Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Prácticas docentes con el robot colaborativo ABB 15000 Gofa en comunicación con Matlab

Autor:

Fraile Alonso, Lucía

Tutor(es):

Herreros López, Alberto

Departamento de Ingeniería de Sistemas y Automática

Valladolid, junio de 2025.

RESUMEN

Este Trabajo Fin de Grado aborda el diseño y desarrollo de un conjunto de prácticas docentes orientadas a la formación en robótica colaborativa y visión artificial, utilizando una estación de trabajo educativa basada en simulación. A través de herramientas como *RobotStudio*, *In-Sight Explorer* y *MATLAB*, se han desarrollado seis prácticas con fines didácticos, que permiten al alumnado interactuar con sistemas robotizados mediante tareas de clasificación, inspección, lectura de códigos y control externo.

A lo largo del proyecto, se han implementado diferentes técnicas de visión artificial, como el reconocimiento de formas y colores, el recuento de patrones, la inspección de defectos y la identificación de códigos QR. Además, se ha integrado la comunicación entre distintos softwares mediante el protocolo OPC UA, estableciendo una conexión bidireccional entre *MATLAB* y el controlador virtual del robot a través de la herramienta *ABB IoT Gateway*. Esta integración permite al usuario gestionar las acciones desde una interfaz personalizada, así como la actualización en tiempo real del estado del sistema, simulando un entorno de automatización propio de la Industria 4.0.

El trabajo tiene como objetivo no solo favorecer la comprensión de conceptos claves en áreas como la automatización, la visión artificial y las comunicaciones industriales, sino también familiarizar al alumnado con los principios fundamentales de la Industria 4.0, tales como la interoperabilidad y la supervisión remota.

PALABRAS CLAVE

RobotStudio, Visión artificial, OPC UA, IoT Gateway, MATLAB

ABSTRACT

This Final Degree Project presents the design and development of a set of educational practices aimed at training in collaborative robotics and computer vision, using a simulation-based educational workstation. Through tools such as *RobotStudio*, *In-Sight Explorer*, and *MATLAB*, six instructional exercises have been developed to allow students to interact with robotic systems by performing tasks such as classification, inspection, code reading, and external control.

Throughout the project, various computer vision techniques have been implemented, including shape and color recognition, pattern counting, defect inspection, and QR code identification. Furthermore, communication between different software platforms has been integrated using the OPC UA protocol, establishing a bidirectional connection between *MATLAB* and the robot's virtual controller through the *ABB IoT Gateway*. This integration enables the user to manage actions from a custom interface and update the system status in real time, simulating an automation environment aligned with Industry 4.0 principles.

The main objective of this project is not only to enhance the understanding of key concepts in areas such as automation, computer vision, and industrial communication, but also to familiarize students with core principles of Industry 4.0, such as interoperability and remote monitoring.

KEYWORDS

RobotStudio, Computer Vision, OPC UA, IoT Gateway, MATLAB

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Este Trabajo Fin de Grado presenta el diseño y desarrollo de un conjunto de prácticas docentes orientadas al aprendizaje de la visión artificial aplicada a la robótica colaborativa, integradas dentro de un entorno educativo basado en la simulación. Para ello, se han empleado herramientas industriales como *RobotStudio*, *In-Sight Explorer*, *MATLAB* y el protocolo de comunicaciones OPC UA, que permiten al alumnado aplicar conceptos adquiridos de forma teórica a lo largo de la carrera, permitiendo trabajar con tecnologías propias de la Industria 4.0 desde un entorno controlado.

A lo largo del proyecto, se han diseñado seis prácticas de complejidad progresiva, que abarcan desde tareas como la clasificación de objetos por forma y color, la lectura de códigos QR, la inspección dimensional, el reconocimiento de patrones hasta la gestión de inventario mediante comunicación externa con *MATLAB*. Cada práctica se ha planteado con un enfoque didáctico, permitiendo al alumnado adquirir competencias técnicas aplicadas y combinando modelado, programación, configuración de sistemas de visión y comunicación entre diferentes softwares.

Además de su orientación técnica, este proyecto también se plantea como una herramienta docente flexible y escalable. Aunque en este trabajo se han explorado solo algunas de las múltiples aplicaciones posibles de la visión artificial, tales como el reconocimiento de patrones, la clasificación por colores, el recuento, la medición o la detección de defectos, el potencial de esta tecnología es mucho mayor, abarcando una amplia gama de herramientas y configuraciones que pueden adaptarse a distintos requisitos y casos de uso.

1.1. Objetivos generales y específicos

Objetivo general

Diseñar y desarrollar un conjunto de prácticas docentes que integren visión artificial, robótica colaborativa y comunicaciones industriales, con el fin de permitir al alumnado adquirir competencias técnicas mediante el uso de entornos simulados, interfaces gráficas interactivas y sistemas de comunicación entre distintos softwares.

Objetivos específicos

1. **Diseño y modelado de estaciones**
   - Diseñar y modelar estaciones de trabajo virtuales que simulen tareas reales.
   - Configurar adecuadamente todos los elementos necesarios para cada práctica: piezas, objetos de trabajo, herramientas y sensores.
2. **Control e integración**
   - Desarrollar componentes inteligentes y señales de entrada/salida que permitan activar procesos desde la interfaz externa.
   - Implementar lógica de control en RAPID que responda a las solicitudes del usuario mediante interrupciones, garantizando la correcta ejecución.

3. Visión artificial
   - Configurar una cámara emulada mediante el software *In-Sight Explorer* y su conexión con *RobotStudio*.
   - Estudiar y aplicar distintas técnicas de visión artificial como clasificación por formas y colores, inspección de medidas, recuento de patrones o lectura de códigos QR.
   - Integrar los resultados del sistema de visión en el programa RAPID, utilizando funciones específicas para realizar acciones acordes a dichos resultados.
4. Interfaz gráfica
   - Crear interfaces de usuario interactivas en *MATLAB App Designer* que permitan la selección de piezas, el control del robot y la visualización del inventario.
   - Garantizar una experiencia de usuario intuitiva mediante diseños funcionales con botones, imágenes y etiquetas actualizadas en tiempo real.
5. Comunicaciones industriales
   - Establecer conexión bidireccional entre *MATLAB* y el controlador virtual del robot mediante OPC UA, usando *ABB IoT Gateway* como servidor.
   - Configurar nodos, señales digitales y certificados para habilitar la conexión entre los distintos entornos.
6. Validación y aprendizaje
   - Validar el correcto funcionamiento de cada práctica mediante simulaciones completas y análisis de resultados.
   - Promover el aprendizaje del alumnado en temas clave de la Industria 4.0, como la interoperabilidad, la automatización flexible y la conectividad entre sistemas.

1.2. Metodología del trabajo

El proyecto se ha desarrollado siguiendo una metodología basada en el aprendizaje práctico por simulación, con un enfoque progresivo.

En una primera fase, se llevó a cabo un estudio de viabilidad con el objetivo de determinar si era posible implementar un sistema de visión artificial utilizando exclusivamente una cámara emulada en el entorno virtual de *RobotStudio*, en lugar de una cámara física real. Este análisis incluyó la creación del emulador de la cámara Cognex a través del software *In-Sight Explorer*, verificando la viabilidad de establecer una conexión con *RobotStudio* y simular adecuadamente la adquisición de imágenes.

Con esta base establecida, se procedió al diseño, modelado y configuración de estaciones de trabajo virtuales dentro de *RobotStudio*, adaptadas a las necesidades específicas de cada práctica. Estas estaciones incluyeron el robot colaborativo, las piezas utilizadas, la herramienta de sujeción y los componentes inteligentes necesarios para simular tareas como el transporte, la adquisición de imágenes y la comunicación con otros sistemas. Un componente clave fue el encargado de capturar las imágenes de la estación, que posteriormente fueron procesadas mediante las herramientas de inspección y localización disponibles en RobotStudio.

Es importante destacar que el software Cognex *In-Sight Explorer* se utilizó exclusivamente para la generación y gestión de la cámara emulada, mientras que todo el análisis de imagen y el desarrollo de las tareas de visión artificial se realizaron íntegramente dentro del módulo de visión de *RobotStudio*.

Cada una de las seis prácticas se ha desarrollado de forma incremental, comenzando por tareas básicas como la clasificación de formas, hasta llegar a sistemas más complejos que integran visión artificial, lógica de control en RAPID y control externo mediante interfaces gráficas.

Para la última práctica, se diseñó una interfaz del usuario utilizando *MATLAB App Designer*, permitiendo la selección de piezas y el envío de órdenes al robot de manera sencilla e intuitiva. La comunicación entre *MATLAB* y el controlador virtual del robot se estableció mediante el protocolo industrial OPC UA, configurado a través del software *ABB IoT Gateway*, que actúa como servidor intermedio entre los distintos entornos.

Finalmente, todas las prácticas fueron validadas mediante simulaciones funcionales en RobotStudio, comprobando la correcta interacción entre los sistemas y documentando los resultados obtenidos. Esta metodología ha permitido desarrollar un conjunto de prácticas didácticas realistas, aplicables en entornos formativos relacionados con la robótica y visión artificial.

1.3. Estructura de la memoria

La memoria está organizada en seis bloques principales, seguidos de las referencias bibliográficas. Esta organización busca facilitar la comprensión del proyecto, permitiendo una lectura clara y sencilla del desarrollo del trabajo:

- **Capítulo 1: Introducción**

  Este primer capítulo introduce el contexto del proyecto, los objetivos generales y específicos, la metodología empleada y la motivación que impulsa el desarrollo del trabajo. También se presenta la estructura general de la memoria y las convenciones editoriales utilizadas para su redacción.

- **Capítulo 2: Marco teórico**

  Este capítulo recoge los fundamentos teóricos necesarios para la comprensión del proyecto. Se abordan los conceptos clave relacionados con la robótica colaborativa, la visión artificial, las comunicaciones industriales y la simulación de entornos automatizados.

- **Capítulo 3: Estado del arte**

  Aquí se analiza el contexto actual en el que se enmarca el trabajo, revisando brevemente soluciones existentes y tecnologías aplicadas en el ámbito educativo e industrial relacionadas con la visión artificial y la automatización. A partir de esta revisión, se justifica el enfoque adoptado, describiendo las

necesidades educativas que pretende cubrir el proyecto y su alineación con los objetivos de la Industria 4.0.

- **Capítulo 4: Herramientas software utilizadas**

  Se describen las herramientas empleadas a lo largo del proyecto, detallando sus funcionalidades y su papel dentro de las prácticas desarrolladas. Entre ellas destacan: *RobotStudio* como plataforma de simulación robótica, *In-Sight Explorer* para la emulación de cámaras Cognex, *MATLAB App Designer* para la creación de interfaces gráficas y *ABB IoT Gateway* como servidor OPC UA. También se incluyen capturas y explicaciones sobre las interfaces de cada entorno de trabajo.

- **Capítulo 5: Desarrollo de la solución adoptada**

  Este apartado constituye el núcleo del proyecto. Se describen de forma estructurada cada una de las seis prácticas docentes, desde la configuración inicial de la estación hasta la obtención de resultados. Se detalla el modelado de componentes, la configuración del sistema de visión, la programación en RAPID, el desarrollo de la interfaz en *MATLAB* y la integración de las comunicaciones entre plataformas.

- **Capítulo 6: Conclusiones y futuros trabajos**

  En esta sección se recogen las conclusiones generales del proyecto, destacando los logros alcanzados y la aplicabilidad de las prácticas desarrolladas en entornos educativos. También se plantean posibles líneas de mejora y evolución futura, con el objetivo de ampliar las capacidades del sistema y adaptarlo a nuevos escenarios o tecnologías emergentes.

- **Bibliografía**

  Finalmente, se incluye la bibliografía consultada durante el desarrollo del trabajo, siguiendo el formato de citación APA 6ª edición.

1.4. Convenciones y consideraciones editoriales

Para la elaboración de este Trabajo Fin de Grado se han seguido una serie de criterios editoriales que garantizan la coherencia visual y formal del documento, facilitando su lectura y comprensión.

- **Tipografía:** Se ha empleado la fuente Franklin Gothic Book en tamaño 11 puntos para todo el cuerpo del texto, manteniendo un interlineado sencillo de 1,15 y los márgenes recomendados por la Escuela de Ingenierías Industriales de Valladolid. Los títulos y subtítulos se han diferenciado mediante un tamaño de fuente mayor, el uso de negrita y, en algunos casos, el uso de mayúsculas, respetando así una jerarquía clara entre los distintos capítulos y subapartados.

- **Citación:** Las referencias bibliográficas se han elaborado siguiendo las normas del estilo APA 6ª edición, incluyendo citas dentro del texto con el formato autor-fecha y una lista de referencias bibliográficas al final del documento ordenada alfabéticamente.
- **Estilo tipográfico:** Para facilitar la lectura, se han aplicado los siguientes estilos tipográficos:
  - **Negrita**: Se ha utilizado para resaltar conceptos clave, nombres de autores junto con sus fechas, así como nombres de inventos relevantes.
  - *Cursiva*: Se ha empleado para los nombres de softwares y para términos en inglés que no han sido traducidos.
  - "Comillas": Se han usado para identificar elementos de la interfaz, como nombres de pestañas (por ejemplo, "Modelado"), botones (por ejemplo, "Aceptar") y funciones o bloques de código.
- **Numeración y estructura:** La estructura del documento se basa en una enumeración jerárquica de capítulos, apartados y subapartados (por ejemplo, 1, 1.1, 1.1.1) lo que facilita la navegación por el documento y la localización de los contenidos.
- **Figuras y tablas:** Todas las figuras y tablas están numeradas de forma secuencial y acompañadas de un pie de figura o tabla explicativo. Además, se han incluido las referencias bibliográficas correspondientes para aquellas figuras y tablas que no son de autoría propia.
- **Notas destacadas:** A lo largo del trabajo se han incluido notas resaltadas que recogen aspectos claves o consideraciones importantes. Estas se han escrito en **negrita,** con un tamaño de fuente inferior de 10 puntos y un interlineado reducido de 1, con el objetivo de diferenciarlas visualmente del cuerpo principal del texto y así facilitar su identificación por parte del lector.

2. MARCO TEÓRICO

2.1. Robótica colaborativa

2.1.1. Introducción

Definición y etimología

El término *robótica* proviene de la palabra *robot*, que fue utilizada por primera vez en 1920 por el escritor checo **Karel Čapek** en su obra de teatro *R.U.R. (Rossum's Universal Robots)*. La palabra **robot** deriva del término checo *robota*, que significa "trabajo forzado" o "servidumbre". En esta obra, los robots eran seres artificiales creados para realizar trabajos en lugar de los humanos, lo que sentó las bases del concepto moderno de la automatización.

Motivación y necesidad de la robótica

Desde la antigüedad, la humanidad siempre ha buscado nuevas formas de facilitar el trabajo y reducir el esfuerzo físico mediante dispositivos mecánicos. Es por esto que la robótica surge como una respuesta a dicha necesidad, con el objetivo de aumentar la eficiencia, precisión y la seguridad en la realización de diferentes tareas.

2.1.2. Historia y evolución de la robótica

La robótica ha evolucionado significativamente a lo largo de la historia, pasando de simples mecanismos automáticos a complejos sistemas de inteligencia artificial y automatización industrial. Aunque el concepto de robot, tal como lo conocemos hoy, surgió en el siglo XX, sus orígenes se remontan a la Antigüedad, cuando se desarrollaron los primeros autómatas mecánicos para imitar movimientos humanos o animales. (Barrientos, Peñín, Balaguer, & Aracil, 2007)

Autómatas de la Antigua Grecia

La **Antigua Grecia** fue pionera en experimentar con la automatización a través de la mecánica y neumática. Uno de los ingenieros e inventores griegos más destacados fue **Herón de Alejandría (siglo I d.C.)**, que diseñó varios dispositivos mecánicos avanzados para su época.

- **El mecanismo de Herón:** Sistema que abría y cerraba automáticamente las puertas del templo, utilizando los principios de expansión térmica del aire y la presión del agua. Al encenderse el fuego, el aire se expandía empujando el agua hacia un depósito que, mediante un sistema de poleas y contrapesos, abría las puertas. Por el contrario, cuando se apagaba el fuego, el aire se enfriaba, reduciendo la presión y consiguiendo que el agua regresara a su depósito original, cerrando las puertas.

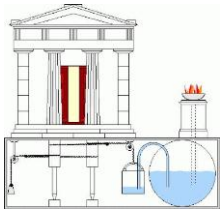


Figura 1. El mecanismo de Herón (Brunete, San Segundo, & Herrero, 2024)

- **El primer dispensador automático:** Máquina con forma de vasija que proporcionaba agua bendita al insertarle una moneda. El propio peso de la moneda al caer accionaba un balancín que levantaba un pequeño émbolo que hacía de tapón y permitía por unos segundos la salida del líquido. (Quellegamos, 2017)

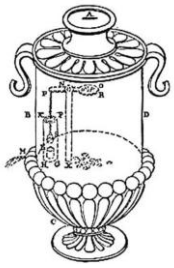


Figura 2. El primer dispensador automático (Blázquez Morales, s.f.)

- **Autómatas teatrales:** Figuras mecánicas utilizadas en espectáculos, que se movían mediante sistemas de poleas y engranajes. Entre sus creaciones destacan los **pájaros de Herón**, que empleaban aire comprimido y sifones para batir sus alas y emitir sonidos, y un **teatro mecánico**, donde las figuras, accionadas por un sistema de cuerdas y poleas, se movían sin intervención humana, cambiaban de escenario y ejecutaban acciones programadas.

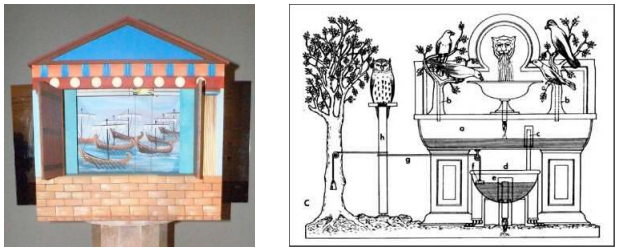


Figura 3. Teatro mecánico de Dionisos (Autómatas, 2010) (izquierda) y Los pájaros de Herón (Zaragoza, s.f.) (derecha)

- **El aeolipile:** Primer dispositivo que transformaba vapor en movimiento. Consistía en una esfera hueca montada sobre un eje, con dos boquillas orientadas en direcciones opuestas. Al calentarse el agua en su interior, el vapor salía por las boquillas, generando un empuje que hacía girar la esfera. Aunque no tuvo aplicaciones prácticas en su época, sentó las bases de la propulsión a chorro y la tecnología del vapor siglos antes de la Revolución Industrial.

Figura 4. Aeolipile (Blázquez Morales, s.f.)

Autómatas de la Edad Media

Durante la **Edad Media,** el ingeniero musulmán **Al-Jazari (1136-1206)** revolucionó el campo de la mecánica con sus innovadores dispositivos automatizados. Fue considerado el "padre de la robótica" y documentó sus inventos en su obra *El libro del conocimiento de dispositivos mecánicos ingeniosos*. En él se describen en detalle multitud de mecanismos basados en sistemas hidráulicos y engranajes. Entre los principales inventos se encuentran:

- **Autómatas hidráulicos:** Diseñó figuras mecánicas que utilizaban la presión del agua para ejecutar movimientos programados. Entre sus innovaciones destaca la **Fuente del Pavo Real**, un mecanismo en el que, al activarse el flujo de agua, las figuras de pavos reales realizaban movimientos sincronizados.

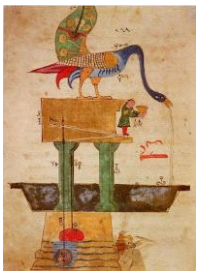


Figura 5. Fuente del Pavo Real (Brunete, San Segundo, & Herrero, 2024)

- **Relojes de agua y de velas:** Medían el tiempo con gran precisión activando distintos mecanismos automáticos. Un ejemplo es el **Reloj del Elefante**, obra maestra de ingeniería que combinaba un complejo sistema de flotadores, engranajes y pesas para indicar la hora mediante una serie de figuras móviles.

Figura 6. Reloj del Elefante (Sutori, s.f.)

- **Barco musical automático:** Funcionaba mediante un sistema de flotadores y levas que activaba figuras mecánicas tocando instrumentos sin intervención humana. Su mecanismo permitía variar la secuencia musical, siendo un precursor de las cajas de música y órganos automáticos.

Figura 7. Barco musical automático (Velasco, 2016)

Otro ejemplo destacado de automatización en la Edad Media es el **Gallo de Estrasburgo**, un autómata de relojería de autoría desconocida, creado alrededor del año 725 d.C. y situado en la Catedral de Estrasburgo. Este gallo utilizaba un sistema de engranajes y pesas para moverse y emitir un canto automático, demostrando el uso de mecanismos de relojería para producir tanto movimiento como sonidos sin necesidad de intervención humana.

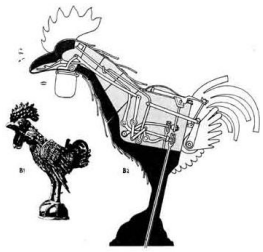


Figura 8. Gallo de Estrasburgo (Brunete, San Segundo, & Herrero, 2024)

Autómatas del Renacimiento

El **Renacimiento (siglos XV-XVI)** fue una época de grandes avances en ciencia, arte e ingeniería. Inspirados en los conocimientos de la Antigüedad, los inventores renacentistas aplicaros principios de engranajes, poleas y sistemas hidráulicos para desarrollar autómatas más complejos.

Uno de los grandes ingenieros y visionarios fue **Leonardo da Vinci (1452-1519),** quien diseñó diversos mecanismos y dispositivos automáticos que pueden considerarse precursores de la robótica.

Algunas de sus creaciones incluyen:

- **El Caballero Mecánico (1495):** Considerado uno de los primeros intentos de construir un robot humanoide, el caballero fue diseñado para moverse de manera autónoma mediante un sistema de poleas, engranajes y cables. Este podía mover los brazos, girar la cabeza e incluso sentarse y levantarse según sus bocetos.

Figura 9. El Caballero Mecánico (Sutori, s.f.)

- **El León Mecánico (1515):** Este autómata fue diseñado para impresionar al rey de Francia. El león podía caminar de forma autónoma y, al llegar a su destino, abrir su pecho para revelar un ramo de lirios.

Figura 10. El León Mecánico (Riccio, 2024)

Autómatas del siglo XVIII-XIX

En esta época, la fabricación de autómatas alcanzó su máximo esplendor, combinando la precisión mecánica con un nivel alto de detalles. Así consiguieron realizar tareas complejas, desde el movimiento de partes del cuerpo hasta la ejecución de melodías musicales, todo ello sin intervención humana directa.

Entre los inventores más destacados de esta época se encuentran **Jacques de Vaucanson** y **Pierre Jaquet-Droz**, quienes crearon autómatas que no solo impresionaban por su complejidad técnica, sino también por su capacidad para imitar la vida y las funciones humanas.

- **Jacques de Vaucanson (1709-1782):** Ingeniero e inventor francés conocido especialmente por el desarrollo de autómatas que podían realizar funciones biológicas y mecánicas complejas. Entre sus creaciones más famosas se encuentra **El Pato Mecánico (1739)** que podía mover las alas, comer granos y "digerir" la comida mediante un sistema de tubos y cámaras.

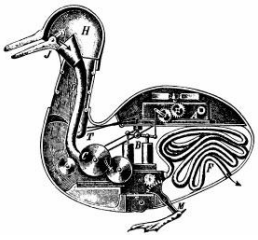


Figura 11. El Pato Mecánico de Vaucanson (Brunete, San Segundo, & Herrero, 2024)

- **Pierre Jaquet-Droz (1721-1790):** Relojero suizo que creó autómatas de gran sofisticación que combinaban relojería y automatización. Sus tres autómatas más conocidos son: **El escritor**, un niño mecánico capaz de escribir frases personalizadas con pluma y tinta, **El dibujante**, que reproducía con precisión dibujos detallados sobre papel, y **El músico**, una figura femenina que tocaba el órgano con movimientos realistas.

Figura 12. El dibujante (Sutori, s.f.)

Primeros desarrollos de robots industriales

El desarrollo de los robots industriales surgió como respuesta a la necesidad de automatizar tareas repetitivas y peligrosas en entornos de manufactura. Su origen se puede remontar al año 1948, cuando **R.C. Goertz** diseñó el primer sistema maestro-esclavo, una tecnología de telemanipulación remota que permitía controlar un brazo mecánico a distancia. Este sistema fue especialmente útil en entornos radiactivos, ya que protegía a los operarios de la exposición directa.

- **Unimate (1961):** El primer robot industrial moderno fue **Unimate**, desarrollado por **George Devol**. En 1961, General Motors lo integró en su planta de ensamblaje de Nueva Jersey, convirtiéndolo en el primer robot industrial operativo de la historia. Este brazo era programable y utilizaba un sistema hidráulico para realizar tareas de soldadura y manipulación de piezas de manera repetitiva y precisa. Su éxito marcó el inicio de la robótica industrial y su expansión a otras industrias.

Figura 13. Unimate (A. Gasparetto, s.f.)

- **Versatran (1963):** Fue uno de los primeros robots industriales, creado por **AMF Corporation**. Usaba un sistema de coordenadas cilíndricas y era accionado por motores eléctricos y sistemas hidráulicos para ciertos movimientos. Tenía una capacidad de carga de hasta 225 kg y se programaba mediante cintas perforadas. Se empleó en tareas repetitivas como la manipulación de piezas y la paletización, facilitando la automatización en líneas de producción.

Figura 14. Versatran (A. Gasparetto, s.f.)

- **Stanford Arm (1969):** Brazo robótico pionero desarrollado por **Victor Scheinman** en la Universidad de Stanford. Fue el primero en utilizar control por computadora y contar con 6 grados de libertad, lo que le permitía realizar movimientos complejos y precisos. Introdujo el uso de la cinemática inversa, una técnica fundamental para calcular trayectorias en entornos tridimensionales, optimizando la manipulación de objetos. Su tecnología fue clave en la evolución de la robótica industrial, influyendo en el diseño de robots como el **PUMA (1978).**

Figura 15. Stanford Arm (A. Gasparetto, s.f.) (izquierda) y PUMA (Fraile Marinero, 2023)(derecha)

- **SCARA (1978):** Fue desarrollado por **Hiroshi Makino** en la Universidad de Yamanashi, Japón, para el ensamblaje de precisión en la industria electrónica. Su estructura permitía movimientos rápidos y repetitivos en un plano horizontal, siendo ideal para tareas como el montaje de circuitos y manipulación de piezas pequeñas.

Figura 16. SCARA (A. Gasparetto, s.f.)

Finalmente, durante las décadas de 1980 y 1990, compañías como **FANUC**, **ABB Robotics** y **Yaskawa Motoman** impulsaron el desarrollo de robots industriales con microprocesadores y controladores programables. Estos avances permitieron una mayor precisión, flexibilidad y autonomía en la manufactura, consolidando la automatización en fábricas de todo el mundo y sentando las bases para la robótica industrial moderna.

Figura 17. Robot Irb6 de ASEA (A. Gasparetto, s.f.)(

2.1.3. Desarrollo de la robótica colaborativa

Los **robots industriales tradicionales** han sido utilizados desde los años 60s para realizar tareas repetitivas en entornos controlados. Estos robots operaban en celdas de seguridad, aislados de los operarios mediante barreras físicas y sensores, con el objetivo de prevenir accidentes.

Sin embargo, la creciente demanda de entornos de trabajo mixtos, donde humanos y robots pudieran colaborar de manera segura y eficiente, impulsó el desarrollo de una nueva generación de robots: los **cobots (robots colaborativos)**. A diferencia de sus predecesores, estos robots están diseñados para operar sin barreras físicas, compartiendo espacio con los trabajadores humanos.

Gracias a tecnologías avanzadas, como sensores de fuerza, sistemas de visión artificial y algoritmos de inteligencia artificial, los cobots pueden detectar la presencia humana y ajustar su comportamiento en tiempo real, minimizando riesgos y maximizando la eficiencia.

Las Tres Leyes de la Robótica y su influencia en la robótica colaborativa

La evolución de los robots colaborativos ha estado marcada no solo por avances técnicos, sino también por consideraciones éticas y de seguridad, influidas en parte por los principios planteados por **Isaac Asimov** en sus **Tres Leyes de la Robótica**, formuladas en 1942:

- **Primera Ley:** *"Un robot no hará daño a un ser humano, ni, por inacción, permitirá que un ser humano sufra daño."*
- **Segunda Ley:** *"Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entran en conflicto con la Primera Ley."*
- **Tercera Ley:** *"Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la Primera o la Segunda Ley."*

Con el tiempo, Asimov introdujo la **Ley Cero**, que, a diferencia de las leyes anteriores centradas en la protección de individuos, establece que la humanidad en su conjunto tiene prioridad sobre cualquier ser humano individual.

- **Ley Cero**: *“Un robot no debe dañar a la humanidad o, por inacción, permitir que la humanidad sufra daño”.*

Aunque estas leyes surgieron en el ámbito literario, han influido en el desarrollo de robots en el mundo real, especialmente en la robótica colaborativa. Los cobots incorporan principios similares a través de tecnologías avanzadas como: sensores de fuerza, sistemas de visión artificial y algoritmos de control, que les permite detectar y evitar situaciones peligrosas para los humanos.

Empresas pioneras en robots colaborativos

Varias empresas lideraron el cambio entre los robots industriales y los colaborativos, desarrollando cobots innovadores con sensores avanzados y algoritmos de control para garantizar la seguridad de los humanos.

Empresa	Primer cobot	Innovaciones
Universal Robots (UR)	UR5 (2008)	Programación intuitiva mediante interfaz gráfica táctil. Sensores de fuerza que detienen el robot ante una colisión.
KUKA Robotics	KUKA LBR iiwa (2013)	Sensores de par en todas las articulaciones. Capacidad de aprendizaje y adaptación a tareas.
ABB Robotics	YuMi (2015)	Diseño de doble brazo con alta precisión. Sensores táctiles y de visión integrados.
FANUC	FANUC CR-35iA (2015)	Alta capacidad de carga (35 kg) Sensores de fuerza para detección de contacto. Interfaz compatible con sistemas industriales.

Tabla 1. Empresas pioneras en robots colaborativos

Figura 18. UR5 (Robots U. , s.f.)(izquierda), YuMi (Robotics, s.f.)(centro) y FANUC CR-35iA (FANUC, s.f.) (derecha)

2.1.4. Definición, características y clasificación de los tipos de robots

Definición de robot

Un robot puede definirse como una máquina programable diseñada para realizar tareas de manera autónoma o semiautónoma. Según la **Asociación de Industrias Robóticas (RIA),** un robot industrial es:

> *"Un manipulador multifuncional reprogramable, capaz de mover materiales, piezas, herramientas o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas."*

Por su parte, la **Federación Internacional de Robótica (IFR)** lo define como:

> *"Una máquina de manipulación automática, reprogramable y multifuncional con tres o más ejes, que puede posicionar y orientar materiales, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o en movimiento."*

Además, la **Organización Internacional de Normalización (ISO 8373:2021)** amplía esta visión al definir un robot como:

> *"Un actuador automático reprogramable, multifuncional y con cierto grado de autonomía, diseñado para moverse dentro de su entorno y realizar tareas específicas."*

Estas definiciones destacan las características fundamentales de los robots: programabilidad, versatilidad y capacidad de manipulación. A su vez, la inclusión del concepto de autonomía en la norma ISO refleja la evolución de la robótica hacia sistemas cada vez más inteligentes e interactivos. (Fraile Marinero, 2023)

Características generales de los robots

Antes de realizar una clasificación de los robots, es importante entender las características técnicas que los definen y diferencian. Estas características determinan su funcionalidad, alcance y aplicabilidad en diferentes entornos. Las principales son:

- **TCP (Tool Center Point):** Es el punto central de la herramienta o efector final del robot. Sus coordenadas se almacenan en el programa, permitiendo un control preciso de su posición y orientación.

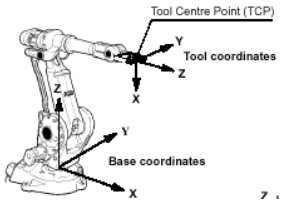


Figura 19. TCP de un robot (Fraile Marinero, 2023)

- **Área de trabajo:** Volumen espacial al que puede acceder el extremo del robot, sin tener en cuenta la herramienta. Esta dependerá del diseño del robot y la longitud de sus articulaciones, determinando el alcance máximo de sus movimientos.

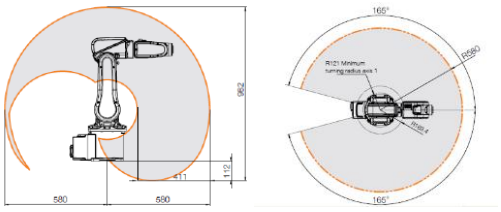


Figura 20. Área de trabajo de un robot ABB IRB 120 (Fraile Marinero, 2023)

- **Grados de libertad (GDL):** Se definen como cada uno de los movimientos independientes que puede ejecutar una articulación de un robot en relación con la anterior. Generalmente, el número de grados de libertad equivale al número de articulaciones, lo que a su vez determina la accesibilidad del robot y su capacidad para orientar sus herramientas.
- **Carga útil:** Es el peso máximo que un robot puede manejar sin comprometer su precisión y estabilidad. Se mide en kilogramos y depende del diseño estructural y la potencia de sus actuadores. Además, se debe tener en cuenta el peso de la pinza o herramienta más el de la pieza manipulada.

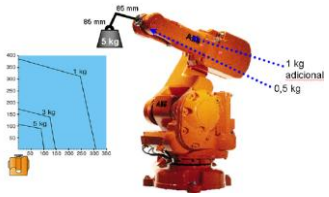


Figura 21. Capacidad de carga de un robot ABB IRB 140 (Fraile Marinero, 2023)

- **Precisión:** Capacidad del robot para posicionar el TCP en un punto específico del espacio, definida como la distancia promedio entre el punto programado y el realmente alcanzado tras varios ciclos.

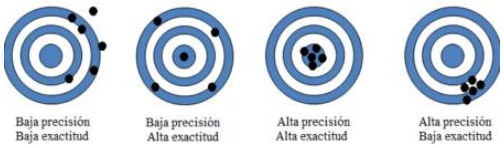


Figura 22. Diferencia gráfica entre precisión y exactitud (Pérez Olguin, s.f.)

- **Repetibilidad:** Capacidad del robot para volver al mismo punto de manera consistente después de múltiples movimientos. Se define como el radio de la esfera que abarca los puntos alcanzados por el robot después de realizar varios desplazamientos hacia el mismo destino programado.
- **Resolución:** Mínima variación de movimiento o ángulo de rotación que puede ejecutar un robot.

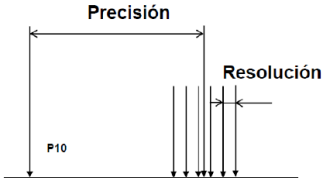


Figura 23. Diferencia gráfica entre precisión y resolución (Vigo, s.f.)

- **Velocidad y aceleración:** Rapidez con la que un robot puede realizar sus movimientos y completar tareas programadas. Esto influye directamente en el rendimiento en aplicaciones que requieren alta velocidad.
- **Puntos singulares:** Son posiciones dentro del espacio de trabajo de un robot en las que este pierde la capacidad de moverse libremente en ciertas direcciones. En estas configuraciones, algunos ejes pueden volverse

indeterminados o requerir movimientos extremadamente rápidos para mantener una trayectoria deseada

- **Sensores y sistemas de control:** Dispositivos que permiten al robot percibir su entorno y ajustar su comportamiento en tiempo real de manera precisa y segura.

Clasificación de los tipos de robots

Los robots pueden clasificarse según diferentes criterios, como su generación tecnológica, su estructura mecánica o su aplicación. (Pérez Cisneros, Valdemar Cuevas Jiménez, & Zaldívar Navarro, 2014). A continuación, se presentan las principales clasificaciones:

Clasificación según su generación tecnológica

- **1ª Generación:** Son robots que realizan tareas secuenciales preprogramadas sin interactuar con su entorno, ya que carecen de sensores para detectar cambios y adaptarse. Son sistemas de lazo abierto, lo que significa que no tienen retroalimentación para verificar resultados ni ajustar sus movimientos.
- **2ª Generación:** Estos robots comienzan a tener la capacidad de percibir su entorno de manera básica. Pueden identificar objetos, medir fuerzas y modificar sus movimientos según las condiciones detectadas, gracias a sensores. Son robots de lazo cerrado, con retroalimentación que les permite verificar algunos de sus resultados.
- **3ª Generación:** Robots que utilizan lenguaje natural para su programación, con la capacidad de planificar tareas de forma autónoma e integrar funciones inteligentes.
- **4ª Generación:** Robots inteligentes capaces de tomar decisiones en tiempo real y moverse con mayor libertad. Incorporan sensores avanzados, aprendizaje automático e inteligencia artificial para adaptarse a diferentes tareas y entornos.
- **5ª Generación:** La generación más avanzada hasta la fecha, estos robots utilizan inteligencia artificial avanzada y aprendizaje automático. Son capaces de adaptarse a entornos dinámicos y desconocidos sin intervención humana, imitando modelos de actuación, razonamiento y conducta. Gracias a su gran capacidad de adaptación, pueden realizar tareas complejas en una amplia variedad de contextos, y su desarrollo continúa en marcha, con aplicaciones que siguen expandiéndose a diversas áreas.

Cada generación de robots ha mejorado en precisión, autonomía e interacción con su entorno, pasando de sistemas mecánicos básicos a máquinas inteligentes y colaborativas. Con cada avance, aumentan su capacidad de adaptación, toma de decisiones y trabajo conjunto con humanos. (PowerBots, 2023)

Clasificación según su aplicación

- **Robots industriales:** Son esenciales en la automatización de procesos industriales ya que son utilizados en fábricas y líneas de producción para tareas desde ensamblaje hasta empaquetado.
- **Robots de servicio:** diseñados para asistir en actividades no industriales, como la limpieza, logística o atención al cliente.
- **Robots médicos:** Aplicados en el sector salud para cirugías asistidas, rehabilitación, diagnóstico y transporte de medicamentos en hospitales.

Figura 24. Robot médico Da Vinci (viamed, s.f.)

- **Robots de exploración:** Se utilizan en entornos extremos o de difícil acceso, como el espacio, el fondo marino o zonas de desastre.

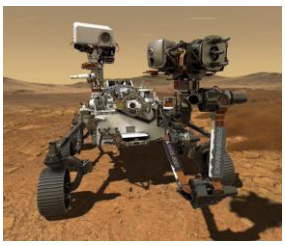


Figura 25. Robot de exploración Perseverance (Andreu, 2022)

- **Robots militares y de seguridad:** Empleados en tareas de reconocimiento, desactivación de explosivos y vigilancia.
- **Robots de entretenimiento y educativos:** Son aquellos diseñados para interactuar con las personas con fines recreativos o de aprendizaje.

Clasificación según su estructura mecánica

- **Robots cartesianos:** Poseen una estructura basada en tres ejes lineales perpendiculares (X, Y, Z). Son ideales para tareas que requieren movimientos rectilíneos y alta precisión.

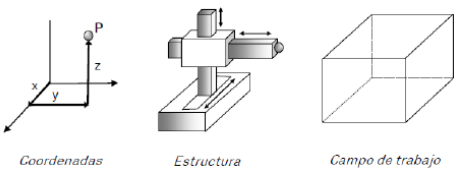


Figura 26. Estructura y campo de trabajo del robot cartesiano (Fraile Marinero, 2023)

- **Robots SCARA (Selective Compliance Robot Arm):** Cuentan con dos articulaciones rotacionales en el plano horizontal y una vertical. Destacan por su velocidad y precisión en movimientos horizontales, siendo ideales para tareas de ensamblaje y manipulación de piezas pequeñas.

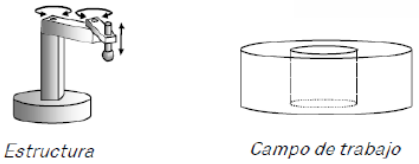


Figura 27. Estructura y campo de trabajo del robot SCARA (Fraile Marinero, 2023)

- **Robots cilíndricos:** Se caracteriza por moverse en coordenadas cilíndricas, combinando desplazamientos lineales y rotación en torno al eje vertical.

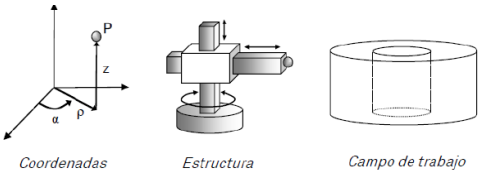


Figura 28. Estructura y campo de trabajo del robot cilíndrico (Fraile Marinero, 2023)

- **Robots antropomórficos:** Son versátiles, ya que su estructura similar a la de un brazo humano, con múltiples articulaciones, les permite realizar una amplia variedad de movimientos en tres dimensiones.

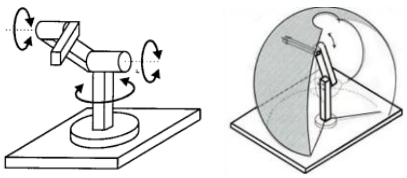


Figura 29. Estructura (Fraile Marinero, 2023) y campo de trabajo del robot antropomórfico (Guerrero, 2013)

- **Robots esféricos o polares:** Se caracterizan porque sus ejes forman un sistema de coordenadas esféricas, lo que les permite moverse en un espacio en forma de esfera parcial en torno a su base. Su diseño les otorga una mayor flexibilidad y alcance, siendo utilizados en aplicaciones que requieren movimientos amplios y precisos.

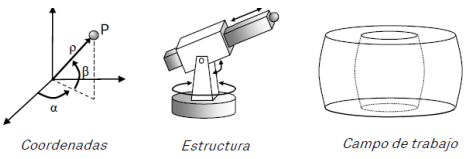


Figura 30. Estructura y campo de trabajo del robot esférico (Fraile Marinero, 2023)

Clasificación según su grado de autonomía

- **Robots teleoperados:** No poseen autonomía y dependen completamente del control humano en tiempo real. Son utilizados en entornos peligrosos o donde la intervención directa no es posible.
- **Robots semiautónomos:** Pueden realizar ciertas acciones de manera automática, pero requieren intervención humana en momentos específicos. Incorporan sensores y algoritmos para tomar decisiones limitadas.
- **Robots autónomos:** Operan de forma independiente sin necesidad de intervención humana, utilizando inteligencia artificial y sensores para adaptarse a su entorno.

Clasificación según su sistema de control

- **Robots programables:** Siguen instrucciones predefinidas y no tienen la capacidad de adaptarse a cambios en su entorno. Estos robots son ideales para tareas repetitivas y estructuradas, donde las condiciones no varían.
- **Robots adaptativos:** Incorporan sensores y algoritmos avanzados que les permiten ajustar su comportamiento en tiempo real según las condiciones del entorno. Este tipo de robots es más flexible y se utiliza en entornos dinámicos donde es necesario responder a cambios imprevistos.
- **Robots inteligentes:** Hacen uso de la inteligencia artificial y técnicas de aprendizaje automático para aprender de su experiencia y mejorar su desempeño de manera autónoma. Estos robots son capaces de tomar decisiones complejas y adaptarse a situaciones imprevistas, lo que los convierte en herramientas altamente avanzadas y versátiles.

Clasificación según su interacción con humanos

- **Robots industriales tradicionales:** Operan en entornos controlados sin interacción directa con humanos. Requieren barreras de seguridad para evitar accidentes.
- **Robots colaborativos (cobots):** Trabajan junto a humanos de manera segura. Incorporan sensores y sistemas de inteligencia artificial para detectar la presencia de personas y ajustar su comportamiento en consecuencia.
- **Robots sociales:** Interactúan directamente con humanos mediante lenguaje natural, expresiones faciales y otros mecanismos de comunicación. Se utilizan para asistencia o entretenimiento.

Clasificación según su movilidad

- **Robots fijos:** Son aquellos que permanecen en un solo lugar y ejecutan tareas dentro de un área específica sin desplazarse. Este es el caso de los brazos robóticos industriales o los colaborativos.
- **Robots móviles:** Son aquellos que pueden desplazarse en su entorno y ejecutar tareas en diferentes ubicaciones.

- **Vehículos de guiado automático (AGV):** Dichos vehículos se mueven siguiendo rutas predefinidas mediante sensores o guías en el suelo.

Figura 31. Vehículo de guiado automático (AGV) (Robots R. d., 2023)

- **Robots móviles autónomos (AMR):** Estos vehículos no dependen de rutas fijas, ya que pueden navegar de forma independiente gracias a sensores avanzados.

Figura 32. Robot móvil autónomo (AMR) (MiR Robots, 2022)

- **Robots humanoides:** Robot diseñado para imitar la forma y las funciones del cuerpo humano. Su estructura incluye partes como cabeza, torso, brazos y piernas, y está construido para realizar tareas similares a las que realizan las personas. Suelen ser utilizados en investigación o asistencia personal.

Figura 33. Robot humanoide (Jizai, s.f.)

- **Robots zoomórficos:** Robot diseñado para imitar las características y comportamiento de los animales.

Figura 34. Robot Spot de Boston Dynamics (Boston Dynamics, s.f.)

Clasificación según el tipo de actuador

- **Actuadores eléctricos:** Son lo más comunes y precisos. Utilizan motores eléctricos para mover el robot, ofreciendo un control preciso de velocidad y posición.
- **Actuadores neumáticos:** Utilizan aire comprimido para generar movimiento, siendo adecuados para tareas rápidas y repetitivas. Sin embargo, tienen menor precisión y requieres de sistemas de compresión de aire.
- **Actuadores hidráulicos:** Funcionan con fluidos a alta presión, generando grandes fuerzas, lo que los hace perfectos para aplicaciones en las que se requiera alta carga y potencia.

2.1.5. Diferencias entre robótica industrial y colaborativa

La robótica industrial y la robótica colaborativa tienen diferencias claves en términos de diseño, seguridad y la forma en que interactúan con su entorno. A continuación, se detallas las principales diferencias entre ambos tipos:

Criterio	Robótica industrial	Robótica colaborativa
Tamaño	Grandes y robustos, ocupan espacios delimitados en fábricas	Compactos y ligeros, diseñados para adaptarse a espacios de trabajo compartidos
Sensores	Limitados, principalmente para tareas específicas como visión artificial o control de fuerza.	Integran múltiples sensores avanzados (visión, fuerza…) para garantizar seguridad y adaptación al entorno.
Interacción con humanos	Operan en celdas de trabajo aisladas, sin contacto directo con humanos.	Diseñados para trabajar junto a los humanos sin barreras de seguridad.
Seguridad	Necesitan vallas, barreras o zonas de seguridad para evitar accidentes.	Incorporan sensores y sistemas de detección para evitar colisiones.
Flexibilidad y programación	Programación compleja, requiere personal especializado.	Fácil de programar y reconfigurar para diferentes tareas.
Capacidad de adaptación	Menos flexibles, diseñados para tareas específicas y repetitivas.	Altamente adaptables a distintos procesos y cambios en la producción.
Costos y mantenimiento	Mayor costo debido a sistemas de seguridad y mantenimiento especializado.	Menor costo en instalación y mantenimiento, no requiere medidas de seguridad adicionales.
Aplicaciones	Producción masiva: ensamblaje, soldadura, pintura, embalaje.	Trabajos flexibles y personalizados: ensamblaje, manipulación, control de calidad.

Tabla 2. Diferencias entre robótica industrial y colaborativa

2.1.6. Aplicaciones industriales de los robots colaborativos

Los robots colaborativos, o cobots, han revolucionado la industria al permitir una interacción segura entre humanos y robots. Algunas de las aplicaciones industriales de estos robots son:

- **Ensamblaje y montaje:** Los cobots son especialmente útiles en tareas de ensamblaje donde la precisión y repetibilidad son esenciales.
- **Pick and Place:** Pueden ser utilizados en líneas de producción para mover productos o componentes de un lugar a otro, optimizando el proceso de flujo y mejorando la rapidez con la que se realizan estas tareas repetitivas.
- **Soldadura y pintura:** Los cobots pueden ser programados para realizar tareas de soldadura y pintura en piezas metálicas o de otro tipo, asegurando precisión y uniformidad. Además, pueden realizar tareas que serían peligrosas para los humanos a largo plazo como la soldadura a altas temperaturas o la pintura con productos químicos.
- **Inspección y control de calidad:** Equipados con cámaras, sensores y tecnología avanzada, los cobots pueden llevar a cabo inspecciones de calidad de productos terminados: verificando dimensiones o detectando defectos, reduciendo el error humano.
- **Empaquetado y paletización:** En procesos de empaquetado, los cobots pueden colaborar en la recolección de productos, empaquetarlos adecuadamente y luego colocarlos en palets para su distribución.
- **Maquinado y acabado:** Pueden realizar operaciones de torneado, fresado o pulido con alta precisión, mejorando la calidad y reduciendo los tiempos de producción. Dado que son muy precisos, estos cobots son ideales para piezas que requieren tolerancias estrictas.

2.1.7. Normativas y seguridad en robótica colaborativa

La **seguridad en la robótica colaborativa** es un aspecto crucial para garantizar la protección de los operarios que interactúan directamente con los robots. Existen diversas normativas y métodos diseñados para asegurar un entorno de trabajo seguro:

Normativas relevantes

- **ISO 10218:** Aunque originalmente diseñada para robots industriales, esta norma también se aplica en el contexto de la robótica colaborativa, ya que aborda aspectos fundamentales de seguridad que deben ser considerados en entornos donde humanos y robots interactúan. Trata temas como la instalación, funcionamiento, mantenimiento y programación, esenciales para cualquier tipo de robot que opere cerca de personas.
- **ISO/TS 15066:** Esta normativa se enfoca específicamente en la seguridad de los robots colaborativos, proporcionando especificaciones detalladas para garantizar que los robots puedan trabajar en conjunto con los humanos sin poner en riesgo su seguridad. Incluye pautas sobre las fuerzas de contacto, velocidad y espacios de trabajo compartidos, reduciendo así el riesgo de accidentes o lesiones.

Métodos de seguridad en cobots

La seguridad es un aspecto fundamental en el diseño y operación de los robots colaborativos. A diferencia de los robots industriales tradicionales, que suelen operar en áreas más restringidas, los cobots están diseñados para trabajar en estrecha colaboración con los humanos. Es por esto que, para garantizar la seguridad en esos entornos, se han implementado diversas tecnologías y métodos.

- **Sensores de fuerza y par:** Permiten a los robots colaborativos detectar cualquier contacto físico de personas u objetos. Estos sensores miden la fuerza aplicada en los ejes del robot, y en caso de detectar una colisión o cierta resistencia, el cobot puede reducir su velocidad instantáneamente, detenerse por completo o retroceder suavemente.
- **Cámaras de seguridad y sensores de visión:** Los cobots suelen incorporar cámaras de seguridad y sensores de visión para monitorear en todo momento su entorno. Estas tecnologías permiten detectar la presencia de personas u obstáculos en su área de trabajo, ajustar el comportamiento del robot en tiempo real para evitar colisiones y crear zonas de seguridad virtuales donde el robot reduce su velocidad o se detiene cuando una persona accede a ella.

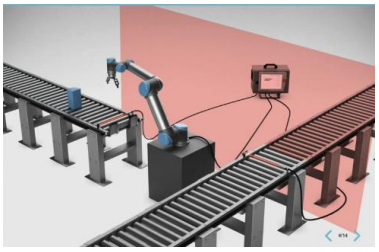


Figura 35. Sensores de seguridad en robots colaborativos (Universal Robots, s.f.)

- **Limitación de velocidad y espacio de trabajo:** Los robots colaborativos operan a velocidades más bajas que los industriales tradicionales. Además, el área de operación del cobot se puede configurar para evitar que se mueva fuera de ciertos límites y así proteger a los operarios que trabajan cerca.
- **Diseño ergonómico y materiales seguros:** Los cobots están diseñados con características ergonómicas y materiales que reducen el riesgo de lesiones: presentan bordes redondeados que evitan cortes o atrapamientos, superficies suaves y peso ligero.
- **Sistemas de parada de emergencia:** Aunque los cobots están diseñados para operar de manera segura, también cuentan con sistemas de parada de emergencia que permiten detener el robot de inmediato en caso de una situación crítica. Estos sistemas pueden ser activados manualmente por el operario o automáticamente por sensores integrados.

Evaluación de riesgos en robótica colaborativa

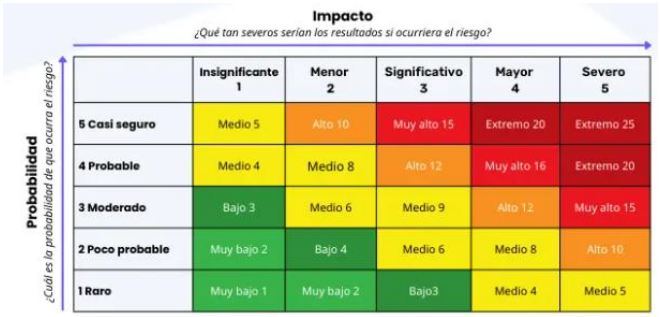
La evaluación de riesgos es un proceso esencial para garantizar que la interacción entre robots colaborativos y operarios humanos sea segura y eficiente. Antes de integrar un cobot en un entorno de trabajo, es indispensable analizar los posibles peligros y establecer medidas de mitigación que reduzcan o eliminen los riesgos. (SafetyCulture, 2025).

Entre los **principales objetivos** de esta evaluación se encuentran:

- Identificar los posibles peligros asociados a dicha interacción
- Establecer medidas de seguridad para reducir o eliminar riegos
- Garantizar que el robot opere dentro de los límites de seguridad establecidos.

Fases de la evaluación de riesgos

1. **Identificación de peligros:** Se analizan los riesgos potenciales asociados a la interacción humano-robot, considerando factores como: áreas de posible colisión, fuerza aplicada en caso de contacto y detección de posibles fallos en los sensores o mecanismos de control.
2. **Análisis de riesgos:** Se evalúan los peligros detectados en la anterior fase en función de su probabilidad de ocurrencia y la gravedad de sus consecuencias. Para ello, se utiliza la **matriz de riesgos**, una herramienta que clasifica los peligros en distintos niveles (bajo, medio, alto) y ayuda a priorizar las medidas de mitigación necesarias.



	Insignificante 1	Menor 2	Significativo 3	Mayor 4	Severo 5
5 Casi seguro	Medio 5	Alto 10	Muy alto 15	Extremo 20	Extremo 25
4 Probable	Medio 4	Medio 8	Alto 12	Muy alto 16	Extremo 20
3 Moderado	Bajo 3	Medio 6	Medio 9	Alto 12	Muy alto 15
2 Poco probable	Muy bajo 2	Bajo 4	Medio 6	Medio 8	Alto 10
1 Raro	Muy bajo 1	Muy bajo 2	Bajo3	Medio 4	Medio 5

Figura 36. Matriz de riesgos (SafetyCulture, 2025)

3. **Implementación de medidas de mitigación:** Para minimizar los riesgos identificados, se aplican estrategias de seguridad como: barreras físicas y delimitaciones virtuales, sistemas de detección y parada de emergencia o sensores y visión artificial para detectar obstáculos y evitar colisiones.
4. **Verificación y validación:** Una vez implementadas las medidas de seguridad, se realizan pruebas para evaluar su efectividad. Este proceso incluye simulaciones y pruebas en entornos controlados, asegurando así que el cobot pueda operar sin riesgos para los operarios.

Este análisis no solo protege a los operarios, sino que también contribuye al éxito de la integración de la robótica colaborativa en la industria 4.0.

2.2. Visión artificial

2.2.1. Introducción

La visión artificial es un campo de estudio dentro de la inteligencia artificial que permite a las máquinas interpretar y analizar imágenes o videos para extraer información útil. Su desarrollo ha sido abordado tanto desde un enfoque teórico como desde una perspectiva aplicada en la industria.

Desde un punto de vista académico, **Berthold K. P. Horn** define la visión artificial como:

> *La disciplina que estudia el procesamiento e interpretación automática de imágenes del mundo real con el objetivo de extraer información útil para la toma de decisiones o la interacción con el entorno, combinando principios de la óptica, la percepción visual, la inteligencia artificial y el procesamiento de señales. (Horn, 1986)*

Por otro lado, empresas líderes en tecnología han desarrollado sus propias definiciones basadas en la aplicación práctica de la visión artificial. **IBM**, por ejemplo, la define como:

> *Un campo de la inteligencia artificial que utiliza el machine learning y las redes neuronales para enseñar a ordenadores y sistemas a extraer información significativa de imágenes digitales, vídeos y otras entradas visuales, y a hacer recomendaciones o tomar medidas cuando se detectan defectos o problemas (IBM, s.f.)*

2.2.2. Historia y evolución de la visión artificial

La visión artificial ha evolucionado desde sus raíces en la neurociencia hasta convertirse en una de las tecnologías más sofisticadas de la inteligencia artificial. Su desarrollo ha sido impulsado gracias a avances en matemáticas, computación y aprendizaje profundo, lo que ha permitido transformar industrias enteras.

Orígenes y primeras ideas (1950s)

La visión artificial tiene su origen en la comprensión de cómo los seres vivos perciben el mundo. En la década de 1950, los neurocientíficos **David Hubel y Torsten Wiesel** realizaron experimentos con gatos para estudiar cómo el cerebro procesa imágenes, descubriendo cómo diferentes neuronas responden a bordes, líneas y patrones específicos. Este hallazgo influyó en el diseño de redes neuronales artificiales utilizadas más adelante en visión por computadora.

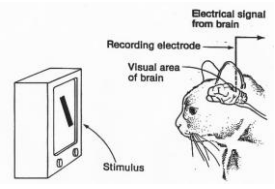


Figura 37. Diagrama del experimento de Hubel y Wiesel sobre la percepción visual en gatos (Datademia, s.f.)

Además, en 1957, **Russell Kirsch**, un ingeniero del National Bureau of Standards (NBS), creó la **primera imagen digital** al escanear una fotografía de su hijo, marcando el inicio de la era digital en el procesamiento de imágenes.

Figura 38. Primera imagen digitalizada (Kirsch) (Datademia, s.f.)

Primeros intentos computacionales (1960s - 1970s)

Con el crecimiento de la informática en la década de 1960, los científicos comenzaron a explorar cómo las computadoras podían interpretar imágenes. En 1960, se desarrolló la primera tecnología de escaneo de imágenes, permitiendo a las computadoras digitalizar imágenes por primera vez.

Pocos años después, en 1963, **Lawrence Roberts**, considerado uno de los padres de la visión artificial, propuso un método para **transformar imágenes 2D en representaciones tridimensionales**. Este trabajo marcó un hito importante, ya que demostró que las computadoras podían ir más allá de simplemente almacenar imágenes y comenzar a interpretar y reconstruir escenas en tres dimensiones. Las ideas de Roberts sentaron las bases para aplicaciones futuras en campos como la robótica, la simulación y la realidad virtual.

Posteriormente, en 1974, surgió el **Reconocimiento Óptico de Caracteres (OCR)**, que permitió a los ordenadores detectar texto impreso. Al mismo tiempo, el **Reconocimiento Inteligente de Caracteres (ICR)**, comenzó a utilizar redes neuronales para interpretar texto manuscrito.

Figura 39. Ejemplo de fuentes OCR-A y OCR-B, utilizadas en sistemas de OCR (Rufenacht, 2020)

Algoritmos y redes neuronales tempranas (1980s - 1990s)

En 1982, **David Marr** propuso un modelo jerárquico para entender cómo los seres humanos y las máquinas pueden procesar imágenes.

Su teoría estableció tres **niveles de análisis**:

- **Computacional**: Define qué problema resuelve el sistema.
- **Algorítmico**: Describe los procedimientos empleados para procesar la información visual.

- **Implementación**: Especifica la tecnología empleada para desarrollar el sistema.

Por otro lado, el procesamiento visual tiene como objetivo proporcionar información sobre la forma y posición de los objetos en el espacio. Este proceso se organiza en tres fases:

- **Esbozo primario:** Se extrae información básica de la imagen bidimensional, como bordes, contornos y cambios de intensidad.
- **Esbozo 2.5D:** Introduce información sobre orientación, profundidad y estructura de las superficies en relación con el observador.
- **Modelo 3D:** Construye una representación tridimensional independiente del punto de vista, permitiendo el reconocimiento de objetos.

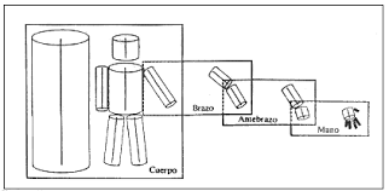


Figura 40. Modelo de David Marr sobre el procesamiento visual en computadoras (Ventosa Ribas, 2021)

En el mismo año, **Kunihiko Fukushima** creó el **Neocognitron**, una de las primeras redes neuronales convolucionales, capaz de identificar patrones visuales sin intervención humana. Más adelante, en 1986, se creó el **operador de Canny**, un algoritmo que utiliza segmentación y reconocimiento de objetos para la detección de bordes en imágenes.

Expansión del aprendizaje automático (2000s - 2010s)

En esta década, la visión artificial vivió avances significativos especialmente en el campo del reconocimiento de objetos. Un hecho importante, fue el desarrollo del **algoritmo de Viola-Jones** en 2001, que permitía el reconocimiento facial en tiempo real por primera vez.

Además, en paralelo, el **algoritmo SIFT (Scale-Invariant Feature Transform)** de 2004 permitió identificar y coincidir puntos clave en imágenes, lo que facilitó el reconocimiento de objetos en diferentes escalas y orientaciones.

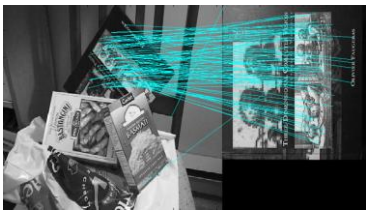


Figura 41. Algoritmo SIFT (Duarte Villaseñor & Chang Fernández)

Inteligencia artificial y redes neuronales profundas (2010s - actualidad)

El mayor avance de esta etapa fue la introducción del **aprendizaje profundo (deep learning)** en visión artificial, impulsado por el desarrollo de **Redes Neuronales Convolucionales (CNNs)**.

En 2012, el modelo **AlexNet**, desarrollado por **Geoffrey Hinton** y su equipo, marcó un antes y un después en el reconocimiento de imágenes. Su capacidad para identificar objetos con una precisión sin precedentes consolidó el uso del deep learning en visión artificial.

Con la llegada del deep learning, surgieron avances significativos en el campo de la visión artificial, como **YOLO (You Only Look Once)**, un algoritmo de detección de objetos en tiempo real, y **GANs (Generative Adversarial Networks)**, capaz de generar imágenes sintéticas con un alto nivel de realismo.

Figura 42. Detección de objetos con YOLO (Datademia, s.f.)

2.2.3. Sistemas de visión artificial en robótica

La visión artificial es un componente fundamental en el campo de la robótica, ya que permite a los robots percibir, interpretar e interactuar con su entorno, facilitando la toma de decisiones autónoma y eficiente. Mediante el uso de cámaras y sensores avanzados, los robots pueden capturar imágenes y analizarlas para reconocer y clasificar objetos, evitar colisiones y optimizar trayectorias de trabajo.

En el caso de los robots colaborativos, o cobots, la integración de visión artificial no solo mejora la precisión en la ejecución de tareas, sino que también les permite adaptarse dinámicamente a cambios en su entorno, lo que les permite trabajar de manera segura junto a los operarios. Además, la combinación de visión 3D y algoritmos de aprendizaje automático les permite reconocer objetos con mayor precisión, manipular piezas de diferentes tamaños y formas e incluso realizar inspecciones de calidad más eficaces. (Corke, Jachimczyk, & Pillat, 2023)

Uno de los desarrollos más avanzados en visión artificial en el campo de la robótica es el **SLAM (Simultaneous Localization and Mapping),** que permite a los robots móviles y vehículos autónomos mapear su entorno mientras se localizan dentro de él. Este sistema combina datos de sensores como cámaras, LiDAR y sensores de profundidad para generar mapas tridimensionales precisos en tiempo real y ajustar la posición del robot, mejorando así su capacidad de evitar obstáculos y planificar rutas de manera eficiente.

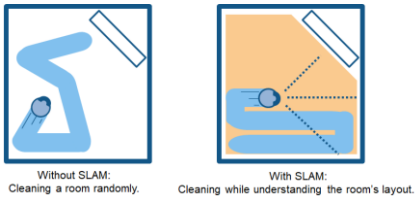


Figura 43. Ejemplo del sistema SLAM de limpieza (MathWorks, s.f.)

2.2.4. Clasificación sistemas de visión artificial industriales

Los sistemas de visión artificial se pueden clasificar en función de diversos criterios, entre ellos el tipo de sensor empleado, los algoritmos de análisis aplicados o la tecnología empleada para el procesamiento de imágenes.

Clasificación según el tipo de imagen capturada

- **Cámaras monocromáticas:** Capturan imágenes en escala de grises, lo que puede ser suficiente para muchas aplicaciones industriales en las que no sea necesaria la información de color. Proporcionan una mayor sensibilidad a la luz y menor ruido.
- **Cámaras a color:** Capturan imágenes con información cromática, lo que es importante en aplicaciones dónde realizar una diferenciación de color es esencial.

Clasificación según la dimensión de la imagen

Características	Visión 2D	Visión 3D
Dimensiones	Ancho y alto	Ancho, alto y profundidad
Precisión	Sistema limitado a contornos y patrones en un plano bidimensional	Permite identificar el volumen y la forma de los objetos con mayor precisión
Complejidad	Menos complejo y más rápido Menor costo de implementación	Más compleja y costosa, pero ofrece capacidades avanzadas y mayor versatilidad
Entornos de uso	Entornos estructurados y tareas planas	Entornos no estructurados y dinámicos
Tipos de sensores	Cámaras estándar 2D, sensores de contraste y color, sensores de iluminación infrarroja (IR)...	Cámaras estéreo, sensores LiDAR, cámaras de luz estructurada, sensores de tiempo de vuelo (ToF)...
Ventajas	Rápida y económica, fácil de implementar y ampliamente adoptada en la industria	Mayor precisión, detalle y capacidad para manejar objetos y espacios complejos
Limitaciones	No puede manejar información de profundidad y es limitada en entornos dinámicos o con objetos irregulares	Mayor costo y complejidad debido al requerimiento de mayor potencia de procesamiento

Principales aplicaciones	Inspección de productos, lectura de códigos de barras, detección de defectos…	Navegación autónoma, inspección de formas complejas, manipulación de objetos en entornos dinámicos…

Tabla 3. Diferencias entre visión 2D y 3D

Figura 44. Cámara 2D In-Sight D900 (Cognex Corporation, s.f.) y sensor 3D LiDAR (Leishen Intelligent System Co, s.f.)

Clasificación según la capacidad de percepción de profundidad

Para aplicaciones industriales en las que se requiere información tridimensional, se emplean diversas tecnologías que les permite conocer datos de profundidad. (Pérez Cisneros, Valdemar Cuevas Jiménez, & Zaldívar Navarro, 2014)

- **Sensores de profundidad:** Utilizan tecnologías como la luz estructurada o el tiempo de vuelo para medir distancias a diferentes puntos de una escena.
- **Visión estereoscópica:** Se basa en el uso de dos cámaras para imitar la percepción de profundidad del ojo humano.
- **Sensores de tiempo de vuelo (ToF):** Realizan una medición del tiempo que tarda la luz en reflejarse en un objeto, generando así un mapa de profundidad preciso.

Figura 45. Sensor de tiempo de vuelo (ToF) (Yoigo, 2025)

Clasificación según la tecnología de procesamiento de imágenes

- **Visión basada en reglas:** Consiste en la aplicación de algoritmos que siguen reglas predefinidas para analizar las imágenes. Algunos de estos métodos son el uso de filtros, umbralización, segmentación o algoritmos de detección de bordes.
- **Visión por computadora basada en IA:** Utiliza técnicas más novedosas, como el aprendizaje automático y el aprendizaje profundo (deep learning). De esta forma, son capaces de analizar las imágenes de una forma más precisa y flexible.

Debido a los avances en visión artificial, los sistemas basados en IA se han vuelto cada vez más populares en aplicaciones industriales. Sin embargo, en industrias donde es esencial mantener un control riguroso y comprensible sobre el análisis de imágenes, la combinación de enfoques tradicionales con IA sigue siendo una estrategia clave.

Clasificación según la aplicación industrial

- **Inspección de calidad:** Sistemas de visión artificial utilizados para garantizar que los productos cumplan con los estándares de calidad requeridos. Algunas aplicaciones son: la detección de defectos superficiales, la medición de dimensiones y tolerancias en las piezas y la verificación de ensamblajes y componentes.
- **Identificación y trazabilidad:** Sistemas que permiten identificar y rastrear productos a lo largo de la cadena de producción. Algunas aplicaciones son: la lectura de códigos de barra, QR y OCR.
- **Guía de robots:** Sistemas que permiten a los robots interactuar y adaptarse al entorno de manera precisa y autónoma. Algunas aplicaciones son: la manipulación de objetos en la línea de producción y la colaboración entre robots y humanos.
- **Seguridad y monitoreo:** sistemas que garantizan la seguridad en entornos industriales y monitorean los procesos para evitar los fallos o accidentes.

Clasificación según el tipo de iluminación utilizada

La iluminación es un factor crítico en los sistemas de visión artificial, ya que determina la calidad y precisión de las imágenes capturadas. Dependiendo de la aplicación industrial, se utilizan diferentes tipos de iluminación para así obtener mejores resultados.

- **Iluminación visible:** Es la forma más común de iluminación en los sistemas de visión artificial.
- **Iluminación infrarroja (IR):** Es especialmente útil en aplicaciones que requieren detección de características no visibles a simple vista, como la inspección de materiales, la detección de fallos térmicos y visión nocturna, permitiendo capturas de imágenes en entornos con poca o ninguna luz visible.
- **Iluminación ultravioleta (UV):** Este tipo de iluminación es ideal para aplicaciones que requieren la detección de características en materiales translúcidos o la autenticación de productos.

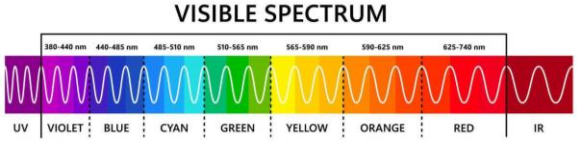


Figura 46. Espectro visible (PngTree, s.f.)

2.2.5. Reconocimiento de objetos

Una de las aplicaciones más destacadas de la visión artificial es el reconocimiento de objetos. Para ello, se emplean distintos métodos de detección en base a determinadas características de los objetos: geometría, color, textura o haciendo uso de deep learning. (Corke, Jachimczyk, & Pillat, 2023)

Detección basada en características geométricas

- **Detección de bordes**: Algoritmos como Canny o Sobel permiten identificar los bordes de un objeto analizando los cambios de intensidad en los píxeles.
- **Detección de contornos:** Se basa en la segmentación de objetos mediante la búsqueda de regiones conectadas con características similares.
- **Coincidencia de patrones:** Comparación de formas predefinidas con objetos en la imagen.

Figura 47. Filtro de detección de bordes sobel (Alaens, s.f.)

Detección basada en el color y textura

- **Análisis de histogramas:** Permite estudiar la frecuencia de los valores de intensidad de píxeles en una imagen. Es útil para identificar patrones globales en la imagen.
- **Segmentación basada en color:** Utiliza espacio de color como RGB o HSV para identificar objetos de colores específicos. Este método es especialmente útil cuando el color es una característica distintiva del objeto.
- **Análisis de textura:** Algunas técnicas, como el **Local Binary Pattern (LBP)** y las **matrices de co-ocurrencia de niveles de gris (GLCM)**, son especialmente útiles para describir patrones de textura en superficies, como la rugosidad, la regularidad o la dirección de las estructuras.

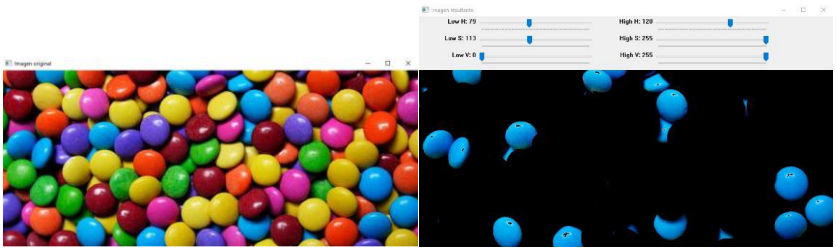


Figura 48. Ejemplo de segmentación basada en RGB

Detección basada en deep learning

Las **redes neuronales convolucionales (CNNs)** han revolucionado el reconocimiento de objetos en la industria al permitir la identificación automática sin necesidad de definir manualmente las características a identificar. Estas redes necesitan ser entrenadas con grandes volúmenes de imágenes etiquetadas para así posteriormente reconocer patrones complejos de forma precisa. Además, su gran ventaja es que son capaces de reconocer objetos en condiciones variables de iluminación, perspectiva, orientación u oclusión. (Corke, Jachimczyk, & Pillat, 2023)

En la actualidad, existen arquitecturas populares como **YOLO (You Only Look Once)**, **Faster R-CNN** y **SSD (Single Shot MultiBox Detector)**, que ofrecen un equilibrio entre alta precisión y velocidad en la detección de objetos. Estas arquitecturas han permitido implementar sistemas de visión artificial en tiempo real, lo que es crucial en aplicaciones industriales.

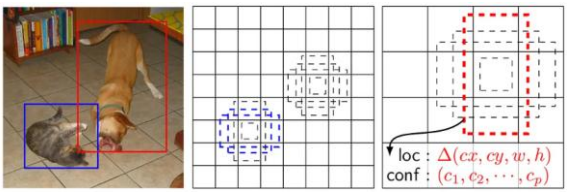


Figura 49. Arquitectura SSD (Single Shot MultiBox Detector) (Annan, 2024)

2.2.6. Iluminación y óptica en visión artificial

Como se ha mencionado anteriormente, **la iluminación es un elemento clave** en los sistemas de visión artificial, ya que afecta directamente a la calidad de la imagen obtenida y, por ende, a la precisión de su análisis.

Una iluminación adecuada influye en aspectos de la imagen como el contraste, la nitidez y la visibilidad de los detalles, lo que facilita el procesamiento e interpretación de la imagen. Por el contrario, una mala iluminación puede generar problemas como sombras o reflejos, que dificulten la detección de características y disminuya así su eficacia.

Es por esto que es importante considerar varios factores al diseñar el sistema de iluminación: la intensidad luminosa, la direccionalidad de la luz, el espectro adecuado según la aplicación y la consistencia de la iluminación.

Tipos de iluminación

Según la empresa **BCNVision** (BCNVision, Sistemas de iluminación para aplicaciones de visión artificial, 2017), existen diferentes técnicas de iluminación diseñadas para optimizar las aplicaciones de visión artificial. Cada una de ellas se adapta a necesidades específicas de cada aplicación.

- **Luz frontal:** La luz y la cámara están alineadas en la misma dirección. Ilumina el objeto de manera uniforme, reduciendo sombras y suavizando texturas.

- **Luz lateral:** La luz incide desde un costado del objeto, resaltando los relieves y bordes mediante la creación de sombras definidas.
- **Iluminación por campo oscuro:** La luz se proyecta en ángulo bajo desde todas las direcciones, iluminando los bordes y defectos de la superficie mientras el fondo permanece oscuro.
- **Iluminación por contraste:** La fuente de luz se coloca detrás del objeto, generando una silueta bien definida que permite diferenciar su contorno con precisión.
- **Iluminación axial difusa:** La luz se dirige lateralmente y es reflejada a través de un espejo semitransparente, proporcionando una iluminación homogénea y minimizando reflejos en superficies planas y reflectantes.
- **Iluminación difusa de tipo Domo:** La luz se distribuye desde una cúpula esférica en todas direcciones, eliminando sombras y reflejos, y suavizando las irregularidades de la superficie del objeto.
- **Iluminación por láser:** Se proyecta un rayo láser sobre el objeto en un ángulo conocido. Analizando la distorsión del haz, se pueden extraer datos sobre la forma y profundidad del objeto.

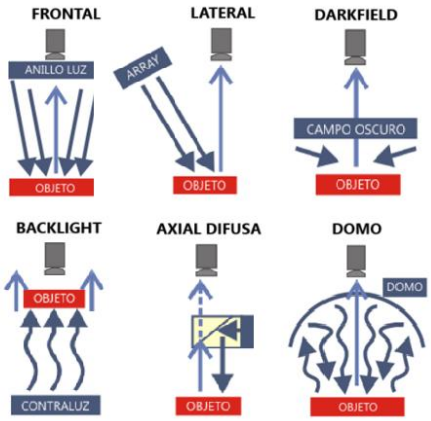


Figura 50. Tipos de iluminación (BCNVision, Sistemas de iluminación para aplicaciones de visión artificial, 2017)

Lentes y ópticas utilizadas en visión artificial

Para llevar a cabo la elección de la lente adecuada, es necesario considerar varios parámetros clave, como la distancia al objeto, el campo de visión, la resolución requerida y las condiciones de iluminación.

Parámetros ópticos importantes

- **Distancia focal:** Determina el grado de ampliación de la imagen y el campo de visión. Una distancia focal corta proporciona un campo de visión amplio, mientras que una larga permite mayor distancia.
- **Profundidad de campo:** Rango de distancias dentro del cual los objetos aparecen enfocados en la imagen. Depende de la distancia focal, el diafragma y la distancia a la que se encuentre el objeto.

- **Apertura (número f):** Indica el tamaño de la apertura de la lente y regula la cantidad de luz que entra. Un número bajo deja pasar más luz, pero reduce la profundidad de campo y uno alto, todo lo contrario.
- **Resolución:** Define la capacidad de la lente para distinguir detalles pequeños en la imagen sin perder nitidez.

Tipos de lentes más comunes

- **Lentes de focal fija:** Dispone de una distancia focal constante, ofreciendo imágenes nítidas y estables.
- **Lentes de focal variable (zoom óptico):** Permite cambiar la distancia focal, ajustando así el nivel de zoom sin necesidad de mover la cámara. Esto ofrece mayor flexibilidad en el campo de la visión.
- **Lentes telecéntricas:** Proporcionan una perspectiva sin distorsión, ideal para mediciones precisas en aplicaciones industriales.
- **Lentes gran angular:** Proporcionan un campo de visión más amplio, permitiendo capturar más área en una sola imagen. Son útiles en espacios reducidos, aunque pueden generar distorsión en los bordes.
- **Lentes ojo de pez:** Ofrecen un campo de visión extremadamente amplio (hasta 180°), generando una imagen con distorsión esférica.
- **Lentes macro:** Diseñadas para enfocar objetos a distancias muy cortas, permitiendo capturar objetos muy pequeños con alto detalle.

Figura 51. Tipos de lentes (BCNVision, 2013)

2.3. Comunicaciones industriales

2.3.1. Introducción

Definición

> *Las comunicaciones industriales son sistemas de interconexión que permiten el intercambio de datos entre dispositivos de campo (sensores, actuadores), controladores (PLCs) y sistemas de gestión (SCADA/MES), caracterizándose por requisitos estrictos de tiempo real, fiabilidad y determinismo. (SICMA21, 2021)*

Motivación y necesidad de las comunicaciones industriales

Desde sus inicios, las comunicaciones industriales han supuesto un reto fundamental para la automatización, impulsando la evolución tecnológica del sector. Además, la constante necesidad de intercambiar información entre dispositivos heterogéneos (sensores, actuadores, robots, PLCs) las han convertido en un pilar indispensable para el correcto funcionamiento de los procesos industriales, garantizando:

- Coordinación precisa de las operaciones
- Trasferencia fiable de datos
- Toma de decisiones en tiempo real

2.3.2. Historia y evolución de las comunicaciones industriales

Década 1970-1980

Durante esta década, la industria comenzó a incorporar dispositivos para automatizar los procesos, lo que llevó a la necesidad de establecer comunicaciones eficientes entre dichos dispositivos. En este período, las conexiones eran principalmente limitadas y se basaban en un esquema punto a punto, donde cada sensor o actuador se conectaba directamente a su controlador mediante cables. Esta configuración resultaba en grandes cantidades de cableado, poca escalabilidad y un mantenimiento elevado.

Década 1980-1990

Con el aumento de complejidad de los procesos industriales y las limitaciones que presentaban conexiones punto a punto, surgieron los primeros protocolos de comunicaciones industriales, conocidos como **buses de datos**. Estos protocolos permitieron compartir un mismo canal de comunicación, lo que contribuyó a reducir el cableado y mejorar su organización. Entre los protocolos más destacados de esta etapa se encuentran **Modbus (1979), CAN (1986)** y **Profibus (1989).**

Sin embargo, debido a la aparición de múltiples protocolos propietarios y la falta de estandarización, fue necesario crear pasarelas que permitieran convertir señales entre distintos dispositivos. Esto introdujo nuevos conceptos como la **comunicación en**

serie y las **arquitecturas maestro-esclavo**, que facilitaron la interoperabilidad entre equipos de distintos fabricantes.

Década 1990-2000

En los años 90, los protocolos de buses de campo se consolidaron, y debido a la creciente complejidad de las aplicaciones industriales, se hizo crucial mejorar la velocidad de transmisión y la fiabilidad de las redes de comunicación. Durante esta década, los protocolos mencionados previamente se adaptaron a diferentes sectores, mientras que comenzaron a surgir nuevos protocolos, como el **Ethernet industrial**, que favorecieron la estandarización y la interoperabilidad entre los dispositivos de distintos fabricantes.

Década 2000-2010

Durante esta década se produjo la evolución y perfeccionamiento de los protocolos basados en Ethernet: **EtherNet/IP** se estableció como estándar para la automatización, **PROFINET** demostró su eficacia en entornos de manufactura y **EtherCAT** emergió como solución óptima para aplicaciones que demandaban alta velocidad. Estos avances superaron las limitaciones iniciales, logrando transmisiones estables a 100 Mbps con una precisión de sincronización sin precedentes y configuraciones de red altamente flexibles

Años 2010s

Con la llegada de la Industria 4.0, las comunicaciones industriales experimentaron una transformación profunda. Los protocolos de comunicación dejaron de ser utilizados únicamente para controlar procesos automatizados y pasaron a desempeñar un papel clave en la recolección y análisis de los datos en tiempo real. Esto permitió no solo optimizar el funcionamiento de los sistemas industriales, sino también anticipar fallos, mejorar efectividad y facilitar la toma de decisiones.

Además, se integraron tecnologías avanzadas como el **Internet Industrial de las Cosas (IIOT)**, Big Data y la Inteligencia Artificial, que aportaron capacidades analíticas, predictivas y adaptativas a los entornos productivos. Como consecuencia de esta evolución, surgieron las denominadas **arquitecturas hibridas** que integraban **sistemas operáticos tradicionales** (**OT**) con **tecnologías de información (IT)**, dando lugar a entornos más inteligentes y flexibles.

Años 2020s

Hoy en día las comunicaciones industriales han permitido sistemas conectados, seguros y descentralizados, donde la conectividad total entre los dispositivos es clave para garantizar la eficiencia. Las tecnologías actuales permiten redes más robustas, seguras y adaptables, capaces de operar en entornos exigentes y dinámicos. Estas redes no solo manejan grandes volúmenes de datos en tiempo real, sino que también ofrecen mayor flexibilidad y fiabilidad, adaptándose a las necesidades cambiantes de la industria moderna.

2.3.3. Niveles de comunicación y pirámide CIM

En los procesos industriales, la comunicación entre los distintos dispositivos y sistemas que los integran es fundamental para garantizar su correcto y eficiente funcionamiento. Para facilitar dicha interacción, las comunicaciones industriales se estructuran en niveles jerárquicos, cada uno con funciones específicas, tecnologías y protocolos adaptados a su propósito.

Estos niveles se relacionan directamente con la **pirámide CIM (*Computer Integrated Manufacturing*),** un modelo que organiza cómo se distribuye la información dentro de una planta industrial. A cada nivel le corresponde un tipo de dispositivo, una función concreta y un conjunto de protocolos de comunicación, lo que facilita la integración vertical entre la automatización del proceso y la gestión empresarial.

Estructura de niveles y protocolos

Nivel CIM	Equivalente en comunicaciones	Dispositivos	Protocolo
Campo	**Campo**	Sensores, actuadores, válvulas	Profibus, IO-Link, HART, AS-Interface
Célula	**Control**	PLCs, DCS, controladores robóticos	Modbus TCP, CANopen o EtherNet/IP, DeviceNet
Planta	**Supervisión**	SCADA, servidores OPC, HMI	OPC UA, PROFINET IRT MQTT, bases de datos SQL
Oficina	**Gestión**	MES, ERP, sistemas Cloud	REST API, SAP interfaces

Tabla 4. Niveles de comunicaciones

Funciones principales por nivel

- **Nivel de campo:** Se encarga de la interacción directa con el proceso físico, recogiendo datos de los sensores y actuando sobre elementos como motores o válvulas. Requiere comunicaciones rápidas.
- **Nivel de célula:** Ejecuta la lógica de control, recibiendo los datos del nivel de campo y tomando decisiones locales a través de PLCs o controladores. Es una parte fundamental para la automatización de un proceso industrial.
- **Nivel de planta:** Actúa como intermediario entre la operación y la gestión, supervisando el proceso global mediante sistemas SCADA o DCS. Además, permite visualizar las variables y generar alarmas e históricos.
- **Nivel de oficina:** Se orienta hacia la gestión y planificación, utilizando los datos del proceso para tomar decisiones estratégicas. Representa la visión global del negocio, centrada en eficiencia y toma de decisiones.

A medida que se asciende en la jerarquía de la pirámide CIM, el volumen de datos tiende a incrementarse, ya que se integra información proveniente de múltiples dispositivos y procesos. Sin embargo, los requisitos de tiempo real se vuelven menos críticos, ya que las decisiones en estos niveles no suelen requerir respuestas

inmediatas. Por lo contrario, en los niveles inferiores, la comunicación debe ser rápida y confiable, ya que cualquier retraso puede afectar a la producción. (Guerrero, 2013).

Figura 52. Pirámide CIM

2.3.4. Tipos de protocolos y redes

Como se ha mencionado anteriormente, los protocolos de comunicación industrial estandarizan el intercambio de datos entre los sistemas y dispositivos que componen un proceso industrial, asegurando seguridad, eficiencia e interoperabilidad entre los distintos fabricantes.

En función de la tecnología empleada y sus prestaciones, se pueden clasificar principalmente en dos grandes grupos:

- **Buses de campo:** Están pensados para entornos industriales simples y robustos.
- **Redes basadas en Ethernet industrial:** Están pensados para una conectividad más avanzada que requiere velocidades elevadas.

Protocolos basados en buses de campo

- **Modbus RTU / Modbus ASCII:** Es uno de los protocolos más antiguos y más utilizados en automatización industrial. Sigue una arquitectura maestro-esclavo y funciona sobre líneas serie RS-232 /RS-485.
  - **Modbus RTU:** Usa codificación binaria, siendo la comunicación más eficiente.
  - **Modbus ASCII:** Emplea caracteres ASCII, más legible pero menos eficiente.

Modbus RTU Frame Format

Start	Address	Function	Data	CRC	End
≥3.5 char	8 bit	8 bit	N * 8 bit	16 bits	≥3.5 char

Modbus ASCII Frame Format

Start	Address	Function	Data	LRC	End
:	2chars	2chars	N * 1 chars	2chars	CR, LF

Figura 53. Comparativa de formatos de trama Modbus RTU y Modbus ASCII (Park, Park, & Kang, 2022)

- **Profibus (Process Fiel Bus):** Este protocolo se divide en dos variantes:
  - **Profibus DP (*Decentralized Peripherals*):** Ofrece comunicación rápida entre controladores maestros y dispositivos de entrada/salida (E/S) que actúan como dispositivos esclavos, ideal para aplicaciones de automatización que requieren intercambio rápido de información.

- **Profibus PA (*Process Automation*):** Es una extensión de Profibus DP, optimizada para su uso en entornos de automatización de procesos, especialmente en zonas clasificadas como entornos peligrosos.

- **CANopen:** Es un protocolo de comunicación basado en **el bus CAN *(Controller Area Network)***, optimizado para aplicaciones embebidas y sistemas distribuidos que requieren transmisión eficiente de datos en tiempo real. Ofrece alta fiabilidad, robustez e integración de dispositivos de diferentes fabricantes.

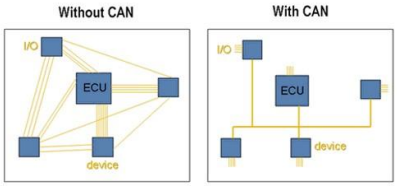


Figura 54. Arquitectura de red con y sin bus CAN (Instruments, 2025)

- **DeviceNet:** Protocolo basado en CAN desarrollado por **Rockwell Automation**. Utiliza un bus lineal que combina alimentación y comunicación en un mismo cable, permitiendo conectar numerosos dispositivos. Destaca por su instalación sencilla, su robustez y la posibilidad de identificar fallos en nodos individuales sin detener la red.
- **AS-Interface (*Actuator Sensor Interface*):** Red simple y económica diseñada específicamente para dispositivos binarios como sensores de proximidad o botones. Opera a baja velocidad y utiliza un único cable que combina alimentación y comunicación. Su diseño la hace ideal para aplicaciones básicas de automatización donde la simplicidad y el bajo costo son prioritarios, aunque queda limitada a señales digitales simples.

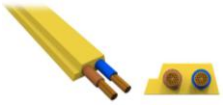


Figura 55. Conectores AS-i en cable plano (Bihl+Wiedemann, s.f.)

Protocolos basados en Ethernet industrial

- **Modbus TCP/IP:** Es una evolución del protocolo Modbus RTU que funciona sobre redes Ethernet utilizando TCP/IP. Usa un modelo de comunicación cliente-servidor y se caracteriza por ser fácil de implementar en sistemas de automatización industrial.

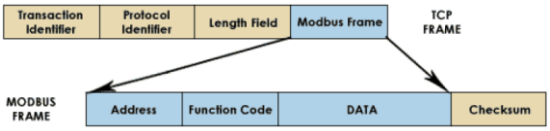


Figura 56. Encapsulamiento de la trama Modbus en TCP (Barragán Piña, 2013)

- **PROFINET:** Protocolo industrial basado en Ethernet, desarrollado por **Siemens**, diseñado para aplicaciones de automatización que requieren alta velocidad y comunicación en tiempo real. Permite múltiples topologías para adaptarse a diversas necesidades de implementación.

- **PROFINET RT (*Real-Time*):** Ideal para el control general de máquinas.
- **PROFINET IRT (*Isochronous Real-Time*):** Proporciona sincronización precisa, siendo perfecto para aplicaciones de robótica y automatización compleja.

- **EtherCAT (*Ethernet for Control Automation Tchnology*):** Protocolo basado en Ethernet, destacado por su alto rendimiento, baja latencia y capacidad para gestionar grandes redes de dispositivos. Está diseñado para aplicaciones de automatización que demandan tiempos de respuesta ultrarrápidos y alta precisión.

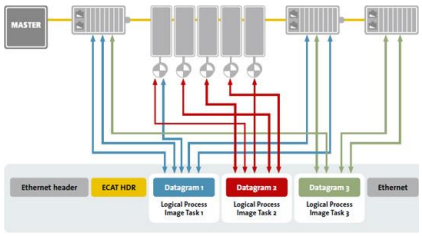


Figura 57. Trama EtherCAT: Datagramas embebidos y procesamiento 'on-the-fly' (Group, s.f.)

- **OPC:** Estándar de comunicación industrial que permite la interoperabilidad entre dispositivos de automatización y sistemas de supervisión sin necesidad de adaptaciones propietarias.

Característica	OPC DA (*Data Access*)	OPC UA (*Unified Architecture*)
Tipo de datos	Datos en tiempo real: valores, estado…	Datos estructurados, históricos, alarmas…
Seguridad	Sin cifrado ni autenticación	Seguridad integrada (cifrado, control de acceso, firma)
Plataforma	Depende de Windows (COM/DCOM)	Multiplataforma
Comunicación	Cliente-Servidor	Cliente-Servidor Publisher/Subscriber
Interoperabilidad	Limitada entre fabricantes	Alta interoperabilidad
Uso actual	Sistemas antiguos (legado)	Estándar moderno Industria 4.0

Tabla 5. Diferencia entre OPC DA y OPC UA

2.3.5. Topologías de redes industriales

La **topología de red** define la disposición física o lógica de los equipos y su cableado en un sistema de comunicación. En un entorno industrial, esta configuración es clave para optimizar el rendimiento, la escalabilidad y la tolerancia a fallos del sistema.

La **topología física** se refiere a la colocación de dispositivos y cables, mientras que la **topología lógica** describe el flujo de datos dentro de la red, sin importar su diseño físico. Ambas topologías no necesariamente tienen que coincidir en una misma red. (SICMA21, 2021).

Entre las topologías más comunes en la industria se encuentran:

- **Punto a punto:** Topología más simple, basada en la conexión directa entre dos dispositivos. Este tipo de configuración no requiere de direcciones de origen y destino dentro del mensaje, ya que la comunicación se produce únicamente entre dos nodos.
  - Ventajas: Simplicidad del cableado y control fácil de la red.
  - Desventajas: Escalabilidad limitada.

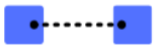


Figura 58. Topología punto a punto (SICMA21, 2021)

- **Bus:** Topología que conecta los nodos mediante una única línea de comunicación compartida. Los nodos deben verificar que el canal esté libre antes de transmitir para evitar colisiones.
  - Ventajas: Económico, fácil de ampliar y un fallo no afecta al sistema.
  - Desventajas: Requiere repetidores para evitar atenuación y reduce el rendimiento.

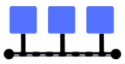


Figura 59. Topología bus (SICMA21, 2021)

- **Árbol:** Esta configuración combina múltiples segmentos de bus organizados jerárquicamente, creando una estructura ramificada.
  - Ventajas: Organización jerárquica por zonas industriales y conectividad extendida para subsistemas distantes.
  - Desventajas: Riesgo de atenuación de señal (necesita repetidores) y mayor complejidad en diseño y mantenimiento.

Figura 60. Topología árbol (SICMA21, 2021)

- **Anillo:** En esta configuración, los dispositivos se conectan formando un circuito cerrado, donde los datos circulan en un único sentido pasando por cada nodo hasta llegar a su destino. Cada estación actúa como repetidor de la señal.
  - Ventajas: Control sencillo sin colisiones, baja atenuación de la señal y sincronización precisa para aplicaciones en tiempo real.
  - Desventajas: Alta vulnerabilidad ante fallos (un solo corte interrumpe toda la red), limitación a distancias cortas y escalabilidad reducida.

Figura 61. Topología anillo

- **Estrella:** En esta configuración, todos los dispositivos se conectan a un nodo central (hub o switch) que gestiona el tráfico de datos.
  - Ventajas: Alto rendimiento al establecer comunicaciones directas entre dispositivos, escalabilidad sencilla y mantenimiento simplificado.
  - Desventajas: Una avería del nodo central paraliza toda la red y mayor costo en cableado. Además, se pueden producir cuellos de botella si el hub no tiene capacidad suficiente.

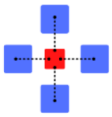


Figura 62. Topología estrella (SICMA21, 2021)

2.3.6. Comunicaciones en robots colaborativos

La comunicación en los robots colaborativos puede clasificarse en dos grandes categorías: **comunicación interna**, que abarca el intercambio de datos entre los componentes del propio robot, como sensores, motores y el controlador; y **comunicación externa**, que se refiere a la conexión con dispositivos o sistemas del entorno, tales como sensores adicionales, sistemas MES o plataformas ERP. Ambas formas de comunicación deben operar con alta velocidad, precisión y sincronización, ya que de ello depende la correcta ejecución de las tareas en tiempo real, sin poner en riesgo la seguridad del operario.

El robot utilizado en este trabajo, el **CRB 15000 GoFa**, ha sido diseñado para integrarse de forma sencilla en entornos industriales gracias a su compatibilidad con múltiples protocolos de comunicación estándar, como PROFINET, EtherNet/IP, Modbus TCP, OPC UA y DeviceNet.

Estos protocolos permiten que el robot se comunique eficazmente con controladores lógicos programables (PLCs), sistemas de supervisión y control (SCADA), sistemas MES o incluso plataformas en la nube. Gracias a esta infraestructura de comunicación, es posible enviar órdenes al robot, recibir información en tiempo real sobre su estado operativo y coordinar su actividad con el resto de los procesos industriales, garantizando una integración fluida y eficiente en el entorno productivo.

Conexiones controlador Omnicore C30

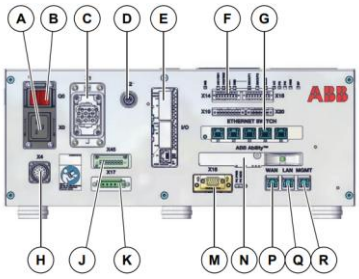


Figura 63. Conexiones del controlador Omnicore C30 (ABB, 2024)

	Descripción	Adhesivo
A	Conector para entrada de la alimentación principal	X0
B	Interruptor de alimentación general	Q0
C	Conector para motor	X1
D	Conector para señal del manipulador (SMB) [1] / Conector de Customer Flange Interface (CFI)[2]	X2
E	Conexión de Scalable I/O	I/O
F	Conexión de interfaz con proxy cliente para intercambio de señal del robot	X14/X15/X19 /X20
G	Conexión para switch Ethernet	ETHERNET SWITCH
H	Conector para FlexPendant (TPU)	X4
J	Conector de salida de alimentación IP20	X45
K	Conector DeviceNet IP20	X17
M	Adaptador de bus de campo esclavo	X18
N	Connected Services Gateway, Con puerto Ability (opción con cable) o conector de antena (opciones 3G o WiFi)	ABB Ability ™
P	Puerto WAN	WAN
Q	Red local, puerto	LAN
R	Puerto de gestión	MGMT

Tabla 6.Conexiones del controlador Omnicore C30 (ABB, 2024)

[1] No disponible para el controlador CRB 15000.

[2] Solo disponible para el controlador CRB 15000.

2.4. Concepto simulación

La **simulación** es una herramienta clave en el desarrollo y la implementación de sistemas de robótica colaborativa. Permite modelar, probar y optimizar el comportamiento de los robots primero en un entorno virtual antes de ser implantados en el mundo real. Esto es especialmente importante en el campo de la robótica colaborativa, donde los cobots interactúan directamente con los operarios y es necesario un comportamiento seguro y eficiente.

2.4.1. Ventajas

Algunas de las ventajas que aporta el uso de simuladores en robótica colaborativa son:

- **Reducción de costes:** Al probar procesos y configuraciones en un entorno virtual antes de implementarlos físicamente, se minimizan los costes asociados a la fase de desarrollo como son los prototipos y las pruebas, además de evitar los posibles daños ocasionados a los equipos o productos
- **Mayor seguridad:** Gracias a las simulaciones es posible identificar riesgos potenciales o fallos antes que los robots interactúen con los operarios en entornos reales.
- **Optimización del tiempo de desarrollo:** La simulación permite realizar ajustes en la programación y configuración del robot sin detener la producción real.
- **Evaluación de múltiples escenarios:** La simulación permite recrear una amplia variedad de condiciones y escenarios, incluyendo situaciones extremas o poco frecuentes, lo que ayuda a garantizar la robustez del sistema.
- **Optimización del diseño:** La simulación facilita la prueba de diferentes configuraciones y diseños de cobots, ajustando parámetros como la velocidad, fuerza o trayectorias, garantizando así un funcionamiento y rendimiento optimo.
- **Entrenamiento y programación:** Los operarios pueden entrenarse en entornos virtuales, familiarizándose con el funcionamiento de los cobots antes de interactuar con ellos en la realidad.

2.4.2. Desventajas

A pesar de las ventajas que presenta la simulación, también existen ciertas **limitaciones**:

- **Diferencias con el entrono real:** Algunos factores imprevistos, como variaciones en las condiciones del entorno, pueden generar discrepancias entre los resultados obtenidos en la simulación y el comportamiento real del robot.
  Además, existen variables externas que no se pueden simular o medir con precisión. Por ello, la toma de decisiones no puede basarse únicamente en la simulación; es necesario realizar pruebas físicas para garantizar el correcto funcionamiento y encontrar la solución final.

- Coste de software especializado
- **Requerimiento de conocimientos avanzados:** Crear procesos de simulación precisos y detallados puede ser un proceso complejo y que consume tiempo, además de que se requiere experiencia en modelado y programación.

3. ESTADO DEL ARTE

Este capítulo tiene como finalidad contextualizar el Trabajo Fin de Grado mediante el análisis de antecedentes académicos relevantes, centrados en proyectos que integran tecnologías como la visión artificial, la robótica colaborativa, la simulación de entornos industriales y las comunicaciones entre plataformas software.

Para ello, se analizan trabajos previos desarrollados tanto en la Universidad de Valladolid como en otras universidades españolas, con el objetivo de identificar enfoques similares, así como sus principales fortalezas y limitaciones. Finalmente, se justifica la solución propuesta en este TFG, destacando las mejoras implementadas y el valor añadido que aporta el diseño de estas prácticas formativas completas.

3.1. Antecedentes

3.1.1. Trabajos previos en la Universidad de Valladolid

En la Escuela de Ingenierías Industriales de la UVa se han llevado a cabo diversos Trabajos Fin de Grado relacionados con la automatización, la robótica industrial y, en menor medida, la visión artificial. A continuación, se presentan algunos de los más relevantes, destacando su proximidad temática y metodológica con el presente proyecto:

- **"Diseño con Autodesk Inventor de una célula de trabajo robótica para realizar montajes multitarea apoyados con visión artificial, coordinado mediante protocolo de comunicación OPC UA entre RobotStudio-MATLAB"** (Sancho García, 2021)

  Este trabajo desarrolla una celda robótica con dos robots ABB (IRB120 y YuMi), destinada a tareas de ensamblaje y manipulación. Se incluye un sistema de visión artificial básico para identificar el color, el tamaño y la posición de los cubos. La simulación y control se realiza mediante *RobotStudio* y *MATLAB*, comunicados a través de OPC UA, e incorpora una interfaz de usuario para gestionar el sistema.

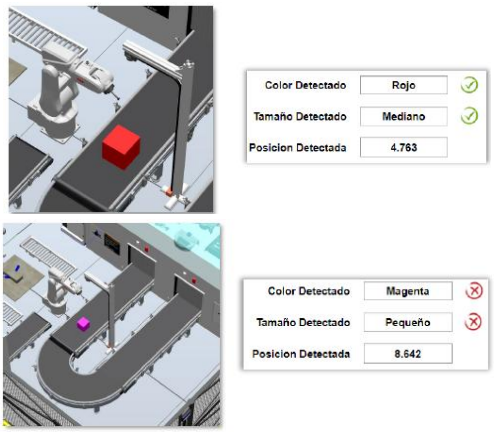


Figura 64. Estación e interfaz gráfica del TFG de Rodrigo Sancho García (Sancho García, 2021)

- **Fortalezas**: Integra visión artificial y comunicación industrial (OPC UA).
- **Limitaciones:** La visión artificial es simple y limitada, sin explotación avanzada de sus capacidades.

- **"Simulación de un Robot colaborativo YuMi (ABB) en entorno RobotStudio comandado desde MATLAB mediante protocolo OPC UA para tocar un Xilófono"** (Pozas Mata, 2022)

  Este trabajo consiste en la programación de un robot colaborativo YuMi para tocar un xilófono, donde las notas se solicitan dinámicamente desde una interfaz hombre-máquina (HMI) desarrollada en *MATLAB* mediante el protocolo de comunicaciones OPC UA. Además, se genera el sonido de la nota seleccionada en cada momento.

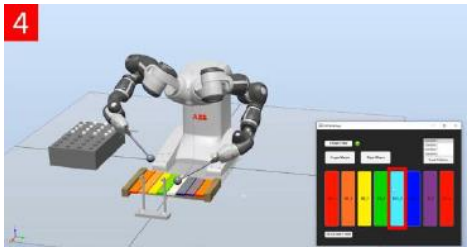


Figura 65. Estación e interfaz gráfica del TFG de Elena Pozas Mata (Pozas Mata, 2022)

- **Fortalezas:** Comunicación bidireccional entre *MATLAB* y *RobotStudio* mediante OPC UA, y presenta un enfoque educativo con una interfaz interactiva.
- **Limitaciones:** No incluye visión artificial y se trata de una aplicación muy concreta y cerrada.

- **"Estación de paletizado en RobotStudio (ABB) con HMI en TIA Portal, controlable por voz y visión artificial desde Python"** (Robles Postigo, 2024)

  Este trabajo desarrolla una estación de paletizado simulada en *RobotStudio* (ABB), que incorpora una HMI programada en *TIA Portal* y control por voz. Además, se incluye un sistema de visión artificial implementado en *Python* para identificar y clasificar cajas defectuosas. La comunicación entre los distintos sistemas se gestiona mediante TCP/IP y sockets.

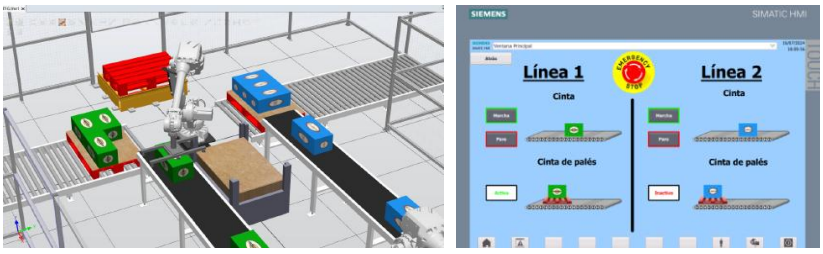


Figura 66. Estación e interfaz gráfica del TFG de Adrián Robles Postigo (Robles Postigo, 2024)

- **Fortalezas:** Incluye visión artificial realista, una interfaz avanzada y coordinación de múltiples plataformas.
- **Limitaciones:** Utiliza comunicación por sockets, menos estandarizada en entornos industriales frente a protocolos como OPC UA. Además, aunque la visión artificial implementada en *Python* es potente, presenta una menor integración con herramientas industriales habituales como *In-Sight Explorer* o Cognex, lo que puede dificultar su aplicación directa en entornos reales.

3.1.2. Trabajos en otras universidades españolas

En el ámbito académico español, también se han desarrollado Trabajos Fin de Grado que abordan temáticas similares a las de este proyecto, particularmente en lo relativo a la visión artificial aplicada a la robótica.

- **"Sistema de inspección visual utilizando Integrated vision de ABB"** (Piera Moreno, 2021)

  Este trabajo desarrolla un sistema robótico capaz de identificar y clasificar piezas geométricas (triángulos, círculos, cuadrados…) utilizando una cámara de visión artificial In-Sight 7402C integrada en *RobotStudio*. Una vez detectadas, las piezas son colocadas por el robot en la base correspondiente a su forma. El sistema se implementa tanto en simulación como en un entorno físico, abordando todas las fases del diseño, la calibración de la cámara y la programación en RAPID.
  - **Fortalezas:** Clasificación automatizada mediante visión artificial integrada e implementación dual (simulación y entorno real).
  - **Limitaciones:** Aplicación limitada a tareas estáticas de clasificación. Además, no se contempla integración con otros sistemas software ni protocolos de comunicación externos.

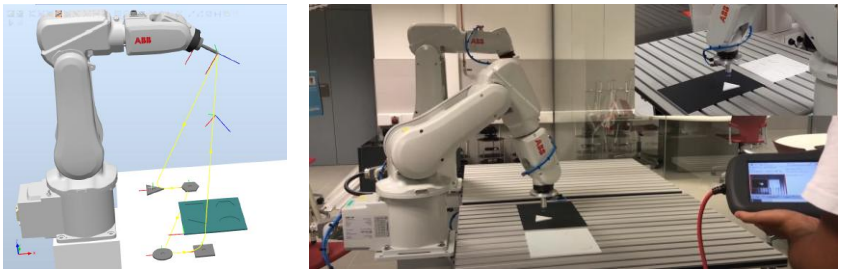


Figura 67. Estación simulada y su implementación en físico del TFG de Eros Piera Moreno. (Piera Moreno, 2021)

- **"Inspección visual utilizando Integrated Vision de ABB"** (Martín González, 2023)

  Este trabajo automatiza el clásico problema de las Torres de Hanoi empleando dos robots ABB IRB120 coordinados mediante visión artificial. Cada uno de los robots está asociado a un sistema de visión que permite determinar el estado inicial del tablero de juego y guiar el movimiento de los discos hasta alcanzar

el estado objetivo, todo ello cumpliendo las reglas del juego. Además, la solución se implementa también tanto en entorno simulado como real, desarrollando una interfaz gráfica de control.

- **Fortalezas:** La coordinación de los dos robots con visión artificial integrada, junto con la interfaz hombre-máquina (HMI) desarrollada en *ScreenMaker* para la interacción con el usuario, constituye una de las principales fortalezas del sistema.
- **Limitaciones:** La comunicación entre sistemas se basa en sockets, lo que resulta ser menos estándar en entornos industriales. Además, no se contempla la integración de plataformas externas, ya que se enfoca únicamente en una aplicación cerrada.

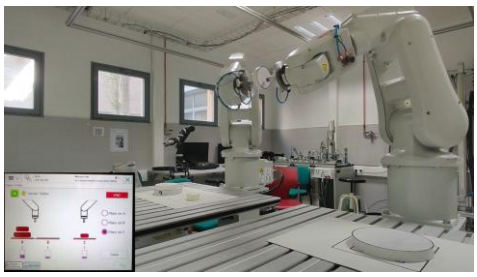


Figura 68. Estación de trabajo e interfaz gráfica del TFG de José Ángel Martín González (Martín González, 2023)

3.2. Análisis del problema

La revisión de trabajos previos revela avances significativos en la integración de visión artificial, simulación robótica y automatización, tanto en la Universidad de Valladolid como en otras universidades españolas. Sin embargo, estos estudios presentan limitaciones importantes que restringen su aplicación en entornos educativos e industriales avanzados.

Uno de los principales problemas identificados es el uso limitado de la visión artificial, que en la mayoría de los casos se limita a tareas básicas como la clasificación por forma o color. Apenas se abordan técnicas más avanzadas, como la lectura de códigos QR, la inspección dimensional, el recuento de patrones o la toma de decisiones basada en resultados visuales. Esta carencia reduce considerablemente el potencial técnico de las soluciones, especialmente en la industria moderna, donde se requieren sistemas automatizados más complejos, precisos y versátiles.

Además, se ha detectado una baja interoperabilidad entre las plataformas de software utilizadas. En gran parte de los casos, las soluciones se centran exclusivamente en herramientas como *RobotStudio* o combinaciones limitadas con algún software externo, lo que dificulta la replicación de arquitecturas completas. En el ámbito industrial, la conectividad robusta y flexible entre diversos entornos es esencial para desarrollar sistemas que sean realmente funcionales, lo que aún no se logra de manera efectiva en la mayoría de los trabajos existentes.

Finalmente, un aspecto crítico es la falta de protocolos de comunicación estandarizados, como OPC UA, lo que limita la escalabilidad y realismo de los sistemas. Aunque se usan TCP/IP o sockets, estas no ofrecen la interoperabilidad necesaria para integrar diversas plataformas de manera eficiente. Además, muchos proyectos adoptan enfoques cerrados, restringiendo su aplicabilidad y dificultando futuras ampliaciones o adaptaciones.

3.3. Solución implementada

Como respuesta a las limitaciones detectadas en trabajos previos, este Trabajo Fin de Grado presenta una solución integral orientada a la formación académica en el contexto de la Industria 4.0. La propuesta combina simulación, visión artificial, robótica colaborativa, comunicación industrial e interfaces gráficas, con el fin de ofrecer una experiencia formativa completa y realista.

- **Integración de visión artificial**

  Se han desarrollado seis prácticas que introducen técnicas de visión artificial más avanzadas como la clasificación de piezas, el recuento de patrones, la medición de distancias, la inspección dimensional y la lectura de códigos QR. Todas se realizan en un entorno simulado con cámaras emuladas configuradas en In-*Sight Explorer* e integradas en *RobotStudio*. Esta integración no solo introduce al alumnado en técnicas más avanzadas, sino que también les permite desarrollar competencias en análisis visual sin necesidad de disponer de hardware físico.

- **Simulación de robótica colaborativa**

  Las prácticas se llevan a cabo en estaciones virtuales desarrolladas en *RobotStudio* utilizando el modelo colaborativo del robot ABB GoFa. Estas estaciones simulan el funcionamiento realista de una célula automatizada. Este enfoque permite a los estudiantes comprender la dinámica y programación de sistemas robóticos industriales que implican visión artificial, sin riesgos ni equipos físicos.

- **Comunicación industrial mediante OPC UA**

  Se ha implementado una arquitectura de comunicación basada en el estándar OPC UA, utilizando *ABB IoT Gateway* como servidor y estableciendo comunicación bidireccional con *MATLAB*. Esto permite enviar comandos, consultar datos del sistema de visión y supervisar la estación en tiempo real.

- **Desarrollo de interfaces gráficas interactivas**

  Se han creado interfaces utilizando *MATLAB App Designer*, que permiten seleccionar tareas, monitorizar el sistema y controlar el robot de forma remota. Su diseño intuitivo facilita el aprendizaje y mejora la experiencia del usuario.

- Estructura modular y formativa

  La propuesta se organiza en seis prácticas independientes con un enfoque progresivo, permitiendo introducir los conceptos clave de manera gradual y facilitando la integración de tecnologías. Esta estructura modular favorece un aprendizaje activo, adaptado a distintos niveles de formación. Además, cada práctica ha sido documentada y validada mediante simulaciones, lo que asegura su reutilización en contextos educativos y su adaptabilidad a futuras actualizaciones.

Además de integrar tecnologías clave de la Industria 4.0, esta solución fomenta el aprendizaje práctico de los contenidos aprendidos de forma teórica a través de entornos realistas y dinámicos. Al basarse en simulación, resulta accesible desde cualquier lugar, lo que la hace adecuada también para formación online.

4. HERRAMIENTAS SOFTWARE EMPLEADAS

En este apartado se describen las herramientas utilizadas en el desarrollo del proyecto, destacando sus principales características y su aplicación en las prácticas docentes diseñadas.

Durante el proyecto se ha trabajado con el **robot colaborativo ABB CRB 15000 GoFa**, operado mediante un controlador **OmniCore C30**. También se dispone de una interfaz hombre-máquina llamada FlexPendant, conectada directamente al controlador. Esta interfaz táctil permite al operario configurar, visualizar y controlar el robot de forma intuitiva y sencilla.

Para la simulación, programación y supervisión offline del robot se ha utilizado *RobotStudio* 2024, el entorno desarrollado por ABB. Esta plataforma permite modelar estaciones robóticas virtuales, configurar posiciones y trayectorias, y validar su funcionamiento antes de la implantación física. En este entorno también se ha configurado la conexión con una cámara COGNEX, integrando su simulación mediante el software In-*Sight Explorer* 6.5.0, que emula el comportamiento de una cámara seleccionada y permite probar su lógica sin necesidad del hardware real.

Uno de los aspectos destacados del proyecto ha sido la creación de una interfaz gráfica en *MATLAB R2024b* utilizando *App Designer*. Para la comunicación se ha empleado el protocolo **OPC UA**, que permite la interoperabilidad entre las distintas plataformas de software utilizadas. En este caso, se ha utilizado el *ABB IoT Gateway* como servidor OPC UA, mientras que *RobotStudio* y *MATLAB* actuaron como clientes, permitiendo el envío y recepción de datos en tiempo real.

La siguiente figura ilustra gráficamente la arquitectura de comunicación implementada y los elementos software involucrados.

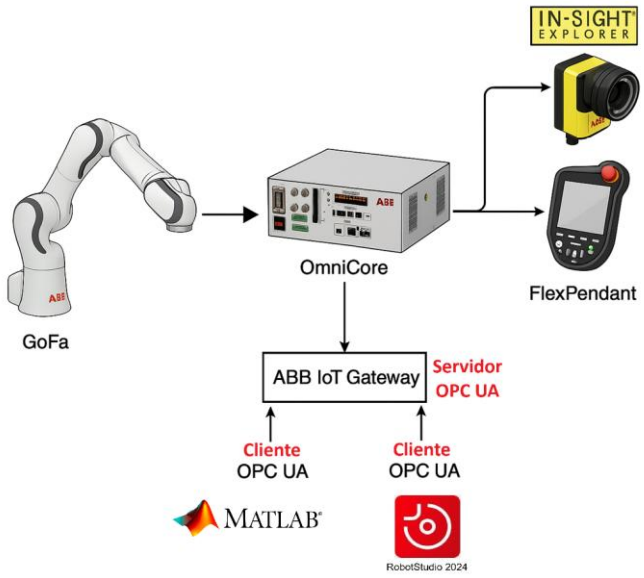


Figura 69. Arquitectura de comunicación implementada

4.1. RobotStudio 2024 (ABB)

RobotStudio es un software de simulación y programación offline desarrollado por ABB, diseñado para facilitar la configuración, programación y prueba de robots industriales y colaborativos en un entorno virtual. Siendo su principal ventaja el desarrollo y validación de programas de robots sin necesidad de disponer de un equipo físico, ahorrando así costos y reduciendo los riesgos operativos.

4.1.1. Funcionalidades relevantes

- **Simulación en tiempo real:** Permite visualizar y evaluar el comportamiento de un robot en un entorno virtual antes de ser implementado en físico.
- **Programación offline:** Hace posible la creación de programas sin interrumpir la producción en la plantando, optimizando así los tiempos de desarrollo.
- **Integración con controladores virtuales:** Permite replicar el comportamiento real del robot utilizando el mismo software que se ejecuta en los controladores físicos de ABB.
- **Importación de diseños CAD:** Permite la importación de modelos de componentes reales.
- **Generación de código en el lenguaje RAPID:** Este lenguaje puede transferirse directamente al robot físico.
- **Análisis de alcance y colisiones:** Proporciona herramientas que nos permiten identificar posibles colisiones antes de la implementación en físico.
- **Módulos adicionales y Add-ins:** RobotStudio cuenta con una serie de módulos especializados y complementos que amplían significativamente sus capacidades de simulación para aplicaciones industriales específicas, como soldadura, paletizado, pintura, ensamblaje o Pick and Place.

4.1.2. Interfaz gráfica

Al iniciar RobotStudio, el software abre automáticamente la pestaña "Archivo", que actúa como punto de partida para la creación o gestión de proyectos.

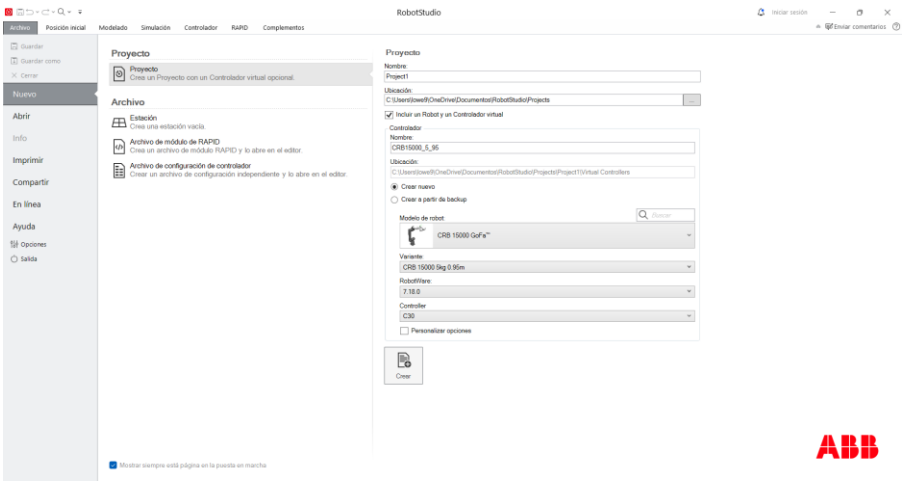


Figura 70. Interfaz gráfica inicial RobotStudio 2024

Desde aquí, el usuario puede crear un nuevo proyecto especificando el modelo de robot a utilizar y configurando un controlador virtual asociado, además de seleccionar la versión de *RobotWare* necesaria para garantizar compatibilidad con el equipo físico y definir la ubicación donde se guardará el proyecto. También ofrece la posibilidad de abrir estaciones o proyectos existentes, ya sean archivos previamente guardados o plantillas proporcionadas por ABB, así como acceder al menú de opciones para personalizar la configuración del entorno de trabajo según las necesidades del usuario.

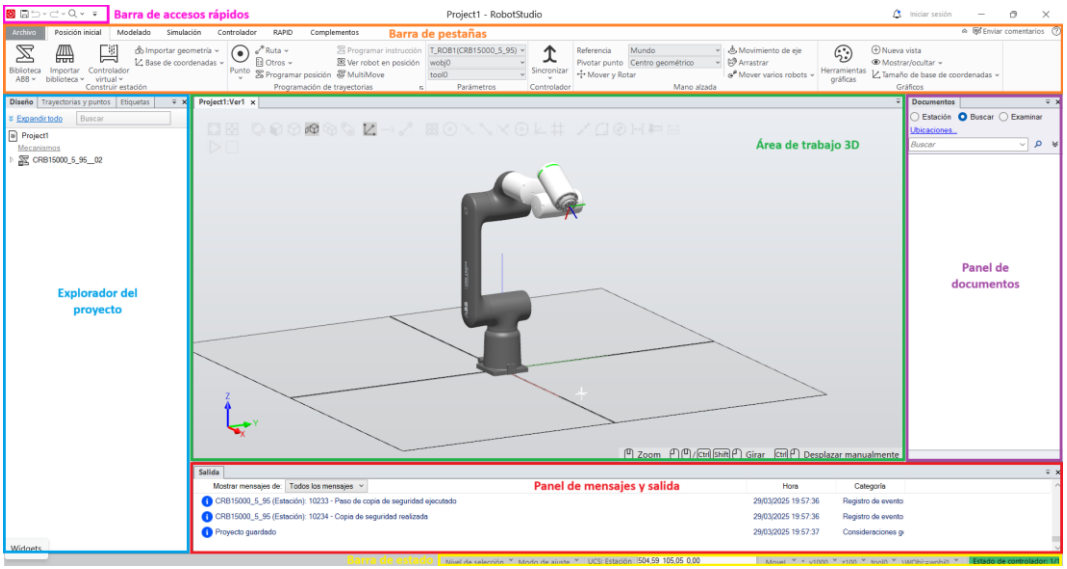


Figura 71. Entorno de trabajo en RobotStudio 2024

Al crear un nuevo proyecto, se accede al entorno principal de *RobotStudio*, diseñado para facilitar la programación y simulación robótica. Este espacio se organiza en varias áreas funcionales:

- **Barra de pestañas (naranja):** Se encarga de organizar las diferentes herramientas y funciones del software en categorías específicas. Cada pestaña agrupa comandos relacionados que facilitan la configuración, modelado, programación y simulación de robots.
  - **Posición inicial:** Esta pestaña contiene las herramientas esenciales para la creación de una estación de trabajo dentro de *RobotStudio*. Desde aquí se pueden importar bibliotecas y geometrías, programar posiciones, definir instrucciones y rutas, sincronizar con RAPID y ubicar el robot u otros objetos en posiciones específicas.

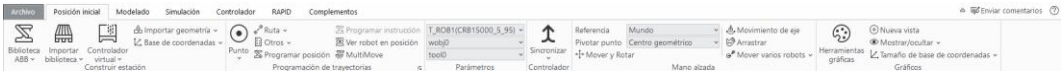


Figura 72. Pestaña Posición inicial en RobotStudio 2024.

Además, incluye un conjunto de herramientas gráficas que, al activarse, abren una pestaña adicional. Estas herramientas permiten: crear distintos puntos de vista en la simulación, especificar qué elementos deben ser visibles en la escena o incluso cambiar el tipo de proyección para mejorar la visualización del entorno.

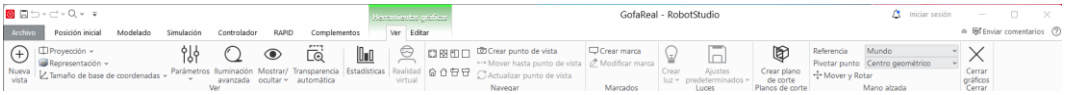


Figura 73. Pestaña Herramientas gráficas en RobotStudio 2024

- **Modelado:** Esta pestaña incluye herramientas para modificar y personalizar el entorno de trabajo en 3D. Algunas de sus características más importantes son: la importación y edición de modelos CAD para simular el entorno real, la creación y manipulación de geometrías personalizadas y herramientas para medir, mover y rotar los distintos objetos.

Figura 74. Pestaña Modelado en RobotStudio 2024

- **Simulación:** Esta pestaña se centra en la ejecución y análisis de simulaciones, permitiendo detectar colisiones, evaluar alcances y controlar en tiempo real el desarrollo de la simulación. Además, ofrece acceso a la lógica de la estación, facilita el análisis de señales y la simulación de entradas y salidas, y permite grabar la aplicación para su posterior revisión.

Figura 75. Pestaña Simulación en RobotStudio 2024

- **Controlador:** Esta pestaña se encarga de la gestión y configuración de los controladores de los robots. Desde aquí, se pueden añadir y gestionar controladores virtuales, realizar copias de seguridad, transferir archivos, configurar el sistema de entradas y salidas e interactuar con herramientas como la FlexPendant. Además, permite cambiar el modo de funcionamiento del robot y ajustar parámetros de seguridad, como la evitación de colisiones y la visión integrada.

Figura 76. Pestaña Controlador en RobotStudio 2024

- **RAPID:** Esta pestaña permite programar el robot utilizando el lenguaje RAPID de ABB. Sus principales herramientas son: insertar fragmentos o instrucciones predefinidas, probar y depurar el código o editarlo.

Figura 77. Pestaña RAPID en RobotStudio 2024

- **Complementos:** Aquí se encuentran paquetes adicionales que amplían las capacidades del software.

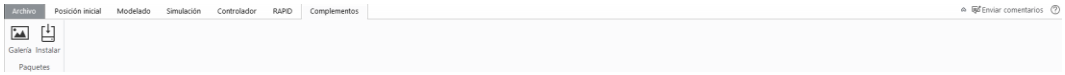


Figura 78. Pestaña Complementos en RobotStudio 2024

- **Barra de accesos rápidos (rosa):** Proporciona iconos para las operaciones más frecuentes, como guardar, deshacer o rehacer. Además, es personalizable, permitiendo añadir accesos directos a las funciones más utilizadas.
- **Explorador del proyecto (azul):** Permite gestionar todos los elementos de la estación de trabajo, como robots, herramientas, trayectorias y modelos importados, facilitando la organización del proyecto. Dependiendo de la pestaña que se encuentre activa, las opciones disponibles en este panel variarán para adaptarse a las funciones específicas de cada sección.
- **Área de trabajo 3D (verde):** Es el espacio principal donde se visualiza y manipula el entorno de simulación en tres dimensiones, permitiendo mover robots u objetos, ajustar posiciones y realizar pruebas visuales.
- **Panel de documentos (morado):** Proporciona acceso rápido a archivos relacionados con el proyecto, permitiendo buscar, examinar y gestionar documentos necesarios para la simulación y programación.
- **Panel de mensajes y salida (rojo):** Muestra información relevante sobre el estado del sistema, registros de eventos, errores y advertencias, facilitando la depuración y el control del proyecto.
- **Barra de estado (amarillo):** Muestra información en tiempo real sobre el estado del proyecto y el entorno de trabajo. Aquí aparecen los métodos de selección o ajuste, el UCS (*User Coordinate System*), datos sobre coordenadas, tipo de movimiento, velocidad, herramienta activa y el objeto de trabajo seleccionado.

4.1.3. FlexPendant

La FlexPendant es una interfaz hombre-máquina (HMI) portátil desarrollada por ABB, diseñada para la operación, programación y supervisión de robots industriales y colaborativos. Está conectada directamente al controlador del robot y permite al operario interactuar de forma rápida e intuitiva con el sistema. A través de ella, es posible realizar tareas como el control manual del robot, la edición de programas, la configuración de parámetros y posiciones, así como la gestión de señales de entrada y salida (E/S).

Además, su diseño ergonómico y su sistema operativo basado en Windows CE lo hacen un dispositivo adecuado tanto para entornos industriales como educativos.

Figura 79. FlexPendant Omnicore (ABB, User manual – FlexPendant, 2024)

Componentes principales

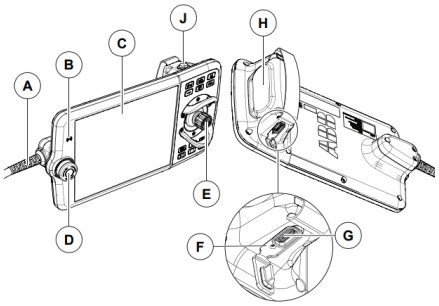


Figura 80. Componentes FlexPendant Omnicore (ABB, User manual – FlexPendant, 2024)

	Descripción
A	Conector
B	Lector RFID (funcionalidad aún no implementada)
C	Pantalla táctil
D	Dispositivo de parada de emergencia
E	Joystick
F	Botón de restablecimiento
G	Puerto USB
H	Dispositivo de habilitación de tres posiciones[3]
J	Botón de pulgar[4]

Tabla 7. Componentes FlexPendant Omnicore (ABB, User manual – FlexPendant, 2024)

Botonera y joystick

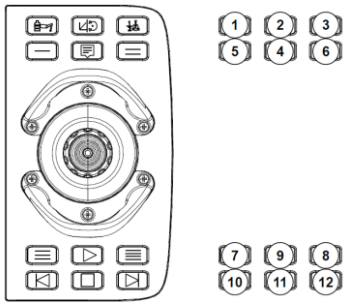


Figura 81. Botonera y joystick (ABB, 2024

[3] También conocido como botón del hombre muerto es un dispositivo de seguridad que permite controlar el movimiento del robot en tres modos: desactivado, baja velocidad y máxima velocidad. Su diseño asegura que el robot solo funcione cuando el operario mantiene presionado el botón, evitando movimientos accidentales y garantizando un control seguro en todo momento.

[4] En los robots colaborativos, este botón se emplea para activar la función de guiado. En aquellos robots que permiten operar en modo manual a máxima velocidad, el botón se utiliza como accionamiento mantenido.

Nº	Descripción
1	Botón de unidad mecánica
2	**Modo de movimiento 1**: Cambia entre movimiento lineal y reorientación.
3	**Modo de movimiento 2**: Alterna entre los ejes 1-3 y 4-6.
4	**Mensajes**: Mantener pulsado para captura de pantalla.
5,6, 7,8	**Botones personalizables por el usuario**
9	**INICIAR**: Ejecuta el programa.
10	**Paso atrás**: Ejecuta la instrucción anterior.
11	**STOP**: Detiene el programa.
12	**Paso adelante**: Ejecuta la siguiente instrucción.

Tabla 8. Descripción botonera FlexPendant Omnicore (ABB, 2024)

Movimientos del joystick

En el apartado Jog del FlexPendant, se puede seleccionar el tipo de movimiento que se desea realizar con el robot. Las opciones disponibles son: modo ejes 1-3 o 4-6, que permite mover los ejes del robot de forma independiente; modo lineal, que permite mover el robot de forma lineal en el espacio tridimensional; y modo reorientación, que rota la herramienta (TCP) sin afectar al resto de la posición del robot.

Al seleccionar cualquiera de estos modos, en la pantalla del FlexPendant se muestra una representación gráfica del robot, con una numeración del 1 al 3 y colores asignados a cada una de las posibilidades de movimiento. Además, se presenta una leyenda que indica los movimientos que deben realizarse con el joystick para llevar a cabo cada una de las operaciones.

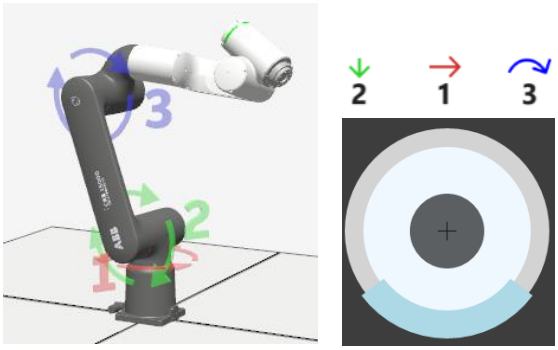


Figura 82. Ejemplo de movimientos del joystick para los ejes 1-3

Pantalla táctil

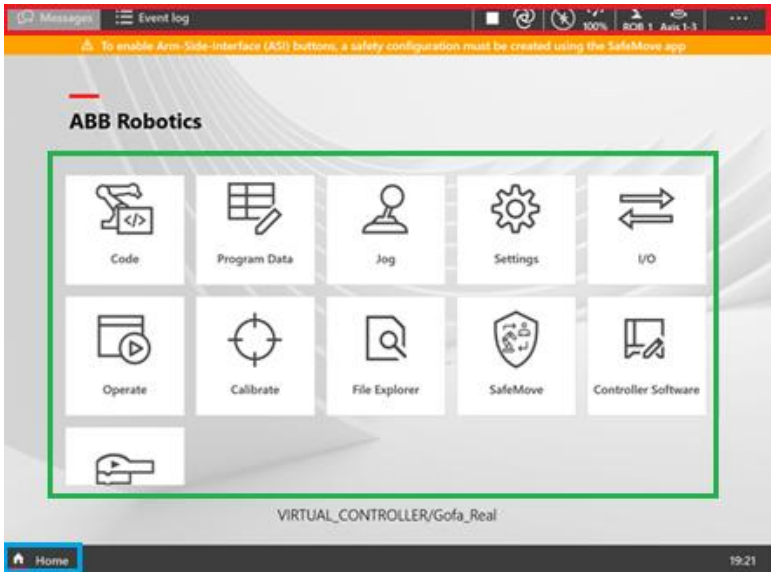


Figura 83. Interfaz FlexPendant Omnicore

- **Botón inicio (azul):** Permite volver a la pantalla principal desde cualquier ventana de la *FlexPendant*.
- **Botones de barra de estado (rojo):** Permiten acceder a los mensajes del operador, los registros de eventos y el panel de control, donde se pueden realizar ajustes rápidos, como el modo de funcionamiento, el estado de los motores o la velocidad.
- **Aplicaciones (verde):** La pantalla principal de la *FlexPendant* muestra todas las aplicaciones disponibles para el operario.

Program Data

Aplicación que permite visualizar y modificar los datos utilizados por el programa RAPID, como posiciones (*robtargets*), herramientas, sistemas de coordenadas (*wobjdata*), velocidades, constantes y otras variables. Facilita la configuración de estos parámetros sin necesidad de acceder directamente al código.

I/O

Esta aplicación permite visualizar y gestionar las redes de comunicación de entradas y salidas (E/S) configuradas en el sistema. En ella se muestran los diferentes buses o interfaces utilizados, como PROFINET, DeviceNet, EtherNet/IP, entre otros. Además, es posible visualizar los dispositivos conectados dentro de cada red, así como acceder a una lista completa de todas las señales de E/S configuradas en el sistema.

Code

Aplicación que permite navegar entre los distintos módulos de RAPID y editarlos directamente desde la *FlexPendant*. Desde ella, el usuario puede añadir o modificar

instrucciones, editar el código y utilizar herramientas de depuración para ejecutar rutinas paso a paso, revisar variables y detectar posibles errores en el programa del robot.

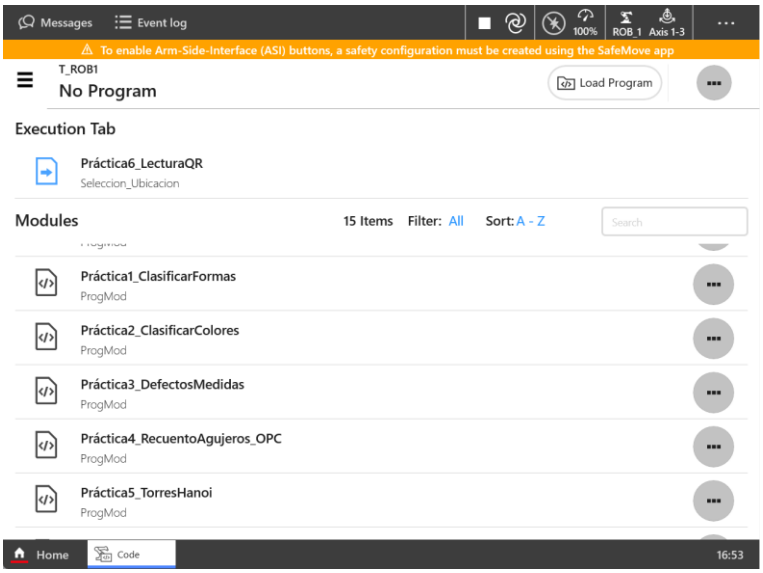


Figura 84. Pantalla de la aplicación Code

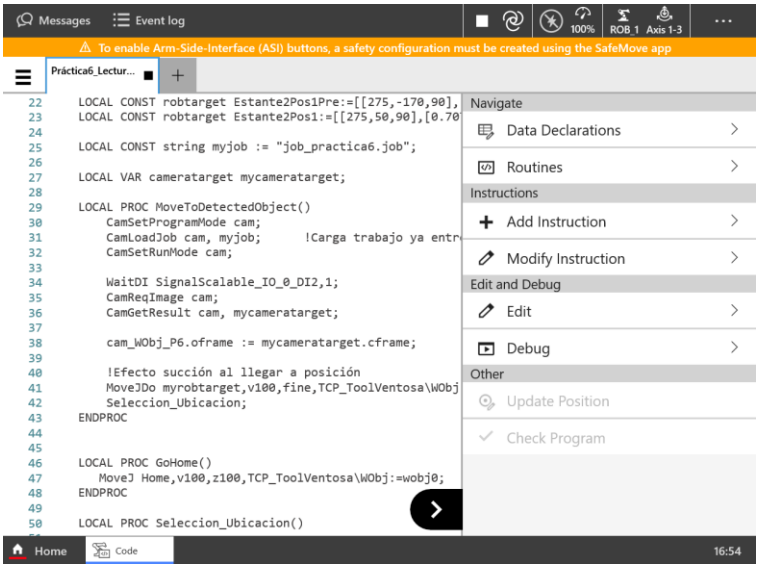


Figura 85. Visualización del programa RAPID activo

Jog

Aplicación utilizada para mover manualmente el robot. Permite seleccionar distintos modos de desplazamiento: por ejes individuales (modo articulado), en coordenadas cartesianas (modo lineal) o mediante sistemas de coordenadas definidos por el usuario o la herramienta. Para su utilización, el robot debe estar en modo manual y con los motores activados. Además, es posible ajustar la velocidad de movimiento y visualizar en tiempo real los valores de posición de los ejes.

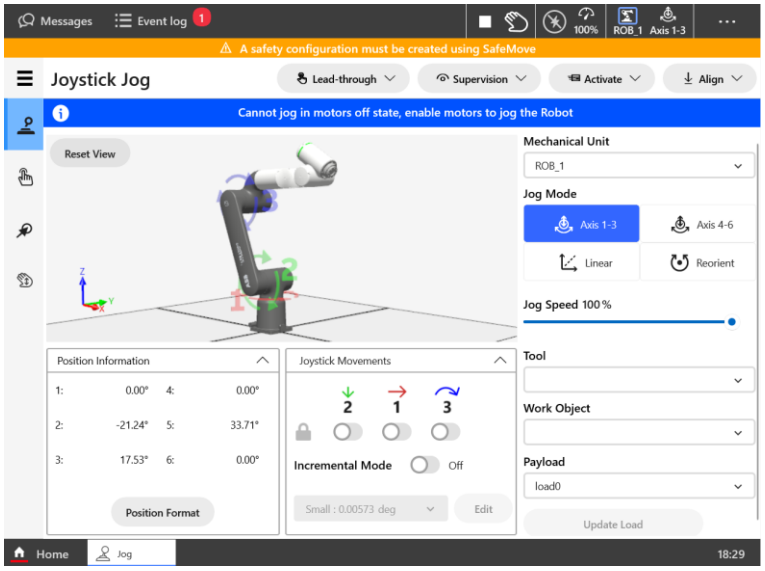


Figura 86. Pantalla de la aplicación Jog

Además, en la parte superior de la aplicación se encuentran cuatro funciones relevantes:

- ***Lead-Through***: Permite guiar el robot manualmente para enseñarle las posiciones, grabando los puntos y movimientos realizados.
- ***Supervision***: Activa un modo en el que el robot está pendiente de posibles colisiones o excesos de fuerza mientras ejecuta una trayectoria.
- ***Activate***: Sirve para activar una trayectoria que ha sido definida previamente.
- ***Align***: Permite alinear el robot con una posición o trayectoria ya programada, garantizando así una correcta ejecución.

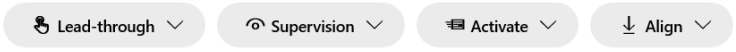


Figura 87. Funcionalidades relevantes aplicación Jog

Operate

Esta aplicación permite visualizar, gestionar y ejecutar programas. Desde esta interfaz, es posible cargar un programa, reiniciarlo estableciendo el puntero en la rutina principal (*main*), así como actualizar posiciones específicas y visualizar en todo momento la ubicación del puntero del programa durante su ejecución.

Además, la aplicación ofrece la posibilidad de crear *dashboards* personalizados para un control más visual del proceso, así como ejecutar rutinas de servicio predefinidas, facilitando tareas de mantenimiento y operación del sistema.

Settings

Esta aplicación incluye las configuraciones generales del sistema. Desde ella es posible cambiar el nombre del robot, configurar la red Ethernet del controlador, gestionar los sistemas conectados a la plataforma **ABB *Ability***, realizar copias de

seguridad del sistema, restaurar configuraciones anteriores y acceder a registros de eventos, alarmas y herramientas de diagnóstico.

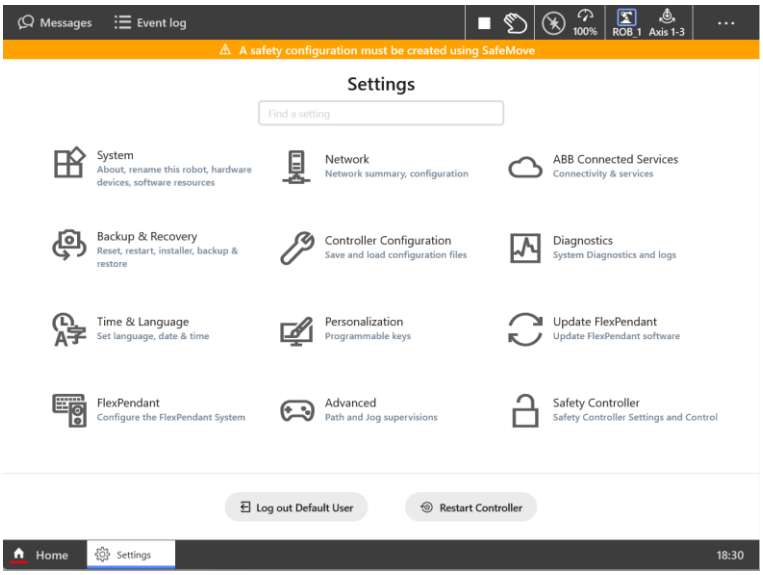


Figura 88. Pantalla de la aplicación Settings

Además, dentro de esta sección se encuentra la opción ***Personalization***, que permite configurar teclas programables y personalizar accesos rápidos, con el objetivo de mejorar la experiencia del usuario.

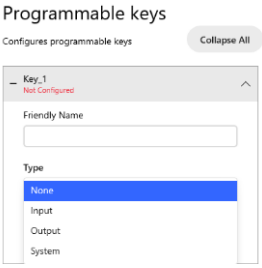


Figura 89. Personalización teclas programables

Calibrate

Esta aplicación proporciona acceso a las herramientas de calibración del sistema. Su función principal es permitir la calibración de los ejes del robot, garantizando precisión y una correcta referencia espacial dentro de su entorno de trabajo. Esta calibración puede llevarse a cabo mediante dos métodos: definiendo una base de coordenadas o utilizando los valores almacenados en la memoria interna del robot, lo que facilita una recalibración rápida en caso de reinicio o mantenimiento.

File Explorer

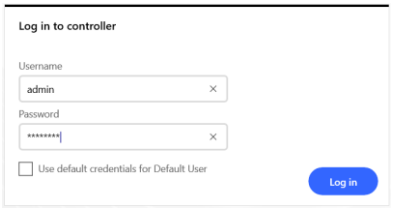
File Explorer es el gestor de archivos interno del controlador del robot. Esta herramienta permite navegar por la estructura de carpetas del sistema, así como copiar, pegar, eliminar, renombrar o cargar programas almacenados tanto en la

memoria interna del controlador como en dispositivos de almacenamiento externos (USB o red).

SafeMove

Aplicación específica para gestionar las funciones de seguridad del robot. Desde esta interfaz es posible configurar zonas seguras, limitar velocidades de movimiento, definir el volumen operativo del robot y de la herramienta (*tool*), así como sincronizar el sistema con dispositivos externos de seguridad, como escáneres láser o PLCs de seguridad.
Esta funcionalidad es fundamental para garantizar un entorno de trabajo seguro, ya que permite que el operario interactúe con el robot sin comprometer su integridad física ni la del sistema.

Para llevar a cabo la configuración de las funciones de seguridad del robot, es necesario iniciar sesión con un usuario que disponga de los permisos adecuados. ABB proporciona un usuario con credenciales predefinidas específicamente para este tipo de configuraciones:



Usuario: admin
Contraseña: robotics

Figura 90. Credenciales predefinidas ABB RobotStudio

Este acceso permite modificar parámetros sensibles relacionados con la seguridad, por lo que su uso debe estar restringido a personal autorizado.

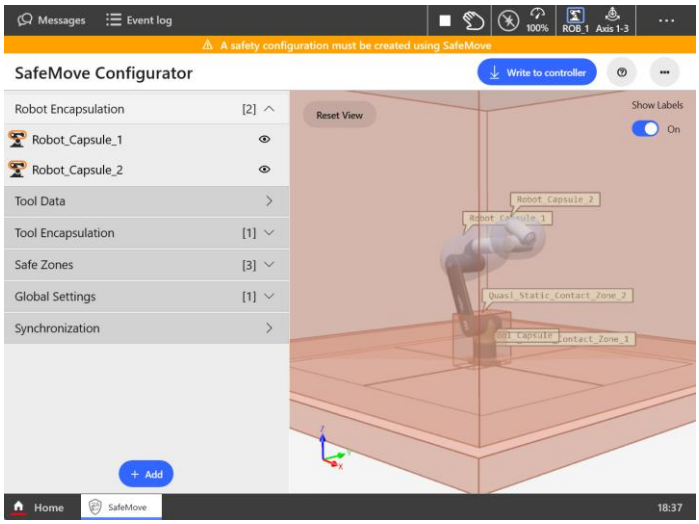


Figura 91. Pantalla de configuración SafeMove

Controller Software

Esta aplicación proporciona información sobre las versiones del sistema operativo del robot, módulos instalados y licencias activas. También se pueden actualizar componentes o instalar nuevas funcionalidades (*Add-Ins*).

Wizard

Wizard es una herramienta de programación gráfica basada en bloques, diseñada para facilitar la creación de programas de forma sencilla, visual e intuitiva, sin requerir conocimientos previos del lenguaje RAPID.

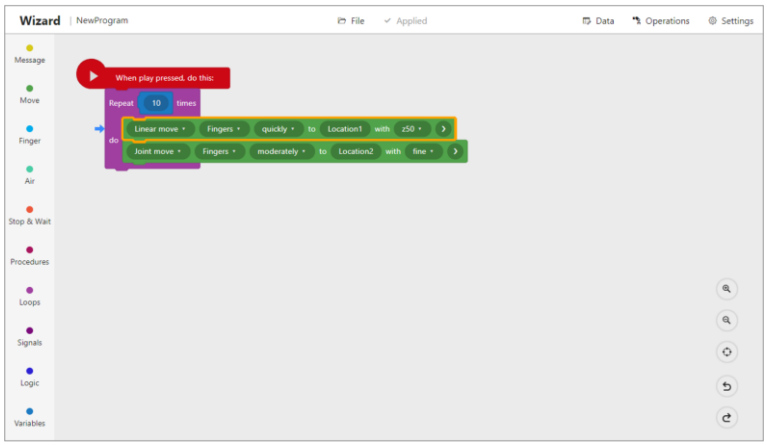


Figura 92. Interfaz de programación Wizard (ABB, Release Notes - Wizard 1.6, 2023)

El desarrollo de un programa se realiza mediante la técnica de arrastrar y soltar bloques predefinidos, los cuales están organizados por categorías como movimiento, paradas y esperas, bucles, señales, entre otros. Cada bloque puede ser personalizado mediante parámetros básicos, lo que permite adaptar su comportamiento a las necesidades de la aplicación.

4.1.4. Integrated visión interface

La Interfaz de Visión Integrada es una solución disponible en los robots ABB con controlador Omnicore o IRC5, que se gestiona a través del entorno de programación RobotStudio. Esta funcionalidad combina hardware y software para integrar de manera sencilla sistemas de visión artificial en las aplicaciones y la programación del robot.

Gracias a las herramientas gráficas basadas en Cognex *EasyBuilder*, los usuarios pueden realizar tareas de detección, localización e inspección de piezas utilizando cámaras que se integran directamente con el controlador del robot. Además, el controlador incluye instrucciones predefinidas del lenguaje RAPID para trabajar con visión artificial, Esto incluye comandos específicos y una interfaz predefinida para la comunicación con cámaras.

El sistema de visión está basado en las cámaras Cognex In-Sight, específicamente de la serie 7000. Es posible conectar hasta tres cámaras al sistema, alimentadas directamente desde el controlador con 24V y Ethernet.

Aunque esta funcionalidad ofrece una amplia gama de aplicaciones, tales como localización y posicionamiento de piezas, inspección de calidad, verificación de presencia y lectura de códigos, también presenta algunas limitaciones, como las siguientes:

- Las cámaras deben conectarse exclusivamente a la red privada del controlador.
- Los programas de visión previamente creados con Cognex *EasyBuilder* podrían necesitar ajustes para funcionar correctamente dentro del entorno ABB.

Interfaz gráfica visión de RobotStudio

La interfaz de Visión Integrada en RobotStudio se encuentra en la pestaña de Visión. Esta proporciona un entorno gráfico intuitivo desde el que se puede configurar completamente la cámara y sus herramientas asociadas.

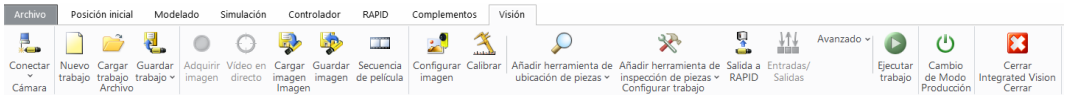


Figura 93. Barra de herramientas de la Interfaz de Visión Integrada en RobotStudio.

Cámara	• Conectar/Desconectar la cámara. • Configurar la red y permisos de usuario. • Añadir sensores físicos.
Archivo	• Crear o cargar trabajos de visión. • Guardar configuraciones.
Imagen	• Adquirir imagen. • Activar video en directo. • Cargar/guardar imágenes y grabar secuencias.
Configuración del trabajo	• Configurar imagen y calibración. • Añadir herramientas de ubicación e inspección de piezas. • Configurar entradas/salidas y generación de datos para RAPID.
Producción	• Cambio de modo de operación

Tabla 9. Funcionalidades interfaz Visión Integrada

4.2. IoT Gateway (ABB)

El *IoT Gateway de ABB* es una herramienta de software diseñada para conectar los controladores de robots de ABB con aplicaciones o plataformas externas. Su función principal es facilitar la interoperabilidad entre los sistemas robóticos y herramientas externas, permitiendo el intercambio de información en tiempo real a través de protocolos industriales abiertos, como OPC UA.

Este *gateway* actúa como un servidor OPC UA, exponiendo variables internas del robot (como posiciones, velocidades y E/S) para que puedan ser leídas o escritas por clientes externos que cumplan con dicho estándar. De esta forma, se consigue una comunicación fluida y bidireccional entre el controlador del robot y los sistemas conectados.

4.2.1. Funcionalidades relevantes

- **Publicación de variables RAPID como nodos OPC UA**: Permite acceder a variables internas del robot desde sistemas externos.
- **Monitoreo del estado del robot:** Proporciona datos en tiempo real sobre el funcionamiento del robot.
- **Configuración de datos accesibles:** Define qué variables estarán disponibles para lectura/escritura.
- **Arquitectura cliente-servidor**: Establece una conexión entre el controlador del robot y las herramientas externas.

4.2.2. Interfaz gráfica

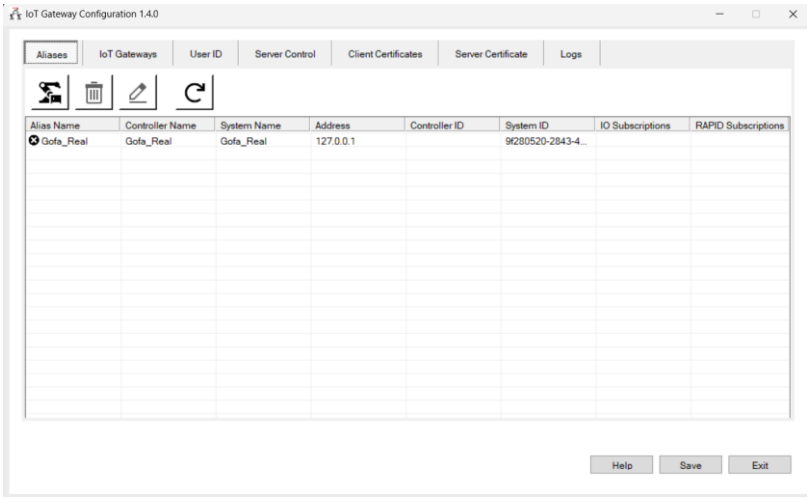


Figura 94. Interfaz IoT Gateway

La interfaz cuenta con varias pestañas para la configuración y gestión del sistema:

- ***Aliases*:** En esta pestaña se muestran los alias configurados para los controladores de los robots conectados. Los alias se utilizan para identificar cada robot, teniendo en cuenta su dirección IP, los identificadores únicos del controlador y sistema, así como el número de variables suscritas (*subscriptions*).
- ***IoT Gateways*:** Desde esta pestaña se puede añadir, configurar o eliminar instancias de *IoT Gateway*, pudiendo elegir entre los modos OPC UA (cliente-servidor) o MQTT Publisher (publicación en tópicos IoT).
- ***User* ID:** Permite establecer o gestionar las credenciales de acceso para los clientes OPC UA. Esto asegura que solo usuarios autorizados puedan interactuar con el sistema de forma remota.

- ***Server Control***: En esta pestaña se pueden iniciar, detener o reiniciar el servidor. Además, ofrece opciones para configurar el número de puerto y el nombre del servidor.
- ***Client Certificates***: Aquí se gestionan los certificados de los clientes que intenten conectarse al *IoT Gateway*. Es posible aceptar o rechazar nuevos certificados, revisar la lista de certificados ya autorizados o importar uno nuevo.
- ***Server Certificate***: Esta pestaña permite gestionar los certificados digitales del servidor. Desde aquí se pueden crear o importar certificados firmados o exportarlos para su uso en otros sistemas.
- ***Logs***: En esta pestaña se encuentran los registros de eventos del sistema, lo que permite revisar la actividad del sistema, como las conexiones con clientes, los errores del servidor y el estado de las suscripciones

4.3. In-Sight Explorer 6.5.0 (Cognex)

In-Sight Explorer es el software oficial de Cognex, diseñado específicamente para entornos industriales. Permite configurar, emular y programar las cámaras inteligentes de la serie *In-Sight*. Esta herramienta facilita la creación, prueba y validación de trabajos de visión artificial de manera virtual, lo que permite desarrollar soluciones sin necesidad de tener la cámara físicamente presente durante las fases iniciales de programación.

El software se utiliza en una amplia gama de aplicaciones industriales, como la inspección de piezas, la detección y localización de objetos y el guiado de robots, contribuyendo a optimizar procesos de producción y automatización.

4.3.1. Funcionalidades relevantes

El software ofrece diversas funcionalidades clave para la integración y configuración de sistemas de visión Cognex. Entre sus principales características se encuentran:

- **Emulación de cámara:** Permite simular el comportamiento de una cámara Cognex real, incluyendo la adquisición de imágenes, la configuración de la cámara y la generación de resultados, que facilitan la puesta en marcha de aplicaciones sin necesidad de hardware físico.
- **Gestión de la comunicación:** Permite configurar la dirección IP de la cámara y definir cómo se enviarán los resultados, soportando protocolos como OPC o FTP para la transferencia de datos.
- **Integración con sistemas externos:** Los resultados obtenidos por la cámara pueden ser enviados a través de *Ethernet* a sistemas de control como PLCs o robots industriales, utilizando protocolos estándar como TCP/IP o Profinet.
- ***EasyBuilder®***: Un entorno visual que permite configurar herramientas de visión de manera sencilla, sin requerir conocimientos avanzados de programación, gracias a sus asistentes fáciles de usar.

4.3.2. Interfaz EasyBuilder

In-Sight Explorer ofrece una interfaz gráfica estándar, especialmente diseñada para facilitar la creación de tareas de visión, tanto para usuarios con pocos conocimientos de programación como para aquellos que necesiten soluciones rápidas y eficientes. Su diseño es intuitivo y permite configurar y ajustar herramientas de visión sin necesidad de escribir código, lo que acelera el proceso de desarrollo.

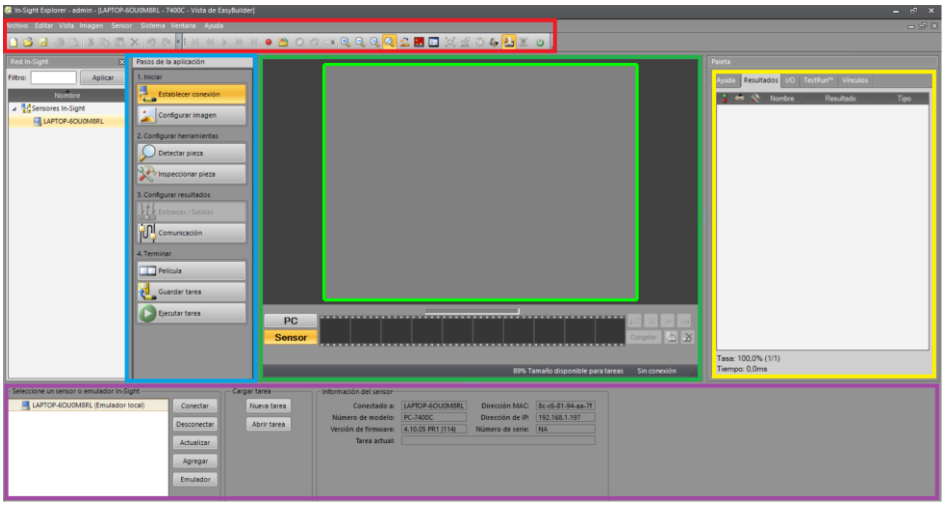


Figura 95. Interfaz EasyBuilder

- **Barra de herramientas (rojo):** Esta sección incluye dos barras: la barra de menús, que contiene pestañas como Archivo, Editar, Vista, Imagen, etc., y la barra de acceso rápido, donde se encuentran funciones como capturar imágenes, abrir, crear o guardar tareas, herramientas de zoom…
- **Panel de navegación (azul):** En esta zona se encuentran los pasos necesarios para crear una aplicación de visión. Desde la conexión al dispositivo hasta la creación, guardado y ejecución de la tarea.
- **Área de imagen (verde):** Es la zona donde se muestra la imagen capturada, ya sea en tiempo real o emulada. Sobre esta área se aplican las herramientas de ubicación e inspección, y se visualizan los resultados gráficos obtenidos.
- **Panel de resultados (amarillo):** Muestra los resultados obtenidos de cada herramienta, como coordenadas detectadas, valores de medición, estado…
- **Panel de sensor (morado):** Aquí se configura la conexión con la cámara, ya sea real o emulada. Permite seleccionar el dispositivo, conectar a una cámara física o activar el modo emulador, según sea necesario.

4.3.3. Interfaz Spreadsheet View

El modo *Spreadsheet View* presenta una interfaz basada en una hoja de cálculo, pensada para usuarios con conocimientos avanzados en visión artificial. A diferencia del entorno *EasyBuilder*, este modo no guía al usuario de forma paso a paso, sino que ofrece un acceso directo y detallado a todas las herramientas, funciones y fragmentos disponibles para crear tareas de visión.

Se trata de una interfaz avanzada y versátil, que ofrece un control más preciso y personalizado para el desarrollo de aplicaciones complejas o especializadas.

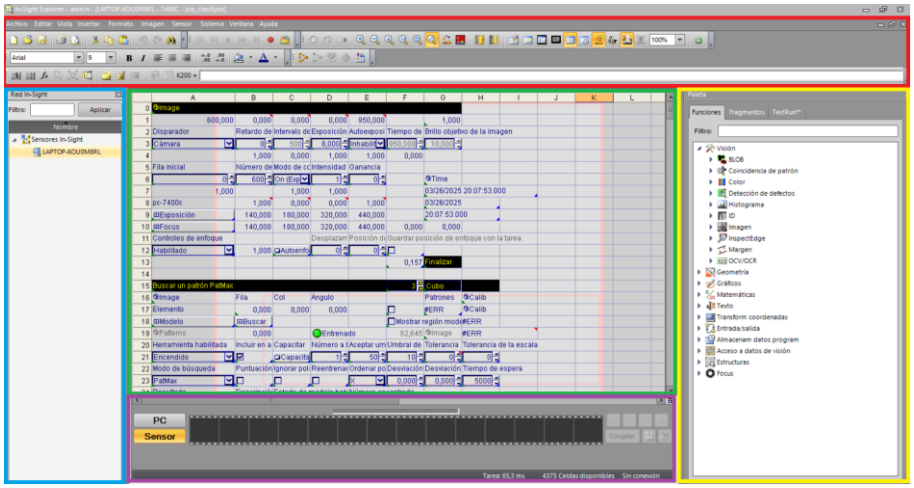


Figura 96. Interfaz Spreadsheet View

- **Barra de herramientas (rojo):** Al igual que en *EasyBuilder*, este modo también incluye dos barras: la barra de menús y la barra de acceso rápido, que proporciona iconos específicos para funciones adicionales. Estas funciones son similares a las del otro modo, pero incluyen opciones exclusivas de esta sección, como ajustes de transparencia, opciones de vista, gráficos, configuraciones de fuentes y alineación, así como la capacidad de realizar referencias a celdas específicas.
- **Hoja de cálculo (verde):** Es la parte central de esta vista, donde se gestionan y visualizan todas las funciones, parámetros y datos necesarios para desarrollar la tarea. Los usuarios pueden manipular directamente cualquier celda, introduciendo valores y modificando o insertando condiciones o expresiones.
- **Panel del sensor (azul):** Muestra los sensores In-Sight disponibles, permitiendo su fácil acceso y configuración.
- **Panel de navegación (amarillo):** En esta sección se encuentran las funciones disponibles para el usuario, como las de visión, geometría, gráficos, matemáticas, E/S, entre otras. También incluye fragmentos predefinidos de funciones que facilitan su rápida implementación y una pestaña llamada *TestRun*, que permite probar y validar las tareas configuradas.
- **Panel de control de adquisición (morado):** Permite seleccionar la fuente de imagen (PC o Sensor), navegar entre capturas anteriores mediante una tira de imágenes, y controlar la reproducción en tiempo real del sistema de visión mediante botones de ejecución, pausa, retroceso o avance.

4.4. MATLAB R2024b (MathWorks)

Según la página oficial de **MathWorks**:

> *"MATLAB es una plataforma de programación y cálculo numérico utilizada por millones de ingenieros y científicos para analizar datos, desarrollar algoritmos y crear modelos." (MathWorks, MATLAB, s.f.)*

Este software se ha consolidado como un entorno estándar en sectores como la ingeniería, la investigación científica, la automatización industrial e incluso la docencia, permitiendo expresar ideas técnicas de forma directa, clara y eficiente. *MATLAB* combina un lenguaje de alto nivel con un entorno interactivo que permite trabajar con matrices, algoritmos, análisis de datos y visualización gráfica.

Una de sus mayores ventajas es su amplia y diversa librería de *toolboxes*, que cubren áreas especializadas, desde el procesamiento de señales e imágenes, hasta sistemas de control, comunicaciones inalámbricas, robótica, y *deep learning*. Estas herramientas permiten a los usuarios personalizar sus entornos de trabajo según las necesidades específicas de sus proyectos. (Corke, Jachimczyk, & Pillat, 2023)

4.4.1. Funciones relevantes

Figura 97. Prestaciones de MATLAB (MathWorks, MATLAB, s.f.)

MATLAB incluye numerosas funcionalidades, entre las que destacan:

- **Cálculo numérico y algebra matricial:** Este entorno está diseñado para realizar operaciones matemáticas complejas, como álgebra lineal, operaciones matriciales, interpolación y transformadas.
- **Visualización de datos:** El entorno permite crear gráficos tanto en 2D como en 3D, con amplias opciones de personalización. Es ideal para representar resultados de simulaciones, análisis estadísticos, o para crear animaciones y visualizaciones interactivas que faciliten la interpretación de los datos.
- ***Toolboxes* especializadas:** Son extensiones específicas que permiten afrontar problemas específicos sin necesidad de desarrollar funciones desde cero.

Algunos de estos campos son: comunicaciones industriales (como OPC *Toolbox*), robótica y visión artificial, control automatizado…

- **Conectividad con otros entornos:** *MATLAB* facilita la interacción con dispositivos y sistemas externos a través de protocolos industriales como OPC UA, TCP/IP o APIs. Además, es compatible con otros entornos y lenguajes, como *Simulink*, *Python*, C/C++, y sistemas embebidos.
- **Diseño aplicaciones interactivas (*App Designer*):** Es una herramienta que facilita la creación de interfaces personalizadas, combinando elementos visuales como botones, gráficos y sliders con código funcional para desarrollar aplicaciones completas.

4.4.2. Interfaz gráfica Matlab

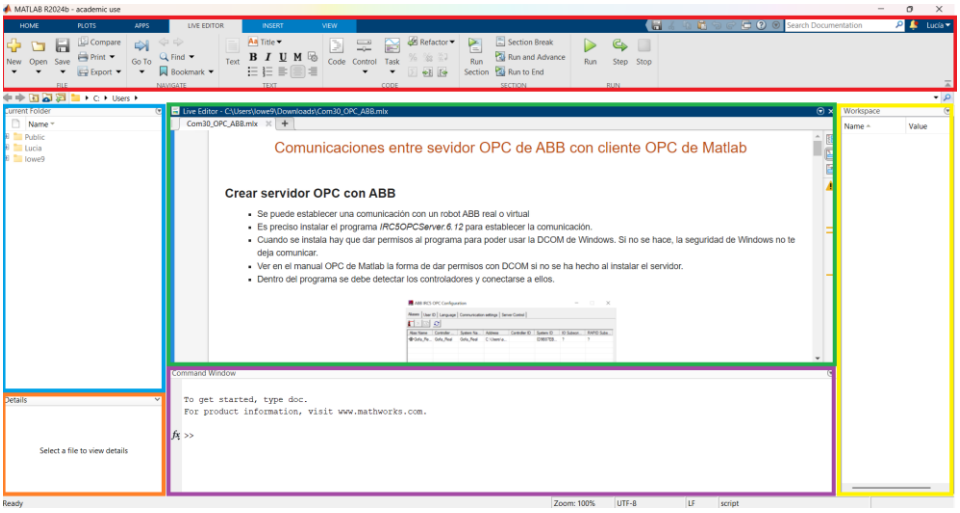


Figura 98. Interfaz gráfica MATLAB R2024b

La ventana principal del software es personalizable, pudiéndose cambiar el tema, redimensionar los paneles y acoplarse/desacoplarse los paneles. por lo que no siempre tiene la misma apariencia. Entre los apartados encontramos estos:

- **Barra de herramientas (rojo):** En la parte superior de la interfaz se encuentran las distintas pestañas por las que se puede navegar. Cada una de ellas cuenta con herramientas específicas para llevar a cabo las tareas.

HOME

Home es la pestaña principal y contiene las herramientas generales de *MATLAB*. Desde aquí, puedes crear o abrir scripts, importar datos, gestionar variables, ejecutar código, limpiar la consola, ajustar *layout* y acceder a *Add-Ons* para descargar extensiones adicionales.

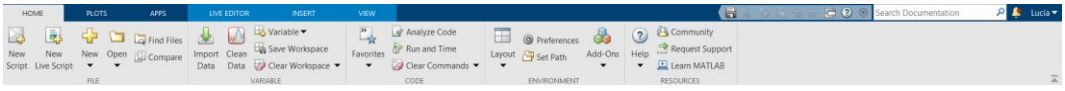


Figura 99. Pestaña HOME en MATLAB R2024b

PLOTS

Esta pestaña se enfoca en la creación y personalización de gráficos. Desde aquí, puedes generar gráficos 2D y 3D, ajustarlos y gestionar su apariencia, lo cual es esencial para un análisis visual efectivo de los datos.

Figura 100. Pestaña PLOTS en MATLAB R2024b

APPS

Esta pestaña da acceso a herramientas especializadas para tareas avanzadas. Incluye accesos directos a las *toolboxes* instaladas por defecto, como *App Designer*, OPC *Data Access Explorer*...

Figura 101. Pestaña APPS en MATLAB R2024b

Pestañas adicionales

Además, dependiendo de la tarea que se esté realizando, se mostrarán pestañas adicionales con herramientas más específicas. Al abrir un script, se activan las pestañas *Publisher, Editor* y *View* para editar y organizar el código. Por otro lado, al abrir un *live script*, se muestran las pestañas *Live Editor, Insert y View*, que permiten trabajar con código interactivo, insertar gráficos y ecuaciones, y personalizar la visualización.

- **Editor (verde):** Es la zona central de la interfaz, donde se escriben y editan los programas. Aquí se pueden trabajar con scripts, funciones o *live scripts*, que permiten incluir, además de código, ecuaciones, imágenes y gráficos dentro del mismo archivo.
- **Ventana de comandos (morado):** Esta ventana permite introducir instrucciones directamente para su ejecución inmediata, sin necesidad de guardar un archivo nuevo. Desde ella, se puede acceder a diferentes partes de la interfaz, aplicaciones y *toolboxes*. Además, los resultados se muestran aquí cuando no se asignan a variables.
- **Espacio de trabajo (amarillo):** En este panel se muestran las variables activas que están almacenadas en la memoria, indicando tanto su nombre como su valor.
- **Visor de archivos (azul):** Este panel muestra el directorio de trabajo actual, lo que permite gestionar los archivos y carpetas asociadas al proyecto. Desde aquí, se puede navegar por la estructura de carpetas, abrir scripts, copiar archivos, entre otras acciones. Es importante destacar que *MATLAB* solo puede ejecutar archivos desde el directorio actualmente seleccionado.

- **Panel de detalles (naranja):** Panel en el que se muestra información adicional del archivo seleccionado por el usuario.

4.4.3. Interfaz gráfica App Designer

Design View

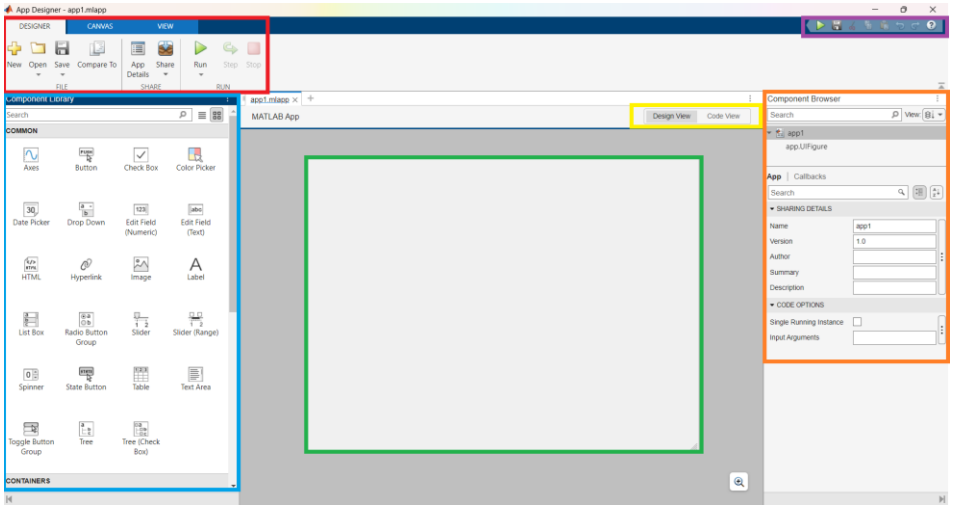


Figura 102. Interfaz gráfica App Designer (Design View)

- **Barra de herramientas (rojo):** Esta sección contiene las pestañas principales, cada una con accesos rápidos específicos según su función.

DESIGNER

Esta pestaña se centra en la gestión del proyecto y la ejecución de la aplicación. Incluye funciones como abrir o guardar archivos, comparar diferentes aplicaciones para detectar cambios y controlar la ejecución (ejecutar, paso a paso o detener la app).

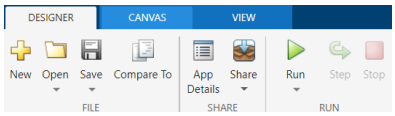


Figura 103. Pestaña DESIGNER en MATLAB App Designer

CANVAS

Esta pestaña se enfoca en personalizar la apariencia del área de diseño. Aquí se encuentran herramientas para alinear componentes, ajustar su tamaño, definir márgenes, establecer espaciado entre elementos y gestionar la superposición de objetos.

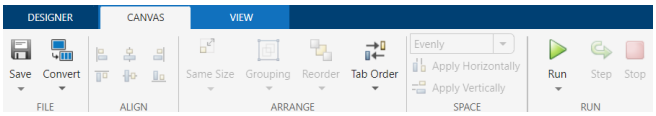


Figura 104. Pestaña CANVAS en MATLAB App Designer

VIEW

Esta pestaña permite configurar la visualización del entorno de trabajo para facilitar el diseño. Incluye opciones como mostrar u ocultar la cuadrícula, ajustar la vista o hacer zoom en el área de diseño.

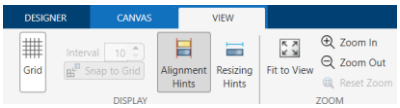


Figura 105. Pestaña VIEW en MATLAB App Designer

- **Panel de componentes (azul):** Contiene todos los elementos que se pueden insertar en la interfaz, desde controles básicos como botones, listas desplegables y *sliders*, hasta contenedores que ayudan a organizar visualmente los componentes.
- **Área de diseño (verde):** Es la zona central donde se construye la interfaz gráfica. Aquí se colocan y ajustan los componentes arrastrados desde el panel, definiendo la estructura y disposición visual de la aplicación.
- **Selector de vistas (amarillo):** Permite alternar entre la *Design View*, para diseñar visualmente la interfaz, y la *Code View*, donde se implementa la lógica del programa mediante eventos, *callbacks* y funciones.
- **Navegador de componentes (naranja):** Muestra la estructura jerárquica de todos los elementos insertados en la aplicación. También permite ver y modificar propiedades del componente seleccionado, como el nombre, etiqueta o valores.
- **Barra de accesos rápidos (morado):** Incluye botones para ejecutar, guardar, copiar, cortar, pegar y deshacer/rehacer acciones en la aplicación.

Code View

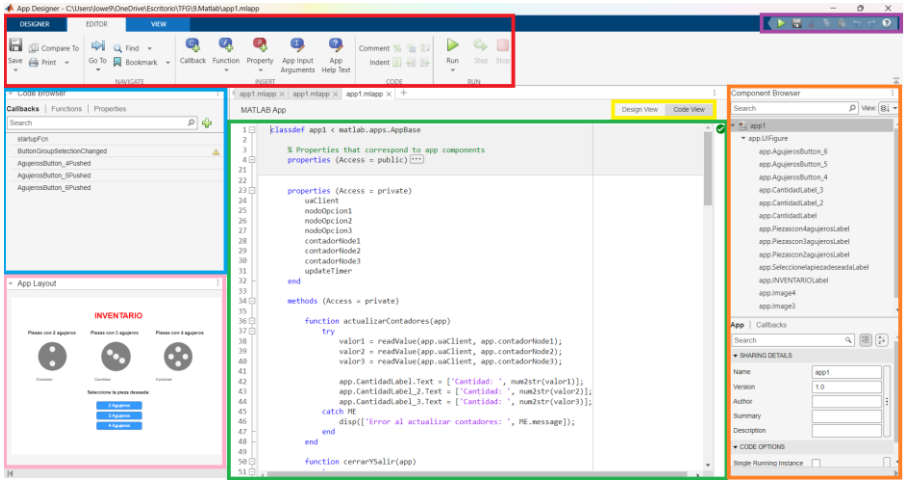


Figura 106. Interfaz gráfica App Designer (Design View)

La vista de código coincide con la de diseño, en la barra de herramientas, selector de vistas, navegador de componentes y barra de accesos rápidos. El resto de ventanas son:

- **Editor de código (verde):** Es el área central donde se escribe y edita el código que define el comportamiento de la aplicación. Aquí se desarrollan los *callbacks* asociados a los componentes, así como las propiedades, métodos y funciones.
- **Navegador de código (azu):** Organiza el contenido del archivo en secciones como funciones, *callbacks* y propiedades, facilitando la navegación dentro del código.
- ***Layout* (rosa):** Área de visualización de la interfaz gráfica de la aplicación, donde se organiza la disposición y apariencia de los componentes insertados.

5. DESARROLLO DE LA SOLUCIÓN ADOPTADA

El desarrollo de la solución propuesta en este proyecto se llevó a cabo de forma progresiva y estructurada, abordando paso a paso las distintas fases necesarias para diseñar y simular una estación robótica colaborativa con visión artificial.

La primera fase consistió en recopilar información técnica sobre el robot ABB CRB 15000 GoFa, su controlador *OmniCore C30* y el dispositivo de programación FlexPendant. Además, se evaluó la viabilidad de realizar una simulación de robótica con visión artificial emulada, investigando los procedimientos necesarios para emular una cámara Cognex, conectarla a *RobotStudio* y programar tareas de visión artificial mediante el módulo *Integrated Vision* y el lenguaje RAPID.

Dado que se dispone de un robot ABB GoFa físico con su soporte correspondiente y una cámara Cognex serie 7000, el primer paso práctico fue configurar el entorno de simulación en *RobotStudio*, garantizando que la estación virtual replicara fielmente las características y funcionalidades del robot físico. Además, se diseñó la estación de trabajo, incluyendo el robot y su soporte, siguiendo el diseño real disponible en el laboratorio. A continuación, se instaló el software *In-Sight Explorer* y se configuró la emulación de la cámara Cognex. Esta cámara virtual, generada en el ordenador, se conectó a *RobotStudio* a través de una dirección IP específica, permitiendo su integración como si se tratara de un dispositivo físico.

Una vez configurado el entorno base, se inició el diseño y desarrollo de las estaciones de trabajo, lo que resultó en un conjunto de seis prácticas docentes. Estas prácticas combinaron el desarrollo de tareas de visión artificial dentro del módulo *Integrated Vision* de *RobotStudio* con la programación de rutinas RAPID, permitiendo la interacción del robot con los objetos detectados mediante visión. Posteriormente, se implementó la comunicación industrial mediante OPC UA, utilizando el software *ABB IoT Gateway* como servidor. A través de esta plataforma, se establecieron conexiones con los clientes *MATLAB* y *RobotStudio*, lo que permitió intercambiar datos en tiempo real entre el robot y el entorno de análisis externo.

Finalmente, se diseñó una interfaz gráfica en *MATLAB* utilizando *App Designer*, que se conectó con una de las prácticas simuladas. Esta interfaz permitió visualizar el estado del robot, monitorear señales y variables, y enviar datos al robot de manera interactiva. De este modo, se integraron todos los componentes del sistema, logrando una solución completa que combina tres aspectos clave: robótica, visión artificial y comunicaciones industriales.

Figura 107. Cronología del desarrollo de la solución adoptada

5.1. Configuración del entorno RobotStudio

5.1.1. Modelado de la estación

Tras instalar la última versión del software *RobotStudio* 2024, se procedió a la creación de una estación virtual que reprodujera fielmente la configuración disponible en el laboratorio. Existen distintas formas de llevarlo a cabo: crear una estación vacía o crear un proyecto con un controlador virtual opcional.

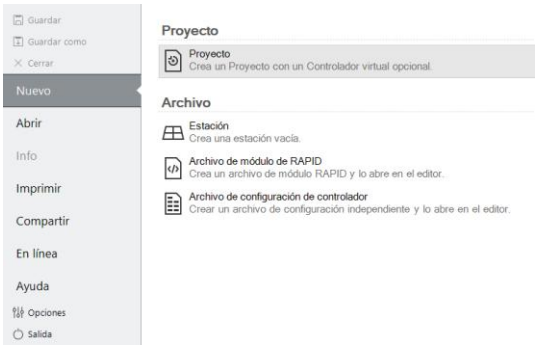


Figura 108. Creación de un nuevo proyecto

En este caso, se optó por crear un nuevo proyecto que incluyera directamente: el robot deseado, la versión adecuada de *RobotWare* y su controlador virtual correspondiente. Durante el proceso de creación se pueden elegir dos alternativas: crear un controlador virtual nuevo o generarlo a partir de un *backup* existente. En este caso, se seleccionó la creación de un controlador nuevo, definiendo el nombre del proyecto y del controlador, la ubicación de los archivos y los siguientes parámetros principales:

- **Modelo del robot:** CRB 15000 GoFa
- **Variante:** CRB 15000 5Kg 0.95m
- **Versión *RobotWare*:** 7.16.2
- **Controlador:** C30

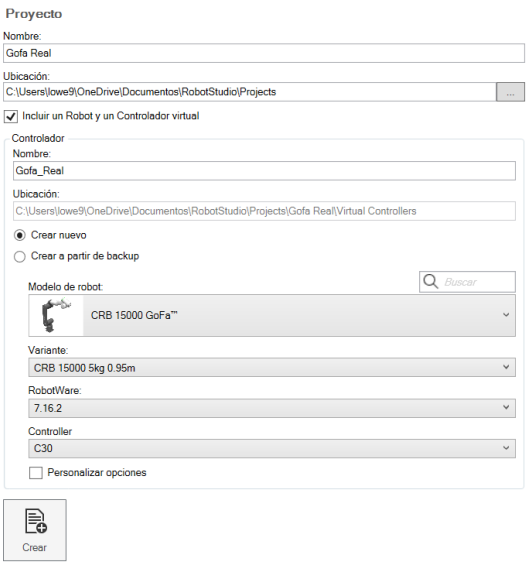
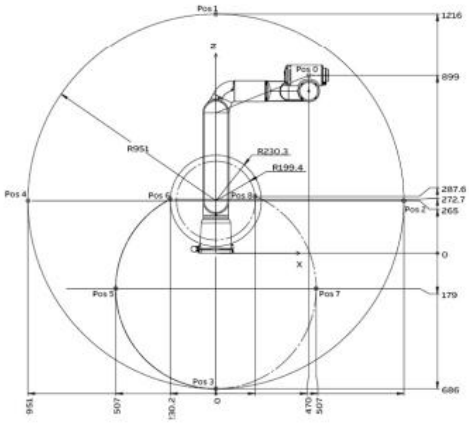


Figura 109. Configuraciones iniciales del proyecto



Movimiento		
	GoFa 5	
Movimiento del eje	Rango de trabajo	Velocidad máx. del eje
Rotación del eje 1	-180° a 180°	125 °/s
Eje 2 del brazo	-180° a 180°	125 °/s
Eje 3 del brazo	-225° a 85°	140 °/s
Eje 4 muñeca	-180° a 180°	200 °/s
Curva del eje 5	-180° a 180°	200 °/s
Eje 6 de vuelta	-270° a 270°	200 °/s

Figura 110. Rango de trabajo y velocidades máximas de ejes del robot GoFa 5 (ABB, 2025)

Una vez completado el proceso de creación, se obtuvo una estación vacía compuesta únicamente por el robot y su controlador virtual.

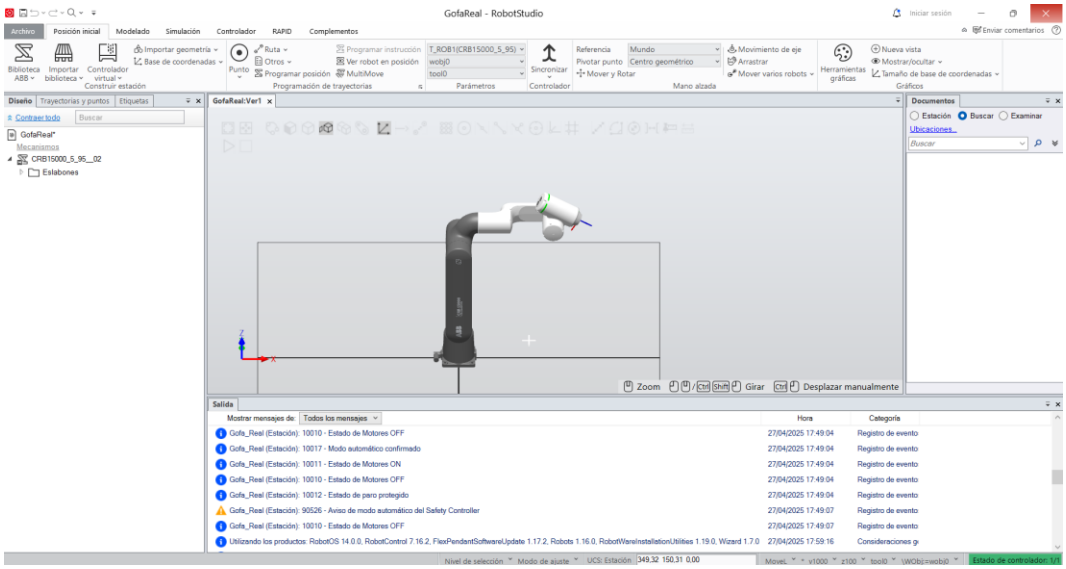


Figura 111. Estación inicial con robot colaborativo GoFa 5

Para replicar con mayor precisión la instalación real, se requirió también la incorporación del soporte físico sobre el que está montado el robot en el laboratorio. Inicialmente, se realizó una búsqueda exhaustiva de modelos CAD oficiales disponibles en la página web de ABB. No obstante, al no encontrar el modelo específico requerido, se procedió a contactar directamente con el servicio técnico de ABB mediante correo electrónico. De esta forma, se obtuvo una respuesta satisfactoria obteniéndose así el archivo CAD solicitado para completar el modelado de la estación.

Para incorporar el CAD en la estación, es necesario acceder a la pestaña de "Posición inicial", seleccionar la opción de "Importar biblioteca" y buscar el archivo CAD previamente guardado. Luego, se debe ajustar la posición del robot para asegurarse de que quede correctamente ubicado sobre la mesa. Una vez colocado en la posición deseada, se debe anclar el robot a la mesa para que ambos queden fijos como una sola unidad.

Figura 112. Estación con el robot colaborativo y su mesa de soporte

Una vez obtenida la estación simulada, idéntica a la que tenemos en el laboratorio, lo siguiente es incorporar las características internas del controlador. Para ello, accedemos a la pestaña de "Controlador", seleccionamos "Instalación" y luego "Modificar instalación". En esta sección, podremos ver el software y las características ya integradas.

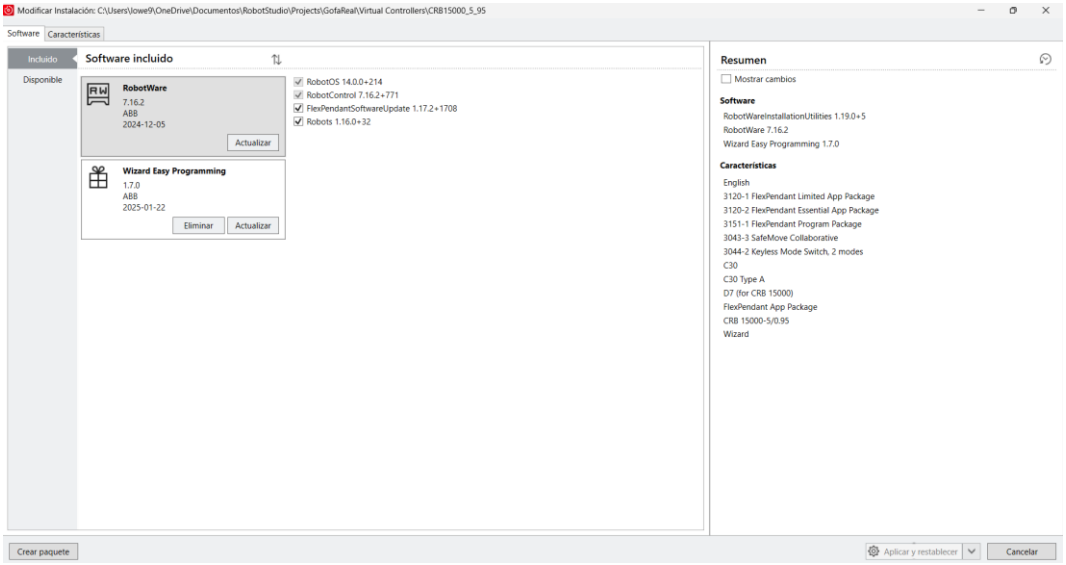


Figura 113. Software y características incluidas inicialmente en la instalación

Si alguna funcionalidad no está disponible, podemos ir a la pestaña "Características", donde encontraremos una lista de opciones que podemos buscar y seleccionar según nuestras necesidades. Estas características están agrupadas en el panel de la

izquierda en diferentes secciones. Una vez seleccionadas, hacemos clic en "Aplicar" y luego en "Restablecer" para completar la instalación.

Para este proyecto, se incorporaron las opciones de *RobotWare* recomendadas en el manual del producto proporcionado por el tutor académico, así como otras funcionalidades necesarias para permitir la visión artificial y las comunicaciones industriales mediante OPC UA.

RobotWare

El RobotWare es el software del controlador del robot. Las opciones de RobotWare incluidas son:

- 3043-3 SafeMove Collaborative
- 3114-1 Multitasking
- 3120-2 Essential App Package
- 3151-1 Program Package
- 3121-1 RW Add-In Prepared
- 3044-1 3 Modes Keyless
- 3106-1 World Zones
- 3113-1 Path Recovery
- 3107-1 Collision Detection
- 3150-1 Collision Avoidance
- 3116-1 FTP & SFTP Client
- 3124-1 EGM

Figura 114. Opciones de RobotWare recomendadas en el manual del producto

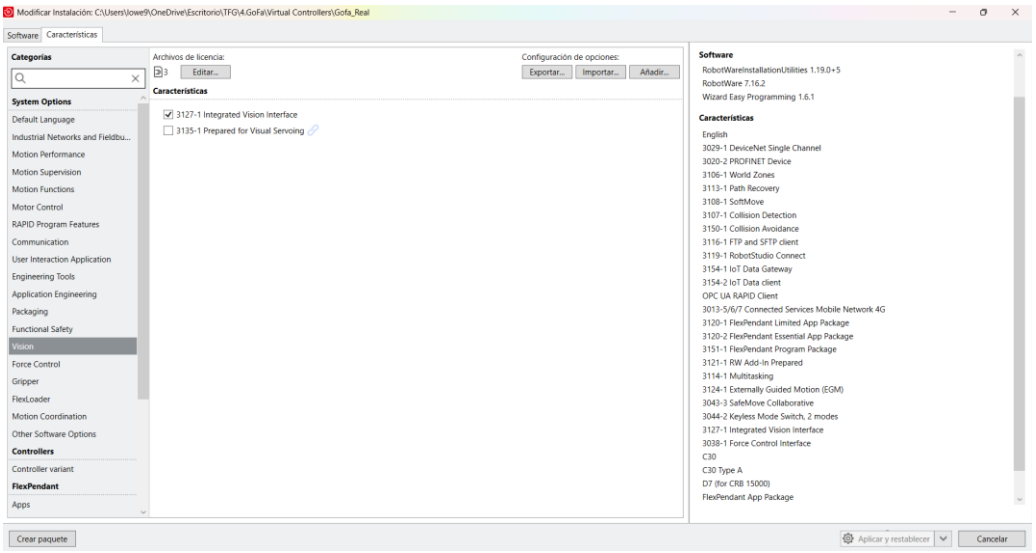


Figura 115. Características incluidas en la instalación

3127-1 *Integrated Vision Interface*: Esta funcionalidad permite integrar cámaras Cognex In-Sight directamente en el entorno de *RobotStudio*, lo que simplifica el desarrollo de tareas de visión artificial. Gracias a esta opción, es posible configurar, programar y ejecutar trabajos de visión sin necesidad de utilizar software externo.

3154-1 *IoT Data Gateway*: Esta funcionalidad convierte al controlador del robot en un servidor OPC UA, permitiendo publicar variables internas y otorgar acceso a sistemas externos que actúan como clientes. Es esencial para establecer comunicaciones industriales y se utiliza junto con *ABB IoT Gateway*, facilitando el intercambio de datos en tiempo real.

5.2. Emulación y conexión cámara Cognex

5.2.1. Instalación y configuración de In-Sight Explorer

Descarga del software

Para obtener el software *In-Sight Explorer*, se debe acceder a la página oficial de Cognex:

🔗 Descarga de In-Sight Explorer

Obtención de la licencia de emulación

Este software permite trabajar sin una cámara física mediante una licencia de programación sin conexión. Para obtenerla:

1. Acceder al enlace: 🔗 Licencia de emulación.
2. Iniciar sesión o crear una cuenta en Cognex.
3. Copiar la clave de ***Offline Programming Reference*** desde el software.
4. Introducir la clave en la web para obtener la ***Offline Programming Key***.
5. Ingresar la clave en *In-Sight Explorer* y activar la licencia.

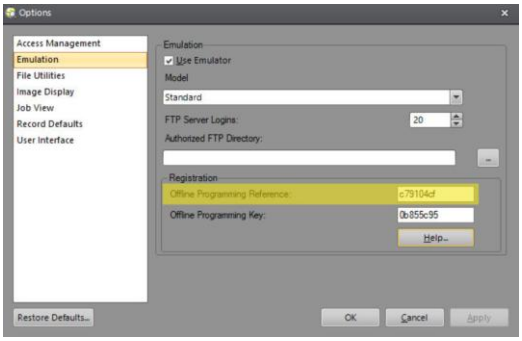


Figura 116. Obtención de la licencia de emulación

5.2.2. Interfaz de In-Sight Explorer

Una vez activada la licencia, se accede a la interfaz principal del software, donde se pueden configurar cámaras, procesar imágenes y generar programas de visión artificial.

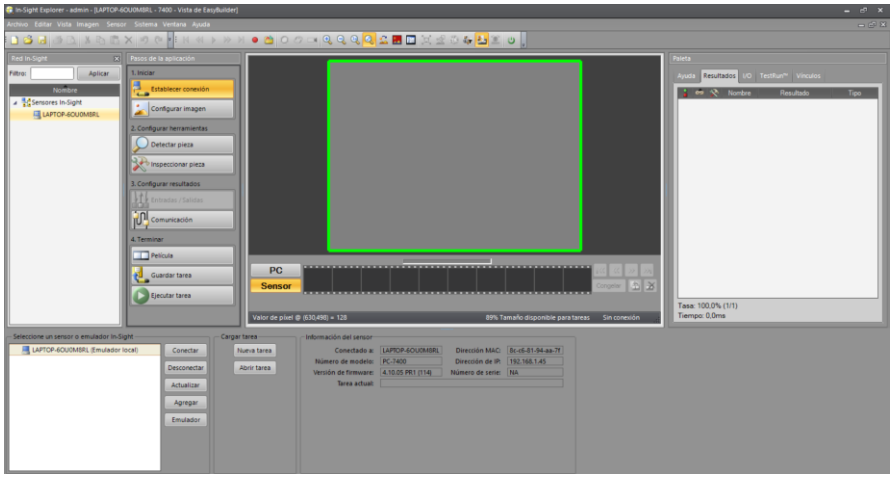


Figura 117. Interfaz In-Sight Explorer

5.2.3. Simulación de una cámara en In-Sight Explorer

Para emular una cámara en *In-Sight Explorer*:

1. En la parte inferior izquierda del software, seleccionar el ordenador.
2. Hacer clic en "Emulador".
3. Elegir el modelo de cámara deseado (por ejemplo, serie 7400 Color de In-Sight).

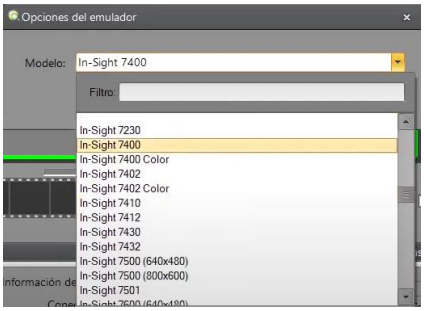


Figura 118. Selección modelo cámara emulada

4. La dirección IP de la cámara simulada corresponderá a la del ordenador.

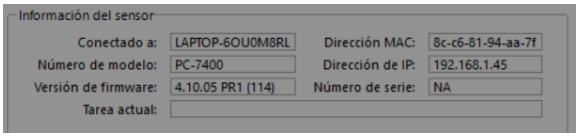


Figura 119. Información de la cámara emulada

5.2.4. Conexión del emulador de In-Sight con RobotStudio

Configuración en RobotStudio

Para establecer la conexión entre la cámara Cognex emulada en *In-Sight Explorer* y el entorno de *RobotStudio*, es necesario activar y configurar el módulo *Integrated Vision Interface*, disponible en las opciones del controlador virtual.

Pasos de configuración en RobotStudio:

1. Acceder a la pestaña “Controlador”.
2. En el panel lateral, hacer clic derecho sobre Sistemas de visión y seleccionar la opción “Visión integrada”. Esto habilita la pestaña adicional de “Visión”.
3. De nuevo, hacer clic derecho sobre “Sistemas de visión” y seleccionar “Configuración del CV”. En esta ventana, se debe seleccionar la dirección IP del emulador de cámara configurado previamente en *In-Sight Explorer.*
4. Para aplicar el cambio, *RobotStudio* solicitará reiniciar el sistema para que la configuración tenga efecto.
5. Una vez reiniciado, aparecerá un nuevo sensor añadido a la lista como "No configurado (dirección MAC:)". Se debe seleccionar “Conectar” y, si se desea, modificar su nombre para una identificación más clara dentro del entorno.

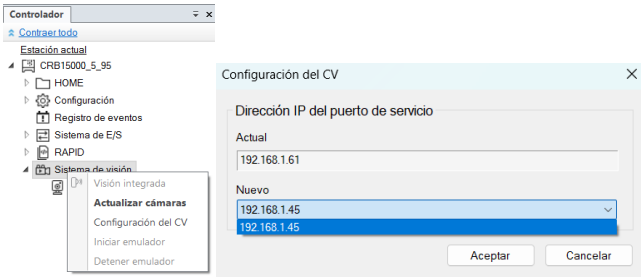


Figura 120. Configuración cámara emulada en RobotStudio

Nota: Dado que la cámara emulada comparte la misma dirección IP que el ordenador, esta dirección puede cambiar al conectarse a una red diferente. Por lo tanto, cada vez que se modifique la red, será necesario acceder nuevamente a la Configuración del CV y actualizar la dirección IP del sensor.

5.2.5. Interfaz de visión en RobotStudio

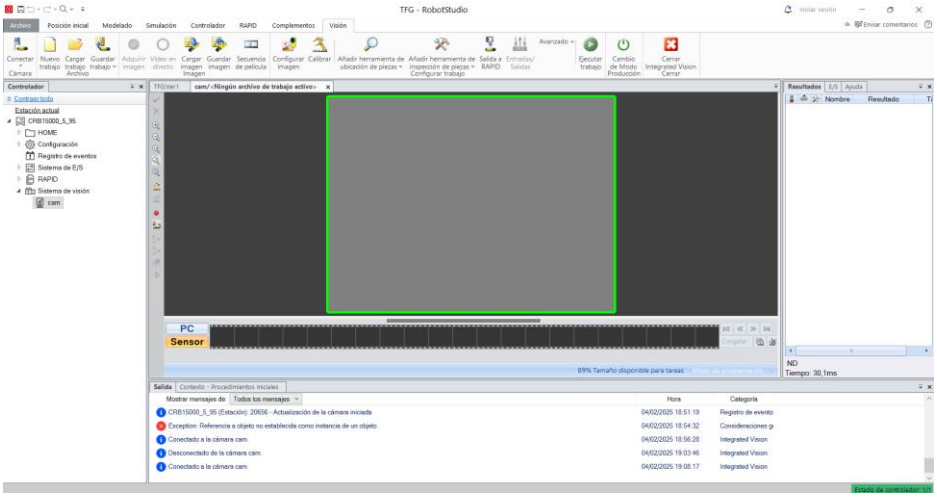


Figura 121. Interfaz visión artificial en RobotStudio 2024

Una vez conectada la cámara emulada, se habilita la pestaña “Integrated Vision”, la cual proporciona una interfaz específica para la configuración y gestión de las tareas de visión artificial dentro del propio entorno de simulación.

Figura 122. Barra de herramientas de la Interfaz de Visión Integrada en RobotStudio.

Esta pestaña organiza las distintas funciones disponibles en varias secciones:

Cámara	• Conectar/Desconectar la cámara. • Configurar la red y permisos de usuario. • Añadir sensores físicos.
Archivo	• Crear o cargar trabajos de visión. • Guardar configuraciones.
Imagen	• Adquirir imagen.

	• Activar video en directo. • Cargar/guardar imágenes y grabar secuencias.
Configuración del trabajo	• Configurar imagen y calibración. • Añadir herramientas de ubicación e inspección de piezas. • Configurar entradas/salidas y generación de datos para RAPID.
Producción	• Cambio de modo de operación

Tabla 10. Funcionalidades interfaz Visión Integrada

5.2.6. Herramientas de procesamiento de imágenes

Herramientas de ubicación de piezas

Estas herramientas permiten localizar objetos en la imagen y determinar su posición y orientación en el espacio. Algunas de ellas son:

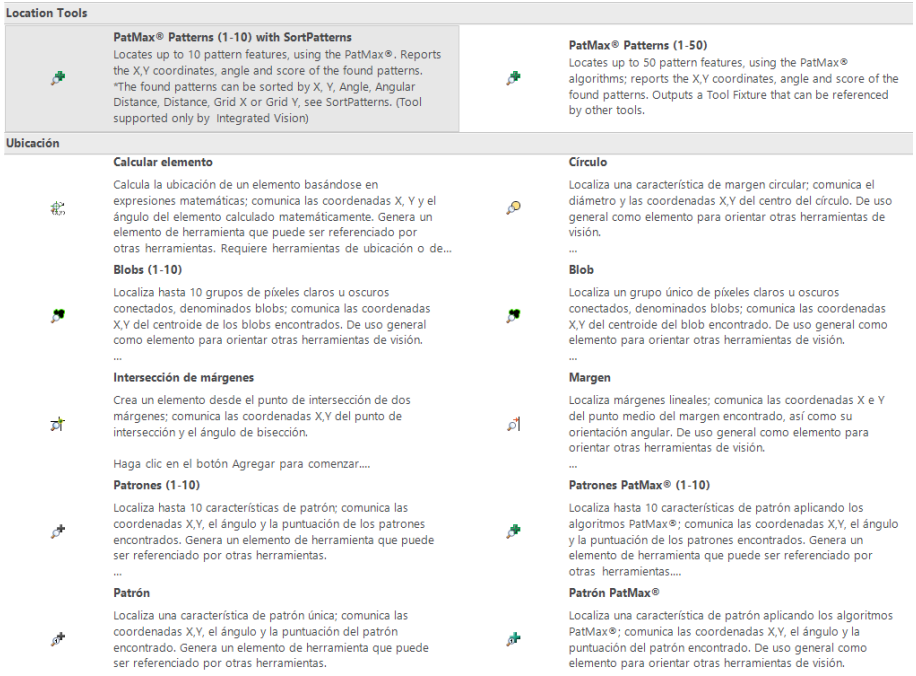


Figura 123. Herramientas de ubicación

Una de las herramientas más avanzadas y robustas para la localización de patrones es **PatMax®.** Esta utiliza algoritmos basados en geometría para encontrar objetos incluso en condiciones difíciles, como cambios de escala, rotación, iluminación variable o presencia de ruido.

Otra herramienta frecuentemente utilizada en sistemas de visión artificial es la de **detección de Blobs (*Binary Large Objects*)**, que permite identificar y contar regiones conectadas de píxeles claros u oscuros dentro de una imagen. Estas regiones pueden representar piezas, defectos o cualquier otro elemento con contraste suficiente respecto al fondo. La herramienta devuelve propiedades como el centroide, el área o la orientación de cada blob detectado.

Herramientas de inspección de piezas

Las herramientas de inspección de piezas son fundamentales en sistemas de visión artificial, ya que permiten verificar con alta precisión la calidad y las características de los objetos identificados. Estas herramientas no solo ayudan a asegurarse de que las piezas cumplen con los estándares de calidad, sino que también son esenciales para mejorar la eficiencia en las líneas de producción.

Algunas de las herramientas de inspección más comunes y sus funciones incluyen:

1. **Geometría**: Estas herramientas permiten evaluar las formas y dimensiones de los objetos. Se utilizan algoritmos que verifican si las piezas tienen la geometría esperada.
2. **Calibración**: Las herramientas de calibración permiten ajustar la relación entre el mundo real y las imágenes capturadas, corrigiendo distorsiones o errores de perspectiva, lo que asegura mediciones y resultados precisos.
3. **Detección de objetos**: Estas herramientas identifican y localizan objetos dentro de una imagen, incluso en condiciones de cambios de iluminación o rotación.
4. **Filtrado de imágenes**: El filtrado de imágenes se utiliza para mejorar la calidad de las imágenes capturadas, eliminando ruidos o distorsiones que puedan dificultar la correcta inspección de las piezas. Esto incluye técnicas como el suavizado o el aumento del contraste.
5. **Matemáticas y lógica**: Las herramientas matemáticas y lógicas permiten realizar cálculos o razonamientos complejos sobre las imágenes. Se pueden aplicar para analizar relaciones geométricas, comparar medidas entre objetos o realizar operaciones matemáticas sobre la imagen para determinar si cumple con ciertos criterios.
6. **Trazado**: El trazado de líneas y contornos es útil para inspeccionar la forma de los objetos, comprobar la alineación o la orientación de las piezas. También se usa para crear referencias visuales para otros procesos.
7. **Identificación**: Las herramientas de identificación incluyen técnicas como el reconocimiento de códigos de barras, lectura de caracteres (OCR), o la identificación de marcas y etiquetas en las piezas.
8. **Medición**: Las herramientas de medición permiten medir dimensiones precisas de las piezas, como longitud, ancho, altura, o incluso ángulos.
9. **Presencia/ausencia**: Estas herramientas verifican si un componente está presente o ausente en una pieza o ensamblaje.
10. **Recuento**: Las herramientas de recuento son utilizadas para contar el número de piezas u objetos presentes en una imagen o en una secuencia de imágenes.

Figura 124. Herramienta de inspección OCRMax

5.2.7. Lógica de funcionamiento de un sistema de visión artificial

Un sistema de visión artificial sigue una secuencia de pasos genéricos que permite realizar tareas de ubicación, inspección y clasificación.

1. **Calibración de la cámara:** Para una correcta correlación entre las coordenadas de la imagen y el espacio físico simulado, se ha realizado una calibración de tipo X/Y 1:1. Esta opción garantiza que cada píxel de la imagen corresponda a un milímetro, facilitando así el posicionamiento y manipulación precisa de las piezas.
2. **Creación de un nuevo trabajo (.job)**
3. **Carga de imágenes de referencia:** En modo emulación, es posible trabajar con capturas de pantalla generadas previamente en RobotStudio, o incluso importar imágenes reales. Desde la pestaña Visión, en la sección Imagen, se pueden cargar imágenes individuales, secuencias o activar el vídeo en directo si se dispone de una cámara física.
4. **Configuración de herramientas:** Desde el entorno de visión, se añaden las herramientas necesarias en función de los objetivos de la práctica docente. Existen herramientas de ubicación y de inspección como se han explicado anteriormente, que requieren ciertos parámetros de configuración para su correcto funcionamiento.
5. **Interpretación de resultados:** Los resultados generados por cada herramienta se visualizan en el panel derecho de resultados. Se muestra un círculo verde si la condición se ha cumplido, o un cuadrado rojo si ha fallado (*Fail*). En tareas de localización también se muestran las coordenadas y la orientación del objeto detectado.

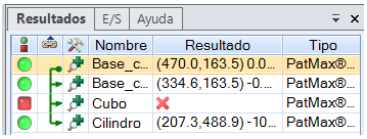


Figura 125. Panel de resultados obtenidos en visión artificial

6. **Envío de resultados a RAPID:** Para que el robot pueda actuar en función de los datos obtenidos por la visión, se configuran salidas a RAPID desde la opción "Salida a RAPID". Estos datos son enviados automáticamente al controlador, quedando disponibles para su uso en el código RAPID.

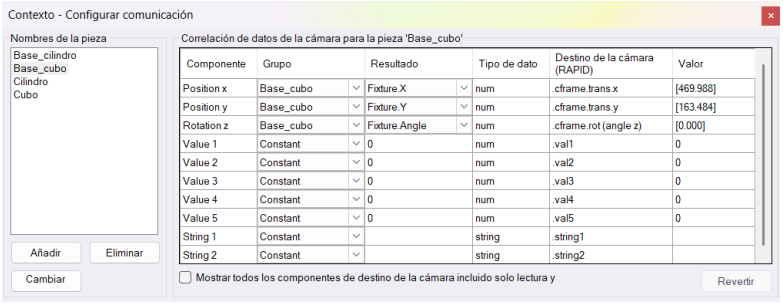


Figura 126. Salidas resultados de visión a RAPID

En esta interfaz es posible definir múltiples piezas y asociarles distintas variables: coordenadas X/Y, ángulo de rotación, valores numéricos adicionales (val1 a val5), o cadenas de texto (string1, string2).

7. **Guardado del trabajo de visión:** Una vez configurada la tarea, se guarda el archivo (.job), seleccionando la opción "Guardar como" y "Sensores In-Sight". Dentro de esta sección se elige el "PC"y se le asigna un nombre representativo al trabajo.

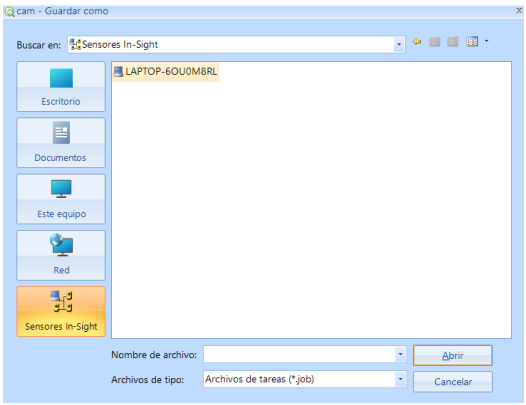


Figura 127. Guardado de trabajos de visión

8. **Implementación del código RAPID:** El entorno RAPID incluye una serie de fragmentos predefinidos para facilitar la programación de tareas de visión. En el desplegable "Fragmentos", encontramos la opción "Integrated Vision" que incluye bloques como: creación de *backup* de la cámara, sincronización de la captura de imagen o lectura de resultados de visión.

   En todas las prácticas desarrolladas se ha utilizado el fragmento denominado ***MoveToDetectedObject*()**, que permite cargar un trabajo de visión previamente guardado, capturar una imagen, obtener los resultados procesados y mover el robot a la posición detectada por el sistema de visión. Este fragmento se divide en los siguientes bloques:

   - **Declaraciones iniciales:** Se define el nombre del trabajo de visión (*myjob.job*) y se declara la variable *mycameratarget*, que almacenará la información devuelta por el sistema de visión.
   - **Variables comentadas:** Las líneas comentadas son ejemplos que el usuario debe adaptar a su caso real. Allí el usuario define parámetros clave como el objeto de trabajo (*mywobj*), la herramienta utilizada (*mytool*), la posición objetivo del objeto detectado (*myrobtarget*) y la posición del robot desde la cual se captura la imagen en caso de disponer de cámara física (*myimagepos*).
   - **Carga y configuración de la cámara**: La cámara pasa a modo programación mediante *CamSetProgramMode*, se carga el trabajo de visión correspondiente con *CamLoadJob*, y finalmente se activa el modo de ejecución usando *CamSetRunMode*. Esta secuencia prepara la cámara para procesar imágenes.

- **(Opcional) Posicionar el robot para captura:** Si la cámara se encuentra montada sobre el robot, antes de capturar la imagen, hay que colocarlo en una posición exacta para obtener siempre la misma vista. En este caso, al tratarse de una cámara emulada, esta parte no es necesaria.
- **Captura de imagen y obtención de resultados:** La instrucción *CamReqImage* solicita la adquisición de una imagen. Sin embargo, como se ha trabajado con emulación, este paso debe ejecutarse manualmente desde la interfaz gráfica, cambiando la cámara a modo programación, cargando la imagen correspondiente, y volviendo al modo de ejecución. Posteriormente, *CamGetResult* recoge los resultados del análisis de visión y los almacena en la variable *mycameratarget*.
- **Asignación del objeto de trabajo:** Se asigna el marco de coordenadas del objeto detectado a la variable *mywobj* a través del resultado de visión *mycameratarget.cframe*. Esto permite que el movimiento del robot sea relativo al objeto localizado.
- **Movimiento del robot al objeto detectado**

```
!Change the job name
CONST string myjob := "myjob.job";
VAR cameratarget mycameratarget;

!Declare robtarget, workobject, tooldata and, in case the camera is
!mounted on a moving part of the robot, the imaging position.
!CONST robtarget myrobtarget:=[[100,200,300],[1,0,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
!CONST robtarget myimagepos:=[[100,200,300],[1,0,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
!TASK PERS wobjdata mywobj:=[FALSE,TRUE,"",[[0,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
!PERS tooldata mytool:=[TRUE,[[0,0,1],[1,0,0,0]],[1,[0,0,0],[1,0,0,0],0,0,0]];

PROC MoveToDetectedObject()
    !Change the camera name
    CamSetProgramMode mycamera;
    CamLoadJob mycamera, myjob;
    CamSetRunMode mycamera;

    !If the camera is mounted on the robot, store this position during setup
    !so that the robot may always return to this position before requesting an image.
    !MoveL myimagepos, v100, fine, tool0;

    CamReqImage mycamera;
    CamGetResult mycamera, mycameratarget;
    mywobj.oframe := mycameratarget.cframe;

    !During the first cycle, run the program until this point,
    !then jog the tool to the desired grip position and modpos myrobtarget.
    MoveL myrobtarget, v100, fine, mytool \WObj:=mywobj;
ENDPROC
```

Figura 128. Fragmento RAPID predefinido "MoveToDetectedObject()"

5.3. Diseño de estaciones de trabajo

Una vez configurado el entorno de simulación y establecida la conexión con la cámara Cognex emulada, se procedió al desarrollo de diversas estaciones de trabajo con fines didácticos. Estas estaciones tienen como objetivo simular tareas colaborativas mediante visión artificial, integrando la captura de imágenes, el procesamiento mediante herramientas de localización o inspección y la posterior transferencia de datos al lenguaje RAPID para ejecutar los movimientos necesarios.

Se han llevado a cabo un total de seis prácticas docentes, cada una centrada en un aspecto fundamental de la visión artificial aplicada a la robótica colaborativa. Aunque las tareas están inspiradas en aplicaciones reales del entorno industrial, su nivel de complejidad se ha adaptado al ámbito educativo, permitiendo que el alumnado se

familiarice progresivamente con las principales herramientas de programación y visión disponibles en *RobotStudio*.

Dado que todas las prácticas comparten una misma estación base y una estructura similar, se ha optado por presentar en primer lugar un apartado con la configuración común de dicha estación. A continuación, se describe de forma individual cada una de las prácticas, detallando su objetivo, desarrollo, componentes implicados y resultados obtenidos.

5.3.1. Configuración común de la estación

Todas las prácticas docentes desarrolladas comparten ciertos elementos comunes. Con el fin de mantener una estructura coherente y evitar repeticiones innecesarias en cada práctica, a continuación, se describen previamente estos componentes y configuraciones.

Organización por grupos de componentes

Cada práctica ha sido organizada dentro de su propio grupo de componentes en el explorador del proyecto de *RobotStudio*. Esta estructura permite agrupar de forma ordenada y modular todos los elementos que intervienen en la práctica, tales como piezas, planos, sensores, herramientas y componentes inteligentes, a excepción de los elementos comunes que permanecen fuera de estos grupos.

Este sistema de organización presenta una ventaja desde el punto de vista docente, permite al profesorado subir al campus virtual únicamente el grupo de componentes correspondiente a cada práctica, sin necesidad de compartir toda la estación completa. Así, el alumnado solo debe importar el grupo correspondiente y trabajar directamente con él.

Para crear un grupo de componentes, se accede a la pestaña “Modelado”, sección “Crear”, y se selecciona “Grupo de componentes”. Tras su creación, es posible asignarle el nombre de la práctica correspondiente.

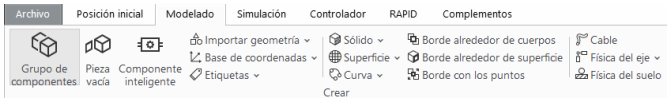


Figura 129. Creación grupos de componentes

Posteriormente, se crean o importan los distintos elementos necesarios (piezas, objetos, herramientas, sensores, etc.) y se arrastran dentro del grupo creado. Una vez completado, se puede guardar como biblioteca al hacer clic derecho sobre el grupo y seleccionar “Guardar como biblioteca...”

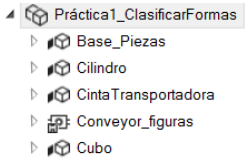


Figura 130. Grupos de componentes

Para cargar la práctica correspondiente, el alumno debe acceder a la pestaña "Posición inicial" y seleccionar la opción "Importar biblioteca", lo que le permitirá localizar y cargar el archivo previamente guardado que contiene el grupo de componentes específico de cada una de las prácticas. Este proceso asegura que cada estudiante trabaje únicamente con los elementos necesarios para la práctica en cuestión y permite a los docentes mantener un control sobre los recursos compartidos y actualizar cada práctica de forma independiente.

Nota: Los componentes inteligentes agrupados dentro de un conjunto de componentes no son accesibles desde la lógica de la estación. Para garantizar la conexión de entradas o salidas del controlador con estos elementos, es necesario establecer previamente dichas interconexiones antes de integrarlos al grupo.

Configuración del sistema de E/S escalables

Para establecer la comunicación entre los distintos componentes inteligentes y el controlador del robot, se ha incorporado el módulo de entradas y salidas escalables (*Scalable* I/O). Esto permite gestionar las señales digitales tanto desde el código RAPID como a través del panel de entradas y salidas de *RobotStudio*, facilitando así el control lógico de sensores, herramientas o dispositivos.

Para su activación, se debe acceder a la pestaña "Controlador", sección "E/S", y seleccionar la opción "Herramienta de ingeniería de E/S". Esta acción habilita una nueva pestaña desde la cual se puede añadir y configurar el dispositivo.

Figura 131. Pestaña Herramienta de ingeniería de E/S en RobotStudio

El proceso comienza haciendo clic en "Add ABB Device" y seleccionando E/S escalable, que corresponde al protocolo EtherNet/IP. En la ventana emergente, se define la configuración del dispositivo. El primer paso consiste en elegir un dispositivo base, entre los disponibles:

- **DSQC1030** – Dispositivo base digital estándar.
- **DSQC1042** – Dispositivo base digital seguro.

Además, es posible incorporar hasta cuatro módulos adicionales.

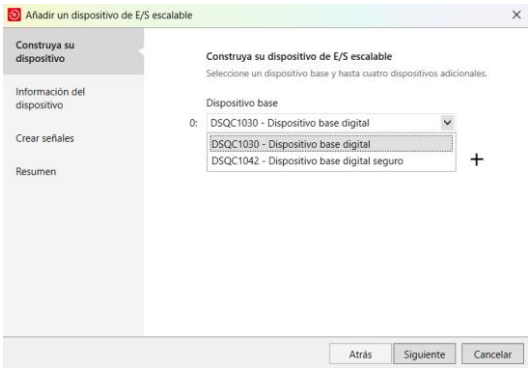


Figura 132. Elección dispositivo base de E/S escalable

En el siguiente paso, se especifica el nombre identificativo del dispositivo, su dirección IP (en caso de hardware real) y se activa la opción “Simular dispositivo”, si se desea utilizar únicamente en entorno virtual. En este proyecto se ha marcado dicha opción, manteniendo el nombre por defecto “Scalable_IO”.

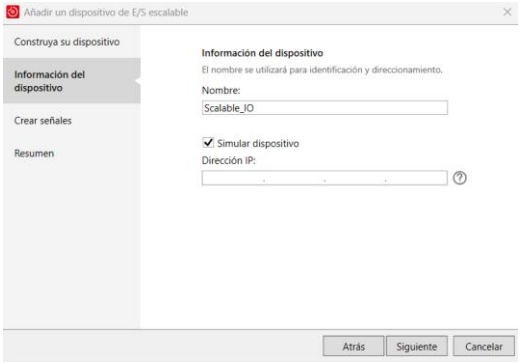


Figura 133. Configuración del dispositivo E/S escalable

Finalmente, se procede a la creación de las señales del dispositivo escalable. Para ello, *RobotStudio* ofrece la posibilidad de generar automáticamente las señales de entrada y salida, lo cual agiliza el proceso. Para ello, se marca la casilla “Crear señales automáticamente”, se define un prefijo de nombre para las señales, y se finaliza la configuración con “Siguiente”.

Una vez configuradas las entradas y salidas necesarias para el funcionamiento de los distintos componentes inteligentes, es imprescindible guardarlas en el controlador para que puedan ser utilizadas correctamente durante la simulación. Para ello, se accede a la herramienta “Herramientas de ingeniería de E/S” y, dentro de esta, a la sección “Controlador”. Allí se selecciona la opción “Lectura y escritura”, eligiendo el controlador asociado al robot utilizado.

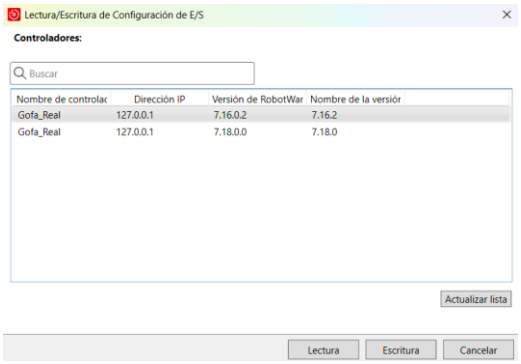


Figura 134. Lectura/Escritura del controlador virtual

A continuación, se marca la opción “Escritura”, se guarda el proyecto y se reinicia el controlador. Una vez realizado este proceso, las entradas y salidas quedan registradas y pueden visualizarse y gestionarse desde el panel lateral de la interfaz de *RobotStudio*.

Figura 135. Componente inteligente controlador virtual

Creación de la herramienta: ventosa

Para implementar la herramienta de agarre del robot, se ha optado por una ventosa neumática. Se ha integrado el modelo 3D en el entorno de *RobotStudio* y asociado al extremo del robot. Posteriormente, con objetivo de simular el comportamiento real de una ventosa, se ha desarrollado un componente inteligente específico destinado a controlar el agarre de objetos.

1. **Creación del componente inteligente**
   Desde la pestaña "Modelado", sección "Crear", se selecciona la opción "Componente inteligente". Esto genera un nuevo componente vacío, sin entradas ni salidas ni componentes subordinados.

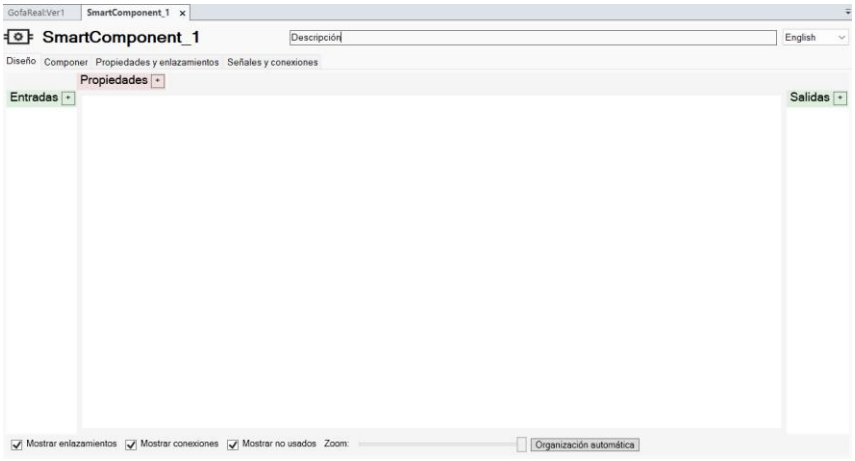


Figura 136. Creación de un nuevo componente inteligente vacío

2. **Señal de activación del sistema de succión**
   La activación del sistema de vacío se produce mediante una señal de entrada digital llamada "ElecVenturiVacio". Esta se crea desde el margen izquierdo del editor del componente inteligente, haciendo clic en el símbolo "+", especificando el nombre de la señal, su tipo y restablecimiento automático.
3. **Sensor de detección de pieza**
   Se ha empleado un bloque "LineSensor" que simula el sensor de presencia en la ventosa. Este campo detecta si hay una pieza dentro del campo de succión y en ese caso, activa una salida lógica. Este sensor mide 15mm y se ha situado en el centro de la herramienta.

4. **Lógica de control**
   Para gestionar la succión y liberación de las piezas, se han implementado dos puertas lógicas:
   - **NOT:** Esta puerta lógica se utiliza para invertir la señal de detección, lo que permite activar la liberación de la pieza cuando se detecta la señal invertida.
   - **LogicSRLatch:** Este bloque funciona como un registro de memoria, manteniendo la succión activa hasta que se cambie alguna condición. Cuando se detecta y agarra una pieza, la señal "Set" se activa, manteniendo el sistema de succión. Una vez que la pieza se libera, se activa la señal "Reset", lo que desactiva la succión.
5. **Unión de la pieza al robot**
   El bloque "Attacher" se encarga de unir virtualmente la pieza al TCP del robot una vez se cumplen las condiciones de succión y detección. Esto simula que la pieza ha sido agarrada correctamente y la mantiene unida hasta que se ordene su liberación.
6. **Liberación de la pieza**
   Cuando se desactiva la succión o el sensor ya no detecta la pieza, se activa el bloque "Detacher". Este bloque rompe la unión entre la herramienta y el objeto, simulando su liberación.
7. **Señal de confirmación de agarre**
   Como señal de verificación, el componente inteligente genera una salida digital llamada "ContactoVacuostato", que indica si la pieza ha sido correctamente agarrada. Esta señal puede utilizarse en el programa principal como condición de seguridad o control. Se configura del mismo modo que la entrada, desde el panel izquierdo del editor

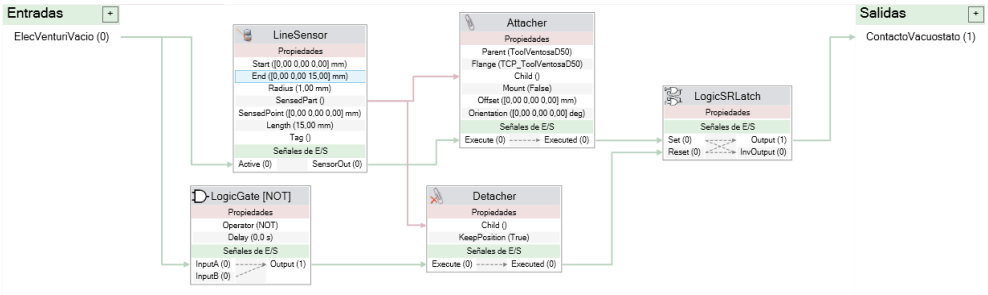


Figura 137. Lógica del componente inteligente ventosa (SC_VentosaD50)

Una vez finalizado el componente inteligente de la herramienta, es necesario colocar la ventosa en el extremo del brazo del robot y definir su TCP. Para ello, se accede al explorador del proyecto, dentro de la pestaña "Trayectorias y puntos", y se hace clic derecho sobre la sección "Datos de herramienta". Desde allí, se crea un nuevo TCP especificando su nombre, posición y orientación relativa a la herramienta.

Para que el robot pueda utilizar este TCP correctamente durante la ejecución del programa, es imprescindible sincronizar los datos con RAPID, asegurando su integración en el módulo de programación correspondiente a la práctica.

Por último, es necesario vincular el componente inteligente de la herramienta a las entradas y salidas digitales del controlador. Esta conexión permite controlar la activación de la succión directamente desde el código RAPID mediante señales digitales.

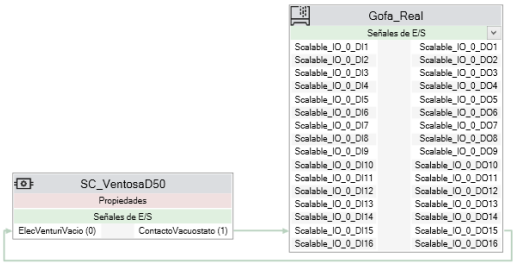


Figura 138. Conexión del componente inteligente ventosa al controlador virtual

Componente inteligente: captura de imagen

Para simular el instante en el que la cámara emulada realiza una captura de imagen, se ha creado un componente inteligente personalizado, llamado "**ImageCapture**". Este componente permite cambiar temporalmente el punto de vista de la estación al área de trabajo visible por la cámara, tomar una captura de pantalla del entorno y volver a la vista general. De esta forma, se emula el proceso de adquisición de imagen.

Este es el proceso de creación de dicho componente inteligente:

1. **Creación del componente inteligente**
2. **Inserción de componentes internos**
   - **"MoveToViewPoint" (2)**: uno de ellos cambia el punto de vista enfocando el área de trabajo visible por la cámara emulada y posteriormente, devolviendo la vista a su posición general tras realizar la captura de imagen.
   - **"ExecuteCommand"**: se configura para ejecutar el comando "FileScreenshot", que genera la captura de imagen de la pantalla de *RobotStudio*.

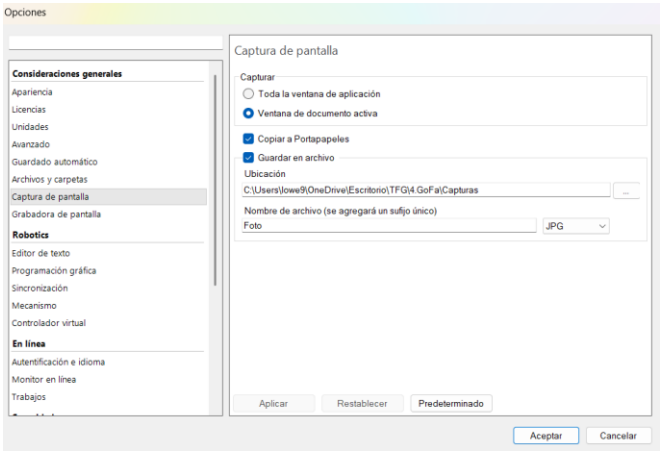


Figura 139. Configuración opciones captura de pantalla

> **Nota: Por defecto, la captura de pantalla incluye toda la interfaz de RobotStudio. Sin embargo, en este caso solo interesa capturar la ventana principal donde se visualiza el entorno de trabajo. Para ajustar esto, se debe acceder a Archivo > Opciones > Captura de pantalla, seleccionar la opción "Ventana de documento activa", definir si se desea cambiar la ubicación de guardado, y opcionalmente establecer un prefijo automático para los nombres de las capturas.**

3. Configuración de señales

Para permitir el control del componente desde el programa RAPID o desde señales externas, se han definido las siguientes señales:

- **Señal de entrada "Foto"**: activa el proceso de captura de imagen.
- **Señal de salida "Foto_guardada"**: Notifica que el proceso de captura ha finalizado correctamente.

Estas señales se crean desde los márgenes laterales del editor del componente inteligente, haciendo clic en el símbolo "+". En cada caso, se debe especificar el nombre de la señal, su tipo y si se desea que se restablezca automáticamente tras su activación.

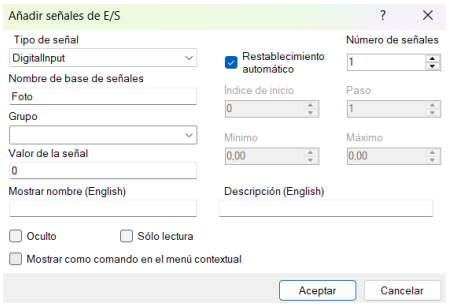


Figura 140. Creación de señales de E/S

Este componente inteligente se ha utilizado de manera común en todas las prácticas docentes desarrolladas, representando el momento en el que se realiza la captura la imagen que luego será procesada por el sistema de visión artificial.

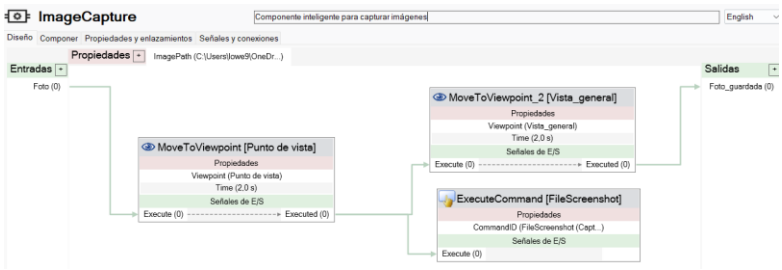


Figura 141. Lógica del componente inteligente de captura de imagen (ImageCapture)

Al igual que en el caso de la herramienta, es necesario vincular el componente inteligente de la cámara a las entradas y salidas digitales del controlador del robot. Esta integración permite gestionar la captura de imágenes directamente desde el código RAPID, activando o desactivando la función de captura mediante señales digitales.

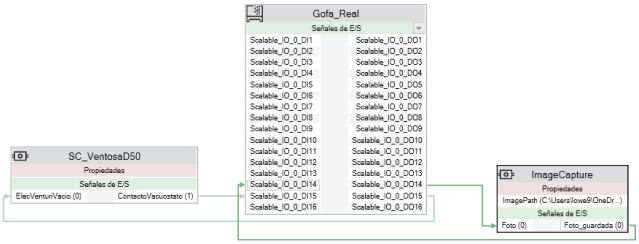


Figura 142. Conexión del componente inteligente "ImageCapture" al controlador virtual

Configuración del WorkObject y alineación con el área de visión

Para representar el encuadre de la cámara emulada, se ha creado una superficie rectangular en la pestaña "Modelado" de *RobotStudio*. Este rectángulo simula el campo de visión de la cámara y debe coincidir con sus dimensiones reales. En este caso, se ha emulado una cámara de la serie In-Sight 7400, con una resolución de 800x600 píxeles.

Con el objetivo de facilitar la calibración y asegurar que cada píxel corresponda a un milímetro en la superficie, se asignan las medidas de la superficie de trabajo como 800x600 mm. De esta manera, se consigue que la relación entre los píxeles de la cámara y las dimensiones físicas del área sea precisa y directa.

Para crear esta superficie rectangular en *RobotStudio*, se accede a la pestaña "Modelado", sección "Crear", y se selecciona Sólido: "Tetraedro". A continuación, se definen las dimensiones del área de visión (800x600 mm) y su ubicación en la estación.

Figura 143. Modelado de sólido Tetraedro

Además, es necesario crear un objeto de trabajo (*WorkObject*) que esté asociado al plano previamente definido. Para ello, se debe acceder al Explorador del proyecto, en la pestaña "Posición inicial", dentro del apartado "Trayectorias y puntos", y seleccionar la opción "Crear objeto de trabajo".

Este objeto ha sido nombrado como "cam_WObj" y se ha ubicado en la esquina superior izquierda del área de visión modelada anteriormente. También es importante definir la altura correspondiente, la cual dependerá de la posición de las piezas. Se pueden crear múltiples objetos de trabajo si se requieren diferentes alturas.

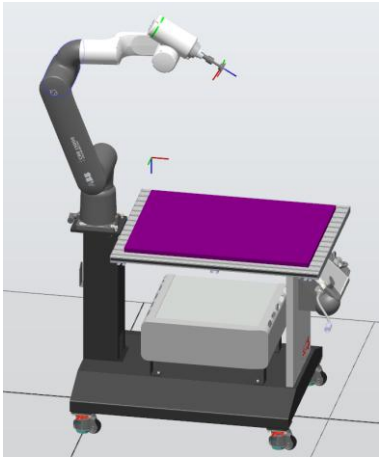


Figura 144. Configuración del WorkObject (cam_WObj) y alineación con el área de visión

Este objeto de trabajo sirve como referencia tanto para el posicionamiento de los objetos como para las herramientas de visión artificial. En la pestaña "Visión", debe seleccionarse la opción de calibración "Escala X/Y" 1:1 en milímetros, asegurando que la correspondencia entre las dimensiones de la imagen y entorno físico sea correcta.

Finalmente, para completar la configuración, es necesario crear puntos de vista personalizados que se alineen exactamente con el rectángulo del área de visión para simular la captura de imagen. Estos se crean desde la pestaña "Herramientas gráficas", habilitada al seleccionar "Herramientas gráficas" en la pestaña Posición inicial. Allí se utiliza la opción "Nueva vista" al posicionar la cámara virtual con precisión. Así se garantiza que, al realizar la captura de pantalla, se obtenga exactamente la imagen que vería una cámara física instalada en la misma posición.

Para verificar que el encuadre es correcto, se utiliza el componente inteligente "ImageCapture" creado anteriormente, que simula la toma de una fotografía desde la perspectiva de la cámara. Mediante un proceso de prueba y error, se ajusta la posición y orientación del punto de vista hasta que la captura de imagen obtenida coincida exactamente con el área rectangular que representa el campo visual de la cámara.

Dado que la captura de pantalla se realiza sobre la ventana de documento activa de *RobotStudio*, y que cualquier cambio en el tamaño de dicha ventana puede afectar al encuadre, se ha optado por configurar la interfaz en modo ventana flotante ocupando la mitad de la pantalla. Esta solución garantiza mayor precisión y uniformidad en las capturas realizadas, evitando desviaciones debidas a variaciones en el tamaño de la interfaz activa.

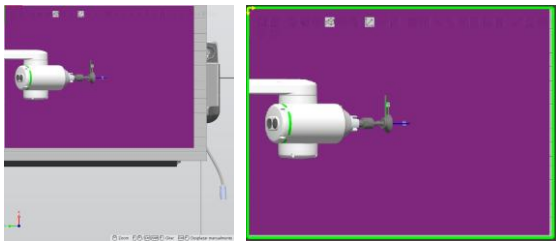


Figura 145. Captura de pantalla realizada por "ImageCapture" y su visualización en la pestaña Visión

Nota: Ya que se prefiere una perspectiva ortogonal para simplificar el tratamiento visual posterior, se debe configurar este aspecto desde la pestaña Herramientas gráficas, donde es posible activar el modo de proyección ortográfica para el punto de vista seleccionado.

Definición de la trayectoria "GoHome" del robot

Por último, como parte de la configuración común a todas las prácticas, se definió una posición de referencia "Home", que actúa como punto de inicio y retorno seguro del robot. Esta posición garantiza un punto seguro de arranque, alejado de posibles colisiones.

Para definirla, se accede a la pestaña "Posición inicial", donde se selecciona tanto el objeto de trabajo correspondiente (*wobj0*) como la herramienta activa (*TCP_Ventosa*). A continuación, se guía manualmente al robot hasta la posición deseada, y se utiliza la opción "Programar instrucción" para registrar la posición actual. Esta queda almacenada dentro del objeto de trabajo previamente seleccionado, y se puede acceder a ella desde el explorador de proyecto en la sección de "Trayectorias y puntos".

Una vez registrada la posición "Home", se procede a la creación de una trayectoria que permita al robot desplazarse automáticamente hasta dicha posición cuando sea necesario. Para ello, se hace clic derecho sobre el apartado "Trayectorias y procedimientos" y se selecciona "Crear trayectoria", la cual se ha nombrado como "GoHome". Finalmente, desde el árbol de proyecto, se localiza la posición "Home" dentro del objeto de trabajo y se arrastra hacia el interior de esta nueva trayectoria.

```
LOCAL PROC GoHome()

   MoveJ Home,v200,z100,TCP_ToolVentosa\WObj:=wobj0;

ENDPROC
```

Figura 146. Código RAPID de la función "GoHome()"

5.3.2. Práctica 1: Clasificación por formas

Esta práctica consiste en la clasificación automática de piezas según su forma, utilizando visión artificial y un sistema robótico con una cinta transportadora. Las piezas, que pueden ser cubos o cilindros, se generan aleatoriamente sobre la cinta. Esta se detiene cuando una pieza es detectada por el sensor situado al final del recorrido. En ese instante, se captura una imagen para identificar el tipo de pieza mediante una tarea de visión configurada anteriormente. Una vez clasificada, el robot la recoge y la deposita en el hueco correspondiente de una base diseñada para tal fin, tras lo cual se reanuda automáticamente el movimiento de la cinta transportadora para continuar el proceso.

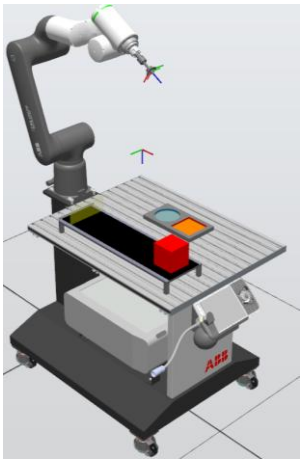


Figura 147. Estación de la práctica 1 Clasificación por formas

Modelado de componentes

Para llevar a cabo esta práctica, fue necesario modelar los distintos elementos que intervienen en ella: las figuras a clasificar (cubo y cilindro) y la base dónde se depositan una vez identificada la forma.

Modelado de las piezas

Los dos tipos de piezas que se generan sobre la cinta transportadora son cubos y cilindros, los cuales se modelan directamente desde la pestaña "Modelado" en *RobotStudio*, dentro del apartado "Crear", utilizando las opciones disponibles en el desplegable "Sólido".

- **Cubo:** Se creó seleccionando la opción "Tetraedro". Se especificaron las dimensiones de ancho, largo y altura, fijándolas todas en 100 mm para mantener la forma cúbica.
- **Cilindro:** Se generó a partir de la opción "Cilindro", configurando un radio de 50 mm y una altura de 100 mm, de forma que su volumen y proporciones fueran comparables a las del cubo.

Figura 148. Modelado de las piezas de la práctica 1 (Cubo y cilindro)

Ambos modelos se han diseñado con dimensiones adecuadas para facilitar su detección mediante visión artificial y permitir una manipulación eficaz con la herramienta de succión del robot. Inicialmente se han posicionado sobre la mesa de

trabajo, aunque esta ubicación será modificada posteriormente al integrarlos en el componente inteligente de la cinta transportadora.

Para facilitar la diferenciación visual tanto en el entorno de simulación como en las pruebas de visión artificial, se ha asignado el color rojo al cubo y azul al cilindro.

Modelado de la base de clasificación

Para permitir al robot clasificar las piezas en función de su forma, se ha diseñado una base con cavidades específicas para cada tipo de figura. Estas cavidades cuentan con una ligera holgura respecto a las dimensiones reales de las piezas, lo que facilita su inserción sin comprometer a la precisión del proceso. Para llevarlo a cabo se han seguido los siguientes pasos:

1. **Creación de figuras molde:** Se han creado dos figuras (un cubo y un cilindro) con dimensiones ligeramente superiores a las de los modelos originales. Estas figuras actúan como moldes para realizar posteriormente las cavidades. El cubo se definió con unas dimensiones de 115 × 115 mm, y el cilindro con un diámetro de 115 mm. La altura de ambas figuras es irrelevante, ya que únicamente se utilizarán para aplicar una operación de resta entre los distintos cuerpos.
2. **Modelado de la base:** A continuación, se modeló un sólido rectangular que actuará como base de clasificación, utilizando el comando “Crear”, ”Sólido” y “Tetraedro” en la pestaña de “Modelado”. Las medidas seleccionadas han sido 155x290x20mm, proporcionando el espacio necesario para disponer ambos huecos de forma equilibrada.

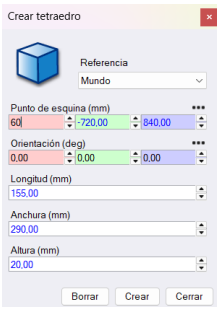


Figura 149. Modelado de la base de clasificación de la práctica 1

3. **Posicionamiento de los moldes:** Los moldes llevados a cabo se colocan sobre la superficie de la base, en la posición deseada y a la profundidad adecuada para generar las cavidades. Se cuidó que la disposición fuera simétrica y con una separación uniforme, tanto entre los huecos como con respecto a los bordes de la base.

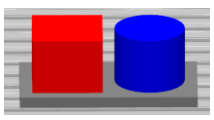


Figura 150. Posicionamiento de los moldes para la elaboración de la base de clasificación

4. **Operación de resta:** Finalmente, se utilizó la herramienta "Restar" dentro del entorno de "Modelado". Para ello, se seleccionó primero la base como sólido principal y, posteriormente, cada una de las figuras molde como cuerpos a sustraer. Además, se desmarcó la opción "Conservar original" para eliminar automáticamente los moldes tras la operación y evitar piezas innecesarias en la estación.

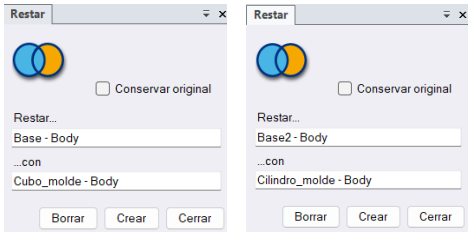


Figura 151. Elaboración de los huecos de la base de clasificación mediante operación de resta

Una vez completado el proceso, la base final se colocó en una ubicación óptima sobre la mesa de trabajo, garantizando tanto el acceso del robot como una visibilidad adecuada para el sistema de visión artificial.

Componente inteligente: Conveyor_figuras

Para simular el transporte de las piezas, se ha incorporado un componente inteligente que representa el funcionamiento de una cinta transportadora. Para ello, en primer lugar, se ha importado el modelo 3D desde una biblioteca externa y se le ha dotado de la lógica necesaria para la generación aleatoria y desplazamiento automático de las piezas y detección de presencia al final del recorrido para coordinar el sistema de visión y las acciones del robot.

Estructura del componente

1. **Entrada digital de control ("Inicio_cinta"):** Esta señal permite activar o desactivar el funcionamiento de la cinta transportadora desde el controlador.
2. **Temporizador ("Timer"):** Se ha incorporado un bloque "Timer" para controlar el intervalo de generación de piezas. En este caso, se ha configurado para un intervalo de dos segundos, repitiéndose de forma indefinida activando la lógica de generación de piezas.
3. **Generador aleatorio y comparadores:** Para garantizar la aparición aleatoria de diferentes figuras, se ha incorporado un bloque "Random", que genera un valor decimal entre 0 y 2. Este valor se evalúa mediante dos bloques "Comparer": si el valor generado es menor o igual que 1, se activa la generación de un cubo; mientras que, si es mayor que 1, un cilindro. Esta configuración permite una probabilidad equitativa del 50% para cada tipo de figura.
4. **Fuentes de piezas ("Source_Cubos" y "Source_Cilindros"):** Se han definido dos bloques de generación de piezas correspondientes a cada figura. En ellos se especifican tanto el modelo 3D que se va a instanciar como su posición y orientación iniciales sobre la cinta. Además, se ha desactivado cualquier comportamiento físico en estas piezas, ya que no es necesario para el desarrollo de la lógica del sistema.

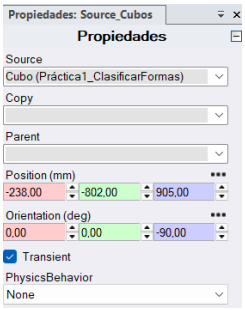


Figura 152. Configuración del componente "Source"

5. **Cola ("Queue"):** El bloque "Queue" actúa como un almacén temporal dónde se gestionan las piezas generadas. Cada copia generada de las figuras se inserta en el back de la cola mediante "Enqueue", y luego son procesadas secuencialmente.
6. **Movimiento lineal:** Se ha utilizado un bloque "LinearMover" para llevar a cabo un desplazamiento de las piezas en la dirección deseada, simulando el avance de la cinta transportadora. Se ha definido una dirección hacia delante (eje Y positivo) y una velocidad de 250mm/s.
7. **Sensor del final de línea:** Se ha incorporado un sensor plano al final del recorrido de la cinta, encargado de detectar la llegada de cada figura a la posición final. Cuando se activa, este sensor emite una señal digital de salida que indica la presencia de una pieza en el extremo de la línea.
8. **Sensor base de figuras:** Adicionalmente, se ha añadido un sensor sobre la base de clasificación que detecta si una figura ha sido posicionada correctamente en su cavidad. Para mantener el flujo del proceso y evitar acumulaciones innecesarias, el sensor está conectado a la entrada "Delete" del bloque "Queue", lo que permite eliminar automáticamente las figuras una vez detectadas.
9. **Salida digital ("Caja_PosicionFinal"):** Como se ha mencionado anteriormente, se ha añadido una señal de salida digital que notifica al usuario cuando una pieza ha llegado al final de la línea.

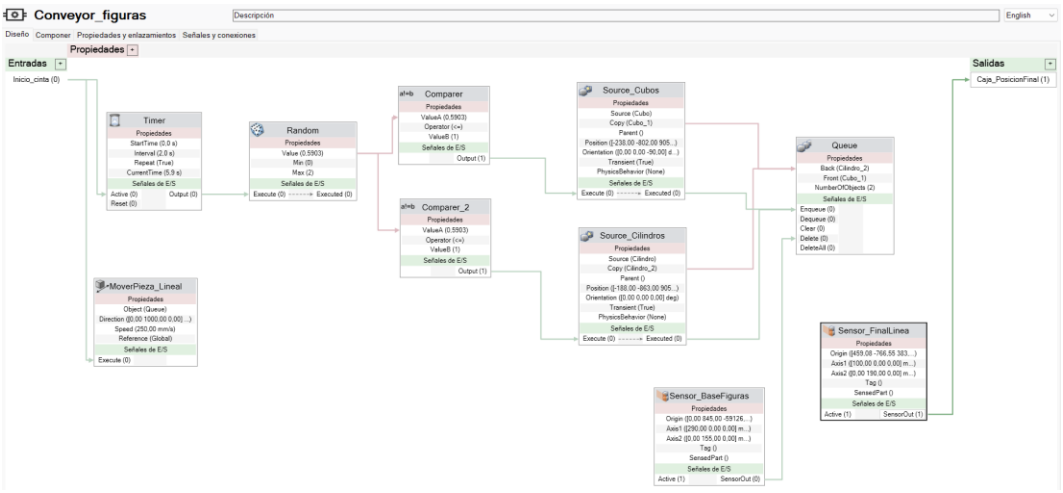


Figura 153. Lógica del componente inteligente de la cinta transportadora de la práctica 1 (Conveyor_figuras)

El comportamiento de este componente inteligente está conectado mediante señales digitales al controlador del robot. De este modo, la cinta se detiene automáticamente cuando una pieza es detectada por el sensor de fin de línea, y permanece parada hasta que el sistema de visión realiza la clasificación y el robot completa la manipulación.

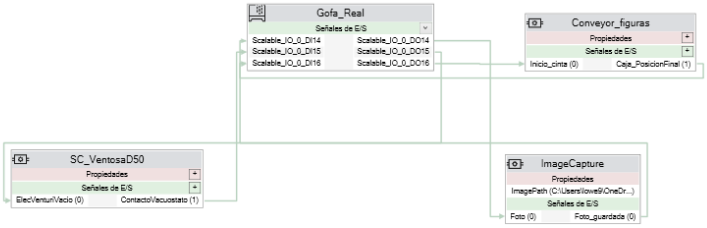


Figura 154. Conexión del componente inteligente "Conveyor_figuras" al controlador virtual

Tarea de visión

La implementación de la tarea de visión para esta práctica tiene como objetivo identificar la forma de la pieza cuando esta alcanza el final de la cinta transportadora.

En primer lugar, se creó un nuevo trabajo de visión mediante la opción "Crear trabajo". A continuación, se realizó la calibración de la imagen seleccionando la opción "Escala X/Y" 1:1 en milímetros, lo que garantiza que las herramientas de visión operen sobre una escala coherente con las dimensiones físicas del entorno simulado, asegurando el posicionamiento preciso del robot.

Para configurar la tarea, se emplearon dos capturas de imagen generadas previamente mediante el componente inteligente "ImageCapture", una correspondiente a un cubo y otra a un cilindro. Estas imágenes simulan el campo de visión de una cámara real al detectar una pieza detenida frente al sensor. Se cargaron de manera individual desde la pestaña Imagen.

A continuación, se integró la herramienta **PatMax Patterns (1–10) with SortPatterns**, una solución para la detección de patrones geométricos complejos. Se configuraron cuatro instancias de esta herramienta, correspondientes a: la base del cubo, la base del cilindro, el propio cubo y el propio cilindro.

Para cada una de ellas, se definió la región de búsqueda (en color morado) y un modelo de patrón (en verde), ajustando manualmente su tamaño y posición para garantizar una detección óptima. Las regiones de búsqueda se basaron en la zona final de la cinta y la base de las figuras.

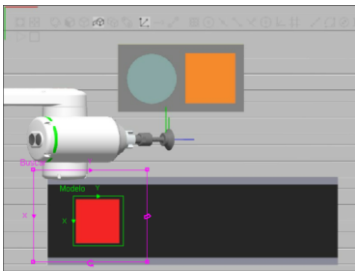


Figura 155. Entrenamiento de la herramienta de ubicación PatMax Patterns (1-10) with SortPatterns

Una vez situadas todas las regiones, se modifica el nombre de la herramienta para facilitar su identificación y se configuran parámetros como el número de coincidencias a detectar, umbrales de aceptación y tolerancias.

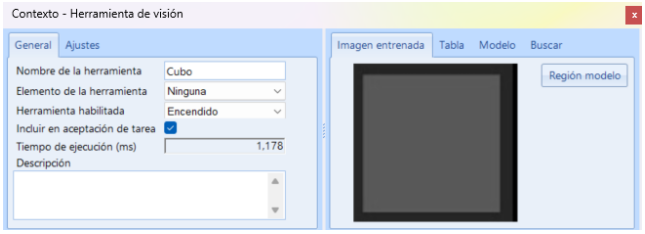


Figura 156. Configuración inicial de la herramienta junto al modelo entrenado

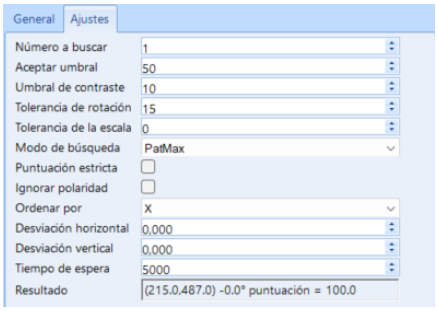


Figura 157. Ajustes de la herramienta de ubicación PatMax Patterns (1-10) with SortPatterns

Posteriormente, se accedió a la opción “Salida a RAPID” para configurar el envío de resultados al programa RAPID. En este caso, se crearon cuatro piezas individuales, una para cada patrón definido. Para cada una de ellas, se seleccionaron como datos de salida la posición en X e Y y la rotación en Z, parámetros esenciales para que el robot pueda ejecutar movimientos precisos en función del objeto detectado.

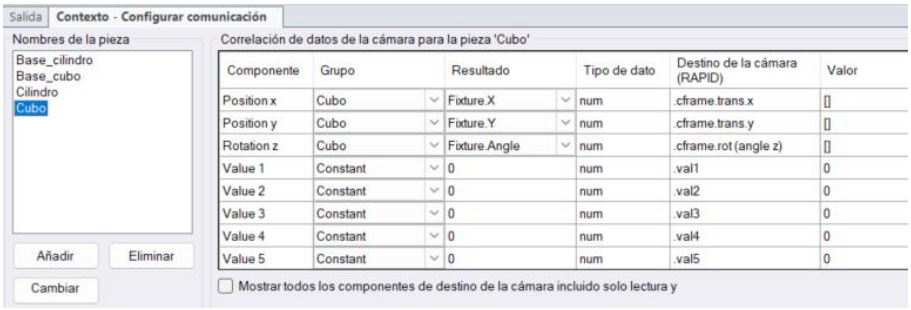


Figura 158. Salidas a RAPID del trabajo de visión job_practica1.job

Finalmente, el trabajo de visión fue guardado bajo el nombre “job_practica1.job” en la carpeta “Sensores In-Sight”, siguiendo la metodología descrita en el apartado [5.2.7 Lógica de funcionamiento de un sistema de visión artificial].

Implementación del código RAPID

Declaración de herramientas, objetos y variables

Al inicio del módulo, se definen todos los recursos necesarios para la ejecución de la práctica. En primer lugar, se declara la herramienta *TCP_Ventosa*, que representa el extremo del robot equipado con la ventosa. A continuación, se crean dos objetos de trabajo (*cam_WObj* y *cam_WObj2*), que actúan como marcos de referencia: uno correspondiente a la altura de las piezas sobre la cinta transportadora y el otro a la altura de la base de clasificación.

Además, se define una constante de tipo string que contiene el nombre del archivo de visión artificial (job_practica1.job) que será cargado en la cámara emulada y se inicializan tres variables de tipo *cameratarget* para almacenar los resultados obtenidos de la tarea de visión: la posición de la base del cubo, del cilindro y de la pieza detectada.

Por último, se declara una posición de referencia denominada “Home”, a la que el robot regresa al inicio y final de cada ciclo.

```
LOCAL PERS tooldata TCP_ToolVentosa:=[TRUE,[[0,0,129.5],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
LOCAL PERS wobjdata cam_WObj:=[FALSE,TRUE,"",[[170,304.269,169.999],[0.00377863,0.999993,0,0]],[[334.591,163.492,0],[1,0,0,0]]];       !Altura cinta
LOCAL PERS wobjdata cam_WObj2:=[FALSE,TRUE,"",[[170,304.269,114.999],[0.00377863,0.999993,0,0]],[[334.591,163.492,0],[1,0,0,0]]];      !Altura base

LOCAL CONST robtarget myrobtarget:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

LOCAL CONST string myjob := "job_practica1.job";
LOCAL VAR cameratarget mycameratarget;
LOCAL VAR cameratarget mycameratarget2;
LOCAL VAR cameratarget mycameratarget3;

LOCAL CONST robtarget Home:=[[540.684087634,0,773.032029102],[0.500000007,0,0.8660254,0],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Figura 159. Declaraciones de la práctica 1

Procedimiento “MoveToDetectedObject()”

Este procedimiento se encarga de toda la lógica de visión artificial. En primer lugar, se configura la cámara pasándola a modo programación (*CamSetProgramMode*), lo cual permite modificar su comportamiento. A continuación, se carga el trabajo de visión previamente entrenado mediante el comando *CamLoadJob*, y posteriormente se establece el modo de ejecución de la cámara (*CamSetRunMode*), dejándola lista para procesar imágenes.

Dado que en este proyecto se emplea una cámara emulada, la adquisición de la imagen no puede gestionarse directamente con el comando *CamReqImage*, por lo que este paso se realiza de forma manual. Para ello, el usuario debe acceder a la pestaña de “Visión” en *RobotStudio*, cambiar a modo programación, cargar manualmente la captura realizada por el componente inteligente “ImageCapture” y regresar al modo ejecución.

Durante ese tiempo, el programa permanece a la espera de la activación de la señal digital “Scalable_IO_0_DI2”, que actúa como confirmación por parte del usuario de que el trabajo de visión ha sido preparado correctamente. Una vez activada esta señal, se procede a obtener los resultados de la tarea de visión utilizando el comando *CamGetResult*.

Dichos resultados se almacenan en tres variables de tipo *cameratarget*, que contienen información sobre la posición y orientación de los objetos detectados.

```
LOCAL PROC MoveToDetectedObject()

    CamSetProgramMode cam;
    CamLoadJob cam, myjob;      !Carga trabajo ya entrenado
    CamSetRunMode cam;

    !Punto de interrupción para carga de imagen manual
    WaitDI Scalable_IO_0_DI2,1;
    CamReqImage cam;

    CamGetResult cam, mycameratarget;
    CamGetResult cam, mycameratarget2;
    CamGetResult cam, mycameratarget3;

ENDPROC
```

Figura 160. Procedimiento "MoveToDetectedObject()" de la práctica 1

Procedimiento "Figura()"

En este procedimiento, en primer lugar, se ajusta el marco de coordenadas *cam_WObj* con la información obtenida desde la tercera variable de tipo *cameratarget*, que corresponde a la posición y orientación del objeto situado al final de la cinta transportadora. A continuación, el robot se desplaza hasta esa posición ejecutando un movimiento conjunto con la activación de la ventosa (*MoveJDo*), simulando la acción de succión para recoger la pieza.

Posteriormente, se evalúa el nombre de la pieza detectada (*mycameratarget3.name*) y, en función del tipo se actualiza el objeto de trabajo (*cam_WObj2*) con las coordenadas correspondientes al destino (extraídas de *mycameratarget* y *mycameratarget2*).

En ambos casos, el robot se traslada hasta dicha posición y durante su movimiento de descenso hacia la posición de liberación, se desactiva la señal de vacío liberando la pieza sobre su base correspondiente.

```
LOCAL PROC Figura()

    cam_WObj.oframe := mycameratarget3.cframe;

    !Efecto succión al llegar a posición
    MoveJDo myrobtarget,v100,fine,TCP_ToolVentosa\WObj:=cam_WObj, Scalable_IO_0_DO15,1;

    IF mycameratarget3.name= "Cubo" THEN
        cam_WObj2.oframe := mycameratarget.cframe;
        cam_WObj.oframe := mycameratarget.cframe;
        MoveJ myrobtarget,v100,fine,TCP_ToolVentosa\WObj:=cam_WObj ;
    ELSEIF mycameratarget3.name= "Cilindro" THEN
        cam_WObj2.oframe := mycameratarget2.cframe;
        cam_WObj.oframe := mycameratarget2.cframe;
        MoveJ myrobtarget,v100,fine,TCP_ToolVentosa\WObj:=cam_WObj ;
    ELSE
    ENDIF

    !Romper vacío ventosa al llegar a destino
    MoveLDo myrobtarget,v50,fine,TCP_ToolVentosa\WObj:=cam_WObj2, Scalable_IO_0_DO15,0 ;

ENDPROC
```

Figura 161. Procedimiento "Figura()" de la práctica 1

Procedimiento "GoHome()"

Este procedimiento traslada el robot a la posición de referencia "Home".

```
LOCAL PROC GoHome()

   MoveJ Home,v200,z100,TCP_ToolVentosa\WObj:=wobj0;

ENDPROC
```

Figura 162. Procedimiento "GoHome()" de la práctica 1

Procedimiento principal "main_Práctica1_ClasificarFormas()"

Este procedimiento representa el ciclo principal que se encarga de coordinar toda la ejecución de la práctica docente.

En primer lugar, se activa la señal de inicio de la cinta transportadora (DO16). El sistema permanece en espera hasta que el sensor situado al final de la cinta detecta una pieza, momento en el cual se detiene el avance de la cinta desactivando dicha señal.

Es en ese momento cuando se activa la señal de captura de imagen (DO14), lo que conlleva que el componente inteligente "ImageCapture" tome una fotografía del área de visión. A continuación, se llama a la función "MoveToDetectedObject()" que carga el trabajo de visión, obtiene los resultados y almacena la información relativa a las piezas detectadas. Con estos datos, se ejecuta el procedimiento "Figura()", que gestiona la recogida y colocación de la pieza en su ubicación correspondiente dentro de la base de clasificación.

Finalmente, se resetean todas las señales digitales utilizadas durante el ciclo y se ejecuta la instrucción "GoHome()", devolviendo el robot a su posición de inicio antes de comenzar un nuevo ciclo.

```
PROC main_Práctica1_ClasificarFormas()

    WHILE TRUE DO

        SetDO Scalable_IO_0_DO16,1;  !Inicio cinta
        WaitDI Scalable_IO_0_DI16,1; !Espera pieza al final
        SetDO Scalable_IO_0_DO16,0;  !Paro cinta

        SetDO Scalable_IO_0_DO14,1;  !Capturar foto
        SetDO Scalable_IO_0_DO14,0;

        MoveToDetectedObject;
        Figura;

        !Reset final de las señales
        Reset Scalable_IO_0_DO14;
        Reset Scalable_IO_0_DO15;
        Reset Scalable_IO_0_DO16;

        GoHome;

    ENDWHILE

ENDPROC
```

Figura 163. Procedimiento principal "main_Práctica1_ClasificarFormas()"

Resultados

La interfaz de visión de *RobotStudio* incluye un panel de resultados visuales que permite validar de forma rápida y eficiente el reconocimiento de piezas. Este panel muestra indicadores clave como un círculo verde (*Pass*) o un cuadrado rojo (*Fail*) para señalar si la pieza ha sido detectada correctamente. También se presentan las coordenadas X, Y y la rotación Z de cada figura identificada, junto con el nombre del

patrón asignado. Si alguna figura no se detecta en la imagen, se indica con una cruz roja en la columna correspondiente a los resultados de posición.

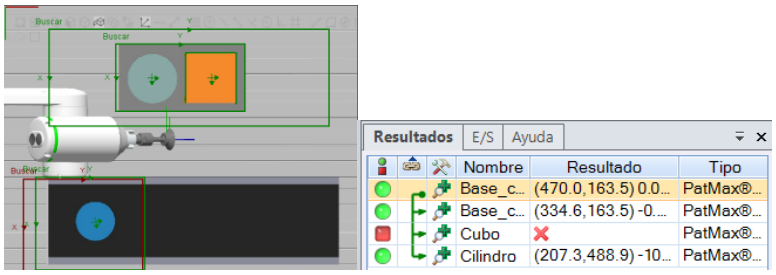


Figura 164. Resultados trabajo de visión con un cilindro

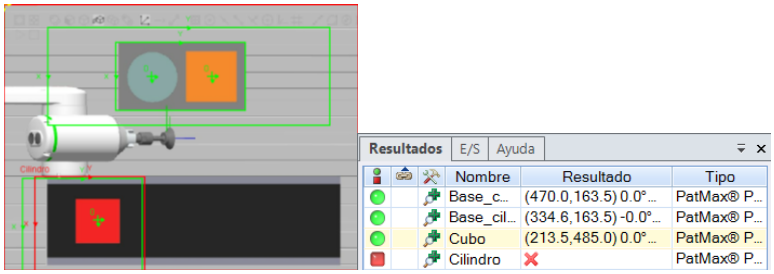


Figura 165. Resultados trabajo de visión con un cubo

La herramienta **PatMax Patterns with SortPatterns** ha demostrado ser altamente precisa, permitiendo reconocer las figuras incluso con ligeras variaciones en posición u orientación.

Para garantizar el correcto funcionamiento, se realizaron multiples pruebas con ambas figuras. En todos los casos, el robot fue capaz de detener la cinta al detectar la pieza, capturar la imagen en el momento oportuno, clasificar la pieza según su forma, manipularla y colocarla correctamente sobre la base correspondiente.

Estos resultados validan el éxito de la práctica como ejercicio docente introductorio a la robótica colaborativa con visión artificial.

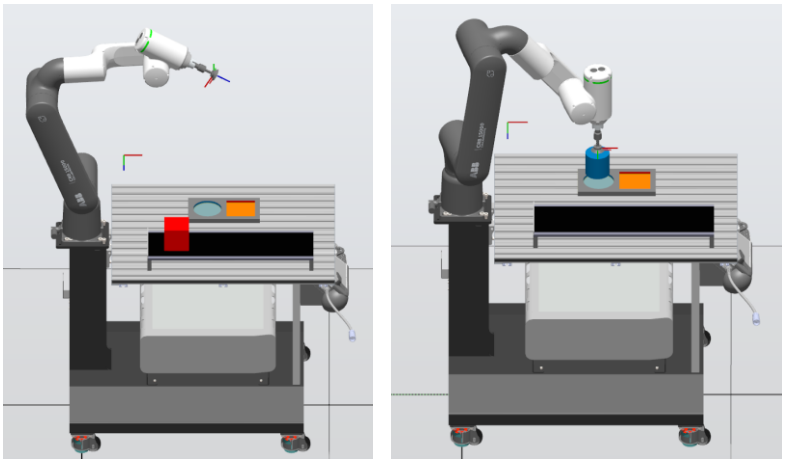


Figura 166. Proceso de clasificación de figuras de la práctica 1

5.3.3. Práctica 2: Clasificación por colores

Esta práctica consiste en la clasificación automática de piezas según su color, utilizando visión artificial y un sistema robótico con un conveyor.

Las piezas, que son cubos de color rojo, verde o azul, se generan aleatoriamente sobre el conveyor. Este se detiene cuando una pieza es detectada por el sensor situado al final del recorrido. En ese instante, se captura una imagen mediante el componente inteligente, la cual es procesada mediante una tarea de visión que utiliza la herramienta **Color Blobs** para identificar el color de la pieza y su posición.

Una vez clasificada, el robot la recoge y la deposita en el contenedor correspondiente a su color. Finalmente, se reanuda automáticamente el movimiento del conveyor para continuar el proceso.

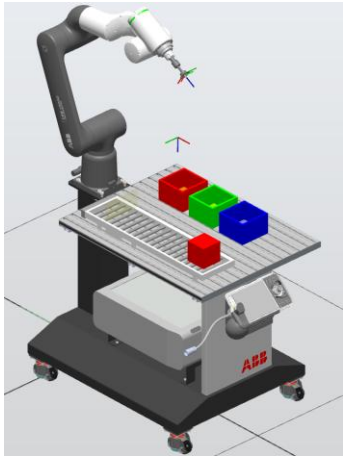


Figura 167. Estación de la práctica 2 Clasificación por colores

Modelado de componentes

Modelado de las piezas

Para esta práctica, se han utilizado cubos de tres colores diferentes (rojo, verde y azul) como piezas a clasificar. Dado que ya se disponía del modelo de cubo empleado en la práctica anterior, se ha optado por reutilizar dicho modelo para evitar tareas de modelado innecesarias.

El procedimiento consiste simplemente en copiar tres veces el cubo original dentro del grupo de componentes correspondiente a la práctica 2, modificando en cada copia el color para representar los distintos tipos de pieza.

Modelado de los contenedores de clasificación

Para la correcta clasificación de los cubos según su color, se han diseñado tres contenedores. Cada uno de ellos se ha construido mediante una operación de resta entre dos solidos para obtener una forma hueca.

1. **Modelado del contenedor:** En primer lugar, se crea un cubo que representa el cuerpo exterior de nuestro contenedor. Se han definido las dimensiones de 150x150x100mm. Para ello, se accede a la pestaña "Modelado", sección "Crear" y se selecciona la opción "Tetraedro" dentro del desplegable sólido.
2. **Modelado del hueco interior:** A continuación, se genera un segundo cubo de dimensiones ligeramente inferiores (130×130×90mm), que actuará como volumen de sustracción. Este define el espacio hueco dentro del contenedor, permitiendo la correcta introducción de las piezas.
3. **Posicionamiento del hueco:** El segundo cubo debe colocarse dentro del primero, procurando dejar un grosor uniforme en las paredes. Se ha centrado en planta y situado ligeramente por encima del fondo, simulando una cavidad profunda pero no completamente perforada.

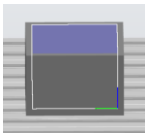


Figura 168. Posicionamiento del hueco dentro del tetraedro

4. **Operación de resta:** Una vez posicionados ambos sólidos, se emplea la herramienta "Restar" del menú "Modelado". Se selecciona el cubo exterior como cuerpo principal y el cubo interior como cuerpo a sustraer. Es importante desmarcar la opción "Conservar original" para eliminar el cubo interior tras la operación.

Una vez obtenido el contenedor final, este se ha duplicado dos veces y se ha aplicado a cada uno un color diferente (rojo, verde y azul). Cada uno de ellos ha sido posicionado en el entorno de trabajo en una ubicación adecuada, accesible para el robot y visible desde el área de captura de la cámara emulada. Finalmente, se han integrado dentro del grupo de componentes correspondiente a esta práctica.

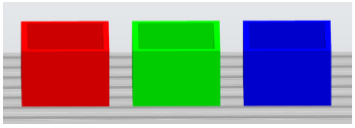


Figura 169. Contenedores para la clasificación por colores de las piezas

Componente inteligente: Conveyor_colores

En primer lugar, se ha importado el modelo 3D de un conveyor. Para la generación automática de cubos de diferentes colores y su desplazamiento a lo largo del conveyor, se ha desarrollado un componente inteligente llamado "Conveyor_colores". Este componente gestiona tanto la creación aleatoria de las piezas como su movimiento y detección al final de la línea.

Estructura del componente inteligente

1. **Entrada digital de control ("Inicio_cinta"):** Esta señal permite activar o desactivar el funcionamiento del conveyor desde el controlador del robot.

2. **Temporizador ("Timer"):** Se ha incorporado un bloque "Timer" para controlar el intervalo de generación de piezas. En este caso, se ha configurado para un intervalo de dos segundos, repitiéndose de forma indefinida activando la lógica de generación de piezas.
3. **Generador aleatorio ("Random"):** Para garantizar la aparición aleatoria de diferentes figuras, se ha incorporado un bloque "Random", que genera un valor decimal entre 0 y 3.
4. **Comparadores ("Comparer"):** El valor generado se evalúa mediante tres bloques "Comparer":
   - **Si valor≤1:** Se genera un cubo rojo.
   - **Si 1<valor≤ 2**: Se genera un cubo verde.
   - **Si valor>2:** Se genera un cubo azul.

   Esta configuración permite una probabilidad equitativa del 50% para cada tipo de figura.
5. **Generador de piezas ("Source"):** Se han incorporado tres bloques de generación ("Source_CubosRojos", "Source_CubosVerdes" y "Source_CubosAzules"), cada uno asociado a un color específico. En ellos se especifican tanto el modelo 3D que se va a instanciar como su posición y orientación iniciales sobre el conveyor. Estos bloques generan una copia del modelo correspondiente en la posición deseada.
6. **Cola ("Queue"):** El bloque "Queue" actúa como un almacén temporal dónde se gestionan las piezas generadas. Cada copia generada de las piezas se inserta en el back de la cola mediante "Enqueue", y luego son procesadas en orden.
7. **Movimiento lineal:** Se ha utilizado un bloque "LinearMover" para llevar a cabo un desplazamiento de las piezas en la dirección deseada, simulando el avance de la cinta transportadora. Se ha definido una dirección hacia delante (eje Y positivo) y una velocidad de 250mm/s.
8. **Sensor del final de línea:** Se ha incorporado un sensor plano al final del recorrido de la cinta, encargado de detectar la llegada de cada pieza a la posición final. Cuando se activa, este sensor emite una señal digital de salida que indica la presencia de una pieza en el extremo de la línea.
9. **Sensores contenedores de clasificación:** Adicionalmente, se ubicado tres sensores individuales en la base de cada uno de los contenedores de clasificación. Estos detectan si una pieza ha sido correctamente depositada en él. Para evitar la acumulación de piezas en la simulación, cada sensor se conecta a la entrada "Delete" del bloque "Queue", eliminando el objeto del entorno cuando este sensor lo detecte.
10. **Salidas digitales:**
    - **"Caja_PosicionFinal":** Se activa cuando el sensor del final del conveyor detecta una pieza.
    - **"Rojo", "Verde", "Azul":** Se activan cuando una pieza ha sido correctamente colocada en su contenedor correspondiente, sirviendo como validación del proceso de clasificación.

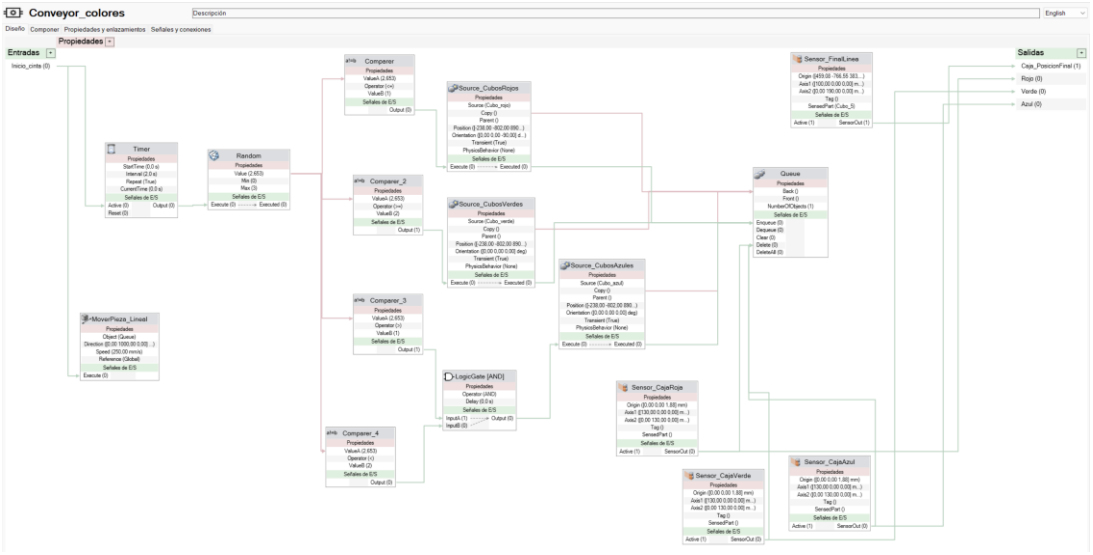


Figura 170. Lógica del componente inteligente del conveyor de la práctica 2 (Conveyor_colores)

El comportamiento de este componente inteligente está conectado mediante señales digitales al controlador del robot. De este modo, el conveyor se detiene automáticamente cuando una pieza es detectada por el sensor de fin de línea, y permanece parada hasta que el sistema de visión realiza la clasificación y el robot completa la manipulación.

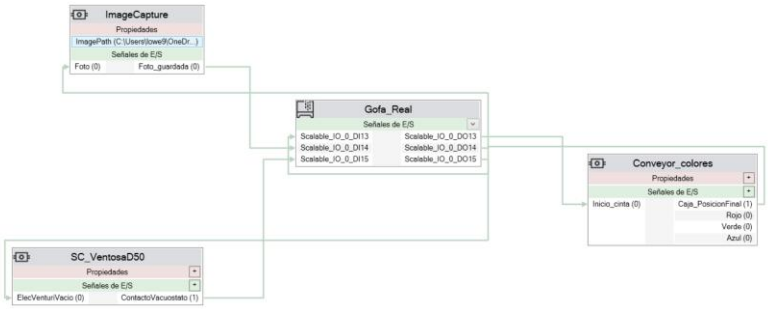


Figura 171. Conexión del componente inteligente "Conveyor_colores" al controlador virtual

Tarea de visión

El objetivo de esta tarea de visión es identificar el color de los cubos que lleguen al final del conveyor, permitiendo clasificarlos correctamente en distintos contenedores. Para lograrlo se ha empleado la herramienta **Blob de color (1-10),** que permite detectar objetos según sus características cromáticas.

En primer lugar, se capturaron tres imágenes distintas con el componente inteligente diseñado para tal propósito, cada una con un cubo de un color diferente, posicionados al final de la línea cuando el sensor detecta dicha pieza.

Una vez dentro del entorno de visión, se creó un nuevo trabajo de visión y para cada imagen se procedió a añadir una nueva herramienta de ubicación llamada Blob de

color al trabajo creado. Al seleccionar esta herramienta, se debe definir la región de búsqueda (área morada) situándola sobre el cubo que se desea identificar.

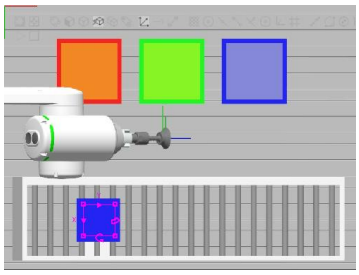


Figura 172. Selección de la región de búsqueda de la herramienta Blob de color (1-10)

A continuación, se accede a la pestaña de "Ajustes" de la herramienta Blob y se selecciona la opción "Entrenar color". Esta acción abre una nueva ventana dedicada al entrenamiento del modelo de color. En dicha ventana se realiza el siguiente procedimiento:

1. Crear un nuevo modelo de color.
2. Hacer clic en "Sumar nuevo color".
3. Aparece un círculo sobre la imagen, el cual debe ser colocado cuidadosamente sobre la zona de la pieza que representa el color que se desea capturar, evitando incluir sombras o reflejos.

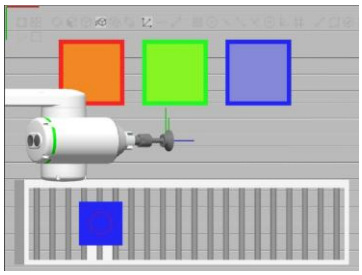


Figura 173. Entrenamiento del color de la pieza mediante la herramienta

4. Una vez situado, se presiona la tecla "Enter", y el color se añade al modelo con un nombre identificativo.

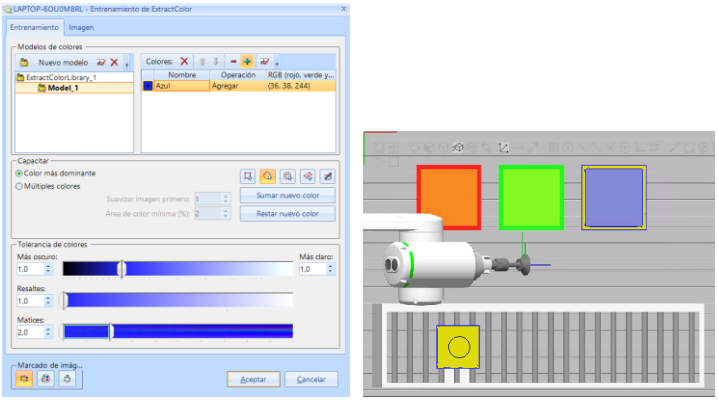


Figura 174. Configuración del modelo de colores creado

5. Este proceso se repite para cada uno de los colores representados por los cubos.

Una vez completado el entrenamiento de los tres colores, en la misma pestaña de ajustes de cada herramienta Blob se selecciona el color correspondiente desde la biblioteca de colores entrenados. Además, se define el modo de operación de la herramienta. En este caso, se ha optado por el modo "Identificar", el cual permite que el sistema reconozca de qué color se trata, devolviendo su nombre como resultado. También hubiera sido posible emplear el modo "Coincidir", útil cuando solo se desea verificar la presencia de un color específico, sin clasificarlo.

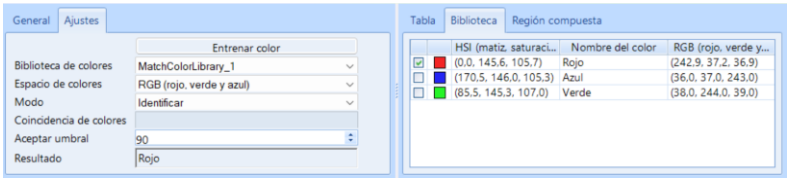


Figura 175. Biblioteca de los colores entrenados

Posteriormente, cada una de estas herramientas de detección se configura para enviar los resultados al programa RAPID. Para ello, se han definido tres salidas distintas, una por cada color detectado, asignándoles nombres representativos para facilitar su identificación en el código.

En este caso, se ha optado por transmitir únicamente un valor numérico como salida: 1 en caso de detección correcta (*pass*) y 0 en caso contrario (*fail*). Aunque la herramienta también permite enviar datos adicionales como coordenadas de posición y orientación del objeto, se ha decidido no utilizar esta funcionalidad, ya que se han definido posiciones fijas dentro del entorno de *RobotStudio*, tanto para el punto de recogida al final del conveyor como para cada uno de los contenedores de clasificación.

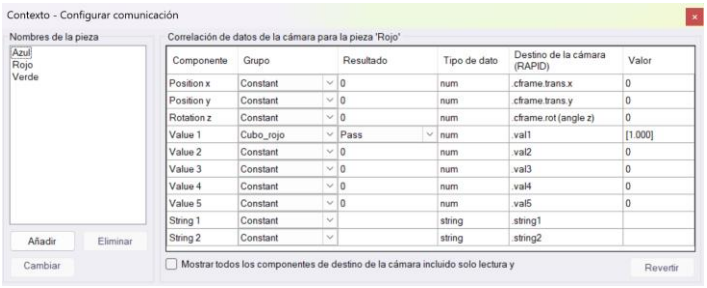


Figura 176. Salidas a RAPID del trabajo de visión job_practica2.job

Finalmente, el trabajo de visión fue guardado bajo el nombre "job_practica2.job" en la carpeta "Sensores In-Sight", siguiendo la metodología descrita en el apartado [5.2.7 Lógica de funcionamiento de un sistema de visión artificial].

Implementación del código RAPID

Declaración de herramientas, objetos de trabajo y posiciones

En primer lugar, se declara la herramienta de succión *TCP_ToolVentosa*, que será utilizada para la manipulación de las piezas. A continuación, se definen tres objetos de trabajo (*Caja_roja*, *Caja_verde* y *Caja_azul*), que representan las ubicaciones correspondientes a cada uno de los contenedores de clasificación.

También se especifican varias posiciones en el espacio de trabajo: una posición de referencia inicial y final del ciclo denominada "Home", las posiciones de precogida y cogida de las piezas al final del conveyor, y las posiciones de predejada y dejada para depositarlas en cada uno de los contenedores según su color.

Además, se declara el nombre del trabajo de visión correspondiente a esta práctica (job_practica2.job), una variable del tipo *cameratarget* para almacenar los resultados obtenidos por el sistema de visión artificial, y tres contadores para registrar la cantidad de piezas clasificadas por color durante la ejecución del programa.

```
LOCAL PERS tooldata TCP_ToolVentosa:=[TRUE,[[0,0,121.5],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];

LOCAL PERS wobjdata Caja_roja:=[FALSE,TRUE,"",[[289.753,229.519,112.094],[0,1,0,0]],[[0,0,0],[1,0,0,0]]];
LOCAL PERS wobjdata Caja_verde:=[FALSE,TRUE,"",[[479.753,229.519,112.094],[0,1,0,0]],[[0,0,0],[1,0,0,0]]];
LOCAL PERS wobjdata Caja_azul:=[FALSE,TRUE,"",[[669.753,229.519,112.094],[0,1,0,0]],[[0,0,0],[1,0,0,0]]];

LOCAL CONST robtarget Home:=[[540.684087634,0,773.032029102],[0.500000007,0,0.8660254,0],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

LOCAL CONST robtarget PreCoger_piezas:=[[387.586876338,-187.999981439,269.715035304],[0.000000014,-0.707106746,0.707106817,0.000000074],[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Coger_piezas:=[[387.586876338,-187.999981439,169.715035304],[0.000000014,-0.707106746,0.707106817,0.000000074],[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

LOCAL CONST robtarget PreDejar_rojo:=[[75.000015117,74.999984623,-135.000000161],[0.707106805,-0.000000011,0.000000008,0.707106758],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Dejar_rojo:=[[75.000023514,74.999990537,-10.000006201],[0.707106786,0.000000014,0.000000006,0.707106776],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

LOCAL CONST robtarget PreDejar_verde:=[[75.000015117,74.999984623,-135.000000161],[0.707106805,-0.000000011,0.000000008,0.707106758],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Dejar_verde:=[[75.000023514,74.999990537,-10.000006201],[0.707106786,0.000000014,0.000000006,0.707106776],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

LOCAL CONST robtarget PreDejar_azul:=[[75.000015117,74.999984623,-135.000000161],[0.707106805,-0.000000011,0.000000008,0.707106758],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Dejar_azul:=[[75.000023514,74.999990537,-10.000006201],[0.707106786,0.000000014,0.000000006,0.707106776],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

LOCAL CONST string myjob := "job_practica2.job";

LOCAL VAR cameratarget mycameratarget;

LOCAL VAR num contador_rojas:=0;
LOCAL VAR num contador_verdes:=0;
LOCAL VAR num contador_azules:=0;
```

Figura 177. Declaraciones iniciales de la práctica 2

Procedimiento "MoveToDetectedObject()"

Este procedimiento se encarga de gestionar la ejecución del trabajo de visión artificial. En primer lugar, se cambia la cámara al modo programación, se carga el archivo .job previamente entrenado y se activa el modo ejecución para iniciar el procesamiento.

A continuación, el programa espera a que el usuario complete manualmente la carga de imagen desde la interfaz de *RobotStudio*, utilizando la fotografía generada previamente por el componente inteligente *ImageCapture*. Una vez procesada la imagen, los resultados se almacenan en la variable *mycameratarget*.

Mediante un bucle WHILE, el sistema consulta de forma continua los datos recibidos hasta que se detecte correctamente uno de los colores entrenados (condición: mycameratarget.val1 = 1). Una vez identificado, se llama automáticamente al procedimiento "Seleccion_caja()" para ejecutar la manipulación correspondiente.

```
LOCAL PROC MoveToDetectedObject()

    CamSetProgramMode cam;
    CamLoadJob cam, myjob;      !Carga trabajo ya entrenado
    CamSetRunMode cam;

    WaitDI Scalable_IO_0_DI2,1;
    CamReqImage cam;

    CamGetResult cam, mycameratarget;

    !mycameratarget.val1=1 cuando se detecta coincidencia de color
    !El bucle espera hasta que la cámara identifique un color entrenado
    WHILE mycameratarget.val1 <>1 DO

        !Actualiza los resultados de la cámara en cada iteración
        CamGetResult cam, mycameratarget;

    ENDWHILE

    Seleccion_caja;

ENDPROC
```

Figura 178. Procedimiento "MoveToDetectedObject()" de la práctica 2

Procedimiento "Selección_caja()"

Este procedimiento realiza el agarre de la pieza ubicada al final del conveyor activando la ventosa. Posteriormente, en función del nombre del color identificado almacenado en mycameratarget.name, se llama al procedimiento asociado: "Pieza_roja()", "Pieza_verde()" o "Pieza_azul()". En caso de que no se identifique un color válido, se muestra un mensaje de error en la FlexPendant.

```
LOCAL PROC Seleccion_caja()

    MoveJ PreCoger_piezas,v100,z10,TCP_ToolVentosa\WObj:=wobj0;

    !Efecto succión al llegar a posición
    MoveLDO Coger_piezas,v50,fine,TCP_ToolVentosa\WObj:=wobj0,Scalable_IO_0_DO15,1;

    !mycameratarget.name almacena el nombre del color identificado
    !Se ejecuta la rutina correspondiente a cada color identificado
    IF mycameratarget.name="Rojo" THEN
        Pieza_roja;
    ELSEIF mycameratarget.name="Verde" THEN
        Pieza_verde;
    ELSEIF mycameratarget.name="Azul" THEN
        Pieza_azul;
    ELSE
        TPWrite "Fallo en la detección del color";
    ENDIF

ENDPROC
```

Figura 179. Procedimiento "Selección_caja()" de la práctica 2

Procedimiento "GoHome()"

Este procedimiento simplemente retorna el robot a la posición inicial "Home" una vez finalizada la operación de clasificación, preparando el sistema para un nuevo ciclo.

Procedimientos "Pieza_roja()", "Pieza_verde()", "Pieza_azul()"

Cada uno de estos procedimientos realiza el movimiento hacia el contenedor correspondiente. El robot se sitúa en la posición previa, desciende para depositar la pieza y desactiva la ventosa. Además, se incrementa el contador de piezas clasificadas de dicho color.

```
LOCAL PROC Pieza_roja()

    MoveJ PreDejar_rojo,v100,z10,TCP_ToolVentosa\WObj:=Caja_roja;

    !Romper vacío ventosa al llegar a destino
    MoveLDO Dejar_rojo,v50,fine,TCP_ToolVentosa\WObj:=Caja_roja, Scalable_IO_0_DO15,0;

    contador_rojas:=contador_rojas+1;

ENDPROC
```

Figura 180. Procedimiento "Pieza_roja()" de la práctica 2

Procedimiento principal "main_Práctica2_ClasificarColores()"

Este es el bucle principal del programa, que ejecuta de forma continua el proceso de clasificación. En cada iteración, se imprime en el FlexPendant el número de piezas clasificadas por cada color, así como el total acumulado.

Seguidamente, se activa el conveyor (DO13) y el sistema espera hasta que el sensor final (DI13) detecte una nueva pieza. En ese momento, se detiene el conveyor y se activa la captura de imagen (DO14) a través del componente "ImageCapture". A continuación, se ejecuta el procedimiento "MoveToDetectedObject()" para realizar la detección y clasificación de la pieza.

Una vez finalizado todo el proceso, se reinician las señales digitales involucradas y el robot vuelve a su posición de inicio mediante "GoHome()", cerrando así el ciclo y preparándose para el siguiente.

```
PROC main_Práctica2_ClasificarColores()

    WHILE TRUE DO

        TPWrite "Piezas almacenadas de cada color";
        TPWrite " Piezas Rojas: "\Num:=contador_rojas;
        TPWrite " Piezas Verde: "\Num:=contador_verdes;
        TPWrite " Piezas Azules: "\Num:=contador_azules;
        TPWrite "Piezas totales: "\Num:=contador_rojas+contador_verdes+contador_azules;

        SetDO Scalable_IO_0_DO13,1;  !Inicio conveyor
        WaitDI Scalable_IO_0_DI13,1; !Espera pieza
        SetDO Scalable_IO_0_DO13,0;  !Paro conveyor

        SetDO Scalable_IO_0_DO14,1;  !Capturar foto
        SetDO Scalable_IO_0_DO14,0;

        MoveToDetectedObject;
        GoHome;

        !Reset final de las señales
        Reset Scalable_IO_0_DO13;
        Reset Scalable_IO_0_DO14;
        Reset Scalable_IO_0_DO15;

    ENDWHILE

ENDPROC
```

Figura 181. Procedimiento principal "main_Práctica2_ClasificarColores()"

Este programa permite ejecutar de forma cíclica y autónoma el proceso de clasificación, ofreciendo también al usuario información actualizada del número de piezas clasificadas por color.

Resultados

Al igual que en la práctica anterior, los resultados del trabajo de visión se pueden visualizar en el panel de resultados del entorno de *RobotStudio*. En este panel aparece un cuadrado rojo en aquellas herramientas que no han logrado identificar una

coincidencia (*fail*) y un círculo verde en las que sí lo han hecho correctamente (*pass*). Cuando se detecta una figura, se muestra su color asignado; en caso contrario, aparece una cruz roja acompañada del mensaje “Ninguna coincidencia: Fuera de rango”.

A continuación, se presentan tres capturas del sistema de visión, cada una correspondiente a un color diferente, junto con su respectivo panel de resultados, lo que permite validar visualmente el correcto funcionamiento del trabajo de visión artificial implementado.

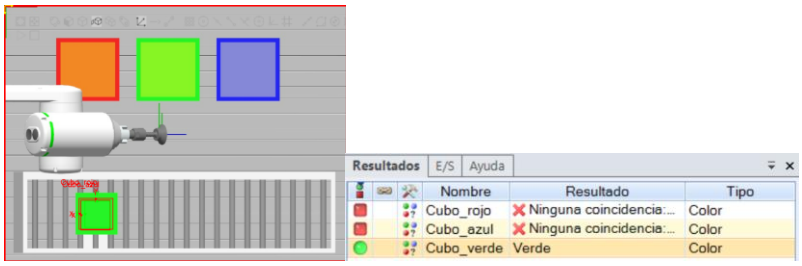


Figura 182. Resultados trabajo de visión con un cubo verde

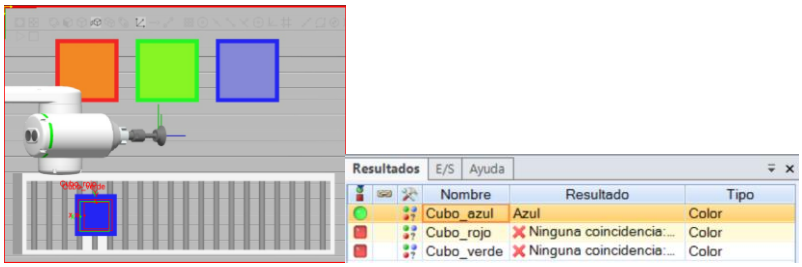


Figura 183. Resultados trabajo de visión con un cubo azul

Figura 184. Resultados trabajo de visión con un cubo rojo

Durante el desarrollo de la práctica, se llevaron a cabo numerosas pruebas con piezas de los tres colores disponibles (rojo, verde y azul). En todos los casos, el sistema fue capaz de detectar correctamente la presencia de la pieza, identificar su color mediante la herramienta de visión, y ejecutar de forma precisa la secuencia completa: detener la cinta transportadora, capturar la imagen, procesar los datos, manipular la figura con la herramienta de succión y depositarla en el contenedor correspondiente. Además, los mensajes de recuento mostrados en la FlexPendant se actualizaron correctamente tras cada ciclo de trabajo, como puede observarse en la figura 186.

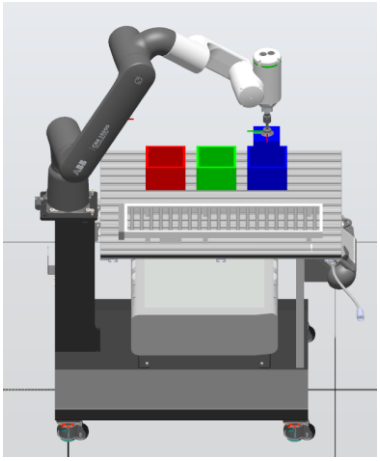


Figura 185. Proceso de clasificación de figuras de la práctica 2

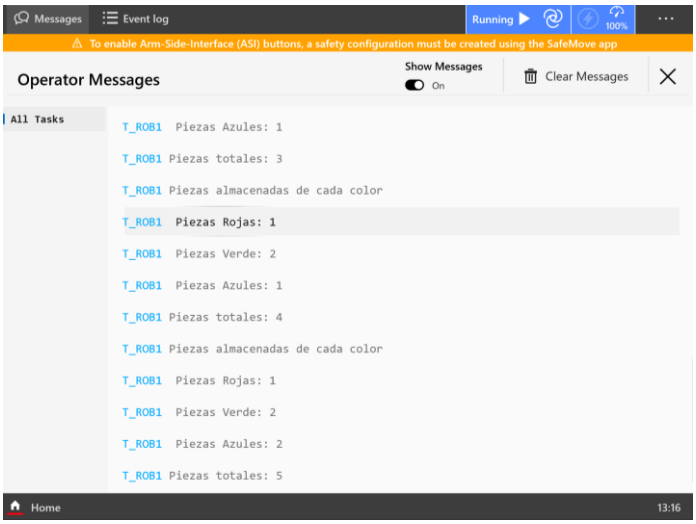


Figura 186. Mensajes de recuento de piezas mostrados en la FlexPendant

La correcta ejecución del sistema confirma su robustez y eficacia en tareas básicas de clasificación por color.

5.3.4. Práctica 3: Detectar defectos en las medidas

Esta práctica tiene como objetivo identificar automáticamente piezas defectuosas en función de sus dimensiones, utilizando visión artificial y un sistema robótico con una cinta transportadora. Las piezas generadas son tetraedros de diferentes tamaños, entre los cuales solo se consideran válidos los cubos que presentan unas dimensiones exactas de 100x100mm.

Estas piezas se generan aleatoriamente mediante un componente inteligente de la cinta transportadora. Cuando una pieza es detectada por el sensor situado en el medio de la línea, la cinta se detiene y se captura una imagen mediante el componente inteligente “ImageCapture”. Esta es analizada por el trabajo de visión configurado previamente, que emplea la herramienta de medición para determinar si las dimensiones de la pieza cumplen con los requisitos.

Una vez inspeccionada, si la pieza se considera defectuosa, el robot la recoge y la deposita en un contenedor destinado a tal fin. En caso de que la pieza sea válida, continúa su trayecto por la cinta transportadora hasta desaparecer al final de la línea, simulando su integración en un proceso posterior.

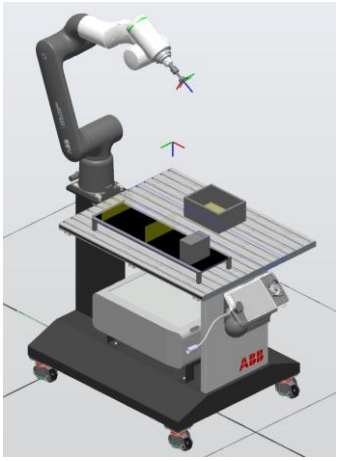


Figura 187. Estación de la práctica 3 Defectos medidas

Modelado de componentes

Modelado pieza

Para la simulación de las piezas inspeccionadas, se ha reutilizado el modelo de cubo desarrollado en la práctica 1. Este ha sido copiado directamente al grupo de componentes correspondiente a la práctica 3, manteniendo sus dimensiones base de 100×100 mm, que se consideran las medidas válidas dentro del proceso de inspección.

El sistema contempla la presencia de piezas defectuosas, las cuales se simulan mediante la variación automática de las dimensiones del tetraedro. Esta variabilidad se gestiona a través del componente inteligente “Conveyor_Cubos”, encargado de generar cubos con diferentes tamaños de forma aleatoria. De este modo, se crea un flujo continuo de piezas válidas y defectuosas, lo que permite validar eficazmente el funcionamiento de la tarea de visión encargada de detectar errores dimensionales.

Modelado contenedor de piezas defectuosas

El contenedor destinado a recoger las piezas defectuosas ha sido modelado siguiendo el mismo procedimiento descrito en la práctica anterior para los contenedores de clasificación por color.

En primer lugar, se ha creado un sólido de tipo “Tetraedro” que actúa como cuerpo exterior del contenedor. Para ello, se han definido unas dimensiones de 240×170×100 mm, lo suficientemente amplias para garantizar la capacidad de almacenar las piezas descartadas. A continuación, se ha modelado un segundo tetraedro, ligeramente más pequeño, que representa el volumen interior o hueco del contenedor.

Ambos cuerpos se han alineado cuidadosamente, asegurando una separación uniforme entre sus paredes. Posteriormente, se ha aplicado una operación de resta mediante la herramienta correspondiente del entorno de modelado, seleccionando primero el cuerpo exterior como sólido principal y luego el interior como objeto a sustraer.

Finalmente, el contenedor resultante se ha integrado en el área de trabajo, dentro del grupo de componentes correspondiente a esta práctica, ubicándose en una posición accesible para que el robot pueda depositar en él las piezas defectuosas con precisión.

Componente inteligente: Conveyor_Cubos

El componente inteligente ha sido diseñado para generar automáticamente piezas con dimensiones variables, simular su transporte por una cinta transportadora y gestionar su clasificación según los resultados obtenidos por el sistema de visión artificial.

Estructura del componente

1. **Entrada digital de control ("Inicio_cinta"):** Señal que activa o desactiva el funcionamiento de la cinta transportadora desde el programa RAPID.
2. **Temporizador ("Timer"):** Este bloque establece el intervalo de generación de las piezas.
3. **Generador aleatorio y comparadores ("Random", "Comparer", "Comparer_2"):** El bloque "Random" genera un valor decimal aleatorio entre 0 y 1, el cual es evaluado mediante dos comparadores para decidir si se generará una pieza correcta o defectuosa.

   La lógica de decisión es la siguiente:
   - **Valor<0.6:** Se genera una pieza correcta (cubo 100x100x100mm)
   - **Valor≥0.6:** Se genera una pieza defectuosa (dimensiones aleatorias)

   Este reparto permite que aproximadamente el 60% de las piezas sean válidas mientras que el 40% sean defectuosas.
4. **Generadores aleatorios ("Random_3" y "Random_4"**): Estos bloques definen las dimensiones X e Y de las piezas defectuosas. Aunque las conexiones no están visibles dentro del componente, estos generadores se encuentran conectados externamente a un bloque "ParametricBox", responsable de generar cubos con medidas variables. Los rangos de los valores aleatorios se han configurado para asegurar que las piezas generadas se mantengan dentro del ancho útil de la cinta transportadora.
5. **Bloques de generación de piezas ("Source_Cubos" y "Source"):** El bloque "Source_Cubos" genera las piezas correctas utilizando el modelo 3D de cubo definido previamente. Por otro lado, "Source" se encarga de instanciar piezas defectuosas, utilizando el bloque externo "ParametricBox" como referencia geométrica.

6. **Cola ("Queue") y movimiento lineal ("MoverPieza_Lineal"):** La cola "Queue" actúa como buffer de almacenamiento para las piezas generadas, mientras que el bloque "MoverPieza_Lineal" simula su desplazamiento a lo largo del eje Y, con una velocidad constante de 250 mm/s, replicando el funcionamiento de una cinta transportadora real.
7. **Sensores ("Sensor_Foto", "Sensor_FinalCinta", "Sensor_Defectos"):**
   - **"Sensor_Foto":** Situado en la mitad del recorrido, detecta cuándo una pieza está en posición para ser inspeccionada y activa el componente "ImageCapture" para capturar una imagen.
   - **"Sensor_FinalCinta":** Ubicado al final de la línea, detecta cuando una pieza ha superado la inspección, por lo que es correcta.
   - **"Sensor_Defectos":** Se encuentra en el fondo del contenedor de piezas defectuosas. Además, también confirma que el robot ha depositado correctamente una pieza rechazada.
8. **"Sink":** Este bloque elimina del sistema cualquier pieza conectada a él. Está vinculado tanto al "Sensor_FinalCinta" como al "Sensor_Defectos", por lo que las piezas son retiradas automáticamente cuando se detectan en cualquiera de estos dos puntos, simulando su retirada del proceso productivo o su reciclaje.
9. **Señal salida digital ("Caja_PosicionFoto"):** Esta señal se activa cuando el sensor ubicado en la mitad del recorrido detecta una pieza. En ese momento, se detiene automáticamente el avance de la cinta transportadora y se activa el componente inteligente de captura de imagen, simulando el disparo de una cámara para iniciar el análisis visual de la pieza.

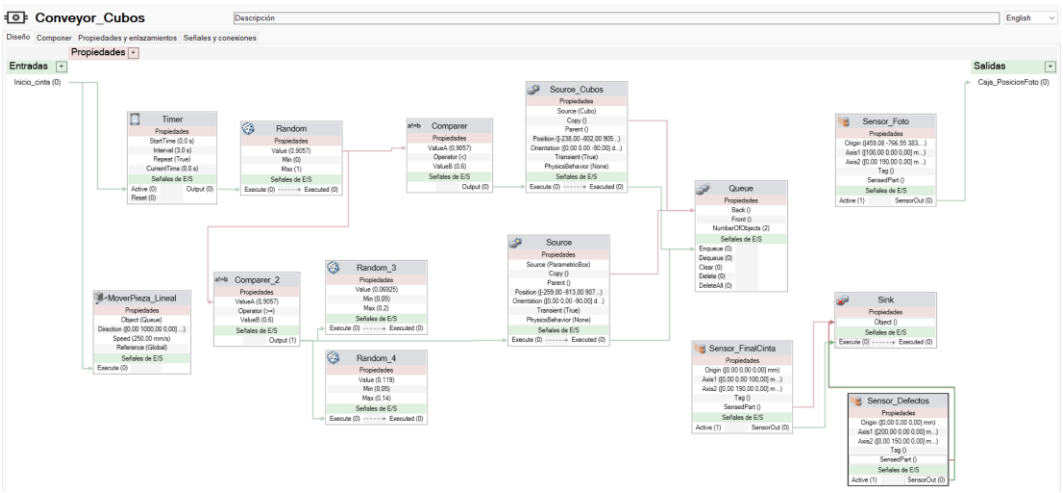


Figura 188. Lógica del componente inteligente de la cinta transportadora de la práctica 3 (Conveyor_Cubos)

Finalmente, este componente inteligente debe ser conectado al controlador del robot mediante señales digitales, lo que permite gestionar su funcionamiento directamente desde el programa RAPID y garantizar la coordinación con el resto del sistema automatizado.

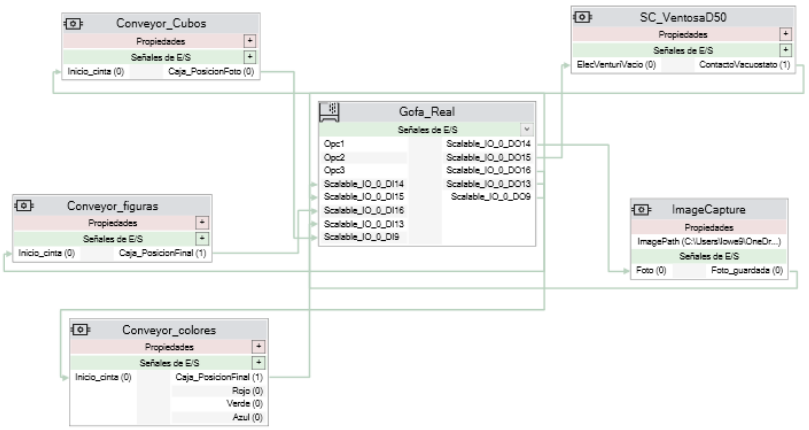


Figura 189. Conexión del componente inteligente "Conveyor_Cubos" al controlador virtual

Tarea de visión

El objetivo de la tarea de visión es verificar si las piezas transportadas cumplen con las dimensiones de 100x100mm. Para ello se utiliza una cámara calibrada y la herramienta de inspección Distancia del software de visión.

1. **Calibración de la cámara:** En primer lugar, se realiza la calibración de tipo X/Y 1:1 en milímetros. Esto permite que todas las medidas extraídas se expresen en unidades reales (mm) y no en píxeles.
2. **Herramienta de inspección (Distancia):** Se añaden dos herramientas de Distancia, una para cada eje (X e Y). Estas herramientas miden la distancia entre los bordes opuestos de la pieza en ambas direcciones.
   Al elegir la herramienta **Distancia**, hay que seleccionar dos de las características inteligentes resaltadas en la imagen, en nuestro caso los bordes de la pieza.
   Además, hay que configurar los siguientes parámetros:
   - **Tipo de medición**: Punto medio a punto medio.
   - **Límites de aceptación**: 97mm ≤ valor ≤ 102mm. Estas tolerancias permiten admitir pequeñas desviaciones sin rechazar piezas que son funcionalmente válidas.

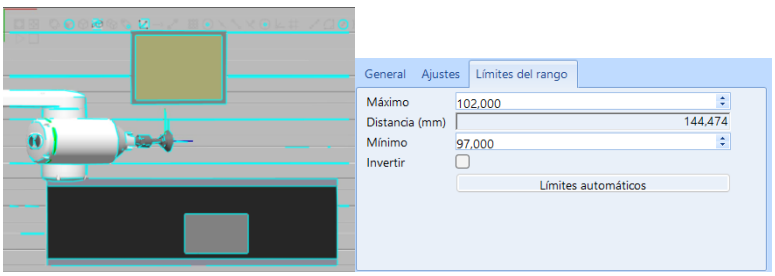


Figura 190. Selección características a medir y límites de aceptación

3. **Herramientas auxiliares (Edges):** Cada herramienta de distancia genera automáticamente dos herramientas **Edge**, que detectan los bordes relevantes de la pieza.

- **Área de búsqueda:** Se ajustan para que abarque toda la posible zona de aparición de los bordes, teniendo en cuenta la variabilidad de dimensiones.
- **Transición de borde:** Como las piezas son grises y se transportan en una cinta transportadora negra, se ha configurado la transición de borde como oscuro a claro. Esto mejora la detección de los bordes de la pieza.

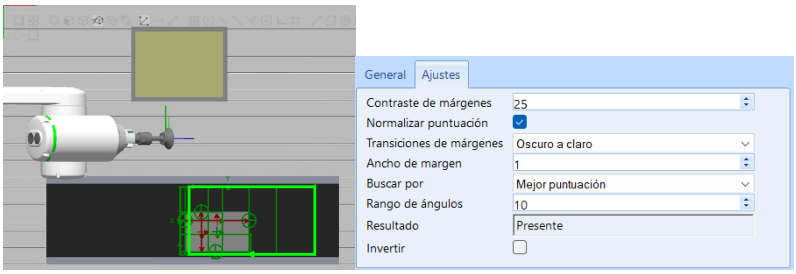


Figura 191. Configuración de la herramienta de inspección Edges

4. **Herramienta de ubicación (Blobs de color (1-10)):** Debido a la variabilidad de las dimensiones de las piezas, se necesita emplear una herramienta de ubicación robusta para localizar el centroide de las piezas defectuosas.
   En primer lugar, se debe situar el área dónde la característica del blob puede estar situada durante la inspección.

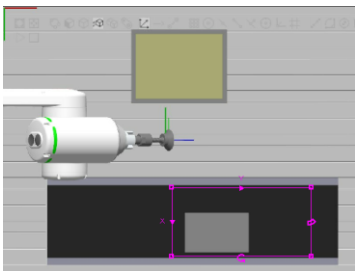


Figura 192. Configuración de la herramienta de ubicación Blobs de color (1-10)

A continuación, se lleva a cabo el proceso narrado en la práctica anterior para la herramienta de ubicación llamada **Blobs de color (1-10).** Primero se crea un nuevo modelo de colores y se entrena el color gris de las piezas seleccionando la opción “Sumar nuevo color”.

Una vez configurado, la herramienta detecta el blob correspondiente y calcula su centroide, el cual se usará como referencia para el movimiento del robot.

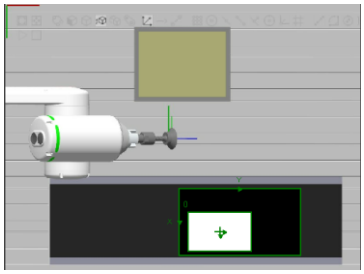


Figura 193. Ubicación del centroide de las piezas

5. **Salida a RAPID:** Se ha configurado una salida de cámara llamada “Distancias”, en la que se vinculan los siguientes parámetros para su envío al programa RAPID:
    - Posición X/Y y rotación Z del centro de la pieza
    - Medidas X e Y como val1 y val2.
    - Criterio de validación para cada medida como string1 y string2, que indican "1" si la medida está dentro del rango aceptado (pieza correcta) o "0" si está fuera (pieza defectuosa).

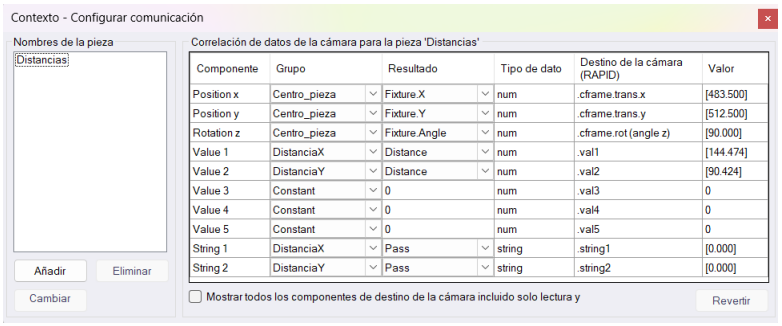


Figura 194. Salidas a RAPID del trabajo de visión job_practica3.job

Finalmente, el trabajo de visión fue guardado bajo el nombre “job_practica3.job” en la carpeta “Sensores In-Sight”, siguiendo la metodología descrita en el apartado [5.2.7 Lógica de funcionamiento de un sistema de visión artificial].

Implementación del código RAPID

Declaración de herramientas, objetos de trabajo y posiciones

Al inicio del módulo se definen los elementos necesarios para llevar a cabo la ejecución del programa. Se declara la herramienta *TCP_ToolVentosa*, que representa la ventosa utilizada por el robot para manipular las piezas. A continuación, se configuran dos objetos de trabajo: *cam_WObj*, que hace referencia al área de trabajo de la cámara situada a la altura de las piezas sobre la cinta transportadora, y *Caja_defectos*, correspondiente al contenedor donde se depositan las piezas que no cumplen con las medidas establecidas.

También se definen tres posiciones: “Home”, que es la posición inicial y final del ciclo; “PreDejar_defectos”, que corresponde a una posición previa de descarga sobre la caja de defectos; y “Dejar_defectos”, que indica la posición final donde se depositan las piezas defectuosas.

Por último, se declara el nombre del trabajo de visión previamente entrenado, una variable de tipo *cameratarget* que almacena los resultados obtenidos por el sistema de visión, y un contador para llevar el registro del número de piezas defectuosas detectadas durante la ejecución del programa.

```
LOCAL PERS tooldata TCP_ToolVentosa:=[TRUE,[[0,0,121.5],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];

LOCAL PERS wobjdata Caja_defectos:=[FALSE,TRUE,"",[[461.769,255.028,110.438],[0,1,0,0]],[[0,0,0],[1,0,0,0]]];
LOCAL PERS wobjdata cam_WObj:=[FALSE,TRUE,"",[[170,304.269,181.785],[0.00377863,0.999993,0,0]],[[474.008,524.504,0],[0.707107,0,0,0.707107]]];       !Altura cinta

LOCAL CONST robtarget Home:=[[540.684087634,0,773.032029102],[0.500000007,0,0.8660254,0],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget PreDejar_defectos:=[[110,85,-150],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Dejar_defectos:=[[110,85,0],[0.707106781,0,0,0.707106781],[0,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget myrobtarget:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

LOCAL CONST string myjob := "job_practica3.job";

LOCAL VAR cameratarget mycameratarget;

LOCAL VAR num contador_defectuosas:=0;
```

Figura 195. Declaraciones iniciales de la práctica 3

Procedimiento "MoveToDetectedObject()"

El procedimiento "MoveToDetectedObject()" se encarga de capturar la imagen de una pieza, obtener los resultados de inspección mediante un sistema de visión artificial, notificar las medidas detectadas y ejecutar una acción en función de estos datos.

En primer lugar, se carga el trabajo de visión previamente entrenado y se configura el sistema en modo de ejecución. Luego, se realiza la carga manual de la imagen, la cual es capturada por el componente inteligente diseñado para este propósito. Esta acción se confirma mediante la activación de la entrada digital DI2.

Una vez capturada la imagen, los resultados del trabajo de visión se almacenan en la variable *mycameratarget*. Además, las medidas detectadas (*mycameratarget.val1* y *mycameratarget.val2*) se notifican al usuario mediante mensajes en el FlexPendant.

La decisión que toma el sistema depende del contenido de las variables string1 y string2, que indican si las medidas están dentro del rango permitido. Si uno o ambos valores son "0", lo que implica que la pieza no cumple con los criterios establecidos, el robot se desplaza hacia ella, la succiona y la deposita en el contenedor de piezas defectuosas. En caso contrario, si ambos valores son válidos, se reactiva la cinta transportadora.

```
LOCAL PROC MoveToDetectedObject()

    CamSetProgramMode cam;
    CamLoadJob cam, myjob;      !Carga trabajo ya entrenado
    CamSetRunMode cam;

    WaitDI Scalable_IO_0_DI2,1;
    CamReqImage cam;

    CamGetResult cam, mycameratarget;

    TPWrite "Estas son las medidas detectadas de la pieza inspeccionada";
    TPWrite "Medidas X: "\Num:=mycameratarget.val1;
    TPWrite "Medidas Y: "\Num:=mycameratarget.val2;

    IF mycameratarget.string1="0" OR mycameratarget.string2= "0" THEN
        cam_WObj.oframe := mycameratarget.cframe;

        !Efecto succión al llegar a posición
        MoveJDo myrobtarget,v100,fine,TCP_ToolVentosa\WObj:=cam_WObj, Scalable_IO_0_DO15,1;

        Pieza_defectuosa;
    ELSE
        SetDO Scalable_IO_0_DO9,1;
        waittime 3;
    ENDIF

ENDPROC
```

Figura 196. Procedimiento "MoveToDetectedObject()" de la práctica 3

Procedimiento "GoHome()"

Este procedimiento simplemente retorna el robot a la posición inicial "Home" una vez finalizada la operación de clasificación, preparando el sistema para un nuevo ciclo.

Procedimiento "Pieza_defectuosa()"

Cuando se detecta una pieza fuera de los rangos definidos, se ejecuta este procedimiento para depositarla en el contenedor correspondiente. El robot se desplaza primero a una posición situada encima de la caja de defectos y luego desciende hasta la posición final. Una vez allí, se desactiva la ventosa para soltar la pieza y se incrementa el contador de piezas defectuosas.

```
LOCAL PROC Pieza_defectuosa()

    TPWrite "Pieza defectuosa detectada";

    MoveJ PreDejar_defectos,v100,z100,TCP_ToolVentosa\WObj:=Caja_defectos;

    !Romper vacío ventosa al llegar a destino
    MoveLDO Dejar_defectos,v50,fine,TCP_ToolVentosa\WObj:=Caja_defectos, Scalable_IO_0_DO15,0;

    contador_defectuosas:=contador_defectuosas+1;

ENDPROC
```

Figura 197. Procedimiento "Pieza_defectuosa()" de la práctica 3

Procedimiento principal "main_Práctica3_DefectosMedidas()"

Este es el bucle principal del programa, encargado de ejecutar de forma continua el proceso de inspección y descarte de piezas. Al inicio de cada ciclo, se muestra en el FlexPendant el número acumulado de piezas defectuosas detectadas. Después, se activa la cinta transportadora (DO9) y se espera la señal del sensor (DI9) que indica que una nueva pieza ha llegado a la posición de inspección. En ese momento, se detiene la cinta, se activa la captura de imagen (DO14) y se ejecuta el procedimiento "MoveToDetectedObject()", que se encarga de evaluar si la pieza cumple con los criterios de dimensiones.

Una vez completada la inspección y manipulación, se llama al procedimiento "GoHome()" y se reinician todas las señales digitales empleadas, dejando el sistema listo para comenzar un nuevo ciclo.

```
PROC main_Práctica3_DefectosMedidas()

    WHILE TRUE DO

        TPWrite " Piezas defectuosas: "\Num:=contador_defectuosas;

        SetDO Scalable_IO_0_DO9,1;  !Inicio conveyor
        WaitDI Scalable_IO_0_DI9,1; !Espera pieza posicion foto
        SetDO Scalable_IO_0_DO9,0;  !Paro conveyor

        SetDO Scalable_IO_0_DO14,1;  !Capturar foto
        SetDO Scalable_IO_0_DO14,0;

        MoveToDetectedObject;
        GoHome;

        !Reset final de las señales
        Reset Scalable_IO_0_DO9;
        Reset Scalable_IO_0_DO14;
        Reset Scalable_IO_0_DO15;

    ENDWHILE

ENDPROC
```

Figura 198. Procedimiento principal "main_Práctica3_DefectosMedidas()"

Resultados

Los resultados obtenidos en esta práctica confirman la fiabilidad del sistema desarrollado para la detección y clasificación de piezas defectuosas en función de sus dimensiones. A través del panel de resultados de la interfaz de visión de *RobotStudio*, se pudo verificar el correcto funcionamiento de la herramienta de medición.

En dicho panel, se visualiza un conjunto de indicadores que permiten validar el proceso: las herramientas identificadas correctamente aparecen con un círculo verde (*Pass*), mientras que aquellas que fallan muestran un cuadrado rojo (*Fail*). En el caso específico de las herramientas de distancia, se presenta el valor medido en milímetros. Si la medida se encuentra dentro del rango establecido (97–102 mm), se muestra únicamente el valor numérico; en caso contrario, aparece una cruz roja junto al resultado indicando que está fuera de tolerancia.

Respecto a las herramientas de bordes (**Edges**), el panel únicamente informa si los márgenes definidos están presentes o no en la imagen capturada. Por último, en la herramienta de ubicación **Blobs de color (1–10),** se visualizan las coordenadas del centroide detectado y su orientación en el eje Z.

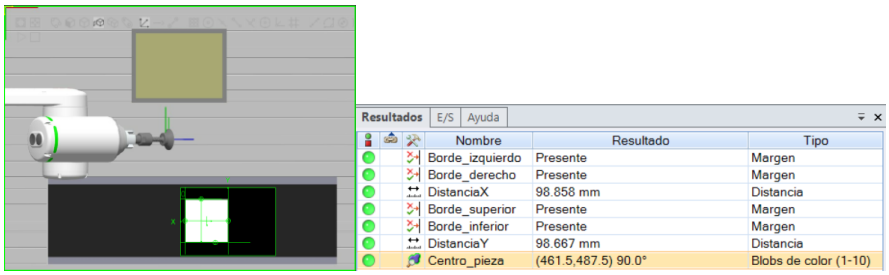


Figura 199. Resultados trabajo de visión con una pieza correcta

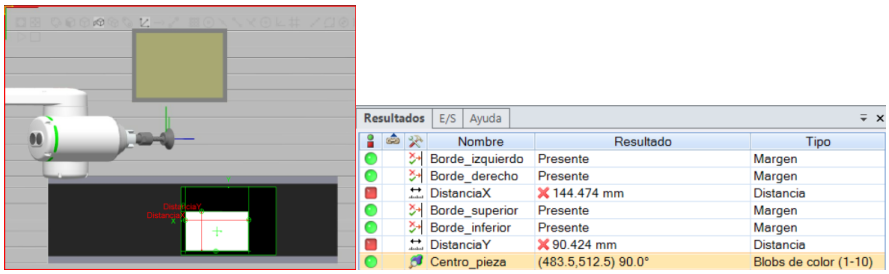


Figura 200. Resultados trabajo de visión con una pieza defectuosa

Adicionalmente, la información relativa a la clasificación de las piezas se muestra en tiempo real en la FlexPendant. En ella, se actualiza de forma continua el número total de piezas defectuosas detectadas, así como mensajes que indican cuándo se ha descartado una pieza por no cumplir con los requisitos dimensionales. Cuando esto ocurre, el robot ejecuta de forma precisa la secuencia de agarre, traslado y depósito en el contenedor correspondiente. En caso de que la pieza sea válida, el sistema reactiva la cinta transportadora, simulando su paso hacia una fase posterior del proceso.

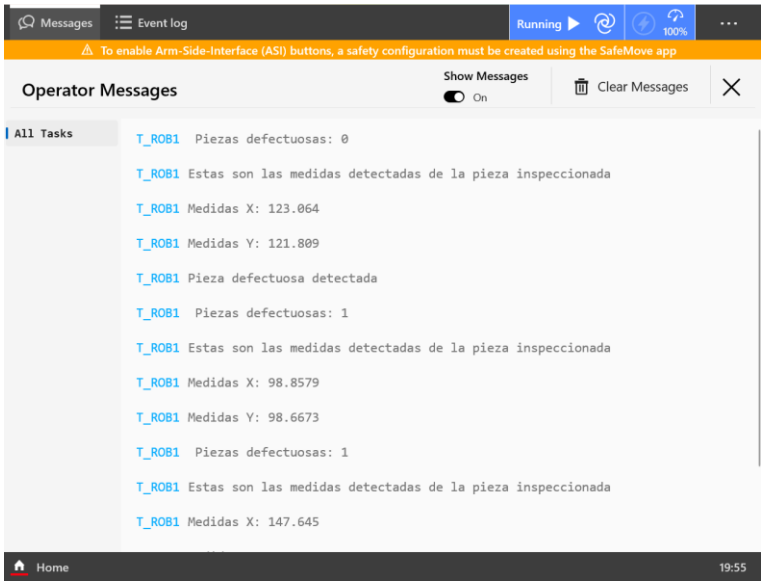


Figura 201. Mensajes en la FlexPendant de las medidas detectadas y el recuento de piezas defectuosas

Tras múltiples ciclos de prueba con piezas generadas aleatoriamente por el componente inteligente, se ha logrado una clasificación robusta y precisa. El sistema fue capaz de diferenciar entre piezas válidas y defectuosas en todos los casos, mostrando una clara separación de resultados en función de las medidas tomadas.

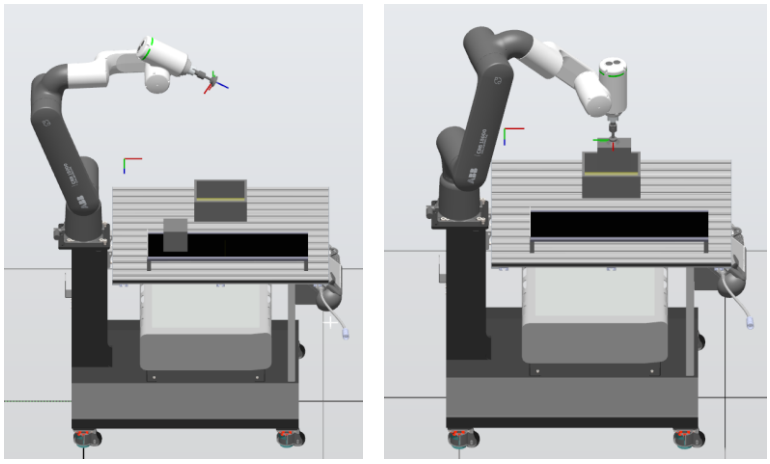


Figura 202. Proceso de clasificación de figuras de la práctica 3

5.3.5. Práctica 4: Torre Hanoi

Esta práctica consiste en la construcción automática de una Torre de Hanoi utilizando visión artificial y un robot colaborativo. Sobre la superficie de trabajo se disponen cinco cilindros de distintos diámetros, colocados inicialmente de forma desordenada sobre un tapete de color verde que ayuda a mejorar el contraste visual y la identificación de los objetos.

El sistema de visión se encarga de analizar las piezas y determinar su tamaño, permitiendo al robot reconocer cada cilindro e identificar su orden correcto. Para ello, se ha empleado una herramienta de medición que permite detectar el diámetro de

cada cilindro con precisión. Una vez evaluadas las piezas, el robot las recoge y las coloca de mayor a menor, formando la torre sobre una zona designada del entorno de trabajo.

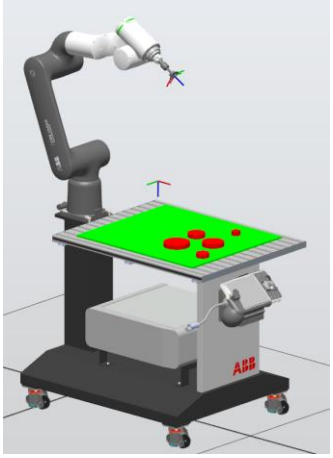


Figura 203. Estación de la práctica 4 Torre de Hanoi

Modelado de componentes

Modelado de las piezas

Para representar las piezas del juego de la Torre de Hanoi, se han modelado cinco cilindros de igual altura, pero con diámetros diferentes, lo que permite su diferenciación mediante visión artificial. Todos los cilindros han sido creados de la misma forma, utilizando la herramienta “Cilindro”, ubicada en la pestaña “Modelado”, dentro del apartado “Crear”, seleccionando la opción correspondiente en el desplegable de “Sólido”.

Cada uno de los cilindros se ha colocado directamente sobre la mesa de trabajo en posiciones aleatorias, simulando una disposición desordenada de las piezas en el entorno de trabajo. A todos se les asignó el mismo color rojo, con el objetivo de generar un contraste claro frente al fondo verde del tapete. Los diámetros de los cilindros varían entre 120 mm y 40 mm, con una diferencia constante de 20 mm entre una pieza y otra.

Figura 204. Modelado de las piezas cilíndricas de la práctica 4

Modelado del tapete

Con el fin de mejorar el contraste visual entre las piezas y la mesa de trabajo, se ha añadido un tapete rectangular sobre el cual se ubican los cilindros. Este tapete se ha creado utilizando la herramienta “Tetraedro”, también ubicada en el menú “Sólido”, definiendo unas dimensiones de 600x400x20mm. Se le ha aplicado un color verde para resaltar las piezas colocadas sobre él.

Aunque el tapete no interviene en la manipulación, su función como fondo visual es fundamental para optimizar la detección de formas por parte del sistema de visión.

Figura 205. Modelado del tapete

Nota: Se ha desactivado la detección del tapete por parte del sensor, ya que su presencia no debe influir en el proceso de manipulación de las piezas.

Tarea de visión

El objetivo de la tarea de visión en esta práctica es identificar correctamente los cilindros dispuestos sobre el tapete y obtener el valor de su diámetro para clasificarlos y organizarlos de mayor a menor.

Para ello, se ha utilizado la herramienta de inspección **Diámetro**, que permite identificar automáticamente características circulares en la imagen y medir su diámetro en milímetros, gracias a la calibración previa de la cámara mediante la opción “Escala X/Y” 1:1.

Una vez capturada la imagen del entorno de trabajo, se selecciona la herramienta Diámetro del menú de herramientas de inspección. Al activarla, se destacan en azul todos los contornos circulares detectados automáticamente por el sistema. El usuario selecciona manualmente el círculo deseado, el cual pasa a marcarse en color morado como región activa de medición.

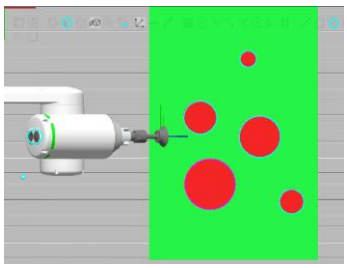


Figura 206. Selección manual del círculo deseado

Tras definir el contorno, se ajustan los parámetros necesarios para garantizar una medición robusta, como el contraste, el ancho o las transiciones de los márgenes. En esta práctica, se ha optado por establecer un margen de tolerancia de ±5mm respecto al valor esperado del diámetro. No obstante, el sistema también permite definir estos límites automáticamente en función de los datos detectados. Visualmente, en la imagen aparecen tres círculos: el que representa el borde detectado y los dos que corresponden a los límites del rango establecido.

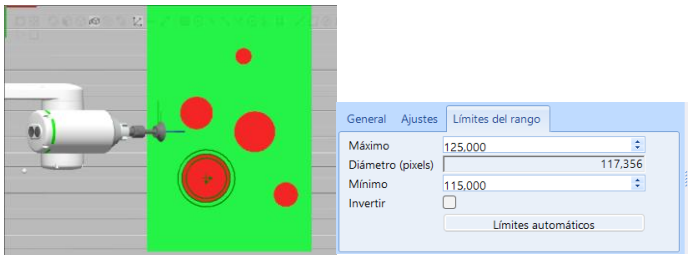


Figura 207. Configuración de la herramienta de medición llamada Diámetro

Este proceso se debe realizar para los cinco cilindros que componen la Torre de Hanoi. Cada una de estas herramientas devuelve dos resultados principales: el valor del diámetro medido y una indicacion de validación en función de si la medición se encuentra dentro de los límites establecidos.

Finalmente, se han configurado las cinco salidas correspondientes a cada uno de los cilindros detectados. En cada una se ha especificado el envío a RAPID de la posicion X e Y, el valor del diámetro (val1) y el estado de validación (string1), que indica si la medición es considerada aceptable.

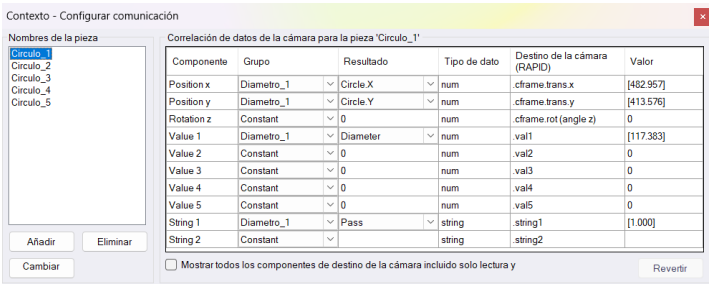


Figura 208. Salidas a RAPID del trabajo de visión job_practica4.job

Finalmente, el trabajo de visión fue guardado bajo el nombre “job_practica4.job” en la carpeta “Sensores In-Sight”, siguiendo la metodología descrita en el apartado [5.2.7 Lógica de funcionamiento de un sistema de visión artificial].

Implementación del código RAPID

Declaración de herramientas, objetos de trabajo y posiciones

Al comienzo del módulo se define la herramienta *TCP_ToolVentosa*, utilizada para manipular las piezas que componen la Torre de Hanoi. Además, se ha configurado el objeto de trabajo *cam_WObj_P4*, correspondiente a la zona de visión sobre la mesa a la altura de las piezas, así como varias posiciones: la posición de inicio y final del ciclo

(*Home*), una posición previa a la descarga (*PreDejar_Torre*) y la posición de descarga (*Dejar_torre*) que representa el punto dónde se apilan las distintas piezas.

También se declara una constante string que contiene el nombre del trabajo de visión previamente entrenado, junto con dos variables del tipo *cameratarget* para almacenar los resultados enviados por el sistema de visión. Por último, se utiliza una variable auxiliar altura para modificar dinámicamente la altura a la que se debe dejar cada cilindro, asegurando así que la torre se construya correctamente.

```
LOCAL PERS tooldata TCP_ToolVentosa:=[TRUE,[[0,0,121.5],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];

LOCAL PERS wobjdata cam_Wobj_P4:=[FALSE,TRUE,"",[[170,304.269,35],[0.00377863,0.999993,0,0]],[[573.025,121.344,0],[1,0,0,0]]];

LOCAL CONST robtarget Home:=[[540.684087634,0,773.032029102],[0.500000007,0,0.8660254,0],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget myrobtarget:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
LOCAL CONST robtarget PreDejar_torre:=[[398.144,1.591,181.000488061],[0,-0.707106781,0.707106781,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Dejar_torre:=[[398.144,1.591,43.000488061],[0,-0.707106781,0.707106781,0],[0,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

LOCAL CONST string myjob := "job_practica4.job";

LOCAL VAR cameratarget mycameratarget;
LOCAL VAR cameratarget aux;

LOCAL VAR num altura:=0;
```

Figura 209. Declaraciones iniciales de la práctica 4

Procedimiento “MoveToDetectedObject()”

Este procedimiento implementa la lógica principal de visión artificial y selección de piezas, determinando en cada iteración cuál es el cilindro de mayor diámetro disponible.

En primer lugar, se configura la cámara estableciendo el modo programación, se carga el trabajo de visión correspondiente a esta práctica y se activa el modo ejecución. Una vez completada la carga manual de la imagen mediante el componente inteligente, se inicia un bucle de cinco iteraciones en el que se realizan lecturas consecutivas de los resultados generados por el sistema de visión.

En cada ciclo del bucle, se comprueba si el cilindro detectado es válido (string1 = "1") y si su diámetro (val1) es mayor que el almacenado en la variable auxiliar “aux”. Si ambas condiciones se cumplen, se actualiza “aux” con los datos del cilindro de mayor tamaño detectado hasta ese momento.

Al finalizar el bucle, se transfiere el contenido de “aux” a la variable *mycameratarget*, que contiene la información definitiva del cilindro seleccionado para ser manipulado. Es fundamental reiniciar aux.val1 a cero tras cada iteración, para evitar que el valor anterior interfiera en la comparación del siguiente ciclo.

A continuación, se actualiza el objeto de trabajo *cam_Wobj_P5* con el marco de coordenadas del cilindro identificado, y el robot se desplaza hasta su posición para recogerlo mediante la herramienta de succión. Posteriormente, el robot traslada la pieza a la zona de apilado, donde la deposita.

Para garantizar un apilamiento correcto, la altura de colocación se ajusta dinámicamente mediante la variable z, que se incrementa tras cada colocación, permitiendo que cada nuevo cilindro se sitúe correctamente sobre el anterior, construyendo así la Torre de Hanoi.

```
LOCAL PROC MoveToDetectedObject()

    CamSetProgramMode cam;
    CamLoadJob cam, myjob;      !Carga trabajo ya entrenado
    CamSetRunMode cam;

    WaitDI Scalable_IO_0_DI2,1;
    CamReqImage cam;

    CamGetResult cam, mycameratarget;

    FOR i FROM 0 TO 4 DO

        CamReqImage cam;
        CamGetResult cam, mycameratarget;

        IF mycameratarget.string1="1" AND mycameratarget.val1 > aux.val1 THEN
            aux := mycameratarget;
        ELSEIF mycameratarget.string1="0" THEN
            Waittime 1;
        ENDIF

    ENDFOR

    mycameratarget:=aux;
    aux.val1:=0;

    cam_Wobj_P4.oframe := mycameratarget.cframe;

    !Efecto succión al llegar a posición
    MoveJDo myrobtarget,v100,fine,TCP_ToolVentosa\WObj:=cam_Wobj_P4, Scalable_IO_0_DO15,1;
    MoveJ PreDejar_torre,v100,z100,TCP_ToolVentosa\WObj:=wobj0;
    MoveLDo Offs (Dejar_torre,0,0,altura),v50,fine,TCP_ToolVentosa\WObj:=wobj0, Scalable_IO_0_DO15,0;

    altura:=altura+22;

ENDPROC
```

Figura 210. Procedimiento "MoveToDetectedObject()" de la práctica 4

Procedimiento "GoHome()"

Este procedimiento simplemente retorna el robot a la posición inicial "Home" una vez finalizada la operación de clasificación, preparando el sistema para un nuevo ciclo.

Procedimiento principal "main_Práctica4_TorreHanoi()"

Este es el procedimiento principal del programa, que coordina todo el proceso de construcción de la Torre de Hanoi. Consiste en un bucle FOR que se repite cinco veces, una para cada pieza situada inicialmente sobre la mesa.

En cada iteración, se activa la señal digital que desencadena la captura de imagen desde el componente inteligente "ImageCapture". A continuación, se ejecuta el procedimiento "MoveToDetectedObject()", encargado de identificar el cilindro de mayor diámetro aún disponible, recogerlo y colocarlo en la torre.

Finalmente se regresa a la posición de reposo (Home) y se reinician todas las señales digitales empleadas en el ciclo. Este proceso se repite hasta que todas las piezas han sido apiladas, completando así la Torre de Hanoi de forma ordenada y precisa.

```
PROC main_Práctica4_TorreHanoi()

    FOR i FROM 0 TO 4 DO

        SetDO Scalable_IO_0_DO14,1;  !Capturar foto
        SetDO Scalable_IO_0_DO14,0;

        MoveToDetectedObject;
        GoHome;

        !Reset final de las señales
        Reset Scalable_IO_0_DO14;
        Reset Scalable_IO_0_DO15;

    ENDFOR

ENDPROC
```

Figura 211. Procedimiento principal "main_Práctica4_TorreHanoi()"

Resultados

Los resultados obtenidos en esta práctica demuestran el correcto funcionamiento del sistema de visión artificial y su integración con el robot para la clasificación y apilado de piezas en función de su tamaño. Gracias a la herramienta de inspección **Diámetro**, el sistema fue capaz de identificar con precisión las dimensiones en milímetros de los cilindros colocados sobre la mesa, permitiendo su ordenación descendente para formar la Torre de Hanoi.

A través del panel de resultados del entorno de visión de *RobotStudio*, se visualizan de manera clara los datos obtenidos por la herramienta: un círculo verde indica que se ha detectado un cilindro correctamente (*Pass*), mostrando su diámetro en milímetros; por el contrario, un cuadrado rojo (*Fail*) con una cruz señala la no detección o una lectura inválida. Esto permite verificar rápidamente si la pieza cumple con los criterios establecidos.

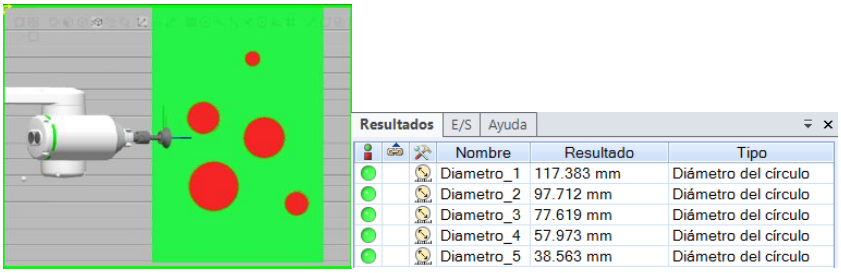


Figura 212. Resultados trabajo de visión con todas las piezas sobre la mesa

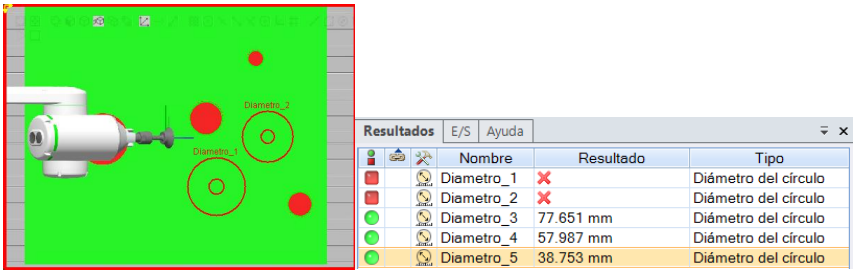


Figura 213. Resultados trabajo de visión cuando falta alguna de las piezas sobre la mesa

Durante las cinco iteraciones del ciclo, el sistema seleccionó correctamente el cilindro de mayor diámetro disponible en cada momento, descartando aquellos ya apilados. El robot realizó exitosamente la secuencia de agarre, traslado y apilamiento, asegurando un posicionamiento preciso sin colisiones entre piezas.

Además, se ajustó la altura de apilado mediante la variable altura, que se incrementó en cada iteración para mantener una separación adecuada entre los cilindros. Esto garantizó una construcción estable de la torre, con un ajuste preciso entre las piezas evitando desplazamientos o solapamientos.

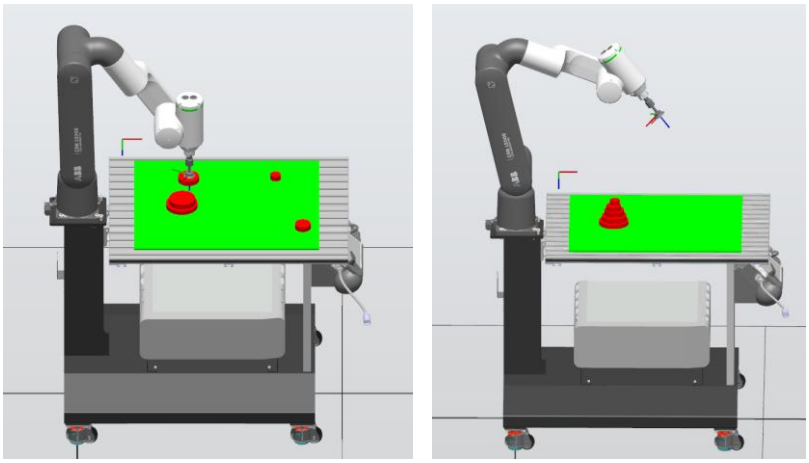


Figura 214. Proceso de montaje de la torre de Hanoi de la práctica 4

5.3.6. Práctica 5: Lectura QR

Esta práctica tiene como objetivo identificar y clasificar automáticamente piezas en función de la información contenida en sus códigos QR, utilizando visión artificial y un sistema robótico. Los cubos se encuentran distribuidos sobre la mesa, y cada uno lleva un código QR en su cara superior con el texto "Estante X. Posicion Y", que indica su ubicación de destino dentro de una estantería.

Una vez capturada la imagen, el sistema de visión emplea la herramienta de inspección **Leer código 2D (1–20)** para interpretar el contenido del QR. El robot recibe esta información, selecciona el cubo correspondiente y lo deposita en la posición indicada, simulando un sencillo proceso de gestión de almacén automatizado.

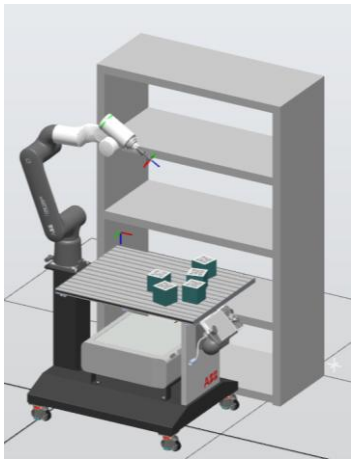


Figura 215. Estación de la práctica 5 Lectura de QR

Modelado de componentes

Modelado de las piezas con código QR

Para esta práctica se ha reutilizado el modelo de cubo ya empleado en la práctica 1. Dicho modelo se ha copiado varias veces y distribuido sobre la mesa dentro del grupo

de componentes correspondiente a esta actividad. A cada uno de estos cubos se le ha asignado un código QR único en su cara superior, que indica su destino dentro de la estantería.

Los códigos QR se han generado previamente utilizando la plataforma web Generador de códigos QR, introduciendo el texto correspondiente al destino de cada pieza siguiendo el formato "Estante X. Posicion Y". Una vez creados, los códigos fueron descargados como imágenes e importados en el entorno de modelado de *RobotStudio*.

Figura 216. Generación de los códigos QR

Para aplicar cada QR a su cubo, se accede al menú contextual del componente (clic derecho) y se selecciona "Apariencia de gráficos". Desde la ventana emergente, se define tanto el color base como la textura del cubo, seleccionando la cara superior como superficie visible y cargando allí la imagen correspondiente mediante la opción "Importar" en el apartado de "Textura de base".

Este proceso asegura que cada código QR sea correctamente orientado y legible por el sistema de visión artificial durante la fase de lectura e identificación.

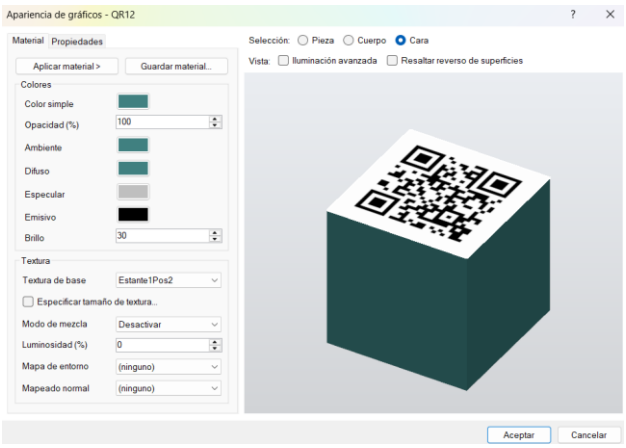


Figura 217. Incorporación del código QR a cada pieza

Modelado de la estantería

La estantería ha sido construida empleando sólidos básicos. En primer lugar, se ha creado un tetraedro de grandes dimensiones (1200x500x1860mm), que actúa como el cuerpo exterior de la estantería. A continuación, se han modelado varios tetraedros más pequeños (600x1100x400mm) que representan los huecos entre baldas.

Estos sólidos se han alineado y posicionado en el interior del bloque principal, manteniendo una separación uniforme para simular distintos niveles de almacenaje. En total se han diseñado cuatro baldas, aunque en esta práctica únicamente se utilizarán los dos superiores. Una vez colocados los huecos, se ha aplicado la operación "Restar" del entorno de "Modelado" para eliminar los volúmenes internos y obtener así los espacios necesarios donde se depositarán los cubos.

La estantería resultante simula un sistema de almacenamiento realista, con compartimentos claramente definidos y accesibles. Se ha colocado estratégicamente dentro del espacio de trabajo del robot, asegurando una buena accesibilidad sin riesgo de alcanzar zonas próximas a la singularidad en ninguna de las posiciones de colocación.

Figura 218. Modelado de la estantería: cuerpo exterior y huecos

Tarea de visión

El objetivo de la tarea de visión en esta práctica es identificar el destino de cada pieza a través de la lectura de los códigos QR situados en la cara superior. Para ello, se ha empleado la herramienta de inspección **Leer códigos 2D (1-20)** del entorno de visión artificial de *RobotStudio*.

En primer lugar, se ha calibrado la cámara utilizando la opción "Escala X/Y" 1:1 en milímetros, lo que permite que la posición y orientación del código QR detectado se correspondan con las coordenadas reales del entorno de trabajo.

La configuración de esta herramienta es sencilla: basta con colocar el área de búsqueda (recuadro morado) sobre la zona donde aparecerán los códigos QR de las piezas.

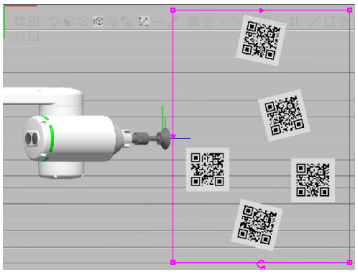


Figura 219. Selección área de búsqueda de los códigos QR

A continuación, se selecciona el grupo de simbología “QR Code”, se establece el número máximo de resultados (en este caso, 5) y se pulsa el botón “Capacitar”. Tras este proceso, se habilita un panel en el que se muestran las secuencias de caracteres detectadas en cada código QR, junto con su posición y orientación.

Índice	Secuencia de caracteres	Simbología	X	Y	Ángulo	Grado
0	Estante 1. Posicion 2	QR Code	592,703	223,420	-104,185	4,000
1	Estante 2. Posicion 1	QR Code	426,450	335,437	-90,000	0,000
2	Estante 1. Posicion 3	QR Code	665,105	359,095	-89,983	0,000
3	Estante 2. Posicion 2	QR Code	548,061	453,412	-76,373	4,000
4	Estante 1. Posicion 1	QR Code	556,237	36,488	-78,077	0,000
5	-	-	-	-	-	-

Figura 220. Lecturas realizadas por la herramienta Leer códigos 2D (1-20) junto a sus posiciones y orientación

Esta herramienta también permite otras opciones como el entrenamiento de perspectiva o la elección del modo de ordenación de los resultados: de arriba abajo, de izquierda a derecha, alfabéticamente o por la longitud de las cadenas de caracteres. Existen dos modos principales: “Leer”, que es el utilizado en esta práctica, ya que se requiere interpretar directamente el contenido de cada código; y “Cadena coincidente”, útil en los casos en los que se desea verificar si un código QR corresponde con una cadena especifica.

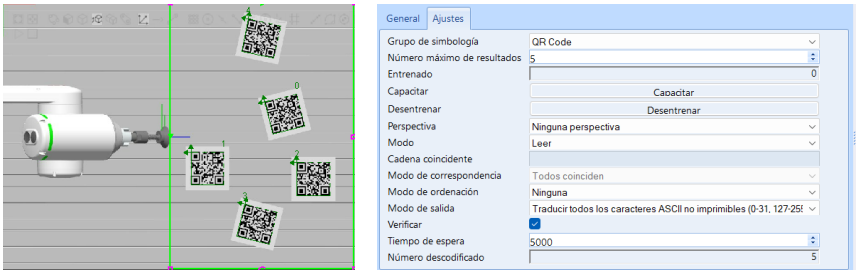


Figura 221. Ajustes realizados a la herramienta Leer códigos 2D (1-20)

Finalmente, se ha creado una salida de comunicación con RAPID especificando los datos que se desean enviar: posición X, posición Y, rotación Z del código QR (esta herramienta proporciona las coordenadas de uno de los vértices del código, no del centro), y string1, que contiene la secuencia de caracteres leída.

A diferencia de otras prácticas, donde se generaban múltiples salidas para cada pieza, en este caso la propia herramienta agrupa todos los resultados en una única estructura, lo que permite gestionar varias lecturas simultáneamente. Como se puede observar en la figura 222, los datos enviados aparecen entre corchetes, reflejando que se trata de una lista de elementos, uno por cada código QR detectado.

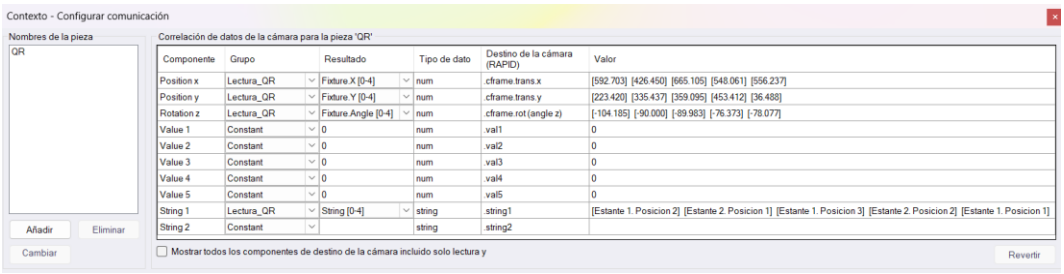


Figura 222. Salidas a RAPID del trabajo de visión job_practica5.job

El trabajo de visión fue guardado bajo el nombre "job_practica5.job" en la carpeta "Sensores In-Sight", siguiendo la metodología descrita en el apartado [5.2.7 Lógica de funcionamiento de un sistema de visión artificial].

Implementación del código RAPID

Declaración de herramientas, objetos de trabajo y posiciones

Al inicio del módulo se define la herramienta de succión *TCP_ToolVentosa*, utilizada para recoger los cubos y colocarlos en sus posiciones asignadas dentro de la estantería. A continuación, se configuran tres objetos de trabajo: *cam_WObj_P5*, que representa el área donde se encuentran los cubos con códigos QR, y se posiciona a la altura de succión de las piezas; así como *Estante1* y *Estante2*, correspondientes a las dos primeras baldas de la estantería donde se depositan las piezas manipuladas.

También se declaran varias posiciones fijas en el entorno: la posición de referencia "Home", y seis posiciones de destino (tres para cada estante), cada una con su correspondiente posición previa para permitir movimientos más seguros y fluidos.

Por último, se define el nombre del trabajo de visión previamente entrenado y una variable del tipo *cameratarget*, que almacena los resultados obtenidos tras cada lectura efectuada por el sistema de visión.

```
LOCAL PERS tooldata TCP_ToolVentosa:=[TRUE,[[0,0,129.5],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];

LOCAL PERS wobjdata cam_Wobj_P5:=[FALSE,TRUE,"",[[170,304.269,120.896],[0.00377863,0.999993,0,0]],[[556.237,36.4884,0],[0.776725,0,0,-0.62984]]];
LOCAL PERS wobjdata Estante1:=[FALSE,TRUE,"",[[-55.024,500,580.456],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
LOCAL PERS wobjdata Estante2:=[FALSE,TRUE,"",[[-55.024,500,129.953],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

LOCAL CONST robtarget myrobtarget:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
LOCAL CONST robtarget Home:=[[540.684087634,0,773.032029102],[0.500000007,0,0.8660254,0],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante2Pos1:=[[275,50,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante2Pos1Pre:=[[275,-170,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante2Pos2:=[[550,50,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante2Pos2Pre:=[[550,-170,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante2Pos3:=[[825,50,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante2Pos3Pre:=[[825,-170,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante1Pos1:=[[275,50,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante1Pos1Pre:=[[275,-170,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante1Pos2:=[[550,50,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante1Pos2Pre:=[[550,-170,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante1Pos3:=[[825,50,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Estante1Pos3Pre:=[[825,-170,90],[0.707106781,-0.707106781,0,0],[0,1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

LOCAL CONST string myjob := "job_practica5.job";

LOCAL VAR cameratarget mycameratarget;
```

Figura 223. Declaraciones iniciales de la práctica 5

Procedimiento "MoveToDetectedObject()"

Este procedimiento se encarga de gestionar la lectura de los códigos QR, determinar la ubicación de destino y realizar la manipulación de recogida de las piezas. En primer lugar, se activa el modo de programación y se carga el trabajo de visión previamente entrenado. Luego, se cambia al modo de ejecución y se lleva a cabo la carga manual de imágenes a través del componente inteligente "ImageCapture".

Una vez confirmada la carga, se consultan los resultados del trabajo de visión. Con esta información, se actualiza el marco de referencia *cam_WObj_P5* y el robot se desplaza hacia la posición del cubo detectado. Se realiza la succión de la pieza correspondiente y una retirada controlada antes de invocar el procedimiento que gestiona el depósito de piezas en la estantería.

```
LOCAL PROC MoveToDetectedObject()

    CamSetProgramMode cam;
    CamLoadJob cam, myjob;      !Carga trabajo ya entrenado
    CamSetRunMode cam;

    WaitDI Scalable_IO_0_DI2,1;
    CamReqImage cam;
    CamGetResult cam, mycameratarget;

    cam_WObj_P5.oframe := mycameratarget.cframe;

    !Efecto succión al llegar a posición
    MoveJDo myrobtarget,v100,fine,TCP_ToolVentosa\WObj:=cam_WObj_P5, Scalable_IO_0_DO15,1;
    MoveL Offs (myrobtarget,0,0,-150),v50,fine,TCP_ToolVentosa\WObj:=cam_Wobj_P5;

    Seleccion_Ubicacion;

ENDPROC
```

Figura 224. Procedimiento "MoveToDetectedObject()" de la práctica 5

Procedimiento "GoHome()"

Este procedimiento simplemente retorna el robot a la posición inicial "Home" una vez finalizada la operación de clasificación, preparando el sistema para un nuevo ciclo.

Procedimiento "Selección_Ubicacion()"

Este procedimiento interpreta la cadena de texto leída desde el código QR, contenida en *mycameratarget.string1*, para determinar la ubicación final en la estantería. Mediante una estructura TEST…CASE, se compara esta cadena con los posibles valores válidos. Según el resultado, el robot se mueve a la posición previa correspondiente del estante, y desde allí avanza hasta la ubicación final para depositar la pieza.

En caso de que el valor leído no coincida con ninguno de los esperados, se muestra un mensaje de error en el FlexPendant notificando al usuario que la pieza es defectuosa.

```
LOCAL PROC Seleccion_Ubicacion()

    TEST mycameratarget.string1

        CASE "Estante 1. Posicion 1":
            MoveJ Estante1Pos1Pre,v100,z20,TCP_ToolVentosa\WObj:=Estante1;
            MoveLDO Estante1Pos1,v50,fine,TCP_ToolVentosa\WObj:=Estante1, Scalable_IO_0_DO15,0;
        CASE "Estante 1. Posicion 2":
            MoveJ Estante1Pos2Pre,v100,z20,TCP_ToolVentosa\WObj:=Estante1;
            MoveLDO Estante1Pos2,v50,fine,TCP_ToolVentosa\WObj:=Estante1, Scalable_IO_0_DO15,0;
        CASE "Estante 1. Posicion 3":
            MoveJ Estante1Pos3Pre,v100,z20,TCP_ToolVentosa\WObj:=Estante1;
            MoveLDO Estante1Pos3,v50,fine,TCP_ToolVentosa\WObj:=Estante1, Scalable_IO_0_DO15,0;
        CASE "Estante 2. Posicion 1":
            MoveJ Estante2Pos1Pre,v100,z20,TCP_ToolVentosa\WObj:=Estante2;
            MoveLDO Estante2Pos1,v50,fine,TCP_ToolVentosa\WObj:=Estante2, Scalable_IO_0_DO15,0;
        CASE "Estante 2. Posicion 2":
            MoveJ Estante2Pos2Pre,v100,z20,TCP_ToolVentosa\WObj:=Estante2;
            MoveLDO Estante2Pos2,v50,fine,TCP_ToolVentosa\WObj:=Estante2, Scalable_IO_0_DO15,0;
        CASE "Estante 2. Posicion 3":
            MoveJ Estante2Pos3Pre,v100,z20,TCP_ToolVentosa\WObj:=Estante2;
            MoveLDO Estante2Pos3,v50,fine,TCP_ToolVentosa\WObj:=Estante2, Scalable_IO_0_DO15,0;
        DEFAULT:
            TPWrite "La pieza es defectuosa";
    ENDTEST

ENDPROC
```

Figura 225. Procedimiento "Seleccion_Ubicacion()" de la práctica 5

Procedimiento principal "main_Práctica5_LecturaQR()"

Este es el bucle principal del programa. En cada iteración, se activa la señal de captura de imagen desde el componente inteligente "ImageCapture", se ejecuta el procedimiento "MoveToDetectedObject()" para realizar la lectura del código QR y la manipulación del cubo, y finalmente se llama a "GoHome()" para reiniciar el ciclo.

Al final de cada iteración, se reinician todas las señales digitales empleada, dejando el sistema preparado para procesar la siguiente pieza. De esta forma, el robot realiza de forma autónoma la lectura, interpretación y almacenamiento de los cubos en sus posiciones correspondientes dentro de la estantería.

```
PROC main_Práctica5_LecturaQR()

    WHILE TRUE DO

        SetDO Scalable_IO_0_DO14,1;  !Capturar foto
        SetDO Scalable_IO_0_DO14,0;

        MoveToDetectedObject;
        GoHome;

        !Reset final de las señales
        Reset Scalable_IO_0_DO14;
        Reset Scalable_IO_0_DO15;

    ENDWHILE

ENDPROC
```

Figura 226. Procedimiento principal "main_Práctica5_LecturaQR()"

Resultados

Los resultados obtenidos en esta práctica validan el correcto funcionamiento del sistema de visión artificial aplicado a la lectura e interpretación de códigos QR, así como la posterior clasificación y manipulación de las piezas por parte del robot.

Durante cada iteración, el sistema fue capaz de identificar de forma precisa el contenido de los códigos QR ubicados en la cara superior de los cubos, gracias a la herramienta de inspección **Leer código 2D (1-20).** Esta herramienta detectó correctamente la secuencia de texto de cada código, extrayendo la información del

destino (estante y posición) de manera fiable incluso cuando los cubos presentaban ligeras variaciones de orientación o posición.

La herramienta permite visualizar en todo momento cuántos códigos QR han sido detectados en la imagen, mostrando para cada uno su contenido, posición y orientación. En la figura 227 se recogen dos ejemplos ilustrativos de esta lectura.

Índice	Secuencia de carac...	Simbología	X	Y	Ángulo	Grado
0	Estante 2. Posicion 1	QR Code	426,450	335,437	-90,000	0,000
1	Estante 1. Posicion 3	QR Code	665,105	359,095	-89,983	0,000
2	Estante 2. Posicion 2	QR Code	548,061	453,412	-76,373	4,000
3	Estante 1. Posicion 1	QR Code	556,237	36,488	-78,077	0,000

Índice	Secuencia de caracteres	Simbología	X	Y	Ángulo	Grado
0	Estante 2. Posicion 2	QR Code	548,061	453,412	-76,373	4,000
1	Estante 1. Posicion 1	QR Code	556,237	36,488	-78,077	0,000
2	-	-	-	-	-	-

Figura 227. Visualizaciones de los códigos QR detectados en dos momentos diferentes

En el panel de resultados del entorno de visión de *RobotStudio*, se muestra un círculo verde (*Pass*) cuando un código QR es detectado correctamente, y un cuadrado rojo (*Fail*) en caso contrario. Así, durante toda la ejecución aparecen resultados de tipo *Pass* mientras hay piezas disponibles sobre la mesa. Únicamente al finalizar la recogida de todos los cubos se genera un resultado de tipo *Rechazo*, acompañado de una cruz roja, indicando que ya no quedan piezas por procesar.

Figura 228. Resultados trabajo de visión cuando hay QRs sobre la mesa

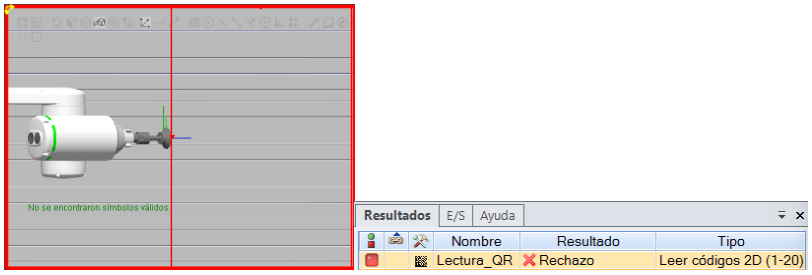


Figura 229. Resultados trabajo de visión cuando no hay más QRs sobre la mesa

Toda esta información se envía al programa RAPID, que ejecuta correctamente la secuencia completa: el robot se desplaza hasta la pieza detectada, la recoge mediante succión y la deposita con precisión en la ubicación correspondiente dentro de la estantería.

Además, se comprobó que el sistema respondió correctamente ante cada uno de los destinos definidos, mostrando una correspondencia clara y precisa entre el contenido

del código QR y la ubicación final del cubo, sin generar colisiones. Gracias a esta práctica, fue posible simular un proceso básico de gestión automatizada de almacenamiento, en el que cada pieza es identificada y clasificada según la información codificada en su superficie.

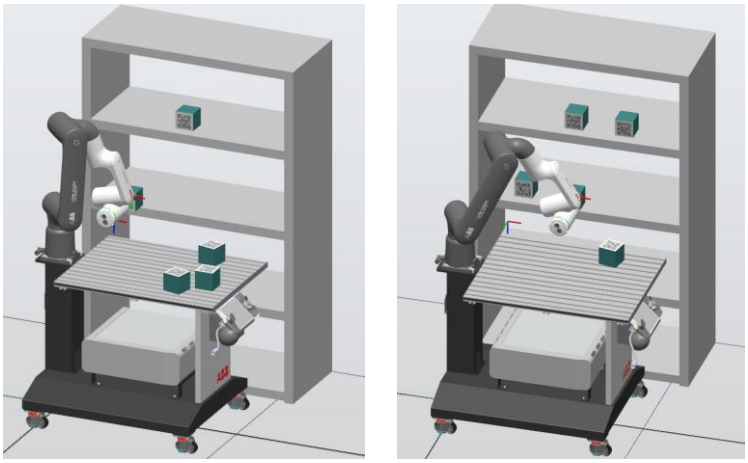


Figura 230. Proceso de colocación de las piezas en la estantería tras detectar su ubicación en el código QR

5.4. Integración con MATLAB y OPC UA

5.4.1. Introducción a la práctica

Esta práctica constituye una extensión del proyecto que se diferencia claramente del resto de las prácticas docentes desarrolladas anteriormente, ya que introduce un nuevo enfoque centrado en la integración del robot con una interfaz desarrollada en *MATLAB* mediante comunicación OPC UA.

A través de esta práctica, se demuestra el funcionamiento de un sistema colaborativo en el que las decisiones de actuación del robot no se toman de forma autónoma mediante visión artificial, sino que son tomadas por el usuario a través de una interfaz gráfica programada en *MATLAB*, simulando la gestión de un inventario.

El planteamiento consiste en una caja con tres tipos de piezas (de dos, tres y cuatro agujeros). A través de la interfaz, el usuario selecciona el tipo de pieza deseado, y el robot, utilizando visión artificial, detecta la pieza correspondiente dentro de la caja y la extrae, depositándola sobre la mesa para su recogida.

La comunicación entre *MATLAB* y *RobotStudio* se establece mediante el **protocolo OPC UA**, utilizando el software ***IoT Gateway de ABB*** como servidor. Esto permite transmitir al robot la orden de actuación y, a su vez, mantener actualizada en tiempo real la información sobre el número de piezas restantes de cada tipo.

Esta sexta práctica tiene como objetivo adquirir experiencia en la integración de sistemas de control externos dentro de entornos de automatización robótica, así como destacar la importancia de las comunicaciones industriales modernas. Su desarrollo permite aplicar de forma práctica conceptos fundamentales de la Industria 4.0, como la interoperabilidad, la conectividad y la supervisión remota.

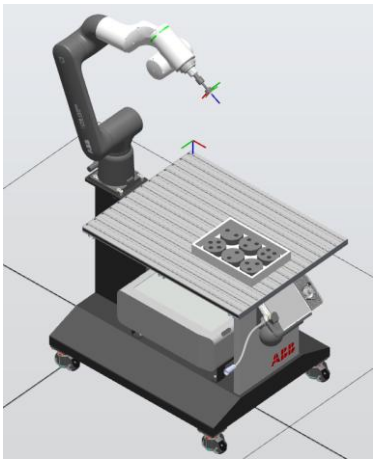


Figura 231. Estación de la práctica 6 Recuento de agujeros

5.4.2. Modelado de los componentes

Modelado de las piezas

Para la realización de esta práctica se han diseñado tres tipos diferentes de piezas, diferenciadas por el número de agujeros: dos, tres y cuatro. Todas ellas comparten una misma base geométrica, consistente en un cilindro de 100 mm de diámetro y 30 mm de altura, creado mediante la herramienta “Cilindro” del entorno de modelado.

A continuación, se ha modelado un segundo cilindro de 20 mm de diámetro y 40 mm de altura, que se emplea como herramienta de corte para generar los agujeros. Este cilindro se posiciona sobre la pieza base en las ubicaciones deseadas, respetando la equidistancia entre los centros de los orificios según el número requerido en cada pieza. Finalmente, se ha aplicado la operación “Restar” para eliminar los volúmenes y crear los agujeros, obteniendo así las tres variantes de piezas que componen esta práctica.

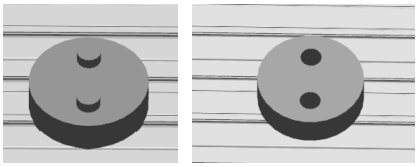


Figura 232. Proceso de modelado de una pieza con agujeros

Modelado de la caja contenedora

Las piezas se sitúan en el interior de una caja diseñada específicamente para esta práctica. Para su modelado, se han utilizado dos sólidos de tipo “Tetraedro”. El primero, de dimensiones 350 × 250 × 50 mm, representa el volumen exterior de la caja. El segundo, ligeramente más pequeño (330 × 230 × 40 mm), se ha utilizado como volumen interno a eliminar.

Ambos sólidos se han alineado cuidadosamente y posteriormente se ha aplicado la operación “Restar” para generar el hueco interior, obteniendo así una estructura que

actúa como contenedor. Para mejorar el contraste con las piezas grises, se ha aplicado un color blanco a la superficie de la caja, facilitando así la identificación de las piezas mediante visión artificial.

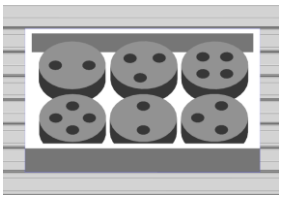


Figura 233. Modelado de la caja contenedora de las piezas

5.4.3. Diseño de la interfaz gráfica en Matlab

La interfaz gráfica desarrollada para esta práctica ha sido diseñada utilizando ***App Designer***, el entorno integrado de *MATLAB* que permite crear aplicaciones interactivas de forma visual. (MathWorks, App Designer, s.f.) Esta interfaz actúa como intermediario entre el usuario y el sistema robótico, permitiendo seleccionar el tipo de pieza deseada y visualizar en tiempo real el inventario disponible.

El diseño ha sido pensado con un enfoque claro, funcional y visualmente atractivo. Se ha optado por una distribución sencilla sobre un fondo blanco, empleando colores llamativos como el rojo o el azul para identificar rápidamente los elementos y facilitar la interacción del usuario sin necesidad de conocimientos avanzados.

En la parte superior se ha colocado un título principal con el texto INVENTARIO, centrado y con un tamaño de fuente grande en color rojo, utilizando un componente de tipo *Label*, con el objetivo de captar rápidamente la atención del usuario.

Justo debajo, se han organizado tres bloques visuales, uno para cada tipo de pieza disponible: piezas con 2, 3 y 4 agujeros. Cada uno de estos bloques contiene:

- **Imagen representativa de la pieza:** Se ha insertado mediante el componente Image y permite al usuario identificar visualmente el tipo de pieza que se desea.
- **Título descriptivo:** Se ha creado con el componente Label, que indica el tipo de pieza.
- **Etiqueta dinámica de cantidad:** Se ha situado debajo de la imagen y muestra el número actual de piezas restantes en la caja para ese tipo de pieza. Esta información se actualiza automáticamente en tiempo real mediante la conexión OPC UA con el robot.

Figura 234. Bloques de los tipos de piezas con su imagen y cantidad disponible

En la parte inferior de la interfaz se han colocado tres botones interactivos que permiten al usuario realizar la selección de la pieza deseada. Estos botones, de tipo *State Button*, están etiquetados como "2 Agujeros", "3 Agujeros" y "4 Agujeros". Al pulsar cualquiera de ellos, se envía una señal al robot para que recoja la pieza correspondiente y la deposite sobre la mesa. Además, tras cada extracción, el sistema actualiza el contador asociado, reflejando la cantidad restante de esa pieza.

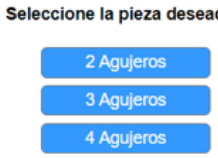


Figura 235. Botonera para realizar la selección del tipo de pieza deseado

Todos estos componentes han sido añadidos al área de diseño de *App Designer* desde el panel de componentes, simplemente arrastrándolos a la posición deseada. A continuación, se han configurado sus propiedades visuales (colores, tamaños, fuentes...) y sus acciones asociadas a través de *callbacks* programados en la ventana de código.

En resumen, esta interfaz gráfica ha sido diseñada para ofrecer una experiencia de usuario clara, sencilla y adecuada al entorno educativo. A través de ella, se demuestra la integración de *MATLAB* con un sistema robótico colaborativo mediante comunicación OPC UA, simulando un escenario de gestión en tiempo real de un inventario.

Figura 236. Interfaz del usuario para gestión del inventario por parte del usuario

5.4.4. Implementación comunicación OPC UA

Para lograr la integración entre el sistema robótico y la interfaz diseñada en *MATLAB*, se ha empleado el protocolo de comunicaciones industriales OPC UA, conocido por su fiabilidad, seguridad y capacidad de interoperabilidad entre distintos sistemas.

En esta práctica, se ha establecido un canal de comunicación bidireccional entre *MATLAB* (cliente) y *RobotStudio* (cliente), a través de un servidor intermedio gestionado por *ABB IoT Gateway*, el cual se conecta directamente con el controlador del robot. De este modo, el usuario puede seleccionar una acción desde la interfaz gráfica en *MATLAB*, y el robot responde ejecutando la orden correspondiente. Al mismo tiempo, el estado del inventario se actualiza en tiempo real en función de las acciones realizadas por el sistema.

A continuación, se detalla el proceso completo de configuración de esta comunicación, desde la creación del servidor y las señales necesarias, hasta su implementación tanto en *RobotStudio* como en *MATLAB*.

Configuración del servidor OPC UA

La configuración del servidor OPC UA se ha llevado a cabo utilizando el software *ABB IoT Gateway*, el cual establece una conexión entre el controlador del robot y los distintos clientes OPC UA, como *MATLAB* y *RobotStudio*.

En primer lugar, se accede al software *IoT Gateway* y se selecciona la opción correspondiente a la configuración del controlador. Una vez dentro, se escanean los controladores disponibles y se selecciona el que se desea vincular. Es posible marcar qué datos se desean mostrar, como el nombre del controlador, dirección IP, ID del sistema o nombre del sistema. Todos estos parámetros pueden ser modificados por el usuario, así como asignar un alias para facilitar su identificación.

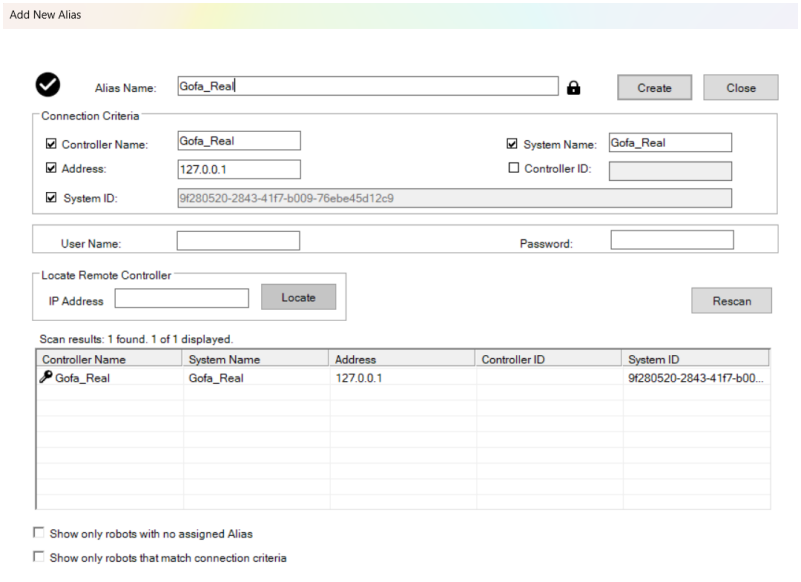


Figura 237. Búsqueda y configuración del controlador del robot

A continuación, se accede a la pestaña “Server Control”, donde se habilita el servidor OPC UA. Desde esta sección se puede activar o desactivar el servidor, establecer el puerto de comunicación (por defecto, el 61510) y consultar el nombre del servidor asignado. Al activar esta funcionalidad, el servidor queda listo para aceptar conexiones externas desde clientes OPC UA autorizados.

Figura 238. Pestaña del servidor IoT Gateway

Finalmente, se gestionan los certificados digitales, necesarios para garantizar una comunicación segura. En la pestaña "Client Certificates", se puede observar que al intentar conectarse desde *MATLAB* o *RobotStudio*, el sistema inicialmente rechaza los certificados al no estar verificados. Estos aparecen automáticamente en la sección de certificados rechazados. Para permitir la conexión, es necesario seleccionar cada certificado y pulsar el botón "*Trust*", lo que habilita el acceso al cliente seleccionado.

Una vez completada esta configuración, *ABB IoT Gateway* queda configurado como servidor OPC UA central del sistema, permitiendo una comunicación fiable y segura entre *MATLAB*, *RobotStudio* y el controlador virtual del robot.

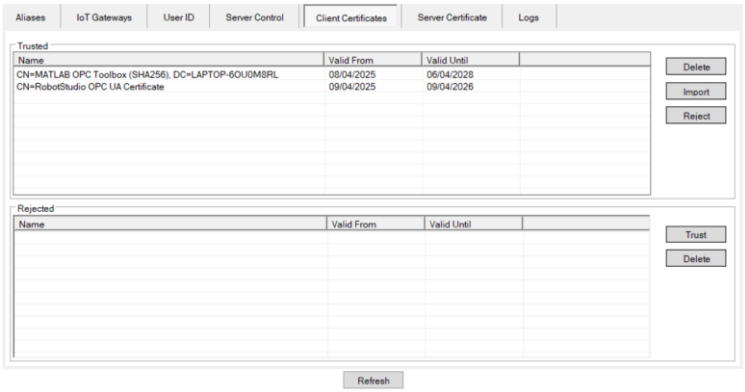


Figura 239. Pestaña de los certificados de clientes

Cliente OPC UA en RobotStudio y definición de señales

Para establecer la comunicación entre *RobotStudio* y el servidor configurado previamente en *ABB IoT Gateway*, ha sido necesario incorporar un cliente OPC UA dentro de la lógica de la estación. Esta conexión permite que *RobotStudio* actúe como cliente, igual que *MATLAB*, conectándose al servidor para intercambiar información con el controlador virtual del robot.

Para ello, se ha añadido un componente inteligente denominado **OpcUaClient**. Este se añade a la lógica de la estación accediendo al apartado PLC desde el árbol lógico de la estación y seleccionando dicho componente.

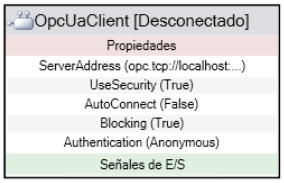


Figura 240. Componente inteligente OpcUaClient

Una vez insertado, se ha procedido a su configuración, especificando los parámetros de conexión correspondientes al servidor OPC UA previamente habilitado en el software *IoT Gateway*. La dirección IP se ha mantenido como *localhost*, ya que tanto el servidor como *RobotStudio* se ejecutan en el mismo equipo, mientras que el puerto se ha fijado en *61510*, que es el definido en la configuración del servidor.

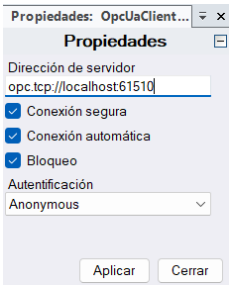


Figura 241. Establecimiento conexión del cliente con el servidor OPC UA

Tras configurar el cliente, se han creado las señales necesarias para el intercambio de información entre el controlador virtual del robot y la interfaz externa de *MATLAB*. En concreto, se han creado varias señales digitales de salida que se vinculan directamente con el controlador virtual del robot, y que permiten ejecutar diversas acciones en función de la selección del usuario en la interfaz.

La gestión de estas señales se ha llevado a cabo desde el menú "I/O System", accesible desde la pestaña "Controlador". En la sección "Signal", se han añadido un total de seis señales nuevas: tres de ellas definidas como salidas digitales, que serán activadas desde *MATLAB* para mandar órdenes al robot, y otras tres como entradas digitales, utilizadas para recibir dichas órdenes dentro del controlador virtual del robot.

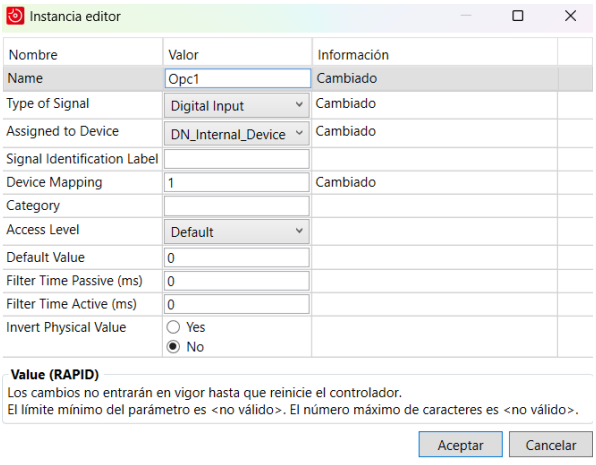


Figura 242. Creación de las señales para gestionar la comunicación OPC UA

Tras definir las señales, el siguiente paso consiste en integrarlas correctamente en el componente inteligente OpcUaClient. Este componente es el que habilita la conexión entre *RobotStudio* y el servidor OPC UA alojado en *IoT Gateway*. Para llevar a cabo esta vinculación, se accede a la opción "Configurar" del componente haciendo clic derecho sobre él. En la ventana emergente, se visualiza el árbol de nodos en el panel izquierdo, desde donde se localizan las señales previamente creadas.

Cada una de las señales de salida (aquellas que serán modificadas por *MATLAB*) se arrastran al apartado "Señales del componente inteligente" y se clasifica como señal de salida. Este paso es fundamental para que *MATLAB* pueda escribir sobre ellas y activar acciones específicas desde la interfaz del usuario.

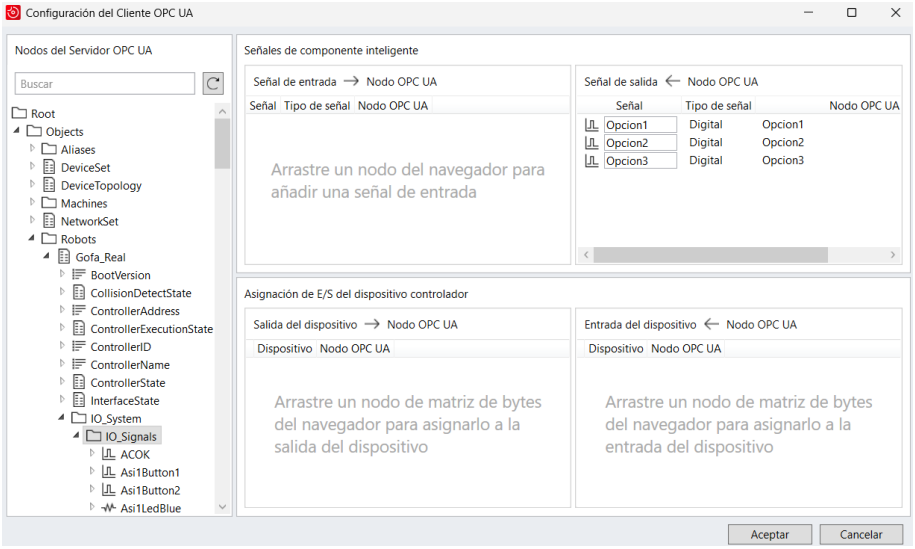


Figura 243. Configuración de las señales de salida del componente inteligente OpcUaClient

Por otro lado, las señales de entrada se asocian directamente al controlador virtual, lo que permite que los cambios realizados por *MATLAB* se reflejen en la ejecución del robot, conectando así todos los elementos del sistema.

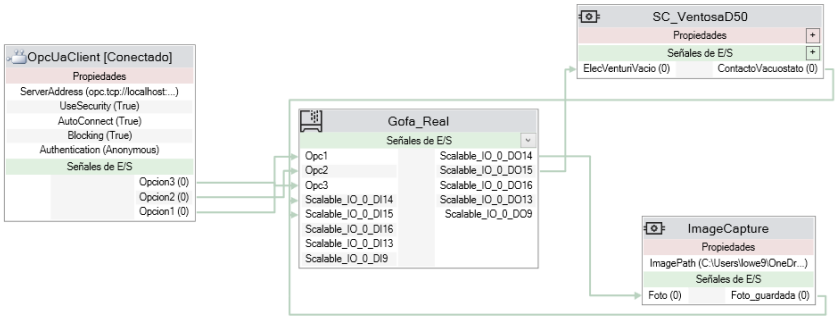


Figura 244. Conexión del componente inteligente OpcUaClient con el controlador virtual

Con esta configuración, *RobotStudio* se integra completamente dentro del sistema de comunicación OPC UA, actuando como cliente que recibe instrucciones desde *MATLAB* y actúa acorde a ello. Esto permite establecer una comunicación bidireccional fluida, controlar de forma remota el sistema robótico y ejecutar las órdenes del usuario con total precisión.

Lógica de comunicación OPC UA en MATLAB

Una vez diseñada la interfaz gráfica con *App Designer*, necesitamos vincular los botones interactivos con las señales digitales del controlador virtual del robot. Esta conexión se lleva a cabo gracias al protocolo de comunicaciones industriales OPC UA,

donde *MATLAB* actúa como cliente, enviando órdenes y recibiendo datos actualizados del robot a través del servidor configurado en *ABB IoT Gateway.*

Propiedades privadas

En la cabecera de la clase se declaran las propiedades que gestionan el estado de la aplicación y su comunicación con el servidor OPC UA:

- **uaClient:** conserva la sesión OPC UA abierta con el servidor.
- **nodoOpcionX:** Identifica los nodos que activan las órdenes de recogida de piezas de 2, 3 o 4 agujeros.
- **contadorNodeX:** Representa los nodos que almacenan la cantidad restante de cada tipo de pieza.
- **updateTimer:** Temporizador que se dispara periódicamente para actualizar en tiempo real los valores del inventario mostrados en la interfaz.

```
properties (Access = private)
    uaClient
    nodoOpcion1
    nodoOpcion2
    nodoOpcion3
    contadorNode1
    contadorNode2
    contadorNode3
    updateTimer
end
```

Figura 245. Propiedades privadas

"startupFcn()"

Esta función se ejecuta automáticamente al iniciar la aplicación. Su principal objetivo es establecer y preparar la comunicación con el servidor OPC UA.

En primer lugar, se ha creado un cliente OPC UA con la dirección IP del servidor (*localhost*) y el puerto 61510, configurado previamente en el software *IoT Gateway*, Una vez definido, se ejecuta la conexión con el servidor mediante la función "connect".

Una vez conectado el cliente, la función localiza los nodos necesarios dentro del espacio de direcciones del servidor. En concreto, se identifican los tres nodos correspondientes a los contadores de piezas (entradas) y los tres nodos que representan las órdenes de solicitud de piezas (salidas). Estos nodos se buscan por nombre mediante la función "findNodeByName" y se asignan a sus respectivas propiedades privadas.

A continuación, se configura un temporizador ("timer") que se ejecuta cada 0.5 segundos de forma periódica y que tiene como objetivo mantener actualizada la interfaz del usuario con la información más reciente del servidor. Este temporizador invoca la función "actualizarContadores()", pasando como argumento la propia aplicación para que pueda acceder a sus propiedades y objetos.

Finalmente, se ejecuta la función "cerrarYSalir()" que se encarga de limpiar recursos antes de terminar la ejecución de la aplicación.

```
% Code that executes after component creation
function startupFcn(app)
    app.uaClient = opcua("localhost", 61510);
    connect(app.uaClient);

    app.contadorNode1 = findNodeByName(app.uaClient.Namespace, 'contador_2agujeros', '-once');
    app.contadorNode2 = findNodeByName(app.uaClient.Namespace, 'contador_3agujeros', '-once');
    app.contadorNode3 = findNodeByName(app.uaClient.Namespace, 'contador_4agujeros', '-once');

    app.nodoOpcion1 = findNodeByName(app.uaClient.Namespace, 'Opcion1', '-once');
    app.nodoOpcion2 = findNodeByName(app.uaClient.Namespace, 'Opcion2', '-once');
    app.nodoOpcion3 = findNodeByName(app.uaClient.Namespace, 'Opcion3', '-once');

    app.updateTimer = timer( ...
        'ExecutionMode', 'fixedRate', ...
        'Period', 0.5, ...
        'TimerFcn', @(~,~) actualizarContadores(app));

    start(app.updateTimer);
    app.UIFigure.CloseRequestFcn = @(src, event) cerrarYSalir(app);
end
```

Figura 246. Función "startupFcn (app)" de App Designer

"actualizarContadores()"

Esta función es la encargada de mantener actualizado en todo momento el inventario que se muestra en la interfaz gráfica.

El temporizador configurado en "startupFcn()" ejecuta esta función periódicamente. Su lógica consiste en leer los valores actuales de los nodos de tipo contador mediante la función "readValue". Estos valores, que representan el número de piezas restantes de cada tipo, se convierten en texto y se muestran en las etiquetas correspondientes de la interfaz. De este modo, el usuario es capaz de visualizar la cantidad de piezas restantes de cada tipo de manera dinámica.

Todo el bloque de código se encuentra incorporado en una estructura try/catch que previene de posibles errores de conexión. En caso de producirse un fallo en la lectura de algún nodo, se muestra un mensaje en la consola de *MATLAB* sin interrumpir la ejecución del programa. que la interfaz siga funcionando mostrando un mensaje de error en la consola de *MATLAB*. De esta forma se consigue una aplicación robusta y tolerante a fallos temporales de comunicación.

```
function actualizarContadores(app)
    try
        valor1 = readValue(app.uaClient, app.contadorNode1);
        valor2 = readValue(app.uaClient, app.contadorNode2);
        valor3 = readValue(app.uaClient, app.contadorNode3);

        app.CantidadLabel.Text = ['Cantidad: ', num2str(valor1)];
        app.CantidadLabel_2.Text = ['Cantidad: ', num2str(valor2)];
        app.CantidadLabel_3.Text = ['Cantidad: ', num2str(valor3)];
    catch ME
        disp(['Error al actualizar contadores: ', ME.message]);
    end
end
```

Figura 247. Función "actualizarContadores(app)" de App Designer

"manejarBoton(nodoOpcion, nodoContador, labelDestino)"

Esta función centraliza la lógica de respuesta a la interacción del usuario cuando pulsa alguno de los botones que solicita piezas.

Recibe como argumentos el nodo de salida correspondiente al tipo de pieza seleccionado, el nodo del contador asociado y la etiqueta visual que debe actualizarse. Su funcionamiento consiste en: escribir un valor 1 en el nodo de acción para simular

un pulso digital, luego hacer una pausa de 0.2 segundos para asegurar que el servidor detecte dicho flanco positivo y restablecer el valor a 0, finalizando así el ciclo del pulso. Después, la función lee el nuevo valor del contador tras la extracción de la pieza y actualiza la etiqueta correspondiente.

Todo el proceso está encapsulado dentro de un bloque try/catch para garantizar que, en caso de producirse un error en la lectura o escritura de los nodos, no se interrumpa el funcionamiento global.

Además, es importante destacar que, al tratarse de una función genérica, puede ser reutilizada con los distintos nodos y etiquetas sin necesidad de duplicar código, favoreciendo la posibilidad de escalabilidad.

```
function manejarBoton(app, nodoOpcion, nodoContador, labelDestino)
    try
        writeValue(app.uaClient, nodoOpcion, 1);
        pause(0.2);
        writeValue(app.uaClient, nodoOpcion, 0);

        valor = readValue(app.uaClient, nodoContador);
        labelDestino.Text = ['Cantidad: ', num2str(valor)];
    catch ME
        disp(['Error al manejar botón: ', ME.message]);
    end
end
```

Figura 248. Función "manejarBoton(nodoOpcion, nodoContador, labelDestino)" de App Designer

Callbacks de los botones

Cada uno de los tres botones interactivos de la interfaz ("2 Agujeros", "3 Agujeros" y "4 Agujeros") está vinculado a un *callback* específico que se activa al ser pulsado. Estos *callbacks* simplemente llaman a la función "manejarBoton()", pasando los nodos y etiquetas correspondientes a su tipo de pieza.

De esta manera, la aplicación interpreta la acción del usuario y la convierte en una instrucción OPC UA que activa una salida digital y actualiza automáticamente el inventario visualizado.

```
% Button pushed function: AgujerosButton_4
function AgujerosButton_4Pushed(app, event)
    manejarBoton(app, app.nodoOpcion1, app.contadorNode1, app.CantidadLabel);
end

% Button pushed function: AgujerosButton_5
function AgujerosButton_5Pushed(app, event)
    manejarBoton(app, app.nodoOpcion2, app.contadorNode2, app.CantidadLabel_2);
end

% Button pushed function: AgujerosButton_6
function AgujerosButton_6Pushed(app, event)
    manejarBoton(app, app.nodoOpcion3, app.contadorNode3, app.CantidadLabel_3);
end
```

Figura 249. Callbacks de los botones

"cerrarYSalir()"

Esta función es la encargada de gestionar correctamente el proceso de cierre de la aplicación, garantizando que todos los recursos empleados en la aplicación se liberen adecuadamente.

Cuando el usuario cierra la ventana principal de la aplicación, esta función se ejecuta automáticamente, ya que ha sido definida como “CloseRequestFcn” en “startupFcn”.

En primer lugar, intenta detener y eliminar el temporizador que mantiene el inventario actualizado, evitando así que continúe ejecutándose en segundo plano. A continuación, libera la memoria asociada a los objetos del cliente OPC UA y cierra la sesión establecida. Por último, ejecuta “delete(app)”, cerrando la interfaz y finalizando la aplicación.

Gracias a esta función, se garantiza un cierre seguro, evitando errores en futuras sesiones o conflictos por recursos no liberados.

```
function cerrarYSalir(app)
    try
        stop(app.updateTimer);
        delete(app.updateTimer);
    catch
    end

    delete(app);
end
```

Figura 250. Función “cerrarYSalir(app)” de App Designer

5.4.5. Tarea de visión

En esta práctica se ha empleado la herramienta de inspección **Recuentro Patrones PatMax** del entorno de visión artificial de *RobotStudio*, con el objetivo de detectar y contar con precisión los agujeros presentes en cada una de las piezas situadas dentro de la caja. Esta herramienta permite localizar de forma precisa un número determinado de patrones previamente entrenados, incluso si presentan ligeras variaciones de posición o rotación.

La configuración inicial de la herramienta comienza seleccionando un patrón de referencia basado en la forma circular de los agujeros. Este patrón se entrena sobre uno de los agujeros de las piezas, empleando un modelo circular como referencia. A continuación, se define un área de búsqueda rectangular sobre cada uno de los cilindros.

Una vez aplicada la herramienta sobre las piezas, cada agujero detectado aparece marcado con una cruz verde que señala su centro, acompañado de un número que indica el orden de detección.

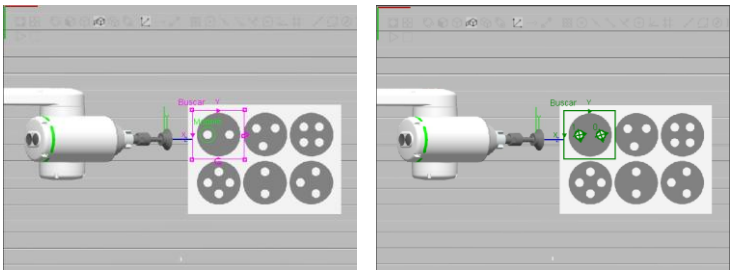


Figura 251. Entrenamiento de la herramienta de recuento Patrones PatMax

Dentro de las opciones de configuración de la herramienta, se han establecido los límites de recuento para filtrar piezas fuera de especificaciones. En este caso, se ha definido un mínimo de dos y un máximo de cuatro patrones, lo que permite asegurar que únicamente las piezas con dos, tres o cuatro agujeros sean consideradas válidas. Cualquier otra pieza que no cumpla con dichos criterios será descartada automáticamente por el sistema. Además, se ha ajustado el umbral de coincidencia para asegurar que se contabilicen correctamente los agujeros de cada una de las piezas.

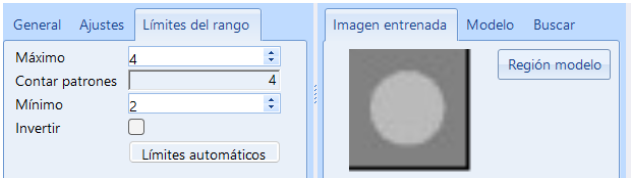


Figura 252. Ajustes de los límites de rango de la herramienta PatMax junto al modelo entrenado

Paralelamente a esta herramienta, se ha incorporado la herramienta de ubicación **Blobs (1-10)** con el objetivo de obtener las coordenadas de cada una de las piezas. Esta herramienta identifica regiones conectadas de píxeles dentro de una imagen, denominadas “blobs”, y calcula el centroide de cada una de ellas. Para su configuración, se ha situado una región circular sobre cada cilindro, de manera que al ejecutar la herramienta se representa un eje sobre el centroide de la pieza correspondiente. Esta información es la que posteriormente se utiliza para localizar la pieza correctamente.

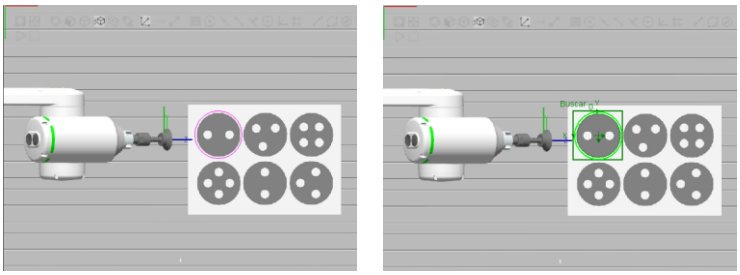


Figura 253. Entrenamiento de la herramienta de ubicación Blobs (1-10)

Finalmente, se han definido un total de seis salidas a RAPID, una para cada una de las piezas. En cada una de ellas se ha configurado el envío de las coordenadas del centroide obtenidas por la herramienta Blobs (posición x, y y rotación z), junto con el número de agujeros detectado por la herramienta PatMax, el cual se transmite mediante la variable val1.

Toda esta información, combinada con la orden enviada desde la interfaz de *MATLAB* a través de la comunicación OPC UA, permite al robot localizar y seleccionar la pieza deseada.

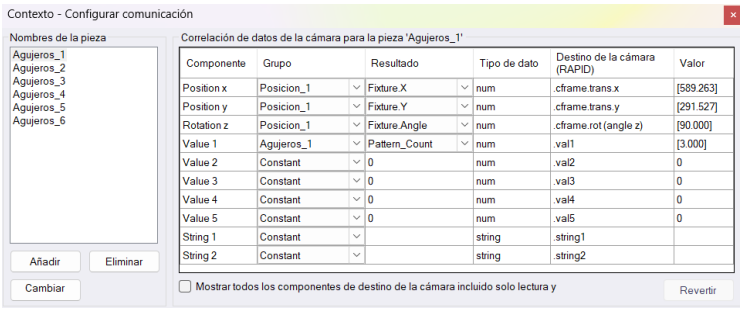


Figura 254. Salidas a RAPID del trabajo de visión job_practica6.job

El trabajo de visión fue guardado bajo el nombre "job_practica6.job" en la carpeta "Sensores In-Sight", siguiendo la misma metodología aplicada en las prácticas anteriores.

5.4.6. Implementación del código RAPID

Declaración de herramientas, objetos de trabajo y posiciones

Al inicio del módulo, se ha definido la herramienta *TCP_ToolVentosa* utilizada para realizar el agarre de las piezas mediante succión. A continuación, se ha establecido el objeto de trabajo *cam_WObj_P6,* que representa el plano de visión donde se detectan las piezas con la cámara y se ha situado a la altura de cogida de las piezas.

Además, se han declarado varias posiciones claves para el robot:

- **"Home"**: Actúa como punto inicial y de espera, asegurando un retorno seguro tras cada operación.
- **"PreDejar" y "Dejar"**: Corresponden respectivamente a la posición previa y final dónde se depositan las piezas extraídas sobre la mesa.
- **"myrobtarget"**: Posición auxiliar que se actualiza en función de los datos obtenidos desde la cámara de visión.

Para llevar el control del inventario en tiempo real, se han declarado tres variables persistentes PERS ("contador_2agujeros", "contador_3agujeros" y "contador_4agujeros"), inicializadas con un valor de dos unidades cada una. Estas variables almacenan la cantidad disponible de cada tipo de pieza y se decrementan automáticamente tras cada extracción. Además, su valor es accesible desde *MATLAB* gracias a la comunicación establecida mediante OPC UA, lo que permite mantener la interfaz del inventario actualizada.

También, se ha declarado una constante de tipo string que contiene el nombre del trabajo de visión artificial previamente configurado y entrenado en el entorno de visión artificial y una variable del tipo *cameratarget* que es la encargada de almacenar los resultados obtenidos por la cámara, incluyendo posición, orientación y número de agujeros detectados.

Finalmente, se han declarado tres variables del tipo *intnum*, asociadas a las interrupciones OPC UA. Estas interrupciones son fundamentales pata detectar en

tiempo real las órdenes enviadas desde *MATLAB*, permitiendo así que el robot pueda responder rápidamente a las solicitudes del usuario mediante la ejecución de rutinas "TRAP".

```
LOCAL PERS tooldata TCP_ToolVentosa:=[TRUE,[[0,0,129.5],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];

LOCAL PERS wobjdata cam_Wobj_P6:=[FALSE,TRUE,"",[[170,304.269,55.7303],[0.00377863,0.999993,0,0]],[[692.815,399.707,0],[0.707107,0,0,0.707107]]];

LOCAL CONST robtarget myrobtarget:=[[0,0,0],[1,0,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
LOCAL CONST robtarget Home:=[[540.684087634,0,773.032029102],[0.500000007,0,0.8660254,0],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget PreDejar:=[[398.144,1.591,181.000488061],[0,-0.707106781,0.707106781,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
LOCAL CONST robtarget Dejar:=[[398.144,1.591,51.000488061],[0,-0.707106781,0.707106781,0],[0,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

LOCAL VAR num num_agujeros;
LOCAL PERS num contador_2agujeros:=2;
LOCAL PERS num contador_3agujeros:=2;
LOCAL PERS num contador_4agujeros:=2;

LOCAL CONST string myjob := "job_practica6.job";
LOCAL VAR cameratarget mycameratarget;

!Interrupciones
LOCAL VAR intnum intOpc1;
LOCAL VAR intnum intOpc2;
LOCAL VAR intnum intOpc3;
```

Figura 255. Declaraciones iniciales de la práctica 6

Procedimiento "DepositarYHome()"

Este procedimiento gestiona el trayecto desde la zona de recogida hasta el área de depósito. En primer lugar, el robot se desplaza a la posición "PreDejar" y, desde allí, desciende hasta "Dejar", donde libera la pieza desactivando la señal de succión (DO15). Tras completar la acción, retorna a la posición Home, quedando listo para ejecutar una nueva orden.

```
LOCAL PROC DepositarYHome()

    MoveJ PreDejar,v100,z100,TCP_ToolVentosa\WObj:=wobj0;
    MoveLDo Dejar,v50,fine,TCP_ToolVentosa\WObj:=wobj0, Scalable_IO_0_DO15,0;
    MoveJ Home,v200,z100,TCP_ToolVentosa\WObj:=wobj0;

ENDPROC
```

Figura 256. Procedimiento "DepositarYHome()" de la práctica 6

Procedimiento "MoveToDetectedObject()"

Este procedimiento se encarga de capturar la imagen del área de trabajo, identificar la pieza correspondiente al tipo solicitado y ejecutar el movimiento de recogida. Para ello, se activa la señal digital que ordena la toma de imagen (DO14), se carga el trabajo de visión previamente configurado y se cambia el modo de la cámara a ejecución.

A continuación, se lleva a cabo la carga de forma manual de la imagen capturada por el componente inteligente "ImageCapture", cargando la imagen y cambiando la cámara al modo ejecución de nuevo. Una vez realizado, se confirma la acción activando la entrada correspondiente (DI2).

Posteriormente, el procedimiento entra en un bucle donde analiza los resultados obtenidos por la cámara hasta encontrar una pieza cuyo número de agujeros coincida con el valor almacenado en "num_agujeros". Cuando se localiza una pieza válida, se actualiza el objeto de trabajo con las coordenadas enviadas por el sistema de visión

artificial. Finalmente, el robot se desplaza hasta la pieza, activa la ventosa para realizar el agarre y la eleva ligeramente para retirarla de la caja antes de llamar al procedimiento "DepositarYHome()".

```
LOCAL PROC MoveToDetectedObject()

    SetDO Scalable_IO_0_DO14,1;  !Capturar foto
    SetDO Scalable_IO_0_DO14,0;

    CamSetProgramMode cam;
    CamLoadJob cam, myjob;      !Carga trabajo ya entrenado
    CamSetRunMode cam;

    TPWrite "Esperando seña de la cámara...";

    WaitDI Scalable_IO_0_DI2,1;
    CamReqImage cam;
    CamGetResult cam, mycameratarget;

    WHILE mycameratarget.val1<>num_agujeros DO
        CamGetResult cam, mycameratarget;
    ENDWHILE

    cam_Wobj_P6.oframe := mycameratarget.cframe;

    !Efecto succión al llegar a posición
    MoveJDo myrobtarget,v100,fine,TCP_ToolVentosa\WObj:=cam_Wobj_P6, Scalable_IO_0_DO15,1;
    MoveL Offs (myrobtarget,0,0,-70),v50,z10,TCP_ToolVentosa\WObj:=cam_Wobj_P6;
    DepositarYHome;

ENDPROC
```

Figura 257. Procedimiento "MoveToDetectedObject()" de la práctica 6

Procedimiento "SetupInterrupciones()"

Esta función configura el sistema de interrupciones que permite al robot responder de forma inmediata a las señales enviadas desde la interfaz de *MATLAB*. Primero, se eliminan las posibles interrupciones activas y se conectan las señales digitales "Opc1", "Opc2" y "Opc3" a sus respectivas rutinas "TRAP". Estas señales digitales están vinculadas a las salidas de *MATLAB* mediante el cliente OPC UA de *RobotStudio*, y su activación desencadena automáticamente la ejecución de las rutinas correspondientes.

```
LOCAL PROC SetupInterrupciones()

    IDelete intOpc1;
    IDelete intOpc2;
    IDelete intOpc3;

    CONNECT intOpc1 WITH trap_Opc1;
    CONNECT intOpc2 WITH trap_Opc2;
    CONNECT intOpc3 WITH trap_Opc3;

    ISignalDI Opc1, 1, intOpc1;  ! Opc1
    ISignalDI Opc2, 1, intOpc2;  ! Opc2
    ISignalDI Opc3, 1, intOpc3;  ! Opc3

ENDPROC
```

Figura 258. Procedimiento "SetupInterrupciones()" de la práctica 6

Rutinas TRAP (trap_Opc1, trap_Opc2, trap_Opc3)

Estas rutinas de interrupción se encargan de gestionar las solicitudes de piezas enviadas desde la interfaz de *MATLAB*. Cada rutina está vinculada a un tipo concreto de pieza: de 2, 3 o 4 agujeros. Cuando se detecta la activación de una de las señales digitales correspondientes, se ejecuta la rutina asociada.

En primer lugar, se verifica si quedan unidades disponibles del tipo de pieza solicitado. Si es así, se asigna el número correspondiente a la variable "num_agujeros" y se

invoca el procedimiento “MoveToDetectedObject()” para proceder con la captura de imagen y recogida de la pieza detectada. Una vez completada la operación, se reduce en uno el valor del contador correspondiente.

Si, por el contrario, no quedan piezas disponibles del tipo solicitado, se informa al usuario mediante un mensaje en el FlexPendant. En ese caso, no se realiza ningún movimiento, y las interrupciones se reactivan para seguir a la espera de nuevas órdenes válidas. De este modo, se evita cualquier acción incorrecta o innecesaria.

```
TRAP trap_Opc1

    TPWrite "Interrupción de OPC1";
    IF contador_2agujeros = 0 THEN
        TPWrite "No quedan piezas de 2 agujeros. Seleccione otra opción.";
        SetupInterrupciones;
        RETURN;     !Salir del trap sin continuar
    ENDIF

    num_agujeros := 2;
    MoveToDetectedObject;
    contador_2agujeros := contador_2agujeros - 1;


ENDTRAP
```

Figura 259. Rutina “trap_Opc1” de la práctica 6

Procedimiento principal “main_Práctica6_RecuentoAgujeros()”

El procedimiento principal comienza con el restablecimiento de los contadores de piezas disponibles a su valor inicial (dos unidades por tipo), lo cual es necesario ya que dichas variables han sido declaradas como persistentes (PERS) y podrían conservar valores residuales de ejecuciones anteriores.

A continuación, se ejecuta el procedimiento “SetupInterrupciones”, que se encarga de configurar y activar las interrupciones asociadas a las señales digitales provenientes de *MATLAB*. Estas interrupciones permiten que el sistema pueda reaccionar de manera inmediata y asíncrona a cada petición del usuario.

Finalmente, el sistema entra en un bucle continuo WHILE...TRUE, mostrando un mensaje de espera en el FlexPendant. Este bucle mantiene el robot en funcionamiento de forma indefinida, preparado para actuar tan pronto como reciba una nueva orden, sin requerir intervención adicional por parte del usuario.

```
PROC main_Práctica6_RecuentoAgujeros()

    !Reset de contadores por persistencia de variables
    contador_2agujeros:=2;
    contador_3agujeros:=2;
    contador_4agujeros:=2;

    SetupInterrupciones;
    TPWrite "Esperando señales OPC...";

    WHILE TRUE DO
        WaitTime 1;
    ENDWHILE

ENDPROC
```

Figura 260. Procedimiento principal “main_Práctica6_RecuentoAgujeros()”

5.4.7. Resultados

Los resultados obtenidos en esta práctica permiten validar el correcto funcionamiento del sistema completo, desde la interfaz en *MATLAB* hasta la ejecución final por parte del robot colaborativo, pasando por el análisis mediante visión artificial.

Durante la ejecución, la interacción entre el usuario y la aplicación fue sencilla. A través de la interfaz, el usuario ha podido seleccionar fácilmente el tipo de pieza deseada y la orden se ha transmitido de forma correcta y fluida al robot mediante el protocolo de comunicaciones industriales OPC UA. El sistema ha respondido de manera rápida ante cada solicitud, manipulando únicamente aquellas piezas que cumplían los requisitos especificados.

En cuanto al trabajo de visión artificial entrenado, se han empleado dos herramientas principales: la herramienta de **recuento de patrones PatMax** y la herramienta de ubicación **Blobs (1-10).**

La herramienta de recuento ha sido capaz de identificar con precisión el número de agujeros presentes en cada una de las piezas de la caja, incluso en variaciones en su posición y la herramienta de ubicación Blobs ha marcado con exactitud el centroide de la pieza, indicando su posición y orientación.

Cuando una pieza válida es detectada (*pass*), se visualizan los agujeros marcados con cruces verdes y numerados consecutivamente. En el panel de resultados aparece un círculo verde que indica que la pieza ha sido correctamente identificada

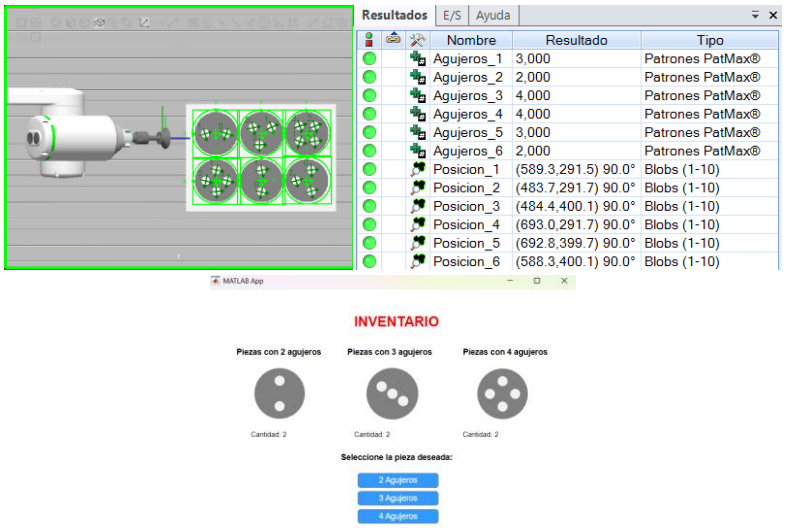


Figura 261. Resultados iniciales del trabajo de visión junto a la interfaz del usuario en dicho momento

Por otro lado, en caso de que no se detectara una pieza en la región esperada o el número de agujeros no estuviera dentro del rango válido (menos de 2 o más de 4), ambas herramientas han devuelto una respuesta de fail. En estos casos, la herramienta de recuento ha mostrado una cruz roja junto al número 0, indicando que no se había encontrado un patrón válido, mientras que la herramienta de blobs también ha mostrado una cruz roja acompañada del mensaje “Ningún blob

encontrado". Este doble mecanismo de verificación aseguraba que el sistema solo actúe sobre piezas correctas, evitando errores de manipulación.

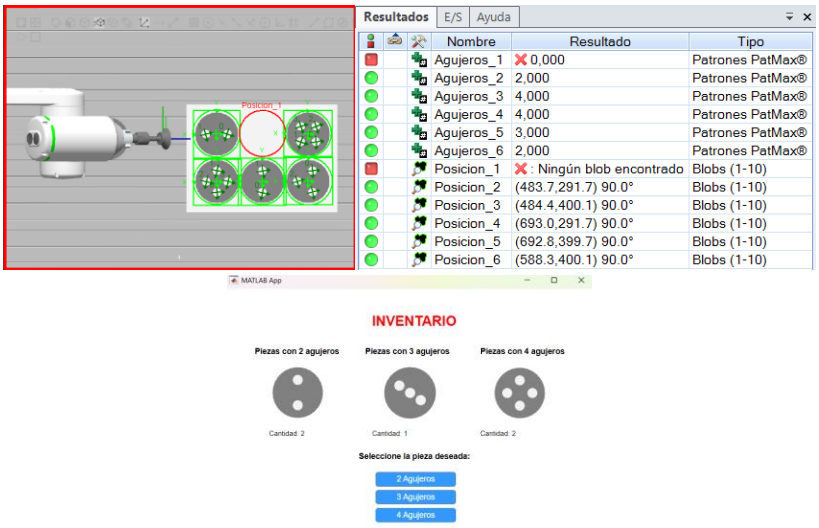


Figura 262. Resultados del trabajo de visión junto a la interfaz del usuario cuando ya se ha extraído una pieza

En todo momento, el robot ha cogido correctamente las piezas validas del interior de la caja y las ha depositado con precisión en la posición designada sobre la mesa. Además, el sistema ha sido capaz de gestionar el inventario en tiempo real, actualizando la cantidad de cada tipo de pieza tras cada extracción. Esto ha sido posible gracias al uso de variables persistentes en el código RAPID y su enlace directo con *MATLAB* mediante OPC UA.

En resumen, esta práctica ha permitido comprobar tanto la capacidad del sistema para identificar piezas específicas mediante visión artificial, como la eficacia de la comunicación bidireccional entre *MATLAB* y *RobotStudio*. La integración de todos estos componentes ha demostrado la correcta implementación de un sistema automatizado completo, característico de la Industria 4.0.

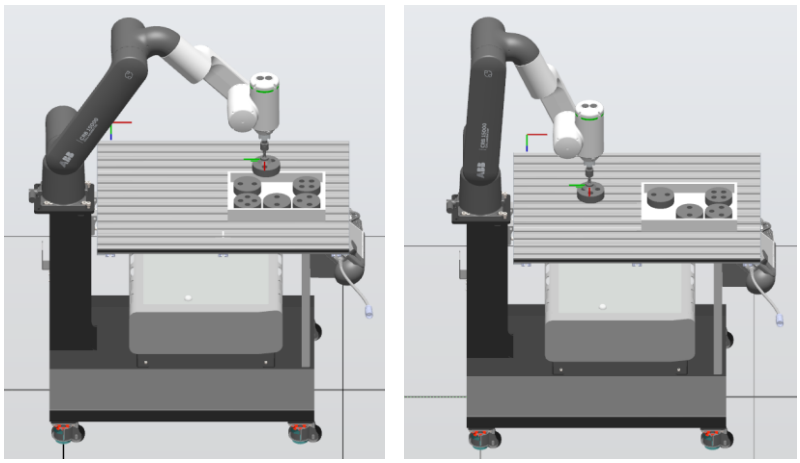


Figura 263. Proceso de selección y extracción de la pieza solicitada por el usuario

6. CONCLUSIONES Y FUTUROS TRABAJOS

6.1. Conclusiones generales

A lo largo del presente Trabajo de Fin de Grado se ha llevado a cabo el diseño, desarrollo y validación de un conjunto de seis prácticas docentes enfocadas en la enseñanza de la robótica colaborativa y la visión artificial. El proyecto ha permitido integrar múltiples tecnologías industriales dentro de un entorno simulado, combinando el uso de ABB *RobotStudio*, el lenguaje de programación RAPID, el software de visión Cognex *In-Sight Explorer* y el protocolo de comunicación OPC UA, todo ello bajo un enfoque didáctico y progresivo.

Cada práctica ha planteado un escenario de aprendizaje diferente, desde ejercicios de clasificación por forma y color, pasando por la inspección de defectos y la resolución de problemas como la Torre de Hanoi, hasta llegar a tareas más avanzadas como la lectura de códigos QR y la gestión de inventario mediante una interfaz gráfica creada en *MATLAB*. Estas experiencias han ofrecido al alumnado la oportunidad de enfrentarse a situaciones diversas, representativas de aplicaciones reales en la industria.

Además, el proyecto ha reforzado la importancia de la interoperabilidad entre sistemas, la programación modular y estructurada en RAPID, así como la utilidad de los entornos simulados para desarrollar soluciones completas antes de su implementación en entornos físicos. En conjunto, estas prácticas han acercado al alumnado a los principios fundamentales de la Industria 4.0, permitiéndoles no solo comprender el funcionamiento de los sistemas robotizados, sino también su integración con otras tecnologías dentro de un flujo de trabajo automatizado y conectado.

6.2. Aplicabilidad en entornos educativos reales

Aunque este Trabajo Fin de Grado no se ha implementado aún con alumnado, el conjunto de prácticas desarrolladas ha sido concebido con un enfoque didáctico, lo que garantiza su posible aplicación en contextos formativos reales dentro del ámbito de la automatización y la robótica industrial.

Las seis prácticas diseñadas abarcan distintos niveles de dificultad, desde tareas básicas como la clasificación por forma y color, hasta aplicaciones más avanzadas que integran visión artificial, programación en RAPID e interfaces gráficas interactivas con *MATLAB*, ofreciendo un aprendizaje gradual que facilita la asimilación progresiva de conceptos clave.

Además, la utilización de herramientas ampliamente extendidas en el entorno industrial, como *RobotStudio*, *MATLAB*, el software *In-Sight Explorer* o el protocolo de comunicación OPC UA, permite al alumnado familiarizarse con entornos profesionales reales sin necesidad de equipamiento físico, eliminando riesgos y costes. A través de estas simulaciones, se facilita el acceso a competencias técnicas que son fácilmente transferibles al mundo laboral.

Si bien en este trabajo se han desarrollado únicamente algunas aplicaciones de visión artificial como la identificación de patrones con PatMax®, el recuento de agujeros, la lectura de códigos QR o la inspección de medidas, el entorno de visión integrado en *RobotStudio* ofrece muchas más posibilidades como la lectura de códigos de barras, OCR, detección de bordes, inspección superficial o análisis de presencia/ausencia.

En este sentido, el proyecto se presenta como una base sobre la que construir futuras actividades docentes más complejas o personalizadas. De este modo, se ofrece al profesorado un recurso flexible, adaptable y de fácil integración en asignaturas como Sistemas Robotizados.

Más allá de su valor técnico, estas prácticas promueven el desarrollo de habilidades transversales como el pensamiento sistémico, la resolución de problemas, el trabajo con arquitecturas reales de automatización y la integración de múltiples tecnologías en un sistema único.

En conclusión, el presente proyecto demuestra un alto potencial como recurso educativo para su implementación en futuros cursos, proporcionando un modelo realista, modular y escalable que conecta eficazmente la teoría con la práctica en el contexto de la robótica colaborativa y la visión artificial.

6.3. Posibles mejoras y evolución del proyecto

Aunque el sistema desarrollado ha mostrado un funcionamiento robusto y cumple con su objetivo formativo, existen varias líneas de mejora y evolución para futuras versiones del proyecto.

- **Integración física del sistema:** El siguiente paso consistiría en migrar la estación virtual al laboratorio para trabajar con el robot colaborativo ABB GoFa disponible en el departamento y la cámara Cognex In-Sight 7600 recientemente adquirida. La implementación real permitiría abordar problemas de calibración, variaciones de iluminación o reflejos, ofreciendo al alumnado el ciclo completo: simulación-puesta en marcha-validación en planta.
- **Ampliación de la interfaz en MATLAB:** La interfaz hombre-máquina creada con App Designer puede crecer en varias direcciones: historial de pedidos, trazabilidad de piezas, gestión avanzada del inventario con alertas de stock, autenticación de usuarios y exportación de datos a CSV/Excel o servicios web REST. Estas mejoras transformarían la práctica del inventario automatizado en un pequeño sistema MES educativo.
- **Conexión a bases de datos y servicios en la nube:** Vincular el sistema OPC UA con una base de datos SQL o con plataformas IoT acercaría el proyecto a los flujos típicos de IIoT, permitiendo dashboards en tiempo real, analítica histórica o integración con ERP académicos.
- **Visión artificial basada en deep learning:** Si bien el módulo de visión artificial integrado en RobotStudio resulta adecuado para los ejercicios planteados, la incorporación de técnicas de aprendizaje profundo (como redes neuronales

convolucionales, CNN) ampliaría significativamente las capacidades del sistema. Estas permitirían, por ejemplo, detectar defectos superficiales complejos o realizar lectura de caracteres (OCR) sobre superficies curvadas.

Todas estas extensiones comparten un mismo propósito: acercar aún más el aula a los retos reales de la Industria 4.0.

BIBLIOGRAFÍA

A. Gasparetto, L. S. (s.f.). *A Brief History of Industrial Robotics in the 20th Century*. Recuperado el 20 de Abril de 2025, de SCIRP: https://www.scirp.org/html/2-2810274_90517.htm

ABB. (2023). *Release Notes - Wizard 1.6.*

ABB. (2024). *Product manual - Omnicore C30 Type A.*

ABB. (2024). *User manual – FlexPendant.*

ABB. (2025). Manual del producto CRB 15000. Obtenido de ABB.

Alaens. (s.f.). *File:Filtro bordes original sobel.jpg*. Recuperado el 22 de Abril de 2025, de WIKIMEDIA COMMONS: https://commons.wikimedia.org/wiki/File:Filtro_bordes_original_sobel.jpg

Andreu, A. (2022). Un nuevo brazo robótico será el encargado de traer de regreso a la Tierra las muestras de rocas recogidas en Marte para su estudio. *Business Insider*. Recuperado el 25 de Abril de 2025

Annan, J. (15 de Junio de 2024). *Single Shot MultiBox Detector (SSD) Explained by A.J*. Recuperado el 25 de Abril de 2025, de Medium: https://medium.com/@jesse419419/single-shot-multibox-detector-ssd-explained-by-a-j-dda10ba42a29

Autómatas, H. d. (3 de Junio de 2010). *GRECIA III: HERÓN DE ALEJANDRÍA*. Recuperado el 24 de Abril de 2025, de Blogspot: https://historiautomatas.blogspot.com/2010/06/grecia-iii-heron-de-alejandria.html

Barragán Piña, A. J. (10 de Junio de 2013). *MODBUS TCP*. Recuperado el 23 de Abril de 2025, de Universidad de Huelva: https://uhu.es/antonio.barragan/content/modbus-tcp

Barrientos, A., Peñín, L. F., Balaguer, C., & Aracil, R. (2007). *Fundamentos de robótica (2ª ed.).* McGraw-Hill.

BCNVision. (12 de Diciembre de 2013). *Ópticas para visión artificial*. Recuperado el 18 de Abril de 2025, de BCNVision: https://bcnvision.es/blog-vision-artificial/opticas-para-vision-artificial/

BCNVision. (11 de Abril de 2017). *Sistemas de iluminación para aplicaciones de visión artificial*. Recuperado el 18 de Abril de 2025, de BCNVision: https://bcnvision.es/blog-vision-artificial/iluminacion-vision-artificial2/

Bihl+Wiedemann. (s.f.). *BW1979. Cable perfilado ASi, amarillo*. Recuperado el 18 de Abril de 2025, de Bihl+Wiedemann: https://www.bihl-wiedemann.de/es/productos/accesorios/cablestarjetas-chip/resumen-de-productos-cables/tarjetas-de-memoria/l/BW1979

Blázquez Morales, L. F. (s.f.). *Herón de Alejandría*. Recuperado el 21 de Abril de 2025, de Oficina Española de Patentes y Marcas: http://historico.oepm.es/museovirtual/galerias_tematicas.php?tipo=INVENTOR&xml=Her%C3%B3n%20de%20Alejandr%C3%ADa.xml

Boston Dynamics. (s.f.). *Boston Dynamics Expands Global Sales of Spot® Robot*. Recuperado el 12 de Abril de 2025, de Boston Dynamics: https://bostondynamics.com/news/boston-dynamics-expands-global-sales-of-spot-robot/

Brunete, A., San Segundo, P., & Herrero, R. (2024). *Introducción a la Automatización Industrial.* Madrid.

Cognex Corporation. (s.f.). *Sistema de visión In-Sight D900*. Recuperado el 16 de Abril de 2025, de Cognex: https://www.cognex.com/es-mx/products/machine-vision/2d-machine-vision-systems/in-sight-d900

Corke, P., Jachimczyk, W., & Pillat, R. (2023). *Robotics, vision and control: Fundamental algorithms in MATLAB (3ª ed.).* Springer.

Datademia. (s.f.). *Computer Vision: ¿Cómo las Máquinas Aprendieron a Ver?* Recuperado el 21 de Abril de 2025, de Datademia: https://datademia.es/blog/computer-vision-como-las-maquinas-aprendieron-a-ver

Duarte Villaseñor, M. M., & Chang Fernández, L. (s.f.). *Clasificación de objetos en imágenes usando SIFT.* Recuperado el 23 de Abril de 2025, de https://ccc.inaoep.mx/~esucar/Clases-mgp/Proyectos/chang-duarte.pdf

FANUC. (s.f.). *Robot colaborativo CR-35iB*. Recuperado el 23 de Abril de 2025, de AeroEXPO: https://www.aeroexpo.online/es/prod/fanuc-france/product-183679-85335.html

Fraile Marinero, J. C. (Octubre de 2023). *Campus Virtual UVa, Dpto. Ingeniería de Sistemas y Automática, Introducción a la robótica*. Recuperado el 15 de Abril de 2025

Group, E. T. (s.f.). *Tecnología EtherCAT*. Recuperado el 21 de Abril de 2025, de EtherCAT Technology Group: https://www.ethercat.org/es/technology.html

Guerrero, V. (2013). *Comunicaciones industriales.* Marcombo.

Horn, B. K. (1986). *Robot Vision*. Obtenido de MIT.

IBM. (s.f.). *Visión Artificial*. Recuperado el 20 de Abril de 2025, de IBM: https://www.ibm.com/es-es/topics/computer-vision

Instruments, N. (2 de Abril de 2025). *Descripción general del protocolo de la red de controladores de área (CAN)*. Recuperado el 25 de Abril de 2025, de National Instruments: https://www.ni.com/es/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/controller-area-network--can--overview.html

Jizai. (s.f.). *AI Robot Services for Companies and Research Institutions*. Recuperado el 25 de Abril de 2025, de Jizai: https://jizai.ai/PartnerRobotEN

Leishen Intelligent System Co. (s.f.). *Productos de mapeo 3D LS LiDAR*. Recuperado el 22 de Abril de 2025, de LSLIDAR: https://www.leishenlidar.com/es/lidar-sensor-productos/mapeo-lidar-2/

Martín González, J. (2023). *Inspección visual utilizando Integrated Vision de ABB.* Alicante.

MathWorks. (s.f.). *App Designer*. Recuperado el 22 de Abril de 2025, de MathWorks: https://es.mathworks.com/help/matlab/app-designer.html

MathWorks. (s.f.). *MATLAB*. Recuperado el 22 de Abril de 2025, de MathWorks: https://es.mathworks.com/products/matlab.html

MathWorks. (s.f.). *SLAM (localización y mapeo simultáneos)*. Recuperado el 18 de Abril de 2025, de MathWorks: https://es.mathworks.com/discovery/slam.html

MiR Robots. (14 de Abril de 2022). *MiR Robots presenta: MiR600, para automatizar entornos industriales exigentes*. Recuperado el 22 de Abril de 2025, de Logística Profesional: https://www.logisticaprofesional.com/texto-diario/mostrar/3550096/mir-robots-presenta-mir600-automatizar-entornos-industriales-exigentes

Park, B.-W., Park, S.-J., & Kang, F.-S. (Enero de 2022). *A Novel Communication Method Using PWM and Capture Function of DSP for Parallel Controlled Power Electronics Systems*. Recuperado el 23 de Abril de 2025, de ResearchGate: https://www.researchgate.net/figure/MODBUS-RTU-and-ASCII-frame_fig1_362987592

Pérez Cisneros, M. A., Valdemar Cuevas Jiménez, E., & Zaldívar Navarro, D. (2014). *Fundamentos de robótica y mecatrónica con MATLAB y Simulink.* RA-MA.

Pérez Olguin, I. (s.f.). *Precisión y exactitud*. Recuperado el 25 de Abril de 2025, de ResearchGate: https://www.researchgate.net/figure/Figura-2-Precision-y-exactitud_fig2_289674541

Piera Moreno, E. (2021). *Sistema de inspección visual utilizando Integrated vision de ABB.* Alicante.

PngTree. (s.f.). *Rayo De Longitud De Onda Del Espectro De Luz Visible*. Recuperado el 26 de Abril de 2025, de PngTree: https://es.pngtree.com/freepng/visible-light-spectrum-wavelength-ray_8654422.html

PowerBots. (22 de Noviembre de 2023). *Las 5 generaciones de la robótica*. Recuperado el 26 de Abril de 2025, de PowerBots: https://powerbots.es/4903/las-5-generaciones-de-la-robotica/

Pozas Mata, E. (2022). *Simulación de un Robot Colaborativo YuMi (ABB) en entorno RobotStudio comandado desde MATLAB mediante protocolo OPC UA para tocar un Xilófono.* Valladolid.

Quellegamos. (21 de Junio de 2017). *La expendedora de Herón*. Recuperado el 22 de Abril de 2025, de https://quellegamos.com/la-expendedora-de-heron

Riccio, G. (28 de Julio de 2024). *Los robots de Leonardo: los increíbles autómatas del genio del Renacimiento*. Recuperado el 24 de Abril de 2025, de Futuro Prossimo: https://es.futuroprossimo.it/2024/07/i-robot-di-leonardo-gli-incredibili-automi-del-genio-rinascimentale/

Robles Postigo, A. (2024). *Estación de paletizado en RobotStudio (ABB) con HMI en TIA Portal, controlable por voz y visión artificial desde Python.* Valladolid.

Robotics, A. (s.f.). *Cobot YuMi® – IRB 14000*. Recuperado el 24 de Abril de 2025, de Direct Industry: https://www.directindustry.es/prod/abb-robotics/product-30265-1635187.html

Robots, R. d. (27 de Mayo de 2023). *PILZ ofrece a los robots AGV protección a bordo*. Recuperado el 21 de Abril de 2025, de Revista de Robots: https://revistaderobots.com/robots-y-robotica/pilz-ofrece-a-los-robots-agv-proteccion-a-bordo/

Robots, U. (s.f.). *UNIVERSAL ROBOTS UR5*. Recuperado el 25 de Abril de 2025, de MACHINETOOLS.COM: https://www.machinetools.com/es/models/universal-robots-ur5

Rufenacht, M. (19 de Mayo de 2020). *The evolution of document capture*. Recuperado el 30 de Abril de 2025, de parashift: https://parashift.io/the-evolution-of-document-capture/

SafetyCulture. (16 de Abril de 2025). *Evaluación de riesgos*. Recuperado el 19 de Abril de 2025, de SafetyCulture: https://safetyculture.com/es/temas/evaluacion-de-riesgos/matriz-de-riesgo/

Sancho García, R. (2021). *Diseño con Autodesk Inventor de una célula de trabajo robótica para realizar montajes multitarea apoyados con visión artificial, coordinado mediante protocolo de comunicación OPC UA entre RobotStudio-MATLAB.* Valladolid.

SICMA21. (22 de abril de 2021). *Redes de comunicación industrial: todo lo que necesitas saber*. Recuperado el 20 de Abril de 2025, de SICMA21: https://www.sicma21.com/que-son-las-redes-de-comunicacion-industrial/

Sutori. (s.f.). *Los primeros robots autómatas*. Recuperado el 26 de Abril de 2025, de Sutori: https://www.sutori.com/es/historia/los-primeros-robots-automatas--2Q7rQE8pzBHArS5hmQcbpiEV

Universal Robots. (s.f.). *Itinerario básico de E-Series*. Recuperado el 16 de Abril de 2025, de Universal Robots Academy: https://academy.universal-robots.com/es/formacion-en-linea-gratuita/formacion-en-linea-de-e-series/itinerario-basico-de-e-series/

Velasco, S. (15 de Noviembre de 2016). *The 800-year-old Cutaway Graphics of Ismail Al-Jazari*. Recuperado el 27 de Abril de 2025, de 5W BLOG: https://5wgraphicsblog.com/2016/11/15/the-800-year-old-cutaway-graphics-of-ismail-al-jazari/

Ventosa Ribas, M. (2021). *PROPUESTA DE NUEVOS CONTENIDOS DE ESTIMULACIÓN COGNITIVA PARA LAS AGNOSIAS VISUALES.* Recuperado el 2 de Mayo de 2025

viamed. (s.f.). *Cirugía Robótica Da Vinci: Qué es, principales beneficios para el paciente y el cirujano*. Recuperado el 6 de Mayo de 2025, de viamed: https://www.viamedsalud.com/tratamientos/urologia/cirugia-robotica-da-vinci-que-es-principales-beneficios-para-el-paciente-y-el-cirujano/

Vigo, U. d. (s.f.). *Equipos para la automatización industrial*. Recuperado el 28 de Abril de 2025, de TV. Universidad de Vigo: https://tv.uvigo.es/uploads/material/Video/1451/ISAD_Tema3_4.pdf

Yoigo. (6 de Mayo de 2025). *Al teléfono del futuro no tendremos que tocarlo: nuestra próxima interfaz son gestos en el aire*. Recuperado el 20 de Abril de 2025, de Bloygo: https://bloygo.yoigo.com/al-telefono-del-futuro-no-tendremos-que-tocarlo-nuestra-proxima-interfaz-son-gestos-en-el-aire/

Zaragoza, U. d. (s.f.). *Autómatas en la historia*. Recuperado el 2 de Mayo de 2025, de http://automata.cps.unizar.es/Historia/Webs/automatas_en_la_historia.htm