Universidad Miguel Hernández de Elche

MASTER UNIVERSITARIO EN ROBÓTICA

“Programación y desarrollo de una estación robótica en Robot Studio”

Trabajo de Fin de Máster

Septiembre 2020

Autor: Juan José Pérez Hernández
Tutor/es: Carlos Pérez Vidal
Jorge Borrell Mendez

Programación y desarrollo de una estación robótica en RobotStudio

Trabajo de Fin de Máster

Autor
Juan José Pérez Hernández

Tutor/es
Carlos Pérez Vidal
Departamento de Ingeniería de Sistemas y Automática
Jorge Borrell Mendez
Departamento de Ingeniería de Sistemas y Automática

Grado en Ingeniería Electrónica y Automática Industrial

EPSE

UNIVERSITAS
Miguel Hernández

ELCHE, Septiembre 2020

Preámbulo

Después de realizar el Trabajo de Fin de Grado, en el que me familiaricé con Robot Studio y el robot apodado YuMi de la marca ABB, me pareció interesante realizar el Máster en Robótica de la UMH, esto me ha permitido seguir colaborando en el proyecto mediante prácticas internas. De este modo he continuado formándome en el campo de la Robótica mediante el Máster, y al mismo tiempo he adquirido algo más de experiencia en la programación de robots de la marca ABB.

En el Trabajo de Fin de Grado realicé algunas aplicaciones básicas para en las que aprendí como funciona la sincronización entre los los dos brazos que tiene el robot, como funciona Robot Studio y también ayudé en la programación de una tarea de Pick&Place con el robot real. En esta ocasión he llevado los conocimientos que he aprendido sobre RobotStudio y el robot un poco más lejos, para programar una aplicación virtual más avanzada, así como una estación de Pick&Place más avanzada también. En el TFG no me dió tiempo a aprender a utilizar la parte de Visión Artificial de la estación, en esta sí, y en el trabajo expondré como lo funciona y enseñaré ejemplos.

Agradecimientos

Gracias a mis padres y a toda mi familia, así como a mis amigos y a mi novia Laura, sin ellos no sería quien soy.

También agradecer a todos los profesores que he tenido desde que empecé el colegio, de todos aprendí algo. Por supuesto, gracias a Carlos y Jorge, mis tutores en esta etapa que parece que acaba, he aprendido y disfrutado mucho.

"El valor de una idea radica en el uso de la misma."

Tomas A. Edison.

Índice general

Índice de figuras

1. Introducción, motivación y objetivos

En este capítulo se presenta una introducción del trabajo realizado, la motivación inicial y las razones que impulsaron el proyecto, así como lo objetivos que se plantearon en un primer momento.

1.1. Introducción

Hoy en día, la mayoría de procesos industriales están o al menos son susceptibles de ser automatizados. Mediante la automatización de los procesos industriales, se mejoran rendimientos en cuanto a economía de tiempo y costes. La automatización no solo está presente en los procesos industriales, si no también en campos como la medicina, la aeronáutica, la automoción o la domótica.

El robot que se ha utilizado para las aplicaciones del proyecto es colaborativo. Una de las más grandes y novedosas revoluciones dentro de la automatización industrial, es la aparición de la robótica colaborativa. La robótica colaborativa es el campo que abarca aquellos robots que son capaces de trabajar en un entorno interaccionando con los operarios, facilitando y agilizando el trabajo de estos, sin ningún sistema de seguridad o protección de personas como las requeridas en los robots industriales. Los robots colaborativos o cobots suelen ser ligeros, sencillos de manejar y fácilmente programables. Cabe destacar que los cobots no sustituyen a los operarios, más bien colaboran con ellos para agilizar el proceso que está realizando el operario.

Por un lado, los cobots (robots colaborativos) son fácilmente adaptables a diversos ámbitos dentro de una industria, con requerimientos mínimos de seguridad para interaccionar con humanos. Por otra parte, ofrecen un rápido ROI (retorno de inversión), lo que disminuye los costes de la industria al invertir en dichos robots. Además, la programación de estos cobots, al no requerir de personal con un nivel de especialización en automatismos elevada, hace que también se reduzcan costes de mano de obra.

Por tanto, el futuro de la industria a largo plazo, es la implantación masiva de cobots para realizar cualquier proceso industrial.

Finalmente, mencionar que en este trabajo se ha profundizado en la programación del robot bimanual colaborativo. Se trata del modelo IRB 14000 de la marca ABB, apodado YuMi. La finalidad de dicha programación será diseñar un par de programas en RobotStudio aplicables a procesos industriales, más adelante se explicarán más en

profundidad.

Figura 1.1: IRB 14000 (YuMi).

1.2. Motivación

La principal motivación del proyecto es aprender y ganar experiencia en el mundo de la robótica, perfeccionar los conocimientos adquiridos en la programación de robots de la marca ABB. Esto puede ser bastante interesante de cara a trabajar en el futuro en un lugar en el que se realice una labor relacionada.

También fue una motivación el poder aplicar los conocimientos adquiridos durante el Grado y el Máster en Robótica, y colaborar con el departamento de automatización de la universidad.

Finalmente, como ya se ha mencionado, el mundo de la robótica colaborativa está en pleno auge, y como aficionado y como futuro profesional es conveniente empezar a conocer y a trabajar con ellos, y en concreto con un modelo tan como es el Yumi.

1.3. Objetivos

Una vez introducido el proyecto, es fundamental fijar el objetivo al realizar el proyecto. Desde el primer momento, el objetivo fue aprender.

Por otro lado, otro de los objetivos del proyecto es estudiar más profundamente el campo de la robótica colaborativa y las mejoras que esta aporta tras su implantación.

Ver cuanta comodidad aporta a los operarios, si garantiza totalmente la seguridad de estos, y la rentabilidad y los beneficios que aporta en la cadena de montaje.

La Universidad dispone de un acuerdo con la empresa de ABB y con una empresa distribuidora de ABB muy cercana a la Universidad, mediante este acuerdo, se ha dispuesto del robot para hacer pruebas reales sobre el mismo, y de licencia para el uso completo del software de simulación llamado RobotStudio. De este modo, una vez que el alumno aprendió a controlar el robot, el lenguaje de programación Rapid, y el entorno de programación (RobotStudio), pasó a llevar a cabo algunas tareas de investigación.

Se trata de un robot relativamente nuevo, que además dispone de funciones poco comunes entre los robots del mercado como disponer de dos brazos, o ser colaborativo, esto da al alumno mucho margen de investigación y estudio.

El trabajo constará de tres partes principalmente a parte de la introducción a la robótica colaborativa y a RobotStudio. La primera es una introducción a RAPID y a RobotStudio, así como a su parte de Visión Artificial, y las otras serán un par de aplicaciones que se explican a continuación.

- La primera es una aplicación en el entorno virtual de RobotStudio, que sirve para hacer un estudio del alcance del robot, de las posiciones en las que los brazos colisionan entre sí, y en caso negativo la distancia que queda entre ellos. Esto se hará mediante el recorrido de ambos brazos por una rejilla de posiciones, y elementos del programa que se explicarán más adelante.
- La segunda se trata de una aplicación real de Pick&Place con piezas flexibles y herramientas con ventosas especiales que se explicarán más adelante. Este sistema cuenta con una cámara que detecta la forma de las piezas e informa de su posición y orientación para que el robot las coja.

Figura 1.2: Logotipo de ABB.

1.4. Estructura del TFG

El presente Trabajo Fin de Máster está distribuido en cinco capítulos más la Bibliografía y los Anexos, se dará una breve explicación a continuación:

Capítulo 1: Introducción,motivación y objetivos. Se expone la motivación y el contexto del presente trabajo, los objetivos que persiguen, se resume y estructura la memoria para una mejor visión general de sus contenidos.

Capítulo 2: Introducción a la Robótica. Se introduce el mundo de la robótica, incluyendo un poco de historia y tipos, se habla de robótica colaborativa, ya que el robot utilizado es colaborativo, y por último, se ha analizado el mercado actual de robots colaborativos, incluyendo el YuMi.

Capítulo 3: Metodología. Se ha realiza una introducción al programa de software RobotStudio ABB, con aprendizaje y creación de una estación en un entorno de simulación con el YuMi. Introducción a Rapid, incluyendo la programación de varios robots trabajando de forma simultanea. También se explica como funcionan y para que sirven los componentes inteligentes de RobotStudio y la parte de Visión artificial.

Capítulo 4: Resultados. Se explicarán las dos aplicaciones realizadas, la primera en el entorno virtual de RobotStudio y la segunda con el robot real.

Capítulo 5: Conclusiones y futuros proyectos. Se exponen las conclusiones del proyecto, comparándolas con los objetivos marcados inicialmente. Se proponen mejoras o nuevas líneas de investigación para futuros trabajos.

Bibliografía. Se indican todas las referencias bibliográficas utilizadas en este trabajo.

Capítulo 6: Anexos. En este apartado se añade información complementaria, incluyendo las hojas de características del robot y programa completo de la última aplicación.

2. Introducción a la robótica

El entorno industrial actual es muy cambiante, por ello se hacen necesarios cambios sustanciales para fabricar productos de forma más versátil y económica posible. Esto se obtiene a través de los novedosos sistemas flexibles tecnológicos de producción que actualmente se están introduciendo en el entorno industrial.

La robótica es una de las principales soluciones en este sector, en este apartado se trata de resumir el estado actual de desarrollo de las aplicaciones robóticas empleadas en los entornos industriales.

2.1. Historia de la robótica

Se hablará de robot como una máquina electromecánica la cual ha sido programada para para moverse, manipular objetos y realizar trabajos a la vez que interacciona con su entorno.

El inicio de la robótica actual puede fijarse en la industria textil del siglo XVIII, cuando Joseph Jacquard inventa en 1801 una máquina textil programable mediante tarjetas perforadas. Con la llegada de la revolución industrial se intentó el desarrollo de máquinas que ayudasen o sustituyesen el hombre en las tareas las cuales tenía que desarrollar.

Los autómatas, o máquinas semejantes a personas, ya aparecían en los relojes de las iglesias medievales, y los relojeros del siglo XVIII eran famosos por sus ingeniosas criaturas mecánicas.

La primera vez que se utilizó la palabra robot fue en 1920 en una obra llamada "*Los Robots Universales de Rossum*", escrita por el dramaturgo checo Karel Capek. Su trama trataba sobre un hombre que fabricó un robot y luego este último mata al hombre. La palabra checa '*Robota*' significa servidumbre o trabajo forzado y cuando se tradujo al inglés se convirtió en el termino robot. Poco después Isaac Asimov comenzó a contribuir con diversas obras, en una de ellas *Runaround* acuñó un término el cual se utilizaría a lo largo de los años, este término era Robótica, el cual se refiere a la ciencia o arte relacionada con la inteligencia artificial y con la ingeniería mecánica. También a partir de este término surgen las denominadas "*Tres Leyes de la Robótica*":

1. Un robot no puede actuar contra un ser humano o, mediante la inacción, que un ser humano sufra daños.

2. Un robot debe de obedecer las órdenes dadas por los seres humanos, salvo que estén en conflicto con la primera ley.

3. Un robot debe proteger su propia existencia, a no ser que esté en conflicto con las dos primeras leyes.

Figura 2.1: Isaac Asimov.

Varios factores influyeron en el desarrollo de los primeros robots en la década de los cincuenta. El desarrollo en la tecnología, donde se incluyen los procesadores electrónicos, los actuadores de control retroalimentados y la transmisión de potencia a través de engranajes han contribuido a flexibilizar los autómatas para realizar las tareas programadas. La investigación en inteligencia artificial desarrolló diferentes maneras de simular el procesamiento de información humana con ordenadores. Las primeras patentes aparecieron en 1946 con los primitivos robots para traslado de maquinaria de Devol. En 1954, Devol diseña el primer robot programable. Durante los siguientes años se desarrollan una gran variedad de robots en todos los sectores de la sociedad, principalmente en el ámbito industrial para agilizar las tareas industriales.

Actualmente, el concepto de robótica ha evolucionado hacia los sistemas móviles autónomos, que son aquellos capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión.

2.1.1. Generaciones de robots

Se puede hacer una clasificación de los robots en función de la evolución que han tenido a lo largo del tiempo. Esta clasificación se puede hacer distinguiendo tres generaciones que se detallarán más detenidamente a continuación.

En primer lugar, la primera generación de robots se desarrolló alrededor de los años 50, siendo estos robots manipuladores, pero sin capacidad de ser programados por un operario humano, en cambio, si podían ser dirigidos para realizar ciertas tareas, pero no conocían el entorno en el cual desarrollaban su función.

Tras esto, alrededor de los años 80, se pasa a la segunda generación (robots de aprendizaje), mediante la implantación de sistemas sensoriales en el robot manipula-

dor, como sistemas de distancia, posición o visión, y la capacidad de ser programado anteriormente por un operario.

La tercera generación se corresponde con el robot de la Figura 2.2, en la cual los robots comenzaron a ser capaces de desenvolverse en ámbitos de trabajo complejos. Actualmente, esta generación no está completa todavía porque se siguen creando nuevos prototipos y modelos.

Por último, respecto a la cuarta generación, son conocidos como robots inteligentes y a parte de recibir información del entorno para realizar la tarea, también son capaces de enviar información sobre el estado del proceso, es decir, intercambian información con el sistema de control.

Figura 2.2: Brazo robot industrial.

2.2. Tipos de robots

Actualmente, existen muchos tipos de robots, con diferentes características y funcionalidades, a continuación se detallarán los tipos de robots más conocidos y empleados.

Robots industriales de manipulación: Son robots con una base fija anclada a una plataforma de trabajo, por tanto, son robots articulados que desplazan un útil de trabajo por el espacio.

Además, pueden ser multifuncionales y reprogramables, y poseen un control absoluto del entorno en el que han de desenvolver su trabajo, sobretodo en ámbitos de producción realizando tareas que requieren mucho esfuerzo físico para el operario. Entre este tipo de robots, se encuentra el brazo robótico con el que se va a trabajar en este proyecto. El de la figura 2.2 también es un robot de este tipo.

Robots de servicio: Son dispositivos controlados por ordenador, suelen ser móviles y sustituyen al hombre en procesos cotidianos. Suelen verse en casas, por ejemplo, los robots empleados en realizar tareas de limpieza como el de la Figura 2.3, que, además, son capaces de adaptarse a cambios en el entorno.

Figura 2.3: Robot de limpieza de hogar.

Robots médicos: Actualmente, la robótica cada vez está más presente en el ámbito sanitario, facilitando muchas tareas al personal sanitario. Para diversas operaciones médicas, se emplean dispositivos láser con una precisión enorme o también en el ámbito de la sanidad dental, donde se disponen de diversos tipos de aparatos. Un ejemplo es el de la Figura 2.4, que facilita y aumenta la precisión de las operaciones realizadas.

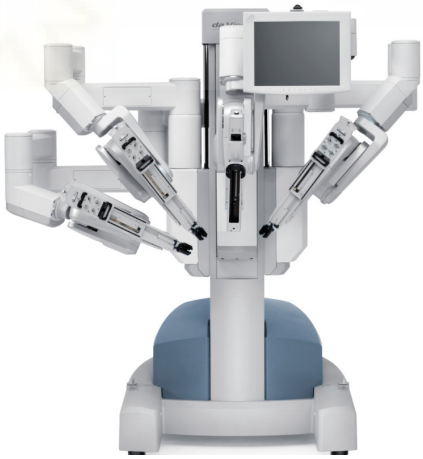


Figura 2.4: Robot quirúrgico.

Nano robots: Los nano robots todavía están en una fase muy prematura, sin embargo, son la tecnología y robótica del futuro, y ya se ha logrado diseñar nano robots, véase la Figura 2.5. La idea es que se inserten dentro del organismo humano y sean capaces de buscar y destruir tumores.

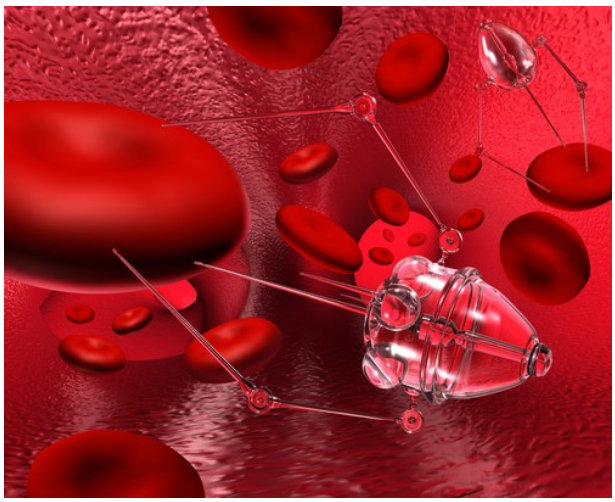


Figura 2.5: Nano robot.

2.3. Robótica Colaborativa

Finalmente, y tras una breve introducción al mundo de la robótica y a su lugar en la sociedad, hay que enfocarse en el campo de la robótica colaborativa. Así pues, gracias a los avances en la tecnología, en especial, de la robótica industrial, es posible hacer realidad el innovador concepto de robótica colaborativa.

La robótica colaborativa aparece de la necesidad del ser humano de interaccionar con robots en el sector industrial, ya que la sociedad y sus exigencias aumentan cada vez más, siendo necesario optimizar los recursos disponibles, disminuyendo tiempos y costes de producción.

Sin embargo esta necesidad no era posible suplirla ya que los brazos robóticos disponibles en la industria no pueden interaccionar con un ser humano cuando están en funcionamiento.Esto supondría un gran peligro para la persona, por eso suelen estar dentro de un espacio de trabajo inaccesible. Además, dichos robots son costosos, de gran tamaño y muy pesados, y disminuir todos estos factores era una de las principales motivaciones que perseguía el concepto de robótica colaborativa. Pero el factor más importante es romper con el peligro que supone la interacción de un ser humano con los robots industriales existentes hasta el momento.

Los llamados “cobots”, están diseñados con una serie de características técnicas que garantizan la seguridad de un trabajador cuando entra en contacto directo con el robot, ya sea deliberadamente o por accidente. Estas características incluyen materiales ligeros, contornos redondeados y sensores en la base del robot o en las articulaciones que miden y controlan la fuerza y la velocidad y se aseguran de que no se excedan los umbrales definidos en caso de que se produzca el contacto.

Lejos de reemplzar a los trabajadores humanos, los cobots mejoran su productividad, liberándoles de tareas monótonas y repetitivas y permitiéndoles centrarse en trabajos

más complejos o finalizar la tarea en colaboración con el robot en un espacio compartido. Los trabajadores se muestran más dispuestos a aceptar la introducción de un robot colaborativo en su entorno de trabajo porque los ven como herramientas que les ayudan y hacen su trabajo más fácil, y no como una tecnología que les vaya a sustituir. Es como trabajar "con un compañero", con posibilidades ilimitadas.

2.4. Estado del arte

Se van a ver por encima a continuación los robots colaborativos de las principales marcas en el mercado.

Es necesario decir que cualquier robot industrial puede llegar a ser colaborativo bajo las circunstancias adecuadas, con ayuda de sensores y utilizando velocidades prudentes para trabajar con un humano. A continuación se explican los más interesantes del mercado a través del prisma del alumno.

2.4.1. KUKA

KUKA es una empresa pionera a nivel mundial en robótica e instalaciones industriales cuya historia se remonta al año 1898.

KUKA ofrece una gran variedad de robots industriales de última tecnología, e igual que muchas empresas pioneras en el sector de la robótica industrial, también ha desarrollado sus propios prototipos de robots colaborativos:

LBR iisy: Con carga máxima de 6 kg y 600 mm de alcance. Dicho por KUKA, fue el primer robot colaborativo de la historia al ser presentado en 2004. Como siempre ocurre, esta afirmación puede ser cierta dependiendo de la definición que se le otorga al concepto colaborativo.

LBR iiwa: Con carga máxima de 14 kg y un alcance de 820 mm. Robot de estructura liviana, iiwa es la abreviatura de "*intelligent industrial work assistant*".

El LBR iiwa sigue el mismo principio en el que están basados los robots colaborativos, trabajar en tareas complejas cooperando con humanos sin ningún peligro o riesgo para estos.

Por otro lado, el LBR iiwa es adecuado para procesos tales como paletizado, embalaje, carga, operaciones de montaje o manipulación de otras máquinas entre otros.

2.4.2. FANUC

FANUC es uno de los principales fabricantes de robots del planeta. No es objetivo de este apartado estudiar la trayectoria de las compañías, pero es importante para

(a) KUKA LBR iisy. **(b)** KUKA LBR iiwa.

Figura 2.6: Modelos de KUKA

entender el calado de estos robots saber que FANUC tiene una cuota de mercado en torno al 25% en robótica tradicional.

Figura 2.7: Gama de robots colaborativos de FANUC.

FANUC comercializa hasta cinco modelos de robots colaborativos, su nombre y características básicas son las siguientes:

- CR-4iA: Colaborativo con cuatro kilos de carga máxima y un alcance de 550 mm.
- CR-7iA y CR-7iA/L: Colaborativo con siete kilos de carga máxima. Disponible con dos alcances, de 717 mm. (modelo estándar) y de 911 mm. (modelo L)

- CR-15iA: Colaborativo de quince kilos de carga máxima. Con alcance de 1441 mm.
- CR-14iA/L: Colaborativo de catorce kilos de carga máxima con un alcance de 911 mm.
- CR-35iA: Colaborativo de treinta y cinco kilos de carga máxima con un alcance de 1813 mm.

Se puede observar que es una gama con unas dimensiones y peso muy variadas, lo que más sorprende es la capacidad de carga de treinta y cinco kilos para el mayor de los robots.

El modelo más grande (CR-35iA), tiene su cuerpo completamente cubierto por un material espumado que absorbe los impactos.

La interfaz de programación es la misma que en los otros robots industriales de FANUC. Esta interfaz se basa en un terminal de programación donde se pueden introducir comandos de forma asistida. No destaca por ser intuitiva ni fácil de usar, pero tampoco llega es necesario tener nivel de programador experto para aprender a utilizarla.

Los robots pueden ser guiados de forma manual para su programación con la ayuda de un sistema especialmente ideado para ellos y que se instala en la muñeca del robot (figura 2.8).

Figura 2.8: Hand Guidance de FANUC.

2.4.3. UNIVERSAL ROBOTS

Universal Robots no es una marca clásica de robots industriales. Esta marca nació en 2005 en Dinamarca buscando ofrecer al mercado robots ligeros, sencillos de instalar y programar. Lideran el nuevo segmento de la robótica que acabará convirtiéndose en los robots colaborativos.

UR siempre apostó por el nuevo tipo de robótica más cercana al ser humano, y con un decálogo de buenas prácticas enseña para qué sirve y para qué no sirve un colaborativo. Esta estrategia de cercanía la llevan desde la programación sencilla de sus productos hasta el aspecto externo que tienen sus robots(ver figura 2.9). Un aspecto que se diferencia mucho de los clásicos robots industriales, invita a tocar el producto e interactuar con él sin miedo.

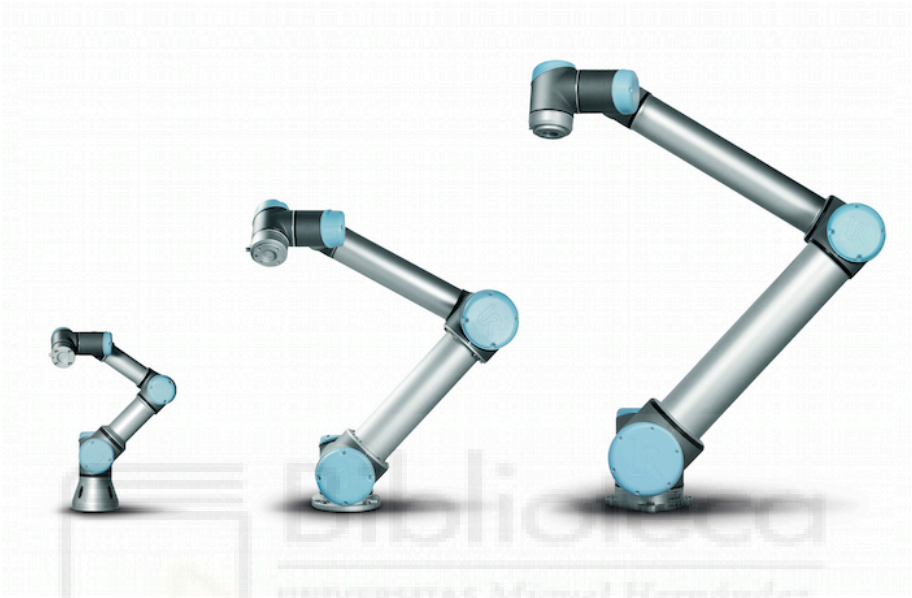


Figura 2.9: Gama de Universal Robots.

Además, esta gama de robots puede integrarse en cualquier tipo de industria, industrias de automoción, para fabricación de productos sanitarios, para industrias alimenticias o para industrias electrónicas.

Actualmente UR consta de los robots:

UR3: De tres kilos de capacidad de carga y con 500 mm de alcance. Además, el peso total es de 11 kg tan solo.

UR5: De cinco kilos de capacidad de carga y con 850 mm de alcance. El peso total es de 18,4 kg.

UR10: El robot más interesante de la gama para la industria en general, tiene diez kilos y 1300 mm de alcance. El peso total es de 29 kg.

UR sacó al mercado en 2018 el modelo mejorado de sus productos, la gama eSeries, con gama de productos similar a la normal, los UR(3,5 y 10) y los URe-Series(e3,e5 y e10). Las dos gamas poseen características similares (la diferencia más notable es la mejora de la resolución en los e-Series). Sin embargo, los URe-Series, fusionan según Universal Robots , "*productividad, adaptabilidad y fiabilidad*" , y además Universal

Robots los describe del siguiente modo: "*La gama e-Series es polivalente, fácil de programar y se puede integrar sin dificultad ninguna en todos los entornos de producción, sea cual sea el tipo de producto o planta de fabricación*, Figura 2.10."

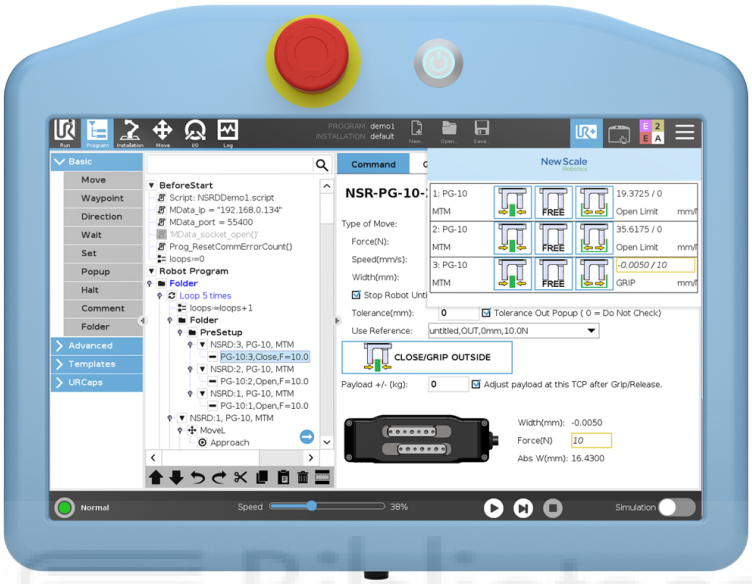


Figura 2.10: Terminal de programación de los modelos UR eSeries.

Los robots UR siempre se han caracterizado y sorprendido a los usuarios gracias a que es posible moverlos con la fuerza humana y así programar sus puntos manejando el robot con las manos directamente. Además tienen una morfología ligeramente diferente a los robots de 6 ejes convencionales, ya que poseen una libertad de movimientos de esfera completa debido a la colocación especial de su cuarto y quinto eje (alternados según la morfología clásica).

El método para hacer estos robots más colaborativos es a raíz de integrar medidores de sobre corriente en los motores de alta sensibilidad, de esta forma pueden identificar si los esfuerzos que está sufriendo el robot son los esperados o si son una colisión. Los modelos eSeries integran un medidor de esfuerzos en la muñeca, de forma que pueden "sentir" el entorno y interactuar con él.

2.4.4. ABB

ABB es otra gran empresa productora de robots tradicionales. Su cuota de mercado es muy similar a la de FANUC, quizás un poco más baja a nivel global pero con una presencia hegemónica en Europa.

ABB fue de las primeras marcas en introducir un robot colaborativo con el robot de dos brazos llamado YUMI (figura 2.11). El YUMI era la apuesta de ABB hacia las

Figura 2.11: ABB Yumi.

nuevas tendencias colaborativas, la forma en la que ellos entendían que se movía la corriente robótica en el año 2015 cuando se presentó.

Este robot colaborativo denominado YuMi (You and Me) es como la recreación de un torso humano con dos brazos que poseen manos flexibles. Incluye un sistema de visión patentado y tecnología de la última generación, según ABB Robotics "*cambiará nuestro concepto de la automatización del ensamblaje. YuMi® es como trabajar "con un compañero", con posibilidades ilimitadas*".

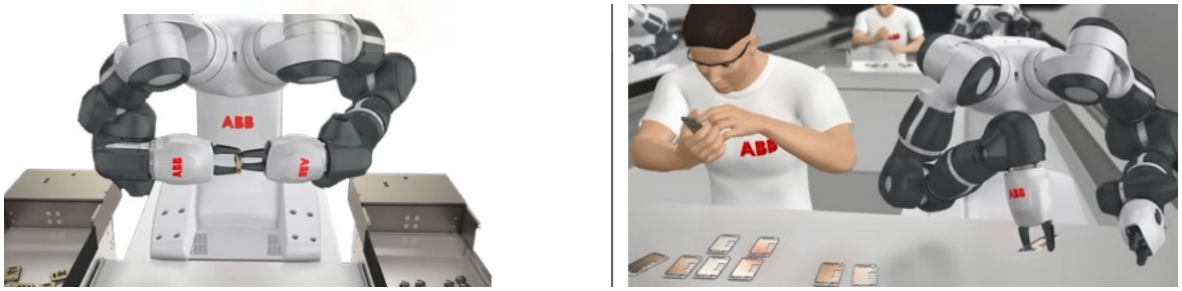


Figura 2.12: Tareas de precisión con YuMi.

YuMi es una solución robótica de colaboración para el ensamblaje de piezas pequeñas, con doble brazo que incluye manos flexibles, sistemas de alimentación de piezas, localización de piezas mediante cámara y control robótico de última generación. Puede colaborar mano a mano con el ser humano en un entorno de fabricación normal, contribuyendo a que las empresas obtengan el mejor rendimiento de sus empleados y sus robots.

Cada uno de los brazos posee 7 grados de libertad, 559mm de rango de alcance y pueden levantar 0.5 kg cada uno.

El robot también monitoriza los esfuerzos que se realizan a través del consumo eléctrico de sus motores. El robot es capaz de dejarse guiar mecánicamente por el programador, para que la programación de sus puntos sea más sencilla.

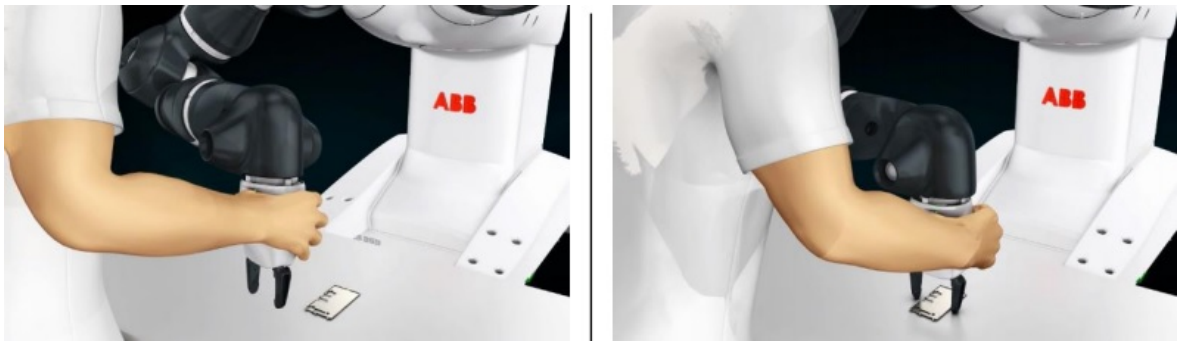


Figura 2.13: Operario programando cogida para el Yumi.

Gracias a sus dos brazos puede realizar los movimientos necesarios en líneas de ensamblaje de piezas pequeñas en un espacio muy reducido, con capacidad similar a la del ser humano. Este factor es determinante para minimizar el espacio ocupado en la planta, y también permite instalar YuMi en estaciones de trabajo que actualmente solo utilizan los operadores humanos.

Debido probablemente a las restricciones que impone poner los brazos fijos en el torso del YUMI original, ABB presenta en 2018 el YUMI de un solo brazo (Figura 2.14).

Figura 2.14: IRB 14050 Brazo individual YuMi.

Este YUMI de un solo brazo tiene el mismo peso admisible, ejes y alcance que su "hermano" de dos brazos (500 gramos, 7 ejes y 559 mm).

Los YUMI ofrecen equipamiento previamente diseñado para el uso del robot. Destacan las herramientas en forma de pinza con el suministro de aire a presión incluido en ellas, y las que tienen cámara para la visión integrada (figura 2.15). Para las pruebas reales del trabajo se dispuso de una pinza normal y otra con cámara con visión integrada.

Figura 2.15: Pinzas equipables para los robots YUMI.

Además, los YUMI se programan como el resto de los robots de la marca ABB, mediante el mismo terminal táctil y mismo código (RAPID), y se pueden realizar simulaciones virtuales mediante el programa de la marca RobotStudio.

Ya que este es el robot utilizado para el trabajo, se enlazara su hoja de especificaciones al final del proyecto.

3. Metodología

3.1. Introducción a RobotStudio

El programa Robotstudio ABB es un software que permite crear, programar y simular estaciones de robots industriales, diseñado y patentado por la empresa ABB, líder en este tipo de robot. Utiliza el lenguaje RAPID.

RobotStudio posee un controlador virtual, una copia exacta del software real que emplean los robots en la producción. Ello permite programar un robot fuera de línea (offline) en un PC sin necesidad de parar la producción en la industria, exportando los resultados obtenidos en simulación a la estación real.

El programa además proporciona herramientas para incrementar la rentabilidad de su sistema robotizado mediante tareas como formación, programación y optimización, sin afectar la producción, lo que proporciona numerosas ventajas, como reducción de riesgos, arranque más rápido, transición más corta e incremento de la productividad.

3.1.1. Interfaz gráfica

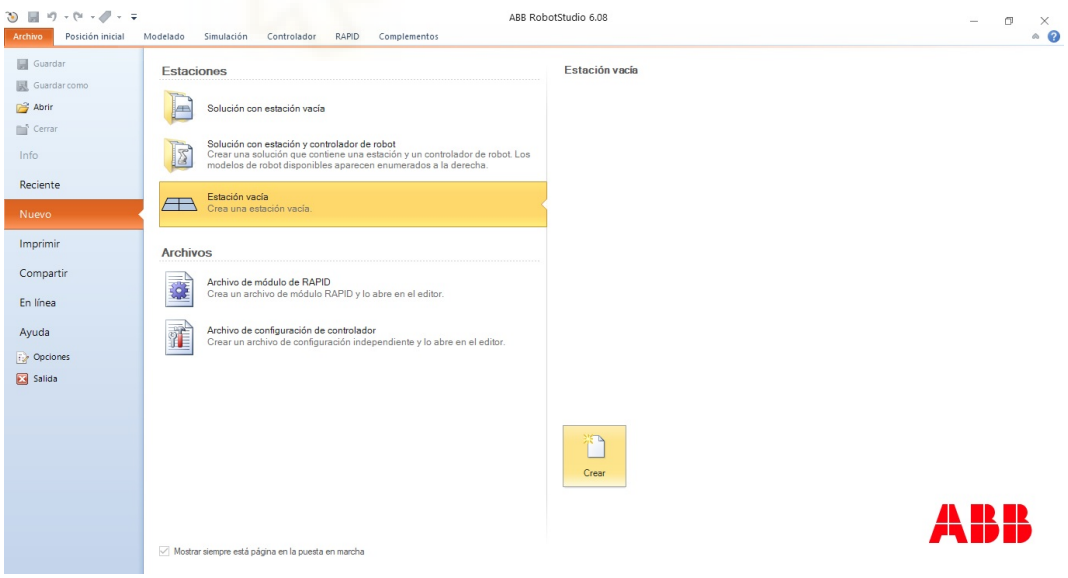


Figura 3.1: Creación nueva estación.

La interfaz gráfica de usuario está dividida en menús que se seleccionan activando la

pestaña.

Archivo: contiene las opciones necesarias para crear una estación o un sistema de robot nuevo, guardar o abrir estaciones existentes, etc. Figura 3.1.

Posición inicial: contiene los comandos necesarios para construir estaciones, crear sistemas, programar trayectorias y colocar elementos.

Figura 3.2: Pestañas de RobotStudio y Menú de Inicio.

Modelado: contiene las funciones necesarias para crear y agrupar componentes, crear cuerpos, mediciones y operaciones de CAD.

Figura 3.3: Pestaña de Modelado.

Simulación: contiene los controles necesarios para crear, configurar, controlar, monitorizar, y grabar simulaciones.

Figura 3.4: Pestaña de Simulación.

Controlador: contiene los comandos necesarios para la sincronización, configuración y tareas asignadas al controlador virtual; así como para gestionar los controladores reales.

Figura 3.5: Pestaña de Controlador.

Rapid: contiene los controladores necesarios para crear y modificar los programas necesarios del robot.

Figura 3.6: Pestaña de Rapid.

Complementos: contiene los controles de los PowerPacs y de VSTA. Además se podrán observar los valores del calor de la caja reductora y descargar componentes adicionales para las librerías del software.

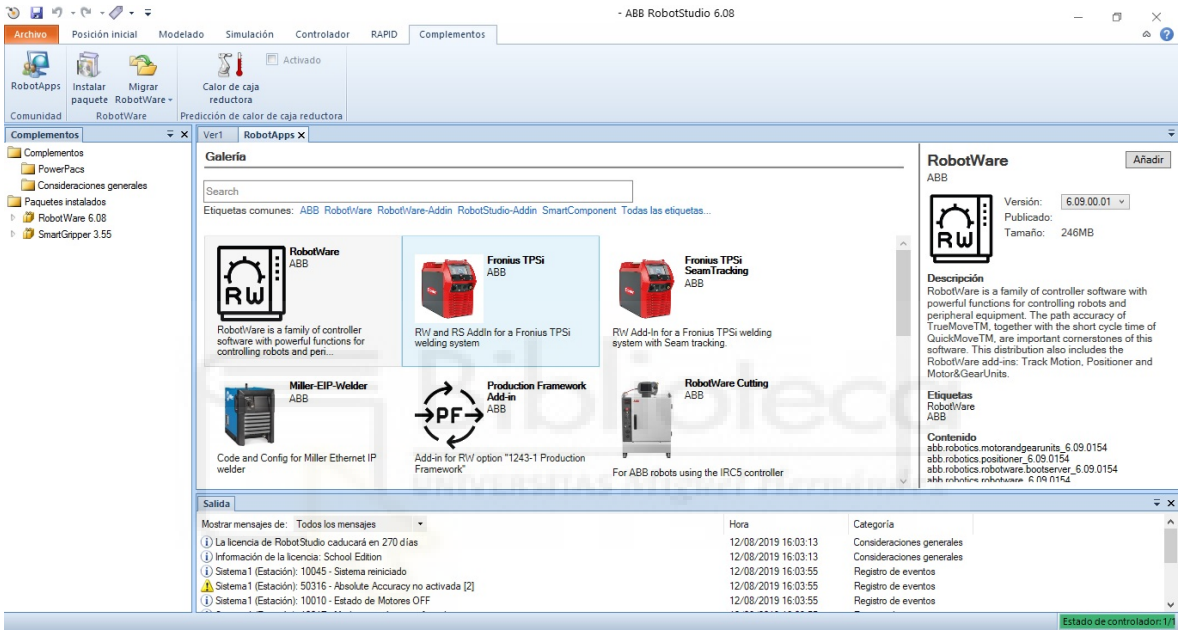


Figura 3.7: Pestaña de Complementos.

Es necesario también conocer el entorno del sistema(Figura 3.8). Se aprecian diferentes áreas en el entorno:

Menús: son menús desplegables, que permiten realizar todo tipo de acciones las cuales han sido explicadas brevemente anteriormente.

Barras de herramientas: dispone de los diversos comandos de las que dispone el programa. Están agrupados por funciones.

Ventana gráfica: en ella se pueden observar los objetos en 3D de la estación, seleccionarlos, etc. Es decir, es la extensión gráfica del navegador. Pueden crearse diversos grupos de pestañas para visualizar a la vez menús.

Accesos rápidos de la ventana gráfica: son accesos rápidos a los comandos del programa más recurridos.

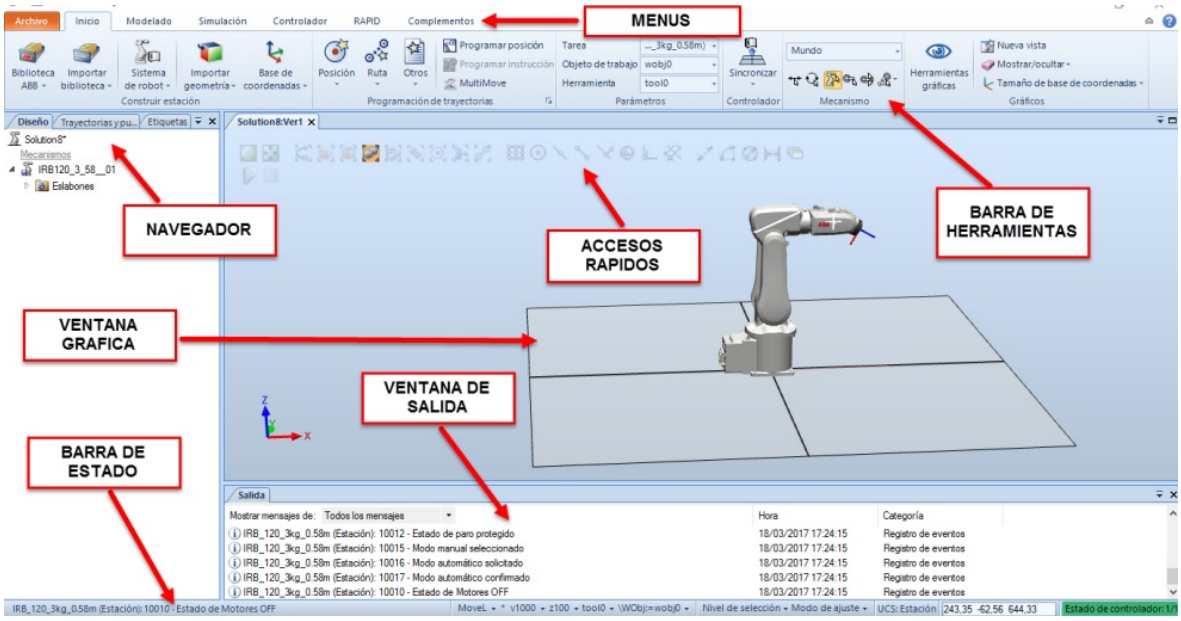


Figura 3.8: Elementos del entorno RobotStudio.

Navegador: es una ventana en la que se muestran de manera jerárquica los objetos de la estación y a su vez se permiten seleccionarlos. Su pueden encontrar dentro de ella otras ventanas diferentes dependiendo del menú en el que se esté.

Ventana de salida: muestra información sobre los eventos que se producen en la estación, por ejemplo cuando se inicia o detiene una simulación. Esta información resulta útil a la hora de solucionar problemas de las estaciones.

Barra de estado: muestra algunos de los parámetros actuales de la estación, como el estado del control virtual, o la posición del sistema de coordenadas del usuario.

3.1.2. Creación de una estación

Una vez instalada la licencia y ejecutado el programa, lo primero que hay que realizar es crear una nueva estación en “Nuevo” y “Estaciones”, tenemos varias opciones, tal y como aparece en la figura 3.1.

A continuación se importa el robot, en la pestaña “Posición Inicial”, clic sobre Biblioteca ABB y se despliega un listado con los modelos de robots ABB, se selecciona el robot IRB 14000 para realizar esta introducción.

El IRB 14000 se carga en la estación, figura 3.9.

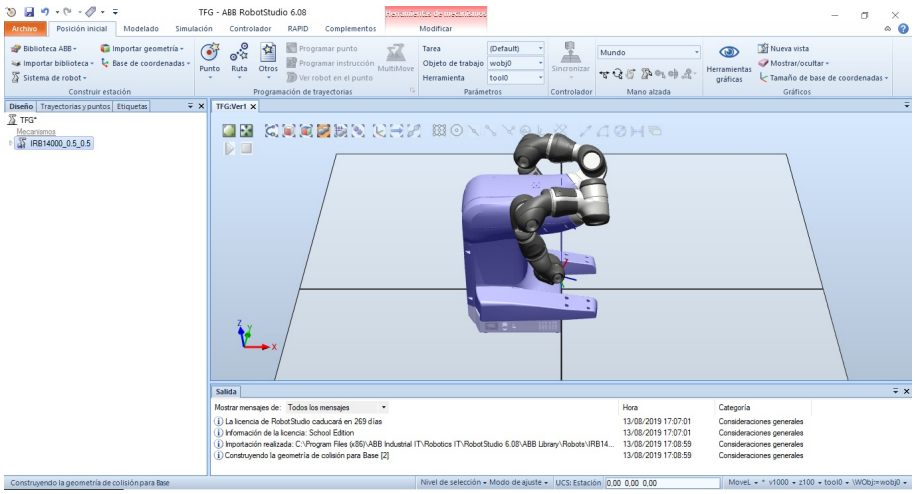


Figura 3.9: Estación con IRB 14000 cargado.

3.1.3. Creación y equipamiento de una herramienta en el entorno virtual

Para poder realizar operaciones y memorizar posiciones hay que hacerlo respecto a una herramienta de trabajo del robot. Esta herramienta va incorporada a la muñeca del mismo. Las hay muy diferentes, pinzas, pistolas de soldadura o pintura, electro-imanes, ventosas, etc.

Existe una gran variedad de herramientas, las cuales se pueden encontrar dentro de la Pestaña Inicio realizando clic en Importar Biblioteca → Equipamiento.

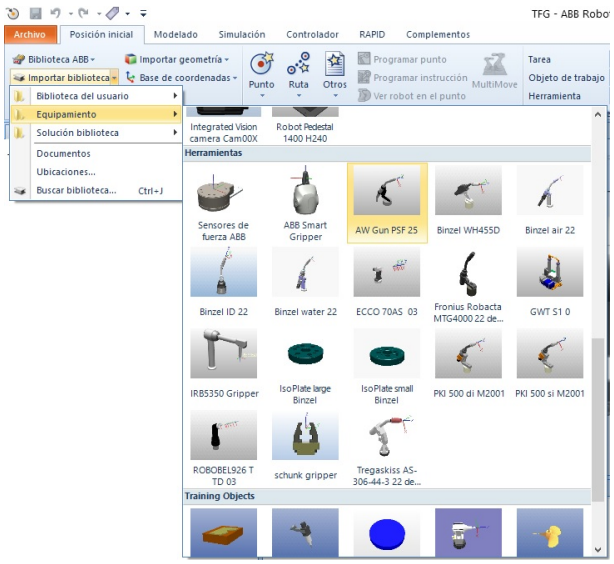


Figura 3.10: Herramientas equipables en RobotStudio.

También existe la posibilidad de crear una herramienta propia partiendo de una geometría, o importarlas de ficheros de diseños propios hechos en otros programas. Por ejemplo, se puede crear un cono dirigiéndose a la Pestaña Modelado y pinchando sobre 'Sólido'.

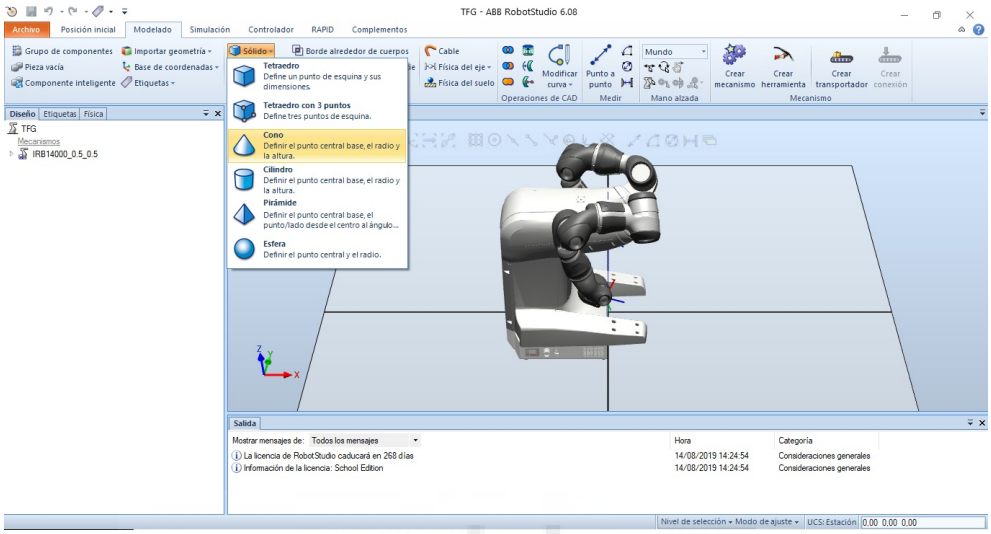


Figura 3.11: Creación de sólidos.

Se configuran los parámetros del cono en la figura 3.12, con radio de 32 mm que es aproximadamente el radio de la última articulación del robot, diámetro de 64 mm y una altura de 100 mm y se selecciona crear. Al no realizar cambios en la posición, la pieza será creada por defecto en el (0,0,0).

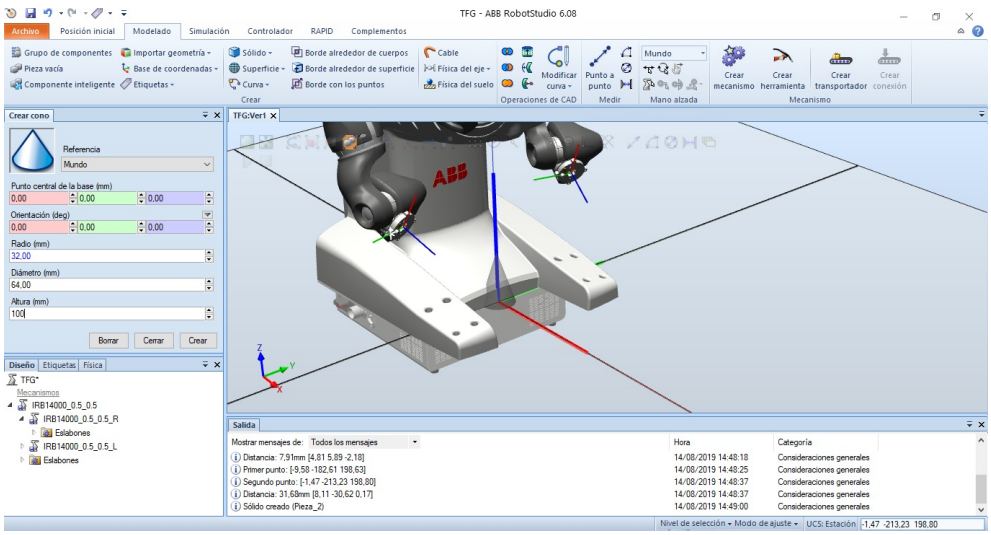


Figura 3.12: Parámetros para el cono.

A continuación, hay que ir a la Pestaña Modelado → Crear herramienta. En primer lugar hay que ponerle un nombre a la herramienta, en este caso 'Cono' y seleccionar como pieza una existente, en este caso la creada anteriormente.

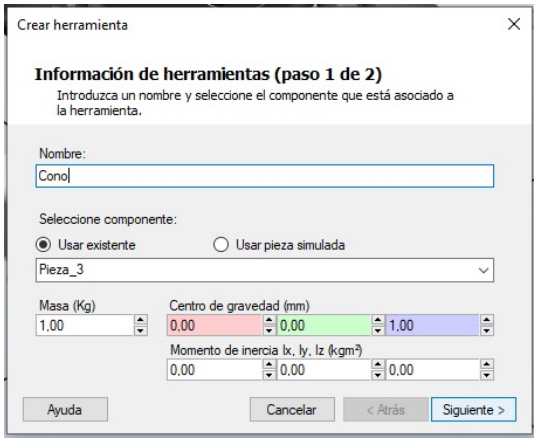


Figura 3.13: Crear herramienta. Paso 1 de 2.

En segundo lugar es necesario asociarle una posición y orientación relativa al extremo de la herramienta, para que las posiciones del robot que se creen posteriormente se realicen respecto a dicho extremo. En este caso, al ser creado el cono con una altura de 100 mm, en el eje Z habrá que asociarle esa posición en ese eje. Dicha posición es denominada TCP (Tool Central Point) y se le asigna realizando clic al botón de la flecha que aparece en la figura 3.14.

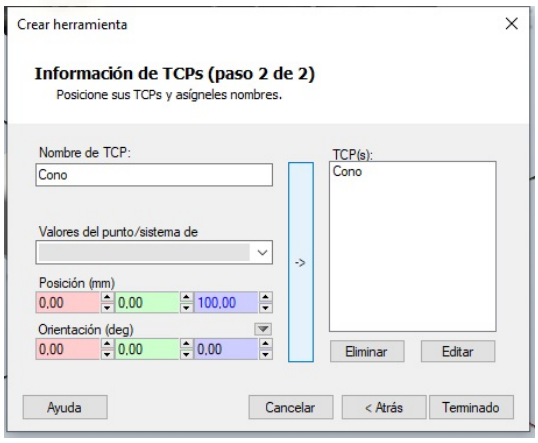


Figura 3.14: Crear herramienta. Paso 2 de 2.

Para finalizar hay que acoplarla al robot como en la figura 3.15, seleccionando "Cono" en diseño y arrastrarlo al robot IRB 14000 o bien hacer clic derecho y seleccionar "Conectar a", y elegir aquel brazo al que deseemos acoplar la herramienta, y automática-

mente la herramienta se acoplaráá en la muñeca del robot.

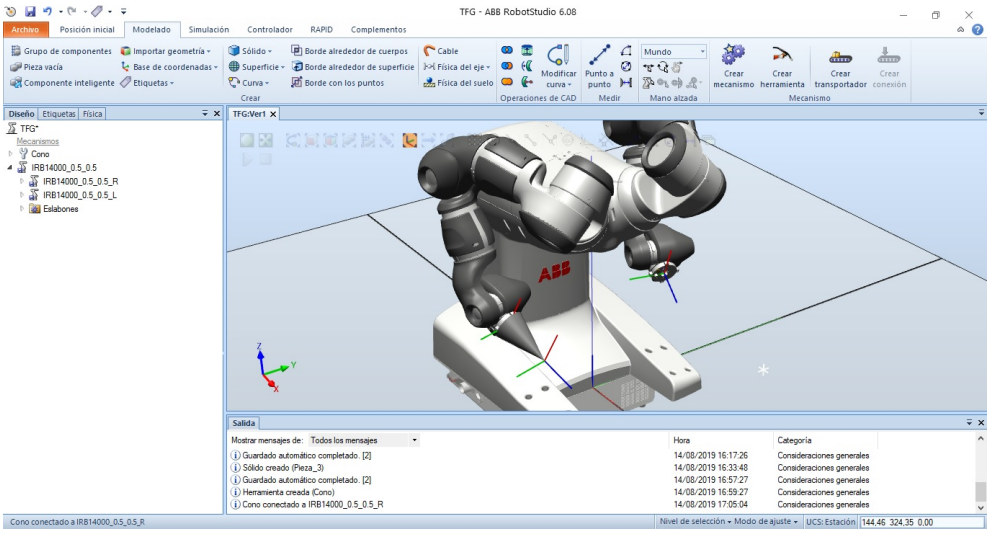


Figura 3.15: Herramienta acoplada.

Para las aplicaciones del proyecto y a partir de ahora se utilizará una pinza de la pestaña equipamiento llamada ABB Smart Gripper, que es lo que también tenemos en el robot real. Es necesario añadir dos pinzas y conectar una a cada brazo (Figura 3.16).

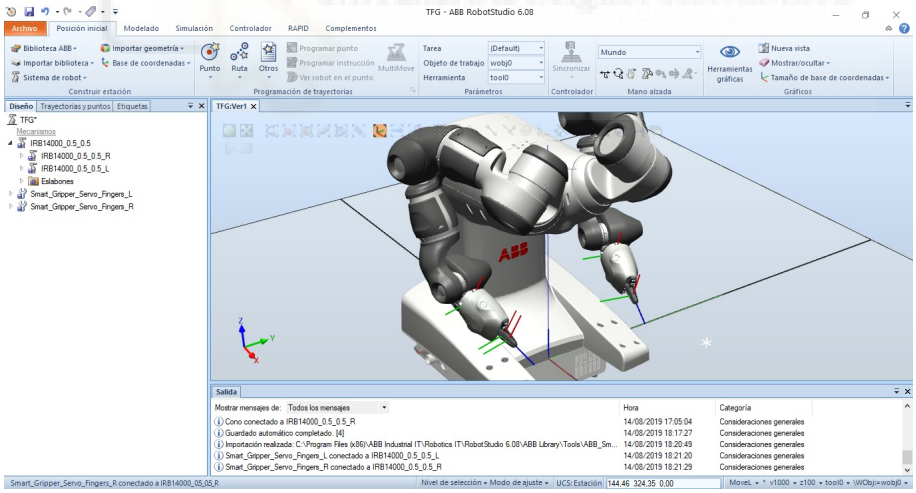


Figura 3.16: Un ABB Smart Gripper por brazo.

3.1.4. Creación del controlador del robot

Hasta ahora, lo que se tiene es un brazo con una herramienta adjunta. Sin embargo, con esto tan solo no se pueden realizar movimientos ni programar trayectorias. Para

dotar de “inteligencia” al robot es necesario crear un controlador asociado al mismo. Como ya se mencionó, esto permitirá al robot virtual moverse exactamente como lo haría el robot real.

Para ello, hay que ir a la Pestaña Inicio → Sistema de robot → Desde diseño…

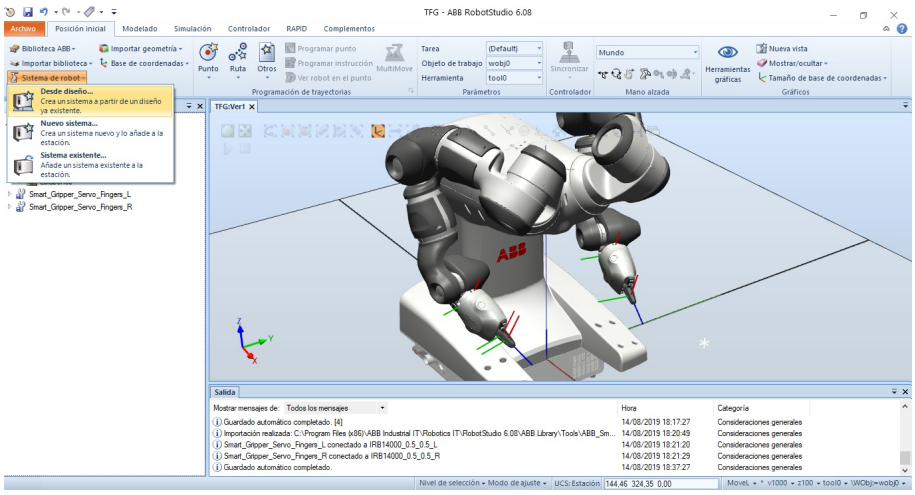


Figura 3.17: Controlador desde diseño.

Le damos un nombre al controlador en la figura 3.18 y pulsamos en “Siguiente”, seleccionamos los mecanismos de ambos brazos y pulsamos de nuevo “Siguiente”, y a continuación ”Finalizar” para terminar la configuración del Controlador.

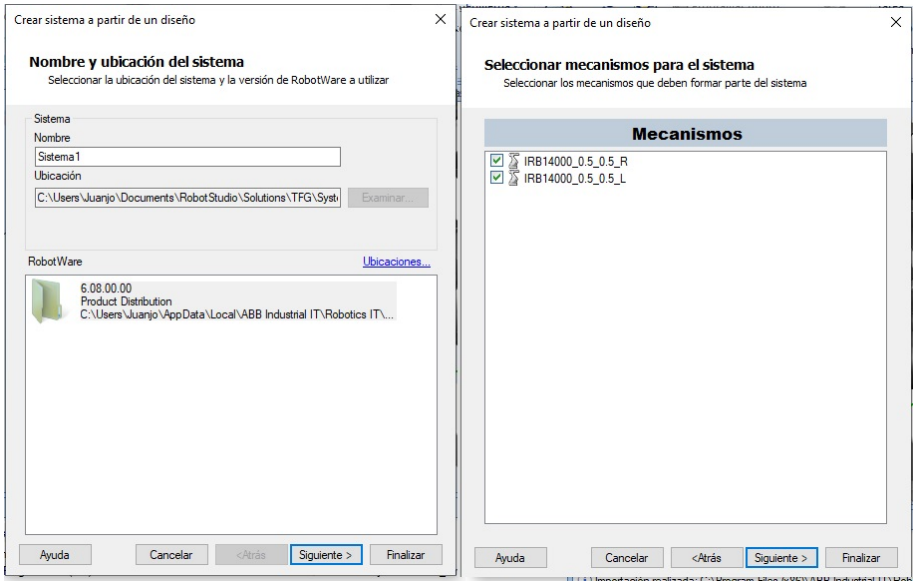


Figura 3.18: Ventana de creación del controlador.

3.1.5. Creación de planos de trabajo y posiciones

Los planos de trabajo son puntos de referencia respecto de los que se referencian los puntos para trabajar con ellos de forma más cómoda. Para crear un plano de trabajo se hace clic en "Pestaña de Inicio", se selecciona "Otros" y después "Crear objeto de trabajo" como en la figura 3.19.

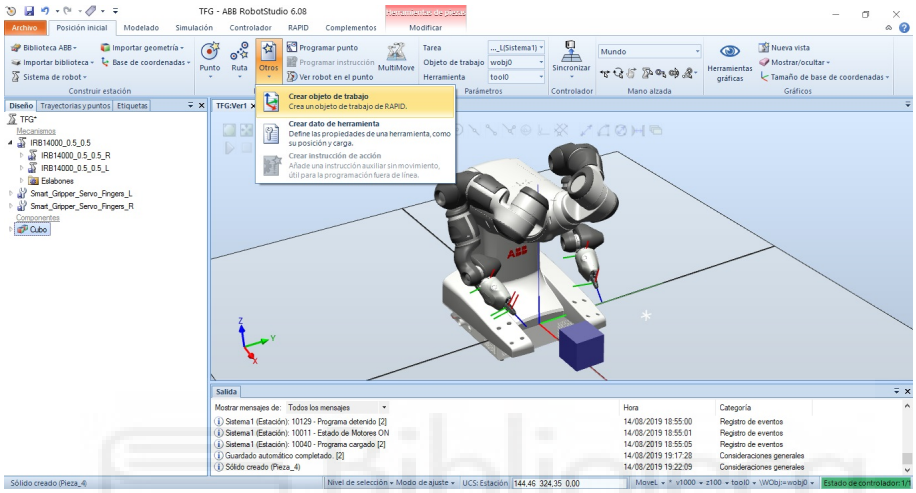


Figura 3.19: Creación de un objeto de trabajo.

Se nombra el objeto de trabajo y se sitúa su sistema de coordenadas en la esquina superior del cubo, que va a ser la trayectoria de ejemplo. Por último, pinchando en Aceptar y Crear ya se tiene el objeto de trabajo creado.

Antes del siguiente paso es importante seleccionar el robot y el objeto de trabajo en el que se desea trabajar, Figura 3.20

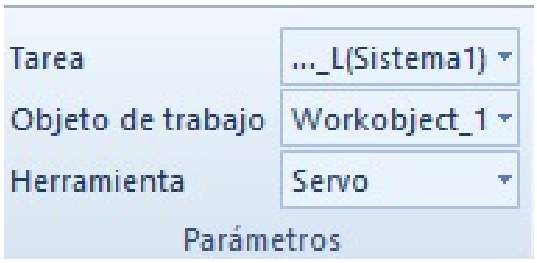


Figura 3.20: Objeto de trabajo y herramienta seleccionados.

El siguiente paso es crear posiciones relativas al Cubo. Para ello, hay que ir a Posición → Crear Punto, dentro de la Pestaña Inicio.

Figura 3.21: Crear punto.

Después, como en la Figura 3.22, creamos cuatro puntos señalando las esquinas superiores del cubo. Para finalizar se pincha en crear, y ya estarían listos los puntos para posteriormente crear una trayectoria.

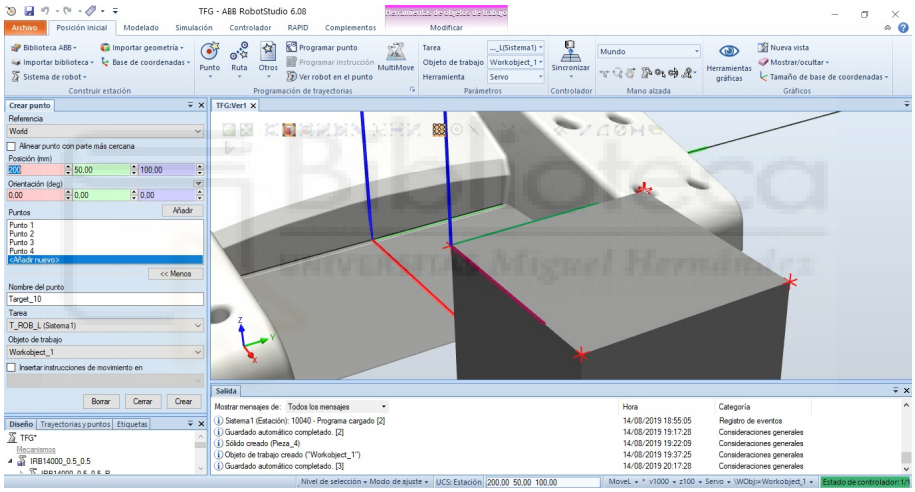


Figura 3.22: Elección de los puntos objetivo.

3.1.6. Creación de trayectorias

Una vez creado el conjunto de puntos, es importante darle a estos puntos la orientación de la herramienta deseada, ya que por defecto se les asigna una deseada y esta seguramente no sea la idónea. Para solucionar esto, es necesario hacer clic derecho sobre cada punto y pinchar sobre Modificar posición → Girar. A continuación hay que configurar el punto girando respecto a los ejes x,y,z de forma que el robot pueda alcanzar cada punto correctamente.

Para realizar la misma operación de giro en el resto de puntos creados basta con clicar con el botón derecho en en el primer punto y pinchar sobre 'Copiar orientación'. Tras esto, hay que señalar los otros puntos y clicando en el botón derecho pinchar sobre

'Aplicar orientación'.

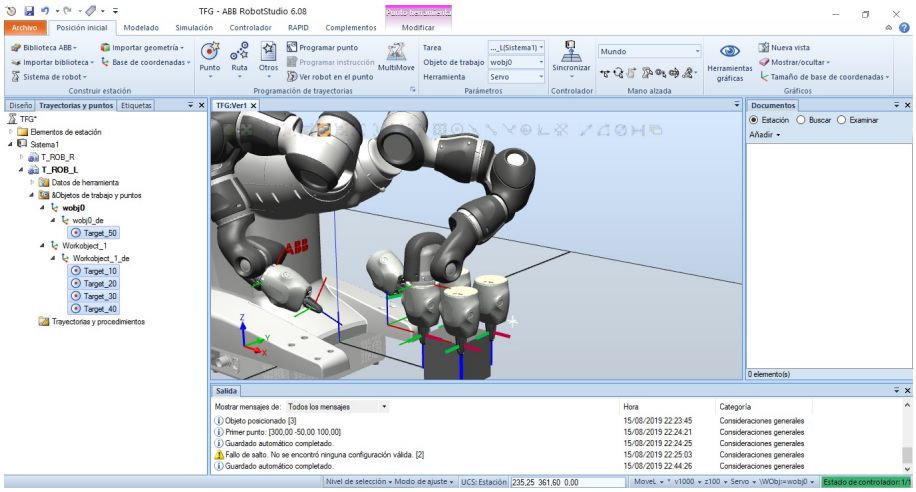


Figura 3.23: Puntos con la orientación correcta.

Para realizar la trayectoria se seleccionan las siguientes opciones, en este orden: "Trayectoria vacía", "Posición Inicial", y finalmente "Ruta".

Al crear una Trayectoria vacía, se creará la trayectoria Path_10, la cual se encuentra vacía. Para ver un ejemplo, para seguir una trayectoria cíclica, se arrastraría primero el punto de origen, a continuación los puntos en el orden correcto y finalmente el punto de origen de nuevo.

Se ha realizado la trayectoria que empiece en el origen, recorra las cuatro esquinas del cubo y vuelva al origen. Por defecto se creará el movimiento de la trayectoria en MoveL como se observa en la Figura 3.24.

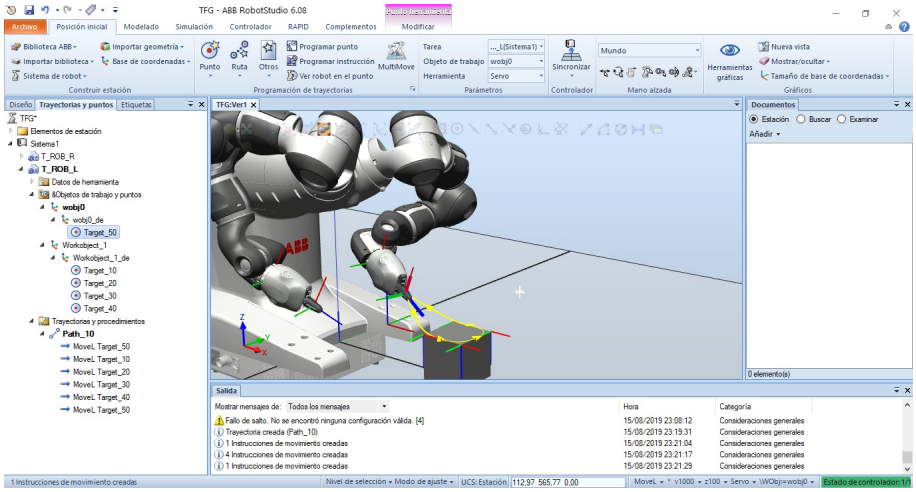


Figura 3.24: Trayectoria por los vértices superiores de un cubo.

En cada punto, se puede además seleccionar la configuración de los ejes del robot. Las configuraciones de los ejes del robot se designan con una serie de cuatro números enteros que especifican en qué cuadrante se encuentran los ejes. Por ejemplo, la configuración [-1, 0, 1, 0] significa:

- El primer entero (-1) especifica la posición del eje 1: en algún punto del primer cuadrante negativo (rotación entre 0 y -90 grados).
- El segundo entero (0) especifica la posición del eje 4: en algún punto del primer cuadrante positivo (rotación entre 0 y 90 grados).
- El tercer entero (1) especifica la posición del eje 6: en algún punto del segundo cuadrante positivo (rotación entre 90 y 180 grados).
- El cuarto entero (0) especifica la posición del eje X: un eje virtual utilizado para especificar el centro de la muñeca respecto a los demás ejes.

Además de la configuración tenemos varias opciones en las trayectorias que cambiar como el movimiento, la velocidad, precisión, herramienta y objeto de trabajo. Seleccionando un punto de trayectoria o varios y con clic derecho, en el menú "Modificar una instrucción" se pueden ver las siguientes opciones:

Plantilla de Instrucciones: podremos cambiar el tipo de movimiento MoveL a MoveJ o MoveC, en el apartado siguientes se verá que significa cada tipo de movimiento.

Speed: se puede 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 hacen referencia 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. Siendo fine el más preciso y z200 el menos preciso, por defecto aparece z100.

Herramienta: para seleccionar la herramienta de referencia.

Objeto de trabajo: finalmente también se puede escoger entre los objetos de trabajo existentes.

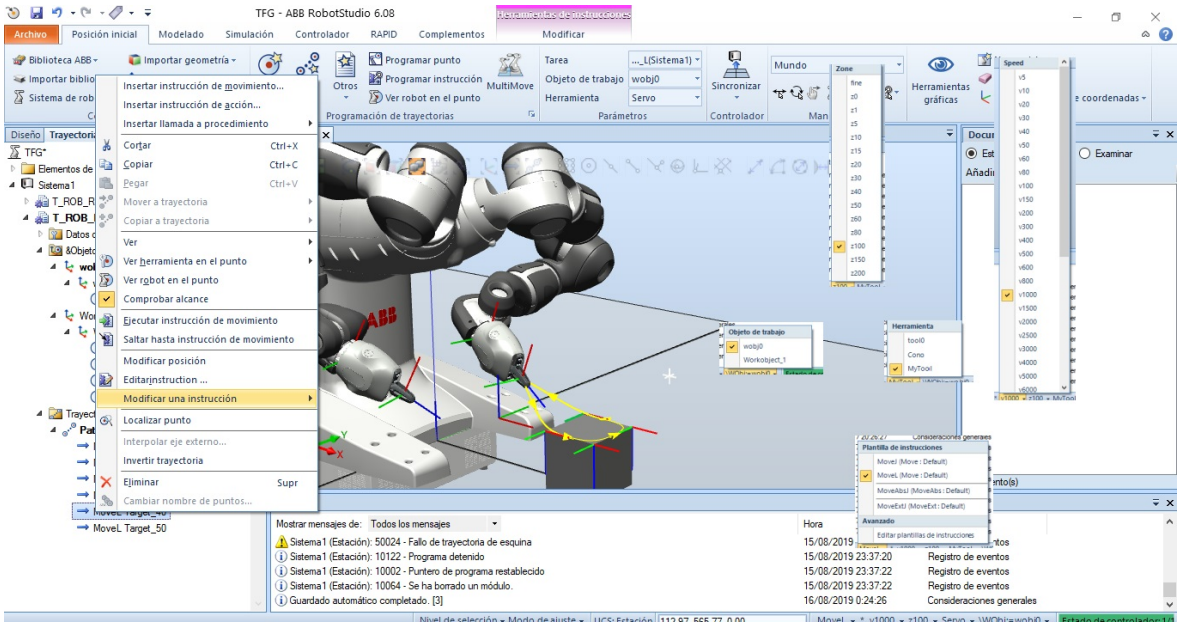


Figura 3.25: Instrucciones de los puntos de la trayectoria.

3.2. Introducción a RAPID

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. Se utiliza tanto para programar las estaciones virtuales como las reales .El programa es una secuencia de instrucciones que controlan el robot y en general consta de tres partes:

- Una rutina principal (main): Rutina donde se inicia la ejecución.
- Un conjunto de sub-rutinas: Sirven para dividir el programa en partes más pequeñas a fin de obtener un programa modular.
- Los datos del programa: Definen posiciones, valores numéricos, sistemas de coordenadas, etc.

3.2.1. Estructura e instrucciones básicas

Para poder programar entramos en la pestaña RAPID y hacemos clic dentro del controlador, abriendo Rapid y en "T_ROB_L" seleccionando el Module1. Después de sincronizar la estación aparecerán los puntos definidos anteriormente, Figura 3.26. Los puntos son constantes llamadas robtarget y también se observa la trayectoria con sus movimientos. El main es el programa principal, es posible realizar cambios del programa en este apartado y aplicar también los cambios realizados.

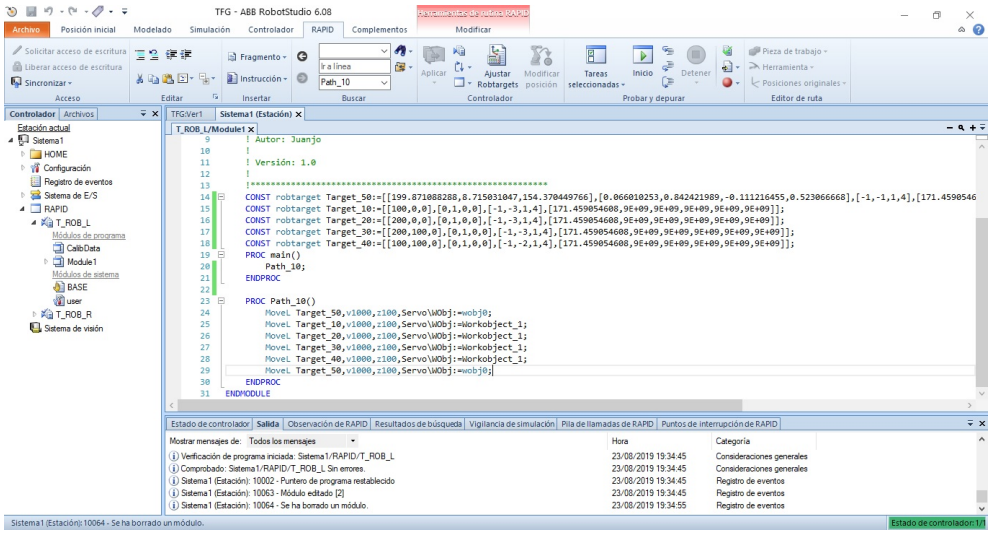


Figura 3.26: Programa RAPID.

En primer lugar, el módulo del programa 'Module1' engloba todos los programas. Éste se inicia con la instrucción 'MODULE Module1' y finaliza con 'ENDMODULE'. Para cada programa, se inicializa con 'PROC Nombre_del_programa ()' para terminarlo con 'ENDPROC'. Dentro de Module1 existe un apartado denominado 'Declaraciones de datos' donde están definidas las posiciones del robot. Esta definición se realiza mediante el comando CONST seguido de la palabra robtarget, que sirve para definir la posición del robot y de sus ejes externos.

```
CONST robtarget Target_40:=[[100,100,0],[0,1,0,0],[-1,-2,1,4],[171.459054608,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Dentro de los corchetes entre los que se define el robtarget, hay otros cuatro corchetes que corresponden a los siguientes datos del robtarget en el orden siguiente:

trans : Posiciones (x, y, z).

rot : Orientación de la herramienta en forma de cuaternio, por eso hay cuatro números.

robconf : Configuración de los ejes, como se explica con mas profundidad en el punto anterior "Creación de trayectorias".

extax : posición de los ejes externos.

Los elementos que se pueden definir en RAPID pueden ser de tres tipos:

CONST (Constantes): representan datos de un valor fijo a los que no se puede reasignar un nuevo valor.

VAR (Variables): son datos a los que se les puede asignar un nuevo valor durante la ejecución del programa.

PERS (Persistentes): se trata de variables en las que cada vez que se cambia su valor durante la ejecución del programa, también se cambia el valor de su inicialización.

En segundo lugar, la instrucción principal de movimiento del robot es MoveL (o MoveJ). Los argumentos que acompañan estas instrucciones especifican la velocidad del movimiento, la precisión con la que se mueve el robot hasta el objetivo, la herramienta asociada a la posición y el plano de trabajo en la que está definida.

```
MoveL Target_10,v1000,z100,Servo\WObj:=Workobject_1;
```

Para el argumento de la velocidad (speeddata) los datos predefenidos van desde v5 hasta v7000, además de un valor vmax que depende del tipo de robot utilizado. Así, el valor v5 indica una velocidad de movimiento de 5 mm/s y v7000 hace referencia a que la velocidad es de 7000 mm/s.

Los elementos zonedata pueden ir desde fine hasta z200. Estos valores hacen referencia al error en distancia (en mm) que es permisible el TCP de la herramienta del robot al punto objetivo. Por ejemplo, z200 significa un error de 200 mm, z50 un error de 50 mm y fine que no existe error a la hora de aproximarse a la posición.

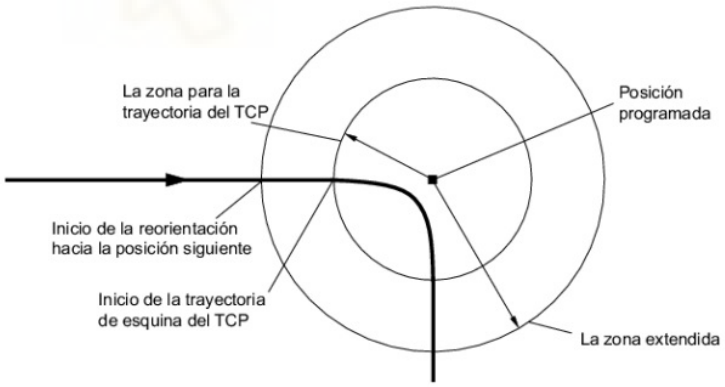


Figura 3.27: Error de precisión en la trayectoria de aproximación del TCP a una posición determinada.

Hay 3 tipos principales de movimiento:

MoveL: desplazamiento del extremo del robot hasta el punto indicado siguiendo una línea recta.

MoveC: desplazamiento del extremo del robot hasta el punto indicado siguiendo un círculo, habiendo dos posiciones, una definiendo un punto intermedio del círculo y otra que define el punto destino.

MoveJ: desplazamiento del extremo del robot hasta el punto indicado rápidamente, sin garantizar cuál es la trayectoria seguida (no hay coordinación de velocidad entre los distintos ejes del robot).

Existe la posibilidad de definir posiciones relativas a otras posiciones ya existentes. Para ello, se tiene las instrucciones Offs y RelTool.

Con la instrucción Offs, movemos el robot al punto añadiendole un desfase, en el siguiente ejemplo se le añaden 20 mm en X y en Y, y 50 mm en Z. También se pueden añadir desfases negativos.

```
MoveJ Offs(Pos_ini,20,20,50),v1000,z100,Servo\WObj:=wobj0;
```

RelTool es similar, pudiendo añadir giros en la orientación de la herramienta respecto de la orientación del punto inicial. En el siguiente ejemplo se añade un giro en el eje Z de 45º positivos.

```
MoveJ RelTool(Pos_ini,50,10,30,\Rz:=45),v1000,z100,Servo\WObj:=wobj0;
```


En los programas a desarrollar en este trabajo, los controladores utilizados tienen asignados un conjunto de señales de entrada y salida digitales (Digital Inputs [DI], Digital Outputs [DO]). Para esperar que una señal de entrada digital se active o desactive se utiliza la instrucción WaitDI seguida de la entrada correspondiente y de un 1 o un 0 según se desee.

```
WaitDI custom_DI_0,1;
```

Para activar y desactivar las salidas digitales se utilizan las instrucciones Set y Reset, respectivamente, seguidas de la señal de salida que se quiera modificar. Como se aprecia, para escribir un comentario se utiliza el comando '!' seguido de lo que se quiera comentar.

```
Set Pinza; !Cierra la pinza para coger un objeto
Reset Pinza; !Abre la pinza para soltar el objeto
```

Si se quiere que el programa espere un tiempo antes de ejecutar la siguiente instrucción se utiliza WaitTime seguido del número de segundos que se quiera esperar.

```
WaitTime 5; !Espera 5 segundos
```

Existen diferentes instrucciones utilizadas para controlar el flujo de funcionamiento, de forma similar a otros lenguajes de programación, del tipo if, for, while, etc.

También existen interrupciones llamadas rutinas TRAP, son funciones utilizadas para tener la posibilidad de tratar directamente un evento, independientemente de qué instrucción se esté ejecutando en cada momento. Por ejemplo, el programa se interrumpe cuando se cambia a uno el valor de una entrada determinada. Cuando esto se produce, se interrumpe el programa normal y se ejecuta una rutina TRAP especial. Una vez ejecutada completamente la rutina, la ejecución del programa se reanuda en el punto en que se interrumpió.

Además podemos disponer de un gestor de errores en cada rutina, el gestor de ERROR puede iniciar un nuevo movimiento temporal y por último reanudar el movimiento original interrumpido y detenido. Por ejemplo, puede usarse para ir a una posición de servicio o para rellenar una pistola de pintura.

Por último, mencionar que para entrar en la rutina de un programa desde otro, simplemente se escribe el nombre del programa que se quiera llamar seguido de punto y coma.

3.2.2. Objetos de trabajo y movimientos coordinados

3.2.2.1. Objetos de trabajo coordinados

Los wobjdata se utilizan para describir el objeto de trabajo que el robot está soldando, procesando, moviendo, pintando, etc. Deben definirse como variables persistentes (PERS), de esta forma, los valores se guardan al guardar el programa y se recuperan al cargarlo. La definición de un objeto de trabajo tiene la siguiente forma:

PERS wobjdata wobj2 :=[FALSE, TRUE, ””, [[300, 600, 200], [1, 0, 0 ,0]], [[0, 200, 30], [1, 0, 0 ,0]]];

Los argumentos que se le pasan para definir el objeto de trabajo son los siguientes en este orden:

robhold: Define si el robot de la tarea de programa actual es el que está sosteniendo el objeto de trabajo. Se le da valor TRUE si el robot sostiene el objeto de trabajo, como por ejemplo si hubiera una herramienta estacionaria y el robot llevase una pieza. Normalmente se pone FALSE ya que el robot suele tener una herramienta.

ufprog Define si se está utilizando un sistema fijo de coordenadas del usuario. Si es fijo se pone TRUE. Si es FALSE significa que es un sistema móvil de coordenadas, es decir, que los puntos asociados a este plano se pueden mover junto con este.

ufmec: La unidad mecánica con la que se mueven el objeto de trabajo. Sólo se especifican en el caso de los sistemas móviles de coordenadas del usuario (ufprog debe ser FALSE).

uframe: El sistema de coordenadas del usuario, es decir, la posición de la superficie o del útil de trabajo actual. Normalmente el sistema de coordenadas del usuario se define en el sistema de coordenadas mundo.

oframe: El sistema de coordenadas del objeto, es decir, la posición del objeto de trabajo actual. El sistema de coordenadas del objeto se define en el sistema de coordenadas del usuario.

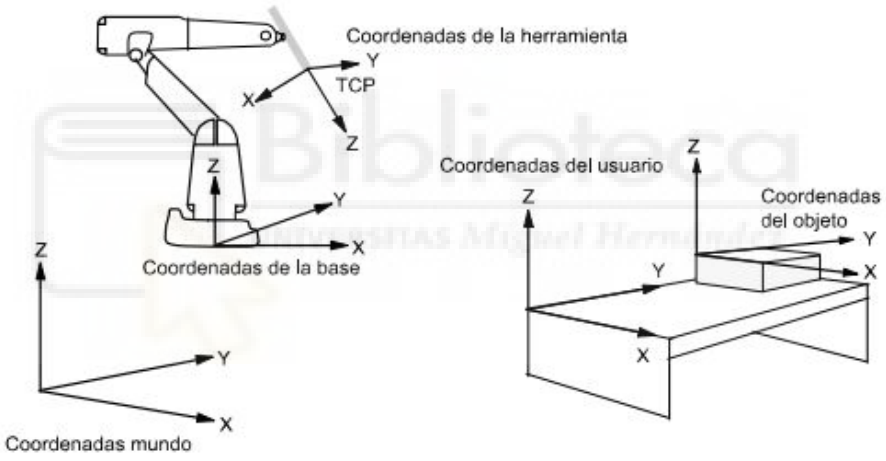


Figura 3.28: Sistemas de coordenadas del objeto y del usuario.

Este sería un ejemplo de objeto de trabajo coordinado en el que el objeto de trabajo del robot 1 sería movido por el robot "ROB_2":

PERS wobjdata wobj_rob1 := [FALSE, FALSE, "ROB_2", [[0,0,0], [1,0,0,0]], [[0,0,250], [1,0,0,0]]];

3.2.2.2. Movimientos semi-coordinados y coordinados

Una vez explicados los objetos de trabajo coordinados, pasamos a la segunda clave de los movimientos coordinados, las instrucciones que permitirán que los robots se esperen entre sí, que se muevan simultaneamente, etc.

Para empezar, es necesario diferenciar cuando utilizar cada tipo de movimiento. En ocasiones, sera necesario que un brazo espere a que termine el otro para realizar su parte de la tarea. Estos movimientos se denominan movimientos independientes, y en rapid se consiguen mediante funciones de espera, y entradas y salidas digitales normalmente.

Del mismo modo, se puede implementar una estación en la que dos robots los dos brazos del YuMi deban trabajar a la vez, pero no sea necesario que acaben los movimientos simultáneos al mismo tiempo. Esto sería necesario por ejemplo cuando los dos brazos trabajan para llenar un recipiente de objetos. Este tipo de movimientos se llaman movimientos semi-coordinados y se suelen realizar mediante la instrucción WaitSyncTask, que sirve para que dos o más robots tengan un punto de espera en un punto determinado y hasta que ambos no llegan no continúan con su siguiente labor.

El modo de movimientos coordinados o sincronizados se inician ejecutando una instrucción SyncMoveOn en cada programa de los robots sincronizado sy se finalizan ejecutando la instrucción SyncMoveOff. El número de instrucciones de movimiento ejecutadas entre SyncMoveOn y SyncMoveOff se realizan de modo que todos los robots que las están ejecutando tardan lo mismo en realizar los movimientos deseados, tardan lo que tarde el más lento. Este tipo de movimientos es útil cuando dos robots trabajan con una misma pieza o transportan una pieza grande entre varios, o en algunas situaciones diferentes más

Los movimientos sincronizados coordinados generalmente ahorran tiempo en los procesos ya que los robots no tienen que esperar mientras el otro hace otra tarea. También permite que los robots cooperen de formas que de otro modo serían difíciles o imposibles de lograr.

En las prácticas realizadas se combinarán los objetos de trabajo coordinados, los movimientos de tipo independiente, semi-coordinado y coordinado, junto con instrucciones de control de flujo y todo lo aprendido.

3.3. Creación y diseño de Smart Components

Los Componentes Inteligentes o Smart Components (SC) son elementos asociados a los sólidos, piezas o robots de la estación, los cuales tienen un comportamiento controlado por señales y propiedades del sistema. Solo sirven para estaciones virtuales.

Se pueden añadir en la pestaña “Modelado” y clicando en “Componente Inteligente” tal y como parece en la figura 3.29.

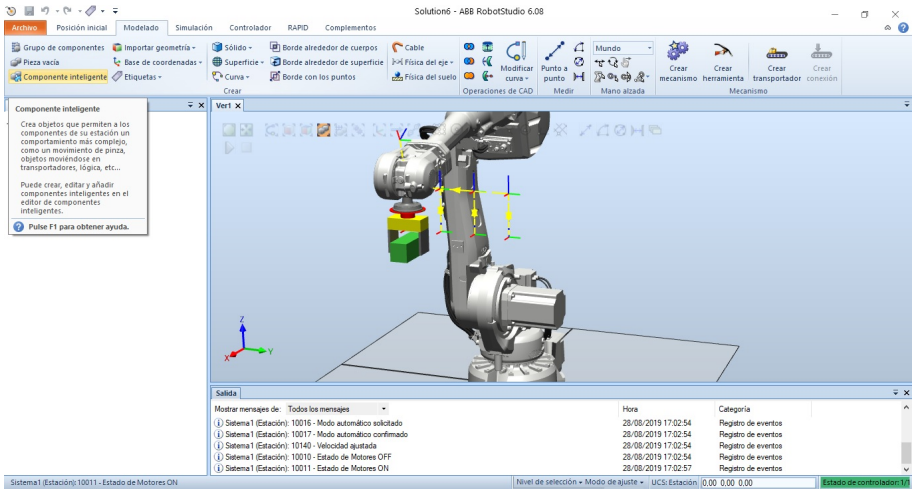


Figura 3.29: Crear componente inteligente (SC).

Los Smart Components pueden agruparse en 6 categorías diferentes: Señales y propiedades, primitivos paramétricos, sensores, acciones, manipuladores y otros.

Señales y propiedades: Dentro de esta categoría se pueden encontrar puertas lógicas, contadores, temporizadores, expresiones matemáticas, convertidores y demás elementos para modificar una señal o una propiedad del sistema a programar.

Primitivos paramétricos: En este apartado existen los componentes necesarios para crear de forma automática sólidos y líneas, así como para generar copias de componentes gráficos ya existentes.

Sensores: Está 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.

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

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

Otros: En esta última categoría se encuentra 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.

A continuación de muestra el esquema de un SC de una estación para tener un ejemplo real.

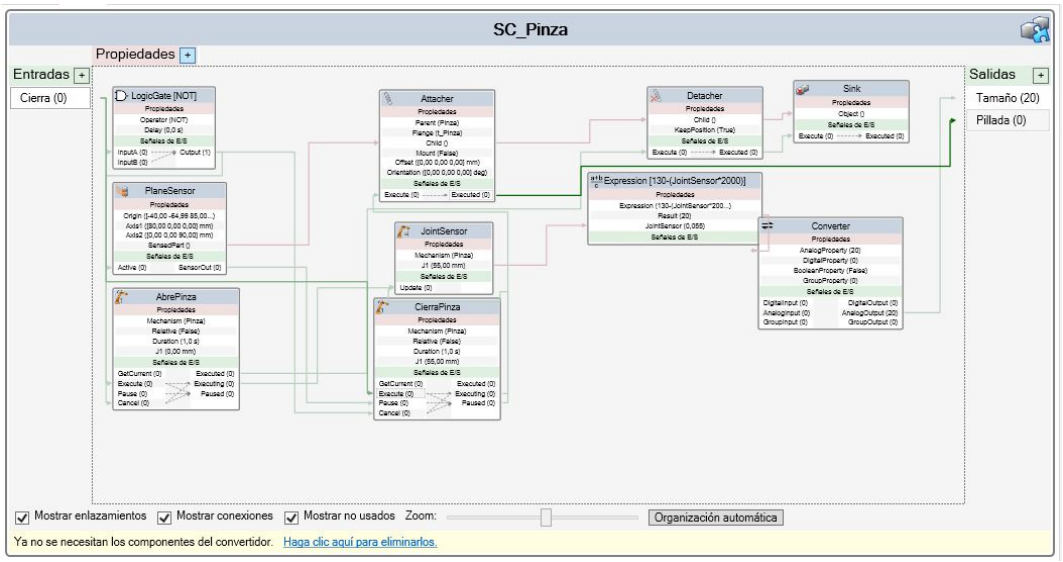


Figura 3.30: Diseño del SC de una pinza.

Se trata del esquema de una Pinza, se cierra cuando se activa la entrada Cierra, detecta cuando coge la pieza y la mueve, informa cuando coge la pieza y cuanto mide la misma mediante dos salidas. Además, cuando Cierra se pone a 0 suelta la pieza y la pinza se abre.

Tiene una puerta lógica de tipo NOT, cuando la entrada Cierra se pone a 0, la salida es 1, entonces cancela CierraPinza y activa AbrePinza. Tiene un sensor de en forma de plano pegado a la pared de una pinza (PlaneSensor) que informa al Attacher de que pieza hay que coger, y a CierraPinza de que pause el cierre de la pinza. CierraPinza y AbrePinza se encargan como su propio nombre indica de mover la articulación para coger y soltar las piezas, cundo las suelta, avisa a Sink, que elimina la pieza. Posee además un sensor de posición (JointSensor), que pasa el dato a Expression, que hace el cálculo de la apertura y la pasa a traves del Converter (lo transforma en una señal analógica) a la salida analógica Tamaño. La salida Pillada es avisada por Attacher cuando la pieza se coge.

Este tipo de componentes se utilizará en la primera de las aplicaciones que será mostrada en el Capítulo 4 del trabajo.

3.4. Visión artificial

La segunda aplicación que será explicada en el siguiente capítulo, se trata de una aplicación con el robot real, en esta se utiliza una cámara para detectar unas piezas para realizar una operación de Pick&Place. La programación de la cámara se ha realizado por medio de la interfaz que RobotStudio proporciona. En este punto se explicarán los pasos principales para la programación de la cámara.

En primer lugar hay que clicar el botón “Visión Integrada” en el menú “Controlador”. Se abrirá el menú de Visión, y en los desplegables de la izquierda aparecerán las cámaras que haya conectadas al robot. Se hace clic derecho en la que se desee programar y clic en conectar, Figura 3.31.

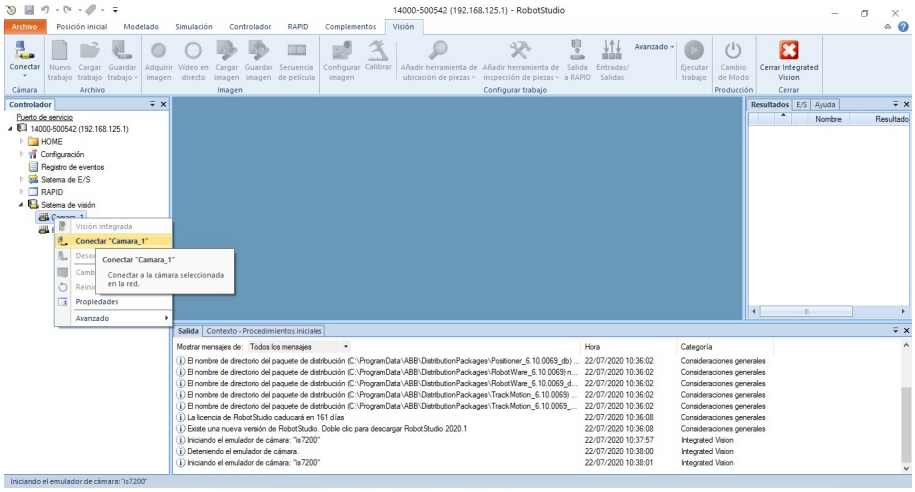
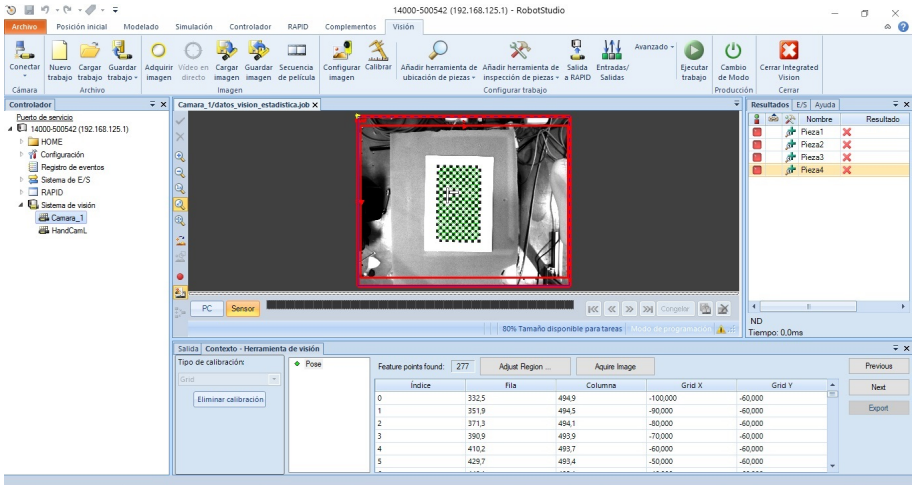


Figura 3.31: Menú de Visión.

A continuación se selecciona el botón “Cambio de modo”, para que la cámara cambie a modo programación y deje cambiar sus parámetros. Lo primero a partir de aquí es calibrar la cámara, para ello se coloca en el centro del espacio de trabajo aproximadamente el llamado damero o tablero de ajedrez, que se puede encontrar en la web de ABB. Se clica Calibración por medio de Grid y se toman varias imágenes del damero. En cada una de ellas hay que ajustar la región y finalmente se pulsa el botón calibrar, este da una estimación de lo buena que ha sido la calibración, hay que asegurarse de que sea al menos buena, Figura 3.32.



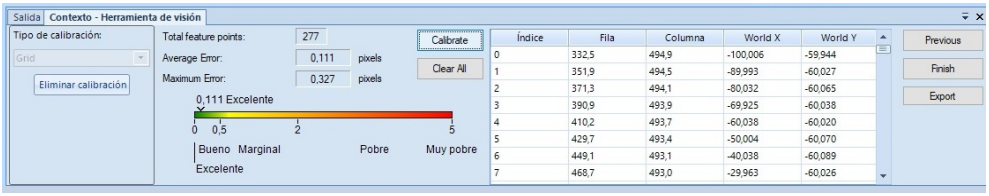


Figura 3.32: Calibración de la cámara.

Una vez calibrada la cámara, es necesario definir las piezas, para ello se toma una imagen con las piezas dentro de ella y se selecciona la herramienta para añadir ubicaciones de piezas en el menú superior. Dentro de estas, selecciona Patrón PatMax para cada una de las piezas, y se realiza un rectángulo a cada una. Esto guarda la silueta de cada pieza y hará que con cada imagen que se tome, el programa busque las piezas e informe de la posición y orientación en que se encuentran cada una de ellas. Se muestra un ejemplo en la Figura 3.33.

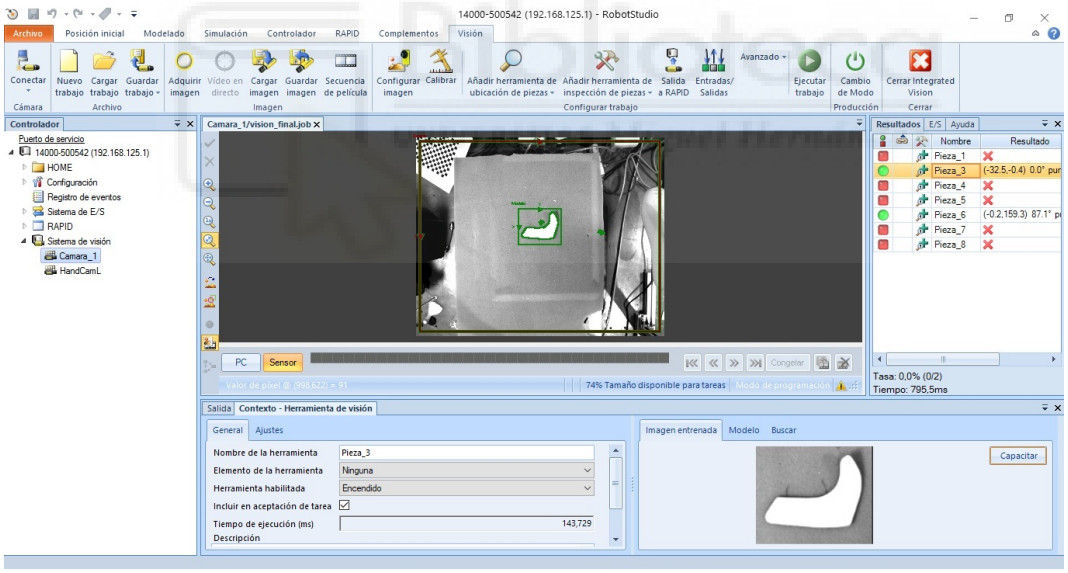


Figura 3.33: Patrón PatMax.

En la siguiente imagen se muestra como al meter seis piezas diferentes y tomar una imagen se detectan las seis. En la ventana de la derecha el programa informa de la posición y orientación en que se ha detectado cada pieza (Figura 3.34).

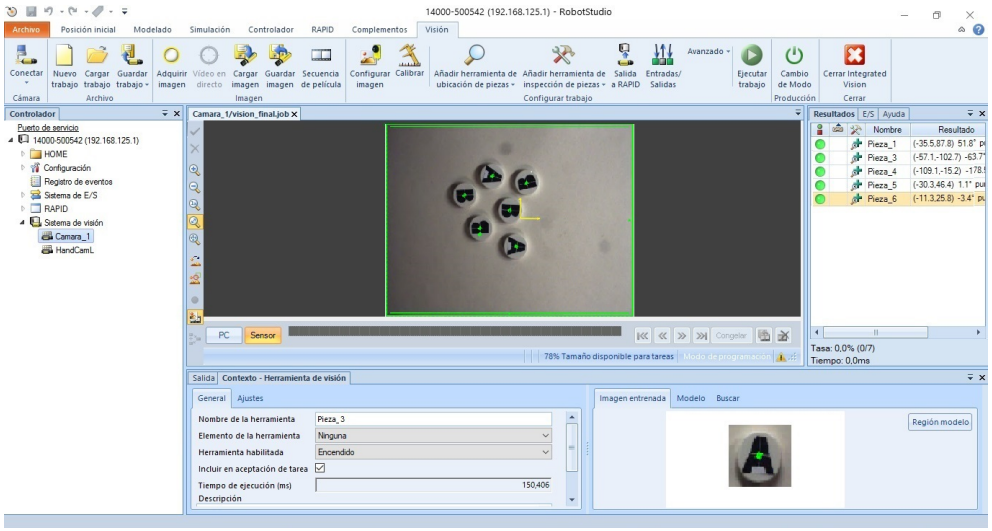


Figura 3.34: Detección de varias piezas.

Finalmente, para que los datos (X, Y y orientación) cuando se toma cada imagen pasen al programa de RAPID, es necesario clicar en el botón “Salida a RAPID” del menú superior, y especificarlo en el menu inferior que aparece como en la Figura 3.35, hay que hacerlo con todas las piezas.

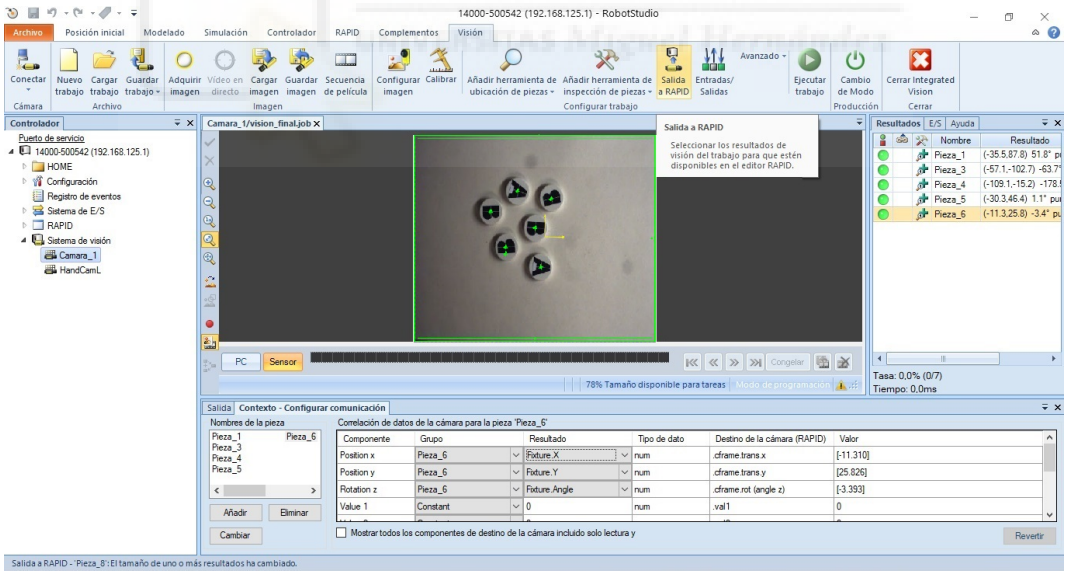


Figura 3.35: Datos de salida a RAPID de visión.

Esta sería toda la parte de Visión en la interfaz, la parte de rapid se mostrará y explicará en el siguiente capítulo junto con el funcionamiento del programa en la segunda de las aplicaciones, que es en la que se usa la visión artificial.

4. Resultados

El trabajo partía del aprendizaje básico de las funciones básicas de RobotStudio, que fue lo que dió tiempo durante el TFG. En esta ocasión se ha profundizado más y se ha alcanzado un nivel más alto, sobre todo en la segunda aplicación. El alumno se ha involucrado en la programación de la visión artificial de la estación y ha aprendido a implementar una estación más compleja. Por ejemplo, en esta ocasión la herramienta del robot tiene una función neumática de succión que se activa en los momentos precisos mediante entradas y salidas del programa.

Ahora que se conoce el funcionamiento del programa y las funciones que tiene,se pasa a explicar las aplicaciones desarrolladas.

Las prácticas se llevaron a cabo en el laboratorio de la universidad empleando la simulación del programa RobotStudio, y en una empresa distribuidora de ABB donde se pudieron probar las simulaciones con el robot real.

El procedimiento que se seguirá sera similar ambos casos, se explicará el objetivo, la estación, las partes importantes de la programación y los resultados.

4.1. Aplicación 1: Estudio de alcance, colisión y distancia entre brazos

Durante el Trabajo de Fin de Grado se realizaron algunas operaciones de Pick&Place desde una bandeja con el mismo robot. Uno de los problemas que surgió fue que el robot no tiene un control durante los movimientos de cuando los brazos van a colisionar, únicamente lo detecta unos instantes antes de que esto ocurra y hace que la tarea se detenga por completo. Otro de los problemas era que muchas posiciones a las que se le mandaba no eran alcanzables por singularidades.

De este problema surgió la idea de esta aplicación, realizar un estudio de en que posiciones los brazos van a detectar colisión, de cuan cerca pueden estar sin que se detecte y salte la alarma, y del alcance que tienen los brazos.

Esta aplicación se trata de un programa en el que los dos brazos recorrerán una gradilla de 5x10 posiciones, y en cada una de las combinaciones, anotarán si cada brazo alcanza la posición deseada, si se detecta la alarma de colisión entre los brazos, y finalmente, si no hay colisión, a que distancia queda un brazo de otro. Puesto que la gradilla tiene 50 posiciones posibles, y hay dos brazos, las combinaciones posibles son de 50 elevado a 2, es decir, 2500 combinaciones. El programa se ejecutó una vez a

velocidad acelerada para recoger los datos, que posteriormente podrán ser estudiados para evitar problemas de colisión y no alcance.

La gradilla escogida va desde -450 mm a 450 mm en el eje Y, y de 200 mm a 600 mm en el eje X, con incrementos de 100 mm en ambos ejes para conseguir la gradilla de 5x10. A continuación se muestra un boceto del robot y la gradilla, Figura 4.1.

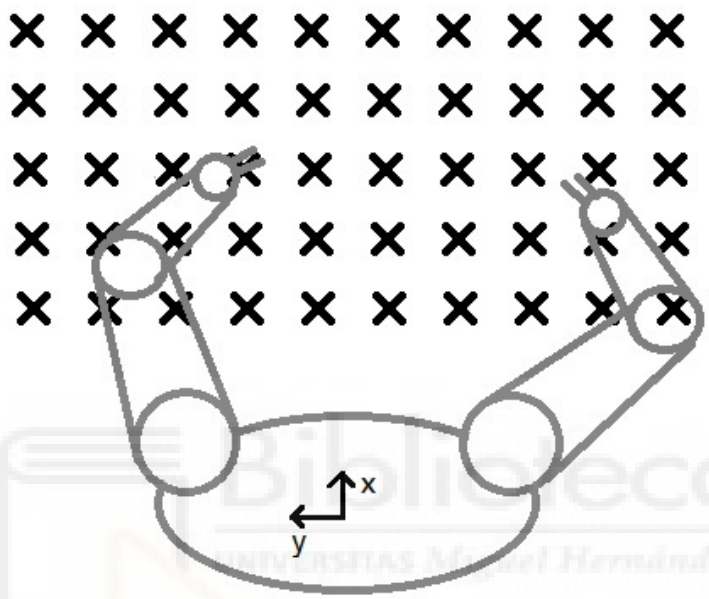


Figura 4.1: Boceto de la gradilla.

Se muestran a continuación algunas filas de la tabla final como debería quedar, Figura 4.2.

	A	B	C	D	E	F	G	H	I	J
1	Xl	Yl	Xr	Yr	Al	Ar	A	C	Ok	Distancia(mm)
2	200	-450	200	-450	0	0	0	1	0	29,892
3	200	-450	200	-350	0	1	0	1	0	29,6789
4	200	-450	200	-250	0	1	0	1	0	30,014
5	200	-450	200	-150	0	1	0	1	0	29,6488
6	200	-450	200	-50	0	1	0	1	0	29,6818

Figura 4.2: Tabla para estudio posterior.

Se explica a continuación el significado de las siglas de cada fila:

Xl Yl: Coordenadas de la bandeja a las que va el brazo izquierdo.

Xr Yr: Coordenadas de la bandeja a las que va el brazo derecho.

Al: El brazo izquierdo alcanza la posición escogida.

Ar: El brazo derecho alcanza la posición escogida.

A: Ambos brazos alcanzan la posición escogida.

C: Este parámetro se pone a 0 si hay colisión entre los brazos y a 1 si no la hay.

Ok: Este parámetro se pone a 1 si tanto A como C están a 1, y a 0 en cualquier otro caso. Quiere decir que si está a 1 ambos brazos alcanzan la posición deseada y no hay colisión entre los brazos.

Distancia (mm): Indica la distancia a la que quedan los puntos más cercanos de los brazos para saber a que distancia quedan en cada posición en milímetros. Si esta distancia es menor a 10 mm el detector de colisión da positivo por el margen de error que pueda tener el robot.

El funcionamiento se basa en un programa con forma de bucle. Las posiciones a las que tiene que ir cada brazo están guardadas en un fichero de texto, en cada iteración el programa lee las posiciones a las que debe ir cada uno, a continuación va,y finalmente, lee la información de los sensores de alcance, colisión y distancia entre los brazos, y escribe los datos en otro fichero. Este proceso se repite en bucle hasta que se acaban los datos. El funcionamiento es sencillo, aunque no tanto de implementar.

A continuación se muestra el código de ambos brazos. ambos leen las posiciones que les toca en cada paso, por ello en el código de ambos brazos leen del mismo fichero, pero tan solo es el derecho el que escribe los resultados en el fichero de resultados.

```
MODULE Module1 !Brazo izquierdo


    PROC main()
        Open rdirectorio, rfile \Read;!Fichero de lectura

        FOR i FROM 1 TO 2500 DO!Bucle
            WaitSyncTask sync3, task_list;
            MoveJ Target_01,v1000,z100,Servo\WObj:=wobj0;!Punto inicial
            Leer;
            GeneraPosicion;
            WaitTime 1;
            WaitSyncTask sync2, task_list;
            IF nl_1_L_out=0 THEN
                MoveAbsJ auxjoint_L,v500,z100,Servo\WObj:=wobj0;!Mueve al punto deseado
                WaitRob \ZeroSpeed;
            ENDIF
            WaitSyncTask sync1, task_list;
        ENDFOR
        Close rfile;
    ENDPROC

    PROC Leer()!Leer el fichero
        x:=ReadNum(rfile\Delim:="\09");
        y:=ReadNum(rfile\Delim:="\09");
        com:=ReadNum(rfile\Delim:="\09");
        com:=ReadNum(rfile\Delim:="\09");
    ENDPROC

    PROC GeneraPosicion()!Sirve para detectar cuando el brazo izquierdo no alcanza la posición deseada
        Target_10:=[[x,y,180],[6.962025E-08,-0.8660255,-5.300921E-08,-0.4999998],[0,-1,0,4],[180,9E+09,9E+09,9E+09,9E+09,9E+09]];
        auxjoint_L:=CalcJointT(Target_10,Servo);
        WaitTime 0.5;

         ERROR
        IF ERRNO=ERR_OUTSIDE_REACH OR ERRNO=ERR_ROBLIMIT THEN
            SetDO nl_1_L_out,1;
            TRYNEXT;
        ENDIF
    ENDPROC

ENDMODULE
```

```
MODULE Module1 !Brazo derecho


    PROC main()
        Open wdirectorio, wfile \Append;!Fichero de escritura
        Open rdirectorio, rfile\Read;!Fichero de lectura

        FOR i FROM 1 TO 2500 DO!Bucle
            WaitSyncTask sync3, task_list;!Punto de sincronización
            MoveJ Target_0r,v1000,z100,Servo\WObj:=wobj0; !Punto inicial
            Leer;
            GeneraPosicion;
            WaitTime 1;
            WaitSyncTask sync2, task_list;
            IF nl_1_R_out=0 THEN
                MoveAbsJ auxjoint_R,v500,z100,Servo\WObj:=wobj0;!Mueve al punto deseado
                WaitRob \ZeroSpeed;
            ENDIF
            WaitSyncTask sync1, task_list;
            EscribirCoord;

        ENDFOR


        Close wfile;
        Close rfile;
    ENDPROC

    PROC Leer()!Leer el fichero
        xl:=ReadNum(rfile\Delim:="\09");
        yl:=ReadNum(rfile\Delim:="\09");
        x:=ReadNum(rfile\Delim:="\09");
        y:=ReadNum(rfile\Delim:="\09");
    ENDPROC

    PROC EscribirCoord() !Escribe los datos de la tabla en cada iteración teniendo en cuenta los criterios antes mencionados
        WaitRob \ZeroSpeed;
        SetDO Mide,1; !Activa el sensor de colisión y de distancia para recoger los datos
        WaitTime 0.2;
        SetDO Mide,0;
        WaitTime 0.1;
        Write wfile, "" \Num:=xl \NoNewLine;
        Write wfile, "\09"\NoNewLine;
        Write wfile, "" \Num:=yl \NoNewLine;
        Write wfile, "\09"\NoNewLine;
        Write wfile, "" \Num:=x \NoNewLine;
        Write wfile, "\09"\NoNewLine;
        Write wfile, "" \Num:=y \NoNewLine;
        Write wfile, "\09"\NoNewLine;
        IF nl_1_L_in=0 THEN
```

```
            Write wfile, "1" \NoNewLine;
            Write wfile, "\09"\NoNewLine;
        ELSE
            Write wfile, "0" \NoNewLine;
            Write wfile, "\09"\NoNewLine;
        ENDIF
        IF nl_1_R_in=0 THEN
            Write wfile, "1" \NoNewLine;
            Write wfile, "\09"\NoNewLine;
        ELSE
            Write wfile, "0" \NoNewLine;
            Write wfile, "\09"\NoNewLine;
        ENDIF
        IF nl_1_L_in=0 AND nl_1_R_in=0 THEN
            Write wfile, "1" \NoNewLine;
            Write wfile, "\09"\NoNewLine;
            A:=1;
        ELSE
            Write wfile, "0" \NoNewLine;
            Write wfile, "\09"\NoNewLine;
            A:=0;
        ENDIF
        IF colision=1 THEN
            Write wfile, "0" \NoNewLine;
            Write wfile, "\09"\NoNewLine;
            col:=0;
        ELSE
            Write wfile, "1" \NoNewLine;
            Write wfile, "\09"\NoNewLine;
            col:=1;
        ENDIF
        IF A=1 AND col=1 THEN
            Write wfile, "1"\NoNewLine;
            Write wfile, "\09"\NoNewLine;
        ELSE
            Write wfile, "0"\NoNewLine;
            Write wfile, "\09"\NoNewLine;
        ENDIF
        ComparaDistancia;
        Write wfile, "" \Num:=distancia*1000;

        WaitTime 0.5;
        SetDO nl_1_R_out,0;
        SetDO nl_1_L_out,0;
        WaitTime 0.5;
    ENDPROC

    PROC GeneraPosicion()!Sirve para detectar cuando el brazo izquierdo no alcanza la posición deseada
        Target_10:=[[x,y,180.0001],[1.158005E-07,-0.8660253,-2.182211E-07,-0.5000002],[0,1,0,4],[180,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
        auxjoint_R:=CalcJointT(Target_10,Servo);
        WaitTime 0.5;
         ERROR
        IF ERRNO=ERR_OUTSIDE_REACH OR ERRNO=ERR_ROBLIMIT THEN
            SetDO nl_1_R_out,1;
            TRYNEXT;
        ENDIF
    ENDPROC

    PROC ComparaDistancia() !Coge la menor distancia de todas las partes de ambos brazos robot
        IF distancia1<distancia2 THEN
            distancia:=distancia1;
        ELSE
            distancia:=distancia2;
        ENDIF
        IF distancia3<distancia THEN
            distancia:=distancia3;
        ENDIF
        IF distancia4<distancia THEN
            distancia:=distancia4;
        ENDIF
    ENDPROC


ENDMODULE
```

Para terminar de comprender el funcionamiento del programa, falta mostrar los componentes inteligentes llamados Sensor de Colisión y Sensor de Distancia que se activan en la rutina Escribir Coord cuando se activa la salida digital Mide. Estos sensores sirven para medir lo que indican sus nombres respectivamente.

Como ya se ha mencionado, determinadas posiciones pueden producir que los brazos colisionen o queden muy cerca de hacerlo, y cuando esto ocurre en RobotStudio con su configuración normal, se produce un error por colisión y el programa se detiene, y se trata de un error no recuperable por lo que no hay manera de continuar el programa si no es de forma manual. Para el programa es imprescindible que no se bloquee nunca, porque si hay 2500 simulaciones que tardarían un par de días y se para en la simulación 5 sería un gran problema, además de que es un dato importante para la tabla. Por ello, es necesario desactivar la "Evitación de colisión" del propio programa (en la ventana "Controlador"), y crear este componente inteligente que informa en caso de colisión.

Aquí se muestra su esquema interno, como se observa tiene cuatro detectores de colisión, ya que cada uno controla la colisión entre dos objetos. Las combinaciones son las siguientes: Brazo izquierdo-Brazo derecho, Pinza izquierda-Pinza derecha, Brazo izquierdo-Pinza derecha y Brazo derecho-Pinza izquierda. También tiene tres puertas OR, para que si cualquiera de los cruces anteriores se da se informe. Al final tiene un bloque de variable RAPID que es el modo de pasar los datos del sensor al programa en cada repetición. Obsérvese la Figura 4.3

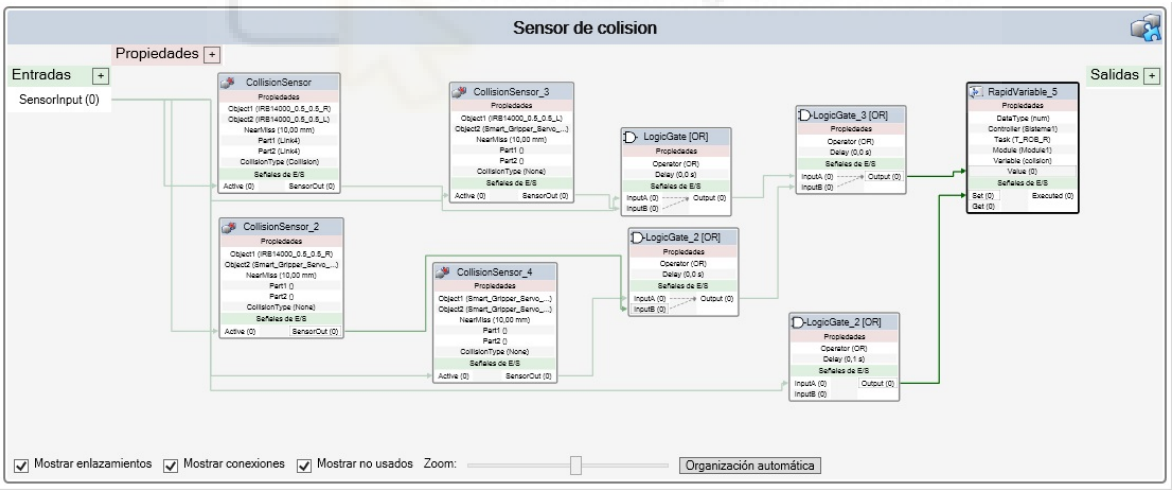


Figura 4.3: Esquema interno del Sensor de Colisión.

Para acabar, se muestra el esquema interno del Sensor de distancia. Funciona de forma similar al anterior, pero los componentes miden distancias mínimas entre objetos a pares, y se realizan cuatro combinaciones: Brazo izquierdo-Brazo derecho, Pinza izquierda-Pinza derecha, Brazo izquierdo-Pinza derecha y Brazo derecho-Pinza izquier-

da. A continuación se pasan los cuatro valores a RAPID y en RAPID se escoge el mínimo de estos.

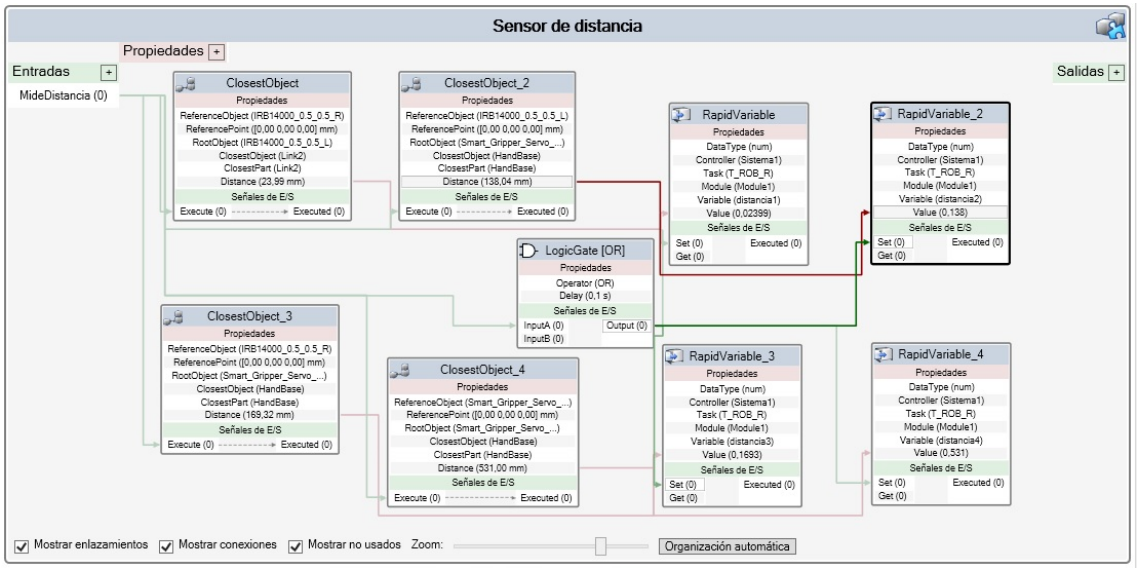


Figura 4.4: Esquema interno del Sensor de Distancia.

El resultado de esta aplicación es la tabla completa, a la espera del estudio estadístico o analítico de los datos, el resultado es positivo.

4.2. Aplicación 2: Tarea de Pick&Place con Visión Integrada

Llegados a este punto, se hizo la propuesta de hacer unas demostraciones para una exposición para la empresa del proyecto. Concretamente, la empresa desea que el robot detecte y coloque una serie piezas desde una superficie plana hasta una tridimensional en el menor tiempo posible, y con el mínimo margen de error, para el posterior proceso de fabricación de un producto en cadena. La superficie a la que han de ir pegadas las piezas puede ser cóncava o convexa y el sistema se ha de adaptar a ambas circunstancias en la medida de lo posible.

La idea era mostrar como el robot toma una imagen a través de la cámara de la disposición de las piezas, y después las coge y las coloca en su destino. Las piezas pueden variar su lugar en la bandeja y el robot las detectara e irá a por ellas. Se consideró que era algo interesante de cara al proceso de fabricación y a la demostración.

A continuación se muestra el programa de rapid de la parte de visión que se encarga de configurar la cámara, recoger la imagen y pasar los datos deseados al programa principal para que el robot sepa las posiciones de las piezas, tiene pequeñas explicaciones de para que sirve cada parte.

```
MODULE mainModule
    VAR num salir:=1;
    VAR num piezas:=1;
    ! Variables cámara:
    CONST string myjob:="vision_final.job";
    CONST string myjob2:="Vision_place.job";
    !nombre del job
    VAR cameratarget mycameratarget;
    VAR cameratarget datos_place;
    !variable donde se guarda la salida del job
    VAR num cont:=0;
    VAR num resultado:=0;

    PROC main()
        IF cont<1 then
            CargaJob;
        ENDIF
        Reset Fallo_vision;
        SetDO Piezas_listas,1;
        Reset Vision_Completa;
        ! WaitDO Comunicacion_establecida,1;
        WaitSyncTask sync27,task_list_brazos_vision;

        IF cont>=1 then
            UIMsgBox "Continue the program ?"\Icon:=iconInfo;
            !WaitDI boton, 1, \Header:="Continue the program ?";
        ENDIF

        LocalizaPiezas;
        WaitSyncTask sync15,task_list_brazos_vision;
        cont:=cont+1;
    ENDPROC

    PROC CargaJob()
        var string jobCargado;
        var string job2Cargado;
        CamSetProgramMode Camara_1;
        CamSetProgramMode HandCamL;
        !Pone la cámara en modo programación. "camara" es el nombre de la cámara

        jobCargado:=CamGetLoadedJob(Camara_1);
        job2Cargado:=CamGetLoadedJob(HandCamL);
        IF jobCargado=myjob THEN
            !Nada
        ELSE
            CamLoadJob Camara_1,myjob;
        ENDIF

        !Se carga el job en la camara
        CamSetRunMode Camara_1;
        CamSetRunMode HandCamL;
```

```
    !Pone la cámara en modo ejecución
endProc

PROC LocalizaPiezas()
    CamReqImage Camara_1;
    waitTime 1;

    SetDO Piezas_listas,1;
    Reset Fallo_vision;
    CamGetResult Camara_1,mycameratarget;
    TPWrite "Posición "+mycameratarget.name+" "\Pos:=mycameratarget.cframe.trans;
    TPWrite "Rotación "+mycameratarget.name+" "\Num:=EulerZYX(\Z,mycameratarget.cframe.rot);
    !Escribe el ángulo en grados entre -180º y +180º
    Pieza_1{1}:=mycameratarget.cframe.trans.x;
    Pieza_1{2}:=mycameratarget.cframe.trans.y;
    Pieza_1{3}:=EulerZYX(\Z,mycameratarget.cframe.rot);
    CamGetResult Camara_1,mycameratarget;
    !Guarda la captura en mycameratarget
    TPWrite "Posición "+mycameratarget.name+" "\Pos:=mycameratarget.cframe.trans;
    TPWrite "Rotación "+mycameratarget.name+" "\Num:=EulerZYX(\Z,mycameratarget.cframe.rot);
    !Escribe el ángulo en grados entre -180º y +180º
    Pieza_3{1}:=mycameratarget.cframe.trans.x;
    Pieza_3{2}:=mycameratarget.cframe.trans.y;
    Pieza_3{3}:=EulerZYX(\Z,mycameratarget.cframe.rot);
    CamGetResult Camara_1,mycameratarget;
    !Guarda la captura en mycameratarget
    TPWrite "Posición "+mycameratarget.name+" "\Pos:=mycameratarget.cframe.trans;
    TPWrite "Rotación "+mycameratarget.name+" "\Num:=EulerZYX(\Z,mycameratarget.cframe.rot);
    !Escribe el ángulo en grados entre -180º y +180º
    Pieza_4{1}:=mycameratarget.cframe.trans.x;
    Pieza_4{2}:=mycameratarget.cframe.trans.y;
    Pieza_4{3}:=EulerZYX(\Z,mycameratarget.cframe.rot);
    CamGetResult Camara_1,mycameratarget;
    !Guarda la captura en mycameratarget
    TPWrite "Posición "+mycameratarget.name+" "\Pos:=mycameratarget.cframe.trans;
    TPWrite "Rotación "+mycameratarget.name+" "\Num:=EulerZYX(\Z,mycameratarget.cframe.rot);
    !Escribe el ángulo en grados entre -180º y +180º
    Pieza_5{1}:=mycameratarget.cframe.trans.x;
    Pieza_5{2}:=mycameratarget.cframe.trans.y;
    Pieza_5{3}:=EulerZYX(\Z,mycameratarget.cframe.rot);
    CamGetResult Camara_1,mycameratarget;
    !Guarda la captura en mycameratarget
    TPWrite "Posición "+mycameratarget.name+" "\Pos:=mycameratarget.cframe.trans;
    TPWrite "Rotación "+mycameratarget.name+" "\Num:=EulerZYX(\Z,mycameratarget.cframe.rot);
    !Escribe el ángulo en grados entre -180º y +180º
    Pieza_6{1}:=mycameratarget.cframe.trans.x;
    Pieza_6{2}:=mycameratarget.cframe.trans.y;
    Pieza_6{3}:=EulerZYX(\Z,mycameratarget.cframe.rot);

    TPWrite ""\Num:=CamNumberOfResults(Camara_1);
ENDPROC
```

Ahora para mover la herramienta a la posición deseada solo queda añadir un offset a la posición dada por visión. Para ello se hace uso de la instrucción explicada en la metodología Offs. Para darle la orientación deseada a la herramienta se utiliza la instrucción RelTool, que permite reorientar la herramienta respecto del TCP de la misma a la orientación que tenga cada pieza.

Los puntos intermedios fueron escogidos y perfeccionados moviendo el robot con las manos y cogiendo los puntos, gracias a sus funciones colaborativas fue bastante cómodo. El código completo con la declaración de todos los puntos está disponible en los Anexos.

Como ya se ha mencionado, la superficie de dejada es complicada para adherir las piezas. Por eso fue necesario el uso de herramientas algo especiales, unas ventosas flexibles que por el otro lado disponen de pinzas. La tarea se realizó con 7 piezas con diferentes formas y tamaño, dependiendo de estos factores la herramienta utilizada era la ventosa o la pinza, y en todos los casos se utiliza la parte flexible para apretar las piezas a la superficie y que queden bien sujetas.

También es importante mencionar como se activa la succión a través de las ventosas. La estación estaba conectada a la red de presión neumática de una nave, que se conectó a una electro-válvula que alimenta las herramientas de ambos brazos, la electro-válvula es controlada por las salidas digitales del robot que se controlan en RAPID. Entre las salidas del robot y la electro-válvula hay unos relés, ya que no funcionan al mismo voltaje. Finalmente, para transformar el aire a presión en succión, cada brazo tiene un dispositivo de Venturi. Se muestra todo el equipamiento de la estación a continuación, Figura 4.6.

El programa completo de uno de los brazos se encuentra en los anexos, ya que el otro es similar, su funcionamiento se basa en las instrucciones ya explicadas en la Metodología. En la Figura 4.5 se muestra un diagrama general del funcionamiento que podría tener el programa final, con un control de calidad en el destino de las piezas.

Una vez explicado todo, llegamos a un resultado satisfactorio, el robot realiza la tarea correctamente aunque aún quedan algunas mejoras que hacer, el objetivo es afinar los puntos de dejada y los movimientos intermedios para hacer una demostración limpia y eficiente para la exposición. Decir también que con esta práctica el alumno ha aprendido mucho y fue muy interesante.En la exposición del trabajo se mostrarán algunos vídeos con el resultado actual y se explicará desde un punto de vista más personal.

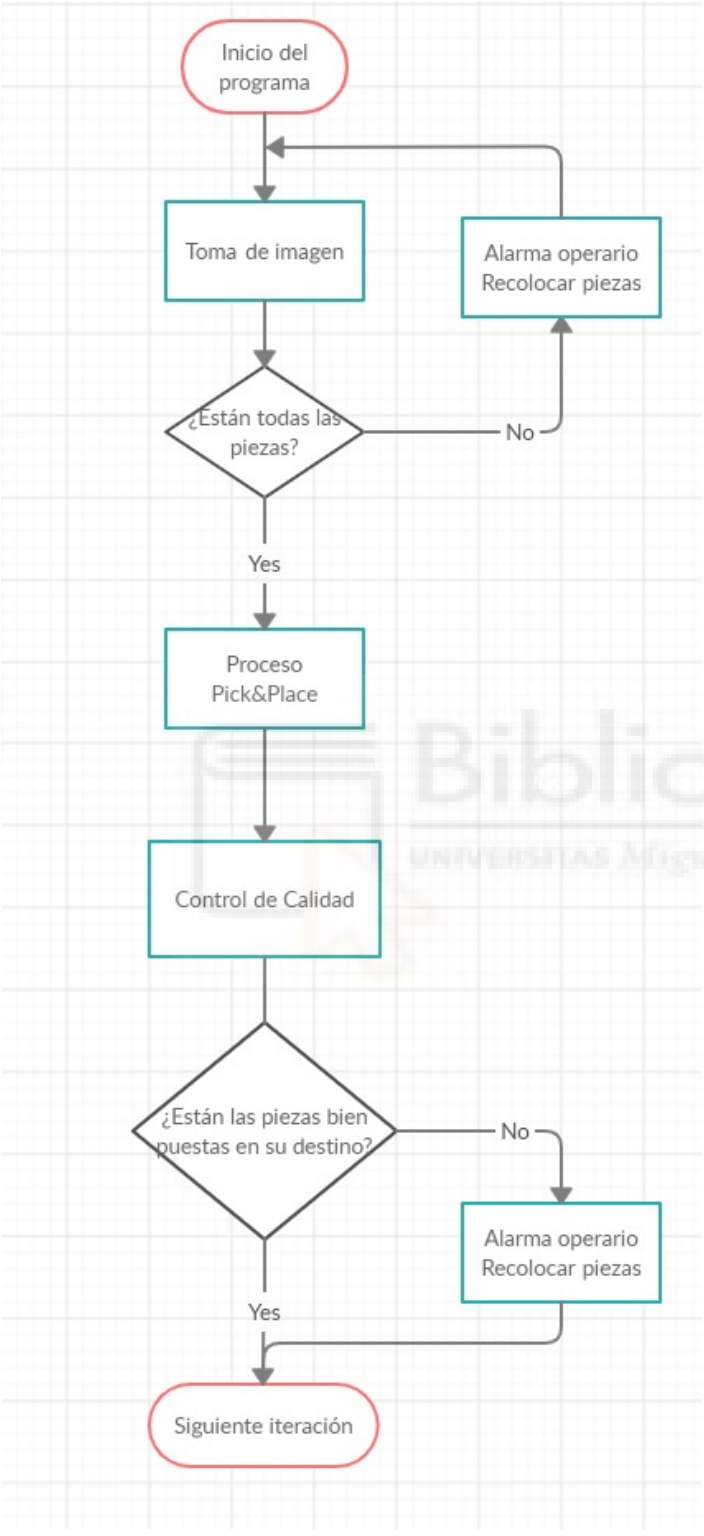
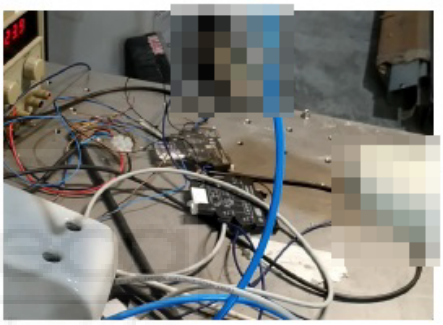


Figura 4.5: Diagrama de funcionamiento del programa.

(a) Cámara y focos.



(b) Relés.

(c) Electro-válvula.

(d) Venturi.

Figura 4.6: Equipamiento estación real.

5. Conclusiones y futuros proyectos

Como ya se ha mencionado, considero que los resultados de ambas prácticas han sido positivos. El resultado de las dos puede servir de ayuda para el proyecto, y al mismo tiempo, que aún hay margen de mejora para lograr resultados más eficientes.

He profundizado en el manejo del programa, la cual fue desde el principio la motivación principal,ahora se usar la visión artificial, y puedo ser un profesional más completo para un trabajo relacionado.

Considero también que la empresa puede sacar partido de las investigaciones en las que he colaborado, y que sería una buena inversión para ellos continuar el proyecto.

Por último, se ha trabajado con el robot real, y se han obtenido unas demostraciones muy interesantes tanto para la exposición como para el Trabajo de Fin de Máster.

Como futuras lineas de trabajo, hay muchas, algunas están ya en marcha:

- Considero que aún se le puede sacar más partido a la programación y que me falta mucho por aprender.
- He aprendido como se trabaja con dos o más robots, pero tan solo he probado dos en la misma estación, hay estaciones mucho más complejas, y esta podría serlo.
- La demostración real todavía necesita bastantes mejoras, las cuales también están en marcha ahora mismo.
- Incorporar a la estación real algunos movimientos para que la cámara del brazo tome imágenes de las piezas en el destino después de dejarlas.
- Preparar el programa para que se adapte a diferentes escenarios.
- Algo en lo que también están trabajando ya en el departamento es en incorporar un control de calidad de las piezas al inicio y final de cada demostración, con una interfaz que informa de si falta alguna pieza o alguna está mal puesta. Esto es interesante de cara al operario, puesto que permitiría que un operario pudiera participar en varias lineas de montaje a la vez, ya que solo tendría que solucionar algunos fallos poco frecuentes.

Bibliografía

ABB. (2004). System data types and routines on-line. En *Rapid reference manual.*

ABB. (2007a). Descripción general de rapid. En *Manual de referencia técnical.*

ABB. (2007b). Introduction to rapid. En *Operating manual.*

ABB. (2010). Instrucciones, funciones y tipos de datos de rapid. En *Manual de referencia técnica.*

ABB. (2017). Multimove. En *Aplication manual.*

Agustín, R. H. (2016). Diseño, programación y simulación de estaciones robotizadas industriales con robotstudio. En *Trabajo fin de grado.* ETSI.

Beatriz, M. A. (2017). Modelado, programación y simulación del robot irb 120 de abb con robotstudio. En *Proyecto de fin de máster.* ETSI.

Cristian, Q. G. (2016-2017). Programación de robot abb para tareas de reparación de defectos en carrocerías de automóvil. En *Trabajo de fin de máster.* UMH.

Elizabet, Z. (2017-2018). Desarrollo de aplicaciones mediante robots colaborativos basadas en interfaces naturales hombre-máquina. En *Trabajo fin de grado.* UPV.

Francisco, S. R. (2018-2019). Implementación de una solución de robótica colaborativa para fabricación de interiores en el sector del automóvil. desarrollo estratégico de su estándar global. En *Trabajo de fin de máster.* UPV.

Jorge, B. M. (2015). Programación de un robot social. En *Trabajo de fin de grado.* UMH.

Jorge, B. M. (2017). Uso de semiconductores de sic y diodo ideal en sistemas de regulación: S3rl. En *Trabajo de fin de máster.* UMH.

A. Datasheet del YuMi

ROBOTICS

YuMi® creating an automated future together.

You and me.

YuMi is the first truly collaborative dual armed robot, designed for a world in which humans and robots work together. It heralds a new era of robotic co-workers which are able to work side-by-side on the same tasks as humans with extreme accuracy while ensuring the safety of those around it.

Collaboration
YuMi is designed to meet the flexible and agile production needs required for small parts assembly in the electronics industry. It is also well suited to other small parts environments, including the manufacture of watches, toys and automotive components. All of this thanks to its dual-arms, flexible hands, universal parts feeding system, camera-based part location and state-of-the-art motion control.

Redefining safety
YuMi has a lightweight yet rigid magnesium skeleton covered with a floating plastic casing wrapped in soft padding, which absorbs the force of any unexpected impacts to a very high degree. YuMi has no pinch points so that sensitive ancillary parts cannot be crushed between two opposing surfaces as the axes open and close.

If YuMi senses an unexpected impact or change in its environment such as a collision with a coworker, it can pause its motion within milliseconds to prevent injury, and the motion can be restarted again as easily as pressing play on a remote control.

YuMi is very precise and fast, returning to the same point in space over and over again to within 0.02 mm accuracy and moving at a maximum velocity of 1,500 mm/sec. This ensures the safety of human co-workers on production lines and in fabricating cells.

Total solution concept
ABB also develops software and manufactures hardware, peripheral equipment, process equipment and modular manufacturing cells. This “total solution” concept is evident in YuMi’s breakthrough design.

Features
- The fifth-generation, integrated IRC5 controller with TrueMove and QuickMove™ motion control technology commands accuracy, speed, cycle-time, programmability and synchronization with external devices.
- I/O interfaces include Ethernet IP, Profibus, USB ports, DeviceNet™, communication port, emergency stop and air-to-hands. YuMi accepts a wide range of HMI devices including ABB’s teach pendant, industrial displays and commercially available tablets.
- The 100-240 volt power supply plugs into any power socket for worldwide versatility.

Benefits
- Can operate equally effectively side-by-side or face-to-face with human coworkers.
- Servo grippers (the “hands”) include options for built-in cameras.
- Real-time algorithms set a collision-free path for each arm customized for the required task.
- Padding protects coworkers in high-risk areas by absorbing force if contact is made.

Specification

Robot version	Reach (mm)	Payload (g)	Armload
IRB 14000-0.5/0.5	559	500	No armloads
Number of axes	14		
Protection	Std: IP30 and Clean Room		
Mounting	Table		
Controller	Integrated		
Integrated signal and power supply	24V Ethernet or 4 Signals		
Integrated air supply	1 per Arm on tool Flange (4 Bar)		
Integrated ethernet	One 100/10 Base-TX ethernet port/per arm		

Performance (according to ISO 9283)

IRB 14000-0.5/0.5	
0.5 kg picking cycle	
25* 300 * 25 mm	0.86s
Max TCP Velocity	1.5 m/s
Max TCP Acceleration	11 m/s*s
Acceleration time 0-1m/s	0.12s
Position repeatability	0.02 mm

Technical information

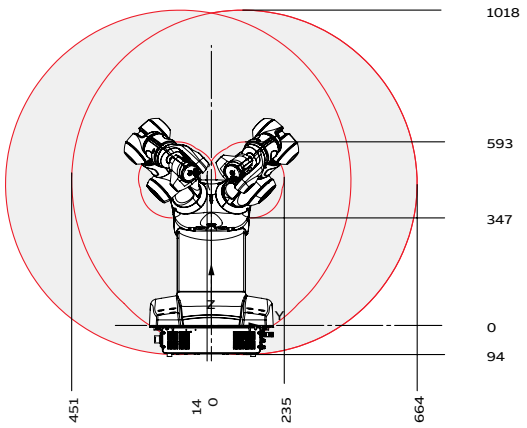
Physical	
Robot base	399 x 496 mm
Robot toes	399 x 134 mm
Weight	38 kg
Environment	
Ambient temperature for mechanical unit	
During operation	+5°C i (41°F) to +40°C (104°F)
During transportation and storage	-10°C (14°F) to +55°C (131°F)
Relative humidity	Max. 85%
Noise level	< 70 dB
Safety	PL b Cat B

Data and dimensions may be changed without notice.

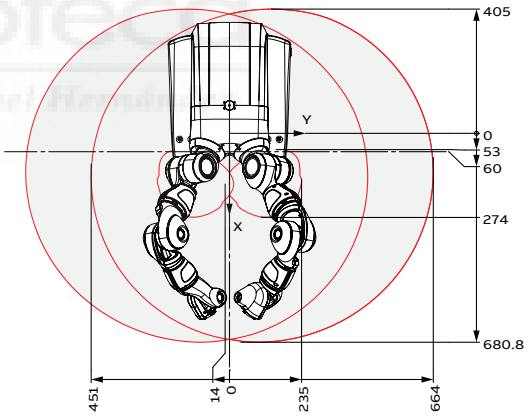
Movement

Axis movement	Working range	Axis max. speed
Axis 1 arm rotation	-168.5° to +168.5°	180°/s
Axis 2 arm bend	-143.5° to +43.5°	180°/s
Axis 7 arm rotation	-168.5° to +168.5°	180°/s
Axis 3 arm bend	-123.5° to +80°	180°/s
Axis 4 wrist rotation	-290° to +290°	400°/s
Axis 5 wrist bend	-88° to +138°	400°/s
Axis 6 flange rotation	-229° to +229°	400°/s

Working range, front view



Working range, top view



Working range, side view

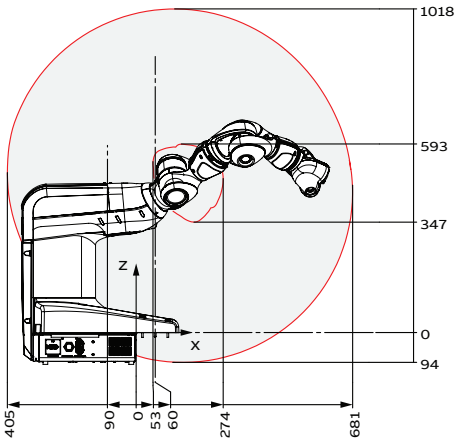


abb.com/robotics

9AKK106354A3254 Rev.H May 2019

B. Programa de RAPID de la Aplicación 2 del brazo izquierdo

```
MODULE Module1

    !VAR robtarget puntoigual:=[[-4.778177,74.06754,108.0856],
[0.5012755,0.5297561,-0.4517601,0.5138037],[-1,-1,0,4],[-148.5594,9E+09,9E+09,9E+09,9E+09,9E+09]];


    PROC main()

        !!!!!!!!!!!!!!   Calibracion Camara, plano y punto maestro !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !MoveJ puntoigual,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        !MoveJ Offs(PuntoMaestro_L,-126,82,80),vnormal,znormal,Servo\WObj:=BandejaVision_L;
!        WaitTime 10;
        MoveJ Offs(PuntoMaestro_L,0,0,150),vnormal,znormal,Servo\WObj:=BandejaVision_L;Stop;
!        WaitTime 10;
!        MoveJ Offs(PuntoMaestro_L,20,20,80),vnormal,znormal,Servo\WObj:=BandejaVision_L;
        !MoveJ PuntoMaestro_L,vnormal,znormal,Servo\WObj:=BandejaVision_L;Stop;
        !MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,0,0,10),0,0,0,\Rx:=0\Ry:=0\Rz:=0),0,0,-25,\Rx:=0
\Ry:=-25\Rz:=0),vnormal,fine,Servo\WObj:=BandejaVision_L;
        !MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,0,0,10),0,0,0,\Rx:=0\Ry:=0\Rz:=0),0,0,0,\Rx:=0
\Ry:=-25\Rz:=0),vnormal,fine,Servo\WObj:=BandejaVision_L;
        !Stop;
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !SetDO custom_DO_6,0;
        MoveJ Home_L,vnormal,z0,Servo\WObj:=Puntos_bandeja_L;
        !Reset Vacuum_Right;
        WaitRob\ZeroSpeed;
        WaitRob\InPos;
     !   WaitDO Comunicacion_establecida,1;
        WaitSyncTask sync27,task_list_brazos_vision;

        !g_Calibrate;
        !g_MoveTo(10);
        !g_MoveTo(0);

        WaitRob\ZeroSpeed;
        WaitRob\InPos;

        WaitSyncTask sync15, task_list_brazos_vision;

        PickandPlace;

        ControlCalidad;

    ENDPROC

    PROC PickandPlace()

        g_MoveTo(0);
        WaitRob\InPos;
        MoveJ ArribaBandeja_L,vnormal,znormal,Servo\WObj:=Puntos_bandeja_L;
```

```
        WaitRob\InPos;
        Coge1_L;
        Deja1_L;
        Coge3_L;
        Deja3_L;
        Coge6;
        Deja6;
        Coge4_L;
        Deja4_L;
        Coge8;
        Deja8;stop;
        !WaitSyncTask sync4,task_list;



    ENDPROC

    PROC Coge2()

        !Definicion del Disparo Vacio
        !TriggIO triggVacio,10\DOp:=Vacuum_Right,1;
        Movej Intermedio,vnormal,z0,Servo\WObj:=BandejaVision_L;

        MotionSup\off;
        g_MoveTo(0);
         WaitRob\InPos;
        !Reset Vacuum_Right;
        Movej RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_2{1},Pieza_2{2},80),0,0,0,\Rx:=-Pieza_2{3}-3
\Ry:=0\Rz:=0),0,15,68,\Rx:=0\Ry:=27\Rz:=0),vnormal,z0,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_2{1},Pieza_2{2},50),0,0,0,\Rx:=-Pieza_2{3}-3
\Ry:=0\Rz:=0),0,15,68,\Rx:=0\Ry:=27\Rz:=0),vnormal,z0,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_2{1},Pieza_2{2},30),0,0,0,\Rx:=-Pieza_2{3}-3
\Ry:=0\Rz:=0),0,15,68,\Rx:=0\Ry:=27\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;

        !TriggL RelTool(RelTool(Offs(PuntoMaestro_L,pieza2{1},pieza2{2},0),0,0,0,\Rx:=pieza2{3}
\Ry:=0\Rz:=0),0,0,20,\Rx:=0\Ry:=25\Rz:=0),vpieza,triggVacio,fine,Servo\WObj:=BandejaVision_L;


        WaitRob \ZeroSpeed;
        !SetDO Vacuum_Right, 1;
        WaitTime 0.75;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_2{1},Pieza_2{2},80),0,0,0,\Rx:=-Pieza_2{3}-3
\Ry:=0\Rz:=0),0,15,68,\Rx:=0\Ry:=27\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_2{1}-20,Pieza_2{2},80),0,0,0,\Rx:=-Pieza_2
{3}-3\Ry:=0\Rz:=0),0,15,68,\Rx:=0\Ry:=27\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;

        MotionSup\On;
    ENDPROC
```

```
    PROC Coge1_L()

        g_MoveTo(6);
        WaitSyncTask sync5,task_list_brazos;
        MoveJ Offs(PuntoMaestro_L_1,-100,40,120),vnormal,fine,Servo\WObj:=BandejaVision_L;

        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_1{1},Pieza_1{2},20),0,0,0,\Rx:=Pieza_1{3}
\Ry:=0\Rz:=0),0,0,-110,\Rx:=0\Ry:=-21\Rz:=0),vnormal,z50,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_1{1},Pieza_1{2},0),0,0,0,\Rx:=Pieza_1{3}
\Ry:=0\Rz:=0),0,0,-90,\Rx:=0\Ry:=-21\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;
        WaitSyncTask sync6,task_list_brazos;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_1{1},Pieza_1{2},2),0,0,0,\Rx:=Pieza_1{3}
\Ry:=0\Rz:=0),0,0,-65,\Rx:=0\Ry:=-21\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;
        WaitSyncTask sync7,task_list_brazos;
        WaitRob\ZeroSpeed;
        g_GripIn\holdForce:=20;
        WaitSyncTask sync8,task_list_brazos;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_1{1},Pieza_1{2},40),0,0,0,\Rx:=Pieza_1{3}
\Ry:=0\Rz:=0),0,0,-65,\Rx:=0\Ry:=-21\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;

        WaitSyncTask sync9,task_list_brazos;
        SyncMoveOn sync10,task_list_brazos;
        MoveJ DejadaPieza1_1\ID:=10,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_2\ID:=20,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_3\ID:=30,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_4\ID:=40,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_5\ID:=50,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_6\ID:=60,v300,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_7\ID:=70,v300,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_8\ID:=80,v300,znormal,Servo\WObj:=BandejaVision_L;
        SyncMoveOff sync11;



    ENDPROC

    PROC Deja1_L()
        MotionSup\Off;
        SyncMoveOn sync12,task_list_brazos;
        MoveJ DejadaPieza1_9\ID:=10,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_10\ID:=20,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        WaitRob\ZeroSpeed;
        WaitTime 2;
        g_MoveTo(4); !Abre la pinza;
        MoveJ Offs(DejadaPieza1_10,-20,30,20)\ID:=40,v100,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_14\ID:=50,v100,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_11\ID:=60,v100,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza1_12\ID:=70,v100,znormal,Servo\WObj:=BandejaVision_L;
        WaitRob\ZeroSpeed;
        WaitTime 2;
```

```
        MoveJ Offs(DejadaPieza1_12,-20,50,20)\ID:=80,v500,znormal,Servo\WObj:=BandejaVision_L;
        SyncMoveOff sync13;
        MoveJ DejadaPieza1_13,v1000,znormal,Servo\WObj:=BandejaVision_L;


        MotionSup\On;
        MoveJ Home_L,vnormal,z0,Servo\WObj:=Puntos_bandeja_L;
        !WaitSyncTask sync26,task_list;

    ENDPROC


    PROC Coge3_L()
        MotionSup\Off;
        g_MoveTo(6);
        WaitSyncTask sync5,task_list_brazos;
        MoveJ Offs(PuntoMaestro_L_1,-100,40,120),vnormal,fine,Servo\WObj:=BandejaVision_L;

        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_3{1},Pieza_3{2},20),0,0,0,\Rx:=Pieza_3{3} ↵
\Ry:=0\Rz:=0),0,10,-100,\Rx:=0\Ry:=-22\Rz:=0),vnormal,z50,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_3{1},Pieza_3{2},7),0,0,0,\Rx:=Pieza_3{3}  ↵
\Ry:=0\Rz:=0),0,10,-90,\Rx:=0\Ry:=-22\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;
        WaitSyncTask sync6,task_list_brazos;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_3{1},Pieza_3{2},7),0,0,0,\Rx:=Pieza_3{3}  ↵
\Ry:=0\Rz:=0),0,10,-70,\Rx:=0\Ry:=-22\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;
        WaitSyncTask sync7,task_list_brazos;
        WaitRob\ZeroSpeed;
        g_GripIn\holdForce:=20;
        WaitSyncTask sync8,task_list_brazos;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_3{1},Pieza_3{2},20),0,0,0,\Rx:=Pieza_3{3} ↵
\Ry:=0\Rz:=0),0,10,-65,\Rx:=0\Ry:=-22\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;

        WaitSyncTask sync9,task_list_brazos;
        SyncMoveOn sync10,task_list_brazos;
        MoveJ DejadaPieza3_1\ID:=10,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_2\ID:=20,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_3\ID:=30,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_4\ID:=40,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_5\ID:=50,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_6\ID:=60,v400,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_61\ID:=61,v400,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_7\ID:=70,v400,znormal,Servo\WObj:=BandejaVision_L;
        SyncMoveOff sync11;
        MotionSup\On;


    ENDPROC

    PROC Deja3_L()
        MotionSup\Off;
        SyncMoveOn sync12,task_list_brazos;
```

```
        MoveJ MoldePieza3_L2\ID:=20,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ MoldePieza3_L1\ID:=10,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ MoldePieza3_L3\ID:=30,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        WaitTime 1;
        MoveJ MoldePieza3_L4\ID:=31,vpieza,znormal,Servo\WObj:=BandejaVision_L;


        WaitRob\ZeroSpeed;
        WaitTime 2;g_MoveTo(4);
        MoveJ Offs(MoldePieza3_L4,-20,30,20)\ID:=40,v200,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_9\ID:=51,v200,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_9\ID:=61,v200,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_11\ID:=71,v100,znormal,Servo\WObj:=BandejaVision_L;
        WaitRob\ZeroSpeed;
        WaitTime 2;
        MoveJ Offs(DejadaPieza3_11,-20,50,20)\ID:=81,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza3_12\ID:=91,v1000,znormal,Servo\WObj:=BandejaVision_L;
        SyncMoveOff sync13;

        MotionSup\On;
        MoveJ Home_L,vnormal,z0,Servo\WObj:=Puntos_bandeja_L;
        !WaitSyncTask sync26,task_list;

    ENDPROC

    PROC Coge4_L()
        MotionSup\Off;
        g_MoveTo(6);
        WaitSyncTask sync5,task_list_brazos;
        MoveJ Offs(PuntoMaestro_L_1,-100,40,120),vnormal,fine,Servo\WObj:=BandejaVision_L;

        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_4{1},Pieza_4{2},15),0,0,0,\Rx:=Pieza_4{3} ↵
\Ry:=0\Rz:=0),0,15,-90,\Rx:=0\Ry:=-20\Rz:=0),vnormal,z50,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_4{1},Pieza_4{2},5),0,0,0,\Rx:=Pieza_4{3}  ↵
\Ry:=0\Rz:=0),0,15,-70,\Rx:=0\Ry:=-20\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;
        WaitSyncTask sync6,task_list_brazos;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_4{1},Pieza_4{2},0),0,0,0,\Rx:=Pieza_4{3}  ↵
\Ry:=0\Rz:=0),0,15,-52,\Rx:=0\Ry:=-20\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;
        WaitSyncTask sync7,task_list_brazos;
        WaitRob\ZeroSpeed;
        g_GripIn\holdForce:=20;  !Cierra la pinza con fuerza;
        WaitSyncTask sync8,task_list_brazos;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L_1,Pieza_4{1}-20,Pieza_4{2},40),0,0,0,\Rx:=Pieza_4 ↵
{3}\Ry:=0\Rz:=0),0,15,-40,\Rx:=0\Ry:=-20\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;
        MotionSup\On;
        WaitSyncTask sync9,task_list_brazos;
        SyncMoveOn sync10,task_list_brazos;
        MoveJ DejadaPieza4L_1\ID:=10,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza4L_2\ID:=20,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza4L_3\ID:=30,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza4L_4\ID:=40,vnormal,znormal,Servo\WObj:=BandejaVision_L;
```

```
        SyncMoveOff sync11;

    ENDPROC

    PROC Deja4_L()
        MotionSup\Off;
        SyncMoveOn sync12,task_list_brazos;
        MoveJ MoldePieza4_L1\ID:=10,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ MoldePieza4_L2\ID:=20,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ MoldePieza4_L3\ID:=30,vpieza,znormal,Servo\WObj:=BandejaVision_L;


        WaitRob\ZeroSpeed;
        g_MoveTo(4);
        MoveJ MoldePieza4_L4\ID:=40,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ MoldePieza4_L4\ID:=50,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        !MoveJ Offs(MoldePieza4_L2,0,20,0)\ID:=30,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        !MoveJ Offs(MoldePieza4_L2,-50,40,50)\ID:=40,v500,znormal,Servo\WObj:=BandejaVision_L;
        SyncMoveOff sync13;
        MoveJ PegadoPieza_4_1,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ PegadoPieza_4_2,v100,znormal,Servo\WObj:=BandejaVision_L;
        WaitRob \ZeroSpeed;
        WaitTime 1;
        MoveJ Offs(PegadoPieza_4_2,0,70,50),v200,z0,Servo\WObj:=BandejaVision_L;
        MotionSup\On;
        MoveJ Home_L,vnormal,z0,Servo\WObj:=Puntos_bandeja_L;


    ENDPROC

    PROC Coge6()

        MotionSup\off;
        g_MoveTo(0);
        WaitRob\InPos;
        !Reset Vacuum_Right;
        !MoveJ Pto_int_Pieza6,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        Movej RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_6{1},Pieza_6{2},80),0,0,0,\Rx:=-Pieza_6
{3}-180\Ry:=0\Rz:=0),0,0,0,\Rx:=0\Ry:=27\Rz:=0),vnormal,z0,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_6{1},Pieza_6{2},50),0,0,0,\Rx:=-Pieza_6
{3}-180\Ry:=0\Rz:=0),0,0,0,\Rx:=0\Ry:=27\Rz:=0),vnormal,z0,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_6{1},Pieza_6{2},25),0,0,0,\Rx:=-Pieza_6
{3}-180\Ry:=0\Rz:=0),0,0,0,\Rx:=0\Ry:=27\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;




        WaitRob \ZeroSpeed;
        !SetDO Vacuum_Right, 1;
        WaitTime 0.5;
        Movej RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_6{1},Pieza_6{2},80),0,0,0,\Rx:=-Pieza_6
```

```
{3}-180\Ry:=0\Rz:=0),0,0,0,\Rx:=0\Ry:=27\Rz:=0),vpieza,z0,Servo\WObj:=BandejaVision_L;
        MotionSup\On;
    ENDPROC

    PROC Deja6()

        MoveJ DejadaPieza6_1,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza6_2,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        WaitSyncTask sync16,task_list_brazos;
        MoveJ DejadaPieza6_3,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza6_4,vnormal,znormal,Servo\WObj:=BandejaVision_L;



        MotionSup\off;
        MoveL DejadaPieza6_5,vmolde,fine,Servo\WObj:=BandejaVision_L;
        WaitRob\ZeroSpeed;
        WaitTime 0.1;
        !WaitSyncTask sync2,task_list_brazos;
        WaitTime 2;
        WaitSyncTask sync17,task_list_brazos;
        !Reset Vacuum_Right;
        WaitRob \ZeroSpeed;
        MoveL Offs(DejadaPieza6_5,0,50,0),vnormal,fine,Servo\WObj:=BandejaVision_L;

        MoveJ DejadaPieza6_6,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza6_7,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        WaitTime 2;
        MoveJ DejadaPieza6_6,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza6_8,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza6_9,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        WaitTime 2;
        MoveJ Offs(DejadaPieza6_9,0,100,50),vnormal,fine,Servo\WObj:=BandejaVision_L;
        MotionSup\On;
        MoveJ DejadaPieza6_10,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ Home_L,vnormal,z0,Servo\WObj:=Puntos_bandeja_L;
        waitrob \ZeroSpeed;



    ENDPROC

    PROC Coge8()

        !Definicion del Disparo Vacio
        !TriggIO triggVacio,10\DOp:=Vacuum_Right,1;

        MotionSup\off;
        g_MoveTo(0);
         WaitRob\InPos;
```

```
        !Reset Vacuum_Right;
        MoveJ ArribaBandeja_L,vnormal,znormal,Servo\WObj:=Puntos_bandeja_L;
        Movej RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_8{1},Pieza_8{2},80),0,0,0,\Rx:=-Pieza_8{3}
\Ry:=0\Rz:=0),0,0,0,\Rx:=0\Ry:=27\Rz:=0),vnormal,z0,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_8{1},Pieza_8{2},50),0,0,0,\Rx:=-Pieza_8{3}
\Ry:=0\Rz:=0),0,0,0,\Rx:=0\Ry:=27\Rz:=0),vnormal,z0,Servo\WObj:=BandejaVision_L;
        MoveJ RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_8{1},Pieza_8{2},30),0,0,0,\Rx:=-Pieza_8{3}
\Ry:=0\Rz:=0),0,0,0,\Rx:=0\Ry:=27\Rz:=0),vpieza,fine,Servo\WObj:=BandejaVision_L;

        !TriggL RelTool(RelTool(Offs(PuntoMaestro_L,pieza2{1},pieza2{2},0),0,0,0,\Rx:=pieza2{3}
\Ry:=0\Rz:=0),0,0,20,\Rx:=0\Ry:=25\Rz:=0),vpieza,triggVacio,fine,Servo\WObj:=BandejaVision_L;


        WaitRob \ZeroSpeed;
        !SetDO Vacuum_Right, 1;
        WaitTime 0.5;
        Movej RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_8{1},Pieza_8{2}+30,40),0,0,0,\Rx:=-Pieza_8
{3}\Ry:=0\Rz:=0),0,0,0,\Rx:=0\Ry:=27\Rz:=0),vnormal,z0,Servo\WObj:=BandejaVision_L;
        Movej RelTool(RelTool(Offs(PuntoMaestro_L,Pieza_8{1},Pieza_8{2}+70,100),0,0,0,\Rx:=-Pieza_8
{3}\Ry:=0\Rz:=0),0,0,0,\Rx:=0\Ry:=27\Rz:=0),vnormal,z0,Servo\WObj:=BandejaVision_L;
        MotionSup\On;
    ENDPROC

    PROC Deja8()


        MotionSup\off;
        MoveJ DejadaPieza8_1,v500,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza8_2,v200,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza8_3,v200,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza8_4,vpieza,znormal,Servo\WObj:=BandejaVision_L;
        WaitRob\ZeroSpeed;
        WaitTime 1;
        WaitSyncTask sync2,task_list_brazos;
        !Reset Vacuum_Right;
        MoveJ Offs(DejadaPieza8_4,0,30,20),vpieza,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ Offs(DejadaPieza8_6,0,50,10),vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza8_6,vpieza,fine,Servo\WObj:=BandejaVision_L;
        WaitRob\ZeroSpeed;
        WaitTime 1;
        !MoveJ DejadaPieza8_7,v100,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ Offs(DejadaPieza8_6,0,30,0),vpieza,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ DejadaPieza8_5,v500,znormal,Servo\WObj:=BandejaVision_L;

        MoveJ Home_L,vnormal,fine,Servo\WObj:=Puntos_bandeja_L;




    ENDPROC
```

```
    PROC ControlCalidad()
        SetDO Calidad,1;
        !MoveJ ControlCalidad_5,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        MoveJ ControlCalidad_1,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        WaitRob \inpos;
        WaitSyncTask sync20,task_list_L_vision;
        WaitSyncTask sync21,task_list_L_vision;

        MoveJ ControlCalidad_2,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        WaitRob \inpos;
        WaitSyncTask sync22,task_list_L_vision;
        WaitSyncTask sync23,task_list_L_vision;

        MoveJ ControlCalidad_3,vnormal,znormal,Servo\WObj:=BandejaVision_L;
        WaitRob \inpos;
        WaitSyncTask sync24,task_list_L_vision;
        WaitSyncTask sync25,task_list_L_vision;
        WaitSyncTask sync26,task_list;
        MoveJ ControlCalidad_4,vnormal,znormal,Servo\WObj:=BandejaVision_L;


    ENDPROC

ENDMODULE
```