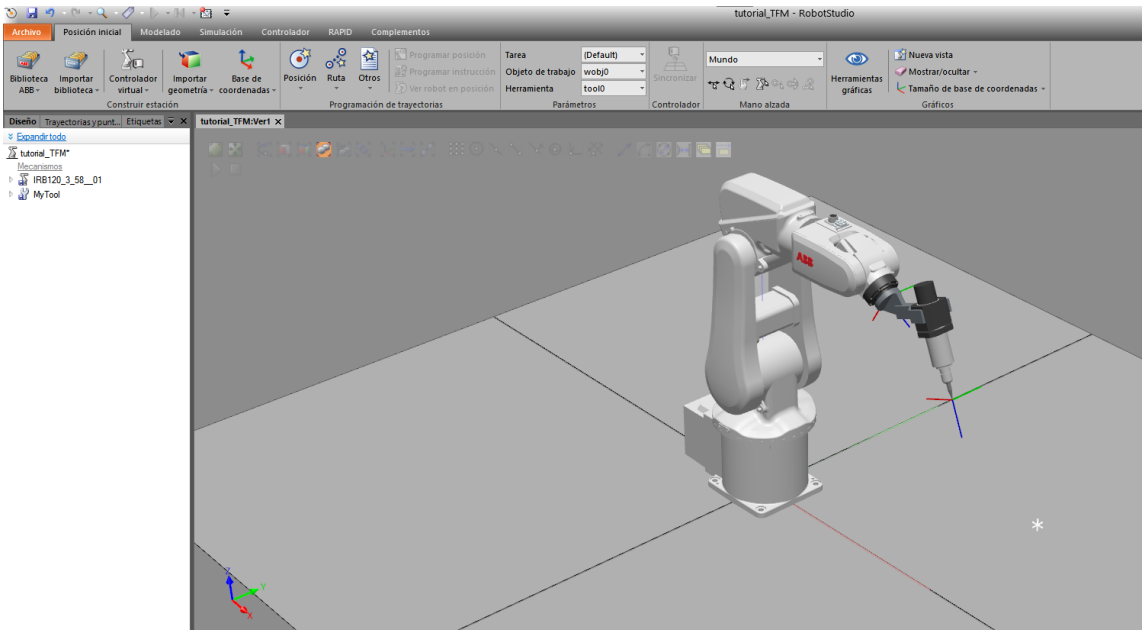


Figura 3.10: Robot IRB120 con herramienta

3.1.3. Creación del controlador de un robot

Hasta el momento, tenemos la herramienta “My tool” conectada al robot, pero aún no podemos programar trayectorias ni tampoco realizar movimientos con el robot. Para poder hacerlo, es necesario crear un controlador asociado al robot y dotarlo de cierta inteligencia. Para añadir el controlador, en la pestaña “Posición Inicial”, en “Controlador Virtual” seleccionamos “Desde diseño”, como se puede ver en la Figura 3.11.

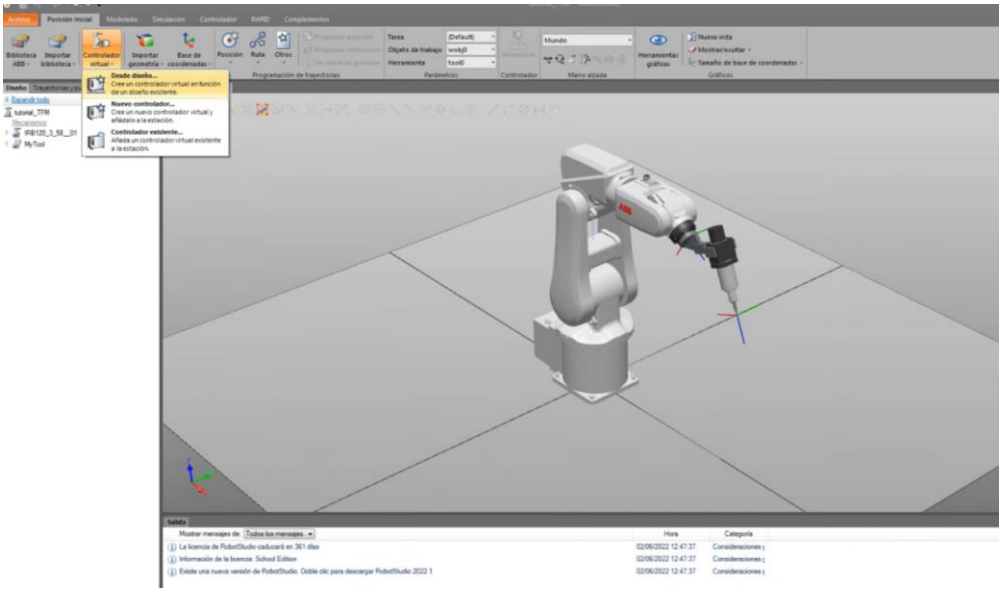


Figura 3.11: Creación de un controlador asociado al robot

Ahora, le damos un nombre al controlador y pulsamos en “Siguiente”, luego seleccionamos el mecanismo al que estamos creando el controlador. En este caso será el “IRB120_3_58__01” y pulsamos de nuevo en “Siguiente”, podemos ver este proceso en la Figura 3.12.

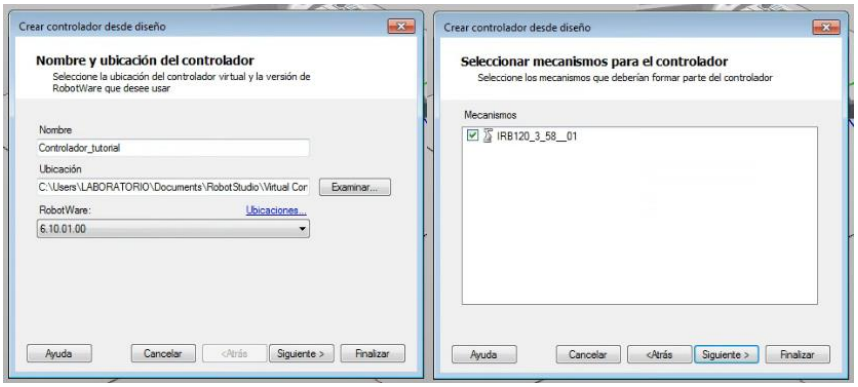


Figura 3.12: Creación del controlador

Para acabar de configurar el controlador podemos cambiar las opciones que viene por defecto en “Opciones…”, como por ejemplo podemos cambiar el idioma en el apartado “Default Language” a Español, luego le damos a “Aceptar” y por último a “Finalizar”, como podemos ver en la Figura 3.13.

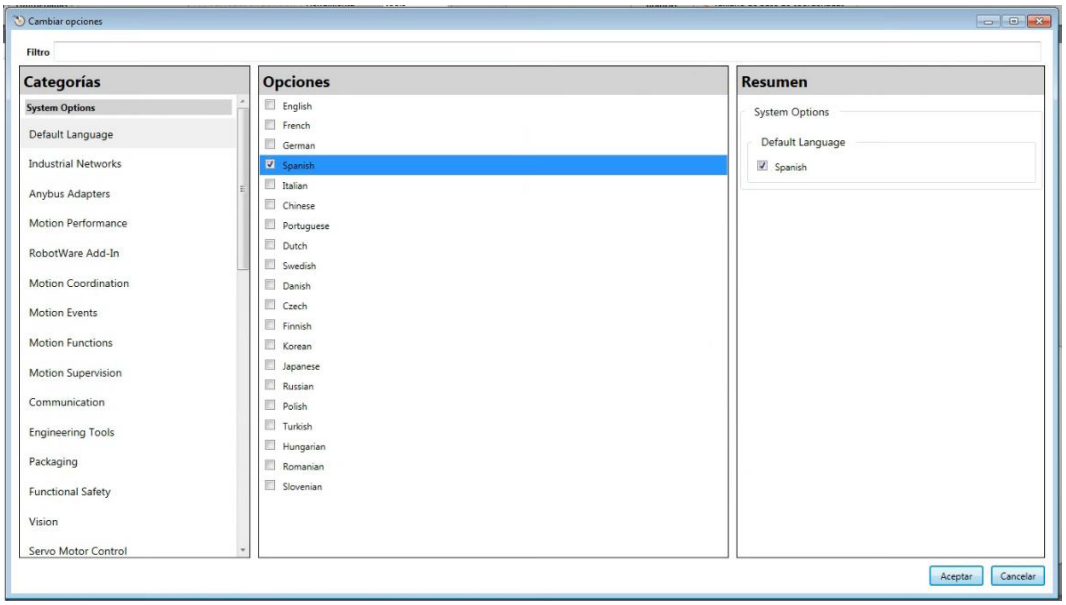


Figura 3.13: Cambiar opciones (idioma) del Controlador

Una vez realizado los pasos anteriores, ya tenemos creado el controlador del robot y se pueden crear posiciones y movimientos con el robot.

3.1.4. Creación de planos de trabajo y posiciones

Antes de crear el plano de trabajo y los objetivos a alcanzar por el robot, debemos de definir que objetos debe haber en la estación. La pieza que vamos a crear va a ser un sólido, en concreto se tratará de un tetraedro de lado y altura de 150 mm. Lo colocaremos en una posición que se encuentra a 400 mm en el eje X respecto de la base del robot (0,0,0) y a 200 mm en el eje Y, donde el robot pueda alcanzar el centro de la cara superior para coger la pieza. Le damos a “Crear” y le cambiamos el nombre a “Cilindro”. Se puede observar en la Figura 3.14 los datos introducidos al crear la pieza y donde se colocaría la zona de trabajo del robot.

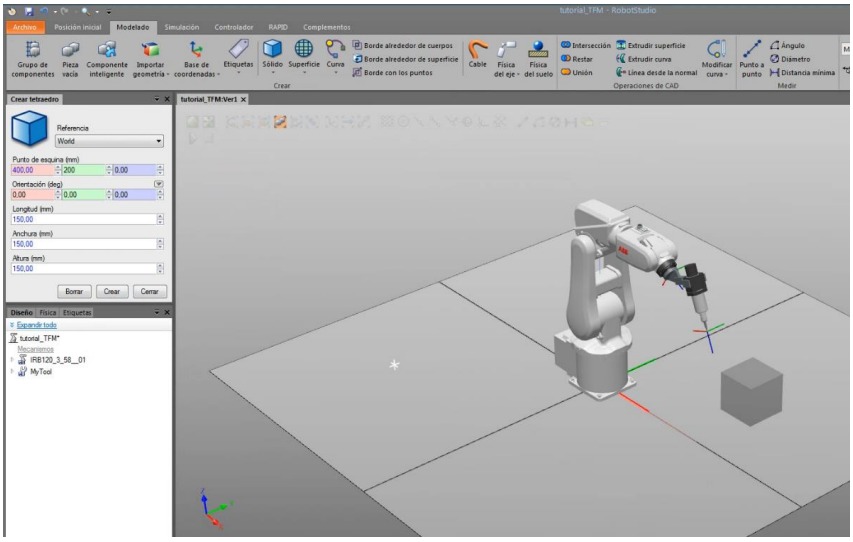


Figura 3.14: Creación de un tetraedro

En este tutorial se van a crear todas las posiciones y movimientos en el objeto de trabajo o Work Object que viene definido en la base del robot. Pero en algunas aplicaciones puede ser más interesante crear uno lo más cerca de la pieza posible y así tener organizados los puntos en distintos objetos de trabajo para el caso de diversas tareas del robot.

El siguiente paso es el de crear los objetivos o targets donde va a moverse el robot, pero antes de ello debemos de seleccionar el Objeto de trabajo y la herramienta sobra la que se van a definir los puntos. Para hacer esto, debemos de ir a la pestaña “Posición inicial”, y en “Parámetros” seleccionar los que se quieran utilizar, como podemos ver en la Figura 3.15.

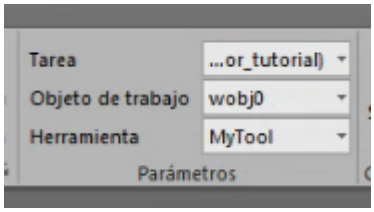


Figura 3.15. Selección de parámetros

Una vez seleccionados, se procede a crear las posiciones necesarias, en la pestaña “Posición inicial”, “Posición” y seleccionamos “Crear objetivo”, como vemos en la .

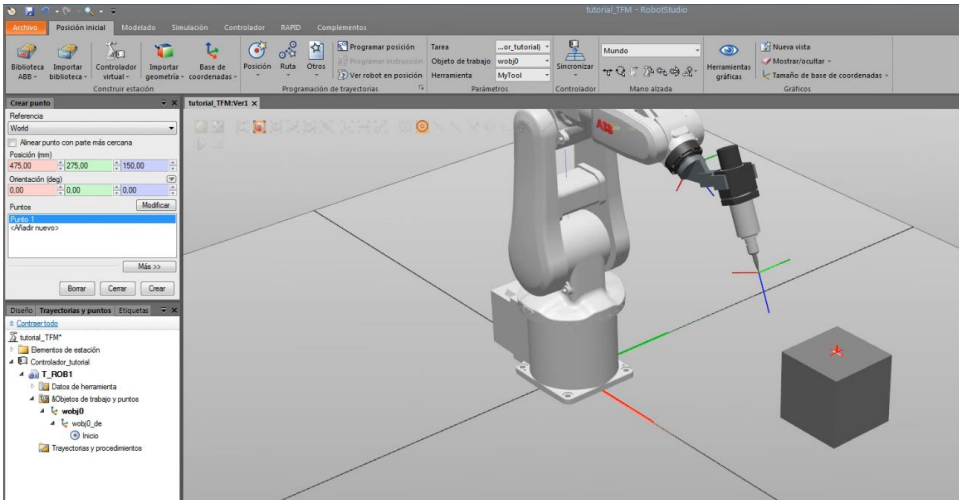


Figura 3.16: Crear punto de Pick.

La tarea que se va a realizar en este tutorial es un proceso sencillo de Pick & Place, para ello, se necesita una posición inicial desde donde partirá el robot, una posición de pick y de aproximación de pick y luego, una posición de place y también de aproximación.

Por lo tanto se ha creado una primer posición de Pick y a partir de ella se van a crear las demás. La posición de Place tendrá las mismas coordenadas pero la coordenada Y será -275 mm. En el caso de las posiciones de aproximación se modificará la altura para que los movimientos hacia el Pick y el Place sea lineal y a menor velocidad. En la siguiente , se pueden observar los 4 puntos creados para realizar la tarea.

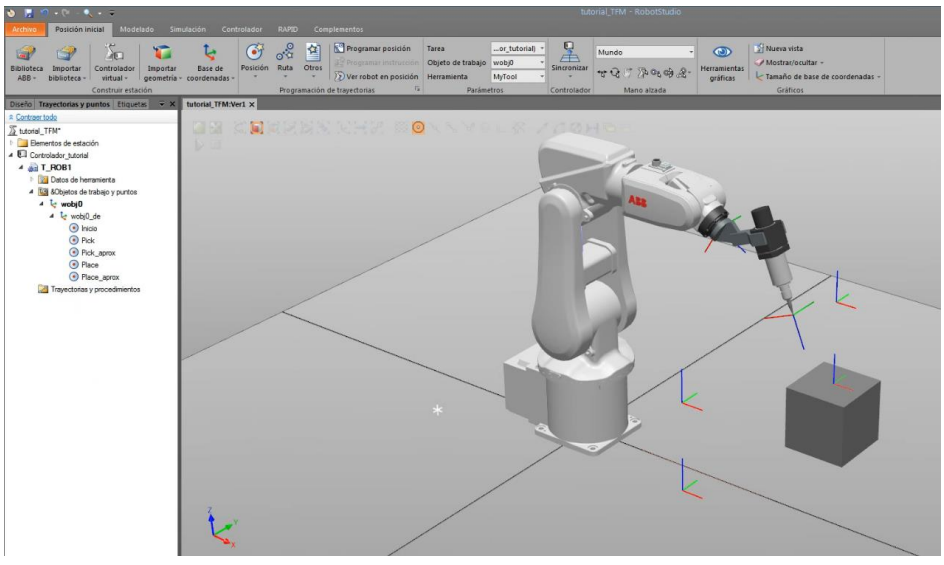


Figura 3.17. Posiciones creadas para la tarea

3.1.5. Creación de trayectorias

Una vez creados los puntos para realizar la tarea, vamos a visualizar la herramienta “My tool” en las posiciones de los puntos creados. La orientación que viene por defecto puede que no sea la idónea o que no pueda ser alcanzada por el robot, por lo tanto se escogerá el primer punto creado como “Pick”, se hace clic con el botón derecho del ratón y seleccionamos “Ver herramienta en la posición” y pinchamos en “My_tool”, ahora la herramienta se visualiza en el punto que hemos creado.

Para colocar la herramienta con la orientación con la que pueda ser alcanzada por el robot, será necesario realizar un giro al sistema de eje de coordenadas. Se hace clic derecho con el ratón sobre “Pick” otra vez, seleccionamos “Modificar posición” y pinchamos en la opción girar. Se debe de realizar un giro hasta llegar a una posición donde el robot alcance el objetivo. En este caso, se realiza un giro de 180 grados en el eje Y. Con la opción “Ver robot en posición” podemos ver que el robot alcanza el objetivo, como se puede ver en la Figura 3.18.

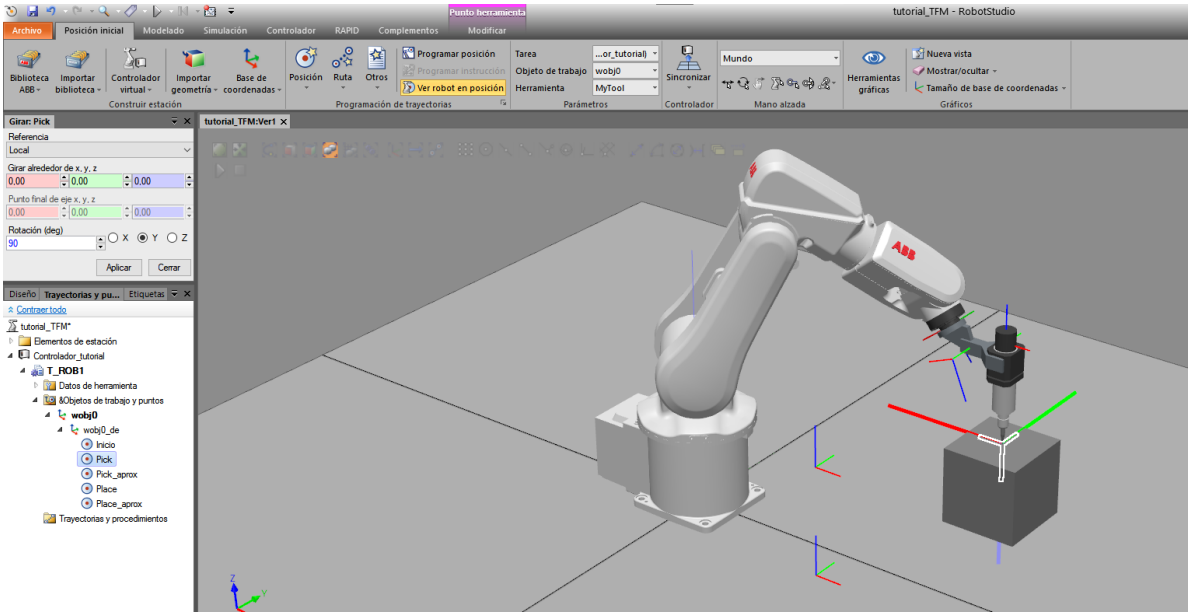


Figura 3.18: Giro de 180º en el eje Y con robot en posición

Tras comprobar que el robot alcanza dicho punto, se debe de hacerlo los mismo para el resto de los puntos. Una forma sencilla de hacerlo es copiar la orientación del punto que acabas de editar y copiarla en el resto de las posiciones.

Para ello, se selecciona el punto “Pick”, se hace clic con el botón derecho y “Copiamos orientación”. Después, se seleccionan el resto de los puntos y “Aplicamos orientación”. En la Figura 3.19 , se puede ver que todos los puntos tienen la misma orientación viendo a la herramienta en dichos puntos.

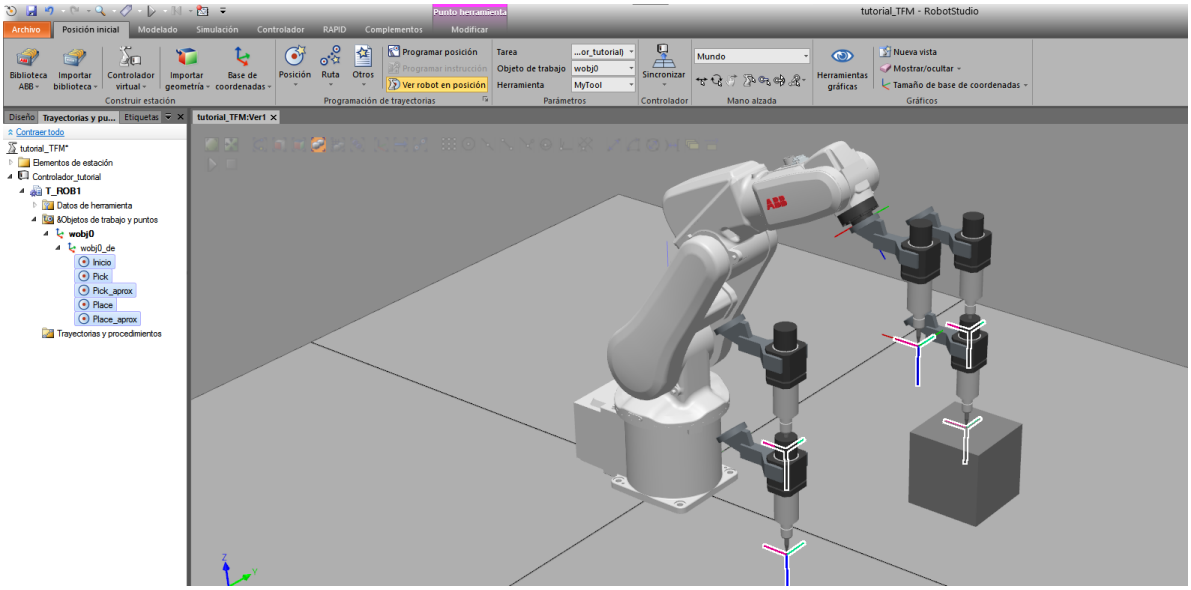


Figura 3.19. Orientación de la herramienta en todos los puntos

Ahora, se debe de crear la trayectoria y para ello tenemos dos opciones de hacerlo. La primera es crear una trayectoria vacía, en la que se deben de añadir los objetivos a la trayectoria y la otra, es crear una trayectoria automática en la que necesitaremos una geometría o curva. En nuestro caso vamos a realizar una “Trayectoria vacía”, seleccionándolo en la pestaña “Posición Inicial” y en la opción “Ruta”.

En el presente tutorial vamos a realizar una trayectoria desde la posición de Inicio hasta la de Pick, y a continuación, desde la posición de Pick hasta la de Place pasando por los puntos de aproximación.

Para crear una trayectoria vacía, en el apartado de trayectorias y botón derecho y crear “Trayectoria Vacía”.

Para crear una trayectoria hay varias opciones que se pueden cambiar, como son el tipo de movimiento, la velocidad, la precisión, la herramienta y el objeto de trabajo. Para cambiar algún dato, se selecciona un punto o varios de la trayectoria y en la parte de abajo a la derecha de la ventana del programa tenemos las siguientes opciones:

- **Plantilla de Instrucciones:** donde podemos cambiar el tipo de movimiento a MoveL, MoveJ o MoveAbsJ, en el apartado 3.1.6 veremos que significa cada tipo de movimiento.
- **Speed:** en este apartado podemos cambiar la velocidad con la que se mueve el robot, por defecto aparecerá v1000, siendo la velocidad en mm/s.
- **Zone:** esta opción se refiere a la precisión del movimiento, los valores referencian al error en distancia (en mm) que hay desde el TCP de la herramienta del robot al punto objetivo en el momento en el que se ejecuta la instrucción y el robot está pasando por dicha posición. Donde fine es el más preciso y z200 el menos preciso, por defecto aparecerá z100.
- **Herramienta:** se podrá escoger entre las herramientas que tengamos asociadas como herramientas al robot.
- **Objeto de trabajo:** el último apartado podremos cambiar el objeto de trabajo.

Figura 3.20: Cambio de parámetros en las trayectorias

3.1.6. Introducción a RAPID

Tal y como se desarrolla en [11], RAPID es un lenguaje de programación textual de alto nivel desarrollado por la empresa ABB. Una aplicación RAPID consta de un programa y una serie de módulos del sistema. El programa es una secuencia de instrucciones que controlan el robot, en general consta de tres partes:

- **Rutina principal (main):** es la rutina principal donde se inicia la ejecución.
- **Conjunto de subrutinas:** estas rutinas sirven para dividir el programa en partes más pequeñas con el fin de obtener un programa dividido en módulos
- **Datos del programa:** en esta parte se definen posiciones, valores numéricos, sistemas de coordenadas, variables, datos de herramientas, etc.

Para realizar la programación en RAPID, en primer lugar hay que sincronizar la estación con el controlador virtual. Para realizar esto en la pestaña “Posición Inicial”, seleccionamos “Sincronizar” y pinchamos en “Sincronizar con Rapid”, por último, daremos “Aceptar” como podemos ver en la Figura 3.21.

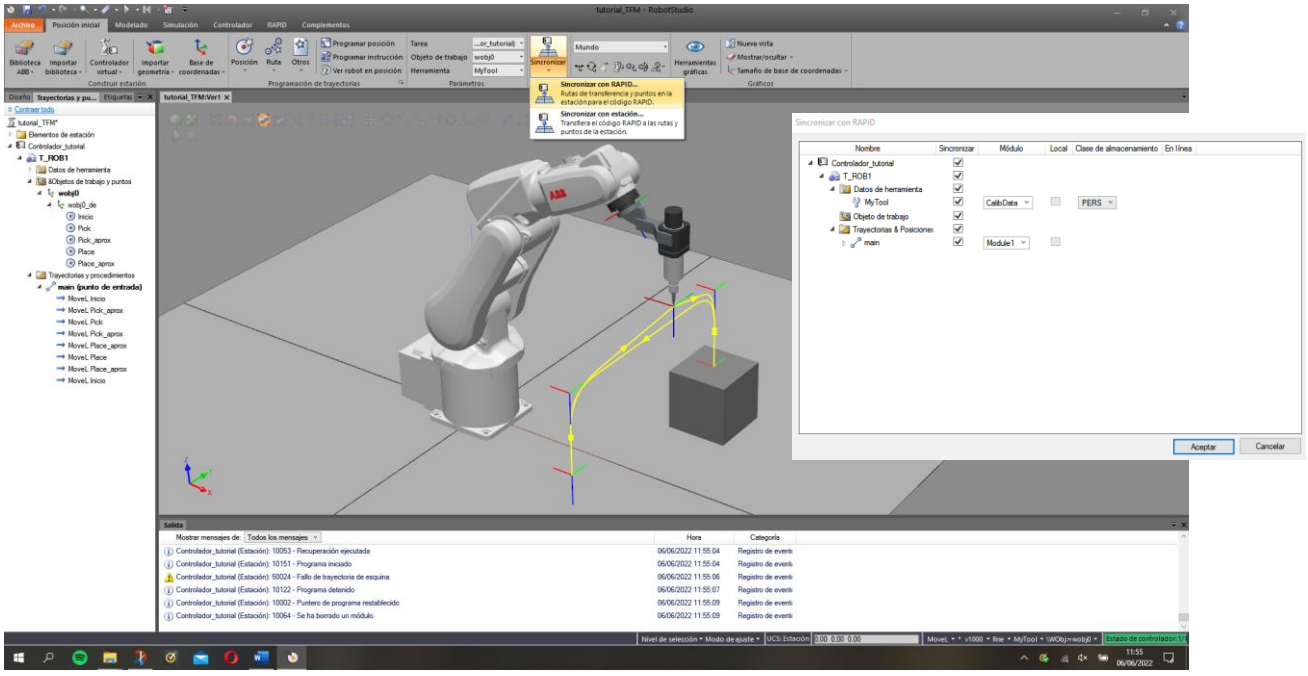


Figura 3.21: Sincronización con RAPID

Al sincronizar se consigue que los datos de la herramienta, objeto de trabajo con todos los objetivos y las trayectorias creadas sean volcados al programa, apareciendo los módulos de programa “CalibData” y “Module1” necesarios para la programación del robot en RAPID.

Ahora se pasa a la parte de programación, para ello debemos ir a la pestaña de “RAPID” y dentro del controlador, abrimos el desplegable llamado “Rapid” y luego “T_ROB1”, haciendo doble clic derecho “Module1” podemos ver, gracias a la sincronización que hemos realizado previamente, definidos los puntos como constantes robtarget y también las trayectorias con sus movimientos.

El programa esta englobado en el módulo del programa llamado “Module1”. Para definir el módulo, tenemos una instrucción inicial “MODULE Module1” y una instrucción final con “ENDMODULE”.

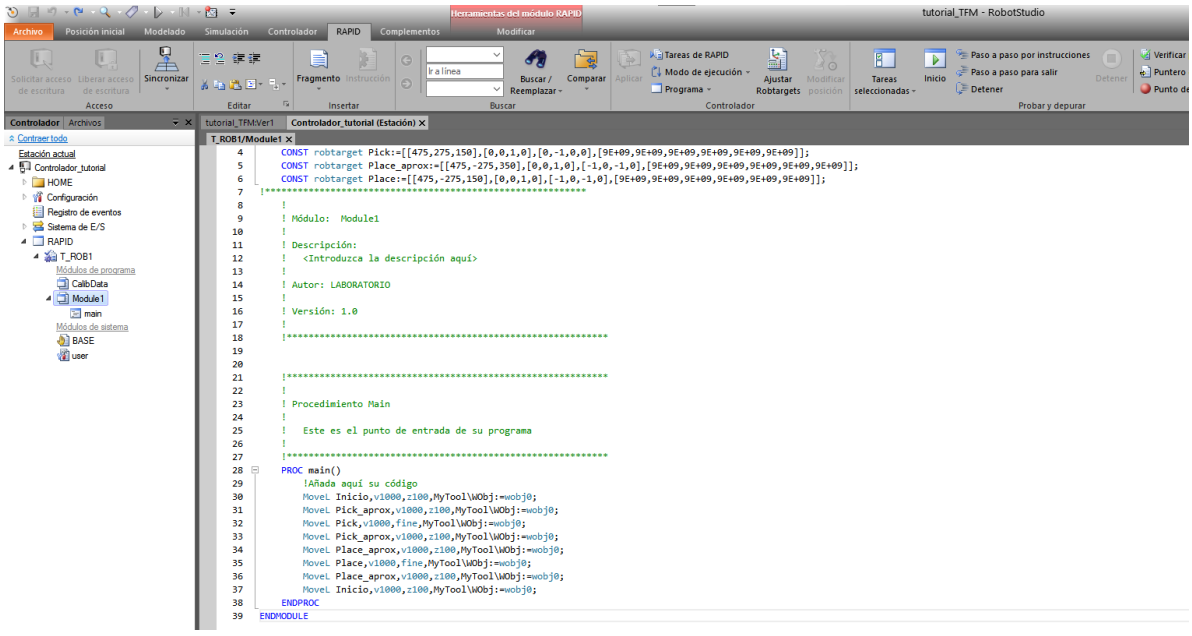


Figura 3.22: Programa RAPID (Module1)

Una vez tenemos el código de RAPID creado y se han definido las instrucción de movimientos en el orden en el que queremos que el robot realice la terea, se puedo pasar introducir aquellas instrucción relacionadas para el uso de la herramienta como “ventosa” para que coja y suelte las piezas.

Para ello, se va a comentar las principales instrucciones para el uso de señales.

- **Instrucciones para la utilización de entradas y salidas**

Dos tipos de operaciones importantes que van a ser usadas durante la simulación son comprobar el valor de una entrada y fijar el valor de una salida.
Para fijar el valor de las salidas utilizamos las siguientes instrucciones:

- **Set:** fija el valor de una salida digital a 1.
- **Reset:** fija el valor de una salida digital a 0.
- **SetDO:** fija una salida digital a un valor simbólico (activado o desactivado).
- **SetAO:** fija el valor de una salida analógica.

La instrucción principal para comprobar el valor de una entrada es la instrucción **WaitDI** (por ejemplo, WaitDI Pieza,1;), esta instrucción hace que el robot espere hasta que la señal “Pieza” tenga el valor indicado. También comprobamos el valor de una señal de entrada con instrucciones de comparación, como lo son las instrucciones IF señal = 1 THEN…, IF señal < 5 THEN…

- **Instrucciones de control de flujo de ejecución**

Son instrucciones que también son utilizadas en otros lenguajes de programación:

- **IF ___ THEN:** ejecuta una serie de instrucciones si se cumple una determinada condición indicado detrás del IF.
- **FOR:** se trata de un bucle que repite una sección del programa un determinado número de veces.
- **WHILE:** se trata de un bucle que repite una sección del programa mientras se cumpla una condición dada detrás del WHILE.
- **TEST/CASE:** ejecuta diferentes instrucciones en función del valor de un dato (similar al switch-case del lenguaje C).
- **GOTO:** salto incondicional a un punto del programa, debemos de definir el punto al que debe de saltar anteriormente con “nombre_punto_salto:”

- **Variables y expresiones**

El lenguaje RAPID permite definir variables u otros tipos de datos. Definimos las variables para crear expresiones aritméticas o lógicas mediante una serie de operadores comunes (suma, producto, comparación…). Los principales tipos de datos en RAPID son ConfData, JointTarget, Load Data, MotSetData, Num, Orient, Pos, Robjoint, Robtarget, StopPointData, ToolData, ZoneData.

Prosiguiendo con el tutorial, se debe de cambiar la precisión “Zone” de las trayectorias para que tenga mayor precisión, es decir, aquellas instrucciones que muevan al robot a las posiciones de Pick y de Place, para ello hemos cambiado el valor “z100” a “fine”. Tras realizar cambios en el código de RAPID, se debe de actualizar con la estación, para que se apliquen los cambios sobre las instrucciones.

Si vamos a la pestaña de “Posición Inicial”, se puede ver que todos los cambios se han realizado y que ahora el robot alcanza las posiciones de dejada y de recogida sin error, como podemos ver en la Figura 3.23.

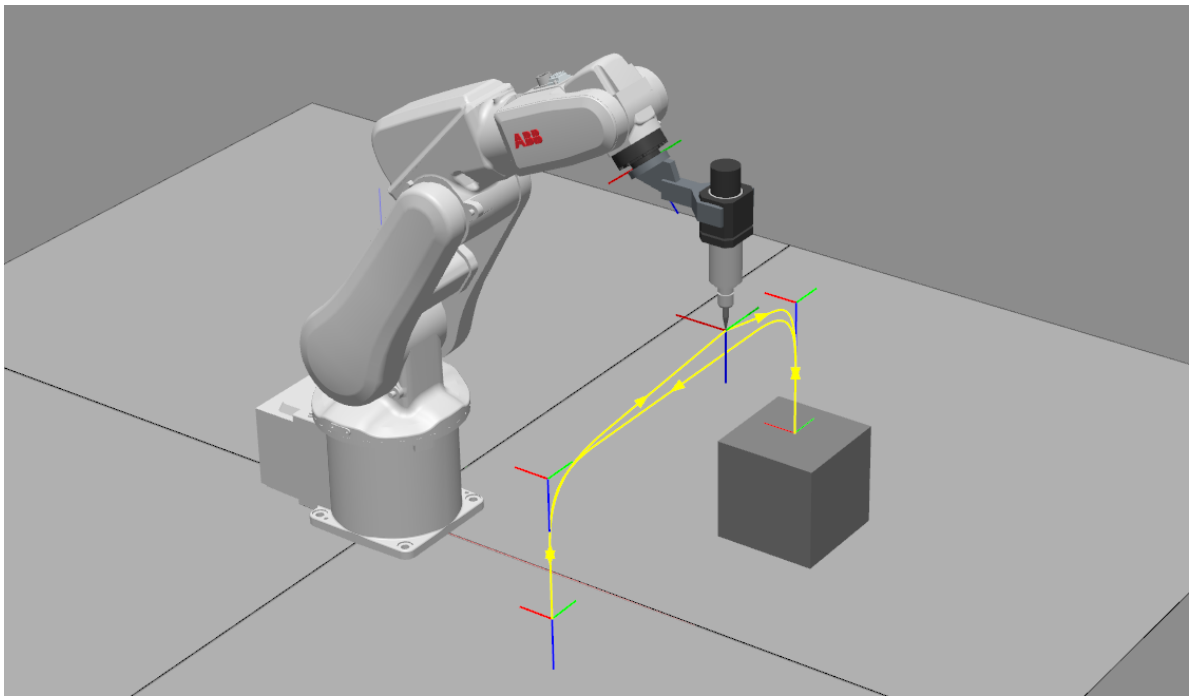


Figura 3.23: Mejora de la precisión de los movimientos

3.2. Diseño y programación de una Estación

3.2.1. Diseño en AutoCAD de la geometría de la estación

En primer lugar, para el diseño de la estación se han de crear los sólidos que la componen, pero el programa RobotStudio no es suficiente ni a la hora de crear un sólido de diseño ni con los objetos que ya vienen incluidos en el programa, como son las cintas transportadoras, mesas, vallas…

En este trabajo se ha utilizado el software de diseño AutoCAD 2020, para realizar el diseño de las piezas necesarias. El conjunto de planos de las piezas que se han creado con AutoCAD y que posteriormente se han importado a RobotStudio se encuentran en el Anexo 8.1 de este trabajo.

El proceso para insertar las piezas en el programa de RobotStudio es el siguiente, lo primero que debemos de hacer es exportar desde el programa AutoCAD la pieza que hemos creado, en el icono de AutoCAD a la izquierda, vamos a “Exportar” y seleccionamos como “Otros formatos”. Aparecerá una venta emergente donde se debe de seleccionar el tipo de archivo (en este caso seleccionamos “.sat” , que es el tipo de archivo que admite RobotStudio) y el lugar donde debemos de guardarlo, por último, le damos a Guardar y seleccionamos el sólido que forma la pieza, se puede ver el proceso en la siguiente Figura 3.24.

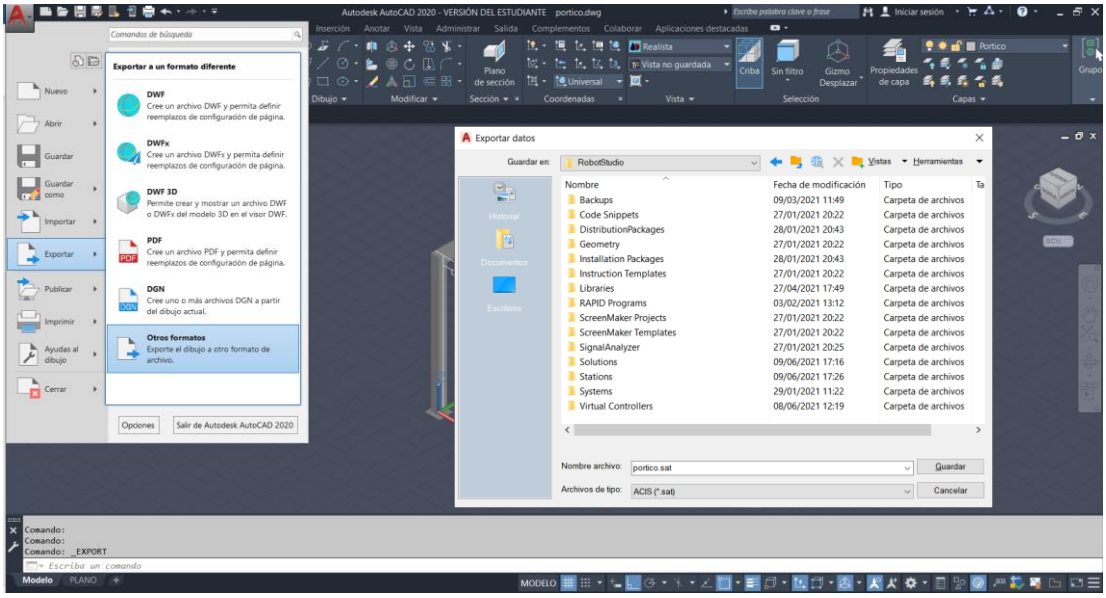


Figura 3.24: Exportar sólido desde AutoCAD

Una vez creado, vamos a RobotStudio y para añadirlo debemos de ir a la pestaña “Posición inicial”, seleccionamos “Importar Geometría” y hacemos clic en “Buscar Geometría”, se seleccionan el fichero .sat con la pieza que queramos importar y se abre.

3.2.2. Creación y diseño de los Smart Components

Los Smart Components (SC) o componentes inteligentes, son elementos asociados a los sólidos, piezas o robots que tenemos en nuestra estación. Los cuales tiene un comportamiento controlado por señales y propiedades del sistema.

En la Figura 3.25, podemos ver como se crear un Smart Component, en la pestaña de “Modelado” y hacemos clic en “Componente Inteligente”.

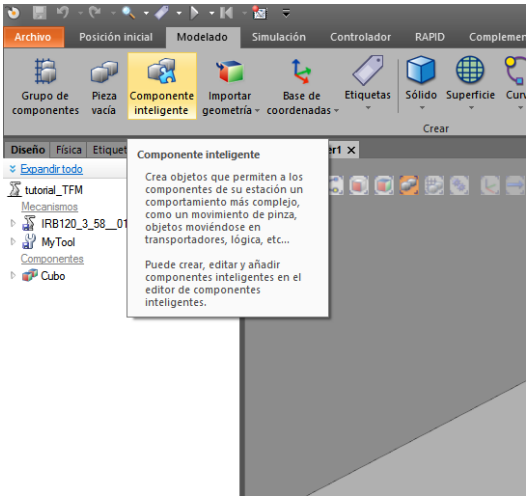


Figura 3.25: Creación de un Componente Inteligente (SC)

Los Componentes Inteligentes los podemos agrupar en 6 categorías distintas: Señales y propiedades, Primitivos paramétricos, Sensores, Acciones, Manipuladores, Controlador, Física, PLC y Otros.

- **Señales y propiedades:** es la primera categoría que encontramos, en ella podemos encontrar puertas lógicas, multiplexores, contadores, temporizadores, expresiones matemáticas, convertidores y otros elementos que permiten modificar una señal o las propiedades del sistema a programa. Como se puede observar en la Figura 3.26.

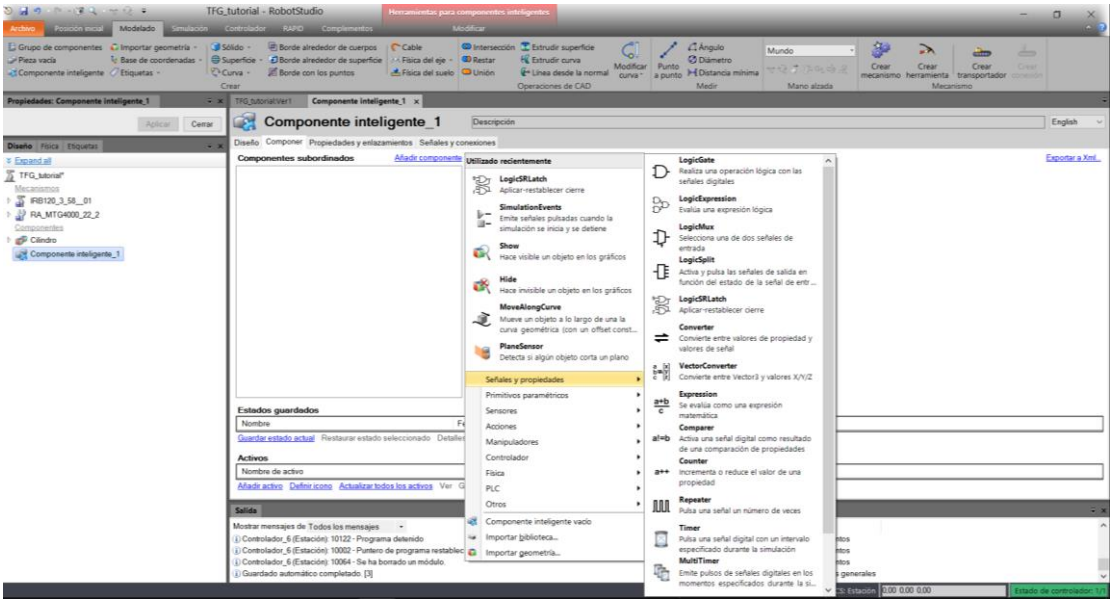


Figura 3.26: Categoría de Señales y propiedades en SC

- **Primitivos paramétricos:** En esta categoría se encuentran los componentes necesarios para crear de forma automática sólidos y líneas, así como generar copias de piezas ya existentes.

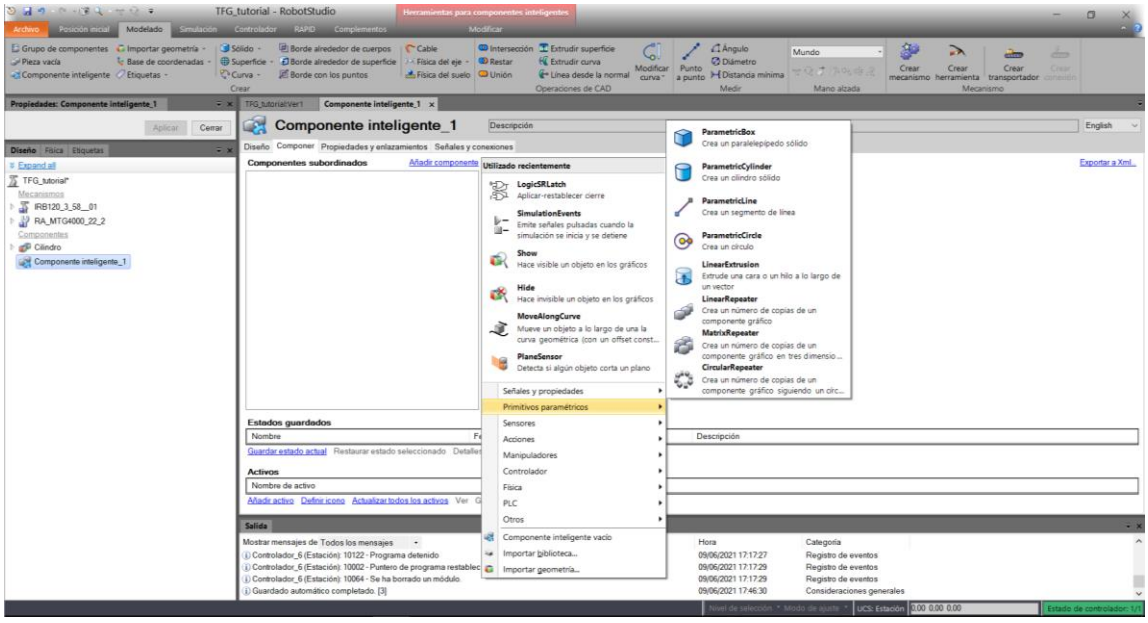


Figura 3.27: Categoría de Primitivos paramétricos en SC

- **Sensores:** Esta categoría reúne el conjunto de sensores a utilizar en los procesos de producción automático de este trabajo. Incluye sensores de colisión, de línea, de superficie, volumétricos, etc.

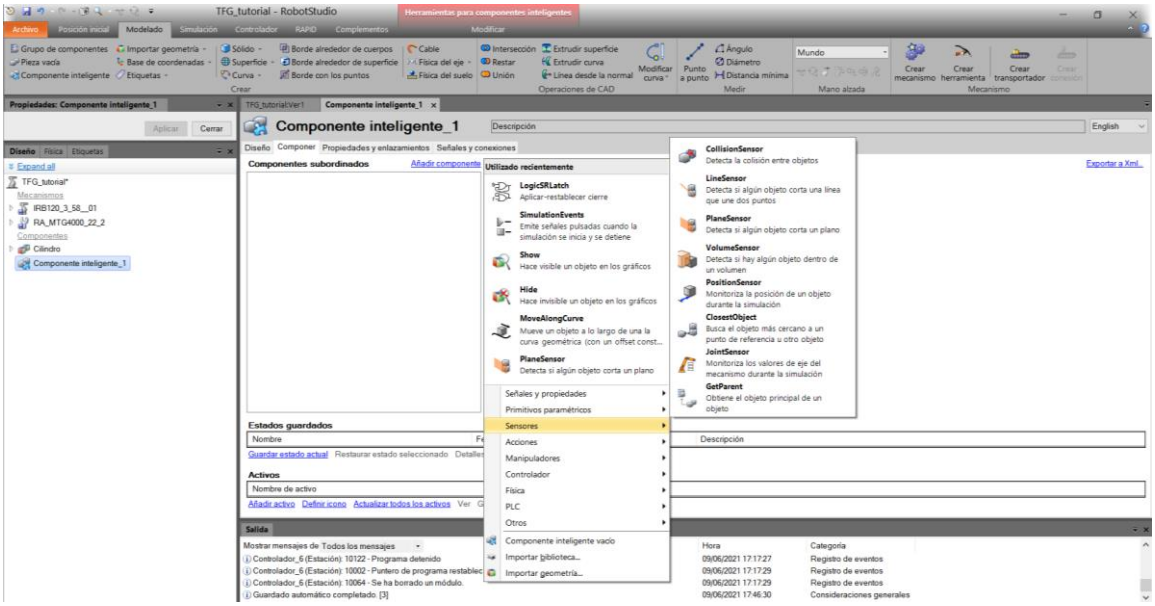


Figura 3.28: Categoría de Sensores en SC

- **Acciones:** Hace referencia a componentes inteligentes como conectar o desconectar dos objetos entre sí, eliminar un objeto o simplemente hacerlo visible/invisible.

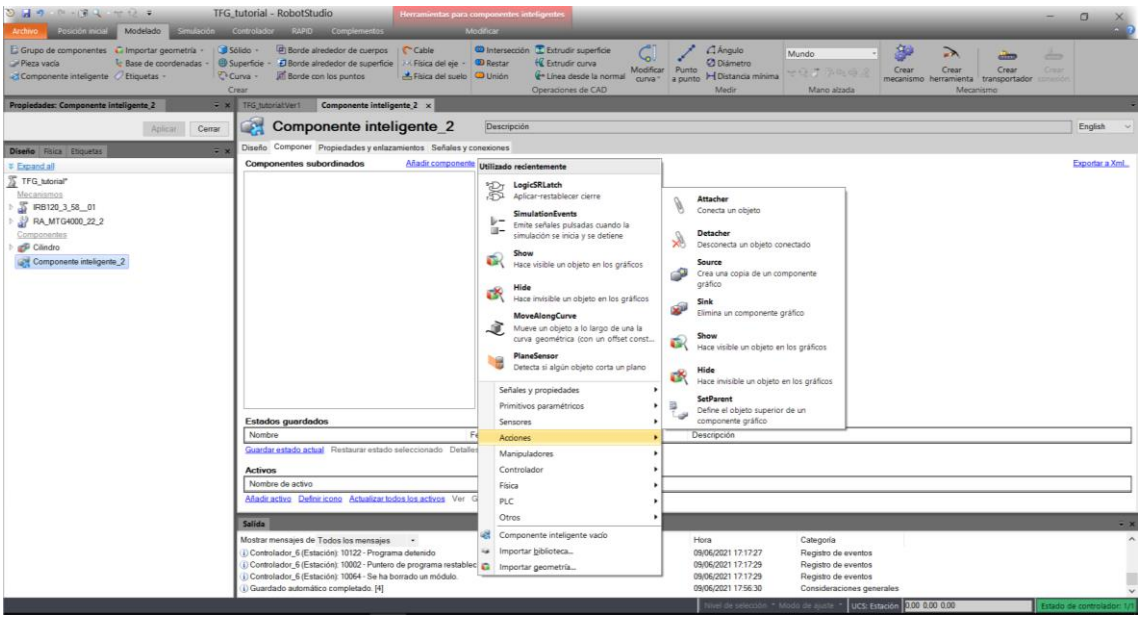


Figura 3.29: Categoría de Acciones en SC

- **Manipuladores:** en este apartado reúne los componentes necesarios para mover un objeto de forma lineal, hacerlo rotar un determinado ángulo , moverse a lo largo de una curva o spline, o posicionarlo en un lugar específico. También permite mover los ejes del mecanismo de un brazo robótico a la posición elegida.

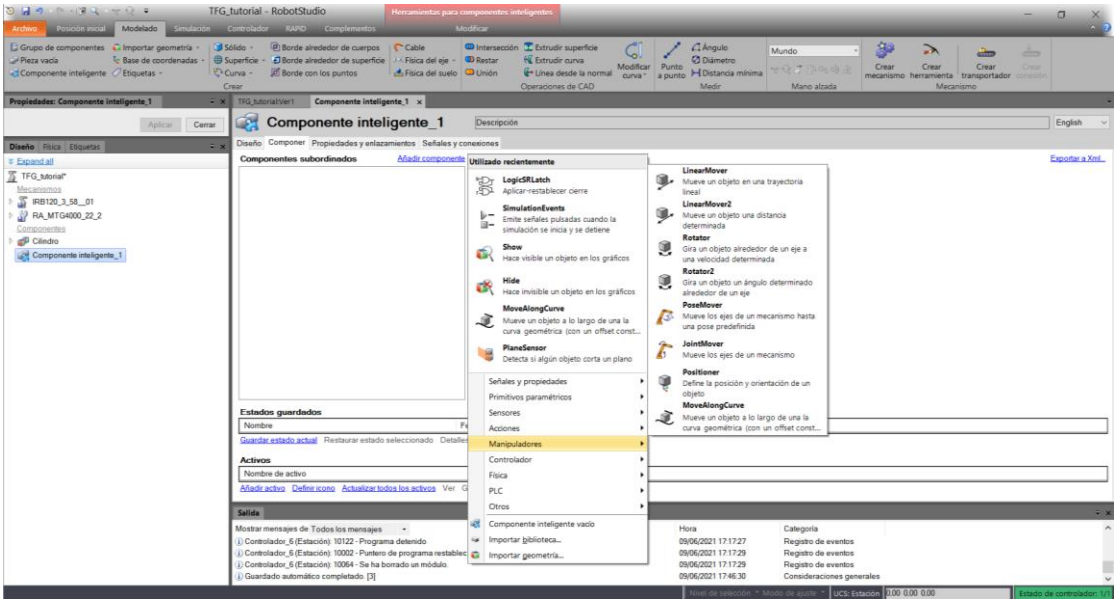


Figura 3.30: Categoría de Manipuladores en SC

- **Controlador:** en este apartado encontramos elementos para establecer u obtener el valor de una variable en RAPID.

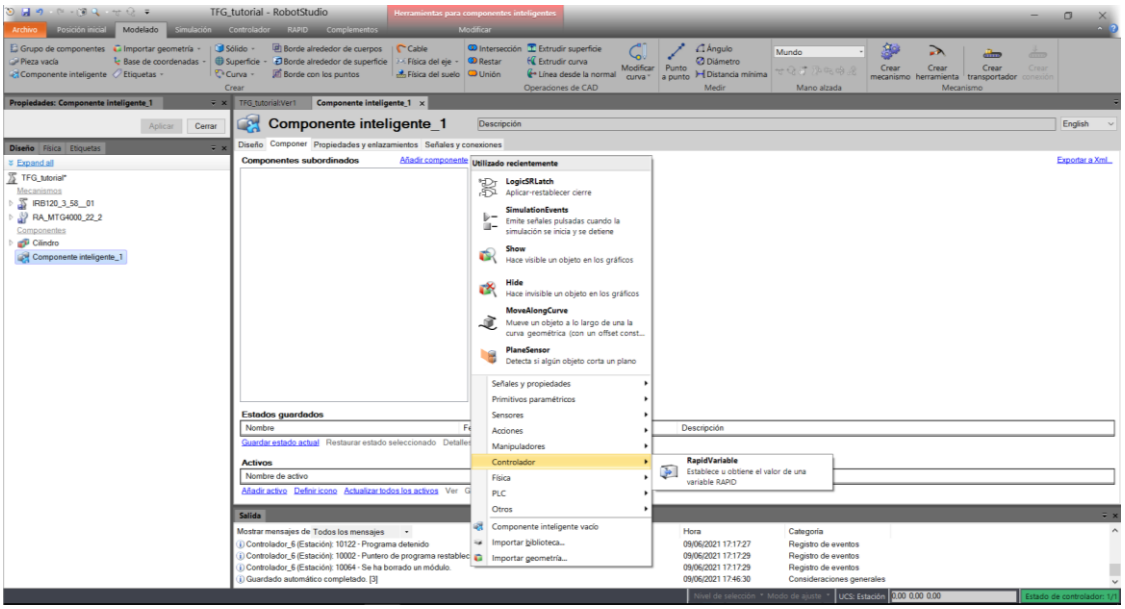


Figura 3.31: Categoría de Controladores en SC

- **Física:** en esta categoría encontramos elementos que permiten controlar la física de un objeto o que los ejes de un robot.

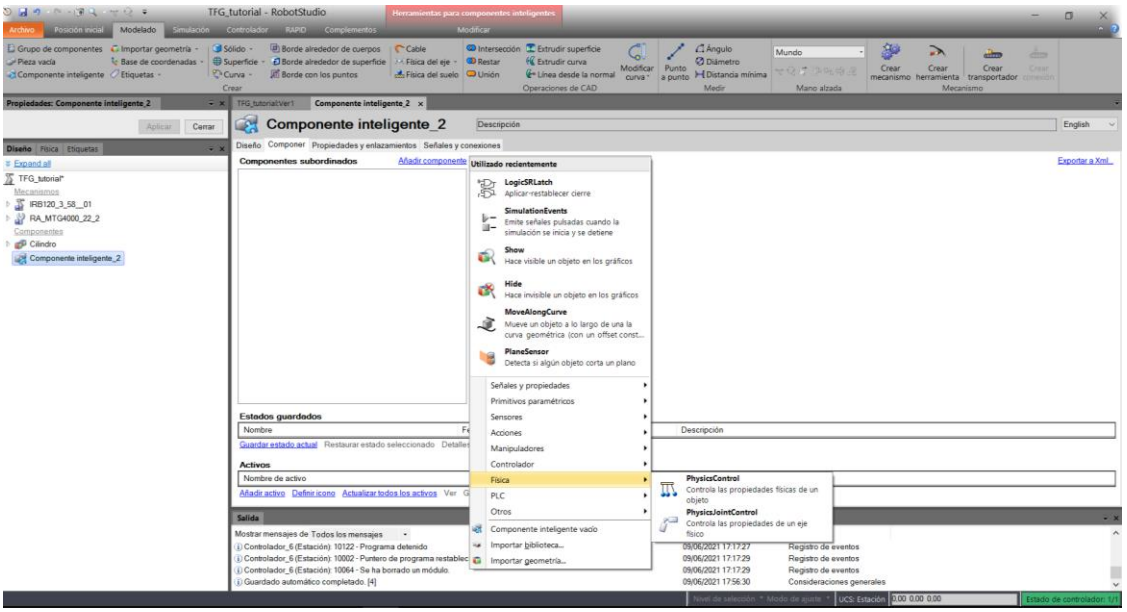


Figura 3.32: Categoría de Física en SC

- **PLC:** en este apartado encontramos un elemento que permite realizar la conexión con Siemens SIMIT.

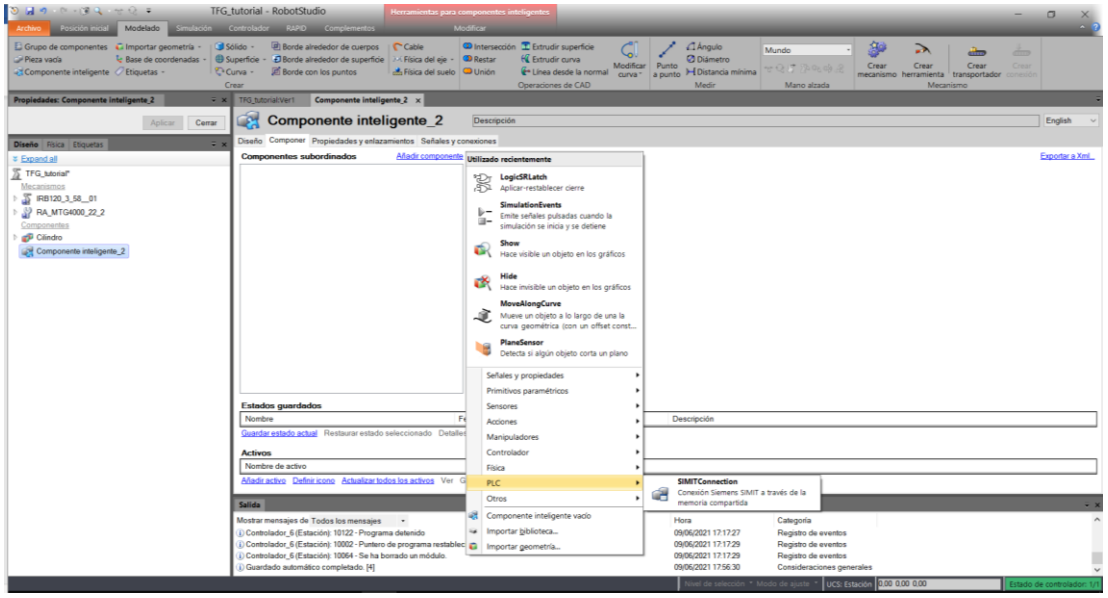


Figura 3.33: Categoría de PLC en SC

- **Otros:** En la última categoría se encuentran un conjunto de SC de distinta variedad. Representación de una cola de objetos, generación de un número aleatorio, detención de la simulación, reproducción de un sonido, etc.

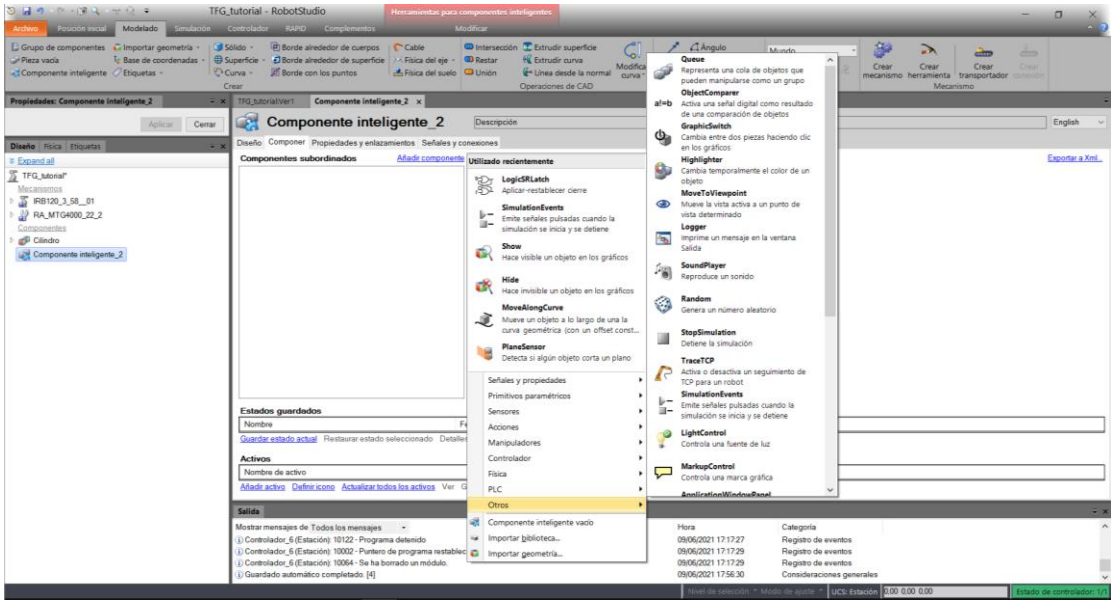


Figura 3.34: Categoría de Otros en SC

Dentro del Componente Inteligente, tenemos una pestaña de Señales y conexiones, donde podemos añadir señales de entrada o salida correspondientes a este componente, además también podemos realizar las conexiones entre los diferentes objetos. Lo podemos ver en la Figura 3.35.

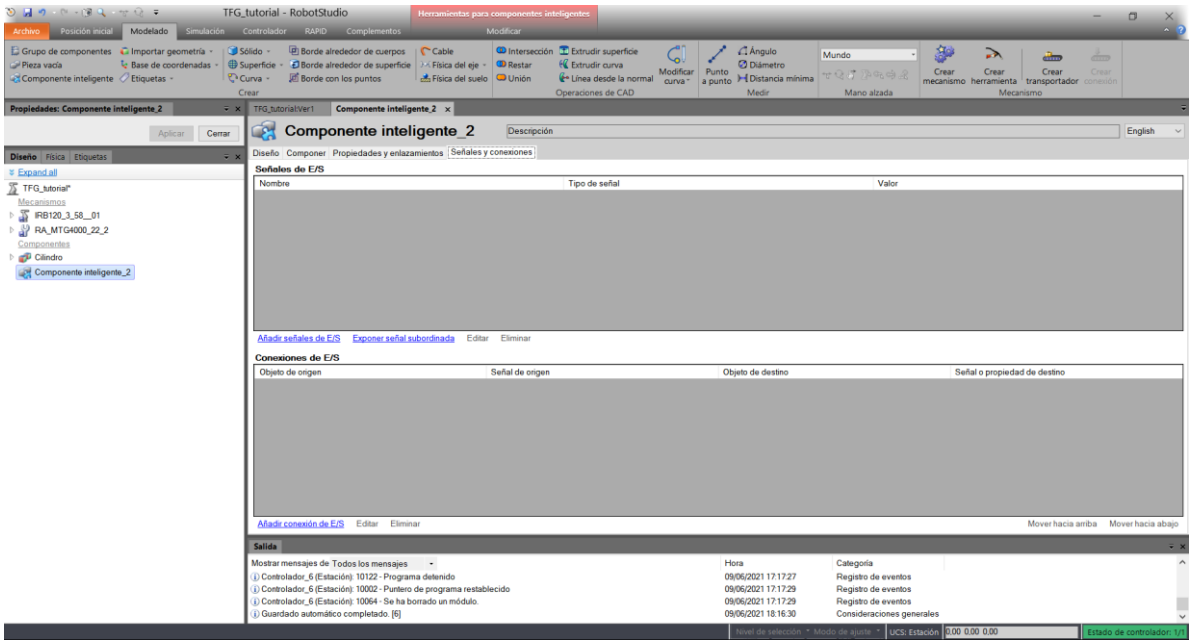


Figura 3.35: Señales y Conexiones en SC

Luego, en la pestaña de Diseño, se puede ordenar los objetos que hayamos incluido en el Componente Inteligente, cambiar su configuración, añadir entradas/salidas o conectar mediante flechas los distintos objetos.

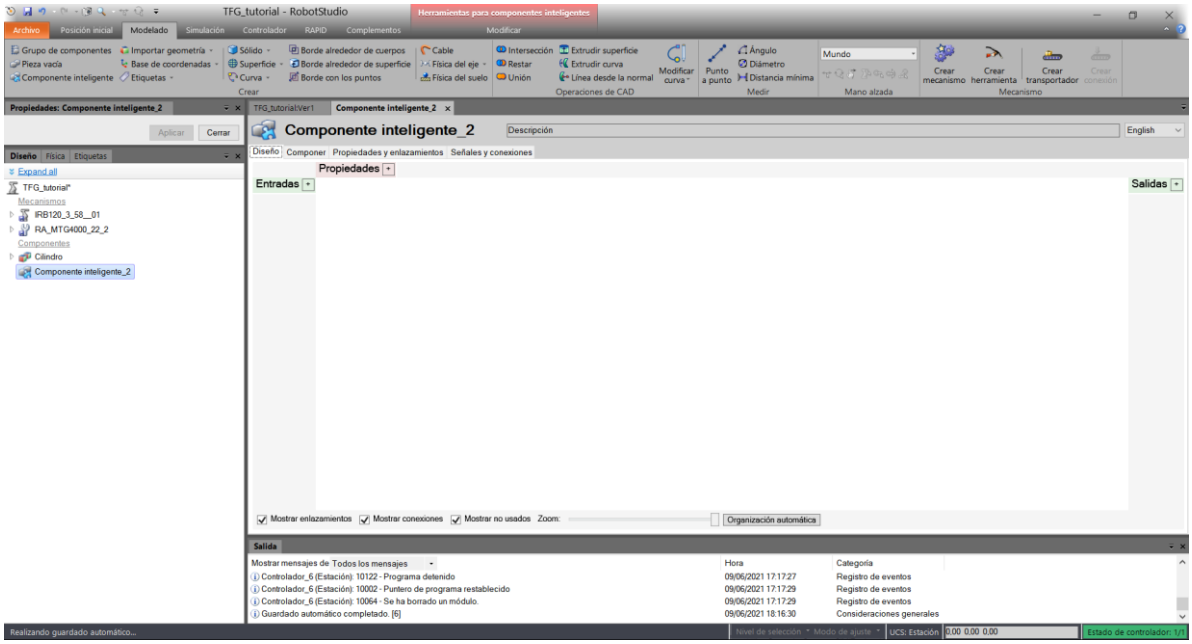


Figura 3.36: Diseño en SC

3.2.3. Creación de señales E/S

Una vez creada la geometría de la estación, el siguiente paso es “Añadirle inteligencia” al robot que tenemos en la estación. El sistema de E/S maneja las señales de entrada y salida del controlador. Para añadir una señal al controlador, desde la pestaña “Controlador”, seleccionamos “Configuración” y hacemos clic en “Añadir señales…”, como se puede ver en la Figura 3.37.

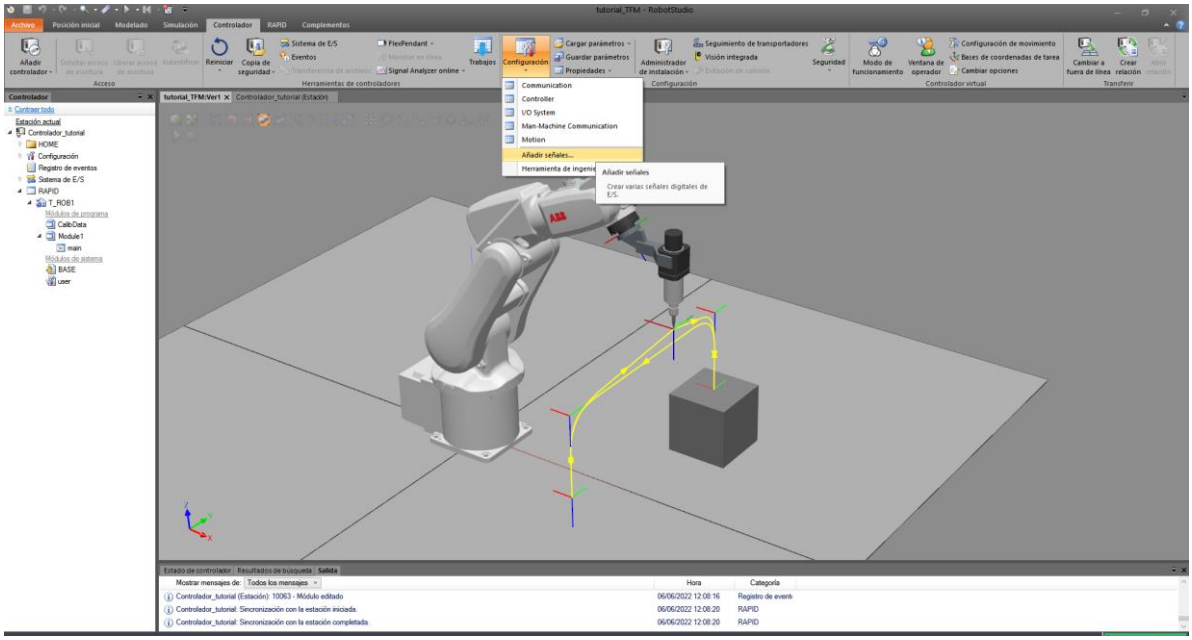


Figura 3.37: Añadir señales al controlador

Para añadir una señal, primero debemos de elegir el tipo de señal queremos añadir. Las más comunes suelen ser las señales de Entrada Digital o Salida Digital, en este caso vamos a añadir un Salida Digital. Se le debe de dar un nombre a la señal (“Activar_ventosa”) y en asignación a dispositivo seleccionamos “ninguno” y para finalizar pulsamos Aceptar, como en la Figura 3.38.

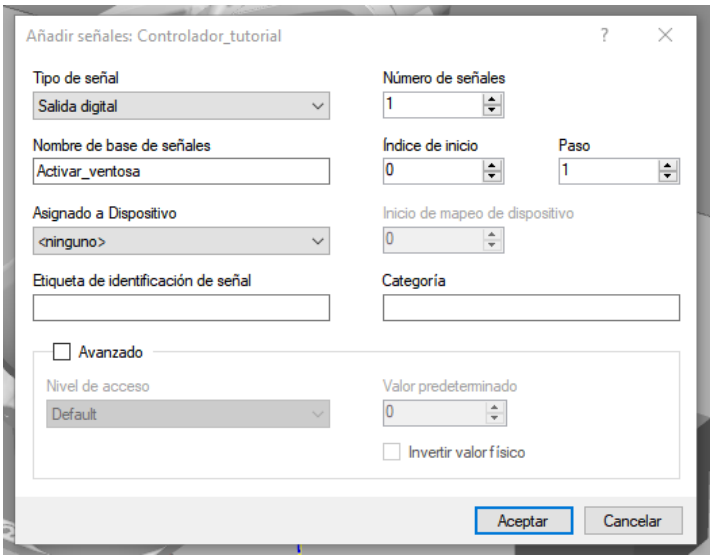


Figura 3.38: Crear Salida Digital en el controlador

Lo más importante a la hora de crear señales es que se tiene que Reiniciar el controlador cada vez que se realice alguna modificación o se añada una señal nueva, para que se guarden y sincronicen correctamente los cambios realizados. Como vemos en la Figura 3.39, vamos a la pestaña “Controlador” y hacemos clic en “Reiniciar” para reiniciar el controlador.

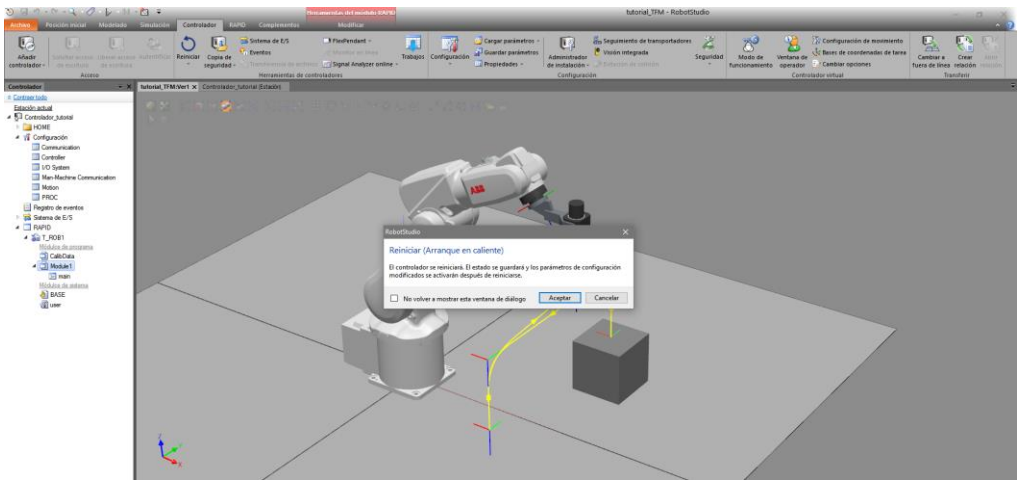


Figura 3.39: Reinicio del controlador

Otra forma de añadir señales es en la pestaña del “Controlador”, en la ventana de la izquierda, dentro de la pestaña del controlador del robot, en el desplegable “Configuración”, hacemos clic en “I/O System”. Para añadir una nueva señal, haciendo clic con el botón derecho en “Signal”, pinchamos en “Nuevo Signal”. También, es posible editar, copiar o eliminar señales creadas anteriormente.

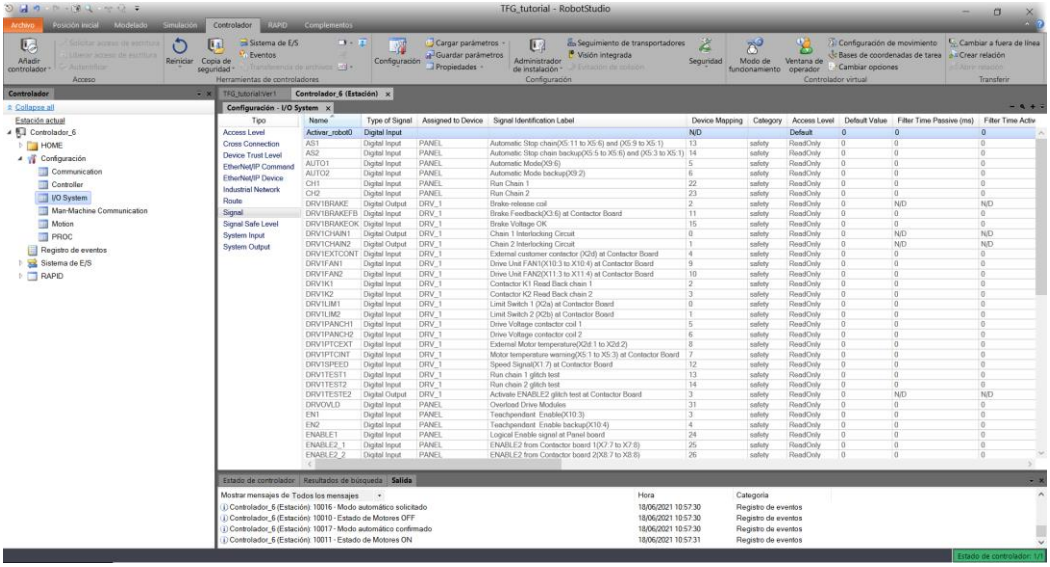


Figura 3.40: Configuración de señales en controlador

Una vez añadidas todas las entradas y salidas necesarias, se crearán un conjunto de señales del controlador que corresponderán a las señales de entrada y salida de los Componentes Inteligentes que sean necesarias para el control de los robots. En el Anexo 8.2 del presente trabajo se encuentra el listado de entradas y salidas de este trabajo.

3.2.4. Lógica de la estación

Tras haber creado el controlador del robot y las entradas/salidas, es necesario conectarlos entre sí. Es un paso importante para que el sistema coordinado de los robots y de los sólidos inteligentes funcionen correctamente durante la simulación.

Para ello, accedemos a la pestaña “Simulación” y dentro de ésta, hacemos clic en “Lógica de estación”. La lógica de la estación estará formada por un conjunto de bloques correspondientes al controlador del robot y a los controladores que se creen.

Para este tutorial, al tratarse de una tarea de Pick & Place, se va a crear un Smart Component con el que las piezas serán recogidas por la herramienta y también, serán soltadas.

El componente inteligente creado se llama “Smart_component_ventosa”. En él se han añadido bloques para coger y soltar piezas que van a ser detectadas por un sensor que se encuentran en el extremo de la herramienta.

El sensor añadido se debe modificar sus características para que sea lo suficientemente grande como para detectar la pieza. En este caso, se dará un valor de 2 mm al radio del sensor y una altura de 50 mm. Una vez modificados los parámetros, se debe de conectar con la herramienta del robot. La manera más sencilla es la de arrastrar el sensor a la herramienta y actualizar la posición, de esta forma el sensor quedará conectado al robot en todo momento, como se puede observar en la Figura 3.41.

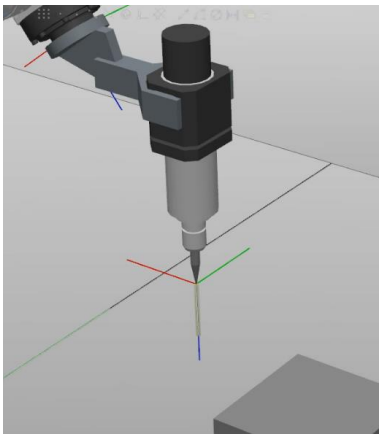


Figura 3.41. Sensor conectado a herramienta

Desde RAPID se activará la señal de “Activar_ventosa” para que se active el sensor y seguidamente, con el bloque “Attacher”, la pieza detectada se enganchará a la herramienta simulando que la está recogiendo. Del mismo modo, cuando la señal esté desactivada la herramienta soltará la pieza que tenga cogida en ese instante.

A continuación, en la Figura 3.42 se muestra como quedaría conectado el Smart Componente de la ventosa.

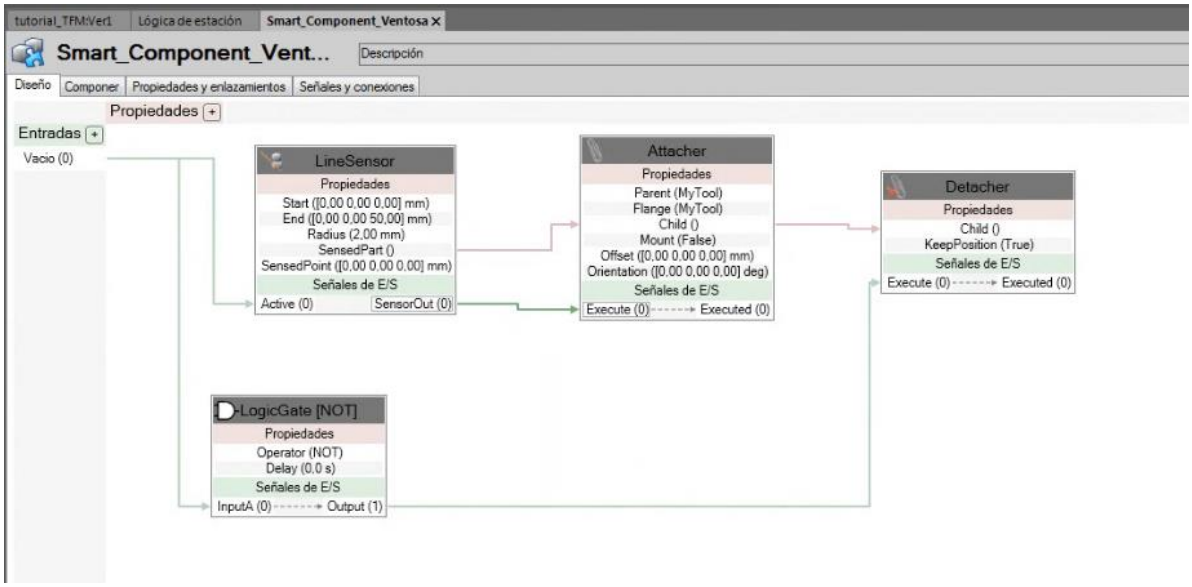


Figura 3.42. Smart Component Ventosa

Además, en la pestaña de Lógica de la estación se puede observar el esquema la estación entera, donde el controlador está conectado con todos los componentes inteligentes.

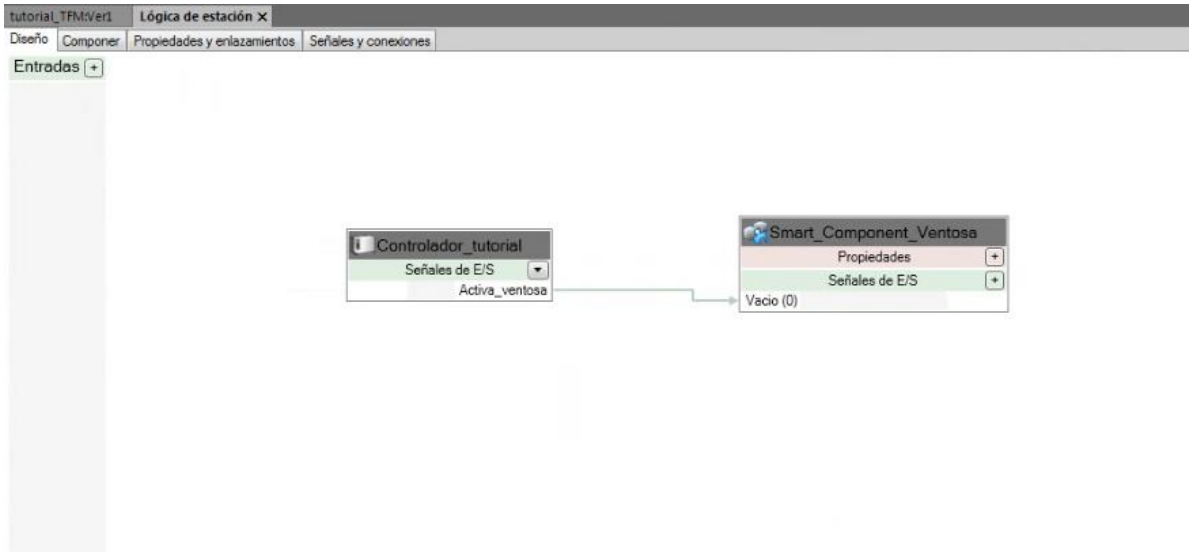


Figura 3.43: Lógica de la estación

3.3. Simulación de una estación

Una vez finalizada la programación de la estación, solo queda realizar la simulación, y así observar el funcionamiento del sistema automatizado que hemos creado. Para observar paso a paso la simulación en el programa RAPID, desde la pestaña RAPID, pulsamos en “Verificar programa”, seleccionamos “Aplicar” para que se guarden los cambios y finalmente hacemos clic en “Inicio” para comenzar la simulación.

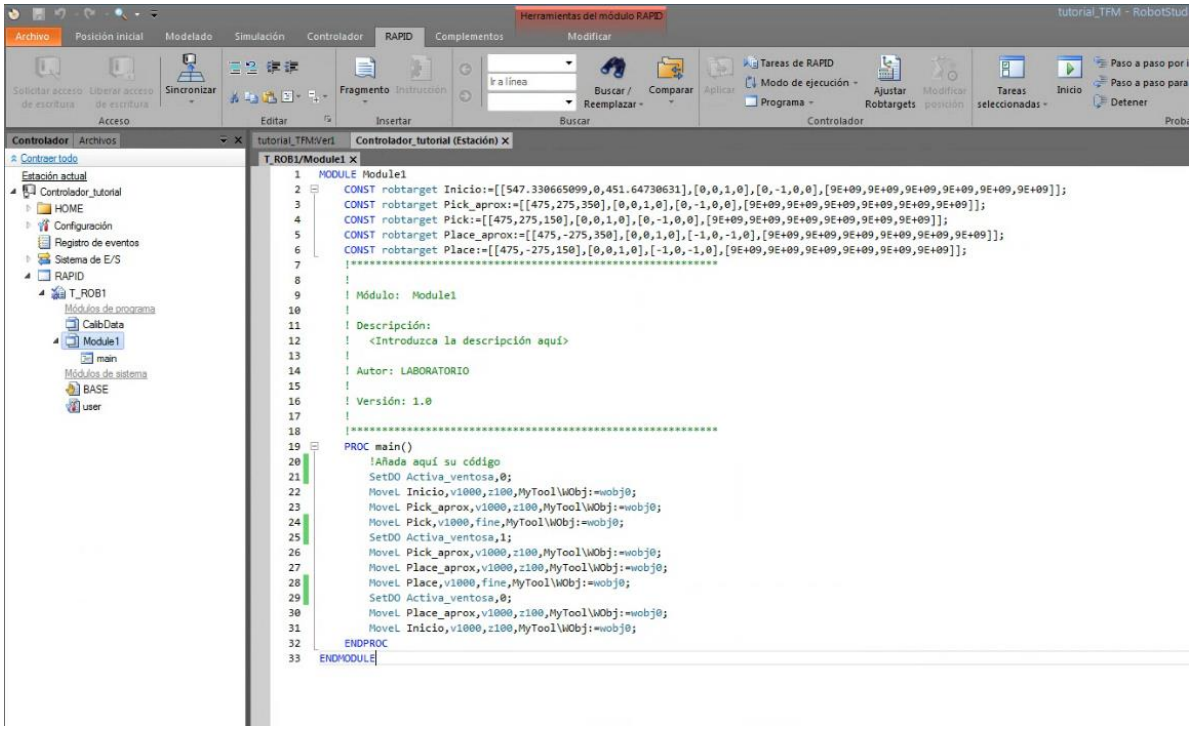


Figura 3.44: Compilación de un programa en RAPID e inicio de la simulación

Para realizar la simulación, podemos configurarla de varios modos, en la pestaña de “Simulación”, haciendo clic en “Configuración de simulación”, aparece un panel con las configuraciones como la que vemos en la Figura 3.45.

Seleccionando “Controlador_tutorial”, aparece unos ajustes donde se puede seleccionar que la simulación se inicie automáticamente. También, se puede seleccionar el modo de ejecución, podemos seleccionar que la tarea se realice una vez, seleccionando “Un solo ciclo” o escogemos el modo “Continuo”, donde la simulación no finaliza y lo realiza de forma continua.

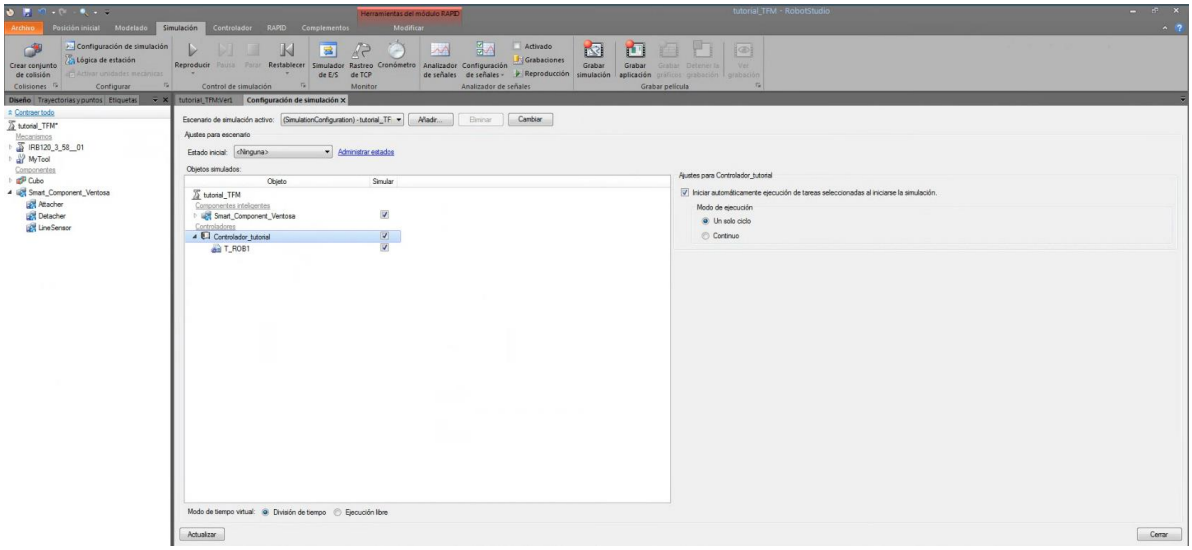


Figura 3.45: Configuración de la simulación

Otra opción es que se puede cambiar de la simulación con los ajustes de T_ROB1 e indicamos que solamente haga la trayectoria seleccionada “main”, como vemos en la Figura 3.46.

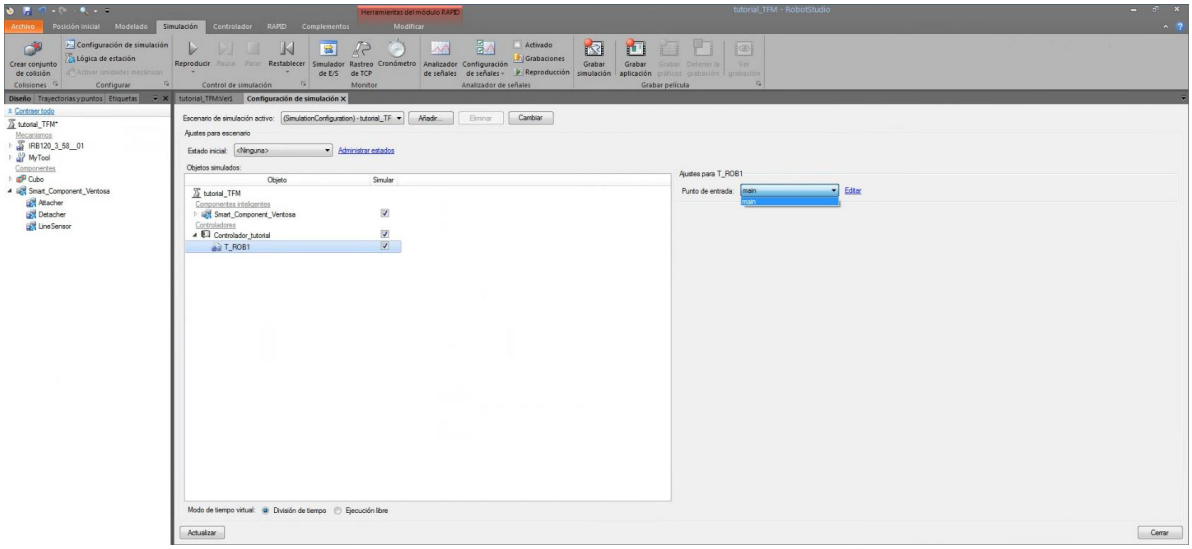


Figura 3.46: Configuración de la simulación, selección programa

En la Figura 3.47, se reproduce la simulación en la pestaña “Simulación”, en “Reproducir”. También, se puede parar la simulación y si vamos clicando varias veces en “Pausar”, se irá moviendo lentamente. Otra opción es la de “Parar” la simulación y de “Restablecer”, de esta última forma, si se ha movido los objetos o el robot vuelvan al estado inicial de la simulación (debiendo guardarlo previamente).

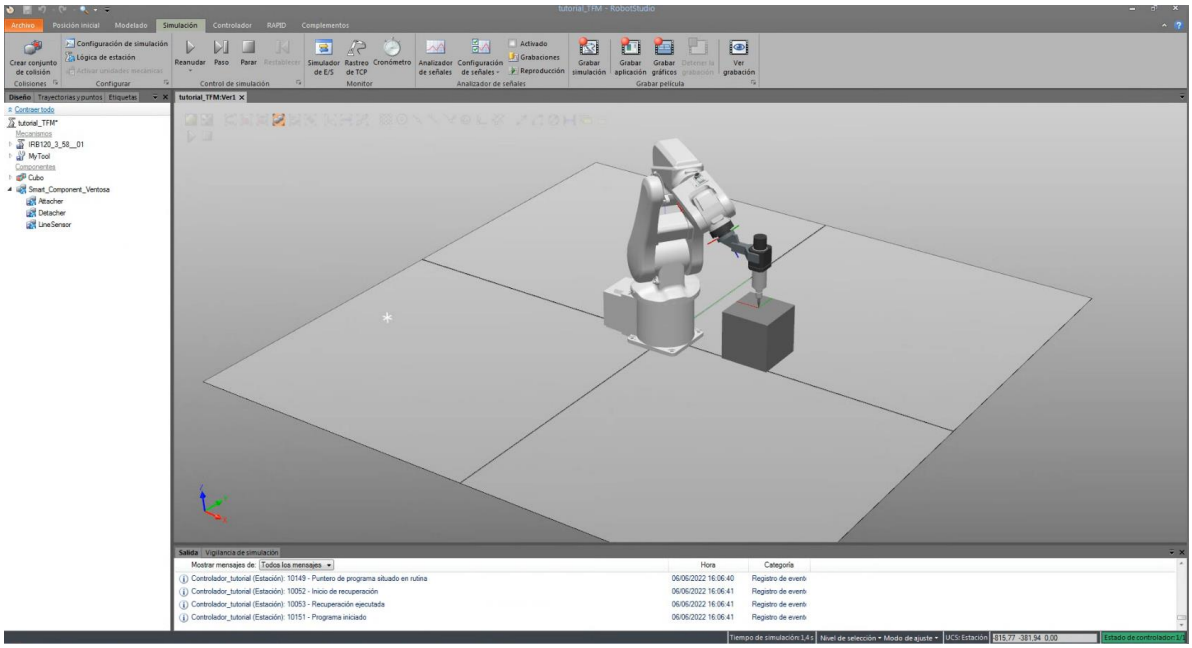


Figura 3.47: Simulación y grabación

4. SIMULACIÓN DE UN PROCESO DE PICK&PLACE DE PIEZAS DEL SECTOR TEXTIL Y CUERO CON RECONOCIMIENTO CON VISIÓN

4.1. Elección de los Robots

En el diseño de la estación para el proceso de Pick&Place se ha instalado un pórtico fijo situado sobre la cinta transportadora en la que es desplazada la plancha con los distintos modelos cortados de piezas. Se ha decidido colocar un robot industrial en el pórtico que sea capaz de realizar la tarea.

Los robots necesarios para este tipo de tareas deben de cumplir una serie de especificaciones entre las que se destacan ser rápidos, precisos, una gran capacidad de repetitividad, alcance suficiente para toda de la cinta transportadora, y un tiempo de ciclo corto durante la tarea.

Basándonos en las características que deben de tener los robots, hemos escogido el robot ABB IRB 1600, ya que cumple todas las especificaciones y se trata del robo adecuado para realizar estas operaciones en las que tenemos una cinta transportadora proveniente de una máquina CNC.

Una vez escogido el robot, en la Tabla 4.1 se pueden ver las distintas versiones que tiene el fabricante de este modelo, variando el alcance y carga que puede conseguir el robot.

Versiones IRB1600	Capacidad de carga (Kg)	Área de trabajo. Diámetro (mm)
IRB 1600-6/1.2	6	1200
IRB 1600-6/1.45	6	1450
IRB 1600-10/1.2	10	1200
IRB 1500-10/1.45	10	1450

Tabla 4.1: Versiones del robot ABB IRB 1600

Entre todas las versiones del robot mostradas en la tabla anterior, se ha escogido el modelo de robot IRB 1600-6/1.2, mostrado en la . Se ha escogido este modelo ya que el área de trabajo que ofrece es perfecta para el tamaño de la cinta trasportadora, al mismo tiempo que puede soportar grandes cargas en el caso que se utilice un material pesado para realizar la tarea.

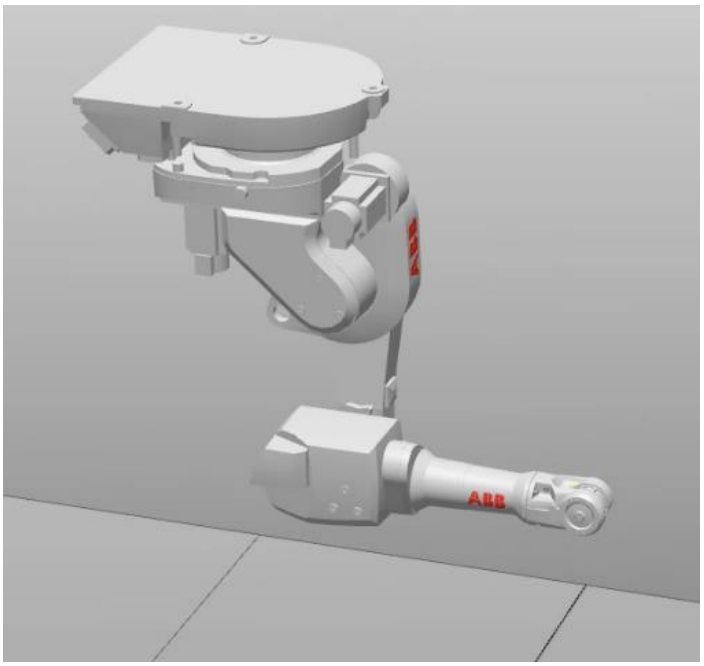


Figura 4.1: Robot ABB IRB 1600-6/1.2

4.2. Elección de la Herramienta

La herramienta creada para la simulación del proyecto ha sido una herramienta con doble ventosa, la cual permite coger las piezas creando el vacío entre la ventosa y la pieza, y soltarla una vez que el robot se encuentre en la posición deseada eliminando el vacío creado. En la Figura 4.2 se puede ver la herramienta creada en AutoCAD y posteriormente importada a RobotStudio, como se explica en el apartado 3.2.1.

Figura 4.2: Herramienta de doble ventosa

4.3. Diseño de los sólidos de la estación

Los sólidos han sido creados específicamente para la estación utilizando el software de Autodesk AutoCAD 2020, los planos correspondiente a todos los sólidos se encuentran en el Anexo 8.1 del presente trabajo.

En primer lugar, como se ha comentado en el apartado 4.2, la herramienta utilizada en la estación ha sido diseñada en AutoCAD. Ha sido diseñada de forma que la herramienta tenga dos ventosas con la suficiente superficie de contacto como para poder crear el vacío con las piezas que debe recoger. En la Figura 4.3 se puede ver la herramienta siendo diseñada en AutoCAD.

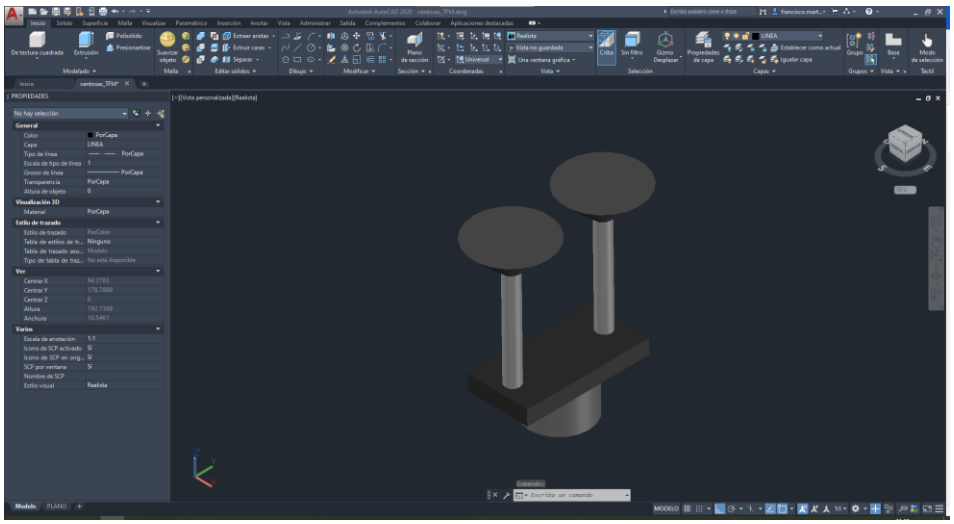


Figura 4.3: Herramienta con doble ventosa

El siguiente sólido diseñado ha sido el pórtico, mostrado en la Figura 4.4. El pórtico es el soporte para el robot en la estación, que se encuentra situado sobre la cinta transportadora por donde se desplazan las piezas a recoger. El pórtico se ha diseñado con la altura necesaria para que el robot pueda alcanzar sin llegar a posiciones singulares toda la superficie donde se van a encontrar las piezas, así como las posiciones de dejada durante el proceso de Pick&Place.

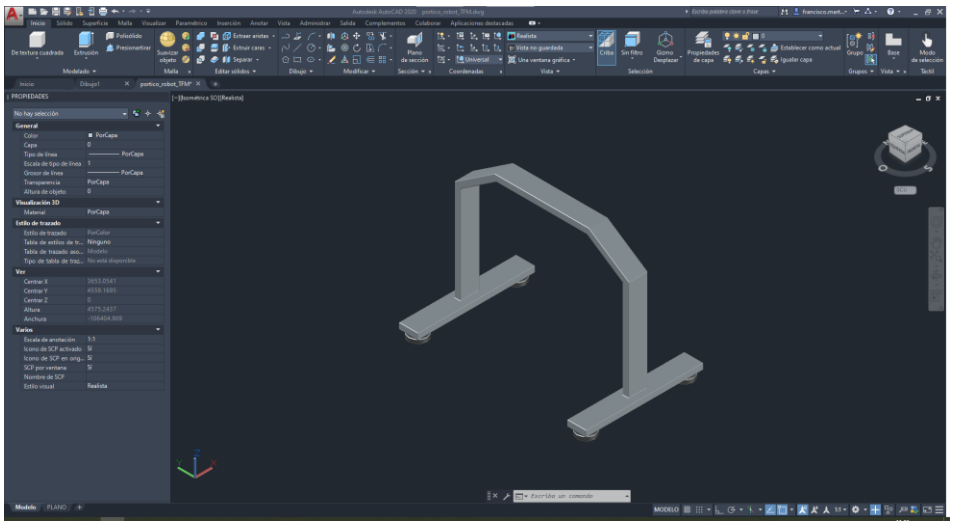


Figura 4.4: Pórtico robot

Otro sólido diseñado en CAD ha sido el soporte para la cámara del sistema de Visión. El diseño es sencillo, simplemente se trata de un soporte que se puede atornillar al pórtico y con una longitud suficientemente larga para poder realizar la foto de las piezas a recoger sin que aparezca el robot en la imagen. En la Figura 4.5 se puede ver el soporte en AutoCAD.

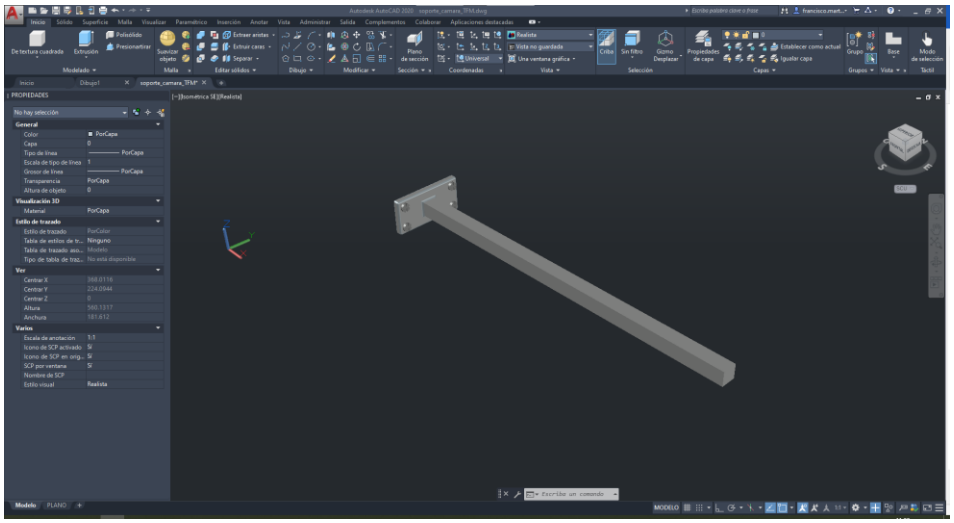


Figura 4.5: Soporte Cámara

Por último, se ha diseñado una pieza textil con distintos cortes de plantillas que se han utilizado para la simulación del proseos de Pick&Place. La forma de la pieza textil es una forma común en el sector para este tipo procesos que pueden provenir de piel de animales o pieles sintéticas. Las piezas cortadas en ella son distintas partes que se utilizan para confeccionar un zapato, como lo son la suela del zapato, los laterales, el talón o el empeine del pie.

Dicha pieza se introducirá a la CNC sin ser cortada y en la simulación se puede ver que sale de la máquina ya cortada, con la visión se realizará una correspondencia entre su posición en la cinta y los datos extraídos del modelo CAD, para posteriormente comunicarle al robot las posiciones. En la Figura 4.6 se puede observar el diseño de la pieza cortada y sin cortar.

Figura 4.6: Pieza textil (izquierda cortada, derecha sin cortar)

4.3.1. Otros solidos utilizados

La máquina CNC ha sido importada y añadida a la estación, gracias a la aportación de Carlos Pérez Vidal a partir de una estación de ejemplo la cual contenía la máquina. Podemos verla en la Figura 4.7.

La máquina está formada por un tapiz rodante sobre el que se desplazan las planchas que van a ser cortadas, dentro se encuentra una herramienta de corte que corta la plancha textil en distintas piezas para que salga por la cinta con el diseño CAD que se le indique.

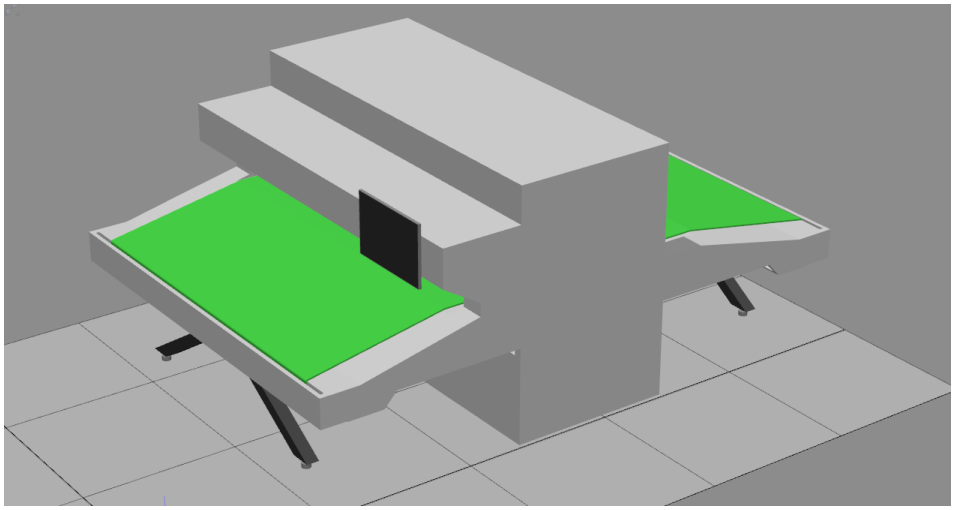


Figura 4.7: Máquina CNC

Por otro lado, la cinta transportadora se ha importado de la biblioteca de RobotStudio y se ha modificado para que el tapiz rodante fuera de las mismas dimensiones que la maquina CNC. En la Figura 4.8 podemos ver el diseño final de la cinta transportadora donde se realizará el proceso de Pick&Place.

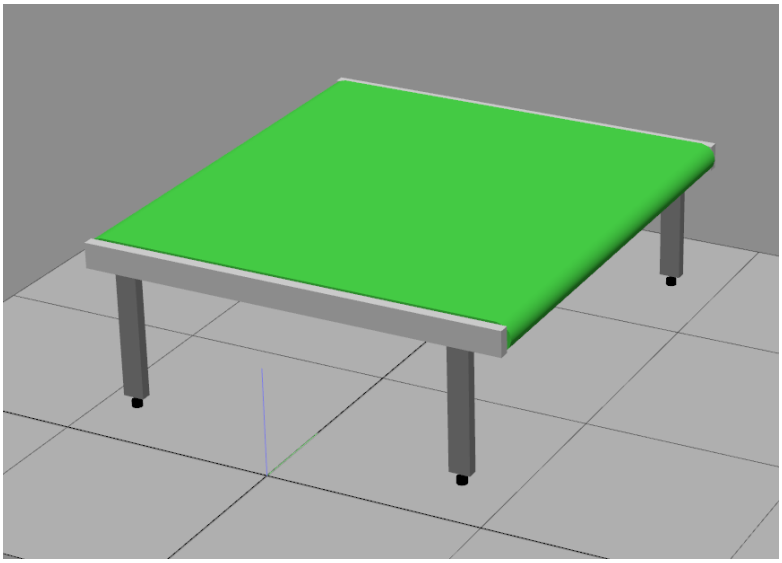


Figura 4.8. Cinta transportadora

Finalmente, también de la librería de RobotStudio se ha añadido la cámara que representa el sistema de Visión, Figura 4.9.

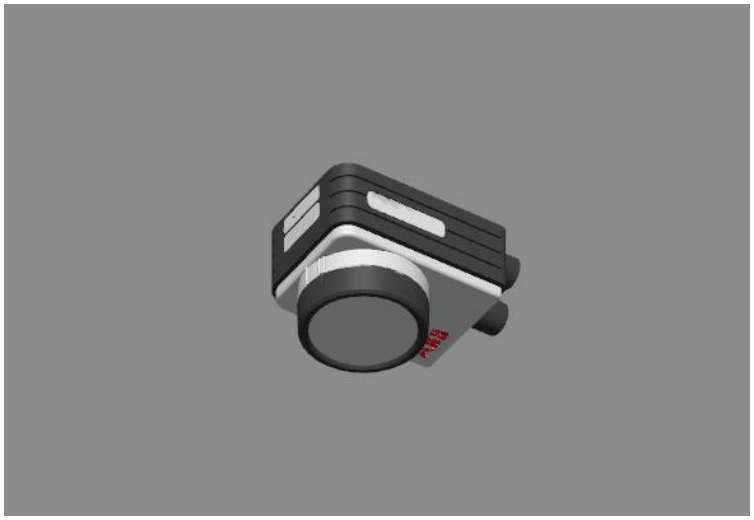


Figura 4.9. Cámara

En la Figura 4.10, se puede observar una imagen global de toda la estación.

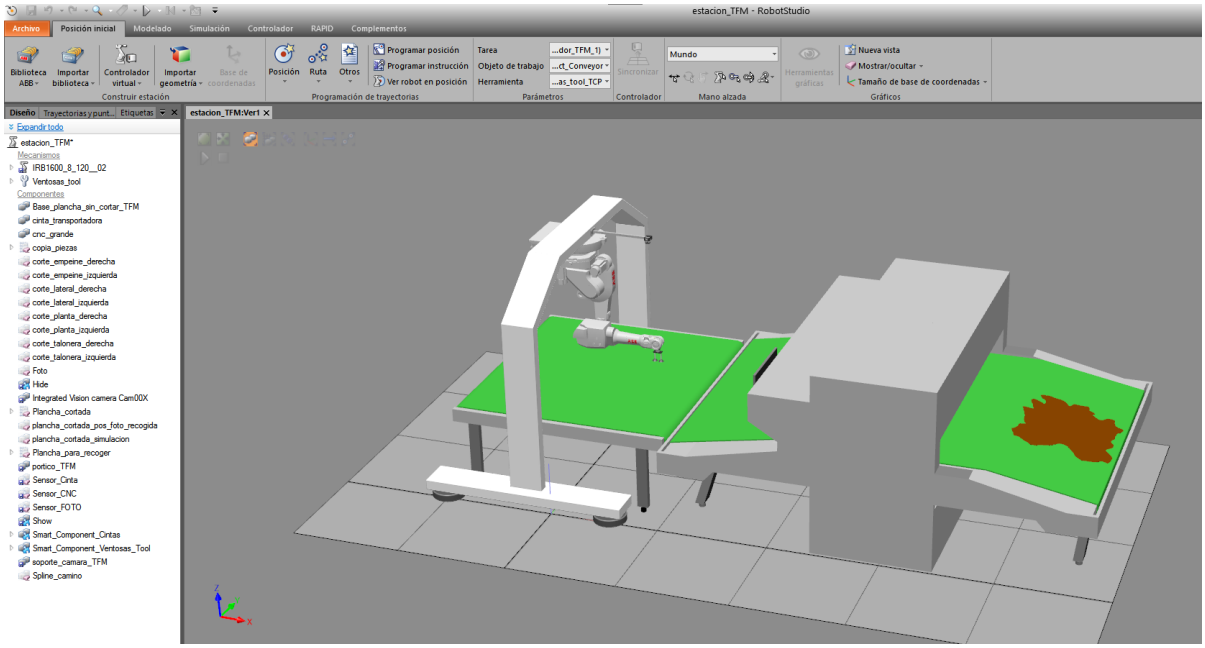


Figura 4.10. Estación de RobotStudio

4.4. Creación de la estación en RobotStudio

En este apartado se va a explicar cada uno de los Smart Componentes creados para la estación, así como todas las señales de entrada y salida del controlador y la lógica general del programa.

4.4.1. Smart_Component_Cintas

Uno de los Smart Components creados es el de las cintas, en el cual se controla el movimientos de las piezas a los largo de las cintas, así como mediante las señales provenientes del controlador y de los sensor que se han colocado en las cintas se para o activa la marcha de las cintas y la toma de la imagen. Se muestra en la Figura 4.11. y está formado por los siguientes componentes:

- **MoveAlongCurve**: Mueve un objecto (Object) a lo largo de una curva geométrica o spline creada simulando en movimientos de las cintas (usando un constante Offset), se le pude indicar una velocidad determinada (Speed) en mm/s y en el apartado (WirePart) se le indica el cable (spline) que se ha creado para recrear el movimiento de las piezas sobre las cinta. La casilla de (KeepOrientation) no se ha seleccionado ya que así conseguimos que la plancha cambie de orientación a lo largo del movimiento de las cintas dependiendo de las pendientes de estas. El movimiento se inicia cuando la señal proveniente del bloque “Simulation Events” indica que se ha iniciado la simulación, por otro lado. Encontramos dos componentes MoveAlongCurve diferentes. Una de ellas se corresponde con la base que aún no ha sido cortada y su movimiento es cancelado cuando el Sensor_CNC es activado. El otro componente es para la base con las piezas ya cortadas, las cuales son canceladas cuando el Sensor_CINTA y Sensor_FOTO se activan.

- **Hide**: hace un objeto (Object) invisible gráficamente. Encontramos dos componentes en el SC, utilizados para las bases de las planchas que no han sido cortadas, estas planchas son visibles inicialmente y una vez que el Sensor_CNC es activado las planchas se hace visibles.

- **Show**: hace un objeto (Object) visible gráficamente. También encontramos dos componentes, pero ahora están asociados a las planchas que ya están cortadas y en las que se pueden ver las plantillas, las cuales inicialmente no son visible y una vez que el Sensor_CNC se activa se hace visibles.

- **Simulation Events**: Este bloque permite usar una señal que se activa cunado se inicia la simulación en Robotstudio, se ha utilizado para hacer visible la base sin cortar y no mostrar la base cortado a la entrada de la cinta.

- **LogicSRLatch**: este bloque permite a partir de una señal hacer que se mantenga activa durante un tiempo hasta que se activa la señal que entra por la opción de reset.

Gracias a la utilización de los componentes Hide y Show se consigue representar el proceso de corte en la maquina CNC, teniendo al inicio del tapiz rodante la base sin contar y una vez que se encuentran en el interior de la maquina se muestra la base cortada, simulando que la base sale cortada de la CNC.

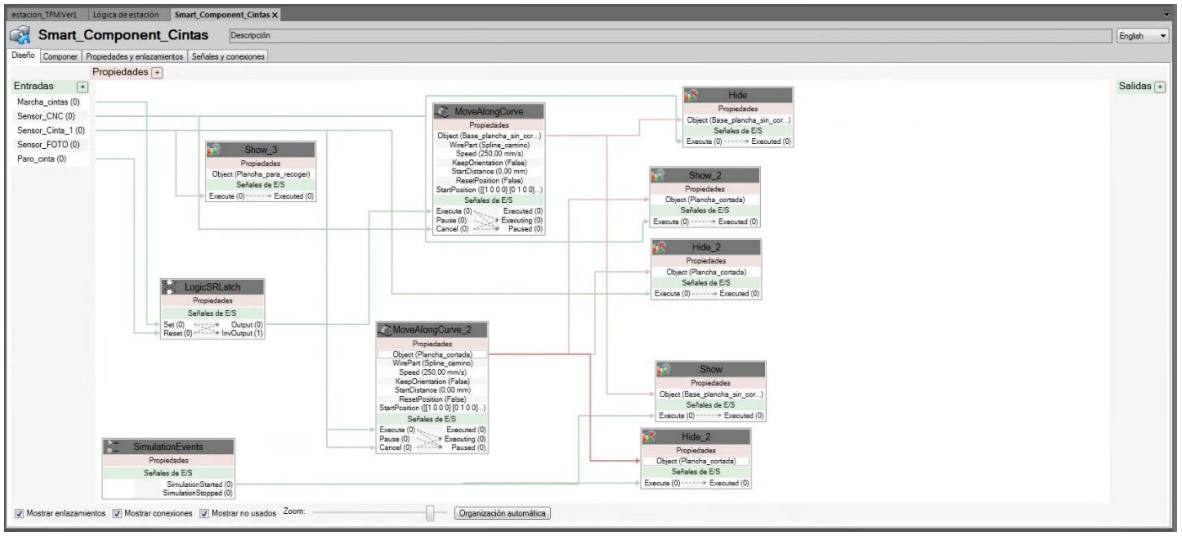


Figura 4.11: SmartComponent_Cintas

Se añaden varias entradas al Smart Component de las cintas, como lo son la entrada de la señal Sensor_CNC, que es la señal utilizada para activar los componentes de Hide y Show, así como el de cancelar el movimiento de alguna de las planchas cuando esté activada.

La señal de Sensor_CINTA y Sensor_FOTO se utilizan cuando se encuentre activadas para cancelar el movimiento del resto de las planchas y parar el movimientos de todas las cintas cuando se encuentre en la posición de recogida, así como de activar el proceso de realizar la foto.

También, desde el controlador son introducidas las señales de Paro y Marcha_cinta con las que se consigue le movimiento de las cintas conectándolas al bloque comentado anteriormente de LogicSRLatch.

4.4.2. Smart_Component_Ventosas_Tool

Este componente inteligente se encarga de realizar la acción de crear el vacío y coger una pieza con las ventosas de la herramienta del robot y luego soltar dicha pieza dependiendo del valor de las entradas. La entrada que se ha añadido al Smart Component es la de “Vacío”, con ella indicamos cuando se debe de activar un sensor del bloque “LineSensor”. Este sensor se encuentra en el extremo de la herramienta, entre las dos ventosas y con el que se detecta la pieza a recoger. Cuando la señal esté activa quiere decir que se debe de recoger una pieza, y para cuando la señal esté desactivada, ninguna pieza debe de ser cogida, o en el caso de que el robot haya cogido una, esta debe de soltarse.

El sensor se ha creado indicando una longitud de 59 mm, en los apartados de (Start) y (End), además, se le ha indicado un radio (Radius) de 2 mm. La propiedad de (SensedPart) indica que solido ha sido detectado y está conectada al componente “Attacher” en la propiedad (Child), así como la salida que ofrece el LineSensor de (SensorOut) que se activa si ha detectado algún objeto y que activa el “Attacher”.

El componente “Attacher” es el encargado de conectar la pieza detectada por el sensor con la ventosa, indicado en la propiedad (Parent), que en este caso es la ventosa del robot “Ventosas_tool”. El componente “Detacher” es el encargado de soltar la pieza y se activa cuando la señal de vacío se encuentra a “0”, tras pasar por un puerta lógica NOT, donde se convierte a “1”, de esta forma cuando no se deba de coger pieza siempre esta activado. La propiedad (Child) se conecta al Attacher para así cuando se deba de soltar el objeto sea el que esté cogido por la ventosa.

En la Figura 4.12 se puede ver la distribución del componente inteligente Smart_Component_Ventosas_Tool.

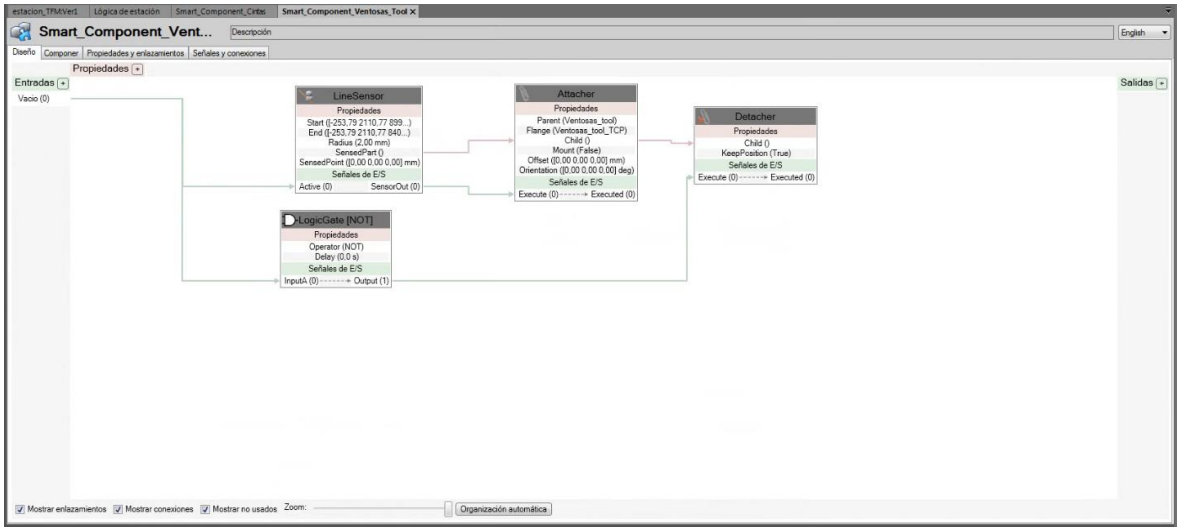


Figura 4.12: SC_ventosa_tool_izq

4.4.3. Controlador y Lógica de la Estación

El controlador del robot IRB1600_6_1.2 tiene tres señales de entrada y tres señales de salida. Las señales de entrada se tratan de las señales provenientes de los tres sensores de la estación, estos son el Sensor_CNC, Sensor_Cinta y Sensor_Foto.

El primero de ellos está colocado en el interior de la maquina CNC y con el que se hace el cambio de la base sin cortar a la base cortada simulando el corte realizado por la CNC. Las señales Sensor_Cinta y Sensor_Foto están colocados en el lugar donde se debe de para la cinta para que se pueda tomar una imagen y posteriormente, se realice el proceso de Pick&Place.

En cuanto a las señales de salida del controlador, tenemos la señal de Paro, Marcha y Coger_pieza. Las dos primeras de ellas son utilizadas a partir de los datos recibidos de los sensores, simular el movimiento de las cintas transportadoras. Para el caso de la señal Coger_pieza su función es ser la señal de entrada al Smart_Component_Ventosas_Tool, para indicar cuando se debe de recoger o dejar una pieza.

También, se pueden observar en la lógica de la estación dos componentes de Hide y Show que son utilizado para mostrar un sólido con forma de pirámide de color amarillento el cuan aparece y desaparece cuando la pieza está en el lugar de recogida simulando que la cámara ha tomado una imagen. En la Figura 4.13 se puede ver la lógica de la estación con sus respectivas conexiones.

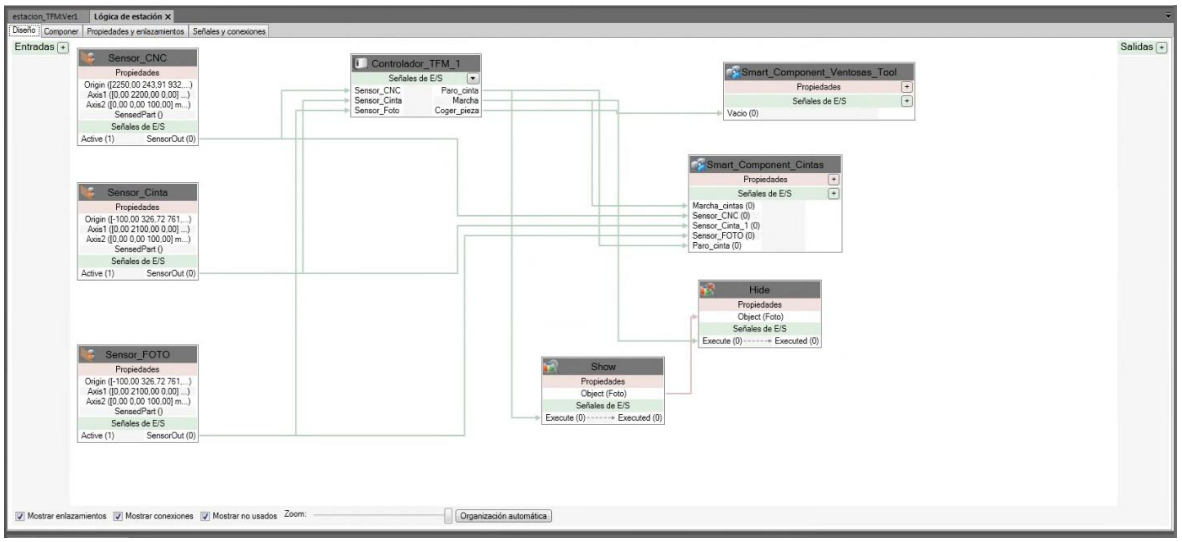


Figura 4.13: Lógica de la estación

4.5. Procesos creados en Python

El proyecto tiene una segunda parte de investigación basada en el lenguaje de programación Python. El objetivo es la obtención de los datos de posición donde han sido cortados todas las piezas en la CNC a partir del modelo CAD, y además, se debe de extraer información suficiente para poder comprobar y detectar si la base ha sido desplazada o girada durante el movimiento de las cintas.

De esta forma, se ha conseguido una mayor automatización del proceso de forma que el operario solamente debe de colocar la pieza al inicio de la CNC y esperar a que el robot las recoja y ordene al final de la otra cinta al finalizar el proceso.

Con el uso del reconocimiento de visón se ha conseguido evitar el problema de tener que estar la pieza en el mismo lugar y orientación durante todo el proceso, ya que ahora es capaz de saber cómo de desplazada está desde que fue cortada en la CNC y recalcular las posiciones a las que debe de desplazarse el robot.

Para esta parte, se parte con el uso del modelo que le es introducido a la maquina CNC. Se trata de un fichero .dwg de AutoCAD desde el que se van a extraer, por un lado, los datos de las piezas que se van a cortar en el base textil sin cortar introducida a la CNC, y por otro lado, los datos de dicha base, para poder enlazar todos los datos con la imagen tomada por Visión.

La otra parte del algoritmo se trata de un proceso que utiliza Visión Artificial. Se debe de detectar el contorno de la pieza que se encuentra en la cinta transportadora a partir de un imagen tomada en la estación. Aparte del contorno, se deberá de calcular el centro y los ejes de mayor inercia para poder enlazarlos con los datos del modelo de AutoCAD.

La última parte del algoritmo de Python se basa en establecer la comunicación entre Python y la simulación en RobotStudio. Para ello, se establece un socket entre ambos programas y mediante señales se le enviará todos los datos al robot cuando este se lo indique.

A continuación, se van a desarrollar cada uno de los procesos que se han programado en el algoritmo para la extracción de datos y la visión.

4.5.1. Obtención datos desde AutoCAD

Como se ha comentado anteriormente, desde AutoCAD se debe de obtener los datos de las piezas a recoger según el modelo que se haya mandado a cortar a la maquina CNC. Por otro lado, también se debe de obtener datos de la base sin cortar.

Primeramente, para la obtención los datos de las piezas, una forma sencilla es utilizando el comando EXTRACDAT de AutoCAD. Con él, se obtiene un fichero de texto con los datos que queremos de las plantillas, ya que el comando permite elegir los datos a extraer, lo que hace más sencillo acceder a ellos desde Python.

Los datos necesarios que se debe de extraer son el modelo de la plantillas, para realizar la clasificación, las coordenadas X e Y, y la rotación de cada una de las plantillas. A continuación, se va a explicar los pasos a seguir para obtener los datos de las plantillas, los cuales serán exportados a un archivo de texto.

- **Comando EXTRACDAT, crear una plantilla de datos:**

Al usar el comando EXTRACDAT de AutoCAD para extraer los datos es necesario haber creado previamente una plantilla de extracción de datos. En ella se indican los parámetros necesarios para que el archivo de texto que se creará nos muestre los datos necesarios.

Lo primero que se debe de hacer es abrir el modelo de AutoCAD del que se quieren extraer los datos del que se quiere obtener los datos y ejecutar el comando EXTRACDAT. Se trata de un comando con 8 pasos, los cuales nos permiten editar la plantilla para que se extraigan los datos necesarios de ciertos elementos que hay en el archivo. En la Figura 4.14 se puede ver el primer paso, en el que se debe seleccionar la opción de crear un nueva plantilla. A continuación le damos un nombre a la plantillas y a aceptar.

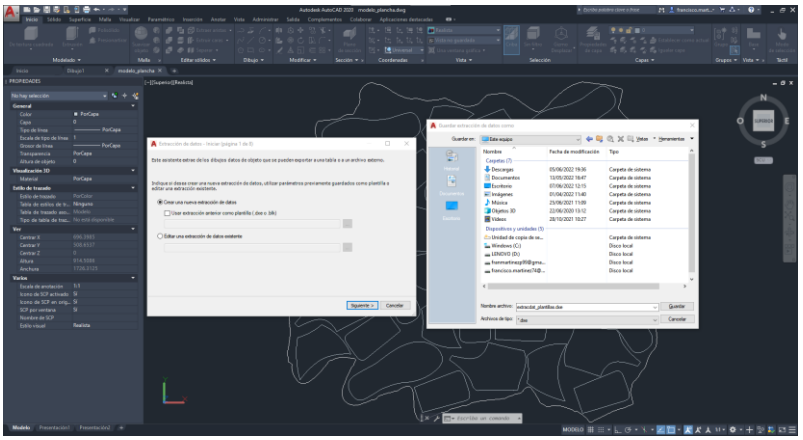


Figura 4.14: Paso 1, creación de nueva plantilla

En el paso dos, se debe seleccionar los dibujos de los cuales se quieren extraer datos, para ello, seleccionamos la casilla “Incluir dibujo actual” y damos a siguiente, como se puede observar en la Figura 4.15.

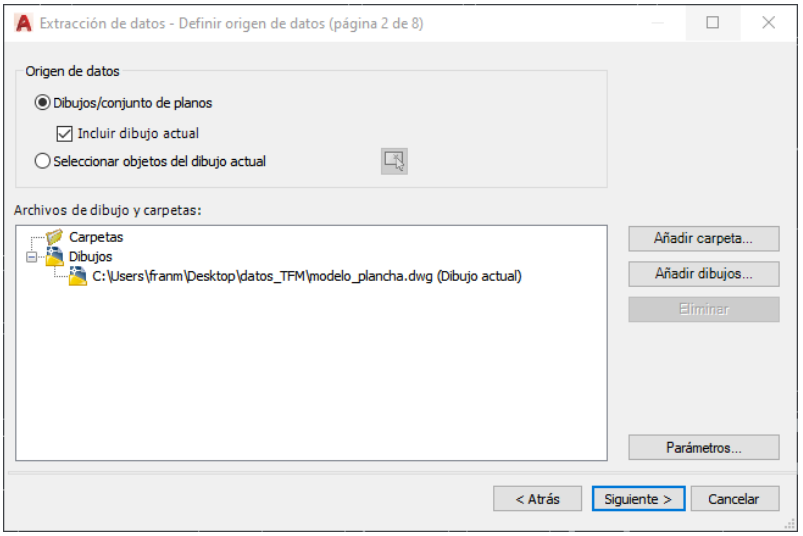


Figura 4.15: Paso 2, selección de dibujo

En el paso 3 del comando, se debe de seleccionar que objetos del dibujo se quiere obtener los datos. En este caso, se selecciona las casillas de todos los bloques que aparecen, como vemos en la Figura 4.16.

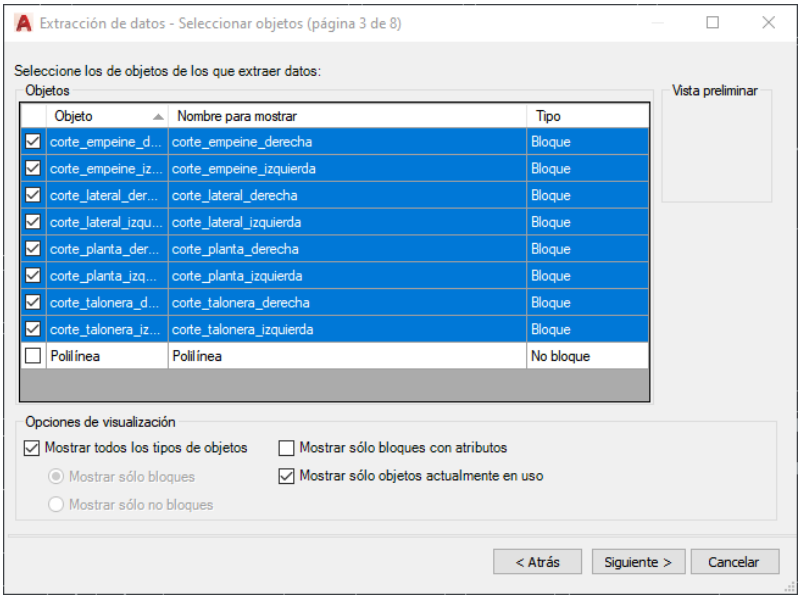


Figura 4.16: Paso 3, selección de objetos

El paso 4 es el más importante, en este paso se elige los datos que se quieren extraer del archivo. Primero, en la parte derecha seleccionamos los filtros de “Dibujo”, “Geometría” y “Varios”, y luego, se seleccionan solo las casillas necesarias, en esta caso son las de “Posición X”, “Posición Y” y “Rotación”, en la podemos verlo.

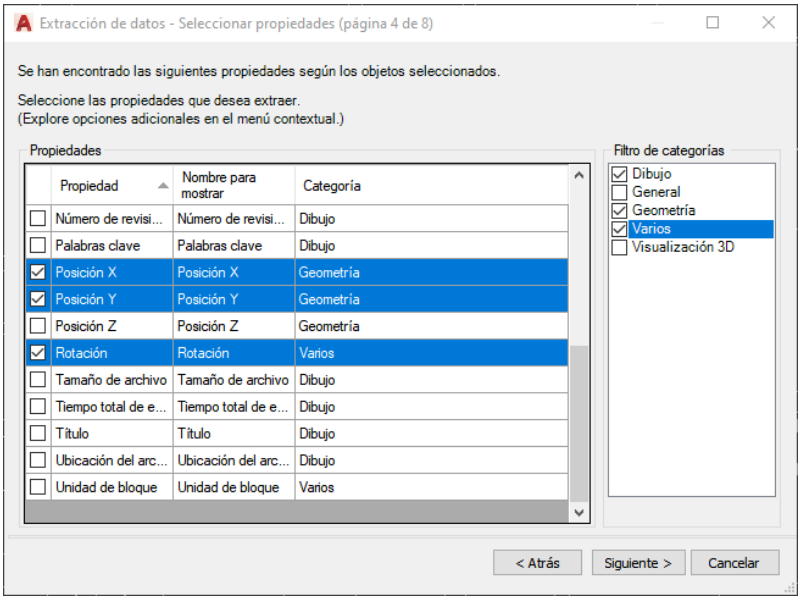


Figura 4.17: Paso 4, selección de los datos a extraer

En el paso 5 (Figura 4.18) se puede ver una vista previa de los datos que serán obtenidos.

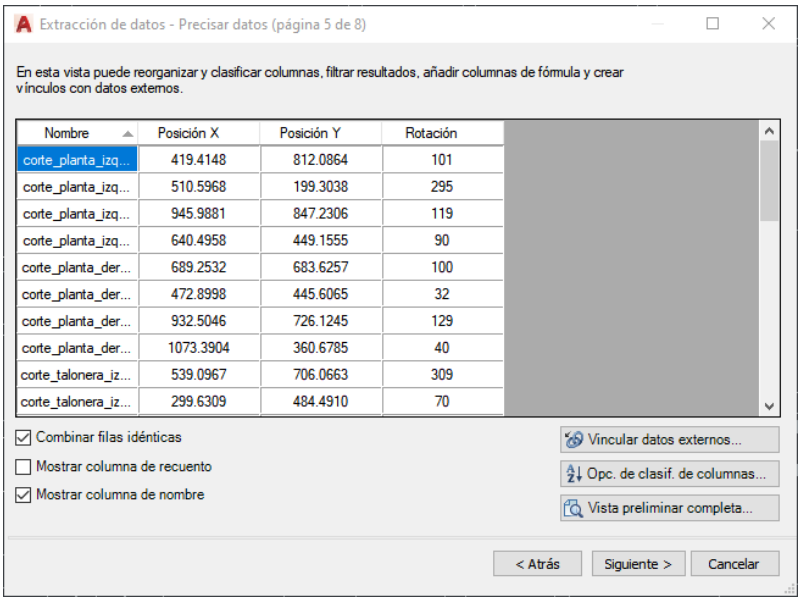


Figura 4.18: Paso 5, vista previa de datos

En el siguiente paso, el 6, es cuando se selecciona que queremos que lo extraiga como un archivo de texto. Para ello, se elige la casilla “Salida de datos a un archivo externo” y se abre el desplegable, donde podemos guardarlo con el nombre que se decida.

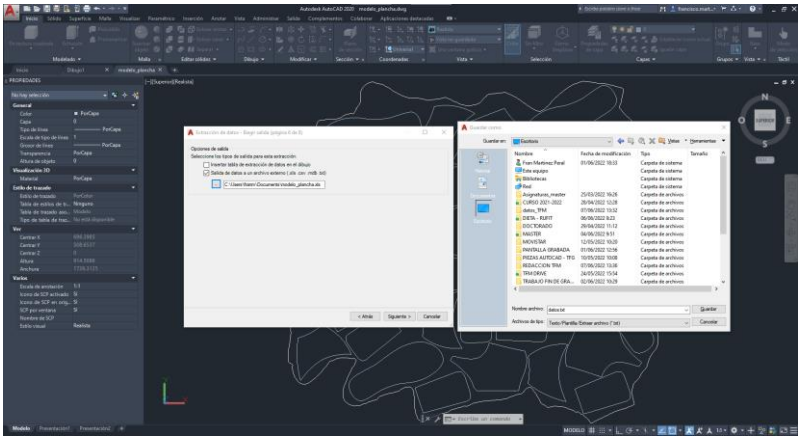


Figura 4.19: Paso 6, guardar en archivo externo

Finalmente, en el último paso, donde se finaliza el comando y son obtenidos los datos y guardados en el archivo de texto externo indicado.

4.5.2. Reconocimiento con Visión

El objetivo en esta parte era la localización de la pieza textil sobre la cinta transportadora y una vez se haya obtenido todos los datos de AutoCAD, estos se utilizarían para comprobar si la pieza se ha girado respecto a como fue cortada en la CNC, al mismo tiempo que se hacer la correspondencia de los sistemas de coordenadas de AutoCAD y de la cámara.

El algoritmo empieza leyendo la imagen que ha sido tomada durante la simulación y que se puede ver en la Figura 4.20.

Figura 4.20. Imagen tomada durante la simulación

Esta imagen es cortada de forma que se eliminar todas aquellas cosas alrededor de la pieza que no son interesante, como es el caso de los laterales de la cinta. De forma que la imagen se centra en la pies textil de la que debe extraer información, como podemos ver en la.

Figura 4.21. Imagen capturada recortada

Una vez que se tiene la imagen recortada y centrada en la pieza, se realizan varias transformación morfológicas para conseguir detectar el contorno de la pieza. A partir de dicho contorno se ha obtenido el centro del mismo con el que se puede hacer una primera relación de sistema de coordenadas con el centro obtenido en la extracción de datos de AutoCAD. Además también se pueden obtener el valor de los vectores propios del contorno, los cuales coinciden con los ejes de inercia de la figura que también son extraídos desde AutoCAD.

Con los datos anteriormente comentados se consigue calcular si la pieza ha sido girada respecto a AutoCAD. Al tener los vectores propios de los ejes tanto de AutoCAD como del contorno extraído de la imagen se pueden dibujar los ejes sobra la imagen y se calculan el ángulo que forman los ejes, con el que sabemos cuánto se ha girado la pieza.

En la Figura 4.22, se puede observar la imagen recortada y en ella se han dibujado el contorno de la pieza, el centro y los ejes, siendo en este caso el giro de 0 grados, por lo que la pieza no ha modificado su orientación a lo largo del movimiento de las cintas.

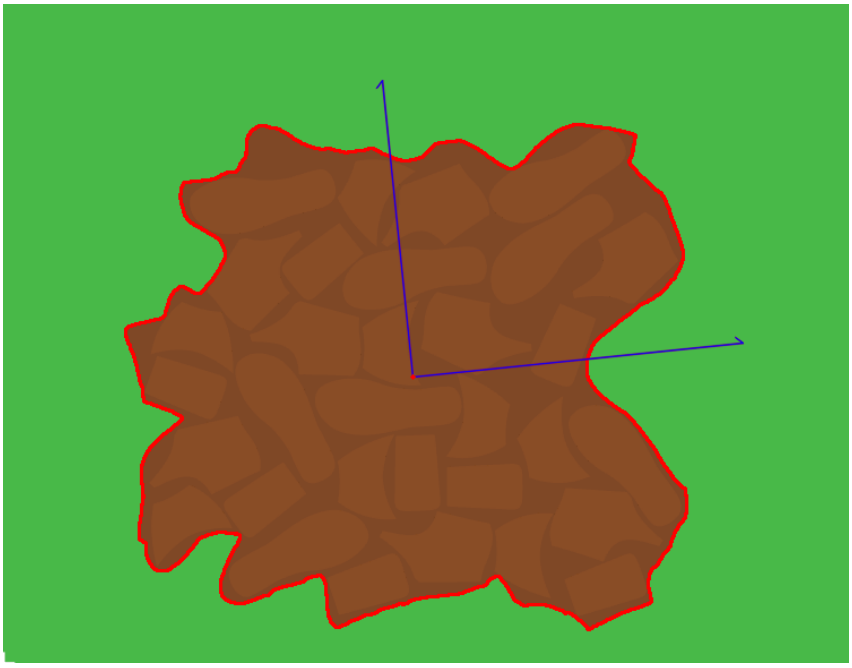


Figura 4.22. Imagen resultado

4.6. Programación en Python

En el presente apartado se va a explicar el funcionamiento de algoritmo creado en Python para el presente trabajo. El programa de Python es ejecutado en el momento en el que la pieza se encuentra en posición para tomar la imagen recoger las piezas.

Lo primero que se hace al ejecutar el algoritmo es establecer la conexión entre ambos programas mediante un socket, lo cual es lo más importante del proyecto, el conseguir esa conexión para el envío de los datos, en la Figura 4.23se puede ver los comandos para crear la conexión entre ambos programas.

```
# CONEXION SOCKET PYTHON-ROBOTSTUDIO
obj = socket.socket()
host = '127.0.0.1'
port = 1024
obj.connect((host, port))
# Esperamos respuesta de RobotStudio como que se ha creado la conexion
respuesta = obj.recv(1024)
print(str(respuesta))
print('Conexion socket creada con RobotStudio')
```

Figura 4.23. Conexión Python-RobotStudio mediante socket

Tras crear la conexión, lo primero que se hace es la extracción de datos explicada en el apartado 4.5.1 desde AutoCAD. A partir del nombre del fichero CAD se crea un script para poder ejecutarlo en el programa. En dicho script encontramos por un lado las instrucciones para ejecutar el comando “Extracdat” de AutoCAD con el que se extraen los datos definidos en la plantillas creada para dicho fin. Por otro lado, tenemos las instrucciones para obtener los datos de la base sin cortar. Dicha base está definida como polilínea, por lo que se deben de editar y convertir en región para posteriormente usar el comando “Propfis” y obtener los datos necesarios en un fichero de texto. En la se puede observar el script de AutoCAD creado desde Python.

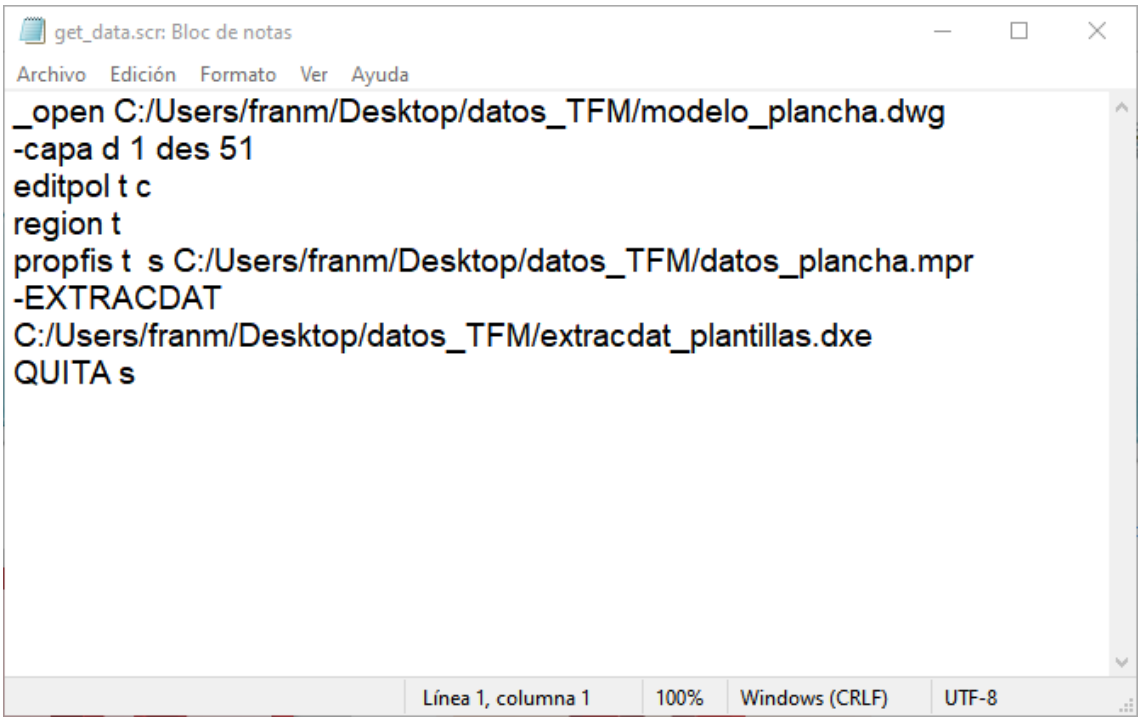


Figura 4.24. Script de AutoCAD para extracción de datos

Una vez se ha ejecutado el programa de AutoCAD en el ordenador, se ha abierto el fichero .dwg del modelo y ejecutado el script, cierra todo y se puede proceder a la lectura de los datos.

El siguiente paso del algoritmo es la lectura de ambos ficheros con los datos necesarios los cuales son guardados en listas de Python para su posterior uso.

A continuación, se inicia el proceso de reconocimiento con visión explicado en el apartado 4.5.2. En este proceso serán usados los datos de AutoCAD obtenidos de la base sin cortar.

A partir de la imagen tomada en la simulación de RobotStudio, se ha realizado una serie de transformaciones para detectar el contorno de la base y a partir de dicho contorno se ha obtenido el centro y los ejes de mayor inercia. Estos datos son también los que se han obtenido desde el fichero CAD.

Con los datos de los centros podemos recalcular las coordenadas de cada una de las piezas a recoger obteniendo la relación entre el sistema de coordenadas de AutoCAD y el de la imagen. Con los datos de los ejes se puede conocer cuánto se ha girado la pieza y a partir de dicho ángulo recalcular el giro de la herramienta para que recoja las piezas correctamente y las pueda ordenar todas de forma correcta.

Una vez recalculados todas las posiciones en las que se encuentran las piezas, pasamos a proceso de enviar los datos desde Python a RobotStudio. Este envío de datos se realiza mediante el socket, el cual se hizo la conexión al inicio de ejecución del algoritmo. Para el envío de todos los datos se han creado una codificación válida para todos los datos. Esta se compone de un string de doce dígitos, en el que el primero de ellos indica a RobotStudio de que modelo es la pieza que está recogiendo y con el que hará la clasificación.

Los siguientes cuatro dígitos corresponden al valor de la coordenada x sumándole un valor de mil. Se hace esto para evitar problemas a la hora de concatenar el string ya que hay piezas cuyo valor de coordenada x tiene tres cifras y otras cuatro y de esta forma todas tiene el mismo tamaña y después se corregirá en Rapid. Para la coordenada y se ha destinado tres cifras, ya que en este caso no hay problema de tamaño y en el rato del último dato, se han utilizado 4 dígitos. El primero de ellos es un “0” o un “1” indicando si el giro es negativo o positivo. El resto de los dígitos son el valor del ángulo girado más un valor de 101 para que todos los ángulos tuvieran tres cifras.

En la Figura 4.25 se puede observar un ejemplo de codificación de los datos de una plantilla.

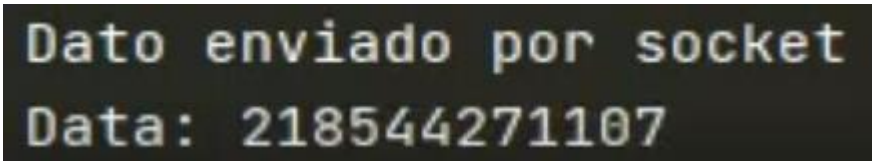


Figura 4.25. Ejemplo de codificación de datos

Los datos son enviados desde Python cuando se ha recibido una señal por parte de RobotStudio con el mensaje de “next_dato”. De esta forma cuando en RobotStudio sea necesario un nuevo dato para que el robot hago el proceso de Pick&Place de dicha pieza, se le comunicará a Python y este lo enviará evitando así introducir esperar en los programas lo que lo hace más eficiente y autónomo.

Tras enviar todos los datos de las piezas a recoger y finalizar el proceso de Pick&Place el programa finaliza su ejecución. En código de Python se puede encontrar en el Anexo 8.3.2.

4.7. Programación en RAPID

Por otro lado, se tiene el código de programación en Rapid en el programa de RobotStudio. Este programa ejecuta todos los comando de la simulación del robot para realizar la tarea de Pick&Place junto con la comunicación mediante el socket con Python.

En la pestaña de Rapid en RobotStudio encontramos un programa en el controlador de la estación. En él se ha programado todos los comandos e instrucciones para la simulación del proyecto, así como se han activado y desactivado las señales necesarias.

La primer aparte del programa se centra en definir todas aquellas variables que serán necesarias durante el resto del programa, como por ejemplo, las variables para la comunicación del socket o para almacenar los datos recibidos de las piezas a recoger, entre otras.

Tras iniciar la simulación, tenemos un primer proceso que se encarga de activar y desactivar la señales de “Marcha_cintas” y de “Paro” a partir de los valores de las señales de los sensores de la estación. Una vez que la pieza textil ha alcanzado el lugar de recogida finaliza este proceso y da paso al siguiente.

El siguiente paso es establecer la comunicación con el algoritmo de Python. Para ellos se crear y establece el socket a la espera de recibir una señal proveniente de Python que confirme la conexión. Tras una buena conexión, se inicia un bucle While que se ejecutará hasta que la variable socket_signal esté a 1 o que se finalice el proceso de recogida y se acabe el bucle con un Break.

Dentro del bucle, el robot se moverá a la posicion de Inicio, la cual se encuentra sobre la zona de recogida y comenzará el proceso para recibir los datos de las piezas. Se enviará una señal a Python cuando se requiera de los datos de la siguiente pieza y así sucesivamente, hasta recoger todas las piezas.

A partir de los datos recibidos de Python, estos se dividen para separar la cadena de valores recibida en los distintos datos de cada pieza, de modelo, coordenada X, coordenada Y y rotación. Estos datos son usados para crear los movimientos que debe de realizar el robot, así como de la clasificación por modelo que debe de hacer a la hora de clasificar las piezas.

La posiciones de dejada han sido definidas previamente de forma que las piezas queden clasificadas a los laterales de la base cortada para que un operario puede recogerlas ya clasificadas. En el Anexo 8.3 podemos encontrar el código de RAPID de la estación.

A continuación, tenemos un diagrama de flujo que explica el programa diseñado para la estación.

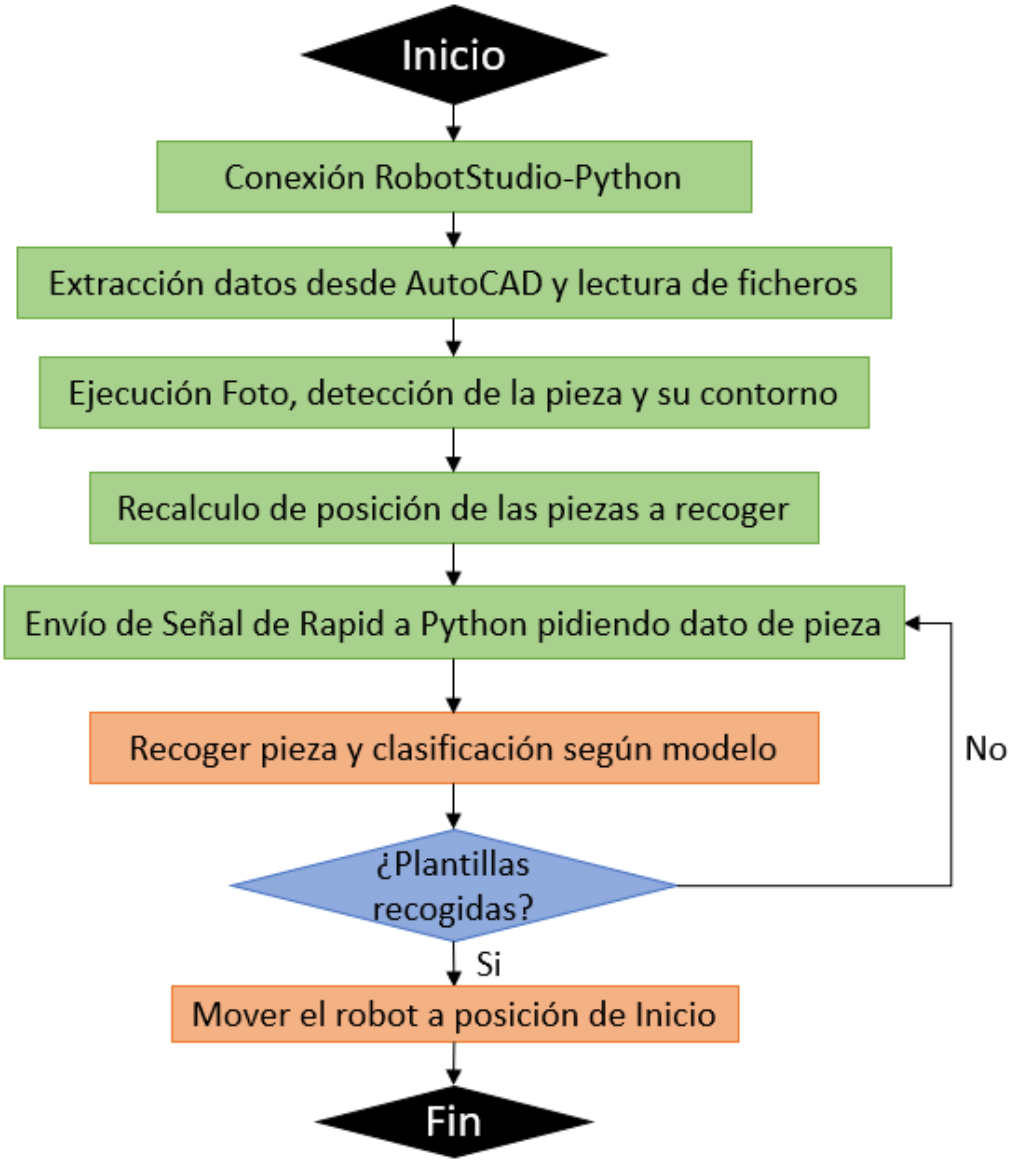


Figura 4.26: Diagrama de flujo de la simulación

5. RESULTADOS

5.1. Simulación estación Conexión Python-AutoCAD-Vision-RobotStudio

En este apartado se van a mostrar imágenes de la simulación de la estación del trabajo. En la Figura 5.1 se puede ver la estación completa antes de iniciarse la simulación.

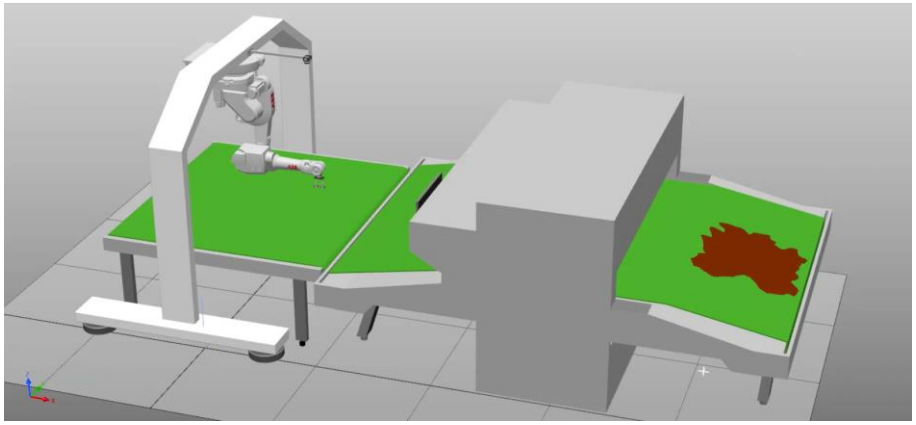


Figura 5.1: Estación al inicio de la simulación

En primer lugar, se introducirá la base sin cortar a la CNC para que esta la corte según el modelo de corte indicado, como se puede observar en la Figura 5.2.

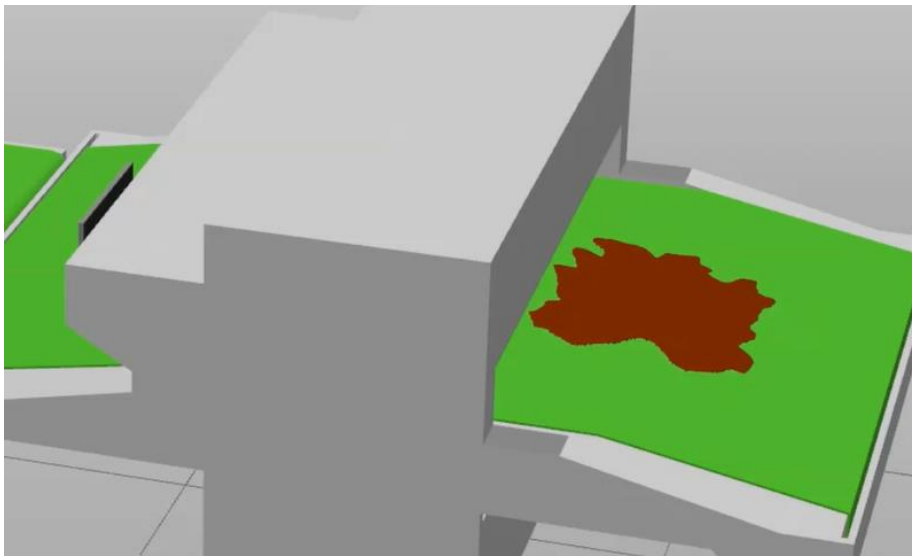


Figura 5.2: Introducción de la base sin cortar a la maquina CNC

Una vez que la máquina CNC ha cortado la base, las piezas se desplazan a lo largo del tapiz rodante hasta alcanzar la zona de recogida y se procede a tomar una imagen de la base cortada, como se puede ver en la Figura 5.3.

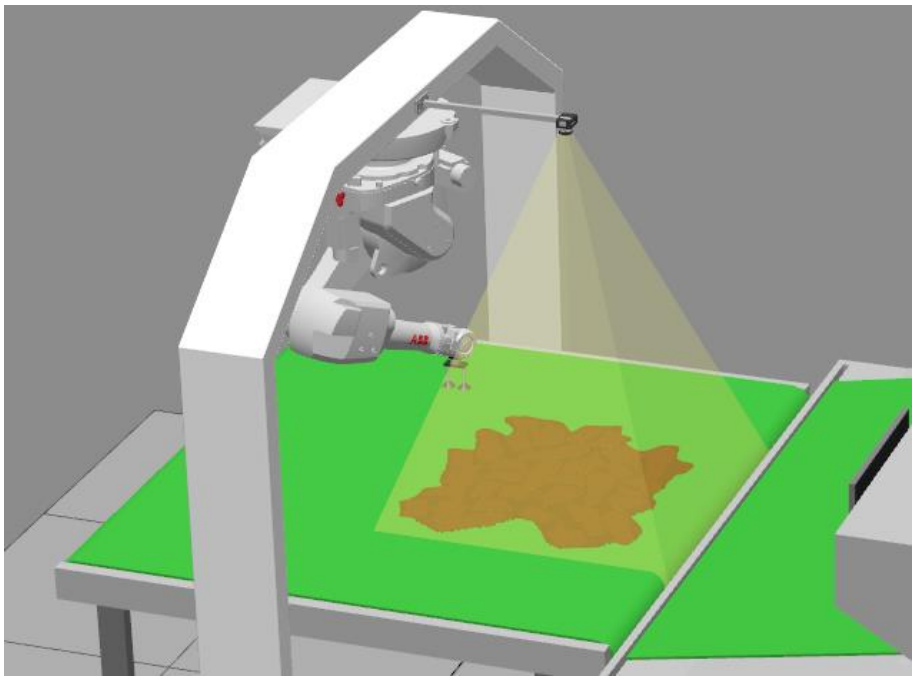


Figura 5.3: Captura de imagen de la base cortada en posición de recogida

Una vez se ha realizado la foto, se inicia el algoritmo de Python. El primer paso es la obtención de los datos, tanto de cada una de las piezas, como de la base sin cortar, desde AutoCAD. En la siguiente Figura 5.4 se puede ver la ejecución de AutoCAD durante la simulación y en la Figura 5.5 se pueden ver los datos extraídos.

Figura 5.4. Extracción de datos de AutoCAD

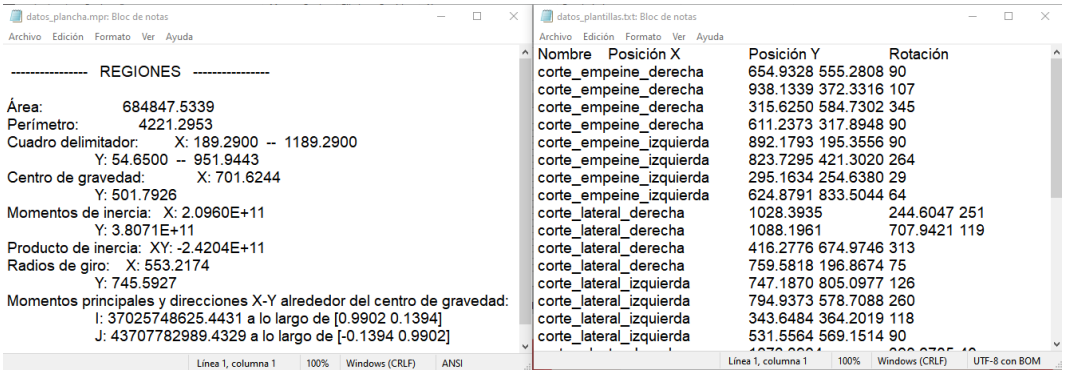


Figura 5.5 Datos extraídos de AutoCAD

Una vez se han extraídos y leídos los datos, comienza el proceso de recalcular todos los datos de las piezas a recoger para relacionarlos con los datos obtenidos de la imagen.

En la Figura 5.6 se puede observar la imagen obtenida a partir de la foto tomado donde el ángulo que se ha girado la pieza es de 0 grados.

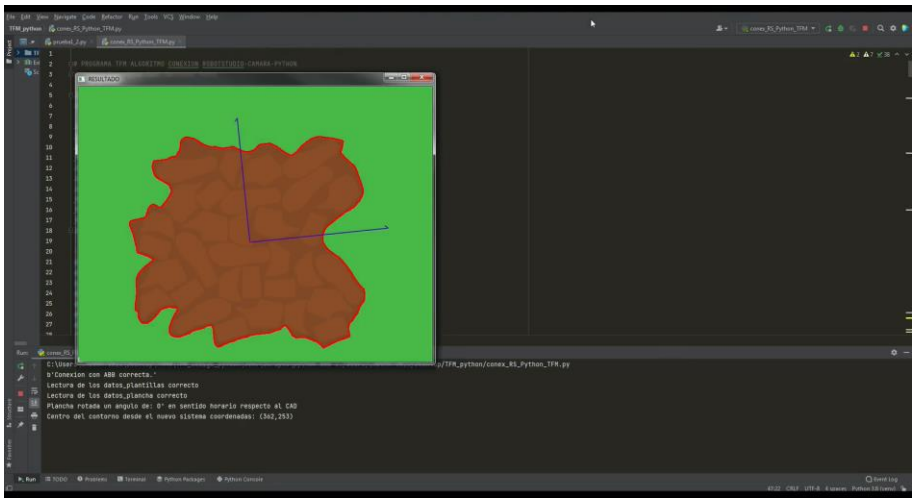


Figura 5.6. Resultado del reconocimiento con Visión

Tras haber recalculado todos los datos de las piezas, comienza el proceso del envío de los datos desde Python hasta RobotStudio. Para ello, el robot debe de haber se movido a la zona de espera para el inicio de la recogido.

Desde RobotStudio se envía la señal para que mediante la comunicación del socket se empiece a enviar datos. En la Figura 5.7 se puede observar el robot en posición a la espera de recibir el primer dato.

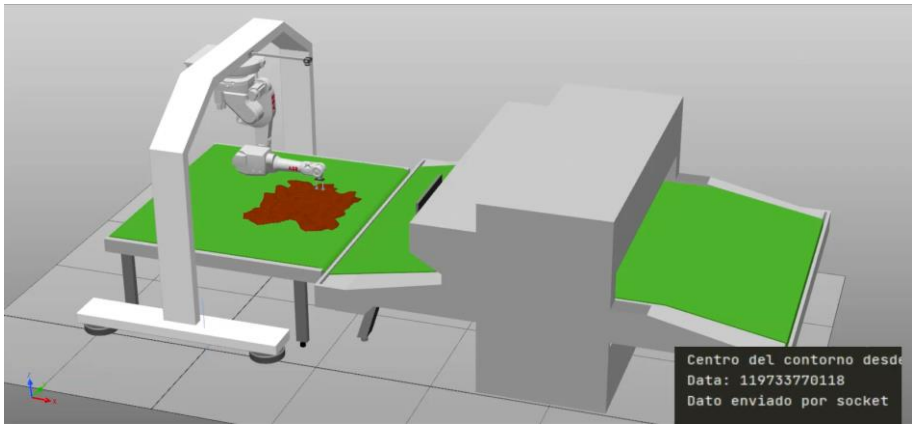


Figura 5.7. Robot en posición de espera y primer dato enviado

El robot realiza la tarea de Pick&Place de forma que cuando recoge una pieza, esta debe de ser depositada en un montón con el resto de las piezas que sen del mismo modelo de corte. De esta forma se consigue una correcta clasificación en el caso de tener tantos modelos cortados.

En la Figura 5.8 se puede ver al robot recogiendo una pieza y moviéndose al lugar de dejada correspondiente.

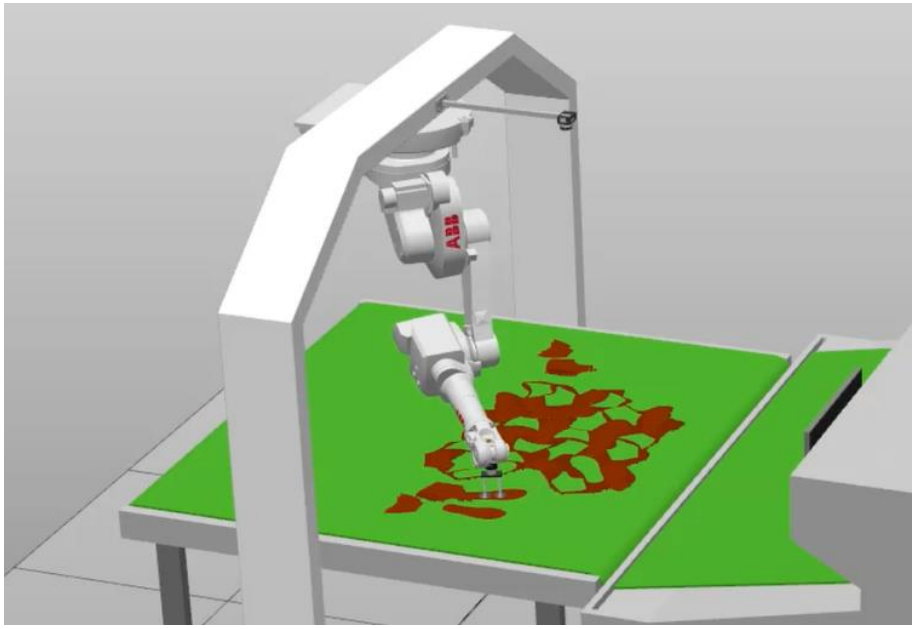


Figura 5.8. Robot recogiendo una pieza cortada

Finalmente, todas las piezas se han recogido de forma correcta depositando cada una de ellas en el montón con el resto de las piezas del mismo modelos y todas con la misma orientación.

En la Figura 5.9 se puede ver el nial del proceso con todas las plantillas recogidas correctamente.

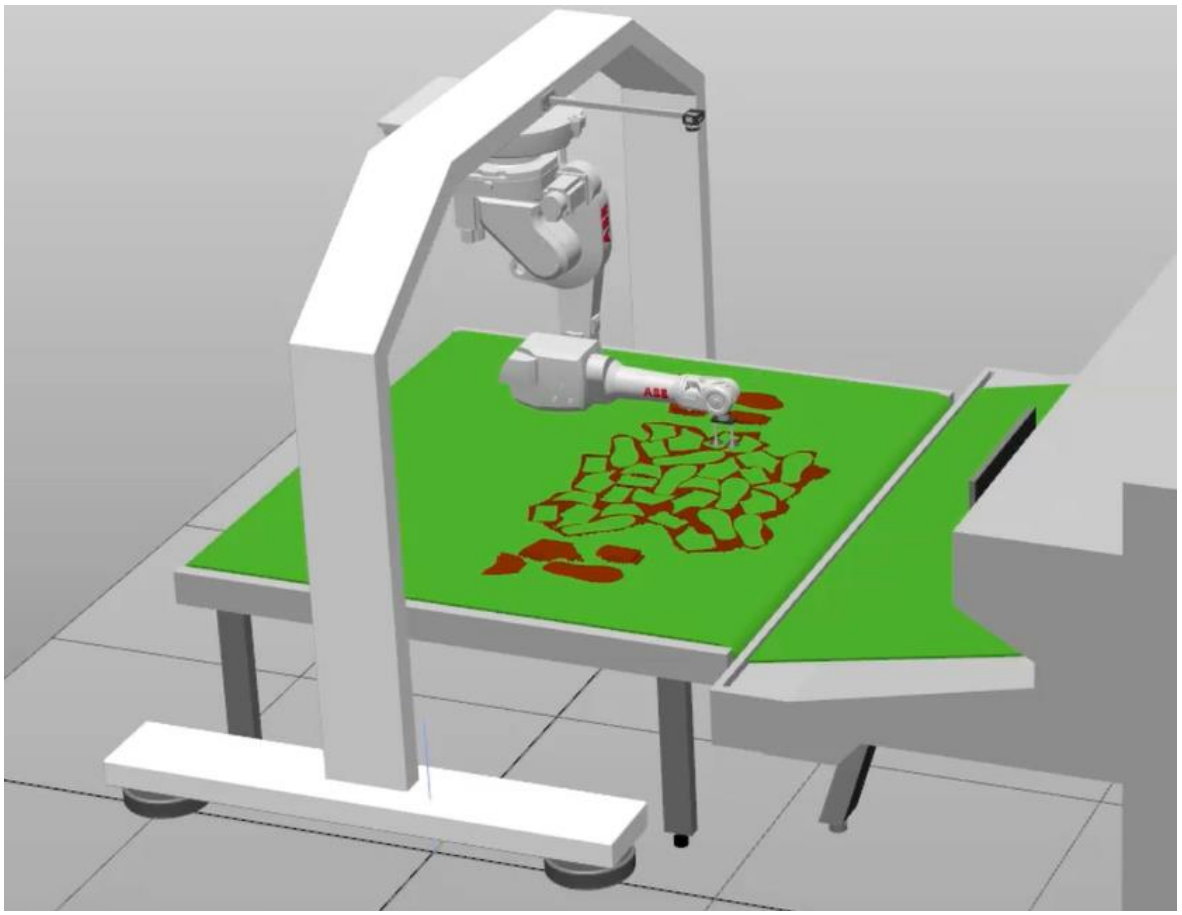


Figura 5.9: Finalización del proceso de Pick&Place

6. CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

6.1. Conclusiones

Una vez finalizado el proyecto de Fin de Máster, se puede concluir que se ha conseguido cumplir con los objetivos planteados al inicio del mismo.

El software de programación RobotStudio de la marca ABB es uno de los más utilizados en la programación de estaciones robotizas en la industria, así como de manera “offline” permitiendo realizar simulaciones cercanas a la realidad de manera virtual antes de ser aplicadas en la estación final.

Para conseguir los conocimientos necesario para utilizar el programa se ha tenido que seguir una serie de videotutoriales de aprendizaje sobre el uso de toda las funciones que ofrece el software. El programa ha permitido realizar la programación de dos estaciones robotizadas, uno a modo de tutorial y la otra como proyecto. Ambas se han basado en aplicaciones que tiene un gran uso en la industria, como son las tareas de Pick&Place.

Para el diseño de las estaciones se han utilizado diferentes objetos de la propia librería de RobotStudio, así como el uso de sensores, cintas, etc. Pero también se han diseñado otros elementos en el software AutoCAD 2020, como el pórtico para el robot, la base con las plantillas cortadas o herramienta con doble ventosa. Se ha conseguido finalmente la programación y simulación de un estación industrial con maquina CNC realizando el proceso de Pick&Place de piezas al mismo tiempo que son clasificadas de manera satisfactoria.

Uno de los objetivos era conseguir una mayor automatización del proyecto desarrollado durante el trabajo fin de grado. La introducción del reconocimiento por visión al algoritmo de Python ha conseguido esa mejora en la automatización. Además, la mejora de la comunicación del socket ha reducido el tiempo ue puede tardar el robot en leer todos los datos de un fichero de texto como hacia anteriormente.

Esta nueva comunicación desarrollada en el proyecto se asemeja más a con se comunican la mayoría de los robots implantado actualmente en la industria con ordenadores mediante sockets, por lo que el diseño desarrollado es mas fiel a la realidad.

Con todas estas mejoras, se ha conseguido un algoritmo capaz de aumentar la productividad del proceso, realizándolo de manera más autónoma y rápida, donde cabe destacar su gran flexibilidad para diversos modelos de corte de plantillas. Haciendo que el operario solamente deba esperar que todas la piezas sean recogidas al final de la cinta.

6.2. Líneas futuras de trabajo

Algunas de la posible mejoras del presente algoritmo son las siguientes. La primera de ellas seria la implementación del proceso desarrollado en el presente trabajo a una línea industrial automatizada real. Se debería de tener en cuenta los tiempos y métodos que se fija en dicha línea para poder introducirla, intentando conseguir una mayor optimización que el proceso tradicional.

Sería conveniente un estudio profundo de los diferentes lenguajes de programación utilizados, ya que tienen un gran margen de mejora para una mayor optimización del proceso.

En cuanto a la extracción de datos, sería conveniente la creación de una plantilla la cual pueda ser utilizada con todos los modelos que sean diseñados en una empresa. Incluso la posibilidad de no ser necesaria la utilización de AutoCAD para la extracción de datos, como por ejemplo alguna librería de Python que sea capaz de leer archivos .dwg.

Por último, la mejor en el reconocimiento de visión haría que el sistema fuera más estable en el caso de una estación real a cambios en la iluminación.

7. BIBLIOGRAFÍA

[1] Antonio Barrientos, Luís Felipe Peñín, Carlos Balaguer y Rafael Aracil. (1997) Fundamentos de Robótica. Universidad Politécnica de Madrid. 342 p.

[2] Tobe, Frank. “Why Co-Bots Will Be a Huge Innovation and Growth Driver for Robotics Industry”. 30 December 2015. IEEE Spectrum (https://spectrum.ieee.org/collaborative-robots-innovation-growth-driver)

[3] John R. Henry CPP, FRSA. “Los Cobots están dispuestos a trabajar contigo”. Octubre 2015, Tecnología del plástico. (https://www.plastico.com/temas/Los-cobots-estan-dispuestos-a-trabajar-contigo+108382)

[4] https://www.edsrobotics.com/blog/evolucion-robotica-industrial/

[5] https://www.edsrobotics.com/blog/tipos-robots-industriales-usos/

[6] https://new.abb.com/products/robotics/es/robots-colaborativos/yumi

[7] https://new.abb.com/products/robotics/es/robots-colaborativos/crb-15000

[8] https://www.universal-robots.com/es/productos/robot-ur10/

[9] https://industrial.omron.es/es/products/robotics

[10] ABB Robotics. (2018) Manual del operador. RobotStudio 6.10. 666 p.

[11] ABB Robotics. (2018) Manual de referencia técnica. Instrucciones, funciones y tipos de datos de RAPID. RobotWare 6.08. 1876 p.

[12] ABB Robotics. (2019) IRB 120 Industrial Robot. 2 p.

[13] ABB Robotics. (2020) IRB 1600 Industrial Robot. 2 p.

Videotutoriales:

https://new.abb.com/products/robotics/es/robotstudio/tutoriales

https://www.youtube.com/watch?v=4iHHf1-veJY&list=PLkEmFyPTnhOmGDFKSvOlJuuxh4c4sVpTy

https://www.youtube.com/watch?v=oDnHaVMZjp4&list=PLVdvHpsfqw1bX194j7Slup6ri5se2668p

https://www.youtube.com/watch?v=6Qks8TGY-_Y

https://www.youtube.com/watch?v=_1_uF7B5F80&list=PLv406soplrbEiggQnTOsNZaEsNxePjNM-&index=7

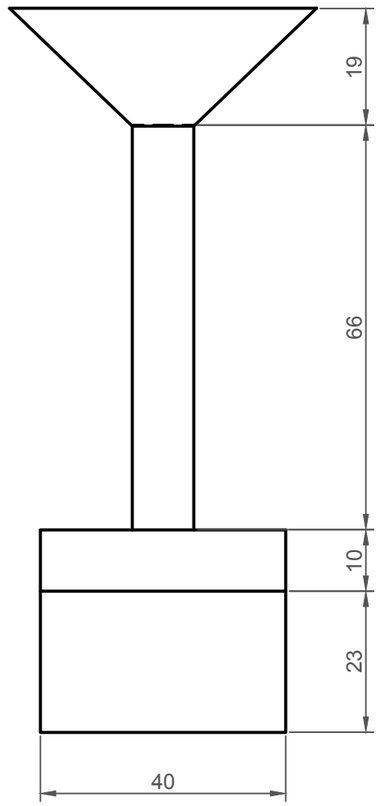
https://www.youtube.com/watch?v=8VvPgNnfgkU&list=PLv406soplrbEiggQnTOsNZaEsNxePjNM-&index=9

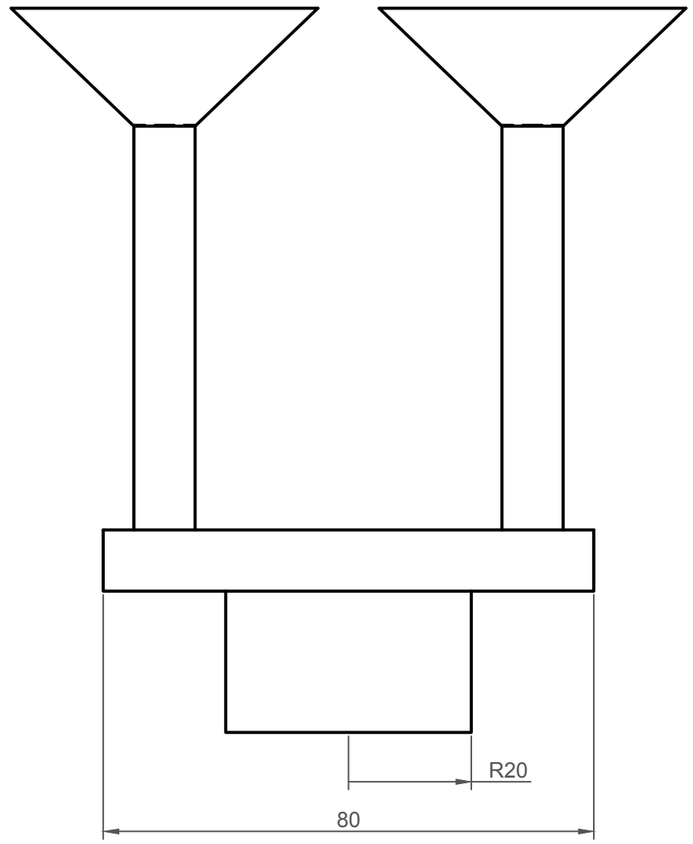
https://www.youtube.com/watch?v=f6O2fyXF_5E&list=PLAzmBEJ4XYtitogcgaWsc5S3ImNpXkr30

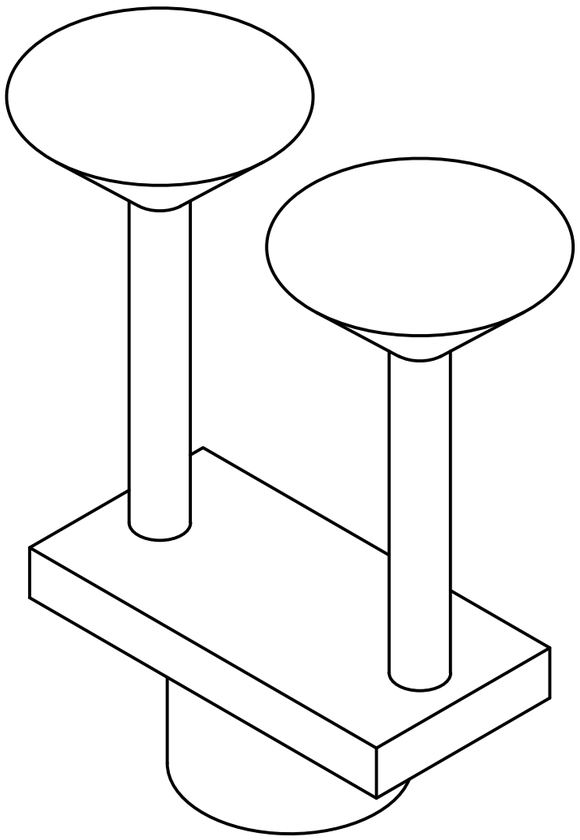
8. ANEXOS

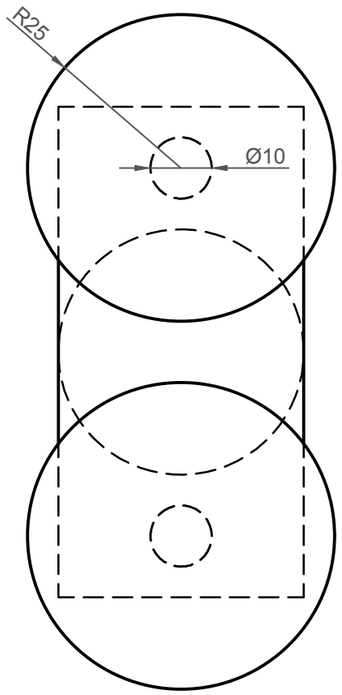
8.1. Planos

8.1.1. Planos de la estación conexión Python-AutoCAD-Vision-RobotStudio

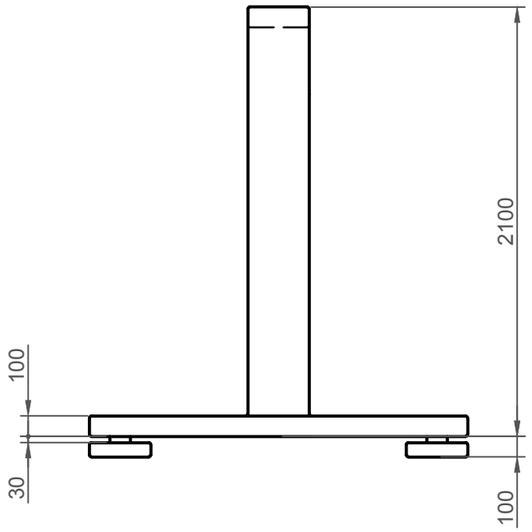


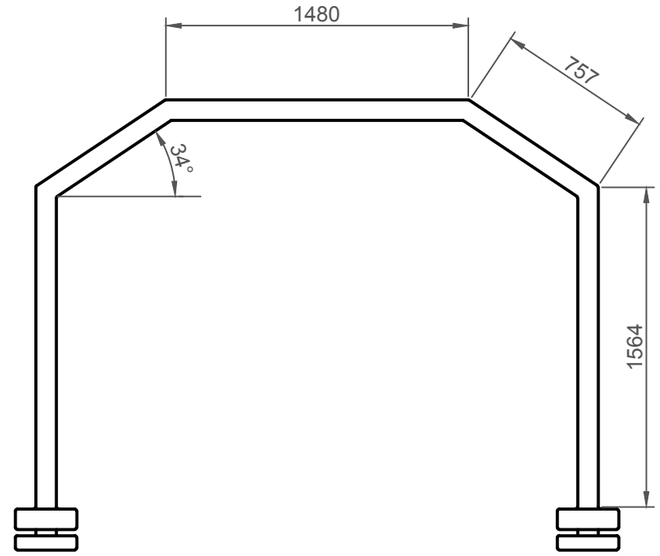


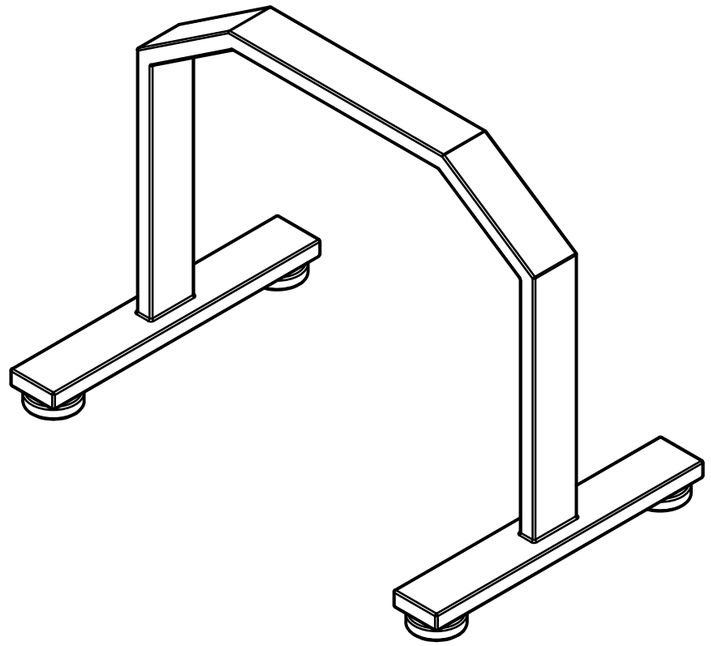




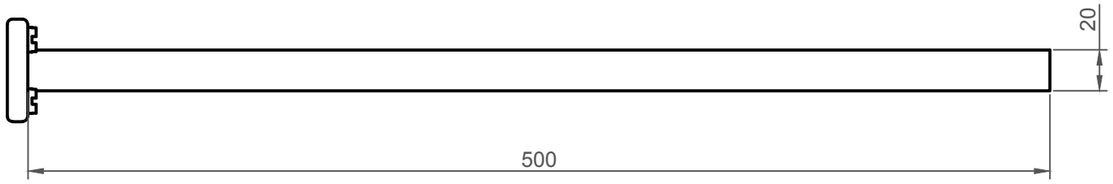
Plano: PLANO 1. HERRAMIENTA VENTOSAS	Proyecto: Trabajo Fin de Máster		Página: 82
Diseño de: FRANCISCO JOSÉ MARTÍNEZ PERAL	Fecha: Julio 2022	Plano Nº: 1	Formato: A3

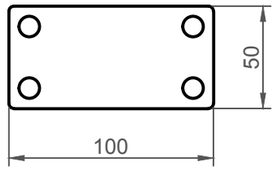


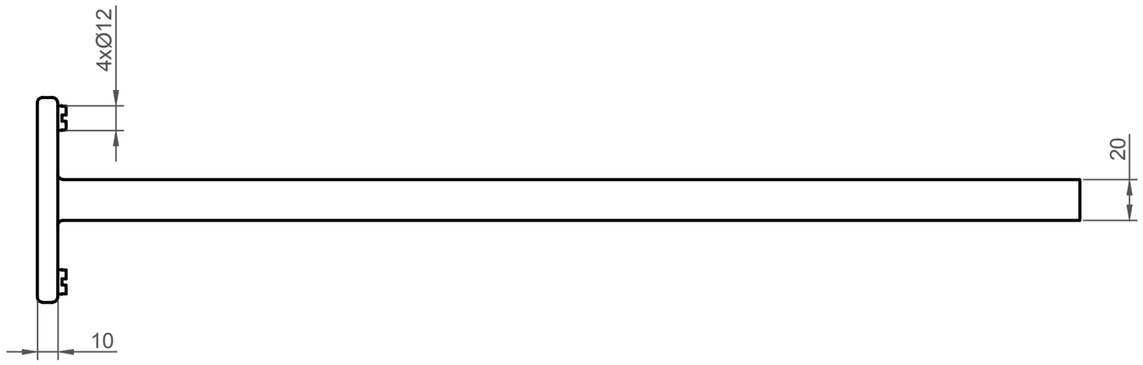


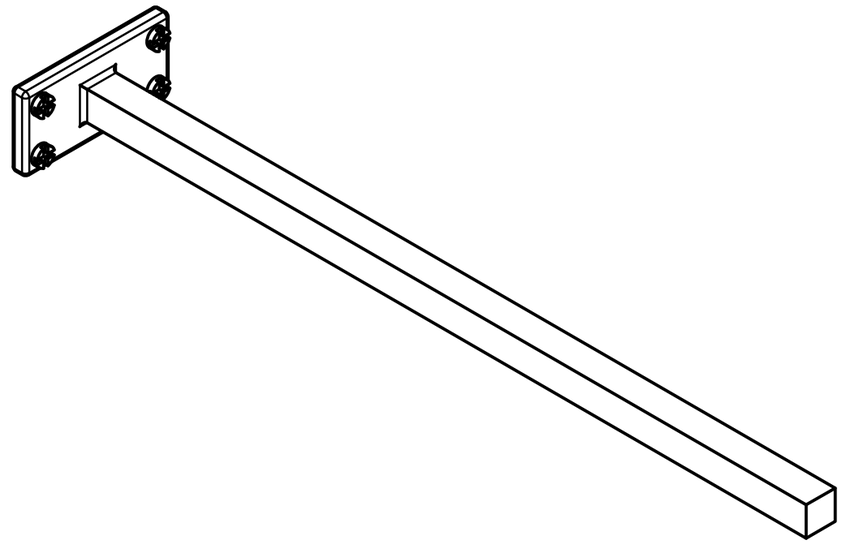


Plano: PLANO 2. PÓRTICO ROBOT	Proyecto: Trabajo Fin de Máster		Página: 83
Diseño de: FRANCISCO JOSÉ MARTÍNEZ PERAL	Fecha: Julio 2022	Plano Nº: 2	Formato: A3

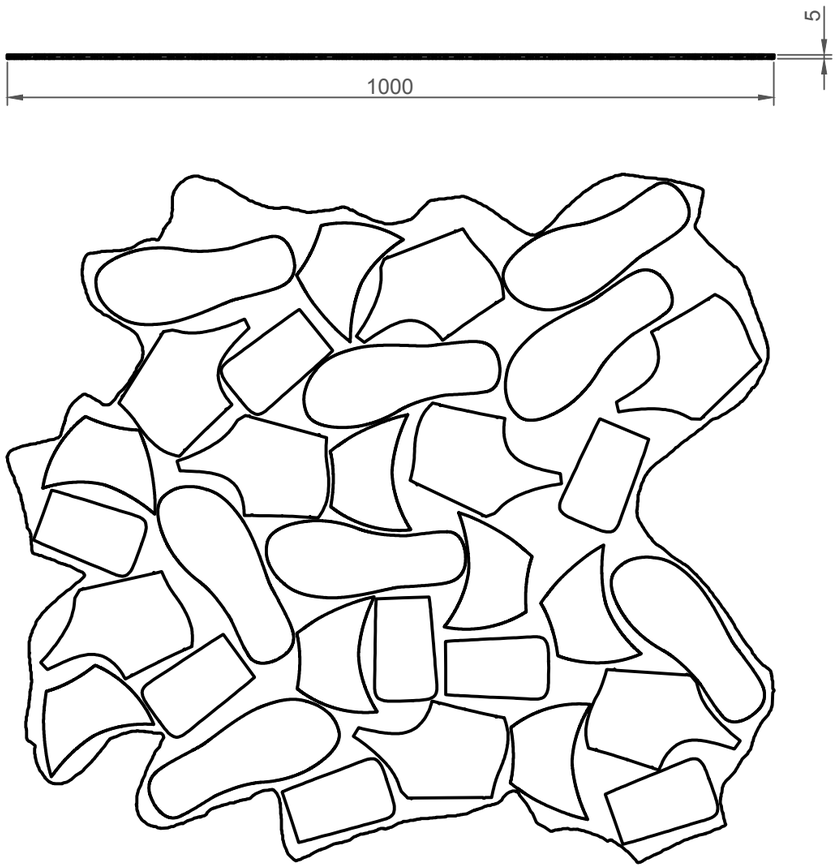


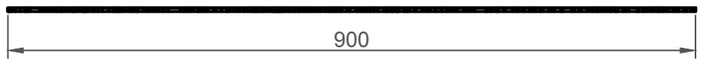


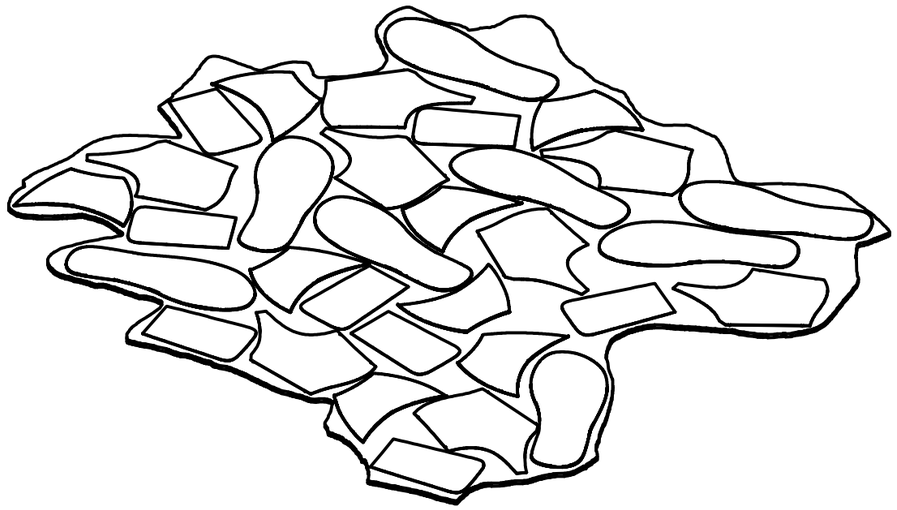




Plano: PLANO 3. SOPORTE CÁMARA	Proyecto: Trabajo Fin de Máster		Página: 84
Diseño de: FRANCISCO JOSÉ MARTÍNEZ PERAL	Fecha: Julio 2022	Plano Nº: 3	Formato: A3







Plano: PLANO 4. PLANCHA PLANTILLAS	Proyecto: Trabajo Fin de Máster		Página: 85
Diseño de: FRANCISCO JOSÉ MARTÍNEZ PERAL	Fecha: Julio 2022	Plano Nº: 4	Formato: A3

8.2. Señales

8.2.1. Señales de la estación de Pick&Place de piezas

Módulo de E/S. Controlador de los robots IRB1600_6/1.2			
Entradas digitales	**Mapeo**	**Salidas digitales**	**Mapeo**
Sensor_cinta	0	Marcha_cinta	10
Sensor_Foto	1	Paro	11
Sensor_CNC	2	Coger_pieza	12

Tabla 8.1: Señales E/S estación Pick&Place de piezas

8.3. Programas

8.3.1. Código RAPID de la Estación tutorial

```
MODULE Module1
    CONST robtarget Inicio:=[[547.330665099,0,451.64730631],[0,0,1,0],[0,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Pick_aprox:=[[475,275,350],[0,0,1,0],[0,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Pick:=[[475,275,150],[0,0,1,0],[0,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Place_aprox:=[[475,-275,350],[0,0,1,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Place:=[[475,-275,150],[0,0,1,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    !***********************************************************
    !
    ! Módulo:  Module1
    !
    ! Descripción:
    !   <Introduzca la descripción aquí>
    !
    ! Autor: LABORATORIO
    !
    ! Versión: 1.0
    !
    !***********************************************************
    PROC main()
        !Añada aquí su código
        SetDO Activa_ventosa,0;
        MoveL Inicio,v1000,z100,MyTool\WObj:=wobj0;
        MoveL Pick_aprox,v1000,z100,MyTool\WObj:=wobj0;
        MoveL Pick,v1000,fine,MyTool\WObj:=wobj0;
        SetDO Activa_ventosa,1;
        MoveL Pick_aprox,v1000,z100,MyTool\WObj:=wobj0;
        MoveL Place_aprox,v1000,z100,MyTool\WObj:=wobj0;
        MoveL Place,v1000,fine,MyTool\WObj:=wobj0;
        SetDO Activa_ventosa,0;
        MoveL Place_aprox,v1000,z100,MyTool\WObj:=wobj0;
        MoveL Inicio,v1000,z100,MyTool\WObj:=wobj0;
    ENDPROC
ENDMODULE
```

8.3.2. Código de Python conexión Python-AutoCAD-Vision-RobotStudio

```
# PROGRAMA TFM ALGORITMO CONEXION ROBOTSTUDIO-CAMARA-PYTHON CON VISION
# Autor: Francisco José Martínez Peral
# Fecha: Julio 2022

import sys
import time
import cv2
import subprocess
import os
from os import remove, mkdir
import numpy as np
import argparse
import math
from math import atan2, cos, sin, sqrt, pi
from matplotlib import pyplot as plt
from shutil import rmtree
import socket

# Variables globales
global modelo
global cgx
global cgy
global cX
global cY
global foto
global alpha
global tipo
global mtx
global dist
global num_frames
global v1
global v2
global eigenvectors
global eigenvalues
global angulo
global datos_plantillas

# CONEXION SOCKET PYTHON-ROBOTSTUDIO
obj = socket.socket()
host = '127.0.0.1'
port = 1024
obj.connect((host, port))
# Esperamos respuesta de RobotStudio como que se ha creado la conexion
respuesta = obj.recv(1024)
print(str(respuesta))
print('Conexion socket creada con RobotStudio')


def draw_axis(p_, q_, colour, scale):
    """
    Proceso que dibuja dos ejes, introduciéndole los dos puntos de ambos extremos de los ejes
    :param p_:
    :param q_:
    :param colour:
    :param scale:
    :return:
```

```
    """
    global foto
    # Create list for data
    p = list(p_)
    q = list(q_)
    # Angle and hypotenuse calculation
    angle = atan2(p[1] - q[1], p[0] - q[0])  # angle in radians
    hypotenuse = sqrt((p[1] - q[1]) * (p[1] - q[1]) + (p[0] - q[0]) * (p[0] - q[0]))
    # Here we lengthen the arrow by a factor of scale
    q[0] = p[0] - scale * hypotenuse * cos(angle)
    q[1] = p[1] - scale * hypotenuse * sin(angle)
    cv2.line(foto, (int(p[0]), int(p[1])), (int(q[0]), int(q[1])), colour, 1, cv2.LINE_AA)
    # Create the arrow hooks
    p[0] = q[0] + 9 * cos(angle + pi / 4)
    p[1] = q[1] + 9 * sin(angle + pi / 4)
    cv2.line(foto, (int(p[0]), int(p[1])), (int(q[0]), int(q[1])), colour, 1, cv2.LINE_AA)


def extrac_datos_autocad():
    """
    Funcion que no elimina los ficheros que se van a crear para AutoCAD si ya existen
    :return:
    """
    global modelo

    # Eliminamos los archivos datos.txt y get_data.scr para no tener porblemas la siguiente vez que
queramos obtener datos desde otro modelo de plantila.
    if os.path.exists('C:/Users/franm/Desktop/datos_TFM/datos_plantillas.txt') is True:
        remove('C:/Users/franm/Desktop/datos_TFM/datos_plantillas.txt')
    if os.path.exists('C:/Users/franm/Desktop/datos_TFM/get_data.scr') is True:
        remove('C:/Users/franm/Desktop/datos_TFM/get_data.scr')
    if os.path.exists('C:/Users/franm/Desktop/datos_TFM/datos_plancha.mpr') is True:
        remove('C:/Users/franm/Desktop/datos_TFM/datos_plancha.mpr')
    # Script para AutoCAD
    script = ['_open C:/Users/franm/Desktop/datos_TFM/' + modelo + '.dwg \n', '-capa d 1 des 51
\n', 'editpol t c \n', 'region t \n',
              'propfis t  s C:/Users/franm/Desktop/datos_TFM/datos_plancha.mpr\n', '-EXTRACDAT
\n', 'C:/Users/franm/Desktop/datos_TFM/extracdat_plantillas.dxe \n',
              'QUITA s\n']
    # Creamos el script get_data.scr con el archivo DWG que se debe de abrir para obtener los
datos
    with open('C:/Users/franm/Desktop/datos_TFM/get_data.scr', 'w+') as fich:
        fich.writelines(script)
    # Ejecutamos AutoCAD en el ordenador y tambien el scirpt creado para obtener los datos
    subprocess.run(['C:/Program Files/Autodesk/AutoCAD 2020/acad.exe', '/b',
'C:/Users/franm/Desktop/datos_TFM/get_data.scr'])
    print('Proceso de obtencion de datos de AutoCAD finalizado con exito.')
    return


def lectura_datos_plantillas():
    """
    Proceso para leer todos los datos de cada una de las plantillas
    :return: datos
    """
    global datos_plantillas
    # Leemos el archivo generado con los datos (datos.txt)
    fich = open('C:/Users/franm/Desktop/datos_TFM/datos_plantillas.txt', 'r')
    read_fich = fich.readlines()
    fich.close()
```

```
    # Se almacenan los datos en una lista de Python (datos)
    datos_plantillas = []
    for i in range(len(read_fich)):
        if i == 0:
            pass
        else:
            datos_plantillas.append(read_fich[i].rstrip().split('\t'))
    # Creamos un fichero data.txt
    for i in range(len(datos_plantillas)):
        datos_plantillas[i][0] = datos_plantillas[i][0].replace(' ', '_')
        datos_plantillas[i][0] = datos_plantillas[i][0].replace('(', '')
        datos_plantillas[i][0] = datos_plantillas[i][0].replace(')', '')
        datos_plantillas[i][1] = datos_plantillas[i][1][0:len(datos_plantillas[i][1])]
        datos_plantillas[i][2] = datos_plantillas[i][2][0:len(datos_plantillas[i][2])]
        datos_plantillas[i][3] = datos_plantillas[i][3][0:len(datos_plantillas[i][3])]

    print("Lectura de los datos_plantillas correcto")
    return datos_plantillas


def lectura_datos_plancha():
    """
    Proceso para realizar la lectura de los datos de la plancha. Obtiene el centro de gravedad de la
plancha y los ejes de los vectores de inercia
    """
    global modelo
    global cgx
    global cgy
    global v1
    global v2
    global foto
    # Leemos los datos de la plancha y dibujamos los ejes sobre el centro de gravedad
    datos_plancha = []
    with open('C:/Users/franm/Desktop/datos_TFM/datos_plancha.mpr', 'r') as f1:
        fichero1 = f1.readlines()
    for i in range(len(fichero1)):
        datos_plancha.append(fichero1[i].rstrip().split(' '))
    plancha = []
    for j in range(len(datos_plancha)):
        if j < 3:
            pass
        else:
            plancha.append(datos_plancha[j])
    # Lectura centro de gravedad
    cgx = float(plancha[4][len(plancha[4]) - 1])
    cgy = float(plancha[5][len(plancha[5]) - 1])
    # Lectura de los vectores de los ejes principales de inercia
    v1 = [float(plancha[len(plancha) - 2][len(plancha[len(plancha) - 2]) - 2][1:]),
          float(plancha[len(plancha) - 2][len(plancha[len(plancha) - 2]) - 1][:6])]
    v2 = [float(plancha[len(plancha) - 1][len(plancha[len(plancha) - 1]) - 2][1:]),
          float(plancha[len(plancha) - 1][len(plancha[len(plancha) - 1]) - 1][:6])]
    print("Lectura de los datos_plancha correcto")


def encuentra_contornos():
    """
    Proceso que encuentra el contorno de mayor area (la plancha) en la foto
    :return: contours
    """
    global foto
```

```
    gray = cv2.cvtColor(foto, cv2.COLOR_BGR2GRAY)
    _, bw = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)
    contours, _ = cv2.findContours(bw, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    return contours


def biggest_area(contours):
    """
    Proceso para encontrar la mayor area (contorno plancha)
    :param contours:
    :return:
    """
    prev_area = 0
    for i, c in enumerate(contours):
        area = cv2.contourArea(c)
        if area > prev_area:
            prev_area = area
            indice = i
        else:
            continue
    return indice, c


def calculo_centro_gravedad(c):
    """
    Proceso que calcula el centro de gravedad del contorno
    :param c:
    :return:
    """
    global cX
    global cY
    # Calculo de los momentos y centroide (centro de gravedad) del contorno
    M = cv2.moments(c)
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])


def get_orientation(c):
    """
    Proceso que obtiene los vectores del contorno encontrado en la foto
    :param c:
    :return:
    """
    global cX
    global cY
    global foto
    global v1
    global v2
    global eigenvectors
    global eigenvalues
    # Pca analysis
    sz = len(c)
    data_pts = np.empty((sz, 2), dtype=np.float64)
    for i in range(data_pts.shape[0]):
        data_pts[i, 0] = c[i, 0, 0]
        data_pts[i, 1] = c[i, 0, 1]
    # Center (mean), eigenvector and eigenvalues calculation
    mean = np.empty(0)
```

```
    mean, eigenvectors, eigenvalues = cv2.PCACompute2(data_pts, mean)
    return eigenvectors, eigenvalues


def calculo_axis():
    global modelo
    global cgx
    global cgy
    global cX
    global cY
    global foto
    global v1
    global v2
    global eigenvectors
    global eigenvalues
    # Centre
    cntr = (cX, cY)

    if eigenvectors[0, 0] > eigenvectors[0, 1]:
        if eigenvectors[0, 1] > 0:
            dif = abs(eigenvectors[0, 0] - eigenvectors[0, 1])
        else:
            dif = abs(eigenvectors[0, 0] + eigenvectors[0, 1])
    else:
        if eigenvectors[0, 0] > 0:
            dif = abs(eigenvectors[0, 0] - eigenvectors[0, 1])
        else:
            dif = abs(eigenvectors[0, 0] - eigenvectors[0, 1])

    if eigenvectors[0, 0] > eigenvectors[0, 1]:
        if eigenvectors[0, 1] > 0 and dif > abs(eigenvectors[0, 1]):
            eigenvec1 = round(abs(eigenvectors[0, 1]), 1)
            eigenvec2 = round(abs(eigenvectors[0, 0]), 1)
            p1 = (cntr[0] + 0.01 * eigenvec1 * eigenvalues[0, 0],
                  cntr[1] - 0.01 * eigenvec2 * eigenvalues[0, 0])
            p2 = (cntr[0] + 0.01 * -eigenvec2 * eigenvalues[1, 0],
                  cntr[1] - 0.01 * eigenvec1 * eigenvalues[1, 0])
        elif eigenvectors[0, 1] > 0 and dif > abs(eigenvectors[0, 1]) / 2:
            eigenvec1 = round(abs(eigenvectors[0, 1]), 1)
            eigenvec2 = round(abs(eigenvectors[0, 0]), 1)
            p1 = (cntr[0] + 0.01 * eigenvec1 * eigenvalues[0, 0],
                  cntr[1] - 0.01 * eigenvec2 * eigenvalues[0, 0])
            p2 = (cntr[0] + 0.01 * -eigenvec2 * eigenvalues[1, 0],
                  cntr[1] - 0.01 * eigenvec1 * eigenvalues[1, 0])
        elif eigenvectors[0, 1] > 0 and dif < abs(eigenvectors[0, 1]) / 2:
            eigenvec1 = round(abs(eigenvectors[0, 1]), 1)
            eigenvec2 = round(abs(eigenvectors[0, 0]), 1)
            p1 = (cntr[0] + 0.01 * eigenvec1 * eigenvalues[0, 0],
                  cntr[1] - 0.01 * eigenvec2 * eigenvalues[0, 0])
            p2 = (cntr[0] + 0.01 * -eigenvec2 * eigenvalues[1, 0],
                  cntr[1] - 0.01 * eigenvec1 * eigenvalues[1, 0])
        elif eigenvectors[0, 1] < 0:
            eigenvec1 = round(abs(eigenvectors[0, 0]), 1)
            eigenvec2 = round(abs(eigenvectors[0, 1]), 1)
            p1 = (cntr[0] + 0.01 * eigenvec1 * eigenvalues[0, 0],
                  cntr[1] + 0.01 * -eigenvec2 * eigenvalues[0, 0])
            p2 = (cntr[0] + 0.01 * -eigenvec2 * eigenvalues[1, 0],
                  cntr[1] - 0.01 * eigenvec1 * eigenvalues[1, 0])
    elif eigenvectors[0, 0] < eigenvectors[0, 1]:
        if dif < abs(eigenvectors[0, 0]) / 2:
```

```
            eigenvec1 = round(abs(eigenvectors[1, 0]), 1)
            eigenvec2 = round(abs(eigenvectors[1, 1]), 1)
            p1 = (cntr[0] + 0.01 * eigenvec1 * eigenvalues[0, 0],
                  cntr[1] - 0.01 * eigenvec2 * eigenvalues[0, 0])
            p2 = (cntr[0] + 0.01 * -eigenvec2 * eigenvalues[1, 0],
                  cntr[1] - 0.01 * eigenvec1 * eigenvalues[1, 0])
        elif dif > abs(eigenvectors[0, 0]) / 2:
            eigenvec1 = round(abs(eigenvectors[0, 1]), 1)
            eigenvec2 = round(abs(eigenvectors[0, 0]), 1)
            p1 = (cntr[0] + 0.01 * eigenvec1 * eigenvalues[0, 0],
                  cntr[1] - 0.01 * eigenvec2 * eigenvalues[0, 0])
            p2 = (cntr[0] + 0.01 * -eigenvec2 * eigenvalues[1, 0],
                  cntr[1] - 0.01 * eigenvec1 * eigenvalues[1, 0])

    if v1[0] and v1[1] > 0:
        eigenvect1 = round(v1[0], 1)
        eigenvect2 = round(v1[1], 1)
    elif v1[0] or v1[1] < 0:
        eigenvect1 = round(v1[1], 1)
        eigenvect2 = round(v1[0], 1)
    p10 = (cntr[0] + 0.01 * eigenvect1 * eigenvalues[0, 0], cntr[1] - 0.01 * eigenvect2 * eigenvalues[0,
0])
    p20 = (cntr[0] + 0.01 * -eigenvect2 * eigenvalues[1, 0], cntr[1] - 0.01 * eigenvect1 * eigenvalues[1,
0])

    return p1, p2, p10, p20


def dibujo_ejes(p1, p2, p10, p20):
    """
    Funcion que dibuja el centro del controno y los ejes de la plancha de la foto como los de
AutoCAD
    :param p1:
    :param p2:
    :param p10:
    :param p20:
    :return:
    """
    global modelo
    global cgx
    global cgy
    global cX
    global cY
    global foto
    # Centre
    cntr = (cX, cY)
    # Draw centre of the contour
    cv2.circle(foto, cntr, 0, (0, 0, 255), 2)
    # Draw the axis of the contour
    draw_axis(cntr, p1, (0, 0, 255), 1)
    draw_axis(cntr, p2, (0, 0, 255), 1)
    draw_axis(cntr, p10, (255, 0, 0), 1)
    draw_axis(cntr, p20, (255, 0, 0), 1)


def calculo_angulo(p1, p2, p10, p20):
    """
    Proceso que calculo el angulo que forman los ejes originales de la plancha y la del contorno
    :param p1:
    :param p10:
```

```
    :return:
    """
    global cX
    global cY
    global alpha
    global angulo
    p1 = p1
    p10 = p10
    # Calculo de las pendientes
    if p1[0] == cX:
        m1 = 0
        m2 = (p10[1] - cY) / (p10[0] - cX)
    else:
        m1 = (p1[1] - cY) / (p1[0] - cX)
        m2 = (p10[1] - cY) / (p10[0] - cX)
    alpha = atan2((m2 - m1), (1 + m1 * m2))
    alpha = math.degrees(alpha)
    if alpha > 45:
        alpha = 90 - alpha
    elif alpha < -45:
        alpha = 90 + alpha
    else:
        alpha = alpha
    angulo = alpha
    if alpha > 0:
        print("Plancha rotada un angulo de: " + str(int(alpha)) + "º en sentido antihorario respecto
al CAD")
    else:
        print("Plancha rotada un angulo de: " + str(int(-alpha)) + "º en sentido horario respecto al
CAD")

    return angulo


def conver_sist_camara():
    """
    Proceso que pasa del sistema de coordenadas de la camara al 0,0 abajo a la izquierda de la
cama (igual que AutoCAd y el robot)
    :return:
    """
    global foto
    global cX
    global cY
    height, width = foto.shape[:2]
    # print(height, width)
    cX = cX
    cY = height - cY
    return cX, cY


def datos_plantilla(orden):
    """
    Proceso para obtener los datos de una plantilla dependiendo del orden que nos mande el robot
    :param orden:
    :return:
    """
    global tipo
    global datos_plantillas
    num_plantilla = orden
    tipo = datos_plantillas[num_plantilla][0]
```

```
    cordX = datos_plantillas[num_plantilla][1]
    cordY = datos_plantillas[num_plantilla][2]
    rotZ = float(datos_plantillas[num_plantilla][3])
    return tipo, cordX, cordY, rotZ


def conver_CAD_a_camara(cordX, cordY, rotZ):
    """
    Proceso para convertir los datos de cada plantilla a datos de pixeles (nuevo sistema de
coordenadas camara)
    :param cordX:
    :param cordY:
    :param rotZ:
    :return:
    """
    global cX
    global cY
    global cgx
    global cgy
    dif_x = cX/cgx
    dif_y = cY/cgy
    poseX = float(cordX) * dif_x
    poseY = float(cordY) * dif_y
    poserotZ = rotZ
    return poseX, poseY, poserotZ


def girar_datos_plantilla(dato):
    """
    Proceso para hacer los cambios en los datos de las plantillas dependiendo del giro
    :param dato:
    :return:
    """
    global angulo
    global cgx
    global cgy
    giro = angulo
    new_giro = dato[2]
    deltax = dato[0] - cgx
    deltay = dato[1] - cgy
    giro_rad = giro * (pi / 180)
    modulo = round(sqrt(deltax * deltax + deltay * deltay), 2)
    if deltax > 0:
        if deltay > 0:
            alpha = math.atan2(deltay, deltax)
            x = modulo * cos(alpha)
            y = modulo * sin(alpha)
            new_alpha = (alpha + giro_rad)
            new_x = modulo * cos(new_alpha)
            new_y = modulo * sin(new_alpha)
        elif deltay < 0:
            alpha = math.atan2(deltay, deltax)
            x = modulo * cos(alpha)
            y = modulo * sin(alpha)
            new_alpha = (alpha + giro_rad)
            new_x = modulo * cos(new_alpha)
            new_y = modulo * sin(new_alpha)
    elif deltax < 0:
        if deltay > 0:
            alpha = math.atan2(deltay, deltax) - (pi / 2)
```

```
            x = - modulo * sin(alpha)
            y = modulo * cos(alpha)
            new_alpha = (alpha + giro_rad)
            new_x = - modulo * sin(new_alpha)
            new_y = modulo * cos(new_alpha)
        elif deltay < 0:
            alpha = - math.atan2(deltay, deltax) - (pi / 2)
            x = - modulo * sin(alpha)
            y = - modulo * cos(alpha)
            new_alpha = (alpha - giro_rad)
            new_x = - modulo * sin(new_alpha)
            new_y = - modulo * cos(new_alpha)
    cordX = int(cgx + new_x)
    cordY = int(cgy + new_y)
    rotZ = new_giro + giro
    if rotZ > 180:
        rotZ = rotZ - 360
    elif rotZ <= 180:
        rotZ = rotZ

    # Ajutes del giro de la herramienta dependiendo del modelo
    modelo = dato[3]
    if modelo == 'corte_empeine_derecha':
        rotZ = rotZ - 90
    elif modelo == 'corte_empeine_izquierda':
        rotZ = rotZ + 90
    elif modelo == 'corte_lateral_derecha':
        rotZ = rotZ + 15
    elif modelo == 'corte_lateral_izquierda':
        rotZ = rotZ - 15
    else:
        rotZ = rotZ
    return cordX, cordY, rotZ


def escalado_pixel_metro(coordenadaX, coordenadaY):
    """
    Funcion para conversion de las coordenadas de pixel a milimetros
    :param coordenadaX:
    :param coordenadaY:
    :return:
    """
    global foto
    global tipo
    conversion_x = 2.0114  # pixeles/milímetros
    conversion_y = 2.0114  # pixeles/milímetros
    coordenadaX = coordenadaX * conversion_x
    coordenadaY = coordenadaY * conversion_y
    return coordenadaX, coordenadaY


# SIMULACION ALGORITMO

# Modelo que es cortado en la CNC
modelo = 'modelo_plancha'
# Extraccion datos plancha y plantillas desde AutoCAD
extrac_datos_autocad()
# Lectura de datos plantillas
datos_plantillas = lectura_datos_plantillas()
# Lectura de datos plancha
```

```
lectura_datos_plancha()
# Lectura de la imagen tomada en RobotStudio
ruta_foto = 'C:/Users/franm/Desktop/datos_TFM/foto.png'
foto = cv2.imread(ruta_foto)
# Reocte imagen para centrarse en la pieza
foto = foto[23:605, 239:987]
cv2.imwrite('foto_recortada.png', foto)
# Encuentra el mayor contorno y lo dibuja
contours_foto = encuentra_contornos()
indice, _ = biggest_area(contours_foto)
indice = indice-1
c = contours_foto[indice]
cv2.drawContours(foto, contours_foto, indice, (0, 0, 255), 2)
# Calculo centro de gravedad a partir de los momentos
calculo_centro_gravedad(c)
# Ejes del contorno tomado por imagen
get_orientation(c)
# Dibujo todos los ejes
p1, p2, p10, p20 = calculo_axis()
dibujo_ejes(p1, p2, p10, p20)
# Centre
cntr = (cX, cY)
# Draw centre of the contour
cv2.circle(foto, cntr, 0, (0, 0, 255), 4)
# Calculo del ángulo de rotación entre ejes
alpha = calculo_angulo(p1, p2, p10, p20)
# Se muestra la imagen de la pieza con el contorno y el giro
cv2.imshow('RESULTADO', foto)
cv2.imwrite('foto_resultado.png', foto)
cv2.waitKey(1000)

# Convertimos los datos del centro de la plancha al nuevo sistema de coordenadas (abajo
izquierda)
cX, cY = conver_sist_camara()
# Recalculamos la posicion y orientación de las piezas y se envian por el socket
for plantilla in range(0, 33):
    # Recibe una señal de robotstudio para mandar siguiente dato
    signal = 0
    if plantilla < 33:
        while signal == 0:
            signal_rs = obj.recv(1024)
            if signal_rs == b'next_dato':
                signal = 1
            else:
                signal = 0
            signal_rs = b'0'

    if plantilla == 32:
        data = "000000000000"
    else:
        orden = plantilla
        modelo, cordX, cordY, rotZ = datos_plantilla(orden)
        cordX, cordY, rotZ = conver_CAD_a_camara(cordX, cordY, rotZ)
        cordX, cordY = escalado_pixel_metro(cordX, cordY)
        dato = [cordX, cordY, rotZ, modelo]
        cordX, cordY, rotZ = girar_datos_plantilla(dato)
        datos_plantillas[orden][0] = modelo
        datos_plantillas[orden][1] = cordY
        datos_plantillas[orden][2] = cordX
        datos_plantillas[orden][3] = rotZ
```

```
        # Codificamos los datos para enviarlos
        # Rotacion en Z positivo (0) o negativo (1)
        if rotZ >= 0:
            rotz = int(rotZ + 101)
            rotz = '0' + str(rotz)
        elif rotZ < 0:
            rotz = int(-rotZ + 101)
            rotz = '1' + str(rotz)
        # Dependiendo del modelo añadimos un numero al pincipo, y despues los datos
        if modelo == 'corte_empeine_derecha':
            data = str('1') + str(cordX+1000) + str(cordY) + rotz
        elif modelo == 'corte_empeine_izquierda':
            data = str('2') + str(cordX+1000) + str(cordY) + rotz
        elif modelo == 'corte_lateral_derecha':
            data = str('3') + str(cordX+1000) + str(cordY) + rotz
        elif modelo == 'corte_lateral_izquierda':
            data = str('4') + str(cordX+1000) + str(cordY) + rotz
        elif modelo == 'corte_planta_derecha':
            data = str('5') + str(cordX+1000) + str(cordY) + rotz
        elif modelo == 'corte_planta_izquierda':
            data = str('6') + str(cordX+1000) + str(cordY) + rotz
        elif modelo == 'corte_talonera_derecha':
            data = str('7') + str(cordX+1000) + str(cordY) + rotz
        elif modelo == 'corte_talonera_izquierda':
            data = str('8') + str(cordX+1000) + str(cordY) + rotz

    print("Data:", data)
    # Enviamos el dato en formato byte mediante el socket a RobotStudio
    signal = data.encode(encoding='utf-8')
    obj.send(signal)
    print("Dato enviado por socket")

print('Fin de la simulacion')
```

8.3.3. Código de RAPID conexión Python-AutoCAD-Vision-RobotStudio

```
MODULE Module1

    VAR socketdev Server_Socket;
    VAR socketdev Client_Socket;
    VAR string client_ip;
    VAR string data;
    VAR string signal;
    VAR num num_piezas;
    VAR string modelo{50};
    VAR num coorX{50};
    VAR num coorY{50};
    VAR num rota{50};
    VAR num altura:=-9.85;

    VAR robtarget plantilla{50};
    VAR num num_plantilla;
    VAR num num_plantillas{50};

    VAR num socket_signal:=0;
    VAR string datos;
    VAR string valor_modelo;
    VAR string valor_cordx;
    VAR string valor_cordy;
    VAR string valor_rotz;
    VAR string valor_rotz_signo;
    VAR string valor_rotz_valor;
    VAR pose pose2;
    VAR num cordX;
    VAR num CordY;
    VAR num rotZ;
    VAR robtarget plantilla2;
    VAR num cambioStrX;
    VAR num cambioStrY;
    VAR num cambioStrZ;
    VAR bool okX;
    VAR bool okY;
    VAR bool okZ;


    ! Posiciones robot
    CONST robtarget Inicio:=[[763.790948903,723.049,-
195.3],[1,0,0,0],[0,2,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Inicio_2:=[[144.633847882,450.885334565,-195.3],[1,0,0,0],[-
1,2,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    ! Posiciones de dejada de las piezas
```

```
    CONST robtarget corte_empeine_der:=[[291.617,1506.997,-
9.8],[0.707106781,0,0,0.707106781],[0,2,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,
9E+09]];
    CONST robtarget corte_lateral_der:=[[379.454,1354.834,-
9.8],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget corte_planta_der:=[[562.829,1484.064,-
9.8],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget corte_talonera_der:=[[610.4,1311.732,-
9.8],[1,0,0,0],[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget corte_empeine_izq:=[[281.617,-49.266,-
9.8],[0.707106781,0,0,-0.707106781],[-
1,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget corte_lateral_izq:=[[379.454,101.558,-9.8],[1,0,0,0],[-
1,2,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget corte_planta_izq:=[[562.829,-8.67,-9.8],[1,0,0,0],[-
1,2,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget corte_talonera_izq:=[[610.4,134.32,-9.8],[1,0,0,0],[-
1,2,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];



    PROC main()

        WaitTime 2;
        num_piezas:=32;
        socket_signal:=0;
        data:="";
        Movimientos_cinta;
        WaitTime 4;
        !Conexion RobotStudio con Python
        SocketCreate Server_Socket;
        SocketBind Server_Socket,"127.0.0.1",1024;
        SocketListen Server_Socket;
        SocketAccept Server_Socket,Client_Socket,\Time:=WAIT_MAX;
        SocketSend Client_Socket\Str:="Conexion con ABB correcta.";
        WHILE socket_signal=0 DO

            MoveL
Inicio,v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
            WaitTime 5;
            FOR i FROM 1 TO num_piezas DO
                Movimientos_socket;
            ENDFOR

            MoveL
Inicio,v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;

            Break;

        ENDWHILE
```

```
    WaitTime 2;

ENDPROC

PROC Movimientos_socket()
    data:="";
    SocketSend Client_Socket\Str:="next_dato";
    SocketReceive Client_Socket\str:=data;
    datos:=StrPart(data,1,12);
    IF data="000000000000" THEN
        socket_signal:=1;
    ELSE
        socket_signal:=0;
        valor_modelo:=StrPart(datos,1,1);
        valor_cordx:=StrPart(datos,2,4);
        valor_cordy:=StrPart(datos,6,3);
        valor_rotz:=StrPart(datos,9,4);

        ! Leemos datos de coordenadas X e Y y hacemos ajuste
        okX:=StrToVal(valor_cordx,cordY);
        cordY:=cordY-1000;
        okY:=StrToVal(valor_cordy,cordX);

        ! Dividimos el string de a rotacion para saber si es positivo o negativo
        valor_rotz_signo:=StrPart(valor_rotz,1,1);
        valor_rotz_valor:=StrPart(valor_rotz,2,3);
        okZ:=StrToVal(valor_rotz_valor,rotZ);
        rotZ:=rotZ-101;
        IF valor_rotz_signo="1" THEN
            rotZ:=rotZ*(-1);
        ELSE
            rotZ:=rotZ;
        ENDIF

        ! Generamos la posicion a la que debe de moverse el robot
        pose2.trans:=[cordX+100,cordY,altura];
        pose2.rot:=OrientZYX(-rotZ,0,0);

plantilla2:=[pose2.trans,pose2.rot,[0,2,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E
+09]];

        MoveL Offs(plantilla2,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
        MoveL
plantilla2,v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
        WaitRob\inpos;
        SetDO Coger_pieza,1;
        WaitTime 0.2;
```

```
        MoveL Offs(plantilla2,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;

        ! Comprobamos que modelo de pieza y se efine posicion de dejada
        IF valor_modelo="1" THEN
           ! "corte_empeine_derecha"
           num_plantillas{1}:=num_plantillas{1}+1;
           dejar_corte_empeine_derecha;
        ELSEIF valor_modelo="2" THEN
           ! "corte_empeine_izquierda"
           num_plantillas{2}:=num_plantillas{2}+1;
           dejar_corte_empeine_izquierda;
        ELSEIF valor_modelo="3" THEN
           ! "corte_lateral_derecha"
           num_plantillas{3}:=num_plantillas{3}+1;
           dejar_corte_lateral_derecha;
        ELSEIF valor_modelo="4" THEN
           ! "corte_lateral_izquierda"
           num_plantillas{4}:=num_plantillas{4}+1;
           dejar_corte_lateral_izquierda;
        ELSEIF valor_modelo="5" THEN
           ! "corte_planta_derecha"
           num_plantillas{5}:=num_plantillas{5}+1;
           dejar_corte_planta_derecha;
        ELSEIF valor_modelo="6" THEN
           !"corte_planta_izquierda"
           num_plantillas{6}:=num_plantillas{6}+1;
           dejar_corte_planta_izquierda;
        ELSEIF valor_modelo="7" THEN
           ! "corte_talonera_derecha"
           num_plantillas{7}:=num_plantillas{7}+1;
           dejar_corte_talonera_derecha;
        ELSEIF valor_modelo="8" THEN
           ! "corte_talonera_izquierda"
           num_plantillas{8}:=num_plantillas{8}+1;
           dejar_corte_talonera_izquierda;
        ENDIF
     ENDIF

     data:="";

  ENDPROC

  PROC Movimientos_cinta()
     SetDO Marcha,1;
     SetDO Paro_cinta,0;
     MoveL
Inicio_2,v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
     WaitDI Sensor_Foto,1;
     SetDO Marcha,0;
```

```
        SetDO Paro_cinta,1;
        WaitTime 0.3;
        SetDO Paro_cinta,0;
        WaitTime 0.3;
        SetDO Marcha,1;
        SetDO Marcha,0;
        SetDO Paro_cinta,0;
    ENDPROC


    PROC dejar_corte_empeine_derecha()

        MoveL Offs(corte_empeine_der,0,0,-
100),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
        MoveL Offs(corte_empeine_der,0,0,-
num_plantillas{1}*5.1),v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Convey
or;
        SetDO Coger_pieza,0;
        WaitTime 0.2;
        MoveL Offs(corte_empeine_der,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;

    ENDPROC

    PROC dejar_corte_empeine_izquierda()

        MoveL Offs(corte_empeine_izq,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
        MoveL Offs(corte_empeine_izq,0,0,-
num_plantillas{2}*5.1),v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Convey
or;
        SetDO Coger_pieza,0;
        WaitTime 0.2;
        MoveL Offs(corte_empeine_izq,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;

    ENDPROC

    PROC dejar_corte_lateral_derecha()

        MoveL Offs(corte_lateral_der,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
        MoveL Offs(corte_lateral_der,0,0,-
num_plantillas{3}*5.1),v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Convey
or;
        SetDO Coger_pieza,0;
        WaitTime 0.2;
        MoveL Offs(corte_lateral_der,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
```

```
    ENDPROC

    PROC dejar_corte_lateral_izquierda()

        MoveL Offs(corte_lateral_izq,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
        MoveL Offs(corte_lateral_izq,0,0,-
num_plantillas{4}*5.1),v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Convey
or;
        SetDO Coger_pieza,0;
        WaitTime 0.2;
        MoveL Offs(corte_lateral_izq,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;

    ENDPROC

    PROC dejar_corte_planta_derecha()

        MoveL Offs(corte_planta_der,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
        MoveL Offs(corte_planta_der,0,0,-
num_plantillas{5}*5.1),v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Convey
or;
        SetDO Coger_pieza,0;
        WaitTime 0.2;
        MoveL Offs(corte_planta_der,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;

    ENDPROC

    PROC dejar_corte_planta_izquierda()

        MoveL Offs(corte_planta_izq,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
        MoveL Offs(corte_planta_izq,0,0,-
num_plantillas{6}*5.1),v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Convey
or;
        SetDO Coger_pieza,0;
        WaitTime 0.2;
        MoveL Offs(corte_planta_izq,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;

    ENDPROC

    PROC dejar_corte_talonera_derecha()

        MoveL Offs(corte_talonera_der,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
```

```
        MoveL Offs(corte_talonera_der,0,0,-
num_plantillas{7}*5.1),v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Convey
or;
        SetDO Coger_pieza,0;
        WaitTime 0.2;
        MoveL Offs(corte_talonera_der,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;

    ENDPROC

    PROC dejar_corte_talonera_izquierda()

        MoveL Offs(corte_talonera_izq,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;
        MoveL Offs(corte_talonera_izq,0,0,-
num_plantillas{8}*5.1),v1000,fine,Ventosas_tool_TCP\WObj:=WorkObject_Convey
or;
        SetDO Coger_pieza,0;
        WaitTime 0.2;
        MoveL Offs(corte_talonera_izq,0,0,-
50),v1000,z10,Ventosas_tool_TCP\WObj:=WorkObject_Conveyor;

    ENDPROC

ENDMODULE
```

8.4. Hojas de características de los Robots

8.4.1. Hoja de características del Robot IRB120 [12]

ROBOTICS

IRB 120

ABB's 6 axis robot – for flexible and compact production

The IRB 120 robot is the latest addition to ABB's new fourth-generation of robotic technology. It is ideal for material handling and assembly applications and provides an agile, compact and lightweight solution with superior control and path accuracy.

Compact and lightweight
IRB 120's compact design enables it to be mounted virtually anywhere at any angle without any restriction - for example inside a cell, on top of a machine or close to other robots.

IRB 120 is also the most portable and easy to integrate on the market with its 25 kg weight. The smooth surfaces are easy to clean and the cables for air and customer signals are internally routed, all the way from the foot to the wrist, ensuring that integration is effortless.

Multipurpose
IRB 120 is ideal for a wide range of industries including the electronic, food and beverage, machinery, solar, pharmaceutical, medical and research sectors.

The Food Grade Lubrication (NSF H1) option includes Clean Room ISO Class 5, which ensures uncompromising safety and hygiene for food and beverage applications.

Optimized working range
IRB 120 has a horizontal reach of 580 mm, the best in class stroke, the ability to reach 112 mm below its base and a very compact turning radius.

Fast, accurate and agile
Designed with a light, aluminum structure, the motors ensure the robot is enabled with a fast acceleration, and can deliver accuracy and agility in any application.

IRC5 Compact controller – optimized for small robots
ABB's new IRC5 Compact controller presents the capabilities of the IRC5 controller in a compact format. It brings accuracy and motion control to applications which have been exclusive to large installations and enables easy commissioning through one phase power input, external connectors for all signals and a built-in expandable 16 in, 16 out, I/O system.

RobotStudio for offline programming enables manufacturers to simulate a production cell to find the optimal position for the robot, and provide offline programming to prevent costly downtime and delays to production.

Reduced footprint
The combination of the new lightweight architecture of the IRB 120 with the new IRC5 Compact controller introduces a significantly reduced footprint.

Specification

Robot version	Reach (m)	Handling capacity (kg)	Armload (kg)
IRB 120-3/0.6	0.58	3*	0.30
Number of axes	6		
Protection	IP30		
Mounting	Any angle		
Controller	IRC5 Compact/IRC5 Single Cabinet		
Integrated signal supply	10 signals on wrist		
Integrated air supply	4 air on wrist (5 bar)		

* 4 with vertical wrist

Performance (according to ISO 9283)

	IRB 120
1 kg picking cycle	
25 x 300 x 25 mm	0.58 s
25 x 300 x 25 with 180° axis 6 reorientation	0.92 s
Acceleration time 0-1 m/s	0.07 s
Position repeatability	0.01 mm

Technical information

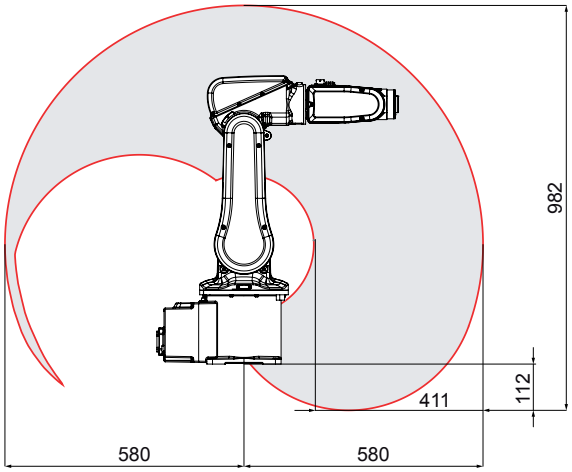
Electrical Connections	
Supply voltage	200-600 V, 50/60 Hz
Rated power transformer rating	3.0 kVA
Power consumption	0.24 kW
Physical	
Robot base	180 x 180 mm
Robot height	700 mm
Robot weight	25 kg
Environment	
Ambient temperature for robot manipulator:	
During operation	+5°C (41°F) to +45°C (113°F)
During transportation and storage	-25°C (-13°F) to +55°C (131°F)
During short periods (max. 24 h)	up to +70°C (158°F)
Relative humidity	Max. 95%
Noise level	Max. 70 dB (A)
Safety	Safety and emergency stops 2-channel safety circuits supervision, 3-position enabling device
Emission	EMC/EMI-shielded
Options	Clean Room ISO class 5 (certified by IPA)**

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

Movement

Axis movement	Working range	Velocity IRB 120
Axis 1 rotation	+165° to -165°	250°/s
Axis 2 arm	+110° to -110°	250°/s
Axis 3 arm	+70° to -110°	250°/s
Axis 4 wrist	+160° to -160°	320°/s
Axis 5 bend	+120° to -120°	320°/s
Axis 6 turn	Default: +400° to -400° Max. rev: +242 to -242	420°/s

Working range



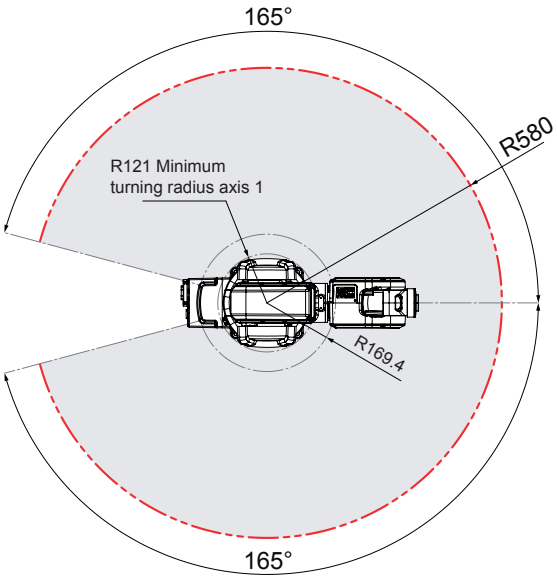


abb.com/robotics

ROBO149EN_D Rev. J November 2019

8.4.2. Hoja de características del Robot IRB1600 [13]

ROBOTICS

IRB 1600

The highest performance 10 kg robot

Performance is often a trade off, optimizing for speed or accuracy. With ABB's IRB 1600, you don't have to choose. The robot's cycle times are shorter, sometimes half that of other robots, allowing you to increase throughput. Meanwhile, you will enjoy the work piece quality that only an ABB robot can offer. Extra everything.

Double your throughput
The IRB 1600 has up to 50 percent shorter cycle times than competing robots in material handling, machine tending and process applications. It speeds up and slows down faster than other robots, saving time while moving between tasks. This is possible due to ABB's patented second generation QuickMove motion control, combined with the robot's strong motors and low friction losses in the spur gears.

No more cutting corners
At high speed, most robots will cut corners. With the IRB 1600, the path will be the same regardless of speed, thanks to the robot's unique combination of brains and brawn. Intelligent second generation TrueMove motion control means that "what-you-program-is-what-you-get". Add muscle – a heavy and stiff design – low vibrations and low friction – and you have a robot that will deliver consistently high work piece quality, high yield and few rejects.

Outstanding reliability
The IRB 1600 offers outstanding reliability, even in the toughest environments and the most demanding 24/7 duty cycles. The entire manipulator is IP 54 classed and sensitive parts are IP 67 classed as standard.

The optional protection Foundry Plus offers IP 67, special paint, rust protection and is tailor made for tough foundry environments. The rigid and heavy design combined with spur gears, make the robot extremely robust. Smart collision detection software further adds to the robot's outstanding reliability.

Easy to integrate
Mounting is fully flexible: on a shelf, on the wall, tilted or inverted. By choosing the compact short-arm version with the 1.2 m reach, you can even fit the IRB 1600 inside a machine, while ensuring sufficient payload as the maximum total load is as high as 36 kg. Sustainable and healthy Low friction spur gears, and no unnecessary moves due to QuickMove and TrueMove, reduces power consumption down to 0.58 kW at max speed, and even less at low speeds. The airborne noise level of just <70 dB (A) secures a healthy sound environment.

Main applications
- Assembly
- Arc Welding
- Material Handling
- Machine Tending
- Material Removal
- Cleaning/Spraying
- Dispensing
- Packing

Specification

Robot version	Reach (m)	Payload (kg)	Armload (kg)
IRB 1600-6/1.2	1.2	6	30.5
IRB 1600-6/1.45	1.45	6	30.5
IRB 1600-10/1.2	1.2	10	20.5
IRB 1600-10/1.45	1.45	10	20.5
Number of axes	6+3 external (up to 36 with MultiMove)		
Protection	Standard IP54 Option FoundryPlus 2 (IP67)		
Mounting	Floor, wall, shelf, tilted, inverted		
Controller	IRC5 Single Cabinet/IRC5 Compact		

Performance (according to ISO 9283)

	Position repeatability	Path repeatability
IRB 1600-6/1.2	0.02 mm	0.13 mm
IRB 1600-6/1.45	0.02 mm	0.19 mm
IRB 1600-10/1.2	0.02 mm	0.06 mm
IRB 1600-10/1.45	0.05 mm	0.13 mm

Technical information

Electrical Connections	
Supply voltage	200-600 V, 50-60 Hz
Energy consumption	0.58 kW
Physical	
Robot base	484 x 648 mm
Robot height	
IRB 1600-6/1.2	1069 mm
IRB 1600-10/1.2	1069 mm
IRB 1600-6/1.45	1294 mm
IRB 1600-10/1.45	1294 mm
Robot weight	250 kg
Environment	
Ambient temperature for mechanical unit	
During operation	+ 5°C (41°F) to + 45°C (113°F)
During transportation and storage	- 25°C (- 13°F) to + 55°C (131°F)
During short periods (max. 24 h)	up to + 70°C (158°F)
Relative humidity	Max. 95%
Safety	Double circiuts with supervisions, emergency stops and safety functions, 3-position enable device
Emission	EMC/EMI shielded

Data and dimensions may be changed without notice.

Movement

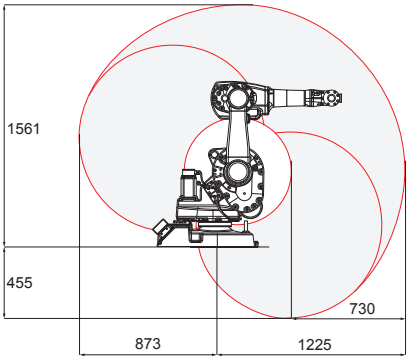
Axis movement	Working range IRB 1600-6/1.2, IRB 1600-10/1.2	Working range IRB 1600-6/1.45, IRB 1600-10/1.45
Axis 1 rotation	+180° to -180°	+180° to -180°
Axis 2 arm	+110° to -63° +136° to -63°[1]	+120° to -90° +150° to -90°[2]
Axis 3 arm	+55° to -235°	+65° to -245°
Axis 4 rotation	Default: +200° to -200° Max. rev: +190° to -190°	Default: +200° to -200° Max. rev: +190° to -190°
Axis 5 bend	+115° to -115°	+115° to -115°
Axis 6 turn	Default: +400° to -400° Max. rev: +288 to -288	Default: +400° to -400° Max. rev: +288 to -288

[1] With axis 1 limited to ±100° [2] With axis 1 limited to ±95°

Axis max. speed

	IRB 1600-6/1.2, IRB 1600-6/1.45	IRB 1600-10/1.2, IRB 1600-10/1.45
Axis 1 rotation	150°/s	180°/s
Axis 2 arm	160°/s	180°/s
Axis 3 arm	170°/s	185°/s
Axis 4 rotation	320°/s	385°/s
Axis 5 bend	400°/s	400°/s
Axis 6 turn	460°/s	460°/s

Working range, IRB 1600-6/1.2, IRB 1600-10/1.2



Working range, IRB 1600-6/1.45, IRB 1600-10/1.45

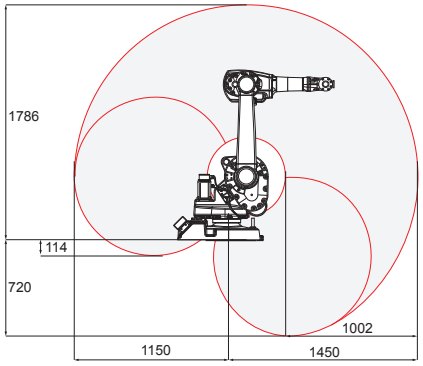


abb.com/robotics

PR:10282EN Rev.E January 2019