Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

15-6-2021

UNIVERSIDAD DE VALLADOLID
ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

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

Autor:

- **Sancho García, Rodrigo**

Tutor:

- **Herreros López, Alberto**

Departamento de Ingeniería de Sistemas y Automática

[Página en Blanco]

Alberto Herreros López, profesor del Departamento de Ingeniería de Sistemas y Automática de la Universidad de Valladolid, CERTIFICA:

Que alumno de la Escuela de Ingenierías Industriales, Sede Paseo del Cauce, de Valladolid, titulación Grado en Ingeniería en Electrónica Industrial y Automática, D. Rodrigo Sancho García, ha realizado bajo su dirección el Proyecto Fin de Grado titulado "*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, 15 de junio de 2021

Tutor del Proyecto

Fdo.: Herreros López, Alberto

Autor del Proyecto

Fdo.: Sancho García, Rodrigo

Reunido el Tribunal designado por el Departamento de Ingeniería de Sistemas y Automática para la evaluación de Proyectos Fin de Carrera, y después de estudiar la memoria y atender a la defensa del trabajo "*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*", presentado por el alumno D. RODRIGO SANCHO GARCÍA, decidió otorgarle la calificación de _____.

Valladolid, de de 2021

El Presidente El Secretario

Fdo.: Fdo.:

El Vocal

Fdo.:

Agradecimientos

Quisiera comenzar el presente proyecto agradeciendo a mi tutor D. Alberto Herreros López, la confianza puesta en mí, para poder desarrollar este trabajo con total libertad en la toma de decisiones. Además, por la ayuda aportada en todo momento, ante cualquier duda que pudiera tener o cualquier necesidad de software que pudiera surgir.

También quiero aprovechar este momento, para agradecer a mi familia todo el apoyo que siempre me ha aportado ante cualquier adversidad que haya podido encontrarme en esta etapa universitaria, que acaba tras 4 años llenos de emociones, con este trabajo final.

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

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

Autor:

Sancho García, Rodrigo

Tutor:

Herreros López, Alberto

Departamento de Ingeniería de Sistemas y Automática

Valladolid, junio, 2021.

Resumen

En este Trabajo de Fin de Grado se desarrolla una célula de trabajo robótica, en la que se trabaja con dos robots de la corporación multinacional tecnológica ABB Robotics. Estos son el modelo IRB120 (idóneo para aplicaciones de 'Pick & Place') y el modelo IRB14000 (YuMi), un robot colaborativo que cuenta con 2 brazos robóticos que pueden actuar de manera independiente o sincronizada.

Durante el desarrollo de la célula se ha usado el Autodesk Inventor para diseñar todos los elementos que la componen. Para la simulación de los movimientos, se han utilizado los programas RobotStudio y Matlab. Debido al sincronismo que debe existir entre ambos programas, se ha utilizado adicionalmente un protocolo de comunicación industrial como es el OPC UA.

Con todo este trabajo se ha pretendido realizar 2 tareas, pero la célula creada da lugar a una infinidad de nuevas posibilidades. La primera tarea ha sido el ensamblaje automático por piezas de un coche deportivo de juguete, el cual su carrocería se basa en el modelo R8 de la marca de coches Audi. Por otro lado, en la segunda tarea, se ha incluido un algoritmo de visión artificial, gracias al cual se puede detectar el tamaño, la posición o el color de una serie de cubos, y realizar unas acciones en función de estos parámetros.

Finalmente, para realizar una ejecución dinámica del software propuesto, se ha desarrollado una interfaz de usuario, con la cual se puede controlar toda la simulación, por medio de una serie de botones y ventanas.

Palabras Clave

Robot IRB120, Robot IRB14000 YuMi, MATLAB, Visión Artificial, Autodesk Inventor

Abstract

In this Final Degree Project, a robotic work cell is developed, working with two robots from the multinational technology corporation ABB Robotics. These are the IRB120 model (suitable for 'Pick & Place' applications) and the IRB14000 (YuMi) model, a collaborative robot with 2 robotic arms that can work independently or synchronously.

During this development, Autodesk Inventor was utilized to design all the components of the cell. Besides, RobotStudio and MATLAB were used for the robotic movement's simulation. Due to the synchronism that must exist between these two software, an industrial communication protocol such as OPC UA has also been used.

The goal of this project is to carry out 2 tasks, but the created cell gives rise to an infinite number of new possibilities. The first task is the automatic assembly of a sport car toy, the body of which is based on the R8 model of the Audi car brand. On the other hand, in the second task, an artificial vision algorithm has been included with the purpose of detecting the size, position or colour of a series of cubes and then perform an action according to these parameters.

Finally, in order to carry out a dynamic execution of the proposed software, a user interface has been developed to be able to control the whole simulation by means of a series of buttons and windows.

Keywords

IRB120 Robot, IRB14000 (YuMi) Robot, MATLAB, Computer Vision, Autodesk Inventor

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ABREVIATURAS

Abreviatura	Significado
TFG	Trabajo de Final de Grado
USA	United States of America
EEUU	Estados Unidos
ASEA	Compañía Eléctrica General Sueca (Traducido al español)
SCARA	Selective Compliance Assembly Robot Arm
PUMA	Programmable Universal Machine for Assembly
ASIMO	Advanced Step in Innovative Mobility
TCP/IP	Transmission Control Protocol/Internet Protocol
ABB	Asea Brown Boveri
OPC	Open Protocol Communication
EII	Escuela de Ingenierías Industriales
UVA	Universidad de Valladolid
ROI	Region of Interest
OPC DA	OPC (Data Access)
OPC HDA	OPC (Historical Data Access)
OPC UA	OPC (Unified Architecture)
GUI	Graphical User Interface
ERP	Enterprise Resource Planning
M2M	Machine to Machine
MES	Manufacturing Execution System
OLE	Object Linking and Embedding
UNE	Una Norma Española
ISO	International Organization for Standardization
IFA-BGIA	Institute for Occupational Safety and Health of the German Social Accident Insurance
CEMA	Conveyor Equipment Manufacturers Association

Nota:

Si las siglas provienen de un nombre que no es ni inglés ni español, se han traducido al español, para la mejor comprensión del lector.

CAPÍTULO I. INTRODUCCIÓN

A lo largo del presente informe, se va a desarrollar y explicar la creación de una célula robótica, en la que se ha pretendido demostrar algunas de las posibilidades que permite el campo de la robótica.

Con la evolución de la sociedad hacia un entorno más tecnológico, los robots y las máquinas automáticas que facilitan el día a día de las personas, son cada vez más frecuentes, por lo que es un campo con infinidad de posibilidades aún por descubrir.

Además, tienen una gran utilidad en los entornos industriales, lugares en los que la mayoría de las tareas son realizadas por máquinas robóticas, ya sea por su capacidad para repetir tareas sistemáticas o por su precisión, mucho más alta que la alcanzada por cualquier ser humano.

CAPÍTULO 1.1. MOTIVACIÓN

La robótica es un campo fascinante de estudio, debido a las diversas ramas que incluye de la ingeniería como se muestra en la siguiente ilustración.

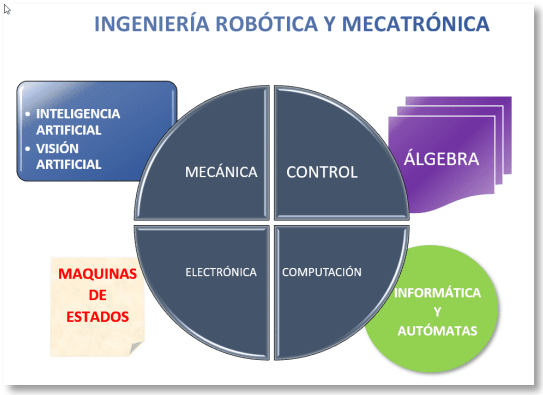


Figura 1. *Campos de conocimiento abarcados por la robótica [Gráfico].* [1]

Este factor ha sido determinante para la selección de este proyecto (TFG), ya que se considera que uno de los objetivos de este proyecto es demostrar las capacidades adquiridas en el grado y con esta temática se engloban varios de los campos abarcados en este tipo de titulaciones.

Otro aspecto por el que se seleccionó esta temática, es por la afición del autor al mundo de la robótica. Ya que dicha afición fue la que en primera instancia le hizo seleccionar el Grado de Ingeniería Electrónica Industrial y Automática en el que pasaría los siguientes 4 años de su vida aprendiendo una gran variedad de aptitudes y conocimientos. Esta etapa no podría terminar de otra manera que con un proyecto en el ámbito robótico.

CAPÍTULO 1.2. OBJETIVOS

Ante la diversidad de proyectos que pueden ser realizados con esta temática, surge la necesidad de establecer unos objetivos generales, para ir delimitando el trabajo que se va a realizar.

El objetivo principal de este proyecto es educativo, ya que se considera esencial seguir desarrollando herramientas para que más gente pueda acceder a este campo de la ingeniería.

Por tanto, se definen los siguientes objetivos, como idea general del proyecto.

- Diseño de una célula robótica con un programa de diseño de objetos 3D
- Control de robots para realizar una serie de acciones
- Programación de una interfaz de usuario para el manejo dinámico de la simulación

En posteriores capítulos (*capítulo 3.1. Análisis de Requisitos*) se comentarán cuáles son los requisitos más específicos a cumplir, para cada objetivo.

CAPÍTULO 1.3. METODOLOGÍA

Durante la lectura del informe el lector se encontrará en primer lugar una introducción teórica del tema seleccionado para el proyecto y se analizaran brevemente los anteriores trabajos realizados en este campo en la Universidad de Valladolid.

Posteriormente se definirá con mayor lujo de detalles, el proyecto que se ha realizado y se propondrán una serie de soluciones (explicando los puntos positivos y los negativos) entre las que se incluye la solución escogida.

Una vez ha quedado claro que se pretende hacer, se explicarán brevemente los 4 software utilizados para la realización de este proyecto y posteriormente se mostrara las fases de desarrollo necesarias para la creación de este. Entre ellas se encuentran el diseño y desarrollo de todos los componentes 3D, la programación de los robots y del software (simulación, interfaz…) y finalmente los resultados.

Para completar el informe del proyecto, se realizará un estudio económico del mismo, y se comentaran las principales conclusiones extraídas durante la realización, así como los trabajos futuros que se han desarrollado si se quisiese seguir con este trabajo.

Tanto las referencias como la bibliografía y los anexos incluidos se encuentran en la parte final del informe.

CAPÍTULO 1.4. CONVENCIONES

Para ayudar al lector durante el transcurso de este informe en este capítulo se definen las normativas de marcado que se han seguido y que se encuentran a lo largo de todo el texto.

- Las palabras extranjeras (generalmente idioma inglés) se remarcarán en cursiva.
- Las citas externas a la obra se encontrarán entrecomilladas.
- Posterior al índice general se pueden encontrar diversos índices, para las figuras, para las tablas y finalmente un índice con las siglas utilizadas a lo largo del texto, colocadas en función del orden de aparición en este.
- El texto se desarrolla con una tipografía '*Franklin Gothic Book*' (12).
- Los títulos y el texto de las figuras constan de una tipografía '*Times New Roman*'. Además, este segundo se remarcará con letra cursiva.

CAPÍTULO II. MARCO TEÓRICO: EL MUNDO DE LA ROBÓTICA

El origen de la robótica, a pesar de la creencia popular de tratarse de un sector muy novedoso y avanzado, se puede encontrar retrocediendo varios milenios en el tiempo en lugares como China o la Antigua Grecia, cuando comenzaron los primeros intentos de representar a personas por medio de máquinas, denominándose dichas máquinas, autómatas.

Las primeras descripciones que se tienen de los autómatas aparecen en el libro 'Lie Zi' (una de las tres obras fundamentales del taoísmo filosófico), redactado por 'Lie Yukou' (véase *Figura 2.A*), en el cual se describe como 'Yan Shi' presentó al rey, una figura de tamaño y formas humanas de su obra mecánica.

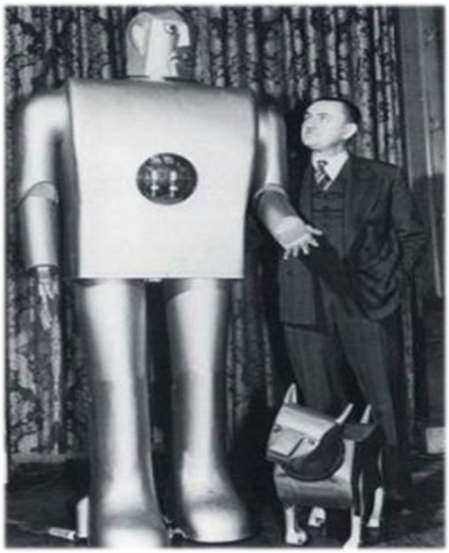
Por otro lado, siglos después, el matemático, físico e ingeniero Herón de Alejandría [10 d.C.-70 d.C.] (véase *Figura 2.B*), se tiene constancia de que diseño más de 100 máquinas y autómatas

A) ***B)***

Figura 2. A) *Lie Yukou (Autor de 'Lie Zi') [Ilustración].* [2]
B) *Herón de Alejandría [Ilustración].* [3]

Posteriormente, seguirían creando humanoides y autómatas, personajes tan reconocidos como Leonardo Da Vinci, Alberto Magno o al-Jazari.

Pero a pesar de lo expuesto anteriormente**,** la etimología del término o el concepto de la palabra *'robot'* no surgiría hasta el 1920, momento en el cual dicho término fue acuñado por un dramaturgo checo llamado Karel Capek en la obra teatral que había creado llamada R. U. R. (*Rossum´s Universal Robots*). La palabra se originó del término '*robota*' que en checo significa trabajo forzado o servidumbre.



No se puede realizar esta introducción histórica de la robótica sin mencionar a *Elektro* (véase *Figura 3*) que es considerado por muchos autores el primer robot de la historia. Este robot humanoide de más de 2 metros, 120 kg de peso, era capaz de realizar 26 movimientos diferentes y fue diseñado por el ingeniero de *Westin House Electric Corporation*, Joseph Barnett.

Figura 3. *Elektro, el primer robot de la historia [Fotografía].* [4]

A principios de los años 40 Isaac Asimov, definió las "Tres Leyes de la Robótica", para sus novelas de ciencia ficción. A pesar de este carácter ficticio, en la actualidad estas leyes son conocidas por toda la comunidad científica interesada en la robótica.

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 cumplir las órdenes dadas por los seres humanos, a excepción de aquellas que entren 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 con la segunda ley.

En los años 50, se origina la Robótica Industrial, de la mano de George Charles Devol, al crear el primer robot industrial (primera máquina programable), denominado '*Unimate*' en 1954. Este junto a Joseph F. Engelberger fundaría '*Unimation*', empresa dedicada íntegramente a desarrollar los primeros robots industriales.

A) ***B)***

Figura 4. ***A)*** *Joseph F. Engelberger con uno de sus robots industriales [Fotografía].* [5]
B) *Unimate, el primer robot industrial [Fotografía].* [6]

Pero no sería hasta los años 60, cuando se implementaría el primer robot en la industria, más concretamente en 'General Motors' en la localidad de Trenton (Nueva Jersey – USA), este se utilizaría para la carga y descarga de piezas en una máquina de fundición.

En los comienzos de los años 70 se empezó a integrar la robótica en la munición de precisión y en las armas inteligentes.

En EEUU se empezaron a desarrollar pequeños brazos robóticos con accionamiento eléctrico (1971) y, a su vez en Japón, se impulsó el desarrollo de los robots humanoides con los proyectos WABOT (1967-1972), llevados a cabo por la 'Universidad de Waseda' (Tokyo)

En los años posteriores, cada vez el desarrollo de robot es mayor y, por tanto, surgen prototipos en muchas empresas y países. La empresa sueca ASEA construye el primer robot del mundo con accionamiento totalmente eléctrico (1973), surge el primer robot SCARA en 1978, y en este mismo año la empresa '*Unimation*' construye el robot industrial PUMA, las primeras cirugías ayudadas con robot se desarrollan satisfactoriamente en 1994, en el año 2000 Honda muestra el proyecto de humanoide ASIMO, Mars rovers '*Spirit*' y '*Opportunity*' aterrizan en Marte en 2003…y de esta manera surge una lista interminable de nuevos proyectos que se originan hasta la actualidad.

Cabe destacar que con esta evolución, todos los prototipos se han ido mejorando y desarrollando hasta llegar al momento actual, en el que cada vez la robótica es más precisa y fiable.

Una vez visto el origen y desarrollo de esta rama tecnológica, se pueden distinguir distintos sectores o familias de robots:

- **Robot industrial:** Poseen brazos mecánicos o poliarticulados con diferentes ejes los cuales pueden ser móviles o fijos.
- **Robot de servicios:** Son los robots sociales y pueden ser del tipo humanoide, zoomorfos o zoomórficos y móviles. Están destinados a sectores como la salud, el ocio o la defensa militar entre otros.
- **Nanorrobótica:** Son los robots que, por sus reducidas dimensiones, han sido creados para realizar funciones científicas.

La familia de robots utilizada en este proyecto son los pertenecientes a la robótica industrial.

Dicho campo hace referencia a los robots destinados a la automatización de los procesos de fabricación dentro del sector industrial. Estos son capaces de realizar trabajos de todo tipo de complejidad y son muy adecuados para realizar tareas repetitivas, de esfuerzo y precisión, o tareas que supongan algún peligro a los humanos si las realizasen.

Entre las tecnologías que más sinergia tienen con este sector se encuentran la Inteligencia Artificial y la Visión Artificial, siendo muy importante a su vez la comunicación entre robots o con dispositivos externos.

Se pueden distinguir dos modalidades entre los robots industriales:

- Colaborativos (Cobots)

  Este tipo de robots suelen ser de dimensiones reducidas, y están diseñados con el objetivo de poder realizar tareas de apoyo en un entorno rodeado de humanos.

  Este factor origina que haya que tener muy en cuenta los factores de seguridad en el entorno laboral, ya que un robot puede ser mortal hacia un humano, si no se han tomado las medidas adecuadas.

  Por ello, normalmente estos robots cuentan con sensores que detectan la presencia humana en un campo de 360º, además de reguladores de fuerza para evitar posibles daños en caso de atrapamiento.

- No colaborativos

  Esta segunda modalidad, era la más habitual en las industrias, ya que existían previamente a los robots colaborativos. En este caso el robot tiene un tamaño considerable debido a la necesidad de realizar ciertas acciones pesadas en la cadena de producción. Además, esta característica supone la necesidad de un espacio de trabajo mayor.

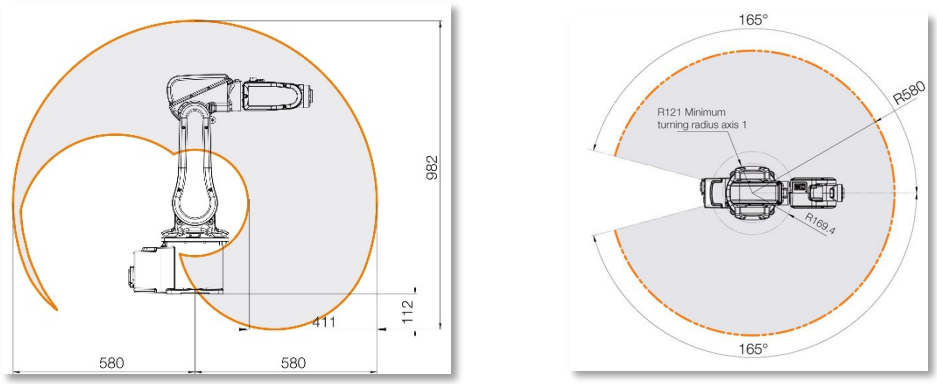
  No están pensados para trabajar colaborando con los humanos, por lo que las medidas de seguridad deben ser muy altas, para evitar cualquier tipo de accidente.

Una vez introducidas las diferentes modalidades que se pueden encontrar en el sector industrial de la robótica, ya se pueden definir los dos robots con los que se trabajara en el presente proyecto.

Robot IRB120

Este robot industrial es el que cuenta con un menor tamaño dentro de la gama de robots proporcionada por ABB (tiene un peso de 25 kg). Debido a su pequeño tamaño las cargas que puede soportar son pequeñas, siendo de 3 kg (4 kg en posición vertical de la muñeca).

Su morfología es la de un brazo robótico de 6 articulaciones rotacionales, con la que se pueden alcanzar una gran variedad de puntos en su espacio de trabajo (véase *Figura 5*). Este tamaño y su gran maniobrabilidad le dota de características idóneas para ser un robot de '*Pick & Place*'.



A) ***B)***

Figura 5. ***A)*** *Área de trabajo (Vista Lateral) IRB120 [Ilustración].* [7]
B) *Área de trabajo (Vista Superior) IRB120 [Ilustración].* [8]

En cuanto a las especificaciones de este modelo se muestran a continuación las siguientes tablas, las cuales se han obtenido del *datasheet* que pone a la disposición el fabricante. [9]

Specification

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

* 4 with vertical wrist

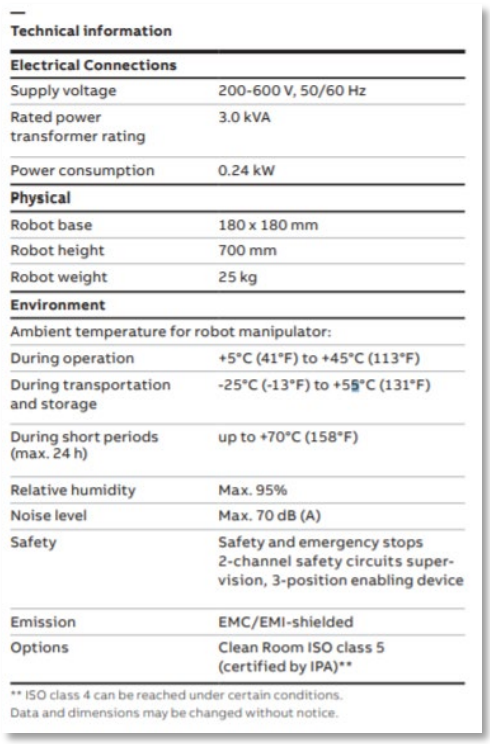
A)

Movement

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

B)

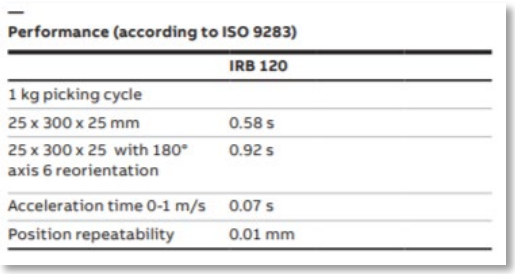
Tabla 1. ***A)*** *Especificaciones generales IRB120.* [9]
B) *Rangos movimiento de las 6 articulaciones IRB120.* [9]



Technical information

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

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



Performance (according to ISO 9283)

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

C) ***D)***

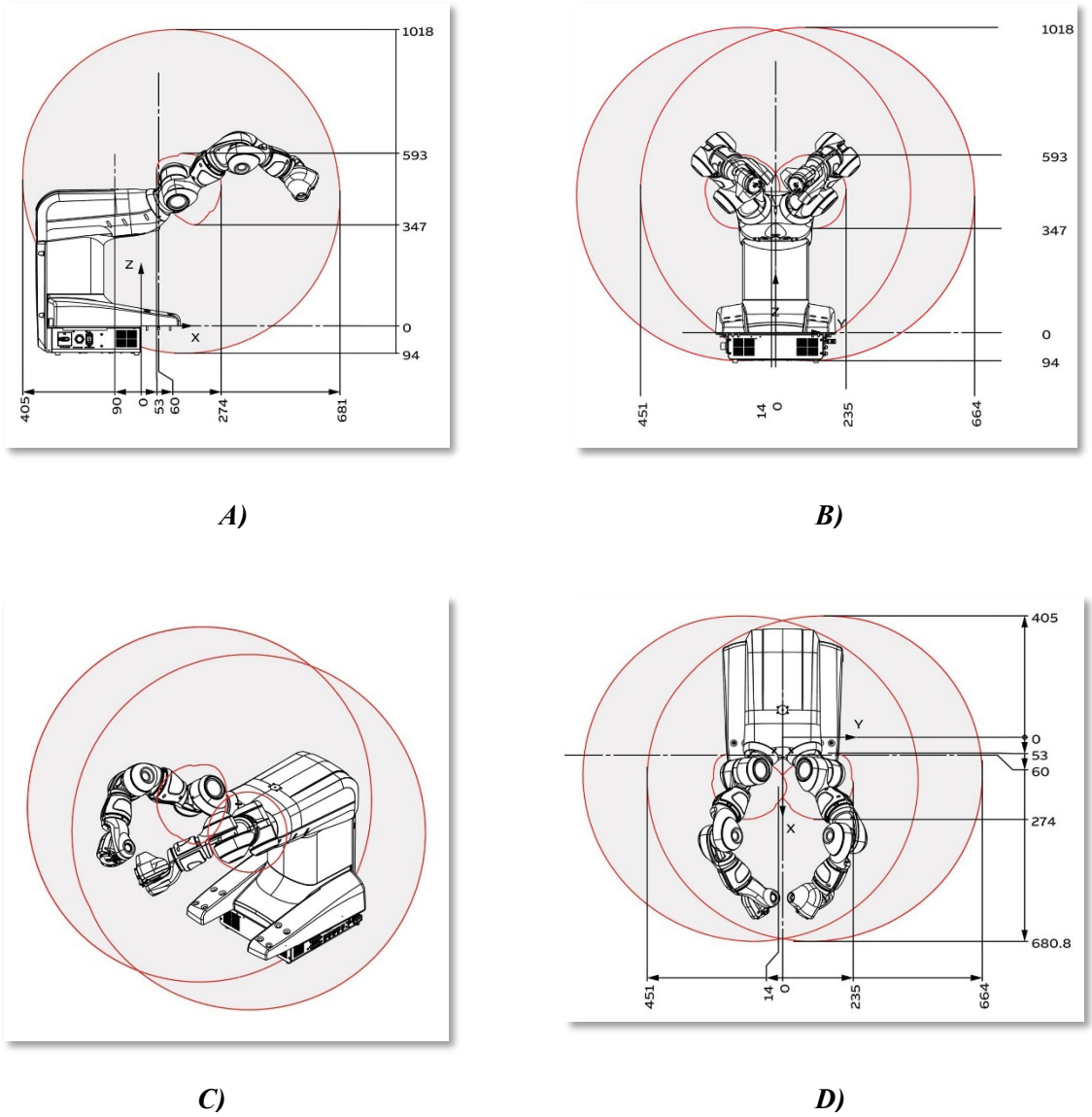
Tabla 1. C) *Información técnica (eléctrica, física y entorno) IRB120.* [9]
D) *Datos del rendimiento IRB120.* [9]

Robot IRB14000 (YuMi)

Este modelo desarrollado por ABB Robotics, fue el primer robot verdaderamente colaborativo que constaba de 2 brazos robóticos, y diseñado con el objetivo de realizar un trabajo conjunto entre robots y humanos.

Cada brazo consta de 7 ejes, lo que supone que se puedan realizar una gran variedad de movimientos, para alcanzar los puntos deseados. A la característica de poder trabajar colaborativamente con humanos, se le une la posibilidad de sincronizar sus brazos, lo que posibilita realizar tareas como las de ensamblaje de piezas pequeñas, en líneas en las que robots y personas trabajan mano a mano en las mismas tareas.

El área de trabajo que puede abarcar se muestra en las siguientes ilustraciones (*Figura 6*) recogidas del *datasheet* del fabricante.



A) ***B)***

C) ***D)***

Figura 6. A) *Área de trabajo (Vista Lateral) IRB14000 [Ilustración].* [10]
B) *Área de trabajo (Vista Frontal) IRB14000 [Ilustración].* [10]
C) *Área de trabajo (Vista Superior) IRB14000 [Ilustración].* [10]
D) *Área de trabajo (Vista Isométrica) IRB14000 [Ilustración].* [10]

Tras mostrar el volumen en el espacio que pueden abarcar ambos brazos, el siguiente aspecto fundamental para definir el robot, son sus principales parámetros y especificaciones. Para ello al igual que en el otro modelo visitando el *datasheet* se pueden encontrar las siguientes tablas (véase *Tabla 2*):

Movement

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

A)

Performance (according to ISO 9283)

Max TCP Velocity	1.5 m/s
Max TCP Acceleration (normal control motion @nominal load)	39.1 m/s²
Max TCP Acceleration (e-stop @nominal load)	85.4 m/s²
Acceleration time 0–1m/s	0.12 s
Pose repeatability	0.02 mm
0.5 kg picking cycle 25 × 300 × 25 mm	0.86 s

B)

Specification

Robot version	Reach (mm)	Payload (g)	Armload
IRB 14000-0.5/0.5	559	500	No armloads
Number of axes	14		
Protection	Std: IP30 and Clean Room		
Mounting	Table		
Controller	Integrated IRC5		
Customer power supply	24V/1A supply		
Customer signals	4 signals (for IO, Fieldbus, or Ethernet)		
Integrated air supply	1 per Arm on tool Flange (4 Bar)		
Functional safety	SafeMove Collaborative included All safety functions certified to Category B, PL b		

C)

Tabla 2. ***A)*** *Rangos movimiento de las 7 articulaciones IRB14000.* [10]
B) *Datos del rendimiento IRB14000.* [10]
C) *Especificaciones generales IRB14000.* [10]

Realizando un pequeño análisis sobre ambos modelos se puede apreciar rápidamente que no van a poder ser utilizados para realizar operaciones pesadas, lo que en el presente proyecto no es un problema, ya que todas las piezas utilizadas, tendrán un tamaño relativamente pequeño y no serán de materiales muy pesados.

Tras introducir al lector en el mundo de la robótica con sus orígenes, algunas de las modalidades existentes y definir las principales características de los dos robots que se van a utilizar, es el momento de mostrar cuales han sido los trabajos previos en esta materia. Para ello en el siguiente capítulo se desarrollan brevemente algunos de los últimos trabajos originados en la Universidad de Valladolid.

CAPÍTULO 2.1. ESTADO DEL ARTE

Una vez se ha introducir el marco teórico en el que se ha desarrollado el presente proyecto, en este capítulo se ha pretendido mostrar los trabajos realizados previamente en esta materia en la Universidad de Valladolid.

En 2015, Gonzalo Muinelo Garrido [11] realizó su proyecto final de grado que consistía en la simulación de una célula robotizada, para el tratamiento de una pieza de aluminio. El objetivo era la programación del robot, para poder gestionar las piezas, ya que tenían que pasar por 3 procesos/máquinas distintas.

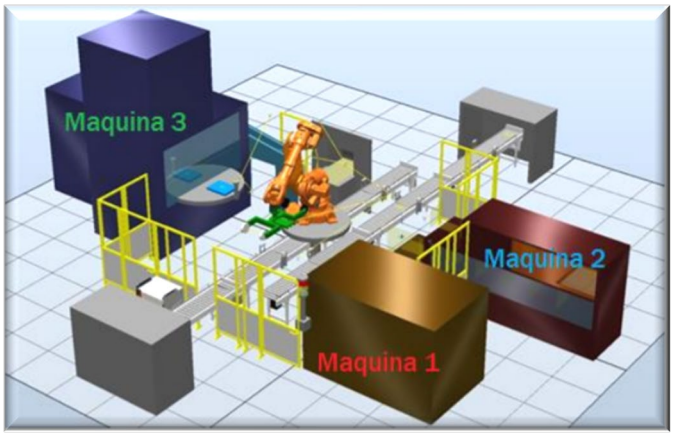
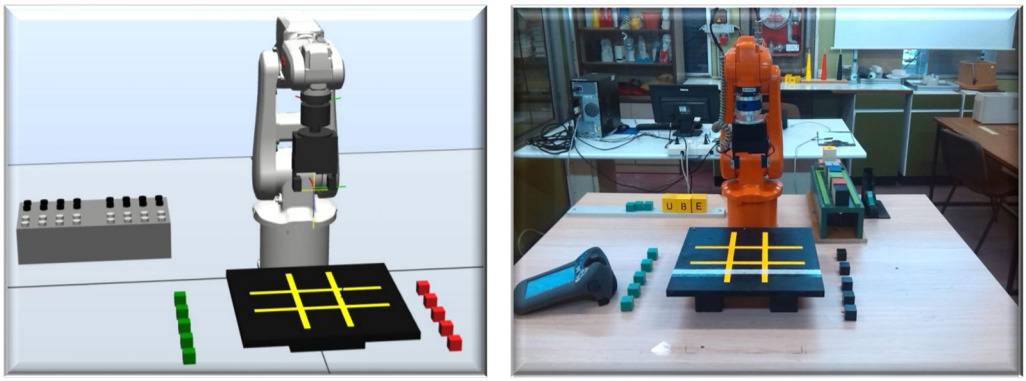


Figura 7. *Célula de trabajo TFG de G.Muinelo [Ilustración].* [11]

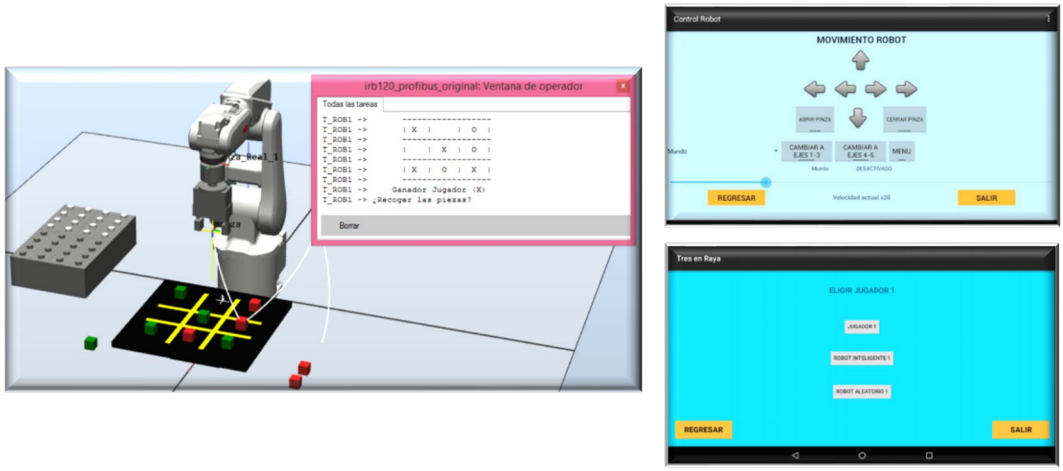
En ese año (2015), se realizaba por parte de Juan Antonio Ávila Herrero [12] como trabajo de fin de grado, el diseño de otra célula robótica pero que, en este caso, estaría enfocada con fines educativos, ya que se diseñaban una serie de componentes como una pinza, una botonera, una caja de rotuladores y dos mesas, y con estos, se proponían una serie de prácticas, para aprender a manejar el software RobotStudio o para manejar la estación real.



A) ***B)***

Figura 8. A) *Célula de trabajo TFG de J.A. Ávila (Simulada) [Ilustración].* [12]
B) *Célula de trabajo TFG de J.A. Ávila (Real) [Ilustración].* [12]

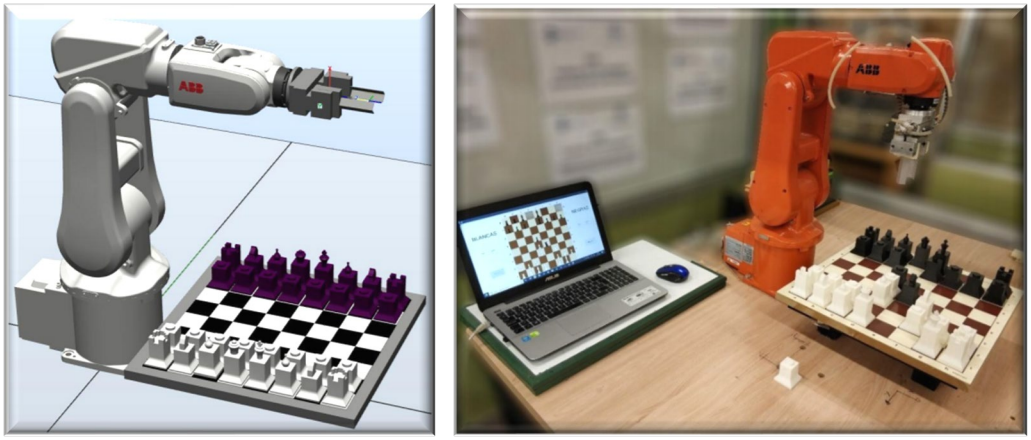
Al año siguiente (2016), Álvaro Galindo de Santos [13], aprovechó las herramientas y componentes educativos que se había desarrollado en años anteriores, para realizar una serie de juegos, controlados por el usuario mediante una *Tablet*. Con esto se buscaba controlar tanto el robot simulado como el real, gracias al establecimiento de una conexión WI-FI.



A) ***B)***

Figura 9. A) *Célula de trabajo TFG de A. Galindo [Ilustración].* [13]
B) *Imágenes tomadas de la Tablet de control [Ilustración].* [13]

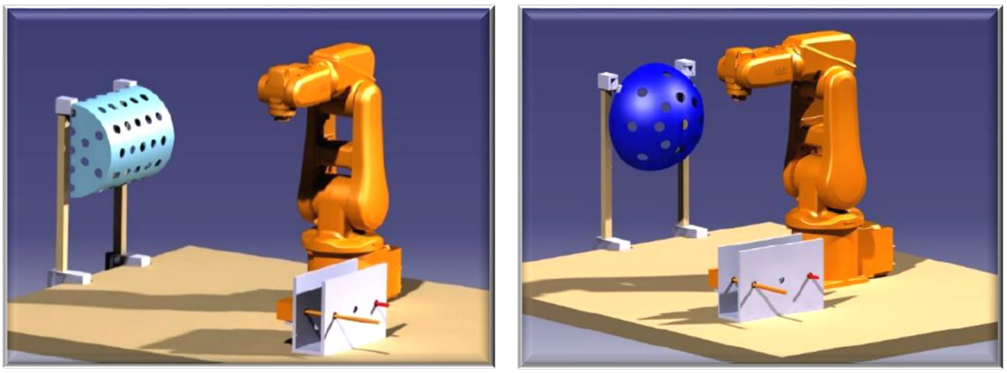
En el 2019, Carlos Jiménez Jiménez [14], para su trabajo de final de máster, desarrollo un sistema robótico educativo con el objetivo de poder jugar al ajedrez contra un robot industrial. El usuario sería el encargado de definir sus jugadas a través de la interfaz desarrollada, y el robot se encargaría de desplazar las piezas por el tablero. Para establecer la comunicación se utilizó el protocolo TCP/IP y los programas utilizados fueron RobotStudio y MATLAB. Este sistema funcionaba tanto en una simulación, como con el robot real.



A) ***B)***

Figura 10. A) *Célula de trabajo TFM de C. Jiménez (Simulada) [Ilustración].* [14]
B) *Célula de trabajo TFM de C. Jiménez (Real) [Ilustración].* [14]

Finalmente, en el año anterior (2020) a la publicación del presente proyecto, el autor Víctor Lobo Granado [15] en su trabajo de final de máster (*Diseño, simulación y fabricación de una estación robotizada universitaria),* desarrollo una serie de componentes (barras de diferentes tamaños, cilindro y semiesfera con una serie de agujeros equiespaciados…) con un objetivo educativo, proponiendo una serie de prácticas en las que dar uso a estos componentes. La comunicación se establecía con el protocolo TCP/IP y se realizó tanto en su versión simulada como en la real.



A) ***B)***

Figura 11. A) *Célula de trabajo TFM de V. Lobo (Cilindro) [Ilustración].* [15]
B) *Célula de trabajo TFM de V. Lobo (Semiesfera) [Ilustración].* [15]

CAPÍTULO 2.2. CRITICA AL ESTADO DEL ARTE

Tras analizar los principales trabajos de fin de grado realizados en la Universidad de Valladolid, en relación con la robótica y más concretamente el software RobotStudio, se puede apreciar como la mayoría de ellos, siguen el mismo patrón de buscar la realización de una tarea didáctica sencilla, ya sea en forma de juego u otras acciones, en la que se establece una comunicación con el protocolo TCP/IP.

Ante el análisis de estos trabajos, al autor de este informe y proyecto, se le ocurrió que se podría desarrollar alguna tarea más compleja, para demostrar el gran abanico de posibilidades que tienen estos robots. Además, en ningún proyecto se ha trabajado con la visión artificial y es una disciplina científica muy recurrente en estos campos de trabajo, ya que permiten la posibilidad de actuar al robot de forma diferente en función de la situación en cada momento.

Al carecer de esta visión, la programación siempre tiene que ser muy cerrada y estricta o controlada por el usuario a través de alguna interfaz.

Otro punto a destacar, es la utilización de un único robot en todos estos proyectos, y más concretamente el modelo de la compañía ABB Robotics, IRB120. Esto es debido a que es el modelo con el que se cuenta en las instalaciones de la universidad.

Pero debido a la situación sanitaria que se ha seguido viviendo en el momento de realización de este proyecto (2020-2021), se sugirió realizar el trabajo entero simulado, sin manipular el robot real. Lo que originó un nuevo mundo de oportunidades, ya que se podrían utilizar cualquier tipo de robot industrial.

Por ello en vez de utilizar únicamente el robot IRB120, se ha decidido hacer uso de un robot colaborativo como es el IRB14000. Lo que originará un desafío mayor para poder coordinar los dos robots correctamente.

Finalmente, como siempre se había utilizado el protocolo TCP/IP, en este caso para seguir con esta política innovadora y debido a las mejores prestaciones que presentaba para este proyecto, se decidió utilizar el protocolo OPC.

CAPÍTULO III. ANÁLISIS DEL PROBLEMA

Ante la falta de proyectos educativos desarrollados en la UVA con el software de RobotStudio, que tratasen otros sectores de la robótica, como la robótica colaborativa en la industria o la visión artificial aplicada en este campo, se ha propuesto el siguiente proyecto, en el que se ocupan estos temas.

Además, como se trata de un trabajo de fin de grado, se ha pretendido reunir todos los campos de la ingeniería posibles en un único proyecto, ya que el objetivo final de este proyecto es demostrar los conocimientos adquiridos a lo largo de estos 4 años de carrera.

CAPÍTULO 3.1. ANÁLISIS DE REQUISITOS

Los campos abarcados en el proyecto de forma muy resumida se muestran en el siguiente esquema:

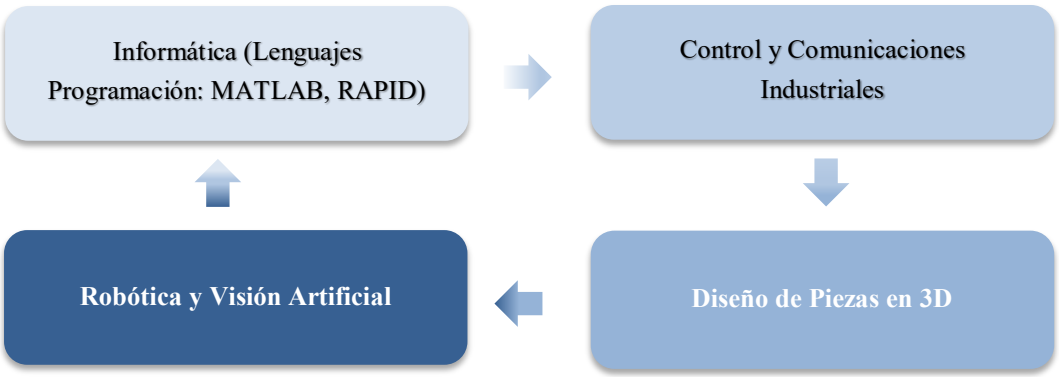


Figura 12. *Esquema campos conocimientos abarcados en el proyecto.*

Antes de comenzar cualquier proyecto, es necesario definir y establecer una lista con los requisitos que se pretenden cumplir, tras la realización de este, por lo que a continuación se muestran dicha lista.

Los siguientes requisitos son los que el autor del presente proyecto, ha tomado de partida para el desarrollo del trabajo realizado.

- Manejo de uno o varios robots (Programación, lógica…).
- Coordinación y sincronización entre los robots y las tareas realizadas.
- Realización de un proceso complejo en el que se muestren la gran cantidad de posibilidades con las que cuenta un robot.
- Investigar métodos de aplicación de la visión artificial en el campo de la robótica.
- Diseño de una interfaz de usuario para el control de la simulación.
- Establecimiento de algún protocolo de comunicación industrial.

CAPÍTULO 3.2. ANÁLISIS DE LAS SOLUCIONES

En la búsqueda de posibles proyectos que cumplan los anteriores requisitos, surgieron una serie de propuestas, que a continuación serán comentadas brevemente y se detallaran los requisitos que no se cumplirían y, por tanto, la razón de que no fuesen seleccionados.

1. **Crear una estación con YuMi que toque un xilófono con las dos manos:**
   Desde Matlab se manda en tiempo real las notas de la música para ambos brazos y el robot toca de forma simultánea y sin colisiones la música usando los dos brazos.

   1. Sincronización de ambos brazos del robot colaborativo YuMi
   2. Comunicación entre RobotStudio y Matlab (Desarrollo de interfaz)
   3. Simulación muy dinámica y entretenida

   1. Utilización de un único robot
   2. No se desarrollarían algoritmos de visión artificial

2. **Cintas trasportadoras para tomar y dejar diferentes piezas:**
   Con un panel de control en Matlab y simulando visión u otro tipo de detección de piezas realizar este programa.

   1. Aplicación de algoritmos de visión artificial
   2. Comunicación entre RobotStudio y Matlab (Desarrollo de interfaz)

   1. Utilización de un único robot
   2. Sencillez en la realización

3. **Mover una bandeja con latas con un robot y definir una trayectoria entre las latas con otro robot de forma coordinada:**
   Con un panel de control desde Matlab realizar procesos paralelos complejos entre dos robots.

   1. Utilización de varios robots
   2. Sincronización entre las diferentes tareas a realizar
   3. Comunicación entre RobotStudio y Matlab (Desarrollo de interfaz)

   1. No se desarrollarían algoritmos de visión artificial

4. **Analizar una geometría triangular 3D de una pieza y reproducir su superficie con el robot:**
   Por ejemplo, para pintar la superficie de la carrocería de un coche.

   1. Aplicación práctica en la industria
   2. Análisis de formas geométricas con software especializado

   1. Utilización de un único robot
   2. No se desarrollan algoritmos de visión artificial

5. **Usar la cámara y sensores de un móvil para mover el robot:**
   Trabajar en paralelo entre la app de Matlab para móviles, Matlab (ordenador) y RobotStudio para dar órdenes al robot desde el móvil.

   1. Utilización del móvil (todo el mundo tiene uno) para el control de un robot
   2. Sincronización entre aplicaciones para realizar los movimientos correctamente
   3. Desarrollo de algoritmos de visión artificial

   1. Utilización de un único robot

Tras el estudio de las anteriores propuestas, se decidió que no eran proyectos suficientemente interesantes, para demostrar todo el rango de posibilidades que se pueden realizar. Por lo que finalmente se optó por la solución explicada en el siguiente capítulo.

CAPÍTULO 3.3. SOLUCIÓN PROPUESTA

La solución seleccionada consiste en el uso de dos robots para realizar una serie de tareas. Los robots son modelos de la compañía ABB Robotics, en el cual uno de ellos es el IRB120 para realizar operaciones de '*Pick & Place*' con objetos de reducido tamaño, y el otro modelo utilizado es un robot colaborativo IRB14000 (YuMi), que cuenta con dos brazos robóticos independientes.

El objetivo es la realización del montaje de un automóvil de juguete a piezas, las cuales alcanzan la posición del robot por medio de una serie de cintas transportadoras. Uno de los robots es el encargado de seleccionar las diferentes piezas y el otro, realiza el montaje de estas.

Con este ejercicio se pretende realizar una sincronización entre las diferentes tareas y los robots, para que todo se complete de manera correcta.

Además, cabe destacar que se tendrán que diseñar todos los componentes, tanto de la estación, como las piezas del automóvil. Por lo que se necesita una revisión de la normativa robótica actual, para el diseño de la célula de trabajo.

Adicionalmente, se añade una segunda tarea a realizar, que consiste en la detección de una serie de parámetros de un objeto gracias al uso de un algoritmo de visión artificial. Más concretamente, se busca que el programa sea capaz de tomar una captura de la simulación y procesar esa imagen, para detectar si existe un cubo, y si es así, que color, tamaño y posición ocupa.

Estas simulaciones se pretenden que sean realizadas de forma interactiva por un usuario, es decir, se necesita desarrollar una interfaz en la que el usuario pueda seleccionar diferentes parámetros de la simulación y en la que pueda seleccionar las diferentes tareas que se pueden realizar.

Finalmente, para la comunicación entre los diferentes softwares utilizados, se va a utilizar el protocolo de comunicación OPC UA.

Con todo esto, se ha considerado que se cumplían y abarcaban todos los campos de la ingeniería que tienen relación con el sector robótico.

Por tanto, a lo largo de los siguientes capítulos, se muestra y explica todos los elementos que han sido necesarios para la realización de esta solución.

CAPÍTULO IV. DISEÑO DE LA SOLUCIÓN

CAPÍTULO 4.1. SOFTWARE UTILIZADO

Durante los siguientes capítulos se introducirá cada uno de los programas/software, que han sido necesarios para desarrollar este proyecto. Ya que, al buscar integrar varias ramas de la ingeniería en un proyecto robótico, se han necesito diferentes programas, para resolver diversas situaciones y problemas que se planteaban.

En el siguiente esquema se muestra con mayor claridad cuáles eran estas situaciones, y que programas se han utilizado:

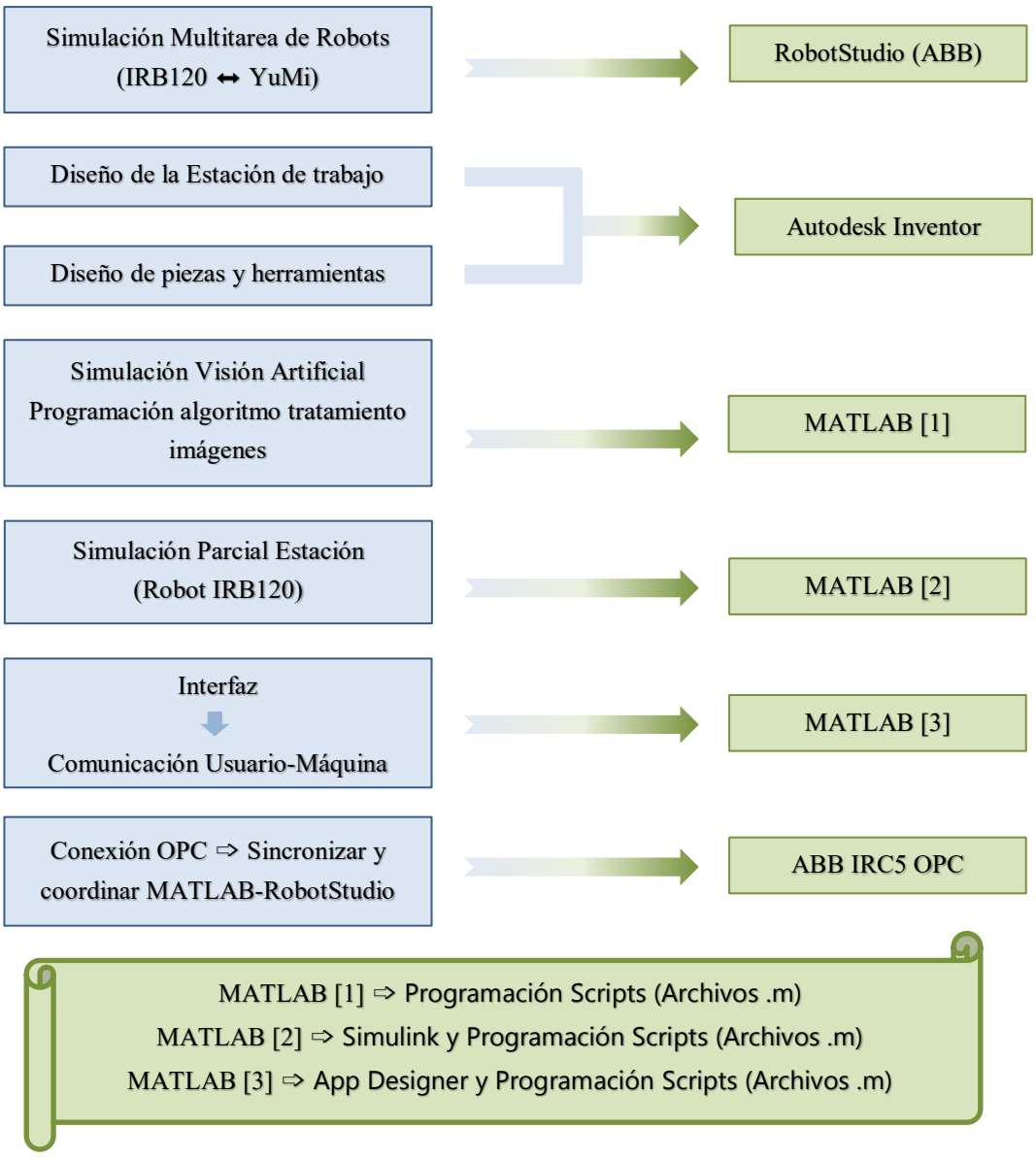


Figura 13. *Esquema software utilizado y sus principales aplicaciones.*

Como se puede visualizar en el anterior esquema (Véase *Figura 13*), para la creación y desarrollo de este proyecto se han necesitado 4 programas, de los cuales 3 de ellos se tratan de software muy potente en su campo de utilización.

Estas herramientas permiten realizar operaciones como el diseño de piezas y ensamblajes en 3D, simulaciones de estaciones robóticas, cálculos matemáticos e infinidad de simulaciones en un amplio campo de trabajo, comunicación OPC entre varios dispositivos, e infinidad de posibilidades más que no serán mencionadas en este proyecto.

Por lo que, tras esta breve explicación, el siguiente paso será la exposición e introducción de dicho software al lector de este informe, de manera que pueda comprender y asimilar el desarrollo y cada una de las maneras de proceder para obtener el resultado final.

CAPÍTULO 4.1.1. ROBOTSTUDIO

El objetivo final de este trabajo es realizar unas operaciones con una serie de robots industriales, para ello se necesita un entorno de trabajo simulado, en el cual se programarán los movimientos y acciones a realizar por estos robots.

En la selección del fabricante y tipo del robot, el factor determinante fue la disponibilidad de estos en las instalaciones de la Escuela de Ingeniería Industriales (EII) de la Universidad de Valladolid (UVA).

Por tanto, la empresa elegida de robótica industrial para este proyecto es ABB Robotics, ya que los robots con los que se trabaja en este entorno de trabajo de la universidad son de esta empresa.

[16] Dicha empresa tiene un gran reconocimiento en este sector y si se realiza un estudio de mercado para las diferentes empresas líderes de robótica industrial se puede obtener lo siguiente (2º Puesto empresa robótica industrial con mayores ingresos):

Empresas Robótica Industrial	*Ingresos (Millones de Euros)*
Mitsubishi Electric (Japón)	11,97
ABB Robotics (Suiza)	6,9
B+M Surface Systems GmbH (Alemania)	4,4
B+M Surface Systems GmbH (Alemania)	4,4
Omron Adept Technologies (Japón)	3,05
Fanuc Robotics (Japón)	1,7
Yaskawa (Japón)	1,5
KUKA Robotics (Alemania)	1,4
Epson Robots (EE. UU.)	1,4
Kawasaki Heavy Industries (Japón)	1,3

Tabla 3. *Estudio de mercado de empresas líderes de robótica industrial.* [16]

Como el proyecto no se ha realizado de forma física, se ha necesitado de algún software capaz de simular el entorno físico de trabajo de los robots con la mayor realidad posible. Para ello al haber seleccionado los robots industriales de la empresa ABB Robotics, se ha utilizado el software del que dispone esta empresa para la programación de sus robots y simulación de las células de trabajo.

El programa es 'RobotStudio' y este es una aplicación de PC destinada al modelado, la programación fuera de línea y la simulación de células de robot.

[17] Se basa en ABB VirtualController, que es una copia exacta del software real que ejecuta sus robots en producción. Es decir, cualquier movimiento que se simule en el programa, el robot lo realizaría de la misma manera en la realidad. Por lo que, a pesar de ser un proyecto simulado, se podría llevar a cabo en la realidad en cualquier momento, sin necesidad de modificar ningún aspecto, gracias a la utilización de este software. Permite a su vez, el uso de archivos de configuración idénticos a los utilizados en el taller.

La primera acción a realizar para iniciar cualquier proyecto robótico en este software, consistirá en la creación de la estación de trabajo. Para ello en *Archivos/Nuevo* se encuentran 3 opciones:

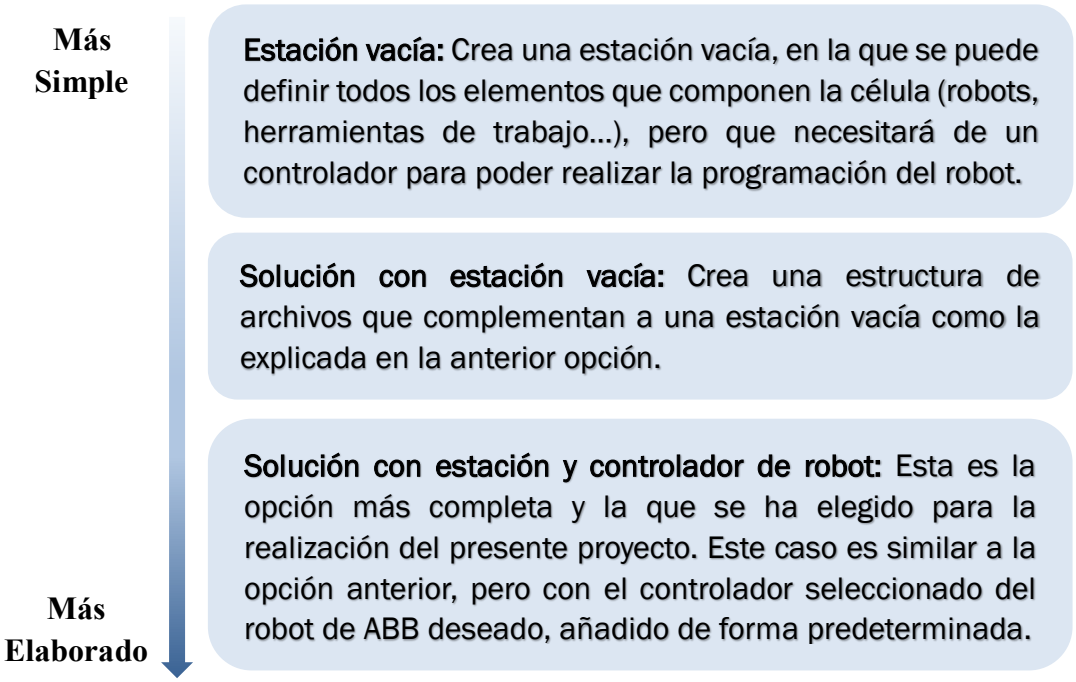


Figura 14. *Opciones creación estación de trabajo en RobotStudio.*

Una vez se ha creado la estación de trabajo, este será el espacio donde se desarrollarán todas las acciones que el programador considere oportunas.

CAPÍTULO 4.1.2. AUTODESK INVENTOR

Este proyecto contiene una gran parte de diseño, ya que todos los elementos que componen la estación de trabajo, a excepción de los propios robots, han sido creados y diseñados con esta potente herramienta, con la que se puede crear prácticamente cualquier pieza. Además, contiene una gran variedad de opciones extras que se comentarán a continuación.

Por lo que el primer paso es preguntar… *¿Qué es Autodesk inventor?*

[18] Autodesk Inventor es un paquete de modelado paramétrico de sólidos en 3D producido por la empresa de software *Autodesk*, en términos más generales, se podría definir como un software de diseño asistido por computadora, para productos e ingeniería. Su lanzamiento en el mercado sucedió a día 20 de septiembre de 1999, para los sistemas operativos Microsoft Windows y Mac os.

Este programa surgió como solución a la creciente demanda en los clientes de diseños mecánicos, de crear modelos en 3D, y con mayor extensión y complejidad. Otros programas que ya existían anteriormente, para satisfacer estas demandas eran:

- SolidWork
- Pro/ENGINEER
- CATIA
- Solid Edge

Entre las posibilidades que se pueden realizar con este software se encuentran las siguientes:

Diseño mecánico 3D, con layouts, trazos, perfiles, diseño de piezas plásticas y sus respectivos moldes de inyección, diseño de piezas de lámina para chapas, diseño de ensambles. Cuenta también con análisis de tensiones por elementos finitos y análisis dinámicos. Además, permite la creación y análisis de estructuras, piping y cableado, y las tecnologías iPart, iAssembly, iMates, iCopy, iLogic hacen que el diseño sea más rápido y eficiente. Por último, también dispone de comunicación de diseños, manejo de datos, visualización 3D, documentación y enlace a manufactura.

Como se puede apreciar las posibilidades de las que dispone este software son muy elevadas, pero para el caso de este proyecto, no se han necesitado muchas de ellas, ya que se ha utilizado simplemente para el diseño en 3D de todos los componentes necesarios, sin entrar en aspectos técnicos de cada pieza.

Curiosidad

Como curiosidad se puede destacar que la denominación del nombre de los primeros códigos fuente en las primeras versiones se obtuvieron de coches populares de esa época. En versiones posteriores, se cambió esa dinámica a nombres de científicos. A continuación, se muestra una lista de los nombres en las diferentes versiones

Nombre Oficial	Versión	Nombre Código	Fecha de Lanzamiento
Inventor	1	Mustang	20 de septiembre de 1999
Inventor	2	Thunderbird	1 de marzo de 2000
Inventor	3	Camaro	1 de agosto de 2000
Inventor	4	Corvette	1 de diciembre de 2000
Inventor	5	Durango	17 de septiembre de 2000
Inventor	5.3	Prowler	30 de enero de 2002
Inventor	6	Viper	15 de octubre de 2002
Inventor	7	Wrangler	18 de abril de 2003
Inventor	8	Cherokee	15 de octubre de 2003
Inventor	9	Crossfire	15 de julio de 2004
Inventor	10	Freestyle	6 de abril de 2005
Inventor	11	Faraday	6 de abril de 2006
Inventor	2008	Goddard	11 de abril de 2007
Inventor	2009	Tesla	16 de abril de 2008
Inventor	2010	Hopper	27 de febrero de 2009
Inventor	2011	Sikorsky	26 de marzo de 2010

Tabla 4. *Estudio de mercado de empresas líderes de robótica industrial.* [18]

Una vez se ha introducido brevemente qué es el programa y su historia, a continuación, se muestra cómo manejarlo y los primeros pasos que hay que tomar para desarrollar cualquier pieza o ensamblaje.

En primer lugar, hay que crear un Proyecto nuevo en el programa, donde se irán introduciendo todas las piezas y ensamblajes que se desarrollen. Para ello en *Iniciar/Proyectos* se abre una nueva ventana y habrá que seleccionar la casilla que indica 'Nuevo'. Por ejemplo, para el caso de este proyecto, se creó el Proyecto denominado 'PiezasTFG' (Véase *Figura 15*)

Figura 15. *Nombre y ruta del proyecto creado en Autodesk Inventor.*

Tras crear y seleccionar el proyecto ya se pueden empezar a diseñar y desarrollar las piezas deseadas. Para ello en *Inicio/Nuevo*, se origina una nueva ventana en la que se aportan distintas opciones como se muestra en la siguiente *Figura 16*:

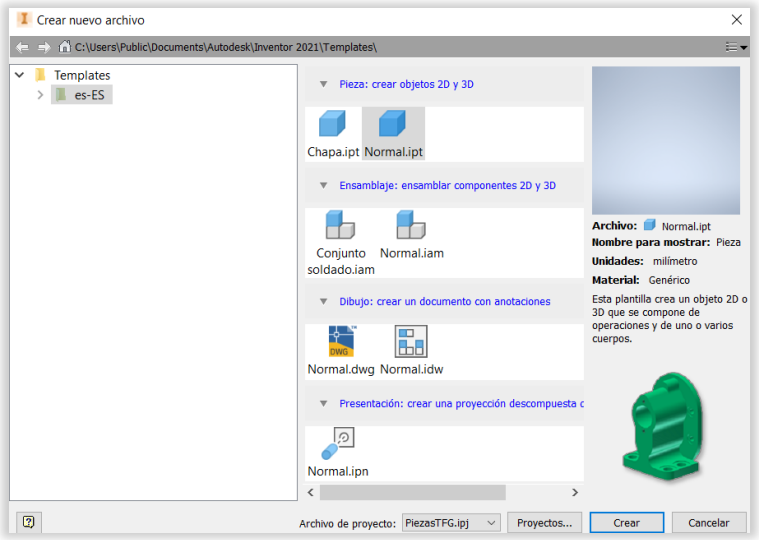


Figura 16. *Ventana para crear una nueva pieza en Autodesk Inventor.*

Para los diseños de este proyecto solo se han necesitado, crear objetos 3D 'Normal.ipt' y ensamblajes 'Normal.iam'.

En el caso de creación de una nueva Pieza en 3D, el primer paso a tomar será ir creando los bocetos en los diferentes planos que se desee, y tras finalizar el trazado de cada boceto, se realizan las operaciones correspondientes (generalmente se ha utilizado la herramienta de 'Extruir', 'Empalme' y 'Agujero').

En piezas más complejas serán necesarias operaciones adicionales que requieren un conocimiento mayor de este programa, como la solevación o el barrido.

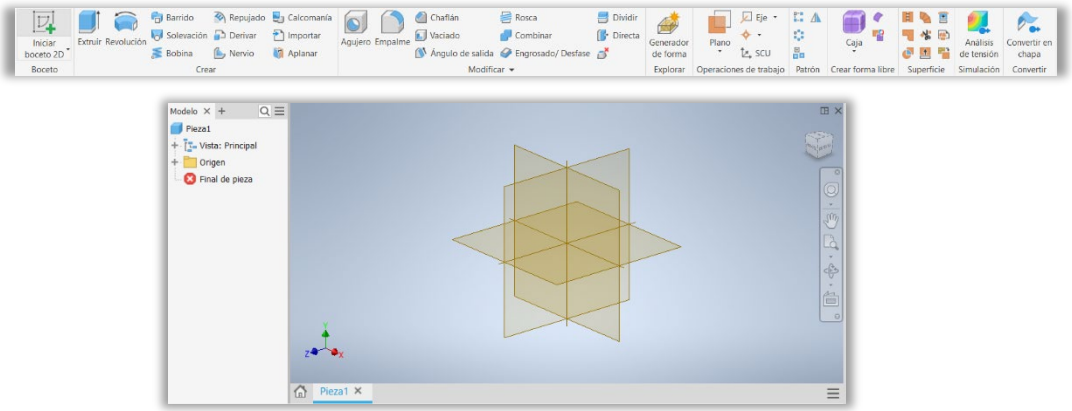


Figura 17. *Ventana de creación de las piezas 3D y herramientas disponibles.*

Una vez se ha diseñado la pieza deseada, hay que aportarle un material y color, para mejorar su aspecto visual. Para ello en *Herramientas/Aspecto*, se origina una nueva ventana, en la que se podrá elegir una gran variedad de materiales para asignar a la pieza creada.

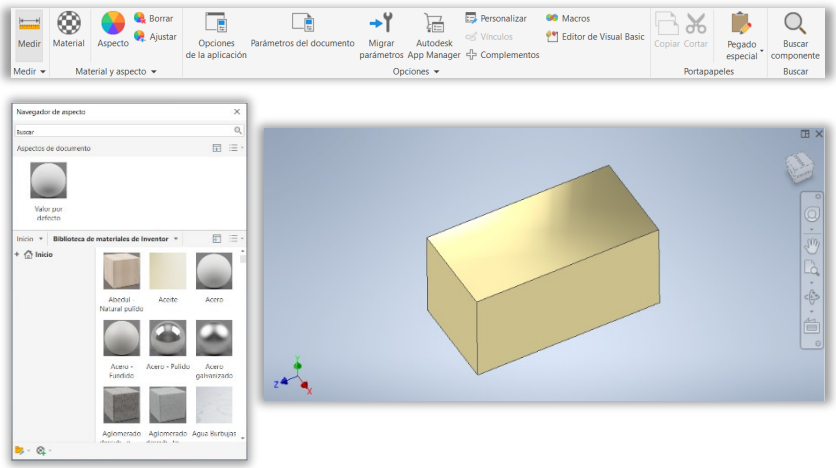


Figura 18. *Asignación de materiales en los objetos 3D diseñados.*

Finalmente, tras diseñar todas las piezas que conformarán el objeto, el último paso es ensamblarlas en un único elemento y establecer el tipo de movilidad que tienen unas piezas con otras. Para ello existen las 'Uniones' y las 'Restricciones', y dentro de cada modalidad como se puede apreciar en las siguientes imágenes (*Figura 19*), existen diferentes tipos.

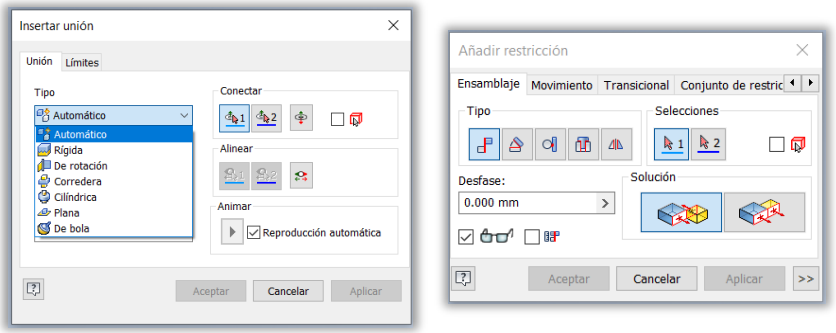


Figura 19. *Ventana para la configuración de los tipos de movilidad en los ensamblajes.*

Una vez realizado el ensamblaje se pueden encontrar montajes tan complejos como el mostrado a continuación:

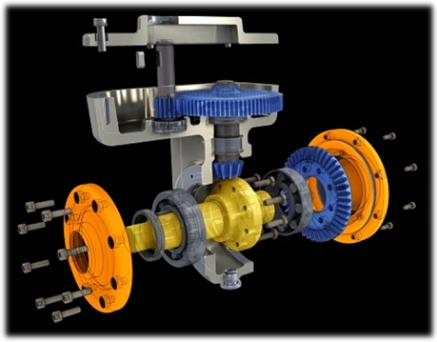


Figura 20. *Ejemplo de ensamblaje tomado de 3DCadPortal [Ilustración].* [19]

CAPÍTULO 4.1.3. MATLAB

Otra herramienta software fundamental para el desarrollo del presente proyecto, ha sido MATLAB. Esta aplicación informática es un sistema de cómputo numérico (software matemático) que combina un entorno de escritorio perfeccionado para el análisis iterativo y los procesos de diseño con un lenguaje de programación que expresa las matemáticas de matrices y *arrays* directamente.

[20] Sus orígenes se remontan al 1984, año en el que este software fue desarrollado por Cleve Moler, utilizando los lenguajes de programación C y Java. El sistema operativo que soporta es Microsoft Windows, macOS y GNU/Linux.

La presente aplicación cuenta con un lenguaje de programación propio (lenguaje M). Este lenguaje es interpretado, y puede ejecutarse tanto en el entorno interactivo, como a través de un archivo de script (archivos '.m'). Además, con estos lenguajes se pueden realizar operaciones de vectores y matrices, funciones, cálculo lambda y programación orientada a objetos.

También permite la representación de datos en 2D y 3D, lo que supone una gran herramienta para la representación de gráficos.

Una de las características que hacen a este software tan potente y que expanden sus posibilidades de uso, son las *toolboxes*. Con estas se pueden desarrollar aplicaciones en una gran variedad de campos como son:

- Sistemas de conducción automática de automóviles
- Biología Computacional
- Sistemas de controla
- Data science
- Redes neuronales
- Sistemas embebidos
- Sistemas de TI y empresariales
- Desarrollo de FPGA, ASIC y SoC
- Procesamiento de imágenes y visión artificial
- Internet de las cosas
- Machine learning
- Mecatrónica
- Sistemas de señal mixta
- Diseño de sistemas de control de electrónica de potencia
- Análisis y diseño de sistemas de energía
- Mantenimiento Predictivo
- Robótica
- Procesamiento de señales
- Prueba y Medición
- Comunicaciones inalámbricas

Esto origina que este software se pueda utilizar en una gran variedad de sectores, como el sector aeroespacial, automoción, biotecnología, comunicaciones, energía, finanzas…

Debido a su gran abanico de posibilidades, en el presente proyecto se han utilizado diferentes aplicaciones y *toolboxes* disponibles al usuario, gracias al uso de la licencia para estudiantes, que ha sido obtenida por ser estudiante de la Universidad de Valladolid.

[21] La *toolbox* requerida para el procesamiento de imágenes con visión artificial es '*Image Processing Toolbox*', en la cual se proporcionan una serie de algoritmos y funciones específicas, para la visualización, análisis y posterior procesamiento de imágenes. Entre las funciones que se incluyen se encuentran la mejora de imágenes, reducción del ruido, detección de bordes, procesamiento de imágenes 3D, etc. Además, permite el desarrollo de algoritmos de visión.

Incluye automatización de los flujos de trabajo habituales durante el procesamiento de imágenes, segmentación de datos, comparación de técnicas y cálculos por lotes de conjuntos de datos. En el aspecto visual, esta herramienta facilita la exploración de imágenes y videos, ajustes de parámetros de la imagen, volúmenes 3D, histogramas y manipulaciones en regiones de interés de una imagen (ROI - *Region of interest*).

Estos algoritmos y funciones se programan en los archivos '.m' (scripts) que se había comentado anteriormente.

[22] Otra *toolbox* utilizada ha sido '*OPC Toolbox'*, en la que se permite el acceso a datos OPC históricos y en tiempo real, y se permite la lectura, escritura y el registro de datos OPC de diferentes dispositivos.

En esta toolbox se permite trabajar con los siguientes estándares OPC, OPC Data Access (DA), OPC Historical Data Access (HDA) y OPC Unified Architecture (UA), siendo este último el utilizado en el proyecto actual.

A su vez establece una seguridad en la conexión gracias a la utilización de algoritmos y métodos de autentificación.

La siguiente herramienta utilizada ha sido Simulink, la cual, es un entorno de programación visual mediante bloques obtenidos de las librerías de MATLAB. Esta herramienta es muy potente ya que, al pertenecer al entorno de programación de MATLAB, se tienen la mayoría de posibilidades que este ofrece, pero dispuestas visualmente mediante bloques de programación.

La creación de sistemas se consigue gracias a la consecución de bloques, que en realidad actúan como cajas negras que realizan alguna operación.

Se podría considerar como una herramienta de simulación de modelos o sistemas, en la cual los fenómenos físicos pueden no ser exactos a la realidad, en función del grado de abstracción del modelo.

Finalmente, se ha utilizado una aplicación de MATLAB en la que se permite la creación de interfaces, cuyo nombre es 'App Designer'. En esta se pueden realizar apps profesionales, sin tener un grado muy alto de conocimientos de programación de software.

Ya que para el diseño dispone de una biblioteca de elementos, los cuales pueden ser utilizados por el usuario para el desarrollo de su aplicación. La interfaz gráfica resultante se caracteriza por el nombre 'GUI' (*Graphical User Interface*).

CAPÍTULO 4.1.4. ABB IRC5 OPC

Con la necesidad de comunicarse y coordinarse entre los programas utilizados, surgió la oportunidad de usar el software ABB IRC5 OPC.

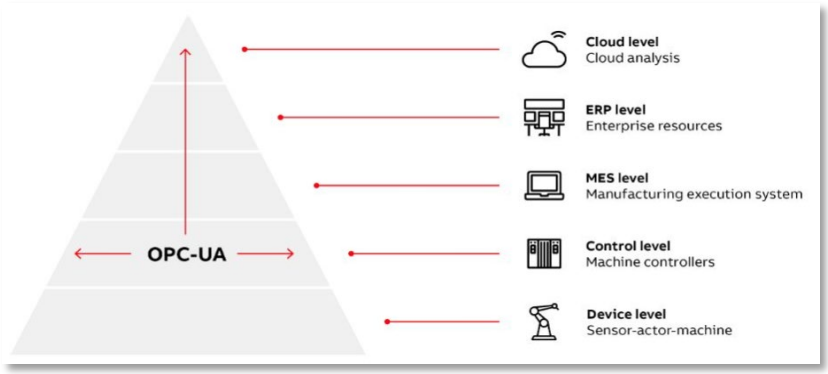


Figura 21. *Ejemplo Niveles Jerárquicos (Comunicación industrial) [Gráfico].* [23]

En este software se permite la comunicación con el protocolo OPC UA, gracias a la creación de un servidor en la máquina que ejecuta el programa.

Este programa está pensado para combinar y sincronizar la planta de la fábrica en la que se encuentran los robots a su sistema ERP (*Enterprise Resource Planning*) con el fin de obtener una visión completa del proceso de fabricación.

Con OPC UA se puede transmitir perfectamente información y datos desde la planta de la fábrica de máquina en máquina (M2M), hasta el sistema de monitorización (*MES-Manufacturing Execution System*), o subir dichos datos a la nube, conectando los robots de ABB a dispositivos de otros proveedores.

Para entender mejor el funcionamiento del software, se va a explicar en primer lugar que es OPC UA.

Las siglas significan lo siguiente:

- ***OPC:*** Open Platform Communications (Comunicaciones de plataforma abierta), aunque en su origen en 1996 se denominaba 'OLE for Process Control' (OLE para control de procesos), siendo OLE- Object Linking and Embedding (Incrustación y enlazado de objetos).
- ***UA:*** Unified Architecture (Arquitectura Unificada)

OPC UA es la evolución de la tecnología OPC clásica, en la que se busca la comunicación industrial multiplataforma, abierta, segura y orientada a servicios.

Fijándose únicamente en OPC, esta es una tecnología de comunicación en la que la característica del tráfico, es decir, los patrones de interacción se realizan por media de la pareja cliente-servidor. Este método para la conexión está basado en los estándares más populares.

Su uso principal es la industria de la automatización (comunicación entre dispositivos, controladores…). Como tal OPC no es considerado un protocolo, si no un estándar (en las redes industriales y buses de campo) para la conectividad de datos, basado en una serie de requisitos gestionados por *OPC Foundation*.

Una de las grandes ventajas de este estándar, es la compatibilidad y conectividad proporcionada, ya que no existe dependencia entre el fabricante del dispositivo y el desarrollador de la aplicación cliente, por lo tanto, resulta una gran solución para estos ambientes industriales.

Esto anterior es debido a que no se necesita de ningún driver para establecer la conectividad. Además, otra característica importante es que un único cliente OPC puede comunicarse con tantos servidores OPC como se necesite.

Una vez sentadas las bases de la comunicación OPC, se puede explicar que supone esta modificación de OPC UA, en la que se expande el horizonte de uso a todo tipo de dispositivos y tecnologías modernas utilizadas en las fábricas inteligentes modernas (dispositivos móviles, inteligencia y visión artificial, mantenimiento predictivo, bases de datos…), ya que en la versión clásica solo funcionaba en dispositivos con Windows de Microsoft. Es decir, busca el poder establecer una comunicación con todas las aplicaciones de la empresa y a través de todos los diferentes niveles que conforman a esta.

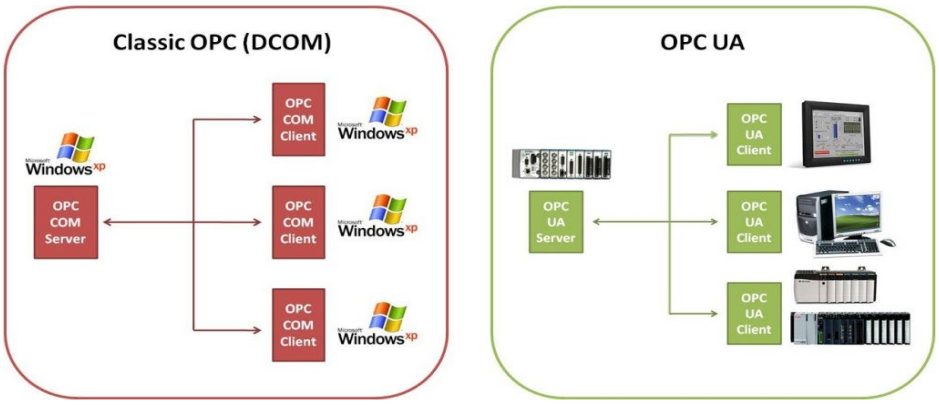


Figura 22. *Diferencias entre OPC clásico y OPC UA [Gráfico].* [25]

Una vez se ha explicado en que consiste el estándar utilizado, en el presente proyecto se dividen dos ramas para el establecimiento de la comunicación:

- ***Servidor OPC UA:***

  Creado con la aplicación introducida en este capítulo (*ABB IRC5 OPC*)

- ***Toolbox OPC de MATLAB:***

  Explicado en anteriores capítulos (véase *Capítulo 4.1.3. MATLAB*)

Para explicar cómo se realiza la comunicación establecida, se ha diseñado el siguiente diagrama:

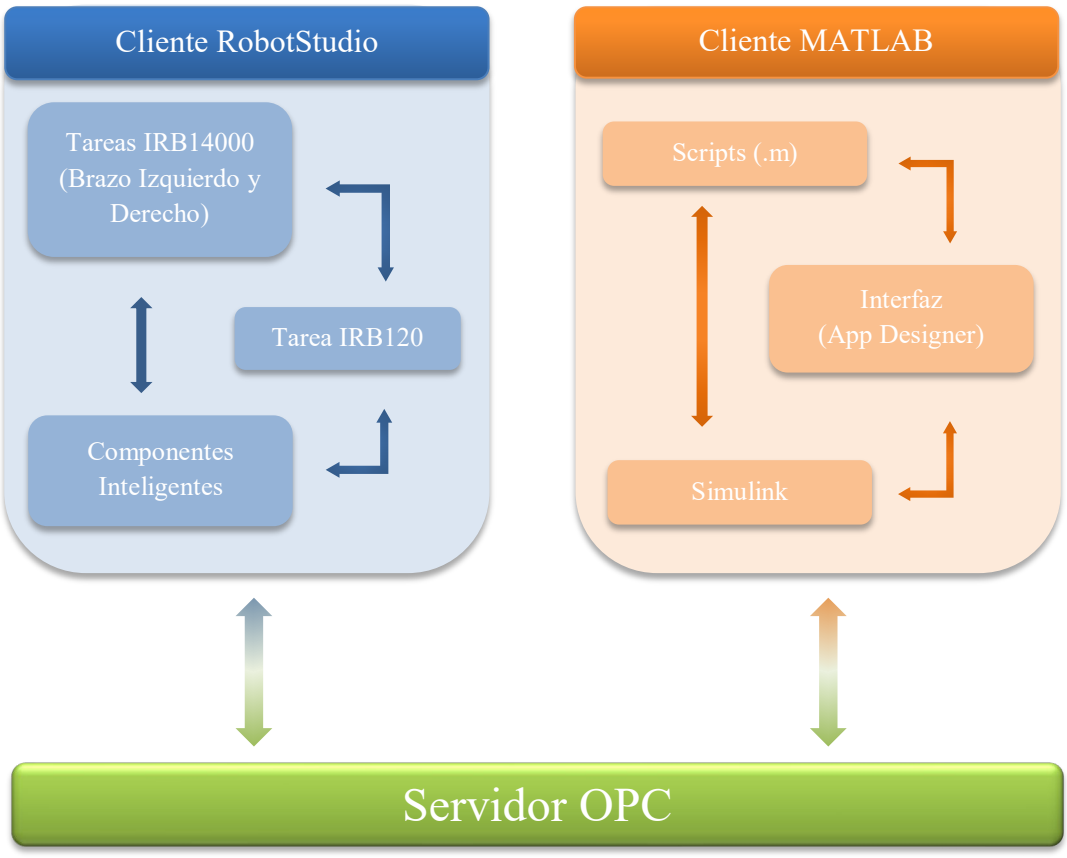


Figura 23. *Comunicación entre el software utilizado [Gráfico].*

CAPÍTULO 4.2. DISEÑO DE LA ESTACIÓN

Al tratarse de un proyecto robótico, una de las partes fundamentales que compone el sistema será el entorno que rodea a los robots utilizados.

Debido a la situación actual de crisis sanitaria, en la que se sigue encontrando el mundo entero, por la propagación de un virus denominado 'COVID-19' o 'Coronavirus' (enfermedad infecciosa causada por el SARS-CoV-2), en el momento de la realización de este trabajo de fin de grado (2021), las medidas de precaución siguen siendo un factor muy importante a tener en cuenta (Véase *Figura 24*).

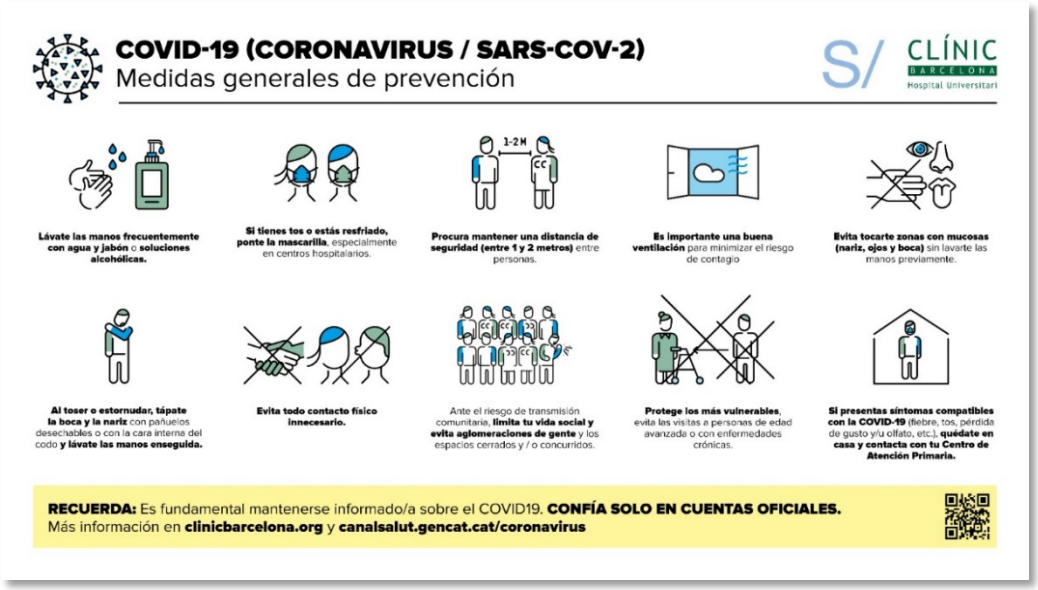


Figura 24. *Medidas generales de prevención COVID-19 [Ilustración].* [26]

Por este motivo la utilización de los robots físicos disponibles en las instalaciones de la Universidad de Valladolid, ha resultado imposible, ya que suponía una serie de factores que dificultarían el seguimiento de estas medidas preventivas de una manera adecuada.

A pesar de esto cabe destacar que los programas utilizados, son totalmente válidos y se utilizan actualmente en las empresas en el ámbito industrial, para el diseño y programación de los elementos físicos. Por lo que, aunque en este caso la realización del proyecto haya sido simulada, todo el código se podría simular de la misma manera en los robots reales, y todas las piezas se podrían crear o imprimir en 3D físicamente.

Por tanto, al tratarse de un trabajo simulado no existe una estación de trabajo definida sobre la que partir y diseñar para asimilar a la realidad. Esto origina la producción de un trabajo creativo, para crear una estación de trabajo totalmente novedosa y a su vez, con suficiente parte realista para que existiera en la industria.

Adicionalmente procede comentar, que la robótica está cada vez más regulada por normas y leyes, lo que influye en la creación de estos entornos de trabajo, ya que, al tratarse de maquinaria peligrosa para el ser humano, necesita de una regulación bastante contundente, de manera que se eviten en la mayor medida posible, accidentes industriales originados por robots.

CAPÍTULO 4.2.1. NORMATIVA ENTORNO DE TRABAJO DE LOS ROBOTS

La normativa que regula el campo de la robótica surge por la necesidad de garantizar la seguridad de las personas y de eliminar en la medida de lo posible, los accidentes que puedan ocurrir. Ya que los robots, a pesar de no poseer datos alarmantes de accidentes ocasionados, son una de las máquinas que poseen mayor índice de riesgo, respecto a máquinas con características similares.

A lo anterior se le añade la difícil aceptación social de los robots en la industria, lo que supone que haya que asegurar ciertos parámetros de funcionamiento.

[27] Por tanto, el primer aspecto a tener en cuenta es el estudio de las posibles situaciones que dan lugar a los accidentes en este campo.

Contactos Mecánicos	Acceso a partes peligrosas durante funcionamiento automático.
	Durante el ajuste y/o programación o pruebas del robot (protecciones pueden estar abiertas o inefectivas)
	Tareas de limpieza o mantenimiento, por arranque intempestivo.
	Robot que excede el área restringida: impactos con resguardos, alcance encima de vallado, pudiendo alcanzar zonas del exterior ocupadas.
	Durante el funcionamiento colaborativo
Proyecciones de materiales del proceso	Partículas, polvo, chispas, salpicaduras, materiales o piezas, etc.
Riesgos térmicos	Quemaduras por contacto, atmósfera inflamable o explosiva por disolventes, polvo metálico, etc.
Riesgos eléctricos	Contacto con partes activas (quemaduras por arco eléctrico, electrocución)
	Durante el mantenimiento (elementos activos mal aislados, circuitería, derivaciones eléctricas, condensadores, confusión voltajes, soldadura altos voltajes, etc.

Tabla 5. *Posibles riesgos y accidentes originados por robots.* [27]

Cabe destacar que, según el *Instituto de Investigaciones de Seguridad en el Trabajo de Tokio*, el 90% de los accidentes en líneas robotizadas ocurren durante las operaciones de mantenimiento, ajuste, programación mientras que sólo el 10% ocurre durante el funcionamiento normal de la línea.

Una vez se han analizado los principales accidentes y sus motivos, es turno de analizar la normativa encargada de evitar estas situaciones.

Hasta los años 90, dicha normativa era muy escasa, y es solo, a partir de estos años, es cuando surgen las primeras normas de carácter internacional.

[28], [29] y [30] No habían surgido anteriormente debido a:

- La insuficiente experiencia en la materia de accidentes ocasionados por robots.
- Tendencia de priorizar problemas técnicos y de mercado en primer lugar.
- Dificultad en la unificación de criterios y niveles de seguridad entre los diferentes países.
- La dificultad y el tiempo necesario para realizar dicha documentación y los procedimientos de evaluación correspondientes.

Las diferentes normativas con mayor relevancia surgidas en esta época son:

Normativa internacional ISO 10218 :1992.

Normativa americana ANSI/RIA R15.06-1992.

Normativa europea EN 775 y española UNE-EN 775

De estas, como no es la temática ni la idea principal de este proyecto (hablar únicamente de normas), solo se va a introducir la última de ellas, ya que concierne al país en el que se ha realizado este trabajo de fin de grado (España). Mas concretamente, se centrará el análisis en las 'Medidas de seguridad a tomar en la fase de diseño de la célula robotizada'.

[31] Al ser normas antiguas han sido anuladas por normativas más modernas como se puede ver en el siguiente esquema:

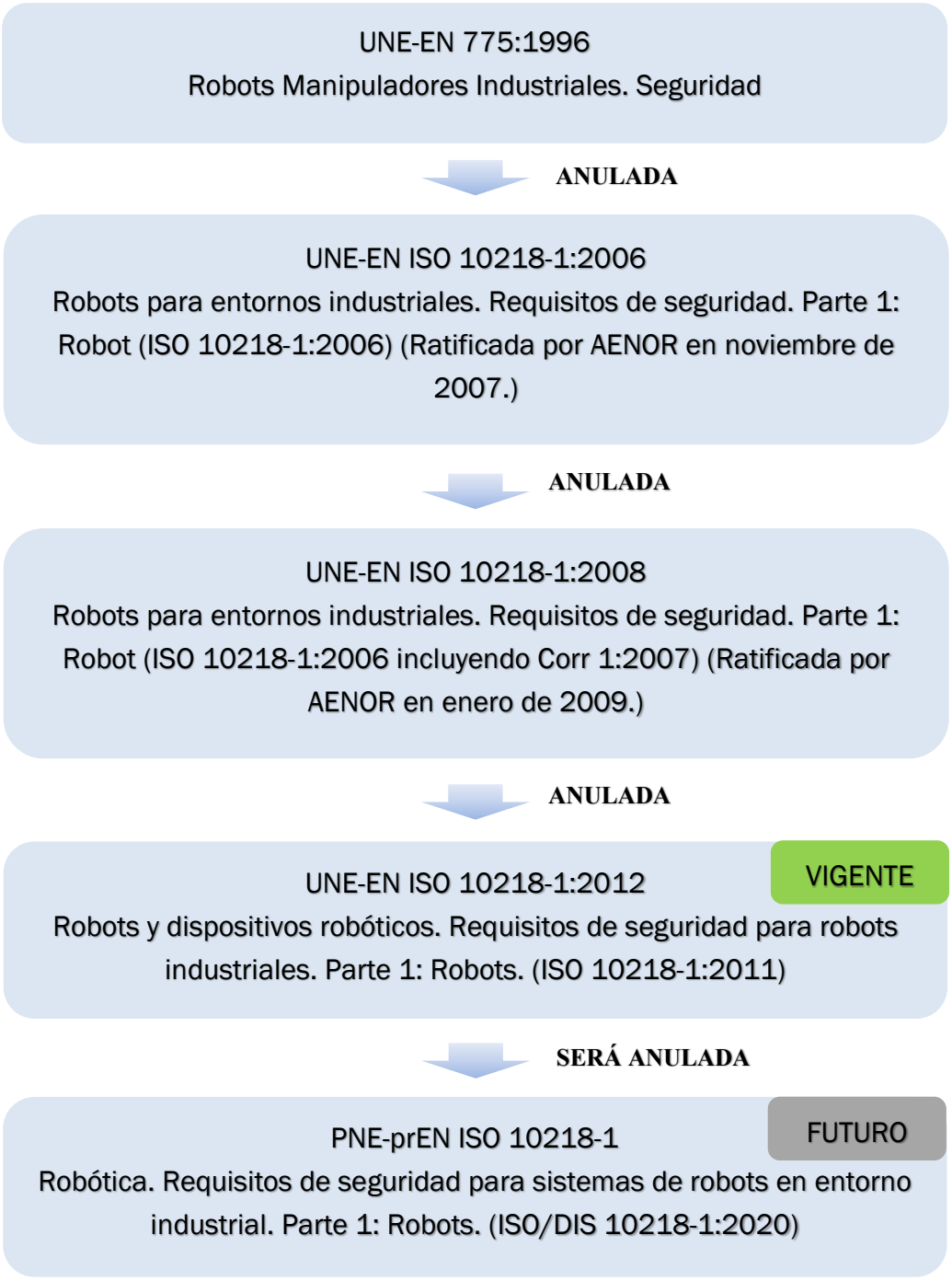


Figura 25. *Evolución de las normativas en el ámbito robótico (España).*

[28], [29] y [30] Medidas de seguridad a tomar en la fase de diseño de la célula robotizada.

Barreras de acceso a la célula

Se limitará el acceso al espacio de trabajo mediante barreras en torno a la célula. En caso de acceso al recinto, se realizará un aparada inmediata de los elementos móviles que se encuentran actuando en dicho espacio.

Dispositivos de intercambio de piezas

Para las situaciones en las que un operario deba poner/recoger piezas situadas dentro del espacio de trabajo del robot, se utilizarán dispositivos que permitan realizar dicha acción a distancia. Por ejemplo, mesas giratorias, conveyors…

Movimientos condicionados

En las situaciones que se requiera de manera inevitable un acceso por parte de un operario dentro del campo de acción del robot, se debe realizar una programación al robot de manera que no efectúa ningún movimiento durante los instantes en los que ocurre esta aproximación del operario.

Zonas de reparación

Deben existir zonas habilitadas para la reparación y el mantenimiento. Estas zonas se encontrarán fuera de la zona de trabajo del robot, pero dentro de su campo de acción. Con estos espacios se pretende evitar que el robot realice movimientos de manera automática, durante los periodos de reparación o mantenimiento.

Condiciones adecuadas en la instalación auxiliar

Todos los sistemas eléctricos, neumáticos o hidráulicos que se requieran para las instalaciones deben estar debidamente protegidos, aislados y con todas las medidas adecuadas para un funcionamiento sin riesgo alguno para el operario, o personas adyacentes a las instalaciones.

Señalización adecuada

La célula estará dotada de una adecuada señalización haciendo referencia al estado del robot o a la línea robotizada mediante señales luminosas y acústicas. De esta manera se advertirá sobre el hecho de que un robot está funcionando en las proximidades.

Instalación de mecanismos de acceso a la célula

Con el objetivo de impedir el acceso de personal no autorizado en las instalaciones, se deben instalar códigos de seguridad, barreras de seguridad fotoeléctricas, sensores de presencia o proximidad y sistemas de visión para reforzar la seguridad.

Instalación de hardware de seguridad

No se debe tener el software como elemento principal de seguridad. Para ello en las instalaciones se deberán instalar un número adecuado de interruptores de parada de emergencia, en los que ante cualquier situación de peligro el operario pueda cortar el suministro de alimentación a los elementos motrices.

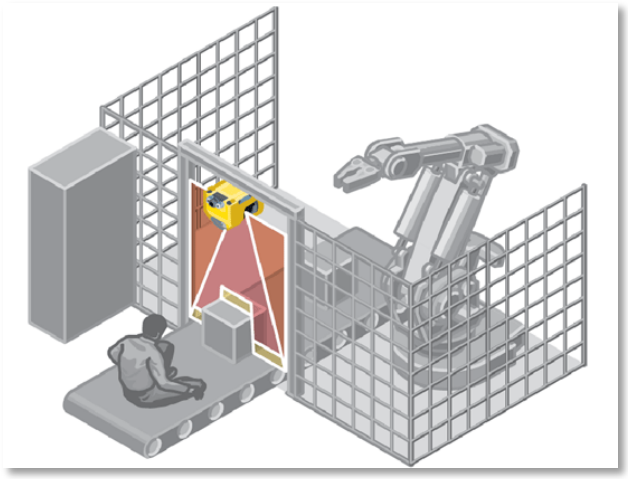


Figura 26. *Seguridad en una célula robótica [Ilustración].* [32]

Estas serían las principales medidas a tomar para el diseño de la célula robótica, pero existe una infinidad de normativa reguladora de todos estos aspectos. Como no es objeto de este trabajo, el siguiente paso va a ser la revisión de los principales aspectos que se deben considerar en el diseño de los componentes de seguridad.

Barreras Materiales

El dimensionado de este tipo de elementos debe ser concordante al tipo de riesgo existente y al robot instalado. La protección se basará, por tanto, en una combinación de los parámetros de distancia y altura, para evitar de esta manera el acceso a cualquier punto peligroso.

Las barreras deberían tener una altura mínima de 1.80 m, pero como no en todos los casos será posible, se debe atener a los datos de la Norma UNE 81-600-85.

Adicionalmente las barreras deberán estar cubiertas de una malla lo suficientemente tupida, para evitar el acceso de cualquier operario con los dedos o las manos a la zona de peligro.

Barreras Inmateriales (Barreras fotoeléctricas)

Diseñadas con el objetivo de interrumpir el suministro energético a las instalaciones ante cualquier obstrucción del camino recorrido por uno o varios rayos luminosos que la forman.

Además, deberán estar dispuestas de manera que sea imposible acceder al espacio de trabajo sin interrumpir dichos rayos (no puede ser eludida).

En la instalación de estos elementos se debe tener en cuenta, la distancia a recorrer para alcanzar el robot, desde que se activan los sensores infrarrojos. Para ello existe una fórmula que relaciona dicha distancia, con la velocidad y los tiempos correspondientes.

$$D \geq V \cdot (t_1 + t_2) + C_3 \qquad \text{Ec. (4.1)}$$

Donde:

D = Distancia de protección.

V = Velocidad de desplazamiento del hombre o operario. Esta velocidad es distinta según los países en Alemania V = 1,6 m/s y en Francia V = 2,5 m/s. 34

t1 = Tiempo de respuesta de la barrera fotoeléctrica.

t2 = Tiempo de parada de la maquina en milisegundos.

C3 = Es una distancia adicional (seguridad). Esta "constante" varía según los países así en Alemania es 180 mm. Y en Francia es de 125 mm

Una vez analizados los aspectos fundamentales a tener en cuenta en el diseño de la célula robótica, se va a dar paso a otro campo que se toca en este proyecto como es el de la Robótica colaborativa.

Seguridad en la Robótica colaborativa

Alrededor del año 2010, surgieron los robots colaborativos o 'cobots', en los que ya no era necesario una protección total, debido a el espacio de trabajo sería compartido por robots y personal al mismo tiempo. Esto supone la creación de nueva normativa para adaptarse a este nuevo tipo de robots.

En la norma ISO 10218-1 y posteriormente en la norma ISO/TS 15066 vienen reflejados 4 modos de trabajo para esta modalidad de robots:

Parada monitorizada de seguridad (Safety-rated monitored stop)

Para permitir la actuación del robot se debe encontrar el operario fuera del espacio de trabajo colaborativo, la reanudación del movimiento del robot se realiza de manera automática una vez el operario abandona el espacio de trabajo. En la norma ISO 60204 viene reflejado dicha actuación como una parada de categoría 2 (se detiene el robot, pero no se corta el suministro de energía).

Movimiento del Robot o Función de Parada		Proximidad del operario al espacio de trabajo colaborativo	
		Exterior	Interior
Proximidad del robot al espacio de trabajo colaborativo	Exterior	Continua	Continua
	Interior y con movimiento	Continua	Parada de protección
	Interior, y en parada monitorizada de seguridad	Continua	Continua

Tabla 6. *Tabla de la verdad para operaciones con paro controlado de seguridad.* [29]

Guiado Manual

La programación de los comandos de movimiento del robot se realiza con un dispositivo manual acorde con la normativa ISO 10218-1. Este dispositivo debe contener un pulsador de parada de emergencia a excepción de cumplir una serie de requisitos.

Control monitorizado de la velocidad y separación

En este caso se dota al robot colaborativo de sensores en tiempo real, con los que poder controlar la distancia robot-operario y la propia velocidad del robot. De manera que cuanto más cerca este el operario del robot, la velocidad de este será menor, hasta llegar a detenerse por completo.

Limitación de fuerza y potencia

Para prevenir múltiples riesgos hacia el operario como puedan ser golpes con partes móviles de geometrías o materiales peligrosos, se regula la fuerza, el par y la potencia de estos cobots.

La normativa ISO/TS 15066 recoge una tabla de umbrales de dolor para las distintas zonas del cuerpo, que provienen de una serie de ensayos que realizó la IFA-BGIA (*Institute for Occupational Safety and Health of the German Social Accident Insurance*).

Según Miguel Mariscal [29] en su artículo Normativa de Robots Colaborativos (cobots) y su influencia en la Prevención de Riesgos Laborales, se consideran las normas principales a tener en cuenta en seguridad de máquina, todas las siguientes:

ISO 12100:2012 Seguridad de las máquinas. Principios generales para el diseño, evaluación del riesgo y reducción del riesgo.
ISO 10218-1 Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 1: Robots.
ISO 10218-2. Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 2: Sistemas robot e integración.
ISO 11161. Seguridad de las máquinas. Sistemas de fabricación integrados. Requisitos fundamentales.
ISO 13849-1 Seguridad de las máquinas: Partes de los sistemas de mando relativas a la seguridad. Parte 1. Principios generales de diseño.
ISO 13850:2016. Seguridad de las máquinas. Función de parada de emergencia. Principios para el diseño.
ISO 13851. Seguridad de las máquinas. Dispositivos de mando a dos manos. Aspectos funcionales y principios para el diseño.
ISO 13855:2010. Seguridad de las máquinas. Posicionamiento de los protectores con respecto a la velocidad de aproximación de partes del cuerpo humano.
IEC 61508-1 Seguridad funcional de los sistemas eléctricos/electrónicos/electrónicos programables relacionados con la seguridad. Parte 1: Requisitos generales.
IEC 62061. Functional safety of safety-related electrical, electronic and programmable electronic control systems.
IEC 60204-1:2007. Seguridad de las máquinas. Equipo eléctrico de las máquinas. Parte 1: Requisitos generales.

Figura 27. *Enumeración de las principales normas en la seguridad de las máquinas.*

Una vez investigada la normativa en relación a los entornos de los robots, el siguiente paso, como ya se comentó, es realizar un esfuerzo creativo para determinar cómo será dicho espacio de trabajo.

Para ello el primer elemento que se va a destacar es la protección a este espacio de trabajo, que se ha realizado por medio de un vallado en todo el perímetro exterior. Esto supone que los operarios no puedan acceder a la zona tan fácilmente, lo que evita posibles accidentes laborales.
Este perímetro contiene:

Elementos	Cantidad
Barreras Protectoras	5
Puertas	2
Cámaras Vigilancia	2
Pared (Conveyor Doble)	1
Pared (Conveyor Simple)	1

Tabla 7. *Componentes perímetro célula robótica.*

A continuación, se explica la fase diseño de cada elemento.

CAPÍTULO 4.2.2. BARRERAS PROTECTORAS

Se ha creado un modelo de barrera de estilo clásico, pero elegante, en el que se ha buscado la simetría y una estructura cúbica repetitiva. De manera que la protección que ofrece es alta, a la vez que mantiene un diseño compacto.

Es un modelo que permite ver parcialmente, lo que sucede en el otro lado de la barrera, factor muy interesante en este caso, para que cualquier operario supervise y monitorice el funcionamiento de forma presencial. Además, las instalaciones disponen de cámaras en las esquinas, para poder monitorizar el correcto funcionamiento desde un puesto de trabajo, evitando de esta manera tener que estar presente para la supervisión del trabajo.

Solo se ha utilizado un material para la construcción de la barrera metálica.

La instalación de este componente es muy sencilla, ya que cuenta con cilindros laterales, de manera que se puedan acoplar unos componentes con otros y se pueden introducir en el suelo, ya que será el soporte de las barreras.

Las dimensiones de este elemento de pueden visualizar en la siguiente imagen (Véase *Figura 28*):

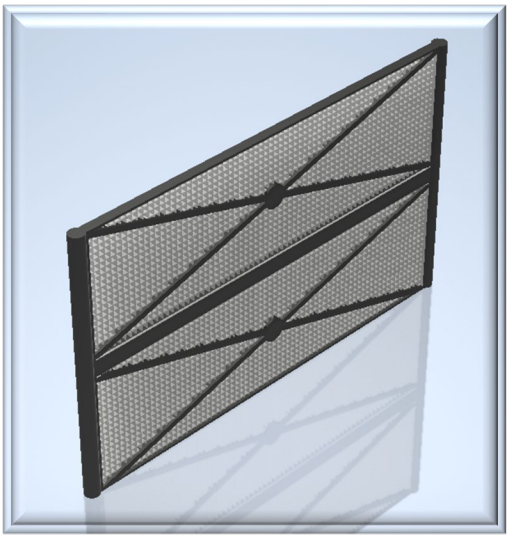


Figura 28. *Primera parte de la valla diseñada por el autor del proyecto R. Sancho.*

Adicionalmente a la barrera se ha añadido un elemento reflectante y llamativo, para indicar de forma inequívoca la zona de trabajo.

Este elemento se dispone en el suelo, y tiene unos pocos decímetros de altura, pero suficiente para poder ser visualizado desde largas distancias, evitando de esta manera cualquier tipo de accidente.

La geometría es muy sencilla ya que consta de dos barras de pequeña longitud verticales, sobre las que se encuentra una barra horizontal a través de un empalme a 90º.

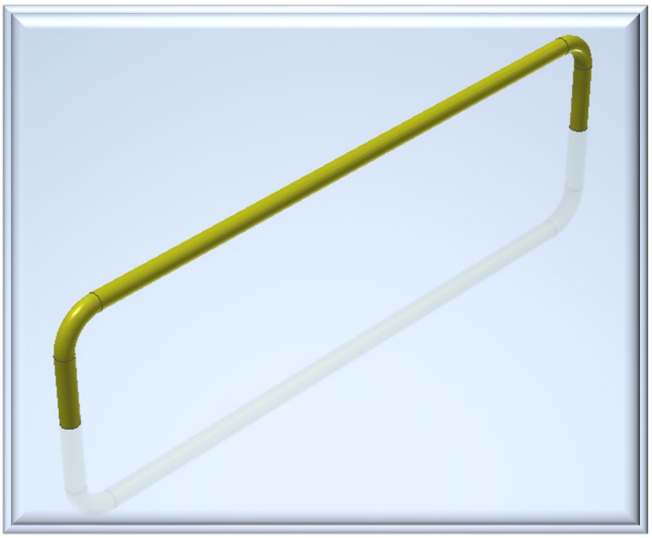


Figura 29. *Segunda parte de la valla diseñada por el autor del proyecto R. Sancho.*

El elemento final que se obtiene tras la combinación de ambos componentes es:

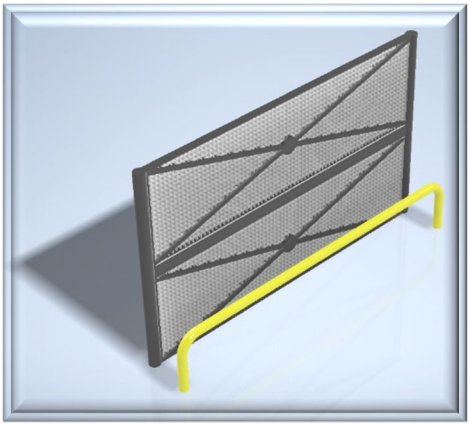


Figura 30. *Valla diseñada por el autor del proyecto R. Sancho.*

CAPÍTULO 4.2.3. ACCESOS INSTALACIONES

El siguiente elemento que se comentará será el acceso a la zona delimitada por las barreras protectoras, que en este caso se realiza por medio de una puerta frontal y una puerta trasera. Ambos accesos cuentan con el mismo diseño, por lo que solo se muestra uno de ellos.

La metodología de diseño ha sido muy similar al caso de las barreras protectoras, por lo que en este componente se siguen mostrando patrones repetitivos cuadrados y el espacio delimitado en franjas gracias a la utilización de barras más gruesas para proporcionar mayor estabilidad y robustez a la puerta.

En cuanto al método de acople, al igual que en las barreras, es por medio de unos cilindros en los laterales.

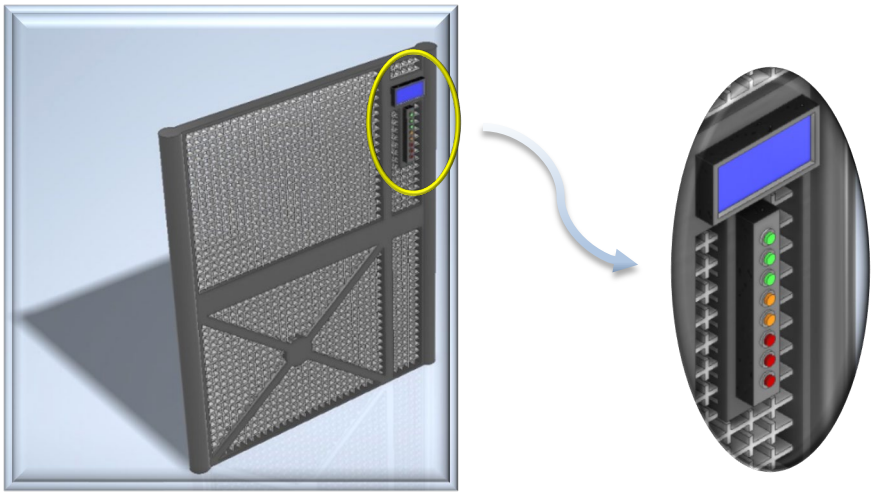


Figura 31. *Puerta de acceso al perímetro diseñada por el autor del proyecto R. Sancho.*

Como se había mencionado anteriormente el control de acceso consta de una pantalla LCD y una serie de pulsadores, para acceder y navegar por el menú del que dispone esta pantalla.

Se trata de una entrada configurable y programable, para que el operario cualificado encargado de la supervisión y mantenimiento, sea la única persona capaz de acceder.

CAPÍTULO 4.2.4. CÁMARAS VIGILANCIA

Como se ha comentado previamente, las instalaciones cuentan con una serie de cámaras que monitorizan en todo momento el funcionamiento de la célula.

Debido a la distribución de la célula se consideró que eran necesarias 2 cámaras para abarcar la zona de acción robótica al completo.

El diseño se basa en una forma cilíndrica, con una pequeña visera para evitar problemas relacionados con la luz incidente en la cámara, y a su vez, cuenta con una serie de leds infrarrojos que permiten la visualización del espacio de trabajo a pesar de que el sitio se encuentre a oscuras.

El modelo diseñado se puede visualizar en la siguiente imagen (véase *Figura 32)*.

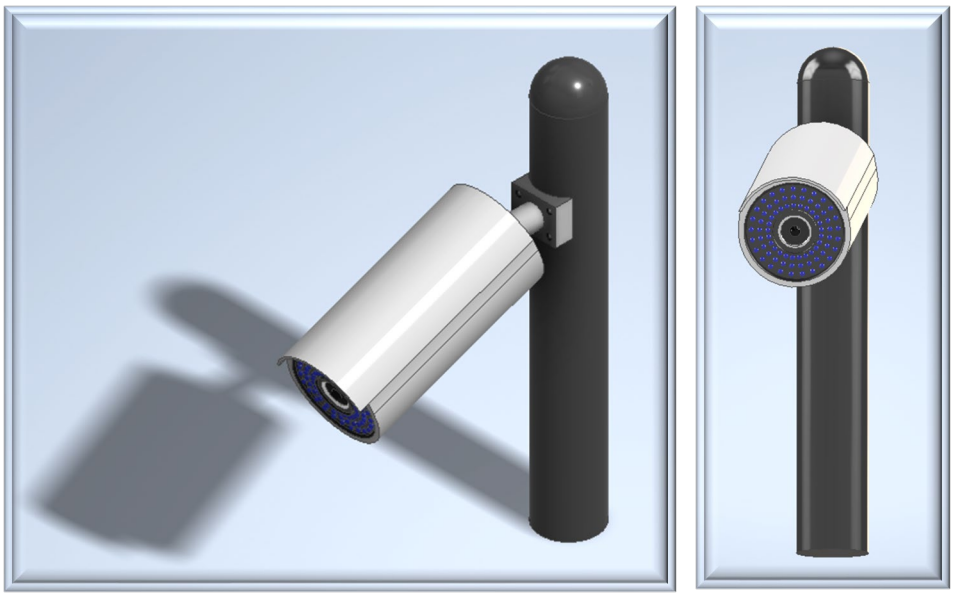


Figura 32. *Cámara de vigilancia diseñada por el autor del proyecto R. Sancho.*

CAPÍTULO 4.2.5. CONVEYOR DOBLE

Para el transporte y distribución de las piezas, sobre las que los robots realizaran posteriormente las acciones correspondientes, se tiene una cinta transportadora en forma de 'U'. De manera que las piezas incorrectas o defectuosas se puedan devolver al lugar de origen.

Se ha decidido denominar por el nombre de 'Conveyor Doble', por que consta en realidad de dos partes diferenciadas y con motores independientes.

- *Primera Parte*
  Encargada del transporte de las piezas hasta la zona de detección (Visión Artificial) y zona de manipulación del Robot IRB120. El recorrido es en línea recta.

- *Segunda Parte*
  Esta segunda parte se ocupa del transporte de las piezas que han sido descartadas, una vez se ha procesado la pieza con visión artificial. Para ello se realiza parte del recorrido mediante una curva de 180º y la parte restante se trata de una línea recta.

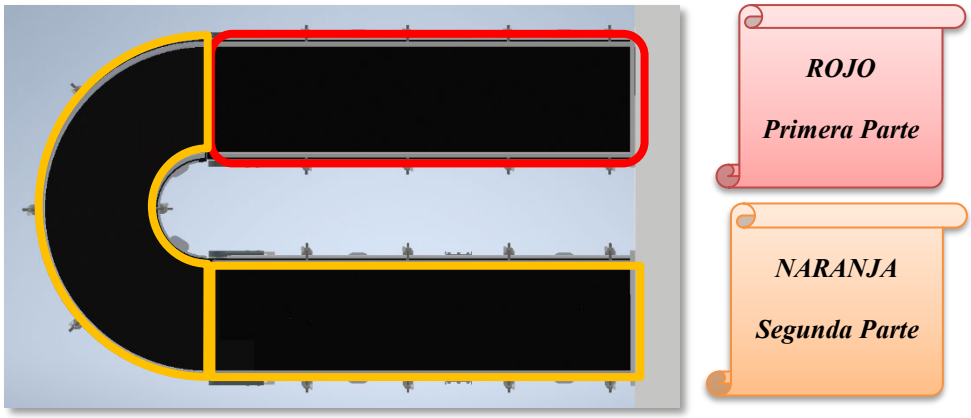


Figura 33. *Explicación de las partes en las que se divide el conveyor doble.*

Para el diseño de este elemento se tienen un total de 591 piezas, que, tras realizar el ensamblaje correspondiente, dan lugar al conveyor doble. Debido a la extensión que supondría la revisión de todas las piezas, se van a exponer únicamente los subgrupos y los componentes más relevantes de este ensamblaje.

Soportes Cinta

En el interior de la cinta, se encuentran unos perfiles metálicos, con la forma mostrada en la *Figura 34*, que tienen como finalidad fijar y sostener la estructura del conveyor.

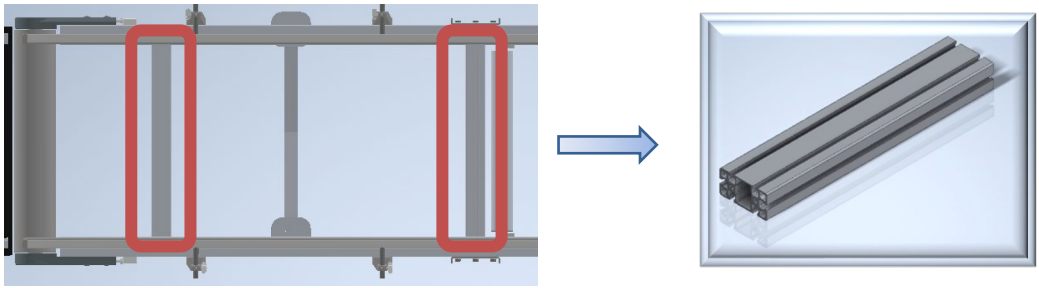


Figura 34. *Detalle de los soportes de las cintas transportadoras en el conveyor.*

Laterales

En los laterales de la estructura que forma el 'conveyor' se puede encontrar el mismo perfil anterior, pero ahora de una extensión mucho más considerable.

El perfil contiene dos huecos por cada lado, de manera que se pueden acoplar diferentes componentes en los laterales de esta estructura

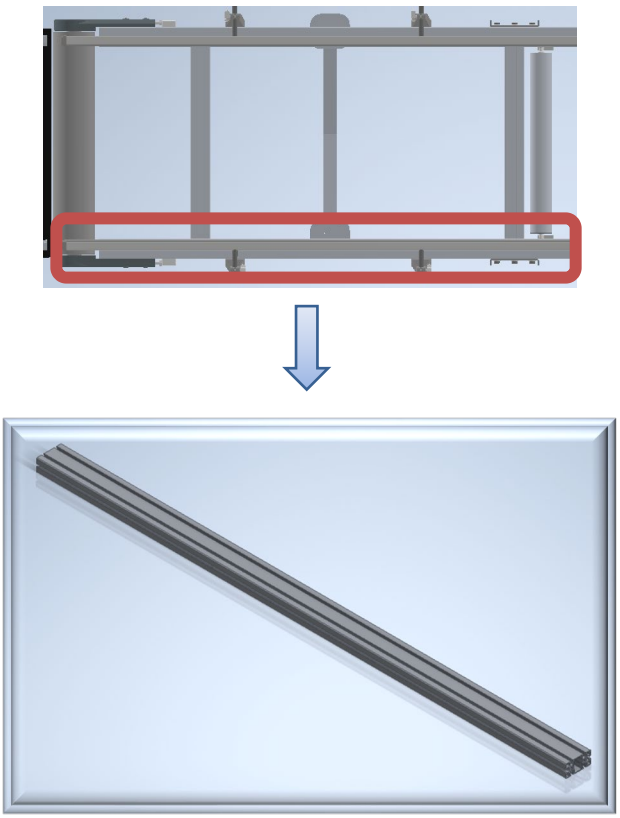


Figura 35. *Detalle de los soportes laterales del conveyor.*

Poleas y Rodillos Cinta

Para permitir el giro de la cinta transportadora, se tiene una polea terminal en el extremo de esta, de manera que permite el movimiento y evita rozamientos durante el funcionamiento.

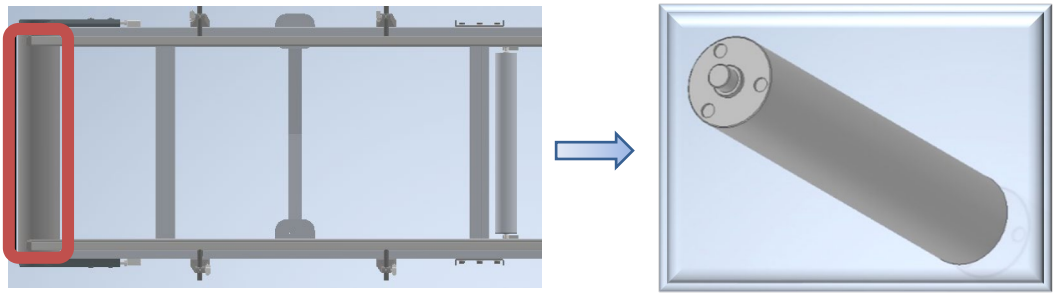


Figura 36. *Detalle del rodillo utilizado en el conveyor.*

La polea se acopla a rodamientos situados en los laterales, que permiten el giro de este componente. Para mejorar el aspecto visual, se han añadido embellecedores como se muestra en la siguiente *Figura 37*:

Figura 37. *Detalle de los embellecedores usados en el conveyor.*

En el lado del motor (extremo restante) se tiene la polea motriz, ya que en este caso el rodillo contiene un enganche para acoplarse al motor y poder ser accionado por este.

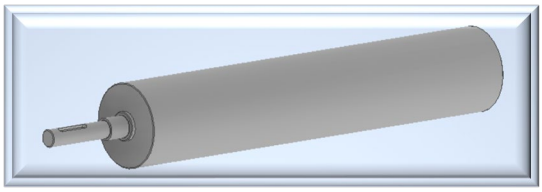


Figura 38. *Detalle de la polea motriz del conveyor.*

Finalmente, existen rodillos más pequeños equiespaciados durante el recorrido de la cinta, que permiten el movimiento y sujeción de la cinta.

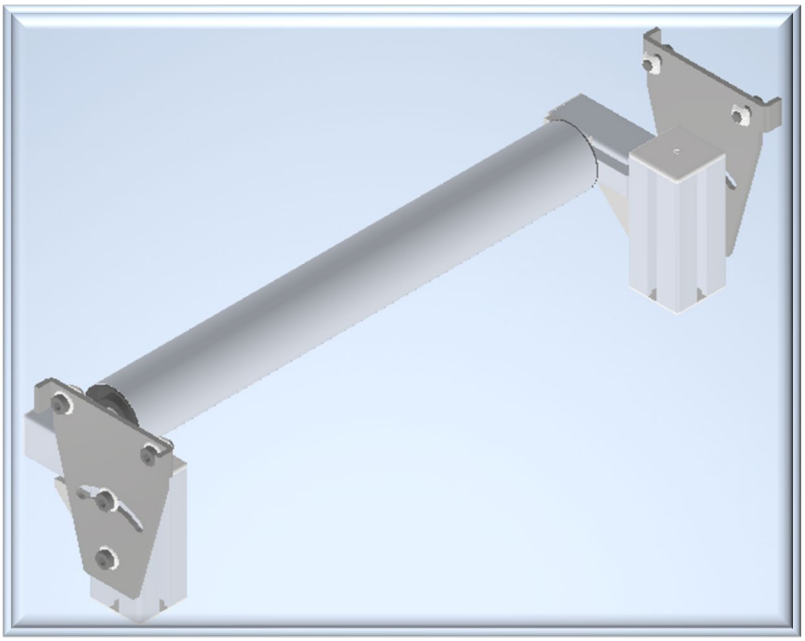


Figura 39. *Rodillos intermedios del conveyor.*

Motor

El accionamiento de la cinta transportadora se realiza por medio de un motor en el cual se acopla el eje de uno de los rodamientos exteriores. Este motor no se encuentra visible en el ensamblaje final, ya que está dentro de una cabina situada en las paredes

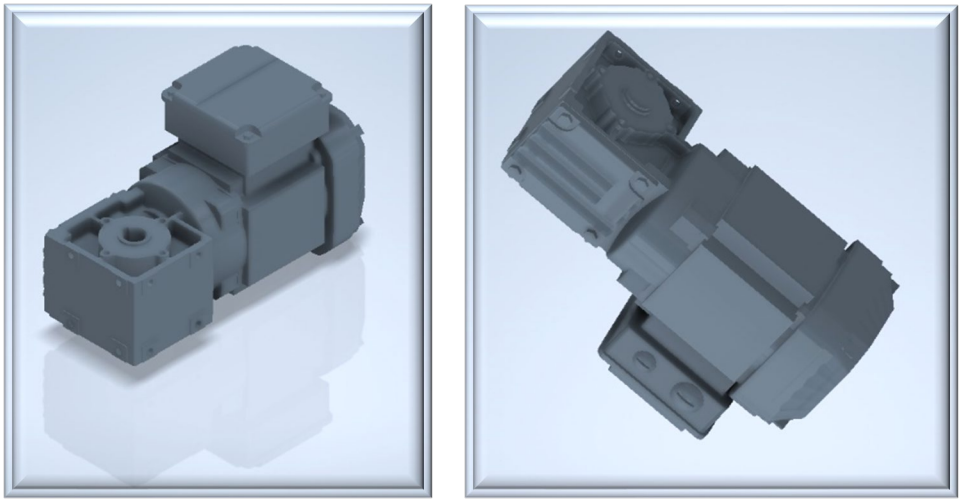


Figura 40. *Modelo de motor usado para desplazar la cinta transportadora.*

Soportes y Barrera protectora

Durante todo el recorrido de la cinta transportadora se encuentran una serie de soportes para la barrera protectora que se encuentra en la parte superior, para evitar que los objetos transportados se puedan desviar, por cualquier motivo, fuera.

El soporte contiene dos tipos de agarres:

- Superior: Encargado de la sujeción de la barrera protectora
- Inferior: Son dos acoples, diseñados con el objetivo de encajar en el perfil lateral del 'conveyor'.

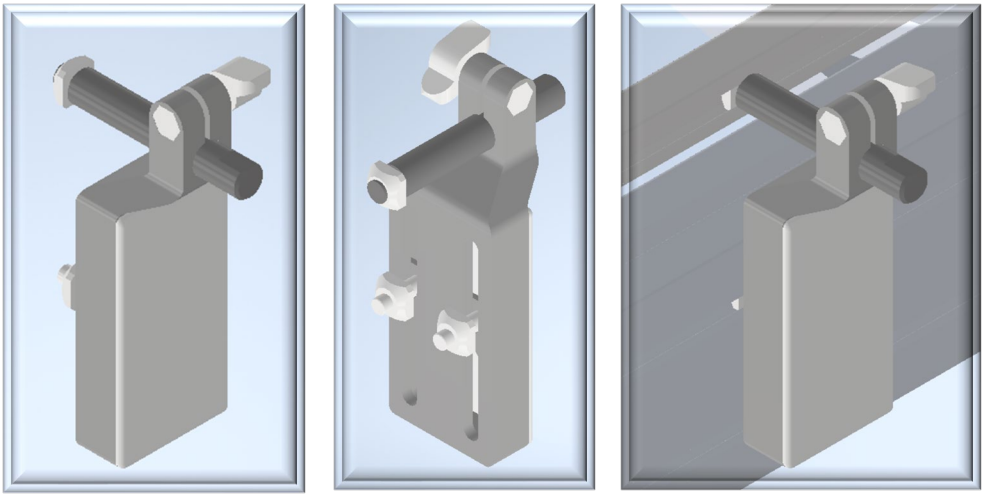


Figura 41. *Detalle de los soportes de la barrera protectora en el conveyor.*

En cuanto a la barrera se pueden dividir dos partes fundamentales, el perfil de acero y la parte acolchada del interior.

El perfil está diseñado para acoplarse con los soportes anteriormente mencionados, de manera que mantengan dicha barrera. Además, sirve para acoplar una semicircunferencia de material blando, que se dispondría en la parte interior del 'conveyor', con esto se consigue evitar a las piezas que establezcan contacto con la barrera, posibles deformaciones o rozaduras.

Figura 42. *Detalle de la barrera protectora en el conveyor.*

Base conveyor

Para el soporte de toda la estructura se han utilizado una serie de apoyos circulares distribuidos a lo largo de esta.

Además de los apoyos con forma cónica, se tiene una estructura que se acopla al perfil lateral del conveyor y le sirve de apoyo. Existen conos a ambos lados para transmitir el peso de una manera uniforme entre estos soportes y están conectados entre sí por unas barras metálicas.

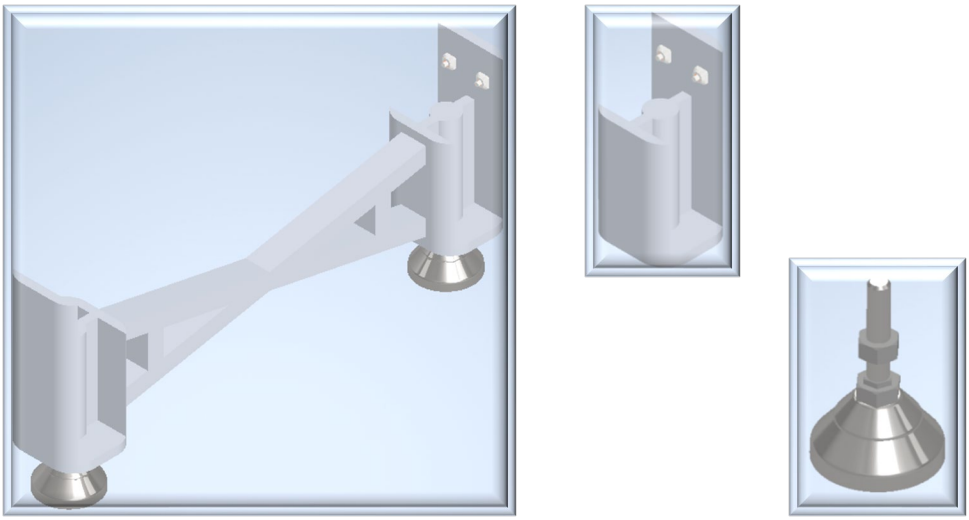


Figura 43. *Base/Soportes del conveyor.*

Banda transportadora

Este es uno de los elementos fundamentales del 'Conveyor', ya que es donde se apoyan los componentes que serán transportados. Por lo tanto, si fuese a desarrollarse en una situación real, tendría que ser de un material resistente. En la industria estos elementos se fabrican de diversos materiales en función de las características deseadas pero los más importantes son:

Materiales	Ejemplos
Termoplásticos	Poliéster, cloruro de polivinilo, silicona y polietileno
Metales	Acero inoxidable, aluminio y acero al carbón
Caucho	Natural y Sintético
Tela	Algodón y Lona
Cuero	Cuero

Tabla 8. *Materiales más empleados en la fabricación de cintas transportadoras.* [33]

El 'conveyor' en 'U' se forma con dos bandas rectas y una banda semicircular como las mostradas en las siguientes imágenes.

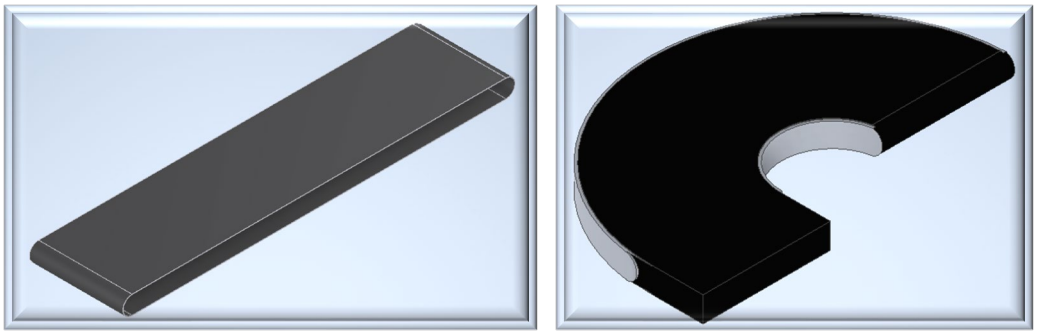


Figura 44. *Cintas transportadoras usadas en el conveyor.*

Tras esta recopilación de componentes en los que se han mostrado las partes más importantes, a continuación, se enseña como es el resultado de la creación del conveyor en su conjunto.

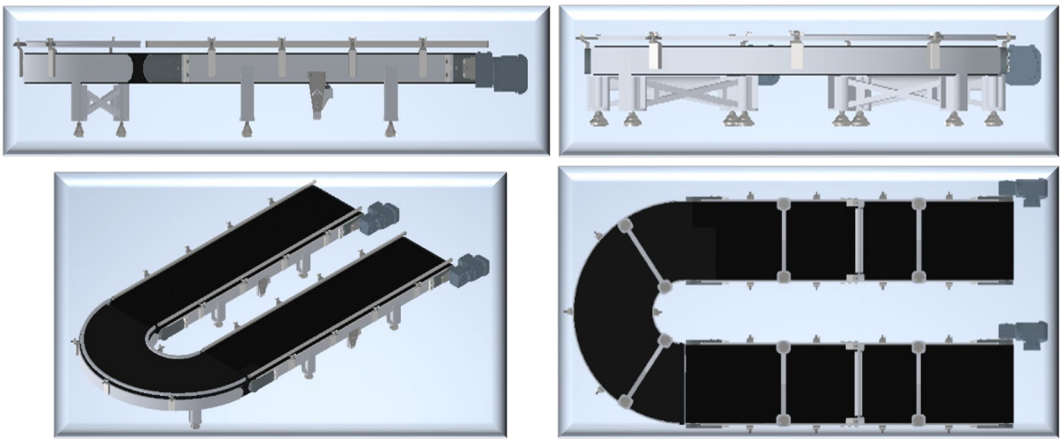


Figura 45. *Vistas (Lateral, frontal y superior) y perspectiva isométrica conveyor.*

Lógicamente, las piezas tienen que llegar de alguna manera a la cinta transportadora, por lo que adicionalmente se ha diseñado una pequeña estructura simulando un edificio con dos accesos para las piezas.

Estas piezas se supone que llegarían de una fase anterior de la fabricación que pertenecería a otro sector y, por tanto, carece de interés en este proyecto, lo que ha originado que no se represente la fase anterior de fabricación de piezas.

Este edificio cuenta con una parte superior vacía, en la que posteriormente se colocará una vitrina con las piezas diseñadas del coche de juguete.

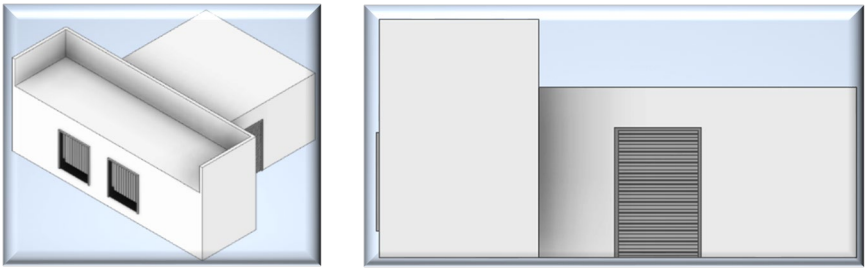


Figura 46. *Edificio diseñado para el conveyor doble.*

Adicionalmente se han dispuesto una serie de carteles de advertencia, debido al riesgo que supone este tipo de instalaciones para cualquier operario que intente manipular cualquier elemento, sin los conocimientos previos o la cautela adecuada.

Figura 47. *Carteles de advertencia dispuestos en el conveyor doble.*

Y con estos últimos componentes ya se habría completado el diseño del conveyor doble que tiene la siguiente forma:

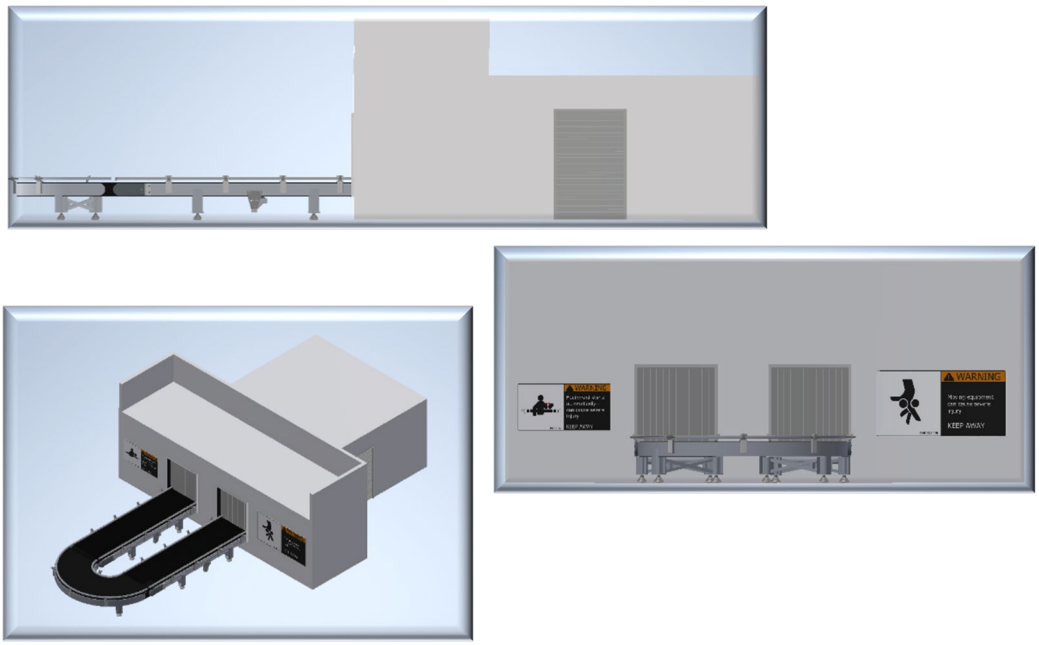


Figura 48. *Vistas generales del conjunto conveyor-edificio.*

Las cantidades de los subconjuntos desarrollados anteriormente se desglosan en la siguiente tabla:

Componente		Cantidad	N.º Piezas/Conjunto
Soportes Cintas		6	1
Laterales	Rectos	4	1
	Semicirculares	2	1
Poleas		4	1
Rodillos		6	52
Embellecedores		4	8
Motor		2	1
Soportes		20	7
Barreras Protectoras	Rectas	4	5
	Semicirculares	2	2
Bases		8	7
Bandas Transportadoras	Rectas	2	1
	Semicirculares	1	1
Edificio		1	1
Puertas		4	1
Carteles		2	1
TOTAL		67	592 (Piezas Totales)

Tabla 9. *Desglose de componentes del conveyor doble.*

Una vez revisado el montaje del componente con el mayor número de piezas diseñado para este proyecto, se va a proceder a analizar el montaje de la segunda cinta transportadora necesaria para este proyecto.

CAPÍTULO 4.2.6. CONVEYOR SIMPLE

En este caso la idea es muy similar al anterior diseño y la mayoría de los componentes son iguales, por lo que solo se mostrará el diseño final y aquellos componentes que difieren respeto al 'conveyor doble'.

Para la señalización del posible peligro de esta instalación, se ha utilizado un cartel de advertencia diferente a los anteriores. En este caso se utiliza una etiqueta de seguridad de la Asociación de Fabricantes de Equipos Transportadores (CEMA), la cual tiene el código CHR930004.

En esta se advierte el peligro que supondría escalar, sentarse o andar en una cinta transportadora en movimiento.

Figura 49. *Cartel de advertencia dispuesto en el conveyor simple.*

El otro elemento que se va a destacar es el edificio para este segundo conveyor. Ahora se tiene una única salida de piezas, y una única puerta lateral, para acceder al interior.

Al igual que en el otro edificio diseñado, se dispone en el interior de una zona hueca en la que se situará el motor de la cinta transportadora.

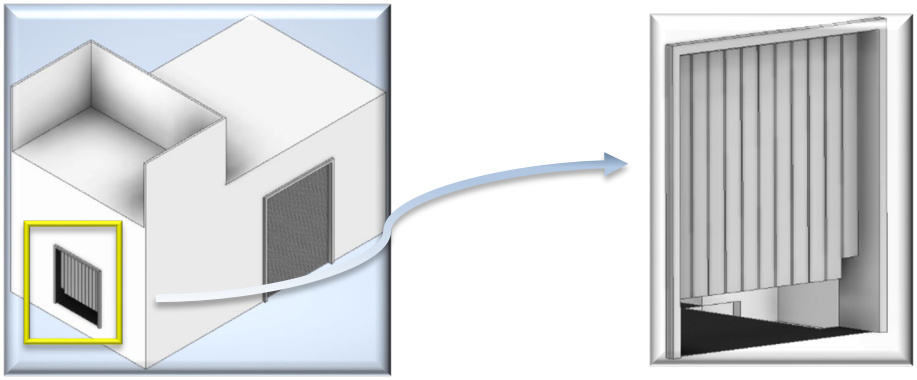


Figura 50. *Edificio diseñado para el conveyor simple.*

El elemento resultante de ensamblar estos dos componentes, con la cinta transportadora se puede visualizar en las siguientes imágenes:

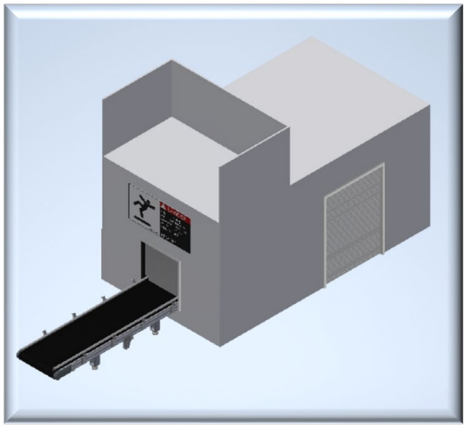


Figura 51. *Vista Isométrica del conveyor simple*

En este ensamblaje se encuentran un menor número de componentes, pero de igual manera sigue siendo un número destacable, por lo que se hace el desglose de piezas en la tabla siguiente:

Componente	Cantidad	N.º Piezas/Conjunto
Soportes Cintas	2	1
Laterales Rectos	2	1
Poleas	2	1
Rodillos	2	52
Embellecedores	2	8
Motor	1	1
Soportes	8	7
Barreras Protectoras	2	5
Bases	2	7
Bandas Transportadoras	1	1
Edificio	1	1
Puertas	2	1
Carteles	1	1
TOTAL	28	212 (Piezas Totales)

Tabla 10. *Desglose de componentes del conveyor simple.*

CAPÍTULO 4.2.7. SETA DE EMERGENCIA

Al disponer de tantos elementos móviles que pueden ocasionar accidentes graves en los operarios, se debe poseer una serie de seguridades de tipo hardware como son los interruptores de parada de emergencia (setas de emergencia).

En cuanto a su disposición, estos elementos estarán repartidos en la zona de trabajo, cerca de la zona que detienen en caso de accionarlos. Además, se han añadido dos interruptores de emergencia, en las inmediaciones del recinto, para que sea posible detener todos los elementos de la célula desde el exterior de esta, con la mayor rapidez posible.

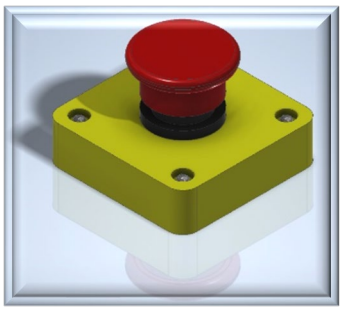


Figura 52. *Seta de emergencia diseñada por el autor del proyecto R. Sancho*

CAPÍTULO 4.2.8. SOPORTE CÁMARA VISIÓN ARTIFICIAL ABB

Como este proyecto incluye una parte de análisis de piezas, apoyándose en las imágenes tomadas de una cámara para un posterior procesado, se necesita disponer en algún lugar del espacio de trabajo esta cámara.

Al ser la zona que se quiere detectar una vista aérea del espacio de trabajo del robot IRB120 y el conveyor doble, se ha diseñado y añadido un soporte, de manera que la cámara se posicione en el lugar deseado.

Para el diseño de este soporte, la inspiración ha venido por parte de la propia compañía ABB, ya que, al visitar su página, y más concretamente la cámara de visión artificial que ofertan, también muestra en un video un soporte para dicha cámara.

Por lo que se ha buscado aproximarse en el diseño lo máximo posible a este soporte ya existente en la realidad. Este soporte se puede visualizar en las siguientes imágenes tomadas de la presentación de visión artificial en el canal oficial de youtube de 'ABB Robotics' (véase *Figura 53*).

Figura 53. *Capturas tomadas del video oficial de visión artificial de ABB Robotics.* [34]

El elemento diseñado en Autodesk Inventor buscando la similitud con este elemento sería:

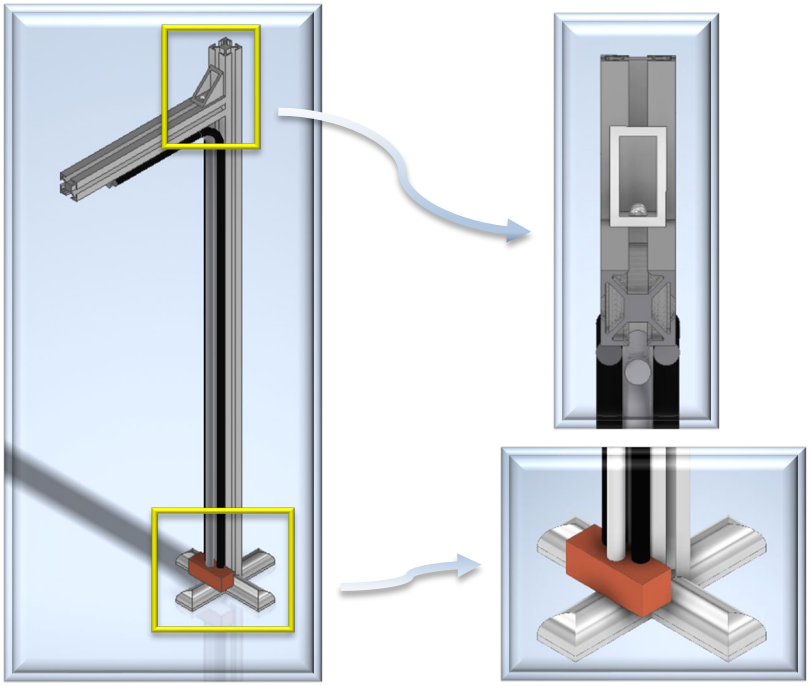


Figura 54. *Soporte cámara visión artificial modelo ABB diseñado por R. Sancho, en base al modelo existente de la compañía.*

CAPÍTULO 4.2.9. ELEMENTO DE ALARMA AUDITIVO Y COLUMNA DE SEÑALIZACIÓN

Como se había revisado en el '*Capítulo 4.2.1. Normativa Entorno de Trabajo de los Robots*', se deben proveer a las instalaciones de elementos advirtiendo de un posible fallo o peligro. Por ello se utilizan altavoces y luces de emergencia, para que el peligro sea reconocido, tanto de forma visual como auditiva.

Buscando la similitud con los elementos que existen actualmente en la industria se han diseñado unas luces de señalización cónicas, compuestas de diferentes colores, para advertir al operario con un mensaje de estado inequívoco.

Adicionalmente, para amplificar el sonido de alarma, se han dispuesto a 120º 3 altavoces.

El diseño se muestra en la siguiente imagen (véase *Figura 55*)

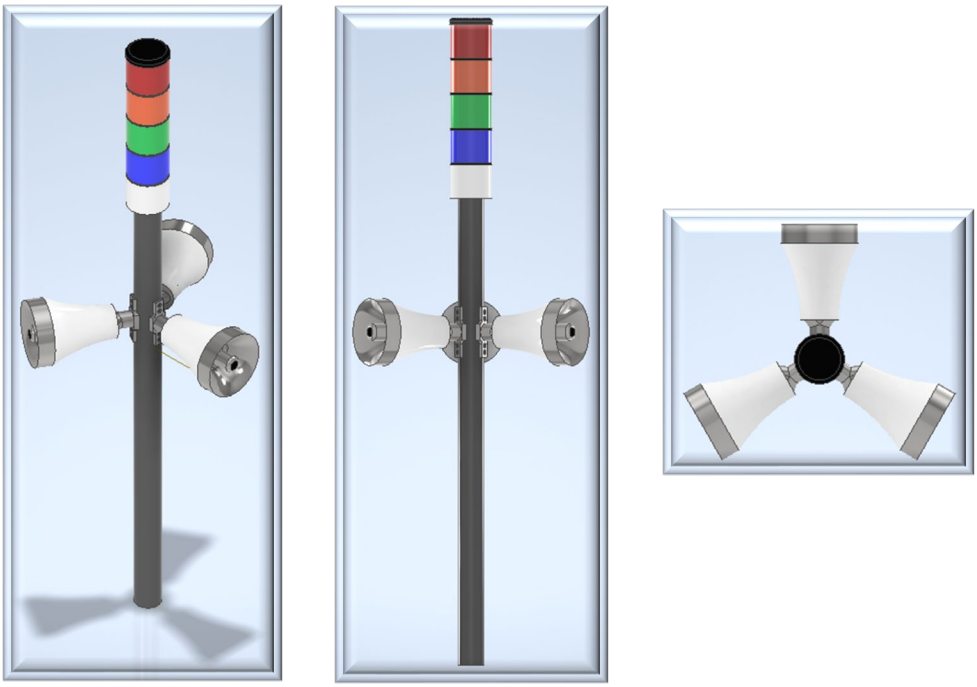


Figura 55. *Combinación de elementos de alarma auditivos y columna de señalización diseñado por el autor del proyecto R. Sancho.*

Ahora solo restaría añadir la cámara de ABB, que esa se obtendrá de la propia aplicación de la compañía 'RobotStudio'.

La finalidad de todos estos diseños es la de incluirlos en el software de simulación de los robots, por lo que si ya están incluidos en la biblioteca de este programa se ahorra una gran cantidad de tiempo y esfuerzo.

Los componentes diseñados con una aplicación externa a RobotStudio y que se desea incorporar en este programa tendrán que guardarse como un archivo '.sat' para poder ser reconocidos correctamente por este.

Por lo tanto, se necesita saber *¿Qué es un archivo con extensión SAT?*

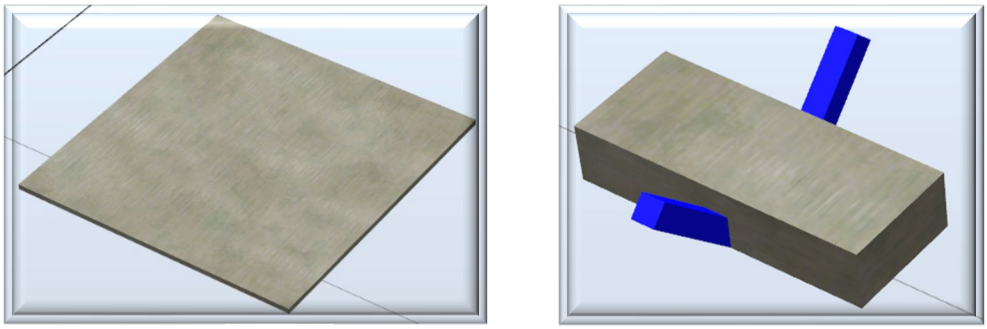
[35] Los autores de 'AbrirArchivos' definen este tipo de archivos como

"Modelo 3D guardado en el formato de modelado sólido ACIS de Spatial; almacena información de geometría tridimensional en un formato de archivo de texto estándar; se usa para intercambiar datos en 3D entre varios sistemas y es compatible con muchos programas de CAD en 3D que incluyen el componente 3D ACIS Modeler de Spatial."

CAPÍTULO 4.2.10. ELEMENTOS ROBOTSTUDIO

Todos componentes anteriores, se han diseñado en Autodesk Inventor, pero en modelos más simples RobotStudio permite a su vez, la creación y diseño de piezas. Por lo que con esta herramienta se han diseñado varios componentes que integran el espacio de trabajo del robot YuMi, como son, la mesa de trabajo y el bloque de trabajo para el montaje del coche.

Ambos componentes se han realizado con metal pulido (metal con un acabado satinado unidireccional). Adicionalmente, el bloque de trabajo contiene dos elementos rociados por una pintura azul, diseñados específicamente para que el robot de ABB YuMi pueda agarrar y manipular este componente con sus pinzas (herramientas dispuestas a final del brazo).



A) ***B)***

Figura 56. A) *Mesa de trabajo robot IRB14000*
B) *Bloque de trabajo para el manejo del automóvil de juguete.*

Tras la revisión de todos los elementos que se han tenido que diseñar para la creación de la célula de trabajo, cabe destacar que no son los únicos elementos que conforman dicha zona. Pero los restantes, ya se encontraban prediseñados en el software de ABB 'RobotStudio', por lo que simplemente se han añadido de la biblioteca de este programa.

Para ello hay que dirigirse al directorio en *HOME/Import Library/Equipment*, donde se dispone de diferentes elementos.

En el transporte de las piezas seleccionadas por el robot IRB120, con destino a los brazos del robot YuMi, se utilizan 'conveyors' prediseñados por ABB. Como este elemento tiene una relación de tamaños considerablemente mayor a los robots utilizados para este proyecto, hay que utilizar una herramienta del software 'RobotStudio', en la que se permite escalar los componentes.

Gracias a dicha herramienta, aplicando una escala de 18/100, se obtiene un tamaño optimo en relación a los elementos de la célula de trabajo.

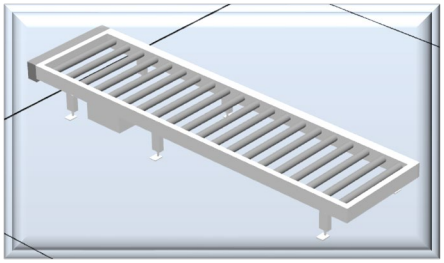
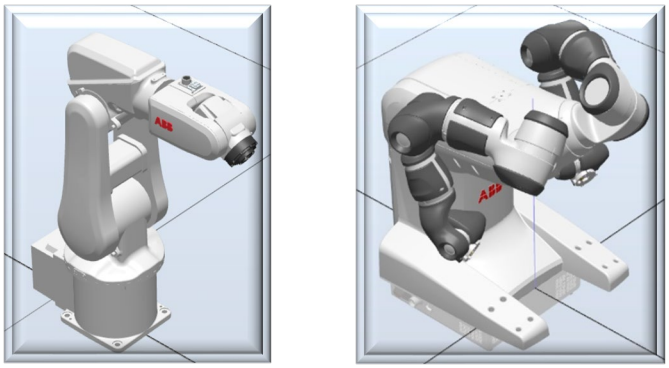


Figura 57. *Conveyor prediseñado por ABB Robotics en el software RobotStudio.*

Además, se ha añadido un armario de control de ABB, encargado de simular todo el hardware que manipularía los robots. En este caso no se ha buscado la exactitud con la realidad, ya que se necesitarían diferentes módulos (control, manejo, proceso…), que no aportan nada a la célula de trabajo y a este proyecto, por lo que todo lo anterior se ha remplazado por este armario de ABB.

Figura 58. *Armario de control prediseñado por ABB Robotics en el software RobotStudio.*

El elemento fundamental de este proyecto son los robots de ABB utilizados, por lo que para añadirlos en la estación de trabajo hay que dirigirse al siguiente directorio *HOME/ABB Library.* Donde se seleccionarán el robot *IRB120* y el robot *IRB 14000* YuMi. Una vez seleccionados, aparecerá en el programa lo siguiente (véase *Figura 59*):



A) ***B)***

Figura 59. A) *Robot IRB120 en RobotStudio.*
B) *Robot IRB14000 (YuMi) en RobotStudio*

CAPÍTULO 4.2.11. DISEÑO FINAL

Una vez se han mostrado y analizado todos los elementos de la célula de trabajo, se expone a continuación la distribución de estos, formando el espacio de trabajo en el que se desarrolla todo el montaje y detección de piezas que ocupa este proyecto.

En las siguientes imágenes se mostrará, las diferentes vistas que tiene dicha célula y se verán elementos adicionales como son las diferentes piezas que componen el coche de juguete o los diferentes tamaños de cubos (elementos expuestos en las vitrinas), que no se han comentado, ya que su análisis se procederá a realizar en el próximo *capítulo 4.3. Diseño de las Piezas y Herramientas.*

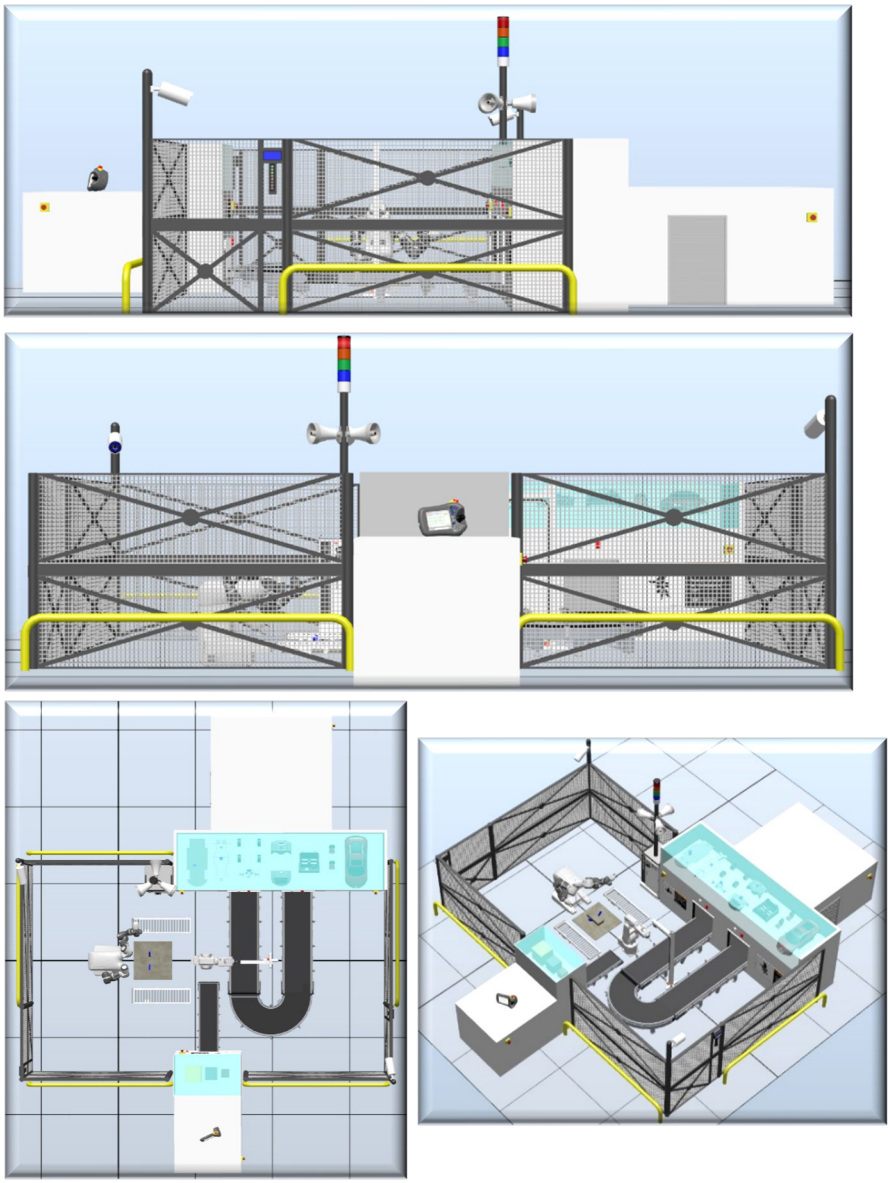


Figura 60. *Célula de trabajo montada en RobotStudio*

Durante la simulación los únicos elementos que interactúan y cambian su color, forma o permiten mover otros elementos son:

- Robots (IRB 120, IRB 14000 YuMi)
- Conveyors (Doble, Simple, Conveyors ABB)
- Luces Conveyors
- Cámaras Vigilancia

Los demás elementos son estáticos y no se verán modificados.

CAPÍTULO 4.3. DISEÑO DE LAS PIEZAS Y HERRAMIENTAS

Una vez se han analizado todos los elementos que se han necesito implementar en la creación de la célula de trabajo, el siguiente paso es analizar las piezas que se han diseñado para la construcción del coche deportivo de juguete y los diferentes cubos a detectar por visión artificial.

Además, se comentaras las diferentes herramientas que se han utilizado para que los robots puedan realizar el montaje/selección de piezas.

CAPÍTULO 4.3.1. COCHE DEPORTIVO DE JUGUETE (R8)

El coche que se ha decidido crear está basado en el modelo R8 de la marca de coches AUDI (fabricante alemán). Dicho modelo es un automóvil de turismo de super lujo y alta tecnología, el cual está próximo al automovilismo de competición por las características que es capaz de desempeñar.

[36] El modelo real tiene una potencia de 419kW (570CV), alcanza una velocidad máxima de 324 Km/h y es capaz de pasar de 0 a 100Km en 3.4 segundos. Estas características se han añadido a modo de curiosidad, ya que estos valores no influyen en este proyecto (se han añadido para introducir al lector en el modelo diseñado) debido a que únicamente se va a simular el aspecto físico de la carrocería.

Con el diseño en Autodesk Inventor no se ha buscado la exactitud del modelo, sino que, por el contrario, solo se han añadido sus características y rasgos más relevantes. Ya que la mayoría de las piezas se han diseñado desde cero, para poder adaptarse a un montaje rápido y realizado por robots de tamaño reducido.

En la siguiente tabla se muestran las partes fundamentales que conforman el coche deportivo de juguete y que serán transportadas por los robots, para realizar el montaje.

Piezas		Cantidad
Parte Inferior		1
Chasis y Motor		1
Transmisión Ruedas Delanteras		1
Ruedas Delanteras		2
Ruedas Traseras		2
Interior	Parte Delantera (Capo)	1
	Parte Trasera	1
	Núcleo (Interior Coche)	1
Asientos (Conductor y Copiloto)		2
Carrocería		1

Tabla 11. *Componentes automóvil deportivo de juguete.*

A continuación, se mostrarán y comentarán cada grupo expuesto en la anterior tabla individualmente.

Parte Inferior

La primera pieza y sobre la que se acoplaran el resto de piezas, es una carcasa inferior, con la misma forma que la carrocería, para poder adaptarse a esta sin ningún problema.

Cuenta con 3 cilindros para poder acoplar el chasis a esta primera pieza. Esta conexión se realiza de manera instantánea como si se tratasen piezas de la famosa compañía de bricks de construcción de plástico, LEGO.

Adicionalmente tiene una estructura rectangular denotada con el color azul, que servirá para facilitar que el robot IRB 14000 (YuMi), pueda manipularlo con sus pinzas sin ningún problema.

En la siguiente imagen se muestra el resultado:

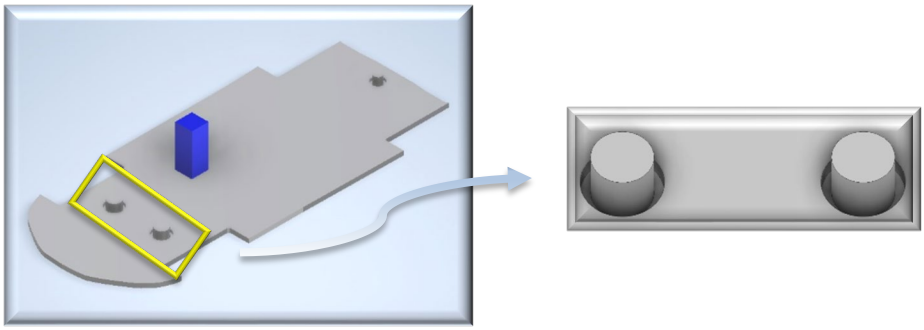


Figura 61. *Parte Inferior (Automóvil deportivo de juguete).*

Chasis y Motor

Para el chasis del coche, se ha realizado una estructura con perfil circular, con 3 ramas principales. En el punto central de la estructura se encuentra un cilindro barnizado con pintura azul, indicando que será la parte que manipulará el robot.

En la cara inferior del componente se encuentran 3 cilindros huecos, que encajan con los cilindros de la 'Parte Inferior'.

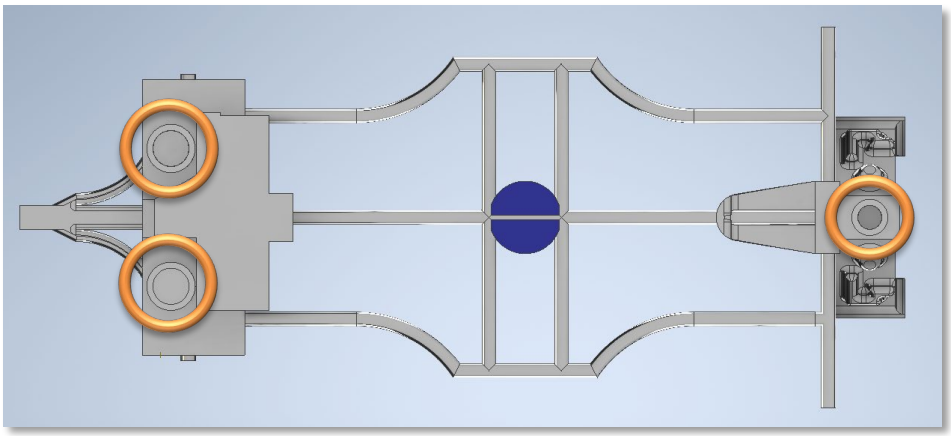


Figura 62. *Detalle enganches chasis (Automóvil deportivo de juguete).*

En la cara superior del componente se encuentra el resto de elementos y detalles de este. Centrando la atención en el frontal, destaca un elemento rectangular hueco, que servirá para acoplar la pieza del capo (posteriormente se comentará).

Retrocediendo desde el frontal el siguiente elemento que se puede encontrar es la base para la trasmisión de las ruedas delanteras del vehículo. Este elemento se ha diseñado de forma personalizada para adaptarse al 100% al diseño de la transmisión utilizada.

Como último elemento a destacar, se ha diseñado una estructura muy simplificada, con el fin de representar el motor del automóvil. Al no tener ninguna utilidad motriz en este coche de juguete, no se ha visto conveniente, realizar un diseño más complejo.

La única característica motriz de esta parte trasera, es el rodamiento en el centro de la estructura del motor, para poder permitir el movimiento del eje que posteriormente moverá las ruedas traseras.

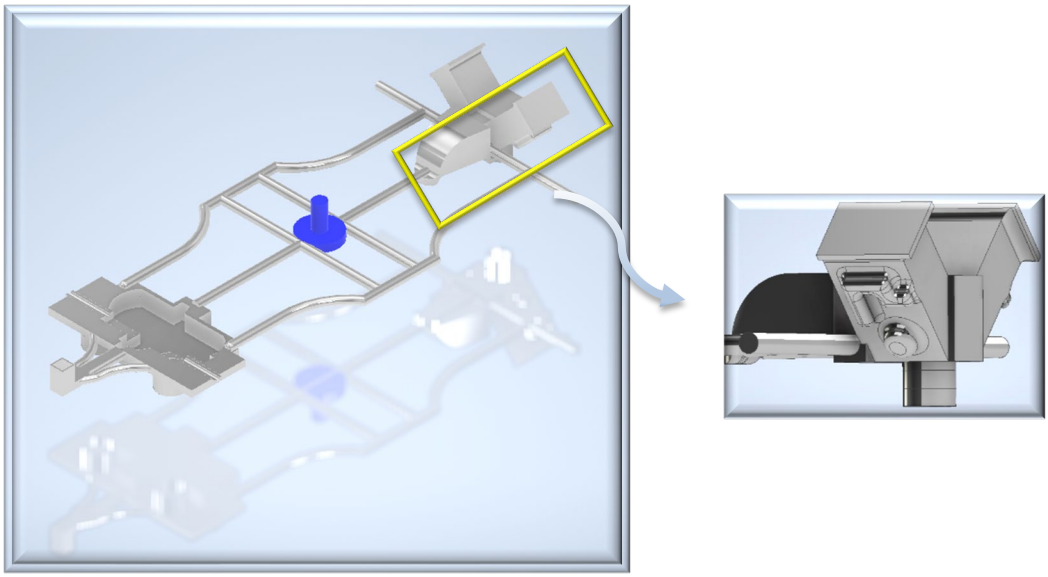


Figura 63. *Chasis (Automóvil deportivo de juguete).*

Transmisión Ruedas Delanteras

Este componente es uno de los elementos más complejos y mejor representados que contiene el coche deportivo de juguete. Con este elemento se ha buscado permitir el movimiento de las ruedas delanteras, tanto de avance y retroceso, como de direccionamiento del coche (derecha, centro o izquierda).

Al igual que en todas las piezas del coche, el elemento azul se ha añadido únicamente para poder manipular la pieza con los robots.

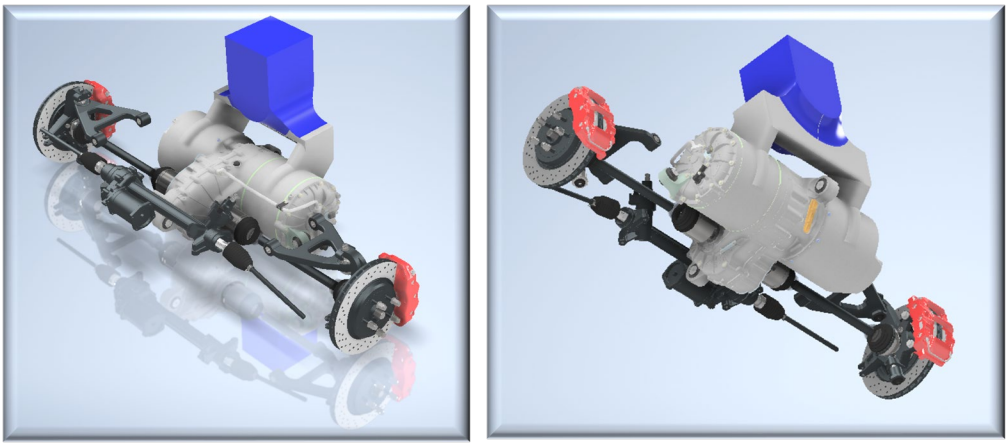


Figura 64. *Transmisión delantera (Automóvil deportivo de juguete).*

Ruedas Delanteras

Las 4 ruedas que componen el coche, se han diseñado siguiendo este modelo (véase *Figura 65*).

La llanta sigue un patrón octagonal y en su núcleo contiene una serie de cilindros y una serie de agujeros (patrón pentagonal) para acoplarse a los frenos de disco que han sido diseñados. En relación a su aspecto se ha buscado un aspecto de metal semipulido con un color de la gama del naranja.

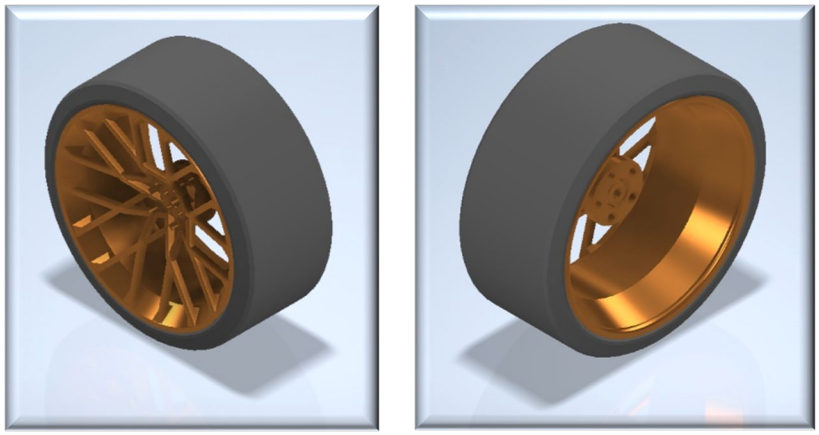


Figura 65. *Ruedas Delanteras (Automóvil deportivo de juguete).*

Ruedas Traseras

Las ruedas traseras siguen el mismo modelo que las 'Ruedas Delanteras', pero en este caso contienen integrados los discos de freno. A su vez estos cuentan con un hueco cilíndrico, para acoplarse al eje motriz trasero del chasis.

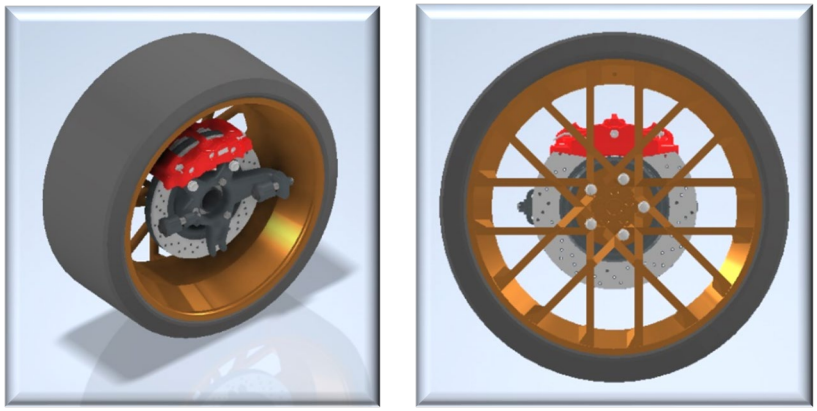


Figura 66. *Ruedas traseras (Automóvil deportivo de juguete).*

Interior

Con el objetivo de evitar dejar el interior del coche hueco, y aportar un poco más de realismo al coche, se han desarrollado 3 componentes, que en su conjunto componen la estructura interior de un coche biplaza.

Para ello se distingue lo siguiente:

- **Parte Delantera (Capo)**
  Contiene un espacio hueco ('maletero frontal'), sin ningún objetivo aparente, más que el de poder introducir pequeños elementos. Con esto se puede dar mayor versatilidad a la hora de ser utilizado por un niño que esté jugando con dicho coche.

  Se han incluido gran variedad de detalles para aportar mayor realismo al modelo.

  En la zona inferior se encuentra la pieza rectangular, que permite el acoplamiento al chasis como se había comentado previamente. Además, se incluyen una serie de agujeros en la zona trasera del componente, con el objetivo de adaptarse a la forma del chasis.

  La pieza de color azul, tiene tres funciones:

  - Estructura para manipular el componente entero con las pinzas del robot IRB 14000
  - Acoplamiento con la estructura rectangular azul de la transmisión
  - Acoplamiento con la carrocería superior

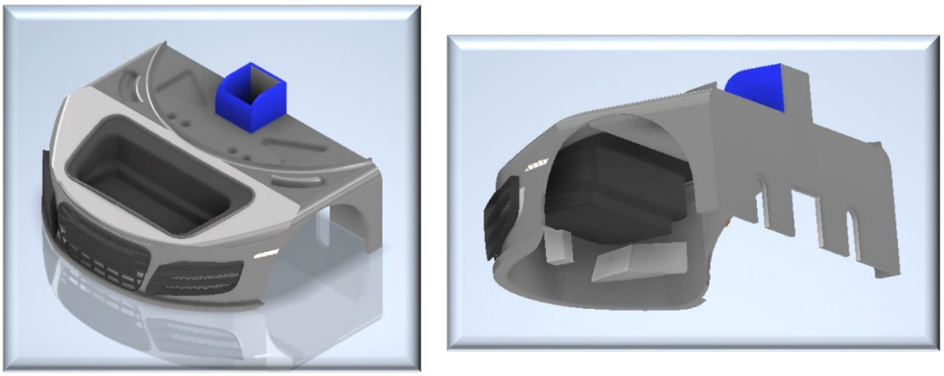


Figura 67. *Capo (Automóvil deportivo de juguete).*

- **Parte Trasera**

  Este componente simula la parte trasera del coche, en la que se encuentran las luces, un logotipo de la marca de coches utilizada (AUDI), posee agujeros para los tubos de escape, y elementos auditivos como serían los altavoces traseros que posee el coche.

  Para adaptarse al chasis, dispone de una serie de agarres cilíndricos a ambos lados y un agujero central.

  La pieza azul para los robots, cuenta con un agujero que servirá para adaptar la carrocería.

Figura 68. *Parte Trasera (Automóvil deportivo de juguete).*

- **Núcleo (Interior Coche)**

  En el interior del coche se han añadido los elementos básicos que todo coche posee (volante, guantera, espejo retrovisor, altavoces laterales…).

  El acoplamiento en el chasis al igual que la parte trasera se realiza por medio de agarres cilíndricos. A su vez, este componente tiene 4 huecos rectangulares, para que el chasis quede al descubierto en dichas zonas. Esto es así, debido a que los asientos se adaptarán posteriormente al chasis en estos lugares, gracias a las barras laterales con las que cuenta este.

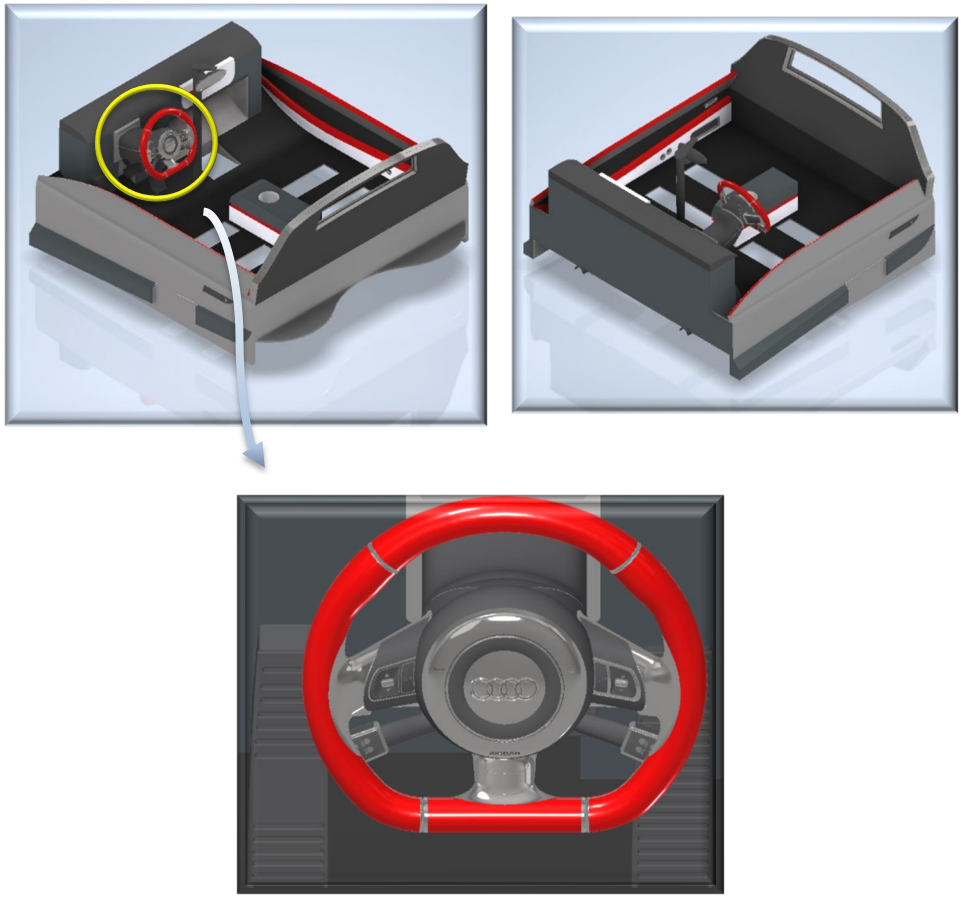


Figura 69. *Interior (Automóvil deportivo de juguete).*

Asientos (Conductor y Copiloto)

Los asientos se han diseñado con una estética deportiva y buscando emparejar la gama de colores del interior del coche (negro, blanco y rojo).

Dispone de 2 agarres cilíndricos para acoplarse en el chasis, una vez se ha montado el interior del coche.

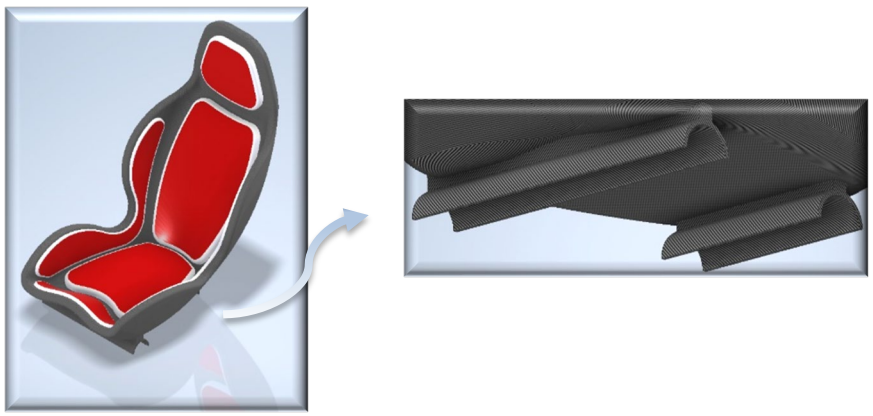


Figura 70. *Asientos (Automóvil deportivo de juguete).*

Carrocería

Este componente final, es uno de los más importantes en la estética del coche, ya que es de los únicos componentes que dan la forma para que el robot sea similar al modelo deportivo R8 real.

Para la pintura del automóvil se ha optado por una combinación de los colores rojo, negro y plateado.

Como ya se había comentado previamente, este componente se acopla a la parte trasera y la parte delantera, por medio de unas estructuras rectangulares que se resaltan en la siguiente imagen:

Figura 71. *Carrocería (Automóvil deportivo de juguete).*

Con esto quedaría finalizado el diseño del coche de juguete, que como se ha podido apreciar tiene un montaje muy sencillo, y rápido de efectuar.

La secuencia de montaje sería la siguiente:

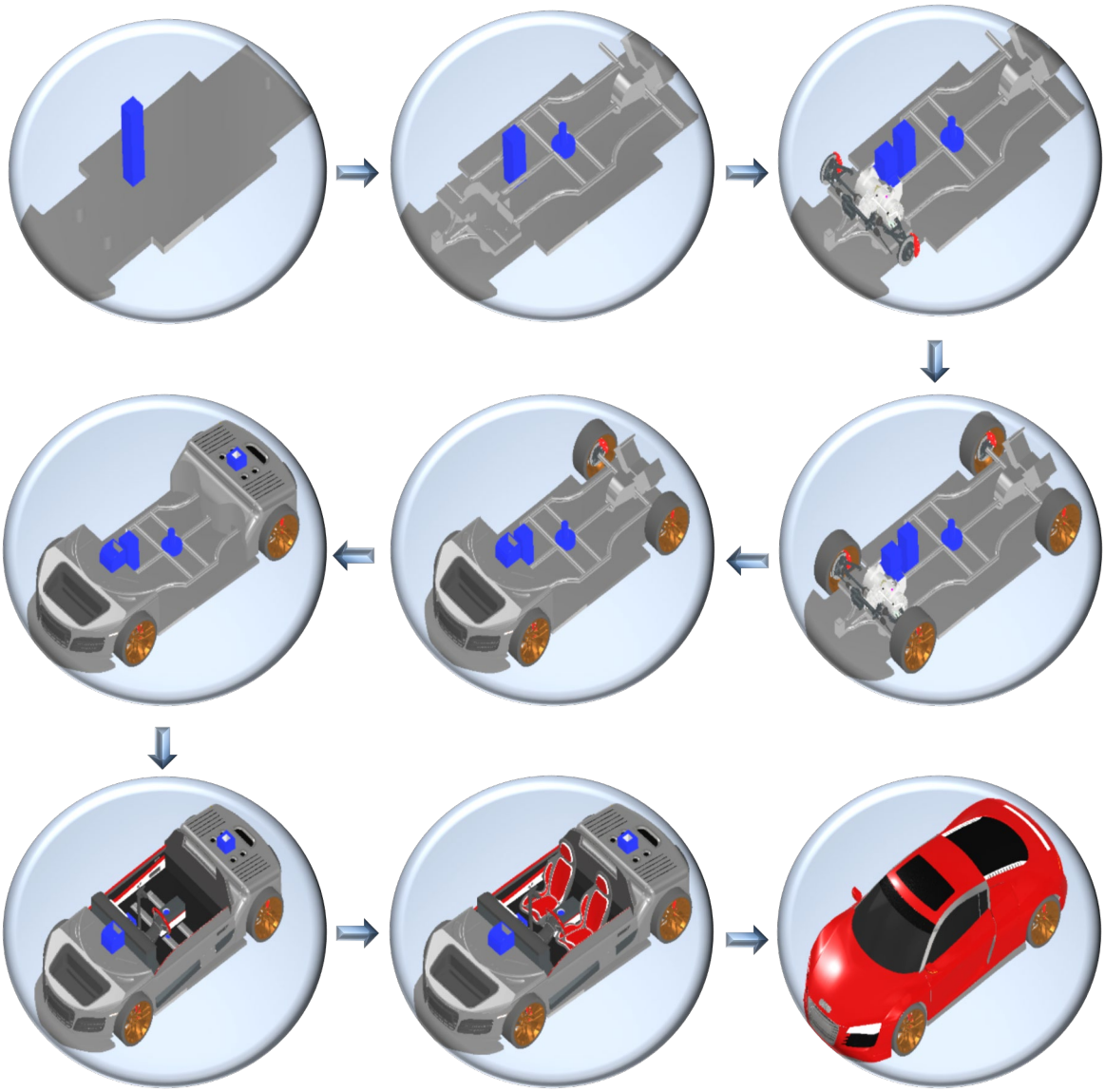


Figura 72. *Esquema fases del ensamblaje del automóvil de juguete.*

CAPÍTULO 4.3.2. CUBOS DETECTABLES POR VISIÓN ARTIFICIAL

Para la detección y procesamiento de piezas se ha incluido en el proyecto una cámara de visión artificial de ABB. Como de momento en este proyecto se ha supuesto que las piezas del coche son correctas y siempre llegan en la misma posición, no se ha incluido el análisis de este tipo de piezas (véase *capítulo IX. Trabajos Futuros*).

Sin embargo, para demostrar la potencia de la visión por computadora, en este proyecto se han añadido cubos de diferentes tamaños y colores. Los cuales tendrán que ser procesados en una imagen para detectar su color, forma, tamaño y posición.

Se han diseñado 3 tamaños diferentes de cubo como se puede apreciar en la siguiente imagen:

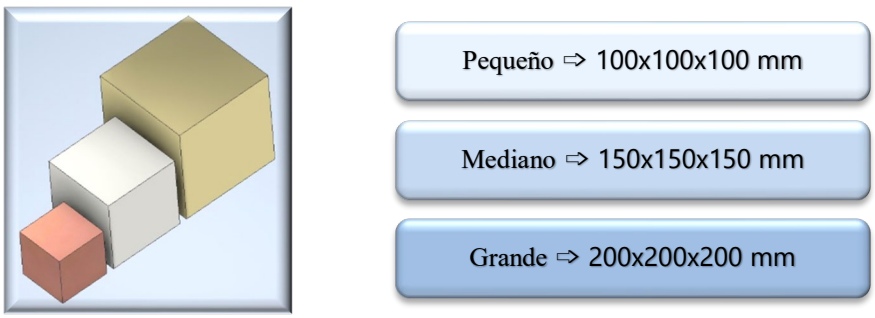


Figura 73. *Cubos y sus tamaños (Utilizados para reconocimiento por visión artificial).*

CAPÍTULO 4.3.3. HERRAMIENTAS DE LOS ROBOTS IRB 120 E IRB 14000

Finalmente, solo restarían por comentar las herramientas utilizadas por los robots. Estas no han necesitado ser diseñadas por el software Autodesk Inventor, ya que todo se ha desarrollado en RobotStudio.

- **Herramienta IRB 120:**

  Para la selección de las diferentes piezas provenientes del conveyor, hay que dotar a este robot de una herramienta.

  En las operaciones de traslado y colocación las mejores soluciones por las que se pueden optar serían unas pinzas o una ventosa (opción seleccionada).

  El uso de una ventosa permite al robot coger elementos de muy diversas formas y volúmenes (los cuales no podrían ser seleccionados mediante pinzas).

En relación con el diseño, esta primera herramienta ha sido creada por el autor Juan Carlos Martín Castillo, y se puede encontrar en los complementos que ofrece RobotStudio (*Add-ins/Gallery*), con el título '*Vacuum Suction Cup, to IRB120*'.

Esta ventosa es un componente inteligente (Smart Component), lo que supone que es un elemento que durante la simulación puede actuar (no tiene únicamente carácter decorativo).

La programación con bloques en RobotStudio de esta herramienta, se verá en los capítulos correspondientes (véase *capítulo 5.1.2. Componentes Inteligentes (Smart Components)*), por lo que ahora solo se muestra su aspecto visual en la siguiente imagen:

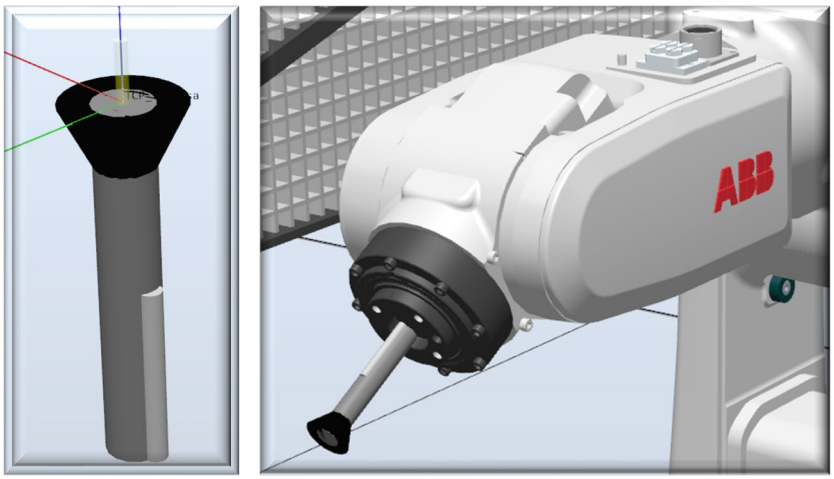


Figura 74. *Herramienta del robot IRB120 (Ventosa).*

- **Herramientas IRB 14000:**

Este robot colaborativo cuenta con dos brazos, lo que lo hace idóneo para tareas colaborativas de montaje, pero esta misma característica, obliga a hacer uso de dos herramientas (una por brazo).

A diferencia del anterior robot, en este caso se ha analizado como más conveniente por las tareas que iba a realizar (ensamblaje automóvil de juguete), el uso de pinzas como herramientas finales de las articulaciones.

La compañía de robots con la que se está realizando este proyecto (ABB Robotics) dispone de unas pinzas denominadas 'ABB Smart Gripper', en las que existen diferentes versiones/combinaciones, en función del uso o no de los siguientes elementos:

- Servomotor
- Extremos pinza
- Cámara Integrada
- Ventosas Laterales (Una o Dos)

En este proyecto las combinaciones utilizadas han sido:

- Brazo Izquierdo: Servo-Cámara-Extremos Pinza
- Brazo Derecho: Servo-Extremos Pinza

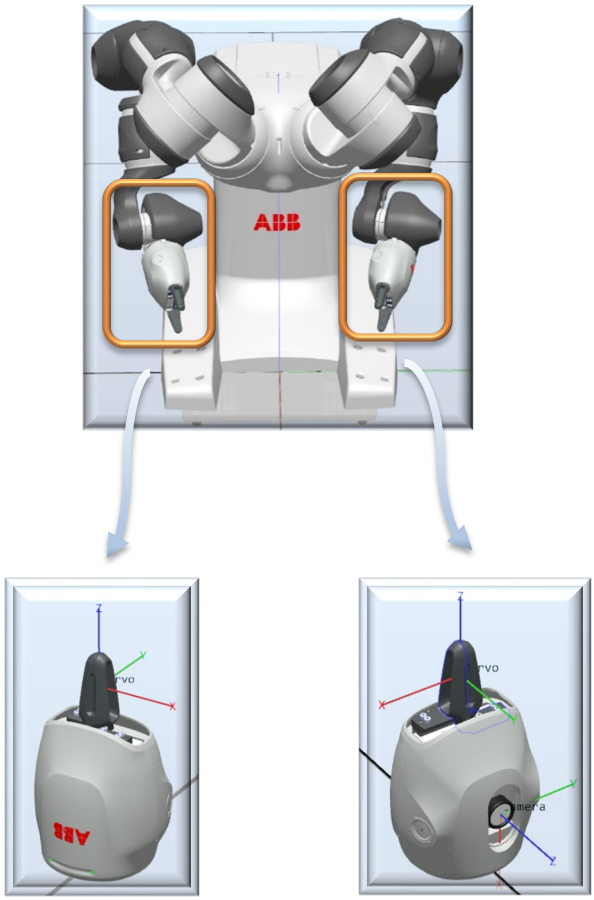


Figura 75. *Herramientas del robot IRB14000 YuMi (Garras/Pinzas).*

Con este último componente se cerraría el capítulo de diseño de elementos requeridos en este proyecto. Lo que supone el origen de un nuevo capítulo, en el que se comentará la programación realizada para desempeñar las tareas deseadas, en el software utilizado.

CAPÍTULO V. DESARROLLO Y PROGRAMACIÓN DEL ENTORNO SIMULADO

En la realización de las tareas definidas para este proyecto, se han utilizado como ya se comentó en capítulos anteriores, dos softwares de simulación muy potentes, en los cuales las posibilidades son infinitas, pero que requieren un alto nivel de conocimiento de programación para realizar los proyectos.

En los siguientes capítulos se desarrolla y explica la manera de programar todas las tareas realizadas, con los diferentes programas utilizados.

CAPÍTULO 5.1. DESARROLLO EN ROBOTSTUDIO

En la realización del presento proyecto en RobotStudio, se pueden distinguir dos partes fundamentales en relación con la programación de movimientos y la simulación.

La primera es el lenguaje de programación utilizado para ejecutar los movimientos de los robots. Ya que este software utiliza el lenguaje de programación textual de alto nivel desarrollado por la empresa ABB, RAPID.

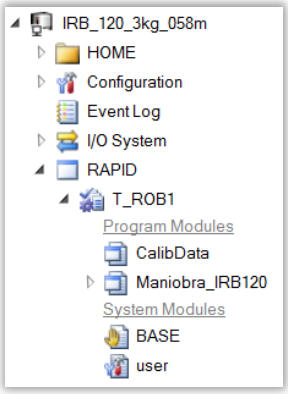
La segunda tiene relación con los denominados componentes inteligentes, que serán elementos encargados de realizar ciertas acciones durante la simulación, por ello hay que programarlos, pero en este caso es por lógica de bloques.

CAPÍTULO 5.1.1. RAPID

Al ser una célula de trabajo que consta de varios robots, van a ser necesarios varios códigos para controlar cada uno de estos. Además, uno de ellos es un robot colaborativo como ya se ha explicado, y cuenta con dos brazos robóticos totalmente independientes (que se pueden sincronizar), lo que supone la necesidad de programar individualmente cada brazo.

Recapitulando el anterior párrafo, en total se van a necesitar desarrollar 3 códigos diferentes, que podrán tener relación entre sí, para sincronizarse. A continuación, se explicarán las partes fundamentales de cada código. Si el lector quisiera visualizar la estructura de uno de los códigos completos puede visitar el *Anexo A.1. Ejemplo de código en Rapid.*

Maniobra_IRB120



La programación en Rapid, para controlar los robots, es una programación modular, en la que se van diseñando diferentes módulos, para realizar una serie de acciones. Todos estos módulos serán mencionados por el módulo principal que ejecuta la tarea.

Esta programación modular, permite el establecimiento de diferentes rutinas a un mismo robot, en función de las tareas que se desean realizar.

Figura 76. *Menú Opciones del robot IRB120 en RobotStudio.*

En este caso como solo se ha de realizar la tarea asignada para el presente proyecto, únicamente se ha hecho uso de un módulo 'Maniobra_IRB120' (véase *Figura 76*), al que le acompaña '*CalibData'*, donde se definen los datos de las herramientas utilizados en el programa.

Antes de proceder a la explicación de las partes fundamentales del código, lo primero es definir el diagrama de bloques realizado para la programación de este módulo.

En caso de que el lector del presente informe no sepa identificar los elementos que forman un diagrama de bloques se muestra en la siguiente ilustración el significado de cada bloque según los autores de 'JuegosRobotica.es'.

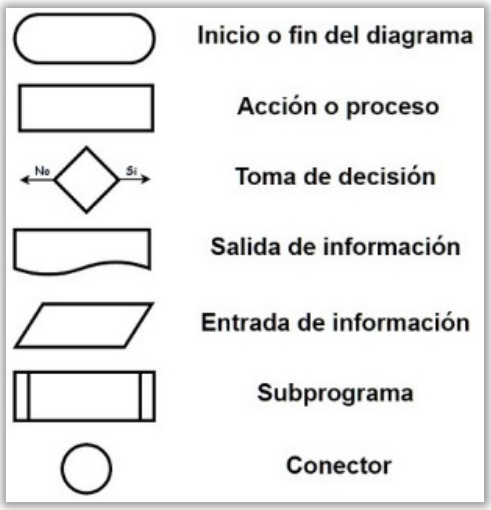


Figura 77. *Símbolos normalizados para los diagramas de bloques [Ilustración].* [37]

Diagrama de bloques – Código robot IRB120

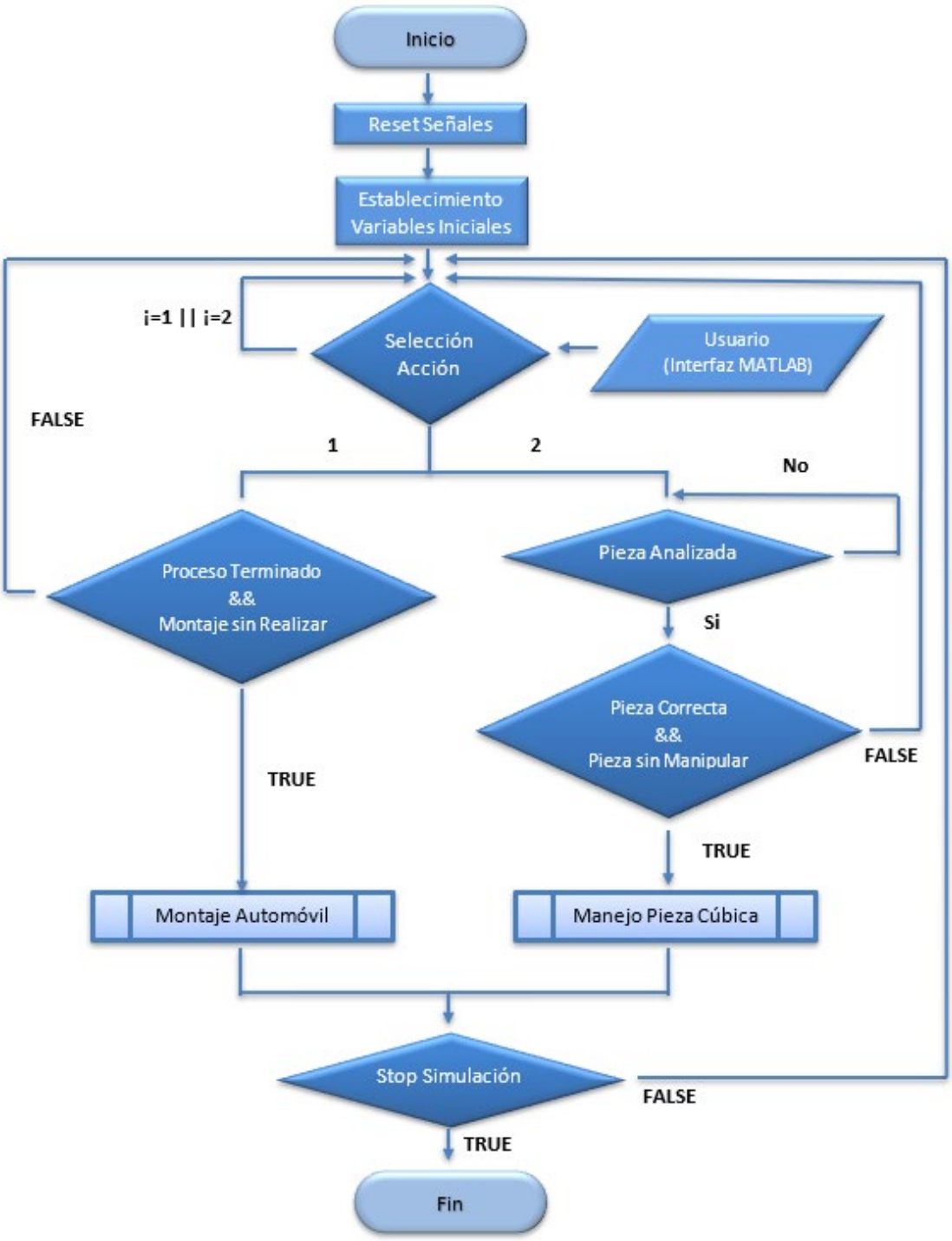


Figura 78. *Diagrama de bloques (código robot IRB120).*

Una vez se ha planteado el funcionamiento general del programa, es turno de realizar la programación con el lenguaje soportado por Robotstudio, RAPID.

Al inicio del programa, se deben declarar todas las variables que serán utilizadas a lo largo de este. Aunque antes conviene definir de que tipo pueden ser las variables utilizadas:

- *Const (Constantes)*
  No pueden ser modificadas. Pueden tener un ámbito local o global.

- *Var (Variables)*
  Pueden ser modificadas durante la simulación. Pueden tener un ámbito local o global.

- *Pers (Persistentes)*
  Deben ser inicializadas y si el valor se ve modificado durante la ejecución del programa, este último valor no se pierde, ya que se guarda en la variable como que se hubiese inicializado a este nuevo valor. Pueden tener un ámbito local o global.

Una vez determinadas las formas de declarar los datos, se van a mostrar y definir las variables utilizadas en este proyecto:

- *Robtarget* ⇨ *Definir la posición del robot y ejes adicionales.*
  En este proyecto se han utilizado para definir todas las posiciones que tiene que alcanzar el robot IRB120 para realizar la selección de las piezas y su posterior colocación en los conveyors correspondientes.

  En el caso del cubo procesado por visión artificial, también se necesita definir estas variables, aunque, como ya se mostrará más adelante, pueden ser modificadas, durante la ejecución del código, para adaptarse con total precisión a la posición de la pieza, en caso de que se modifique la trayectoria de esta.

  En la siguiente ilustración se muestran algunos ejemplos de posiciones definidas.

```
!VARIABLES ROBTARGET
CONST robtarget HOME:=[[416.3153533,0,564],[0.5,0,0.866025404,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget CogerUPIzda:=[[442,0,400],[0,1,0,0],[0,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget CogerUPDcha:=[[442,0,400],[0,1,0,0],[-1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Coger_ParteInferior:=[[487,15,210],[0,1,0,0],[0,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Coger_Chasis:=[[485,0,223.625],[0,1,0,0],[-1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

- ***Pos*** ⇨ *Describe las coordenadas de una posición (Ejes X,Y,Z).*
  Estas variables recogen los valores introducidos en la interfaz de MATLAB, para modificar las características del cubo creado (Posición y Color).
  Posteriormente, serán leídas por el componente inteligente, encargado de la creación de dicho cubo.

```
!VARIABLES POS
PERS pos PosInicial_Cubo:=[1590,1000,200];
PERS pos Pos_Cubo:=[1565,1000,200];
PERS pos Color_Cubo:=[0,255000,0];
```

- ***Num*** ⇨ *Definir variables numéricas.*
  Al igual que el caso anterior, son variables que serán modificadas, desde la interfaz de MATLAB.

```
!VARIABLES NUM
PERS num TamañoX:=0.1;
PERS num TamañoY:=0.1;
PERS num TamañoZ:=0.1;
PERS num SeleccionAccion:=1;
PERS num EjeX:=0;
PERS num Altura:=0;
```

- ***String*** ⇨ *Definir variables que contienen cadenas de caracteres.*
  Se ha utilizado para determinar de una forma más visual, de cara a la programación, en qué estado se encuentra la simulación, si ha terminado o no de realizar la acción.
  Se ha definido como variable persistente, para poder ser modificada durante la ejecución de la simulación

```
!VARIABLE STRING
PERS string EstadoProceso:="Empezando";
```

- ***Bool*** ⇨ *Definir valores lógicos (TRUE/FALSE).*
  Con esto se consigue determinar si se han realizado una serie de acciones o no, al cambiar el estado de las variables tras finalizar la acción.

```
!VARIABLES BOOL
PERS bool MontajeRealizado:=FALSE;
PERS bool SeleccionFiguraRealizada:=FALSE;
```

Una vez definidas las variables utilizadas, se comentarán los distintos procedimientos utilizados. Cabe destacar que únicamente se ejecuta el procedimiento principal (*main*), los demás serán llamados desde este, o desde procedimientos utilizados en el *main*.

El procedimiento principal se muestra en el fragmento de código de esta página, siguiendo el diagrama de bloques de la *Figura 78*.

Al inicio se invoca el procedimiento de reseteo de señales (en su interior se utiliza la instrucción '*Reset*' seguida del nombre de la señal) y se inicializan las variables. Con '*WaitUntil*' no avanza el programa hasta que desde la interfaz se haya seleccionado una opción y se comunique dicho dato por OPC hasta *Robotstudio*.

Una vez se ha seleccionado la opción, se entra en el bucle infinito y se selecciona la opción. Gracias a las sentencias condicionales '*IF*', solo se realiza una única vez hasta que el usuario no vuelva a elegir otra acción.

En el caso del análisis del cubo, no se realiza la acción del robot, hasta que la pieza ha alcanzado la posición y se ha procesado mediante visión artificial.

```
PROC main()
    ResetSeñales;
    SeleccionAccion:=0;
    EstadoProceso:="Terminado";
    WaitUntil SeleccionAccion<>0;
    MontajeRealizado:=FALSE;
    SeleccionFiguraRealizada:=FALSE;
    WHILE TRUE DO
        TEST SeleccionAccion
        CASE 1:
            IF EstadoProceso="Terminado" AND MontajeRealizado=FALSE THEN
                MontajeCoche;
            ENDIF
        CASE 2:
            WaitDO PiezaAnalizada,1;
            IF PiezaIncorrecta=0 AND SeleccionFiguraRealizada=FALSE THEN
                ManejoPieza;
            ENDIF
        ENDTEST
    ENDWHILE
ENDPROC
```

Para la ejecución de las acciones se necesitan los siguientes procedimientos secundarios:

- *MontajeCoche*
  Se inicializan las variables indicativas del comienzo del montaje y a partir de este paso se va a repetir la misma secuencia, pero con diferentes trayectorias y señales.

  La secuencia es posicionar el robot a la posición de HOME, activar la señal para la creación de la pieza correspondiente, esperar a que alcance la posición designada y realizar la trayectoria que convenga en función de la pieza manipulada.

  Tras manipular la penúltima pieza, se realiza la secuencia de sincronización con el robot IRB 14000 para el montaje final. Y, por último, se modifican las variables y resetean las señales oportunas.

  Parte del código se muestra en la siguiente ilustración:

```
PROC MontajeCoche()
    SetDO ComienzaMontaje,1;
    EstadoProceso:="Empezando";
    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza1,1;
    WaitDI SensorPieza,1;
    Trayectoria_ParteInferior;

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza2,1;
    WaitDI SensorPieza,1;
    Trayectoria_Chasis;

              ⋮

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza6,1;
    WaitDI SensorPieza,1;
    Trayectoria_FinalSincronizada;
    EstadoProceso:="Terminado";
    MontajeRealizado:=TRUE;
    Reset ComienzaMontaje;
ENDPROC
```

  Donde los procedimientos de las trayectorias tienen la siguiente forma:

```
!Trayectorias Implicadas en el Montaje del Coche
PROC Trayectoria_ParteInferior()
    MoveJ Coger_ParteInferior,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDO CogerPieza,1;
    MoveL CogerUPIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Dejar_ParteInferiorIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_ParteInferiorIzda,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
ENDPROC
```

  Se realizan los movimientos oportunos gracias a los *robtargets* definidos al principio y al uso de los comandos *MoveL* (movimiento recto) y *MoveJ* (movimiento cómodo).

Para la adhesión de las piezas se activa la señal 'CogerPieza', ya que se ha programado de esta manera en los componentes inteligentes y en la lógica de la estación (se verá en el próximo *capítulo 5.1.2. Componentes Inteligentes (Smart Components)*).

- *ManejoPieza*
  Al igual que en el caso anterior, comienza con la modificación de las variables y el posicionamiento del robot en la posición inicial.

  Como para ejecutar este procedimiento ya se ha tenido que procesar la imagen del cubo y se ha tenido que determinar la validez de este, se conocen parámetros como el tamaño o la posición del cubo. En función de estos parámetros el robot se desplazará a una determinada posición para alcanzar el cubo y transportarlo al conveyor del brazo izquierdo del robot IRB14000 en la posición adecuada.

```
PROC ManejoPieza()
    EstadoProceso:="Empezando";
    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    CogerFigura.trans.x:=EjeX+(Altura/2)-1150;
    CogerFigura.trans.y:=(Altura/2)-150;
    CogerFigura.trans.z:=Altura+200;
    MoveJ CogerFigura,v100,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO CogerPieza,1;
    WaitTime 5;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
    DejarFigura.trans.z:=Altura+75;
    MoveJ DejarFigura,v500,z100,TCP_Ventosa\WObj:=wobj0;
    WaitTime 1;
    Reset CogerPieza;
    WaitTime 1;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
    SeleccionFiguraRealizada:=TRUE;
    EstadoProceso:="Terminado";
ENDPROC
```

Con esto se da por finalizada la explicación del código necesario para ejecutar la maniobra_IRB120, se repite al lector que el código completo se encuentra en el *Anexo A.1. Ejemplo de código Rapid*.

Las tareas de ambos brazos del robot IRB 14000, son muy similares en cuanto al planteamiento, pero difieren en los movimientos realizados como es lógico.

El planteamiento del funcionamiento de los dos brazos es muy similar al módulo anteriormente explicado, por lo que se puede realizar un diagrama de bloques que represente a los dos módulos.

La diferencia vendrá en los procedimientos secundarios, que ya se verán en cada capítulo. El diagrama de bloques por tanto para estos dos brazos robóticos resulta:

Diagrama de bloques – Código brazos robóticos IRB14000

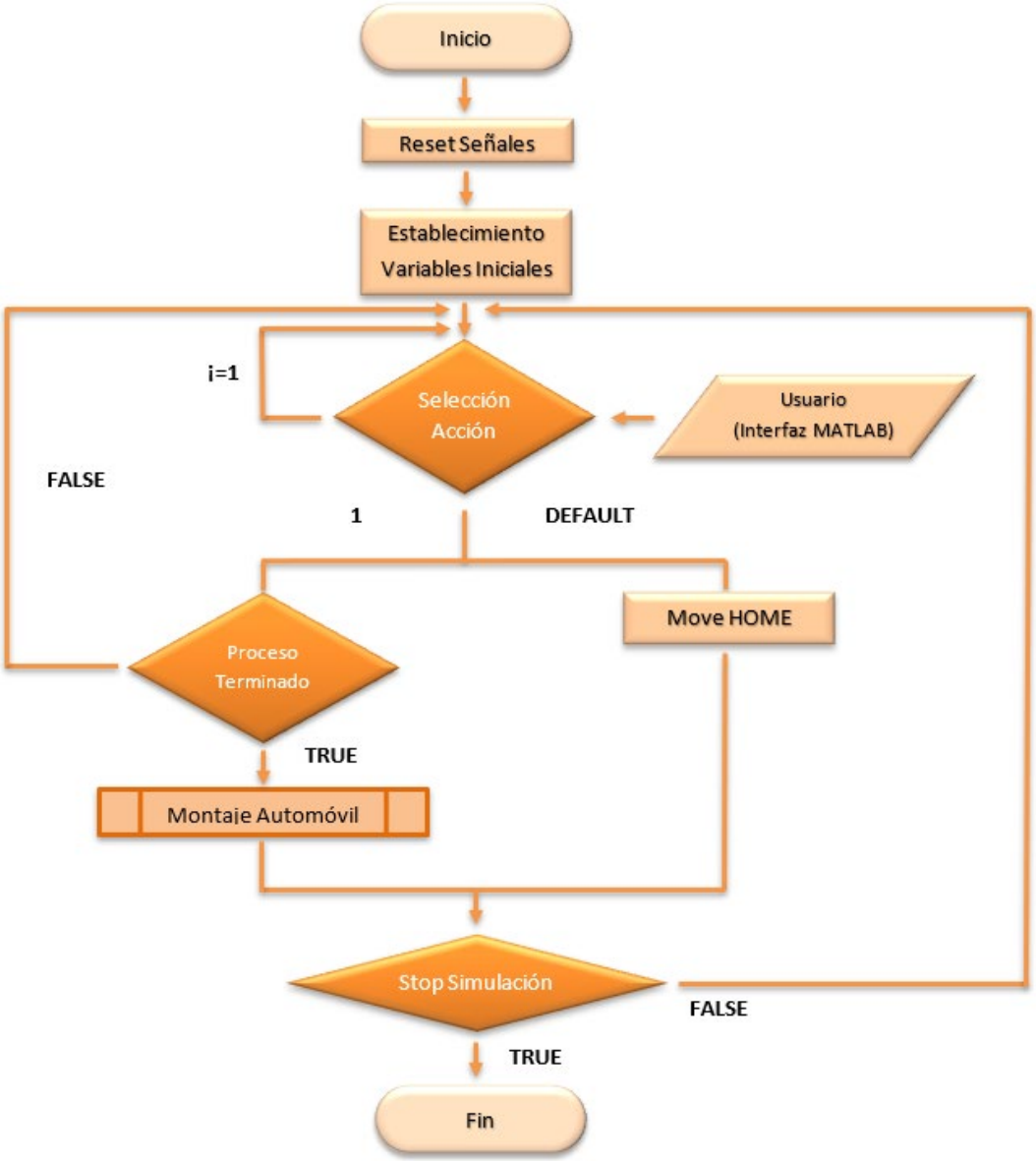


Figura 79. *Diagrama de bloques (código brazos robóticos IRB14000).*

Una vez definido el esquema general, se explicarán las diferencias entre los brazos robóticos, que sobre todo residen en el procedimiento 'MontajeCoche'

Maniobra_IRB14000_Brazolzdo

El diagrama de bloques para el montaje del automóvil en el brazo izquierdo es el siguiente:

Diagrama de bloques – Montaje automóvil brazo robótico izquierdo IRB14000

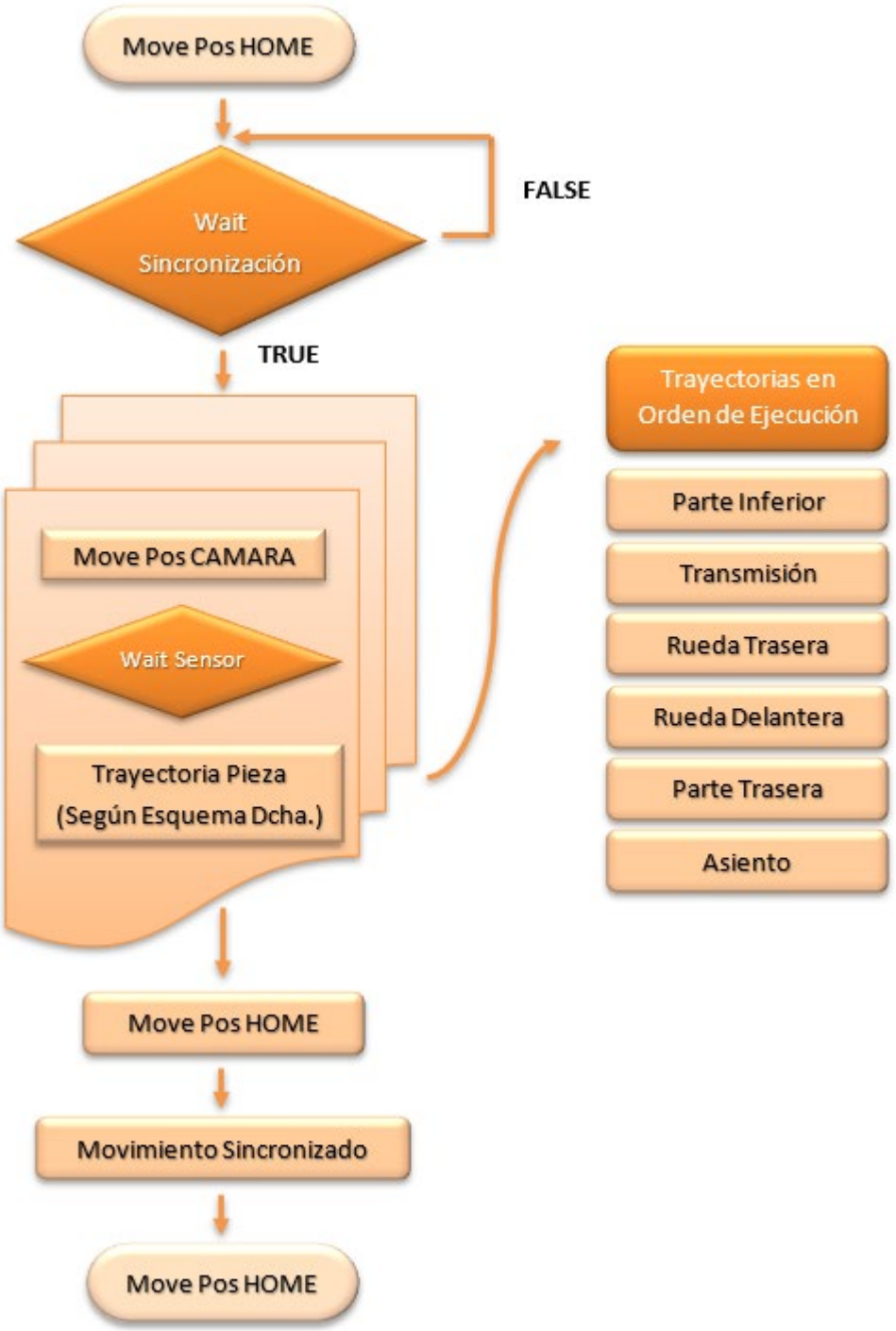


Figura 80. *Diagrama de bloques (montaje coche brazo derecho).*

Una vez definidos los diagramas necesarios para realizar la programación, se van a explicar las partes más importantes del código.

Al igual que en el anterior módulo, al inicio se declaran las variables utilizadas durante el desarrollo del código. En este caso se han utilizado variables que ya se han explicado como pueden ser '*Robtarget*', '*Num*' y '*String*', pero también aparecen dos nuevos tipos de variables que son:

- ***Tasks*** ⇨ *Definir varias tareas de RAPID.*
  Con esta variable, se definen las dos tareas que mueven los brazos robóticos, para poder estableces posteriormente el sincronismo entre ellos, gracias al uso de funciones como '*WaitSyncTask*' y '*SyncMoveOn*'.

- ***Syncident*** ⇨ *Especificar el nombre de un punto de sincronización.*
  Con el uso de estas variables se consiguen crear los puntos de sincronización en los que la tarea más rápida, tendrá que esperar hasta que la otra tarea alcance dicho punto.

```
!VARIABLES Tasks y syncident (Sincronización)
PERS tasks Tareas_Yumi{2}:=[["T_ROB_L"],["T_ROB_R"]];
VAR syncident SincronizacionTareas;
VAR syncident SincronizacionInicial;
VAR syncident SincronizacionFinal;
```

Tras la declaración de variables, se programa el procedimiento principal, el cual ya se había definido mediante un diagrama de bloques (véase *Figura 79*).

Es muy similar al módulo 'Maniobra_IRB120', pero con menor número de variables y con una diferencia en el 'TEST'. En este caso como los brazos solo realizan una acción en el montaje del automóvil, si se trata de otro caso, se coloca en la posición inicial y no realizará ningún movimiento hasta que no sea necesario para el montaje.

```
PROC main()
    ResetSeñales;
    SeleccionAccion:=0;
    EstadoProceso:="Terminado";
    WaitUntil SeleccionAccion<>0;
    WHILE TRUE DO
        TEST SeleccionAccion
        CASE 1:
            IF EstadoProceso="Terminado" THEN
                MontajeCoche;
            ENDIF
        DEFAULT:
            MoveJ HOME,v500,fine,Servo\WObj:=wobj0;
        ENDTEST
    ENDWHILE
ENDPROC
```

En el montaje del coche ya se ha definido como será la programación en el diagrama de bloques (véase *Figura 80*), pero cabe destacar el uso de la instrucción '*WaitSyncTask*', con la que se consigue detener la tarea, hasta que la otra tarea que se desea sincronizar, alcance este punto en el código (lógicamente debe aparecer esta instrucción en ambos códigos).

```
PROC MontajeCoche()
    MoveJ HOME,v500,fine,Servo\WObj:=wobj0;
    WaitTime 1;
    WaitSyncTask SincronizacionTareas,Tareas_Yumi;
    MoveJ Camara,v500,fine,Servo\WObj:=wobj0;
    WaitDI SensorPiezaIzda,1;
    TrayectoriaParteInferior_Coche;
```

Finalmente, la última novedad es el procedimiento '*syncmove*', en el cual se consigue realizar una serie de movimientos sincronizados de los dos brazos. Es decir, siempre van a empezar y terminar los movimientos a la vez, sin importar que sean con recorridos diferentes.

Esto se consigue mediante variaciones de la velocidad de actuación de los brazos robóticos.

Este tipo de movimientos que se desean sincronizados, tienen que contener un ID, para emparejarse con el movimiento del otro módulo.

Además, para la sincronización con el otro robot, como *RobotStudio*, no dispone de instrucciones entre diferentes robots, se han utilizado señales para simular este sincronismo. De manera que un robot se queda inactivo, hasta que el otro robot llega al mismo punto y activa la señal, para que el robot inactivo continúe la simulación. Esto se ha conseguido con las señales 'Sincronización' y 'SincronizacionIN'

```
PROC syncmove()
    SyncMoveOn SincronizacionInicial,Tareas_Yumi;
    MoveJ CogerCoche\ID:=10,v100,fine,Servo\WObj:=wobj0;
    SetDO CogerGrupoIzda,1;
    WaitTime 1;
    SetDO Sincronizacion,1;
    WaitDI SincronizacionIN,1;
    Reset Sincronizacion;
    MoveJ DejarCoche\ID:=20,v100,fine,Servo\WObj:=wobj0;
    Reset CogerGrupoIzda;
    WaitTime 5;
    MoveJ CogerCoche\ID:=30,v100,fine,Servo\WObj:=wobj0;
    SetDO SoltarBloqueTrabajo,1;
    Reset SoltarBloqueTrabajo;
    WaitTime 1;
    SyncMoveOff SincronizacionFinal;
UNDO
    SyncMoveUndo;
ENDPROC
```

Por último, se mostrará uno de los múltiples procedimientos que se han definido para realizar los movimientos en función de la pieza que se desee ensamblar.

Con la instrucción '*Offs*' se consigue añadir un offset a la posición definida en las variables iniciales (*robtarget*).

En función del procedimiento que se analice, se pueden visualizar diferentes señales, ya que la primera (en este caso 'CogerRuedaIzda'), se utiliza para la apertura de las pinzas, que cambiara en función de la pieza. La segunda señal (PinzaABB_Izda), será la que realmente adhiere la pieza a la herramienta, por lo que esta no cambiará en los diferentes procedimientos.

Estas señales se comprenderán mejor al leer el siguiente capítulo, en donde se explica la creación de los componentes inteligentes.

```
PROC TrayectoriaRuedaTra_Coche()
    MoveJ Offs(CogerRueda_Coche,0,0,100),v500,fine,Servo\WObj:=wobj0;
    MoveL CogerRueda_Coche,v500,fine,Servo\WObj:=wobj0;
    SetDO CogerRuedaIzda,1;
    SetDO PinzaABB_Izda,1;
    WaitTime 1;
    MoveJ Offs(CogerRueda_Coche,0,-200,150),v500,fine,Servo\WObj:=wobj0;
    MoveJ DejarRuedaTra_Coche2,v500,fine,Servo\WObj:=wobj0;
    MoveL DejarRuedaTra_Coche,v50,fine,Servo\WObj:=wobj0;
    Reset PinzaABB_Izda;
    Reset CogerRuedaIzda;
    WaitTime 1;
ENDPROC
```

Maniobra_IRB14000_BrazoDcho

Como se ha comentado anteriormente, el planteamiento de este segundo brazo es muy similar al brazo izquierdo, por lo que no merece la pena volver a repetir el código, siendo solo las posiciones y las señales lo que cambia.

Por tanto, se va a repetir el diagrama de bloques que se planteó para el brazo izquierdo en el montaje del coche (véase *Figura 80*), pero en este caso con las piezas que se manipulan, que son completamente distintas al otro brazo.

Diagrama de bloques – Montaje automóvil brazo robótico derecho IRB14000

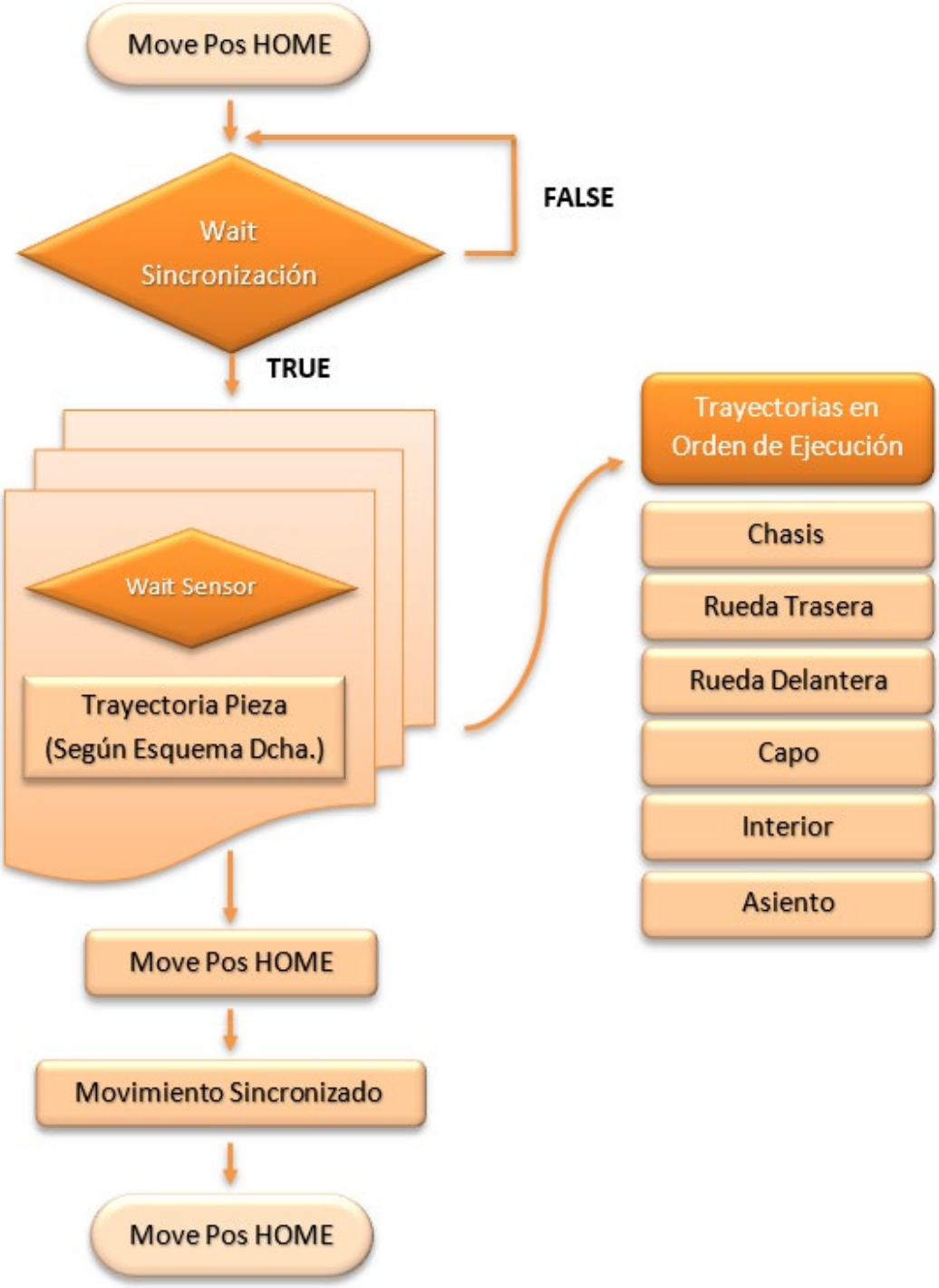


Figura 81. *Diagrama de bloques (montaje coche brazo derecho).*

CAPÍTULO 5.1.2. COMPONENTES INTELIGENTES (SMART COMPONENTS)

Una vez se han diseñado los componentes de la célula de trabajo, se necesitan definir los comportamientos de dichos mecanismos, con el fin de recrear su funcionamiento real durante la simulación. Para conseguir esto, es necesario añadirles una lógica de bloques en la que se programa los mecanismos, los sensores, el comportamiento ante ciertos estímulos, etc.

Para definir esta lógica en RobotStudio se hace uso de los componentes inteligentes los cuales permiten la entrada y salida tanto de señales como de propiedades (véase *Figura 82*)

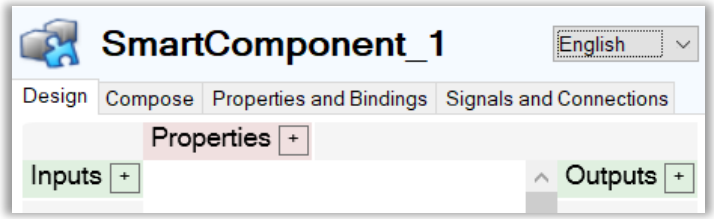


Figura 82. *Diagrama de bloques (montaje coche brazo derecho).*

Los bloques lógicos o componentes subordinados se añaden en la ventana 'Compose' y se distinguen hasta 9 categorías diferentes de bloques:

1. **Señales y propiedades:** Se encuentran puertas lógicas, contadores, temporizadores, expresiones matemáticas, convertidores y demás elementos para modificar una señal o una propiedad del sistema a programar.
2. **Primitivos paramétricos:** Los bloques de este capítulo son necesarios para crear de forma automática sólidos y líneas, así como para generar copias de componentes gráficos ya existentes.
3. **Sensores:** En esta categoría se hayan todos los sensores que se podrían necesitar en los procesos de producción automáticos básicos. Incluye sensores de colisión, de línea, de superficie, volumétricos, etc.
4. **Acciones:** Bloques muy útiles para la simulación de las herramientas debido a que permite la unión de dos objetos entre sí. Además, se encuentran opciones como crear/eliminar un objeto o simplemente hacerlo visible/invisible.
5. **Manipuladores:** Aquí se incluyen los bloques necesarios para la simulación de los conveyors, ya que se encuentran elementos para mover un objeto de forma lineal, hacerlo rotar un ángulo dado, moverse a lo largo de una curva o posicionarlo en un lugar determinado. También permite mover los ejes del mecanismo de un brazo robótico a una posición elegida.
6. **Controlador:** Contiene un único bloque que sirve para obtener o modificar los valores de una variable utilizada en RAPID.

7. **Física:** En el caso de que se quiera establecer cierta física a algún objeto o articulación, de manera que la simulación sea lo más realista posible.
8. **PLC:** Esta herramienta existe para la conexión con los PLCs de Siemens (SIMIT connection) vía memoria compartida.
9. **Otros:** En esta última categoría se encuentran una gran variedad de bloques lógicos con diversas funcionalidades como los siguientes: representación de una cola de objetos, generación de un número aleatorio, detención de la simulación, reproducción de un sonido, modificar algún color de un objeto, etc.

Para el desarrollo de los diferentes componentes móviles utilizados en este proyecto, van a ser necesarios una gran variedad de bloques lógicos que se mostrarán a lo largo del capítulo presente.

Definir las diferentes conexiones entre los bloques lógicos se puede realizar de dos maneras diferentes:

- **Visual:** En la ventana de diseño se pueden trazar segmentos para conectar los parámetros de los diferentes bloques.
- **No visual:** En la ventana de señales y conexiones se pueden ir definiendo los elementos que se quieren conectar y que propiedades o señales se desean unir

Tras esta breve introducción en la que se pretende introducir al lector en este campo de programación, se van a mostrar los distintos componentes inteligentes desarrollados.

Conveyors IRB 14000 (YuMi)

El primer componente inteligente que se ha desarrollado, es el conveyor encargado de transmitir las piezas depositadas por el robot IRB 120, hacia el destino de los brazos del robot YuMi.

Al activarse cuando se deposita una pieza, no necesita señales de entrada, pero sí que consta de una señal de salida que se activa a nivel alto (1) cuando la pieza depositada a recorrido la longitud del conveyor, hasta alcanzar la posición final (detectada por el sensor lineal final).

La velocidad de transporte se ha asignado a un valor de 0.5 $\frac{m}{s}$.

Figura 83. *Menú en RobotStudio con los componentes de la estación. (Conveyors YuMi)*

La lógica de funcionamiento sería la siguiente:

Se deposita una pieza en el *conveyor*, lo que origina que el sensor de nuevas piezas la detecte y se active a nivel alto (1), esta señal se enviará al *SR flip-flop* (circuito secuencial para guardar el valor de un bit, circuito biestable que cuenta con dos entradas), donde se almacenará ese valor alto (1), que producirá el movimiento de la pieza depositada. Desde que se detecta la pieza, hasta que se solicita el movimiento, trascurre 1 segundo, ya que se ha añadido un pequeño '*delay*' para aportar realismo al modelo.

Este movimiento se realizara hasta el momento que la pieza sea detectada por el sensor de pieza final, momento en el cual, se envía un '1' en el *reset* del *SR flip.flop*, para detener el movimiento (salida a 0).

Adicionalmente cabe destacar que se ha añadido una cola de elementos, por si se quisiera simular el movimiento de más de una pieza al mismo tiempo.

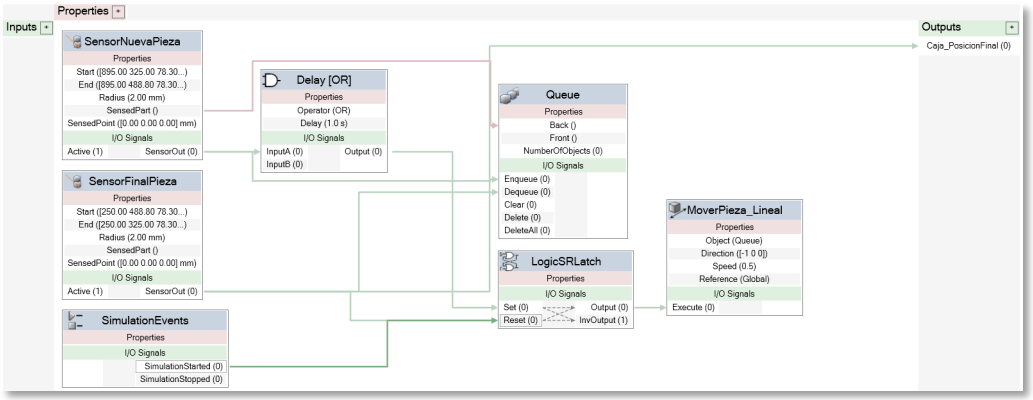
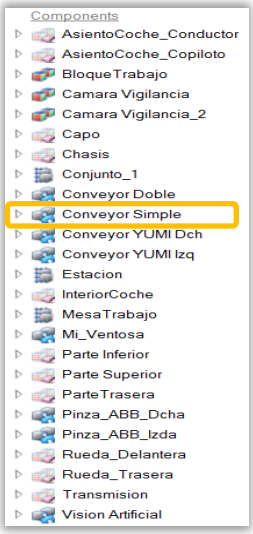


Figura 84. *Lógica de programación por bloques del conveyor para YuMi en RobotStudio.*

Esta misma lógica se implementa en el conveyor que dirige las piezas al brazo derecho, por lo que no se ha considerado necesario mostrar dicho componente inteligente.

Cinta transportadora Simple



El principio de funcionamiento de este componente inteligente es el mismo que en el anterior conveyor. Es decir, al detectar una pieza la desplaza por el recorrido de la cinta transportadora hasta detectar el sensor final, momento en el cual se detiene.

A pesar de que la idea es la misma, tiene diferencias conceptuales como que ahora ya no se permite el transporte de varias piezas a la vez, tiene que ser un transporte individual por turno y ahora la velocidad de transporte ha disminuido a 100 $\frac{mm}{s}$ (0.1 $\frac{m}{s}$).

Figura 85. *Menú en RobotStudio con los componentes de la estación. (Conveyor simple)*

Al ser un componente al que se le han añadido una serie de luces indicativas del funcionamiento, se han incluido '*Highlighters*' que cambian el color de las luces (verde-blanco, rojo-blanco), en función de si se encuentra transportando alguna pieza.

Tras finalizar el montaje del coche de juguete, y haberlo trasportado a la siguiente fase, en la simulación hay que eliminar todos los componentes que lo forman. Esto se realiza, ya que, si se volviera a ensamblar otro automóvil, no puede existir el anterior por que daría problemas en la simulación y se supone que ya se transportó a otra sección de la fábrica, por lo que no debería aparecer.

Para eliminar los componentes se realiza una pulsación cada 0.1 segundos desde el momento que el automóvil se encuentra en contacto con el sensor final y por cada pulsación, se va eliminando un componente. Como el automóvil seguiría avanzando, se van eliminando los componentes desde el frontal hasta el lado posterior.

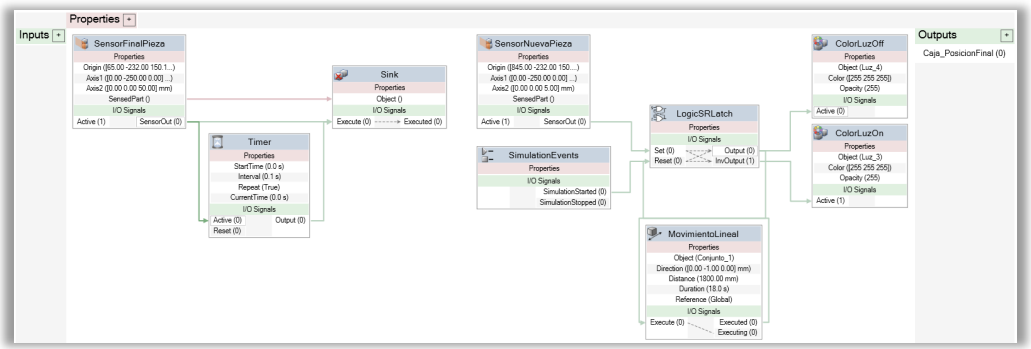


Figura 86. *Lógica de programación por bloques del conveyor simple en RobotStudio.*

Cinta transportadora Doble

El siguiente componente inteligente que ha sido diseñado para esta célula de trabajo ha sido la cinta transportadora en 'U', en la que se ha seguido la misma dinámica de los anteriores elementos. Con la única diferencia de que en este caso se ha añadido adicionalmente el control de la creación de las piezas necesarias para el montaje del automóvil del juguete.

Como en este primer proyecto se considera que las piezas son correctas, no es necesario programar el movimiento de pieza incorrecta, lo que supondría que esta realizase la semicircunferencia y regresase al edificio de origen.

Figura 87. *Menú en RobotStudio con los componentes de la estación. (Conveyor doble)*

Por tanto, como se mostrará a continuación la lógica de movimiento de piezas es muy similar, pero se difiere en la creación de estas.

Cada explicación se denota por un color, el cual, pertenece a los subconjuntos marcados de dicho color en la *Figura 88*. Con esto se ha pretendido que el lector del presente informe identifique mejor cada parte explicada.

Creación de copias de las piezas del automóvil

Al ser un proyecto que el usuario puede elegir realizar el montaje del automóvil, una y otra vez tras finalizar el anterior, no se puede disponer de un único set de piezas. Como solución a este problema, surgió la creación de copias del set de piezas originales, de forma que cada vez que el usuario seleccione este montaje, se vayan creando copias, las cuales serán manipuladas y eliminadas tras el montaje.

Cada copia creada se introduce en una cola de objetos, con el objetivo de desplazar linealmente a lo largo de la cinta estas copias, sin tener que añadir por cada una de ellas un bloque lógico de movimiento lineal. Ya que gracias a esto es la cola quien es desplazada, no los elementos individuales, de manera que se consigue el desplazamiento de los elementos que estén en ese momento en dicha cola.

Desplazamiento de las piezas

El desplazamiento viene regulado por un sensor de plano, dispuesto a la altura del robot, de manera que al entrar en contacto cualquier elemento con este, se activa a nivel alto (1) y detiene el funcionamiento de la cinta. Explicado en términos de bloques lógicos, ante la salida a nivel alto del sensor, se resetea el SR flip-flop que hacía funcionar el desplazamiento, ya que tenía en memoria un '1'.

Adicionalmente se incluye la opción de regular la posición de creación de piezas, con el bloque 'Posicionador (inglés: *Positioner*)'. Por si se quisiese en futuros proyectos eliminar la suposición de que las piezas del coche llegan de manera correcta y simular un posicionamiento incorrecto y posterior adaptación del robot para cogerlas

Creación de copias de las piezas del automóvil

En este último bloque se coordinan las luces indicadoras de movimiento del conveyor, de manera que solo se activa la luz verde si hay algún objeto moviéndose en este.

Para detectar si hay algún objeto en movimiento basta con obtener el número de objetos que hay en la cola, y si este número es distinto de cero y se está moviendo (sensor planar no detecta nada), activar el *highlighter* del color verde. Si alguna de las anteriores condiciones no se cumple, el color activado será el rojo.

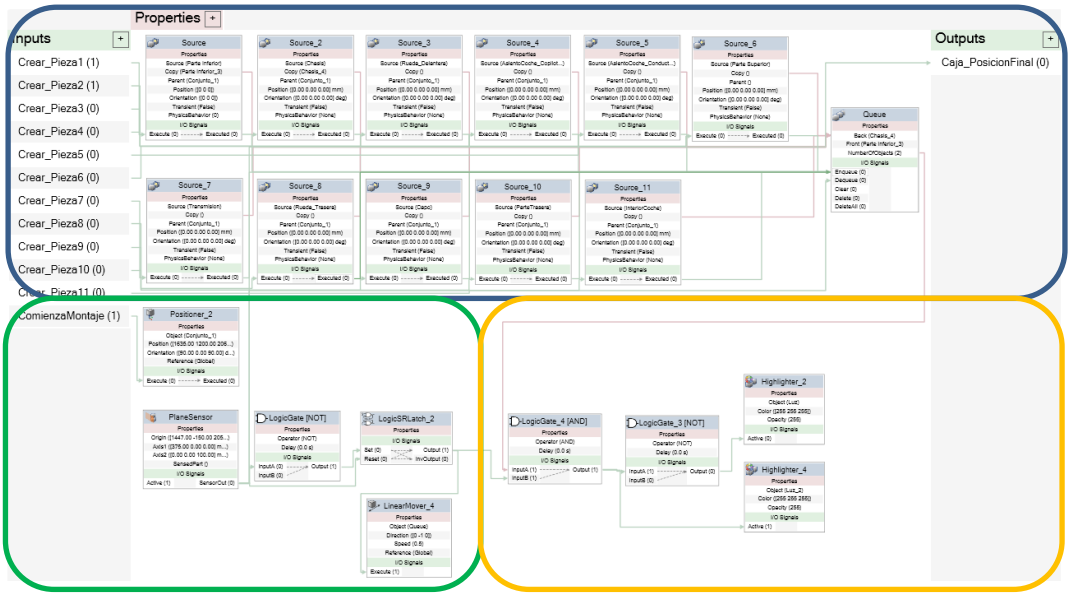


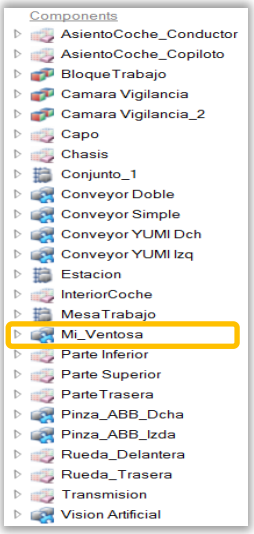
Figura 88. *Lógica de programación por bloques del conveyor doble en RobotStudio.*

El componente inteligente cuenta con 12 señales de entrada y una única señal de salida. Esto ya se verá más adelante en la lógica de la estación que se debe a que son los robots (mediante el lenguaje de programación de RAPID), los encargados de activar o desactivar las señales, para poder crear las piezas.

Con cada señal se controla la creación de un modelo de piezas. En cuanto a 'ComienzaMontaje' esta sirve para posicionar el origen de creación de piezas, de manera que se podría controlar en futuros proyectos este factor.

La salida indica que la pieza ha alcanzado destino de manera que tendrá que ser dirigida al robot, para que este se disponga a coger la pieza.

Herramienta: Ventosa



Este componente como se explicó en el '*Capítulo 4.3.3 Herramientas de los robots IRB 120 e IRB 14000*' es un elemento obtenido del autor Juan Carlos Martín Castillo, por lo que, a pesar de ser el único componente no diseñado por el autor del presente informe, a continuación, se resalta la lógica de bloques que presenta para su funcionamiento.

Cabe destacar que, si ha sido modificado con el objetivo de adecuarse a las tareas del proyecto, por lo que no es exactamente igual que el obtenido de este autor.

Figura 89. *Menú en RobotStudio con los componentes de la estación. (Herramienta IRB120 'Ventosa')*

La función de esta herramienta es poder adherirse a otras piezas, para conseguir que el robot las manipule.

Para ello al activar la señal de coger un objeto, se activa el sensor lineal de la herramienta y si detecta una pieza, se adhiere a ella. En el momento que se desactive esta señal, el objeto se soltará de la herramienta. Con esto se consigue simular el funcionamiento de la ventosa para selección de piezas.

Adicionalmente, para manipular el automóvil de este proyecto, se han añadido bloques extras encargados de lo siguiente:

- *Característica adicional 1*
  Cada vez que se seleccione con la ventosa una pieza, esta pasará a formar parte del grupo Conjunto 1 (conjunto con todas las piezas del automóvil, para conseguir moverlas a la vez durante la simulación). Esto se consigue con el bloque 'SetParent', pasando la propiedad de la pieza detectada por el sensor de la herramienta, y activando una señal si se desea añadir dicha pieza al conjunto o no.

- *Característica adicional 2*
  Funciones de adhesión y desunión del conjunto formado para el automóvil. Ya que antes solo era capaz de coger un objeto por vez. Y al final del montaje se requiere que el robot IRB120 desplace al automóvil montado hasta la cinta transportadora de salida.
  Esto se realiza con bloques de 'Attacher' y 'Detacher'.

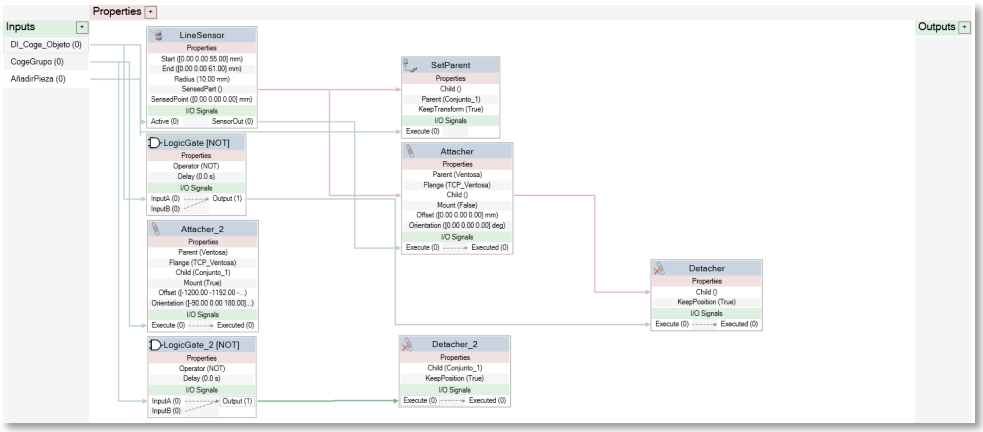


Figura 90. *Lógica de programación por bloques de la herramienta 'Ventosa' en RobotStudio.*

Herramienta: Pinza Brazo Derecho

Tanto el brazo derecho como el izquierdo del robot IRB 14000, necesitan de una herramienta con extremos motrices, para poder coger las piezas y realizar el ensamblaje de estas.

Por ello se utilizaron los 'smart grippers' que pone a su disposición la compañía ABB Robotics. Pero lógicamente solo se proporcionaba el modelo mecánico, es decir no disponen de programación propia, por lo que hay que diseñar por completo la lógica de bloques de estos componentes.

Figura 91. *Menú en RobotStudio con los componentes de la estación. (Herramienta IRB14000 'PinzaDcha')*

Para simular el movimiento mecánico de las pinzas se han utilizado los bloques lógicos 'PoseMover', en los que se asigna un mecanismo (en este caso las pinzas inteligentes) y un movimiento para ese mecanismo.

Como la pinza tiene que tener diferentes longitudes de apertura en función de la pieza que se vaya a seleccionar, el primer paso es definir estas longitudes y guardarlas como una acción de la pinza, que será realizado por el bloque 'PoseMover'.

Las diferentes posiciones de la pinza se modifican en la siguiente ruta '*Botón Derecho sobre el mecanismo/Modify Mechanism'*, de forma que aparece el menú mostrado en la siguiente imagen, en la parte derecha del programa.

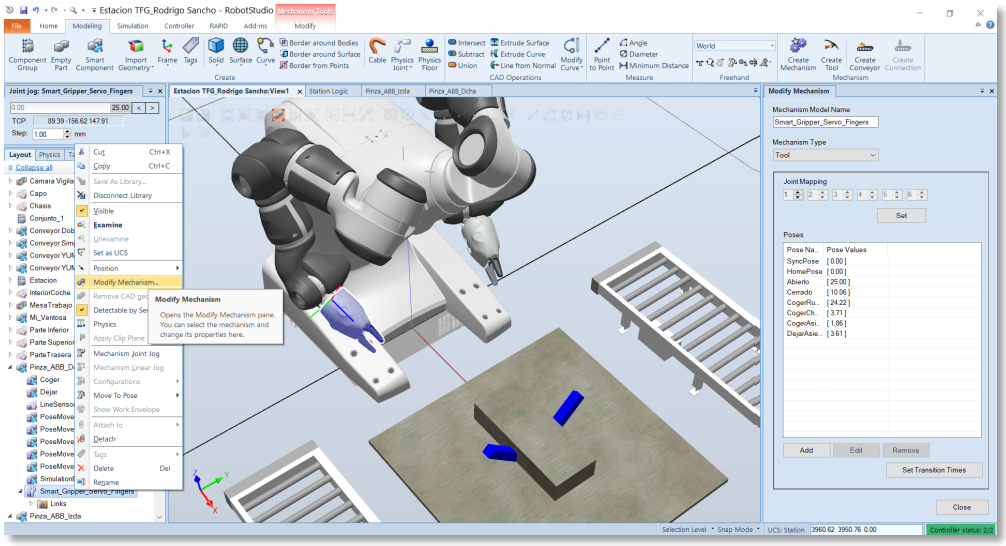


Figura 92. *Ventana educativa para mostrar cómo se programa la apertura de las pinzas.*

Como se puede apreciar aparecen definidas una serie de posiciones, en función de la pieza seleccionada. Si se desea añadir alguna más hay que pulsar con el cursor del ratón en el botón 'Add', momento en el cual se desplegara la siguiente ventana (véase *Figura 93*)

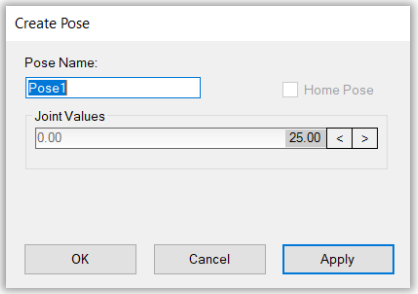


Figura 93. *Ventana para ajustar el valor de apertura de la pinza.*

Tras seleccionar la longitud adecuada para la pieza deseada, se le asigna un nombre y se aprieta '*Apply*', de esta manera se conseguiría guardar la posición.

Regresando a la lógica de bloques, ahora el lector ya se encuentra en posición de comprender mejor dicho diagrama y la razón de porque existen varios 'PoseMover' (uno por cada longitud de apertura de las pinzas). Tras realizar el movimiento deseado en función de la señal de entrada activada, se activa a nivel alto el *SR flip-flop*, para indicar en la salida que la pinza está en el estado 'Cerrada'.

Hasta ahora solo se ha programado el movimiento, pero como se ha enseñado en anteriores herramientas hay que programar la adhesión de las piezas, ya que el programa no interpreta las propiedades físicas. Por lo que se lanza la pregunta… *¿Cómo se programa la adhesión de los objetos a las pinzas cuando estas se encuentran cerradas?*

Si se ha leído y entendido la programación del anterior componente, esta pregunta no debería suponer ningún problema al lector para responderla. Al igual que antes se ha provisto a la pinza de un sensor lineal, encargado de detectar los objetos que la pinza se dispone a agarrar.

Si se detecta un objeto y se ha activado la señal solicitando el cierre de la pinza (en cualquiera de sus aperturas), se adhiere dicho objeto gracias al uso del bloque 'Attacher'. Si la señal solicitando el cierre se desactiva, las pinzas se abren y actúa el bloque 'Detacher', desuniendo la pieza de la herramienta.

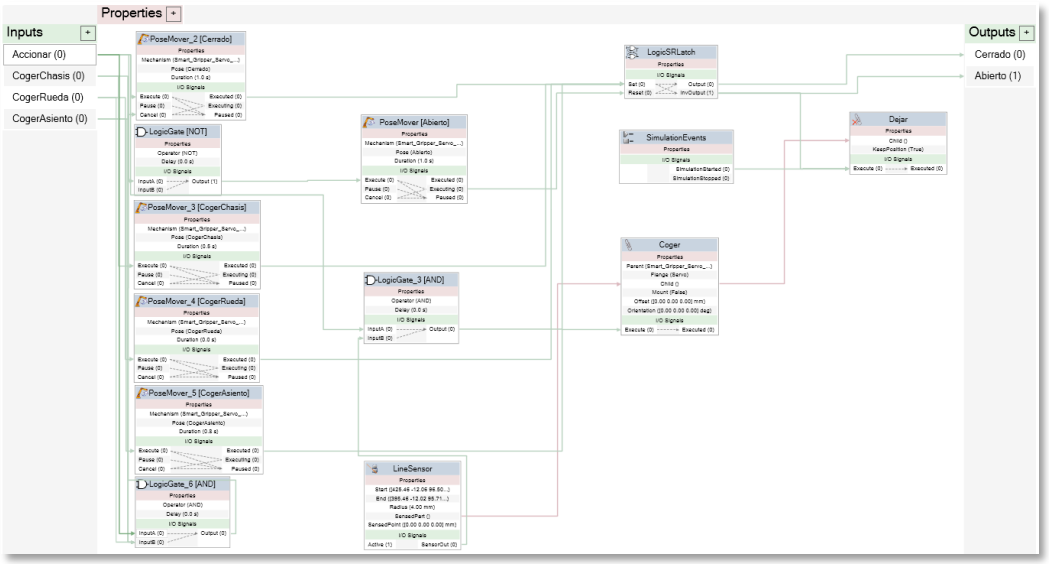


Figura 94. *Lógica de programación por bloques de la herramienta 'PinzaDcha' en RobotStudio.*

Herramienta: Pinza Brazo Izquierdo

Este componente tiene mayor complejidad que el equivalente del lado derecho, por ello se explica posteriormente a este. Una vez visto la base de la que parte, (ya que consta de los mismos elementos que la pinza derecha, pero con elementos adicionales), solo se van a comentar las novedades que esta segunda pinza implementa.

La primera novedad, se encuentra en las señales, ya que, además, de las encargadas de accionar la pinza y de coger las respectivas piezas, aparecen 2 nuevas señales para la manipulación del bloque de trabajo y del coche en conjunto.

Figura 95. *Menú en RobotStudio con los componentes de la estación. (Herramienta IRB14000 'PinzaIzda')*

Con estas señales y un conjunto de bloques '*attacher*' y '*detacher*', se consigue realizar la maniobra en la que los dos brazos del robot IRB14000 se sincronizan y alzan a la vez el bloque de trabajo que contiene al automóvil de juguete montado.

La segunda novedad, es que en los bloques 'PoseMover', a diferencia de la anterior herramienta, ahora realizan desplazamientos de las pinzas con diferentes recorridos, ya que en este caso se van a manipular piezas de diferentes tamaños a las anteriormente manipuladas.

El esquema de bloques se muestra en la siguiente ilustración

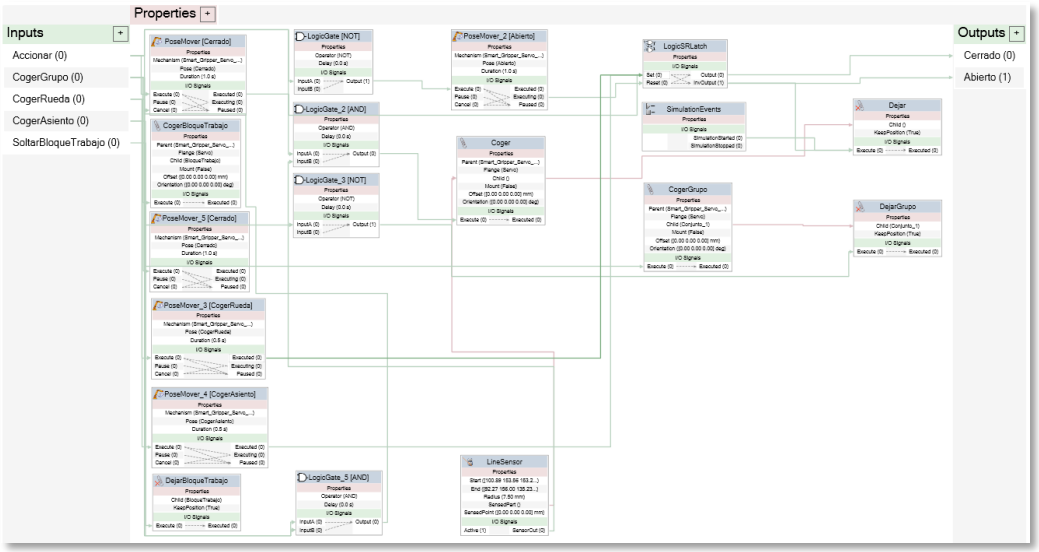


Figura 96. *Lógica de programación por bloques de la herramienta 'PinzaIzda' en RobotStudio.*

Visión Artificial

Finalmente, para la simulación de toda la parte de visión artificial, se ha creado este componente inteligente que a diferencia de los anteriores no está definido por un mecanismo como tal.

En su lugar, este componente será el encargado de crear el cubo, con las características (color, tamaño, posición) que el usuario haya determinado y de desplazar dicho cubo la trayectoria de regreso al edificio de origen, en caso de que la pieza fuese descartada (iluminando las luces indicativas de la cinta trasportadora).

Figura 97. *Menú en RobotStudio con los componentes de la estación. (Visión Artificial)*

También dispone de la posibilidad de conocer la posición actual del cubo, y actualizarla cada 0,1 segundos durante la duración de la simulación.

Adicionalmente, tiene la función de rotar las cámaras de vigilancia de las que dispone la célula durante toda la simulación.

Estas rotan a una velocidad de $1\frac{grado\ (°)}{segundo(s)}$ y se les permite un recorrido de 70 grados(°), para abarcar todo el campo de visión desde las esquinas de la célula.

Al igual que se propuso en la explicación de la 'Cinta transportadora doble', a continuación, se explicará la lógica de bloques desarrollada para cada función realizada, y se denotará con un color, el cual se identificará a su vez en la *Figura 98*, señalando la zona en la que se ha programado dicha función.

Creación Cubo personalizado por usuario desde interfaz

Este primer conjunto se caracteriza por la cantidad de bloques lógicos 'RapidVariable' que contiene. Esto se debe a la necesidad de adquirir de Rapid (que a su vez son adquiridos de MATLAB), los parámetros que el usuario ha decidido para crear dicho cubo.

Una vez adquiridos estos valores, se utilizan los bloques 'Parametric Box', 'Highlighter' y 'Positioner', para crear el cubo, cambiarle de color, y posicionarlo en una determinada posición, respectivamente.

Desplazamiento Cubo e Iluminación Luces

Al igual que en las cintas transportadoras, para realizar el desplazamiento del cubo, se necesitan 'LinearMover' y 'PlaneSensor', para poder determinar en qué posición se encuentra la pieza y realizar el movimiento adecuado en función de esta posición.
A su vez se utilizan SR flip-flops para almacenar los valores a nivel alto de los sensores de plano, cuando entran en contacto con la pieza.
Adicionalmente, aparece un nuevo bloque que no había sido utilizado hasta el momento (MoveAlongCurve), con el que se consigue la realización de una trayectoria curva con la pieza. Lo único que hay que introducirle es dicha trayectoria curva, que se ha trazado con la herramienta de RobotStudio (*Modeling/Curve/Arc*)

Localización Cubo durante simulación

Para la localización del cubo se utiliza un sensor de posición (PositionSensor) y con un bloque de 'RapidVariable' y un 'Timer', se actualiza el valor obtenido de la posición, a una variable en Rapid cada 0.1 segundos.

Rotación Cámaras durante simulación

Para la rotación de un objeto en RobotStudio se necesita, el bloque 'Rotator', en el cual se especifica, el objeto a rotar, la velocidad, dirección y el centro de rotación. Como las cámaras rotarían sobre si mismas, el centro de rotación se debe encontrar en el centro del objeto.
Al interesar solo una pequeña porción de área (70º de los 360º que puede abarcar), se han añadido unos temporizadores 'Timer' que cambian la señal cada 70 segundos (el desplazamiento ya se comentó que era a 1 $\frac{grado\,(°)}{segundo(s)}$), para cambiar el sentido de rotación.

En la siguiente ilustración se puede visualizar la explicación anterior:

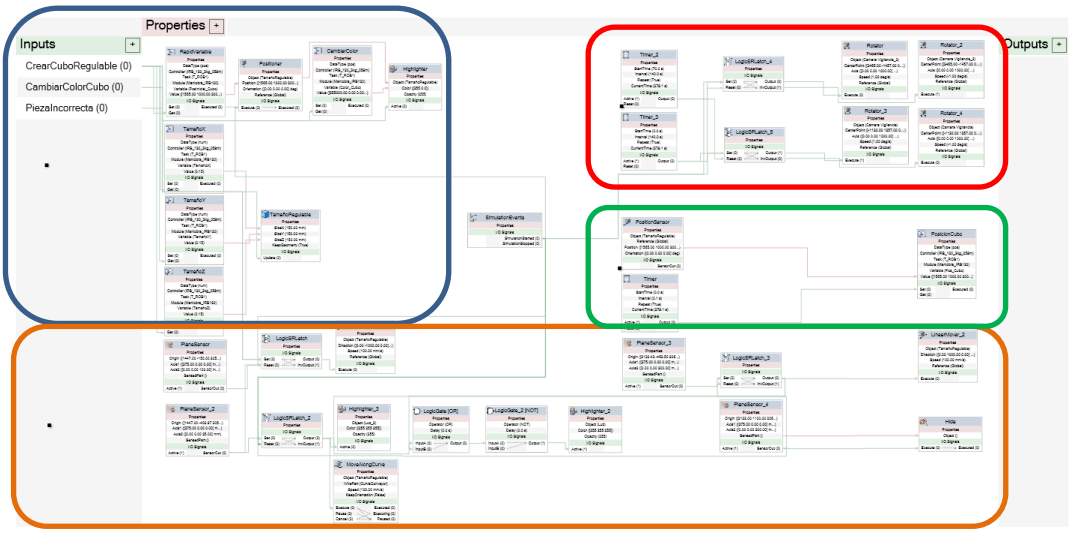


Figura 98. *Lógica de programación por bloques de la visión artificial en RobotStudio.*

Una vez programadas todos los componentes inteligentes utilizados en la realización del presente proyecto, hay que programar la lógica de la estación, en donde se definirá que robot será el encargado de manipular las señales de entrada, ya que estas señales se deben crear en el controlador utilizado por los robots.

También se pueden definir las posibles interacciones que pueden existir entre componentes inteligentes, aunque no es el caso de este proyecto.

Lógica de la Estación

La lógica utilizada para esta célula de trabajo entre robots y los componentes inteligentes, se muestra en la siguiente ilustración:

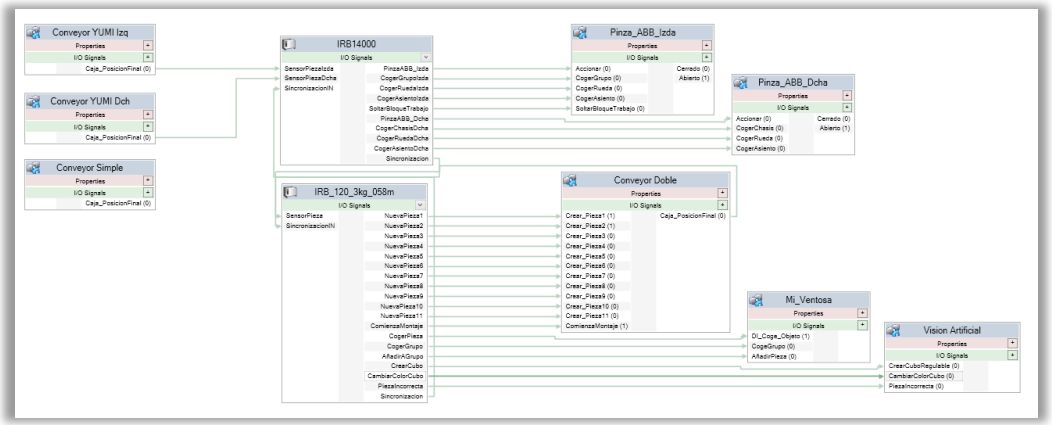


Figura 99. *Lógica de programación por bloques de la estación diseñada en RobotStudio.*

Visualizando la anterior imagen se pueden diferenciar dos zonas claramente divididas en la célula de trabajo:

- *Control de IRB14000*
  En este robot colaborativo el controlador es el encargado de manipular los conveyors YuMi y las dos pinzas que sirven de herramienta para dicho robot.

  De los conveyors, se obtiene la señal de que la pieza ha alcanzado la posición final, se manera que en ese momento el robot pueda disponerse a manipular esta pieza.

  Las señales de salida del controlador del robot, servirán para controlar las diferentes acciones que pueden realizar las pinzas, las cuales ya se vieron anteriormente.

  Existe una última señal de salida y otra de entrada que sirven para poder sincronizar los dos robots utilizados en la célula de trabajo.

- *Control de IRB120*
  Para este segundo robot manipulador, sus funciones son la manipulación de la cinta transportadora doble, su herramienta (ventosa) y el componente inteligente de visión artificial.

  En este caso en la cinta trasportadora, también se obtiene la señal para conocer cuando la pieza llega a la posición alcanzable por el robot. Del robot parten todas las señales necesarias para la creación de las copias de las diferentes piezas utilizadas en el automóvil, además de una señal para indicar el comienzo del montaje.

  Para la ventosa y la visión artificial también se disponen una serie de señales con origen en el robot, de manera que este pueda controlar los componentes.

  Al igual que antes, las señales de sincronización son para poder establecer este sincronismo con el otro robot utilizado.

Finalmente, la cinta trasportadora simple, al no necesitarse coger ningún elemento de ella por parte de los robots, no se precisa la conexión de ninguna señal con otro componente.

CAPÍTULO 5.2. DESARROLLO EN MATLAB

Como ya se había adelantado, la segunda parte del proyecto se ha realizado en MATLAB, debido a las mejores características que ofrece este software matemático, para el procesamiento de imágenes con visión artificial.

Como se necesitaba capturar la imagen de una forma automatizada, con ánimo de no interferir en el desarrollo de la simulación en RobotStudio, se ha decidido hacer uso de Simulink y herramientas como 'Mechanical Explorer', para simular parcialmente la célula de trabajo en este entorno y poder realizar la captura automática.

Esto conlleva que la simulación en MATLAB y en RobotStudio deben ser la misma, porque en caso contrario, surgirían problemas de que los parámetros detectados en un programa no se corresponden con los simulados en otro.

Toda esta problemática se irá desarrollando y resolviendo en los siguientes capítulos.

Finalmente cabe destacar que MATLAB también ofrecía un entorno gráfico para desarrollar una interfaz de usuario, de manera que la simulación de los dos programas se controle desde ahí y resulte más interactiva de cara al usuario.

CAPÍTULO 5.2.1. CÓDIGO MATLAB (ARCHIVOS '.M')

En el desarrollo del proyecto en MATLAB, ha ido surgiendo la necesidad de creación de diversos Scripts, para completar las siguientes acciones:

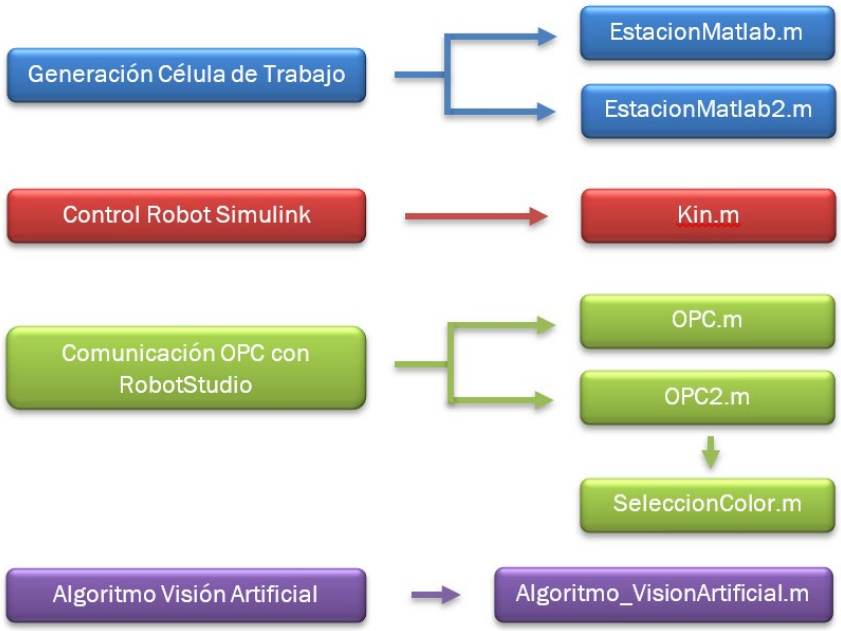


Figura 100. *Gráfico con los scripts desarrollados en MATLAB y su función.*

Durante el desarrollo del capítulo, se irán explicando las partes más importantes de cada código.

EstacionMatlab.m y EstacionMatlab2.m

En estos dos 'scripts' se encuentran todas las funciones, instrucciones y líneas de código necesarias para definir la célula de trabajo y los movimientos realizados por el robot y el cubo. Para ello necesita definirse esta célula en Simulink, proceso que ya se verá posteriormente en el *capítulo 5.2.2. Simulink.*

1.- EstacionMatlab.m

En primer lugar, cabe destacar, que este fichero describe una función, en la que los parámetros de entrada son el color, el tamaño y la posición que el usuario ha introducido, y los parámetros de salida son estos mismos, pero con los valores obtenidos por visión artificial.

Esto es debido a que realiza la simulación y dentro de la función se invoca el algoritmo de visión artificial de la imagen capturada de la estación.

```
function [Color,Area,Posicion]=EstacionMATLAB(Color,Size,PosicionX)
```

Para asignar un determinado movimiento al cubo, hay que crear unos vectores con las posiciones (Ejes X, Y, Z y Rotación), en las que se desea que se desplace dicho objeto. Como se muestra en la siguiente ilustración, esto se consigue con funciones como 'linspace' o 'zeros'.

```
% Variables que determinan el movimiento del Cubo (Traslacion y Rotacion)
% Linspace: Generar un vector espaciado linealmente
% Zeros: Crear un array de ceros
nPts= 1e3;
t= linspace(0,10,nPts)';
x= zeros(nPts,1);
y= linspace(0,1,nPts)';
z= zeros(nPts,1);
rot=zeros(nPts,1);
Ini=zeros(nPts,1);
```

Como MATLAB no permite simular el agarre de un objeto por los robots, hay que crear la percepción visual de que se está realizando dicha acción. Por ello se declara la siguiente instrucción, con la que se consiguen copiar un objeto en otro objeto destino, para ir creando las ilusiones de que el objeto ya no está adherido al robot o de que el robot está manipulando cierta pieza.

```
% Instrucción para modificar la herramienta o los objetos de trabajo.
copiar= @(orig, dest) eval(['! copy .\Visual\',orig,'.stl  ','.\Visual\',dest,'.stl']);
```

En el siguiente ejemplo se clarifica la explicación anterior, ya que se está definiendo la herramienta del robot IRB120 (recuerde que era una ventosa), copiando dicho objeto en 'tool1', que será la herramienta que moverá el robot.

En cuanto al objeto manipulado, como al inicio no tendría que agarrar ninguno, se le asigna un objeto 'null', que es tan pequeño que no se aprecia en la simulación.

```
% Posición de las herramienta 1 respecto de la herramienta 0
copiar('Ventosa', 'tool1');
tool1.tcp= [[0,0,60]*1e-3, [0,0,0]*deg];
% Posición de las herramienta 2 respecto de la herramienta 1
copiar('Null', 'load1');
```

Los valores definidos por el usuario en la interfaz se leen de la siguiente manera:

1. Se asignan las variables de entrada de la función a variables manipulables en el código.
2. Se ajustan a los valores de la simulación en MATLAB (ya que el valor no es el mismo que en RobotStudio)
3. Se realizan las acciones necesarias en función de estos parámetros. Por ejemplo, asignar un tipo de cubo en función del tamaño al objeto manipulado por el robot y la definición de los puntos que alcanza el robot en función de este tamaño.

```
% Asignacion del Color y la Posición del Cubo elegidos por el Usuario en
% la interfaz,a las variables utilizadas en la simulación
Pos=[0,1,0];
ColorCubo=Color;
PosicionFigura=PosicionX-1760; % Se resta una constante para ajustar a los valores de la simulación
Pieza1.base= [[PosicionFigura,1000,220]*1e-3, [0,0,0]*deg];
```

```
% Asignacion del tamaño elegido por el Usuario en la interfaz, a la
% simulacion del cubo. En función del tamaño de este, las posiciones
% tomadas por el robot para la recogida y entrega del cubo en los conveyors
% son diferentes (Siempre busca el centro, pero su altura es diferente)
if Size==[0.2,0.2,0.2]
    copiar('CuboGrande','Figura');
    P_Recogida= [[PosicionFigura+650,0,420]*1e-3,[0,0,180]*deg];
    P_Entrega= [[-350,435,280]*1e-3,[0,0,180]*deg];
elseif Size==[0.15,0.15,0.15]
    copiar('CuboMediano','Figura');
    P_Recogida= [[PosicionFigura+650,0,370]*1e-3,[0,0,180]*deg];
    P_Entrega= [[-350,435,230]*1e-3,[0,0,180]*deg];
elseif Size==[0.1,0.1,0.1]
    copiar('CuboPequeno','Figura');
    P_Recogida= [[PosicionFigura+650,0,320]*1e-3,[0,0,180]*deg];
    P_Entrega= [[-350,435,180]*1e-3,[0,0,180]*deg];
end
```

Todos los objetos de la estación de trabajo se asignan con la instrucción 'Copiar', anteriormente comentada. A continuación, se muestra un ejemplo en la siguiente ilustración.

```
% Definición de los objetos de trabajo introducidos en la Estación
copiar('Conveyor_4', 'wobj1');
wobj1.base= [[1350,-400,-470]*1e-3, [90,0,0]*deg];
copiar('Cristal1_2', 'wobj2');
wobj2.base= [[2229,1612.5,804.60]*1e-3, [90,90,0]*deg];
copiar('Luz', 'wobj3');
```

Cabe resaltar que estos objetos se encuentran en una carpeta en el interior del proyecto, con extensión 'stl' ya que dicho formato es una tecnología de creación rápida de prototipos utilizada para producir piezas tridimensionales, lo que resulta idóneo para la representación de objetos en 3D.

Figura 101. *Componentes 3D usados en el modelo de simulink.*

La manera de establecer una especie de 'memoria compartida' entre los diferentes programas y aplicaciones de MATLAB, es el uso del 'Espacio de trabajo'(Workspace), por ello para 'escribir en memoria' se utiliza la instrucción '*assingnin*', con la que se asigna un valor a la variable en el espacio de trabajo (tanto si esta creada, como si se crea en la asignación).

Por ello todas las variables que son trasladadas a Simulink se declaran de esta manera, como se puede visualizar en la siguiente ilustración.

```
% Asignación de todas las variables utilizadas durante la simulación
% en Workspace, para poder ser asignadas en el modelo de Simulink
assignin('base','ColorCubo',ColorCubo);
assignin('base','Pos',Pos);
assignin('base','q0',q0);
assignin('base','pose',pose);
assignin('base','tool1',tool1.tcp);
```

Esta técnica también es válida para asignar el movimiento a realizar el cubo, con las variables comentadas al inicio del capítulo. El único aspecto a destacar es que para determinar un movimiento y poder asignarle una velocidad, se tiene que declarar la posición y el tiempo en cada posición, de ahí, que sean vectores de dos dimensiones [tiempo, posición].

```
CoordenadaX=[t,x];CoordenadaY=[t,y];CoordenadaZ=[t,z];RotacionZ=[t,rot];Inicial=[t,Ini];
assignin('base','CoordenadaX',CoordenadaX);assignin('base','CoordenadaY',CoordenadaY);
assignin('base','CoordenadaZ',CoordenadaZ);assignin('base','RotacionZ',RotacionZ);
assignin('base','Inicial',Inicial);
```

Finalmente, tras la asignación de todos los parámetros, objetos y movimientos que se desean realizar, se debe simular el modelo en simulink, y para ello se utiliza la instrucción 'sim(NombreModeloSimulink)'.

```
% Ejecución Simulación del modelo de la estacion creado en Simulink
sim('Estacion_TFG_RodrigoSancho');
```

Tras la simulación, el cubo se ha desplazado hasta la posición del robot, momento en el cual se debe tomar la captura para posteriormente analizarla y proceder a realizar la acción oportuna en función de los resultados.

En primer lugar, se toma un video de la simulación realizada del segundo cuadro con la vista superior [1]. Tras ejecutar el video, se captura el último fotograma (cubo ha alcanzado la posición deseada) con la instrucción 'read(video, frames deseados)'

```
% Crea un video de la simulación
% NOTA IMPORTANTE %
% En Mechanical Explorer se debe tener la distribución de la vista en 4
% ventanas, para ello establezca: View > Layout > Four Standard Views
% El video se graba de la segunda vista, que es la vista aérea
smwritevideo('Estacion_TFG_RodrigoSancho','video','tile',2);
pause(15); % Pausa en lo que ejecuta el video grabado
v= VideoReader('video.avi'); % Lectura del video grabado
p= read(v, Inf); % Captura de imagen del último fotograma del video grabado
```

Tras capturar la imagen se analiza invocando el algoritmo creado de visión artificial, que se comentará posteriormente.

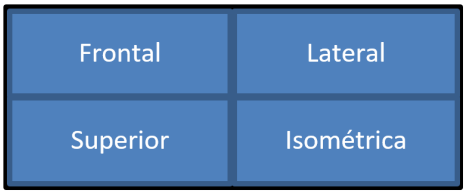
```
% Tratamiento de la Imagen capturada con visión artificial.
% Se obtiene el Color, Tamaño y Posición del Cubo, en el que caso de que
% exista y se detecte tal figura.
[Color,Area,Posicion] = Algoritmo_VisionArtificial(p);
```

Con esto finalizaría la primera etapa de la simulación y la información recolectada se enviará a RobotStudio.

---

[1] *Aclaración sobre la ventana de simulación en Matlab*

La simulación se muestra en 4 ventanas distintas, con las diferentes vistas (Isométrica, Frontal, Lateral, Superior). Ya se mostrará en los resultados esto con más detalle, pero para captar la idea se ha dibujado el siguiente esquema.



2.- EstacionMatlab2.m

En esta segunda etapa se ha procesado la imagen capturada y ya se tienen todos los parámetros del cubo, por lo que se va a buscar una determinada acción en función de la validez de estos parámetros.

Al igual que antes se trata de una función, pero en este caso no tiene parámetros de salida y su único parámetro de entrada es una variable indicadora de la validez del cubo detectado.

```
function EstacionMATLAB2(DescartarPieza)
```

En el comienzo se declaran unas variables que carecen de interés, pero el primer punto a resaltar en el código surge en el siguiente fragmento de código, ya que se utilizan 2 nuevas funciones.

- Importrobot: Importar el modelo de árbol de cuerpo rígido desde el modelo SimscapeMultibody
- Evalin: Extraer el valor de la variable en el espacio de trabajo (workspace), o con la analogía que se hacía anteriormente, esto sería como 'leer de memoria' una variable.

```
% Importa el modelo de árbol de cuerpo rígido desde el Modelo SimscapeMultibody
robot= importrobot(Estacion);
% Evalua del Workspace una serie de parámetros
P_Recogida= evalin('base','P_Recogida');
P_Entrega= evalin('base','P_Entrega');
PosicionFigura= evalin('base','PosicionFigura');
```

Una vez, se ha definido el robot (IRB120) y la estación de trabajo, se puede crear el objeto de la clase 'Kin' con el que se podrá controlar el robot, como si se tratara un programa de robótica (introducir movimientos absolutos, MoveL, MoveJ…). Esta clase se tratará en el siguiente punto.

```
% Se crea el objeto rob, con el que se controla el movimiento del Robot
rob= Kin(robot, Estacion, load1.name);
```

El siguiente fragmento de código con especial relevancia es el análisis y la actuación y en función de si la pieza era correcta o en caso contrario, era necesario descartarla.

Si es correcta, se actúa sobre el movimiento del robot, para que coja la pieza, se mueva a su posición de 'HOME' y finalmente deposite el cubo en el conveyor izquierdo que direcciona las piezas al robot YuMi.

MoveJ, Tool y MoveAbsJ son funciones declaradas en la clase 'Kin'

```
% Sentencia condicional para decidir si realiza una acción u otra, en
% función de la validez de la pieza
if DescartarPieza==0
    % Pieza Válida -> Robot coge la Pieza
    y= zeros(nPts,1);
    CoordenadaX=[t,x];CoordenadaY=[t,y];CoordenadaZ=[t,z];
    assignin('base','CoordenadaX',CoordenadaX);assignin('base','CoordenadaY',CoordenadaY);
    assignin('base','CoordenadaZ',CoordenadaZ);
    rob.MoveJ(P_Recogida);
    rob.Tool(tool1.name);
    tool1.tcp= [[0,0,60]*1e-3, [0,0,0]*deg];
    assignin('base','tool1',tool1.tcp);
    copiar('Figura','load1');
    copiar('null','Figura');
    rob.MoveAbsJ(HOME);
    Izda= [90,0,0,0,90,0,0,0,0,0]*deg;
    rob.MoveAbsJ(Izda);
    rob.MoveJ(P_Entrega);
    copiar('load1','wobj7');
    copiar('null','load1');
    rob.MoveAbsJ(HOME);
```

Si, por el contrario, la pieza es incorrecta, hay que ir asignando los puntos del espacio por los que se desea desplazar a la pieza y posteriormente simular la estación, para visualizar ese movimiento. De esta manera si se analiza el siguiente código, se puede ver como se simula tres veces la estación, ya que en cada vez se ha realizado un movimiento distinto de desplazamiento.

1. Movimiento Lineal Sentido Eje Y positivo
2. Movimiento Curvo alrededor del eje Z
3. Movimiento Lineal Sentido Eje Y negativo

```
else
    % Pieza No Válida -> Pieza Se desplaza por el Conveyor hasta regresar
    % al lugar de origen
    % Secuencia de Avance
    x= zeros(nPts,1);y=linspace(0,0.06,nPts)';z= zeros(nPts,1);rot=zeros(nPts,1);Ini=ones(nPts,1);
    CoordenadaX=[t,x];CoordenadaY=[t,y];CoordenadaZ=[t,z];RotacionZ=[t,rot];Inicial=[t,Ini];
    assignin('base','CoordenadaX',CoordenadaX);assignin('base','CoordenadaY',CoordenadaY);
    assignin('base','CoordenadaZ',CoordenadaZ);assignin('base','RotacionZ',RotacionZ);
    assignin('base','Inicial',Inicial);
    sim('Estacion_TFG_RodrigoSancho');

    % Secuencia de Rotacion sobre un punto exterior a la Pieza
    Pieza1.base= [[PosicionFigura,-400,215]*1e-3, [0,0,0]*deg];
    x= zeros(nPts,1);y=zeros(nPts,1);z= zeros(nPts,1);rot=linspace(0,0.45,nPts)';
    CoordenadaX=[t,x];CoordenadaY=[t,y];CoordenadaZ=[t,z];RotacionZ=[t,rot];
    assignin('base','CoordenadaX',CoordenadaX);assignin('base','CoordenadaY',CoordenadaY);
    assignin('base','CoordenadaZ',CoordenadaZ);assignin('base','RotacionZ',RotacionZ);
    assignin('base','Pieza1',Pieza1.base);
    sim('Estacion_TFG_RodrigoSancho');

    % Secuencia de Retroceso hasta el lugar de origen
    Pieza1.base= [[PosicionFigura+690,-400,215]*1e-3, [0,0,0]*deg];
    x= zeros(nPts,1);y=linspace(0,-0.25,nPts)';z= zeros(nPts,1);rot=zeros(nPts,1);
    CoordenadaX=[t,x];CoordenadaY=[t,y];CoordenadaZ=[t,z];RotacionZ=[t,rot];
    assignin('base','CoordenadaX',CoordenadaX);assignin('base','CoordenadaY',CoordenadaY);
    assignin('base','CoordenadaZ',CoordenadaZ);assignin('base','RotacionZ',RotacionZ);
    assignin('base','Pieza1',Pieza1.base);
    sim('Estacion_TFG_RodrigoSancho');
```

Y con esto se da por finalizada la explicación de estos dos primeros códigos para definir y simular la estación.

Kin.m

El siguiente archivo con gran relevancia es la clase 'Kin' con la que se crean las funciones necesarias para simular los movimientos más habituales del robot y otras características que no se comentaran.

Este es el único archivo que no ha sido creado por el autor, si no que fue creado por el tutor encargado de supervisar al autor en la realización de este proyecto (Alberto Herreros ⇨ albher@eis.uva.es).

A pesar de no haber sido desarrollado por el autor, se han realizado unas pequeñas modificaciones para adaptarlo al presente proyecto, por lo que no se comentará el código de la clase creada, pero si las funciones principales y las modificaciones.

Funciones:

- Creación del objeto de la clase 'this= Kin(robot, mdl, tool)'.
- Tool, Wobj, Pose, Joint, Show, Sim, Rec, Rep.
- Movimientos: MoveJ, MoveL, MoveAbsJ, MoveC.

En cuanto a los cambios, esta clase estaba pensada para ejecutarse con un modelo de 6 articulaciones móviles en Simulink, pero en este proyecto entre el robot y la movilidad del cubo, se tienen un total de 10 articulaciones (7 rotacionales y 3 prismáticas). Esto ha supuesto la necesidad de adaptación de código en detalles como el mostrado en la siguiente ilustración.

```
%this.q= [0,0,0,0,0,0];
this.q= [0,0,0,0,0,0,0,0,0,0];
```

OPC.m, OPC2.m y SeleccionColor.m

Con los dos primeros scripts se establece y desarrolla la comunicación OPC entre las variables de MATLAB y las de RobotStudio. Adicionalmente se ha desarrollado un fichero para realizar el cambio de formato entre estos dos programas, a la hora de poder leer una variable con los datos del color del cubo.

A continuación, se desarrolla la explicación de cada script de forma individual:

1. OPC.m

   La primera función que realiza este fichero, es el establecimiento de la conexión con el servidor OPC creado previamente. Para ello se han utilizado instrucciones como 'opcserverinfo' y ' .ServerID', en donde se muestran datos como la información del servidor y los servidores disponibles.

Una vez se sabe a qué servidor conectarse, se crea el cliente y se conecta, con las instrucciones 'opcda' y 'connect'.

```
% Inicialización de la comunicación OPC
InformacionHost = opcserverinfo('localhost'); %Informacion del Host
ServidoresDisponibles = InformacionHost.ServerID'; %Servidores Disponibles
Cliente = opcda('localhost', 'ABB.IRC5.OPC.Server.DA'); %Crear un cliente
connect(Cliente) %Conectar el cliente creado
Variables = addgroup(Cliente);%Variables modificadas con la comunicación
% Si se desea ver las variables que tiene el servidor, con un cierto nombre
% VerItemsServidor = serveritems(Cliente, '*NombreVariable*');
assignin('base','VariablesRobotStudio',Variables);
```

Con la conexión establecida, ya se podrán trasmitir datos, pero lógicamente hay que configurar previamente, que variables van a ser modificadas. Por ello se declara un grupo de variables con 'addgroup', en el que se contienen todas las variables a modificar del otro lado de la conexión (en este caso Robotstudio).

El siguiente paso sería definir todas estas variables con el nombre de RobotStudio y añadirlas al grupo creado ('Variables'). Cabe destacar que pueden declararse tanto variables de Rapid, como señales del controlador del robot.

```
% Señales y Variables RobotStudio
PosInicial_Cubo=additem(Variables,'DESKTOP-F0D612S_IRB_120_3kg_058m.RAPID.T_ROB1.Maniobra_IRB120.PosInicial_Cubo');
Pos_Cubo=additem(Variables,'DESKTOP-F0D612S_IRB_120_3kg_058m.RAPID.T_ROB1.Maniobra_IRB120.Pos_Cubo');
Color_Cubo=additem(Variables,'DESKTOP-F0D612S_IRB_120_3kg_058m.RAPID.T_ROB1.Maniobra_IRB120.Color_Cubo');
TamanoX=additem(Variables,'DESKTOP-F0D612S_IRB_120_3kg_058m.RAPID.T_ROB1.Maniobra_IRB120.TamañoX');
TamanoY=additem(Variables,'DESKTOP-F0D612S_IRB_120_3kg_058m.RAPID.T_ROB1.Maniobra_IRB120.TamañoY');
TamanoZ=additem(Variables,'DESKTOP-F0D612S_IRB_120_3kg_058m.RAPID.T_ROB1.Maniobra_IRB120.TamañoZ');
CrearCubo=additem(Variables,'DESKTOP-F0D612S_IRB_120_3kg_058m.IOSYSTEM.IOSIGNALS.CrearCubo');
```

Una vez establecida la conexión y declaradas las variables a modificar, solo restaría conocer los datos que se desean introducir en estas, por ello se utilizara la instrucción 'evalin' que ya se ha comentado anteriormente, con la que se evaluaban los datos de una variable contenida en el espacio de trabajo (workspace) de MATLAB.

```
% Evalua los parámetros asignados por el Usuario desde la interfaz, para
% poder escribirlos posteriormente en Robotstudio y simular la pieza
% deseada
Color = evalin('base','Color');
ColorDeseado = evalin('base','ColorDeseado');
Size = evalin('base','Size');
```

Para escribir los datos obtenidos en las variables de RobotStudio, se utiliza la instrucción 'write' como se muestra en el siguiente fragmento de código.

```
% Reset Señales
write(CrearCubo,0);write(CambiarColorCubo,0);write(PiezaNoValida,0);
write(SeleccionT1,0);write(SeleccionT2,0);write(SeleccionT3,0);
write(PiezaAnalizada,0);

% Escribe el modo en Robotstudio que ha sido seleccionado por el usuario
write(SeleccionT1,SeleccionModo);write(SeleccionT2,SeleccionModo);write(SeleccionT3,SeleccionModo);
```

Tras la escritura de todas las variables necesarias (en la anterior imagen solo se mostraron algunas), el siguiente punto a destacar es la invocación de 2 scripts. El primero ya se comentará posteriormente y sirve para cambiar el formato de la variable con el color y en el segundo se ejecuta 'EstacionMATLAB', para realizar la simulación de la célula en dicho software.

```
% Función para cambiar el formato del Color definido
% Por ejemplo: RobotStudio=[255000,0,0] y MATLAB=[1,0,0];
[ColorFormato1]=SeleccionColor(Color);

% Se ejecuta la simulación de la pieza en MATLAB y se consiguen los
% parámetros obtenidos por visión artificial
[ColorDetectado,AreaDetectada,PosicionDetectada] = EstacionMATLAB(ColorFormato1,Size,PosicionX);
```

Tras obtener los parámetros del cubo con visión artificial, hay que realizar otra conversión de variables, para adecuarse con los datos manejados en los programas. Adicionalmente se escriben variables, en las que se indica el tamaño del cubo, por lo que hay que realizar un filtro a través de una sentencia condicional 'if...elseif...else' en la que se analiza el tamaño detectado.

```
% Conversión de variables en función del tamaño detectado
if (AreaDetectada>0 & AreaDetectada<=30)
    SizeDetectado='Pequeño';
    Altura=100;
    PosicionDetectada = mapfun(PosicionDetectada,12,21,1490,1690);
    assignin('base','PosicionDetectada',PosicionDetectada-1590);
elseif (AreaDetectada>=70)
    SizeDetectado='Grande';
    Altura=200;
    PosicionDetectada = mapfun(PosicionDetectada,13.1786,19.8772,1490,1590);
    assignin('base','PosicionDetectada',PosicionDetectada-1540);
elseif (AreaDetectada<70 & AreaDetectada>30)
    SizeDetectado='Mediano';
    Altura=150;
    PosicionDetectada = mapfun(PosicionDetectada,14.3711,18.5,1490,1640);
    assignin('base','PosicionDetectada',PosicionDetectada-1565);
else
    SizeDetectado='NULL';
    Altura=0;
    assignin('base','PosicionDetectada',0);
end
```

Una vez se han obtenido todos los parámetros del cubo, hay que compararlos con los deseados inicialmente por el usuario, para determinar la validez de la pieza. Esto se realiza con sentencias condicionales 'if...else' y con sentencias de control de flujo 'switch...case', que determinan el valor de dos variables booleanas (ColorCorrecto, SizeCorrecto).

```
% Sentencias condicionales para determinar si el tamaño y el color
% detectado son los correctos
if length(ColorDetectado)==length(ColorDeseado)
    if ColorDetectado==ColorDeseado
        ColorCorrecto=true;
    else
        ColorCorrecto=false;
    end
else
    ColorCorrecto=false;
end

SizeCorrecto=false;
switch SizeDeseado
    case 'Pequeño'
        if AreaDetectada<=30
            SizeCorrecto=true;
        end
    case 'Mediano'
        if AreaDetectada>30 & AreaDetectada<70
            SizeCorrecto=true;
        end
    case 'Grande'
        if AreaDetectada>=70
            SizeCorrecto=true;
        end
end
```

El resto de código serían asignaciones de variables y escritura de estas en el otro lado de la conexión OPC, por lo que no merece la pena mencionarlas.

Analizando los fragmentos de código anteriores, el lector puede apreciar como aparece la función 'mapfun' que en realidad no pertenece a la librería de MATLAB. Esto fue un recurso del autor, en el cual se creó una función emulando la instrucción 'map' en el lenguaje 'Arduino'.

Esta función se encarga de establecer equivalencias entre números con diferentes rangos, de manera que era muy útil para la conversión de variables que se comentaba anteriormente.

```
%Funcion Map de Arduino
function output = mapfun(value,fromLow,fromHigh,toLow,toHigh)
narginchk(5,5)
nargoutchk(0,1)
output = (value - fromLow) .* (toHigh - toLow) ./ (fromHigh - fromLow) + toLow;
end
```

2. OPC2.m

En el anterior 'script' ya se ha realizado toda la primera etapa de la simulación y se ha determinado la validez de la pieza detectada.

En este segundo código, se busca finalizar la segunda etapa de la simulación. Por ello, en primer lugar, se leen las variables que determinan si los parámetros son correctos y en función de estas, se escribe una nueva variable indicativa de si es necesario descartar o no la pieza.

Tras determinar si hay que descartar la pieza, este dato se transmite a RobotStudio (escritura de variables) y a Matlab (Ejecución de EstacionMatlab2.m), y se realizaran las acciones oportunas en dichos softwares.

```
% Evalua del WorkSpace las variables a utilizar
SizeCorrecto = evalin('base','SizeCorrecto');
ColorCorrecto = evalin('base','ColorCorrecto');

% Determina si la Pieza es correcta o en caso contrario se descarta
if SizeCorrecto==true & ColorCorrecto==true
    DescartarPieza=0;
else
    DescartarPieza=1;
    fprintf('PiezaIncorrecta\n')
    fprintf('Descartando...')
end

% Escribe en RobotStudio si la pieza era correcta
write(PiezaNoValida,DescartarPieza);
write(PiezaAnalizada,1);

% Ejecución en MATLAB de la segunda parte de la simulación, en el que se
% selecciona la pieza o se descarta
EstacionMATLAB2(DescartarPieza);
```

Una vez se ha completado la simulación, se desconecta el cliente, para que no pueda ocasionar ningún problema en la comunicación OPC.

Con las instrucciones 'disconnect', 'delete', 'clear' y 'close' se realiza esta desconexión y borrado de variables.

```
% Desconexion y Borrado de la comunicación OPC. Limpieza de variables OPC
disconnect(Cliente)
delete(Cliente)
clear Cliente Variables
close(gcf)
```

3. SeleccionColor.m

Como ya se había adelantado previamente, en este script se realiza la conversión de la variable 'Color'. Para determinar un color se pueden utilizar varios métodos, en este caso se ha basado en RGB(Red-Green-Blue), que es la composición del color en términos de la intensidad de los colores primarios de la luz.

Por ello para determinar cualquier color se necesita un vector tridimensional, pero la diferencia surge en los valores que admite cada software de la intensidad de estos colores.

- MATLAB/Simulink: Rango de 0-1
- RobotStudio: Rango de 0-255

Esta conversión se realiza con un 'switch', en el que se determina en cada caso un color. Así se podrían haber añadido todos los colores que se quisiesen.

```
function [ColorOut]=SeleccionColor(Color)
    switch Color
        case '[255000,0,0]'
            ColorOut=[1,0,0];
        case '[0,255000,0]'
            ColorOut=[0,1,0];
        case '[0,0,255000]'
            ColorOut=[0,0,1];
        case '[255000,255000,0]'
            ColorOut=[1,1,0];
        case '[255000,0,255000]'
            ColorOut=[1,0,1];
        case '[0,255000,255000]'
            ColorOut=[0,1,1];
        case '[0,0,0]'
            ColorOut=[0,0,0];
    end
end
```

Algoritmo_VisionArtificial.m

En este último 'script' se desarrolla el algoritmo creado para analizar una imagen, en esa imagen centrarse en la región de interés y poder determinar de dicha región, la existencia de una pieza cúbica. Si se detecta la pieza, se extraen los valores de la posición ocupada, el área de un lado y su color.

El algoritmo se trata de una función en la que el parámetro de entrada es una imagen, y los parámetros de salida son el color, la posición y el tamaño del cubo, como se puede visualizar en el siguiente fragmento de código.

```
function [Color,Area,Posicion] = Algoritmo_VisionArtificial(Imagen)
```

Cabe destacar que se distinguen dos códigos diferentes, en el desarrollo de este algoritmo, el primero es para la detección del cubo y su posición, y el segundo es para la detección del color de la pieza. A continuación, se explicarán cada uno de ellos por separado.

Detección Área y Posición

El primer paso en el tratamiento de la imagen es transformarla a escala de grises. Para ello se ven 2 métodos:

- Instrucción rgb2gray: Convierte imagen RGB a escala de grises
- Instrucciones 'graythresh' y 'im2bw': Obtener el umbral de imagen global utilizando el método de Otsu y convertir imagen a imagen binaria, basada en el umbral obtenido.

Una vez se tiene la imagen en escala de grises hay que asignar la zona de interés sobre la que se va a trabajar, por ello se han definido unas coordenadas predeterminadas, consideradas las más óptimas por el autor del proyecto, para que utilizando la instrucción 'imcrop', la imagen resultante sea la zona de interés únicamente.

Como en la interfaz se dispone de una herramienta para cambiar estas coordenadas por parte del usuario, si se selecciona esta opción (configuraEspacioVision=1), aparecerá la imagen para que este elija la región de corte y después introduzca las coordenadas en la interfaz.

```
ConfiguraEspacioVision = evalin('base','ConfiguraEspacioVision');
CoordenadasCorte=[187.5 127.5 30 45]; %Coordenadas de la zona de interes
% Método 1 de pasar la imagen a escala de grises
Img=rgb2gray(Imagen);
% Método 2 de pasar la imagen a escala de grises (con umbral)
umb=graythresh(Imagen);
ImgGray=im2bw(Imagen,umb);

% Configuración Zona a Detectar si el Usuario ha seleccionado esta opción
if ConfiguraEspacioVision==1
    imcrop(ImgGray);
    CoordenadasCorte = evalin('base','CoordenadasCorte');
end

%Recorte Zona interes
ImgGrayCut=imcrop(ImgGray,CoordenadasCorte);
```

El siguiente paso que se realiza para el procesamiento del cubo es la detección de todos los bordes que pueda encontrar en la imagen, para ello se utiliza la instrucción 'edge' que soporta diferentes métodos para calcular estos bordes.

[21] El método usado ha sido 'canny' que según la guía de MATLAB se define:

> Encuentra bordes buscando el máximo local del gradiente de .I La función calcula el degradado utilizando la derivada de un filtro gaussiano.edge. Este método utiliza dos umbrales para detectar bordes fuertes y débiles, incluidos los bordes débiles en la salida si están conectados a bordes fuertes. Mediante el uso de dos umbrales, el método Canny es menos probable que los otros métodos se dejen engañar por el ruido y, más es probable, detectar bordes débiles verdaderos.

Una vez se detectan los bordes, se rellenan todos los huecos que estén rodeados de bordes, de manera que así se pueda visualizar el cubo como un área y no como un conjunto de líneas.

Con la instrucción 'label2rgb' se dota de distintos colores estos rellenos (se ha incluido con intenciones de mejorar únicamente el aspecto visual de la imagen tratada). Pero para detectar los componentes conexos que se han rellenado se hacen uso de las funciones 'bwconncomp' y 'labelmatrix'.

```
% Deteccion de Bordes de la imagen con método 'Canny'
ImgBordes=edge(ImgGrayCut,'Canny');
% Método 2: Rastrear los límites de la región en la imagen binaria
[B,L]=bwboundaries(ImgGrayCut,'noholes');

% Rellenar huecos de la imagen anterior en la que se detectaron los bordes
ImgRellenada=imfill(ImgBordes,'holes');

% Tratamiento de la imagen, para marcar el cubo detectado y nombrarle
f=figure('visible','off');
S6=subplot(1,2,1);
set(S6,'Visible','off');

% Encuentra componentes conexas y las asigna etiquetas (1, 2, 3...)
% Devuelve una estructura con diversos campos, entre ellos:
%      cc.NumObjects:  número de objetos encontrados
CC = bwconncomp(ImgRellenada);

% L:  imagen de etiquetas  (0 fondo; 1, 2... las distintas comp. conexas)
% P.ej.: para separar objetos y rotarlos
L=labelmatrix(CC);

% Muestra imagen de etiquetas
% ('jet, 'k' para fondo negro)
RellenoColores=label2rgb(L, 'jet', 'k');
imshow(RellenoColores);
```

Hasta ahora se ha realizado un tratamiento de imagen, pero lo que interesa en este proyecto es poder detectar el cubo. Para ello se utiliza la instrucción '*regionprops*' con la que se miden todas las propiedades de las regiones de la imagen procesada.

```
% Medir las propiedades de las regiones de la imagen procesada
propied=regionprops(ImgRellenada,'basic');
```

Una vez se tienen las propiedades, hay que recorrer la variable que las contiene, para remarcar todos los objetos que encuentre con un rectángulo rojo, y si detecta el cubo, remarcarlo en un rectángulo verde y asignarle un texto indicativo ('Cubo').

Entre las propiedades que se pueden obtener del cubo, en el presente proyecto interesan el área y la posición o centroide de este.

```
% Bucle para detectar todos los objetos rectangulares de la imagen y
% marcarlos. Si detecta el cubo lo marca en verde y le asigna un nombre,
% los demás objetos serán marcados en rojo.
% Además, se obtiene el tamaño y posición (Centroide) del cubo detectado
for n=1:size(propied)
    rectangle('Position',propied(n).BoundingBox,'EdgeColor','r','LineWidth',2);
    centroide=propied(n).Centroid;
    Area=propied(n).Area;
    plot(centroide(1),centroide(2),'ko');
    if(centroide(1)<25 & centroide(1)>10 & centroide(2)<30 & centroide(2)>15 & Area<105)
        rectangle('Position',propied(n).BoundingBox,'EdgeColor','g','LineWidth',5);
        text(centroide(1)-4,centroide(2)+7,'Cubo','Color','g','Fontsize',15,'Fontweight', 'bold');
        AreaFigura=propied(n).Area;
        X=centroide(1);
        Y=centroide(2);
    end
end
```

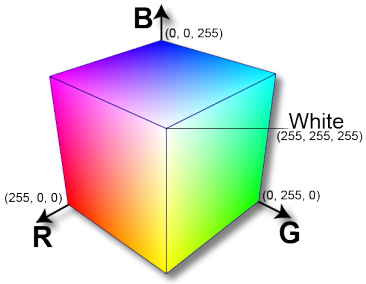
Detección Color

Hasta ahora se ha conseguido tratar la imagen y obtener unos parámetros del cubo, pero hay que recordar que todavía falta por detectar un parámetro fundamental que es el color.

Para este propósito se ha desarrollado un sencillo algoritmo que se basa en las siguientes suposiciones:

- En la zona de interés de la imagen capturada en la simulación de MATLAB, solo existen tres colores, el blanco del suelo, el gris de la cinta transportadora y el naranja del robot IRB120.
- El cubo no puede ser de alguno de los colores mencionados en el anterior punto.
- La imagen siempre se toma desde la misma altura.

Con esto se define el algoritmo como un recorrido pixel a pixel de toda la zona de interés (buscando en las 3 capas RGB de una imagen), en el que, si encuentra una serie de valores en el mismo pixel en las 3 capas, puede determinar con cierta fiabilidad de que se trata de un cierto color.



Indice	Color	RGB	HEX	Sistema
0		0, 0, 0	#000000	Black
1		255, 255, 255	#FFFFFF	White
2		255, 0, 0	#FF0000	Red
3		0, 255, 0	#00FF00	Green
4		0, 0, 255	#0000FF	Blue
5		255, 255, 0	#FFFF00	Yellow
6		255, 0, 255	#FF00FF	Magenta
7		0, 255, 255	#00FFFF	Cyan
8		128, 0, 0	#800000	
9		0, 128, 0	#008000	
10		0, 0, 128	#000080	
11		128, 128, 0	#808000	
12		128, 0, 128	#800080	
13		0, 128, 128	#008080	
14		192, 192, 192	#C0C0C0	
15		128, 128, 128	#808080	

Indice	Color	RGB	HEX	Sistema
16		153, 153, 255	#9999FF	
17		153, 51, 102	#993366	
18		255, 255, 204	#FFFFCC	
19		204, 255, 255	#CCFFFF	Relleno de gráficos
20		102, 0, 102	#660066	
21		255, 128, 128	#FF8080	
22		0, 102, 204	#0066CC	
23		204, 204, 255	#CCCCFF	
24		0, 0, 128	#000080	
25		255, 0, 255	#FF00FF	
26		255, 255, 0	#FFFF00	
27		0, 255, 255	#00FFFF	Líneas de gráfico
28		128, 0, 128	#800080	
29		128, 0, 0	#800000	
30		0, 128, 128	#008080	
31		0, 0, 255	#0000FF	

Indice	Color	RGB	HEX	Sistema
32		0, 204, 255	#00CCFF	
33		204, 255, 255	#CCFFFF	
34		204, 255, 204	#CCFFCC	
35		255, 255, 153	#FFFF99	
36		153, 204, 255	#99CCFF	
37		255, 153, 204	#FF99CC	
38		204, 153, 255	#CC99FF	
39		255, 204, 153	#FFCC99	
40		51, 102, 255	#3366FF	
41		51, 204, 204	#33CCCC	
42		153, 204, 0	#99CC00	
43		255, 204, 0	#FFCC00	

Indice	Color	RGB	HEX	Sistema
44		255, 153, 0	#FF9900	
45		255, 102, 0	#FF6600	
46		102, 102, 153	#666699	
47		150, 150, 150	#969696	
48		0, 51, 102	#003366	
49		51, 153, 102	#339966	
50		0, 51, 0	#003300	
51		51, 51, 0	#333300	
52		153, 51, 0	#993300	
53		153, 51, 102	#993366	
54		51, 51, 153	#333399	
55		51, 51, 51	#333333	

Figura 102. *Gama de colores RGB y sus códigos.* [38]

Para recorrer toda la imagen se utilizan dos bucles 'for' (filas y columnas) y por cada pixel se realizan sentencias condicionales 'if' para determinar si es del valor adecuado de un color. El código se muestra en la siguiente ilustración:

```
[Filas,Columnas] = size(ZonaInteres);
for i=1:Filas
    for j=1:(Columnas/3)
        VectorColores=impixel(ZonaInteres,j,i);
        %%%%%ROJO%%%%%
        if VectorColores(1)>100 & VectorColores(2)<40 & VectorColores(3)<40
            ColorFigura='Rojo';
        %%%%%VERDE%%%%%
        elseif VectorColores(1)<40 & VectorColores(2)>100 & VectorColores(3)<40
            ColorFigura='Verde';
        %%%%%AZUL%%%%%
        elseif VectorColores(1)<40 & VectorColores(2)<40 & VectorColores(3)>100
            ColorFigura='Azul';
        %%%%%AMARILLO%%%%%
        elseif VectorColores(1)>100 & VectorColores(2)>100 & VectorColores(3)<40
            ColorFigura='Amarillo';
        %%%%%MAGENTA%%%%%
        elseif VectorColores(1)>100 & VectorColores(2)<40 & VectorColores(3)>100
            ColorFigura='Magenta';
        %%%%%CYAN%%%%%
        elseif VectorColores(1)<40 & VectorColores(2)>100 & VectorColores(3)>100
            ColorFigura='Cyan';
        %%%%%NEGRO%%%%%
        elseif VectorColores(1)<40 & VectorColores(2)<40 & VectorColores(3)<30
            ColorFigura='Negro';
        end
    end
end
```

Con esto se habría conseguido determinar el color del cubo, pero se han añadido otra serie de tratamientos adicionales, para completar la detección de colores con visión artificial.

Aunque cabe destacar, que solo funcionan para los casos en el que el cubo tenga alguno de los colores primarios (Rojo, Azul o Verde).

- **Máscaras RGB:**

  Como una imagen tiene 3 capas en las que se contienen los pixeles de cada color primario, se puede enmascarar cada parte de la imagen de estos colores de la siguiente manera, se guardan en unas variables cada capa por separado, y después para el color que se desea representar, se restan las otras dos capas a la capa de ese color.
  De esta manera el color se mostrará en la imagen en blanco y el resto de colores junto al fondo en color negro.

```
% Divide las 3 capas de una imagen RGB
imR=double(ZonaInteres(:,:,1));
imG=double(ZonaInteres(:,:,2));
imB=double(ZonaInteres(:,:,3));
% Se eliminan el resto de colores de la imagen excepto Red ó Green ó Blue
% Creando unas máscaras para esos colores
% Fondo: Negro y Color correspondiente: Blanco
imagenR=imR-imG-imB;
imagenG=imG-imR-imB;
imagenB=imB-imR-imG;
```

- **Aislar un color (RGB):**

  En este caso una vez que se tienen las máscaras anteriores, hay que realizar un filtrado a cada imagen (‘medfilt2’) y aplicar las máscaras obtenidas a los dos colores restantes que no se desean mostrar en la imagen.

  De esta forma a diferencia del método anterior, el fondo y los demás elementos de la imagen se muestran en tonos de grises y el color deseado se muestra correctamente.

  Para concatenar las arrays con los valores de los pixeles en las diferentes capas se utiliza la instrucción ‘cat’ de MATLAB.

```
imagen_binaria=imagenR>50;
imagen_binaria1=imagenG>50;
imagen_binaria2=imagenB>50;

% Se realiza el filtrado medio de la imagen en dos dimensiones
img_filt_bin=medfilt2(imagen_binaria);
img_filt_bin1=medfilt2(imagen_binaria1);
img_filt_bin2=medfilt2(imagen_binaria2);

% Se filtran las imagenes, para mantener unicamente el color buscado
% (R,G,B)
mascara=1-img_filt_bin;
mascara1=1-img_filt_bin1;
mascara2=1-img_filt_bin2;

imagen_roja=double(ZonaInteresGris)/255;
imagen_verde=double(ZonaInteresGris) .* mascara/255;
imagen_azul=double(ZonaInteresGris) .* mascara/255;
img_final=cat(3,imagen_roja,imagen_verde,imagen_azul);
```

Con esto se da por concluida la explicación de las partes fundamentales del código desarrollado en los ‘scripts’ de MATLAB. Si tras esta explicación el lector desea ver la estructura del código de uno de los ficheros, recuerde que este se encuentra en el *Anexo A.2. Ejemplo de código (scripts) en Matlab*.

CAPÍTULO 5.2.2. SIMULINK

Para simular la célula de trabajo, el robot y el cubo en movimiento, se comentó previamente que se necesitaba una programación por bloques en Simulink. Con lo cual en el siguiente capítulo se desarrollarán las partes fundamentales de esta programación.

El primer sistema que uno se encuentra al abrir el modelo de Simulink, se muestra en la siguiente ilustración (véase *Figura 103*), que consta de los siguientes subgrupos y elementos:

- Inicialización (Subconjunto): Contiene todos los bloques necesarios para iniciar el módelo.
- Base_link (Subconjunto): Crea un sistema de referencia base para todos los elementos de la célula.
- Weld Joint (Bloque): Representa una articulación de cero grados de libertad (Junta de Soldadura)
- WObjX (Subconjuntos): Contiene los diferentes modelos 3D, representados en el modelo.
- Robot IRB 120 (Subconjunto): Define las articulaciones y herramientas necesarias para la creación de este modelo del robot.
- Desplazamiento Pieza (Subconjunto): Define las articulaciones necesarias para poder desplazar y rotal a un objeto en el espacio cartesiano.

Nota:

Este modelo de Simulink no ha sido desarrollado únicamente por el autor del proyecto, si no que parte de un modelo proporcionado por el tutor (Alberto Herreros López) y se ha ido modificando sustancialmente para adaptarlo a este proyecto. Por lo que el desarrollo de este modelo en Simulink se puede considerar un trabajo colaborativo entre autor-tutor

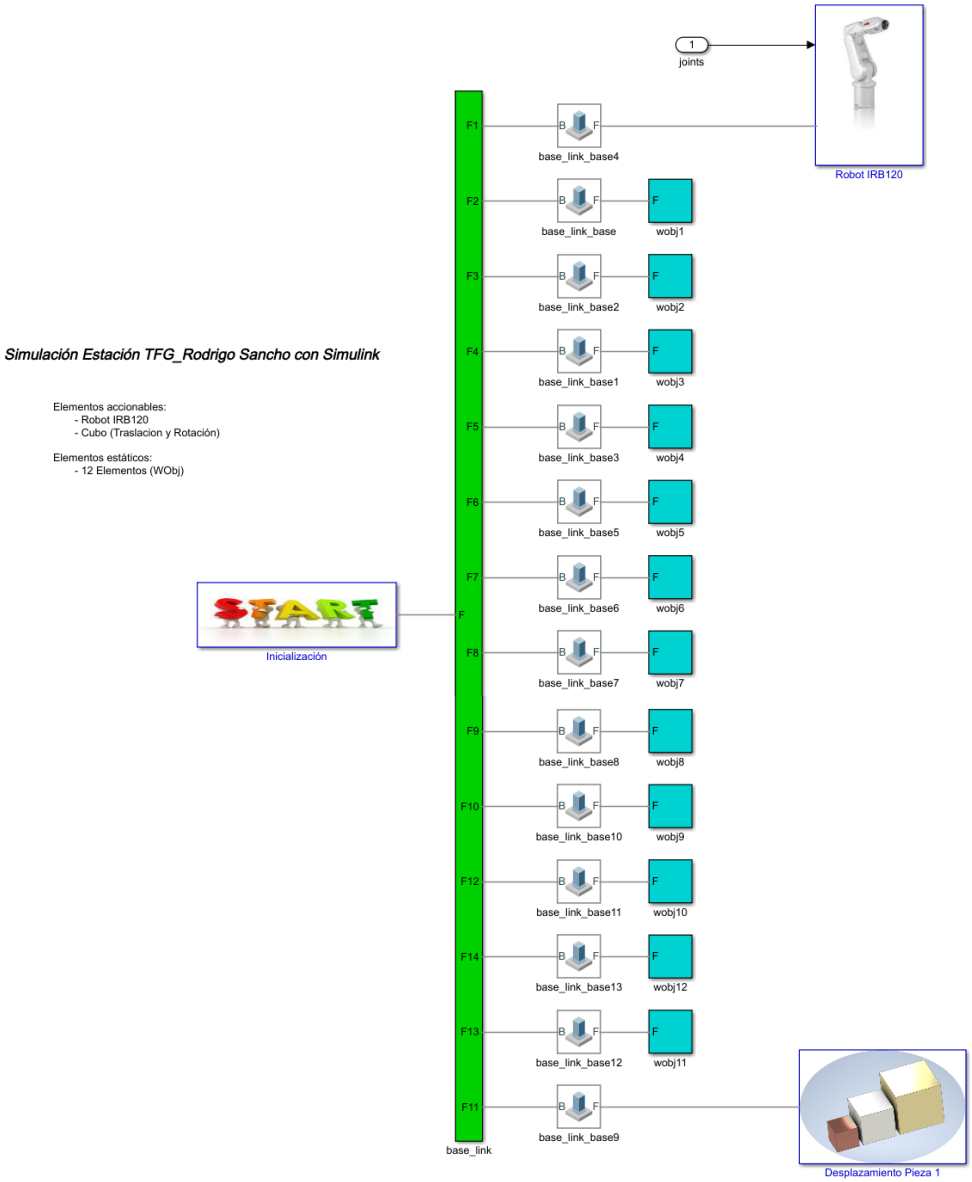


Figura 103. *Modelo en Simulink para la célula de trabajo desarrollada.*

A continuación, se explicarán con más detalle cada uno de los subconjuntos.

Inicialización

Para describir un modelo simulado en el espacio cartesiano con elementos mecánicos, se necesita un sistema de referencia general (1), un bloque para definir la configuración de los mecanismos (2) y otro para definir la configuración del solucionador utilizado para los cálculos necesarios en la ejecución (3). (Véase *Figura 104*, para situar cada número con su bloque).

Adicionalmente se ha añadido un bloque denominado 'spline', con el que se interpolan las trayectorias curvas realizadas por el robot IRB120.

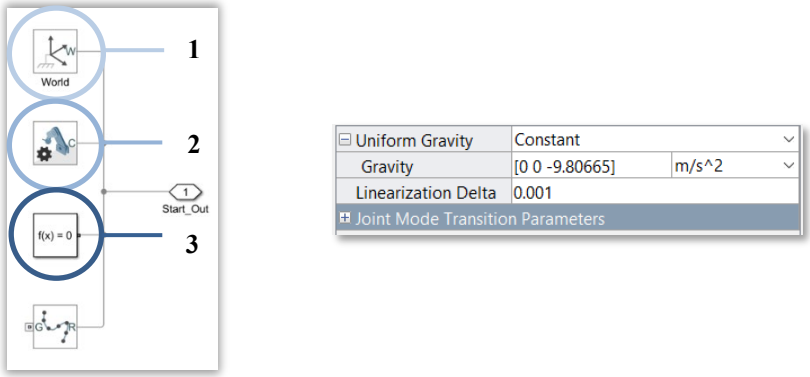


Figura 104. *Bloque de inicialización modelo Simulink.*

Base_Link

Para poder modificar el resto de sistemas de referencia se ha creado este subconjunto, en el que, a partir del sistema de referencia global, se puede trasladar cada sistema de referencia de cada objeto de manera individual.

Para ello como se aprecia en la *Figura 105*, se define un vector de 6 dimensiones, en el cual los 3 primeros valores definen la traslación y los 3 restantes la rotación. Este valor es el que se modificaba en los 'scripts' de 'EstacionMatlab', explicados en el capítulo anterior.

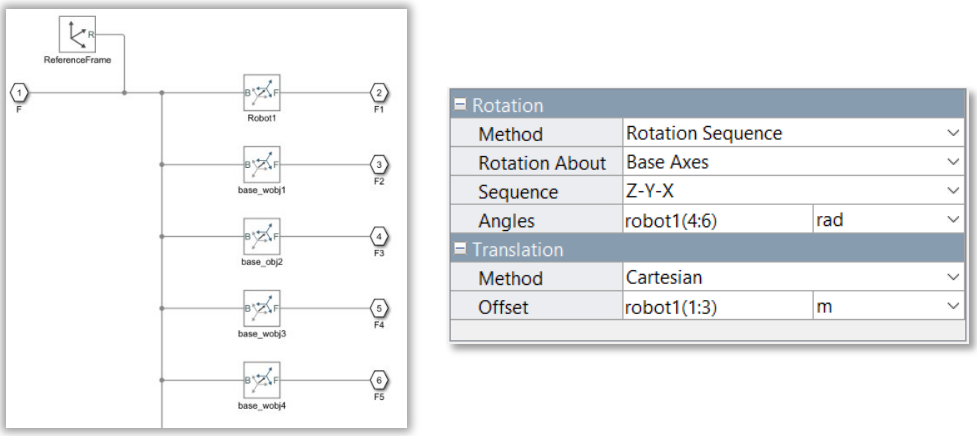


Figura 105. *Bloque definición de las bases de referencia del modelo Simulink.*

En la imagen solo se muestran algunos bloques, pero el patrón continúa hasta alcanzar los 14 elementos (12 objetos de trabajo, 1 robot y 1 pieza móvil).

Objetos de Trabajo (WObj x)

Con el fin introducir los modelos 3D, de los objetos que definen la estación, se han creado estos subconjuntos en los que se define cada objeto con una llamada a la carpeta que contiene todos los modelos. El bloque que hace esto posible se denomina 'File Solid'.

Como se muestra en la siguiente ilustración, en este ejemplo se define la ruta en la que se encuentra la cinta transportadora doble (como archivo '.stl'). Para optimizar el algoritmo de visión artificial este componente se ha declarado de color blanco en la simulación.

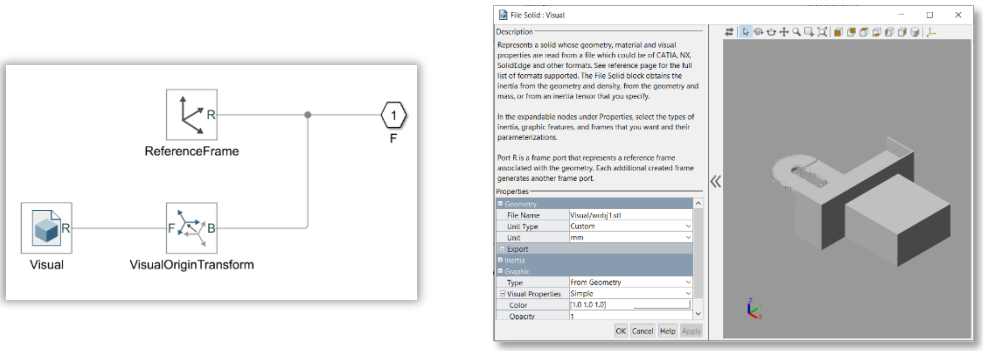


Figura 106. *Ejemplo de un bloque de los objetos de trabajo en el modelo Simulink.*

Con los otros dos bloques, se ajusta que el sistema de referencia definido para el objeto de trabajo, sea el mismo que el elemento 3D definido. Esta misma lógica de funcionamiento servirá para los 11 elementos restantes.

Robot IRB 120

En la creación del robot IRB 120, se han necesitado definir 6 articulaciones rotacionales (1 grado de libertad) para definir el movimiento de las articulaciones de este robot y 3 juntas de soldadura (sin grados de libertad), para definir la herramienta, la carga y el TCP de este.

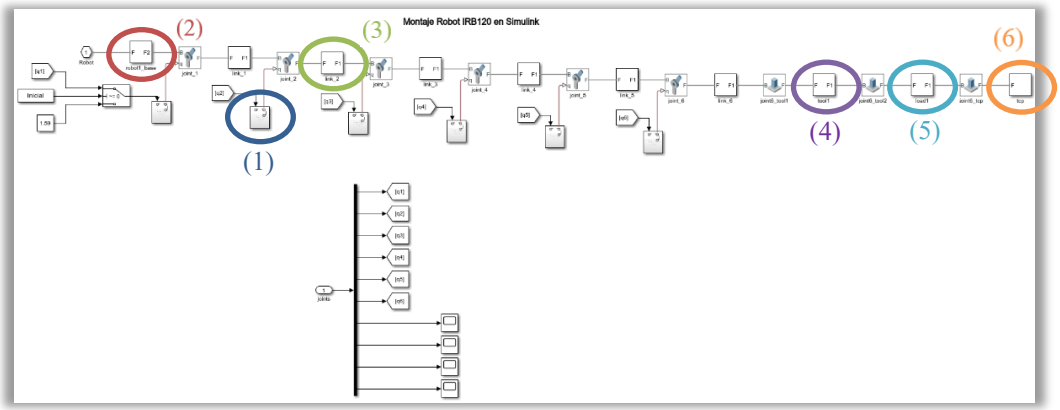
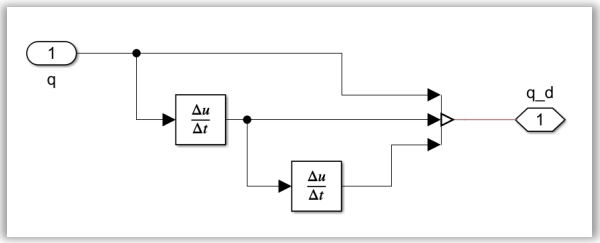


Figura 107. *Bloque donde se define al robot IRB120 en el modelo Simulink.*

En la programación de los robots se suele introducir el valor del ángulo en el que tiene que posicionarse cada articulación, para conseguir una cierta posición del TCP (cinemática directa), pero el único problema que surge en este modelo, es que las articulaciones rotacionales no admiten el valor del ángulo, por lo que como se muestra en la siguiente ilustración, es necesario realizar una doble derivación a este valor, para que introduciendo el ángulo, se ejecute el movimiento deseado.



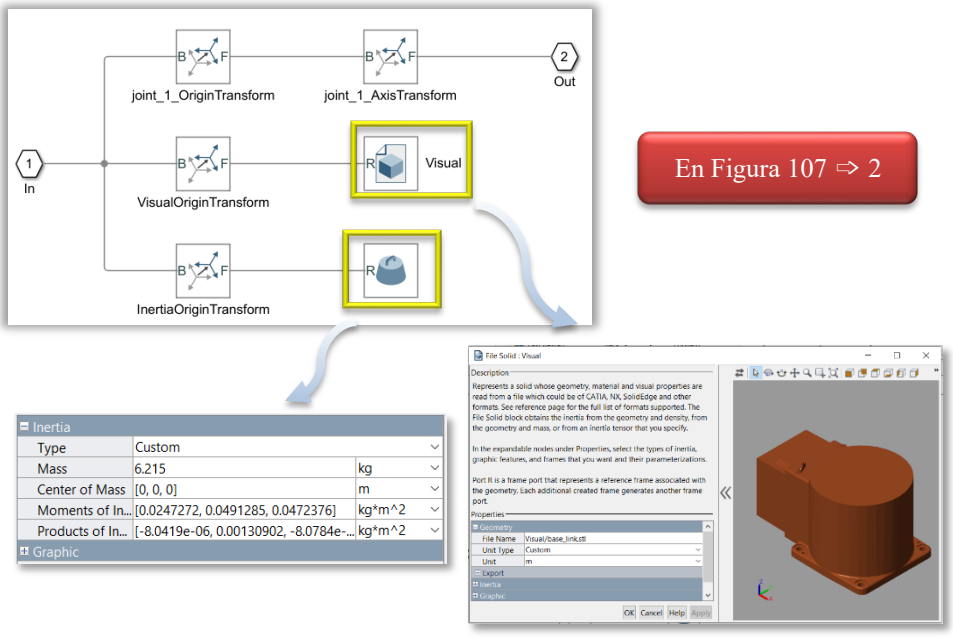
En Figura 107 ⇨ 1

Figura 108. *Bloque de derivación (Simulink).*

Si el lector analiza el esquema de bloques del robot, puede visualizar que entre cada articulación se encuentra un subgrupo en el que se define el sistema de referencia y las transformaciones necesarias, así como el componente 3D, que forma la articulación siguiente y sus características de inercia.

Estos subconjuntos son:

- ***Base del Robot***
  En este subconjunto se define la ruta del fichero con la base del robot en 3D y adicionalmente como se puede apreciar en la *Figura 109*, hay que definir una serie de parámetros para calcular la inercia del movimiento de esta parte.



En Figura 107 ⇨ 2

Figura 109. *Definición de la base del robot IRB120 (Simulink).*

- *Diferentes links entre articulaciones [1-6]*

  Cada conexión entre las 6 articulaciones disponibles, tiene la siguiente forma (véase *Figura 110*). La única diferencia entre ellos son el componente 3D que se introduce y los parámetros de inercia introducidos.

  En este caso como se están estableciendo unas conexiones entre los ejes, se han necesita incluir en la entrada y en la salida de cada subgrupo, un bloque denominado 'RigidTransform', para definir la transformación rígida entre los dos sistemas de referencia que se unen.

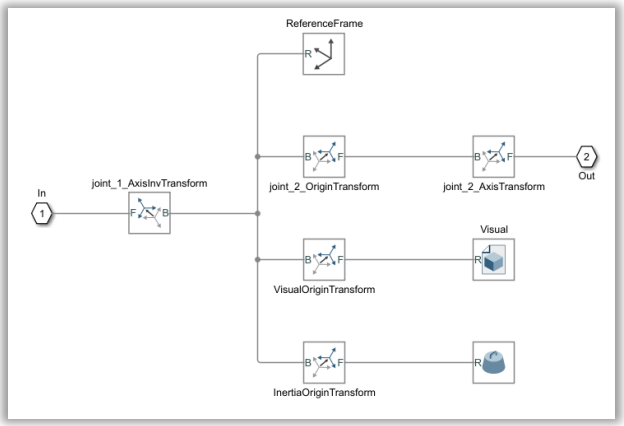


Figura 110. *Definición de una de las articulaciones del robot IRB120 (Simulink).*

- *Herramienta y Carga*

  Estos subconjuntos son muy similares al anterior, pero con la única diferencia de que en este caso no se necesitan los bloques de 'RigidTransform'. La herramienta no es un eje más del robot.

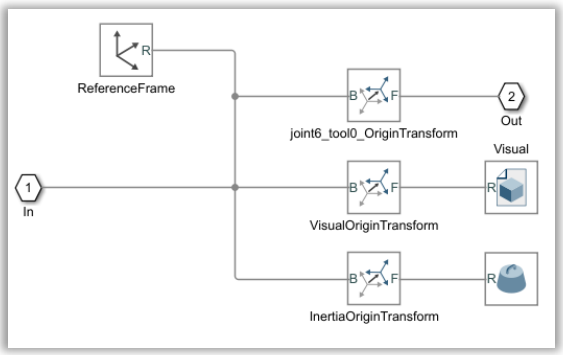
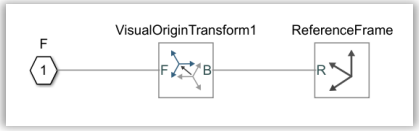


Figura 111. *Definición de la herramienta y del objeto cargado por robot IRB120 (Simulink).*

- *TCP*

  Finalmente, con una transformación adicional, se describe donde se sitúa el TCP o Punto Central de la Herramienta del Robot.



En Figura 107 ⇨ 6

Figura 112. *Definición del TCP del robot IRB120 (Simulink).*

Desplazamiento Pieza

Para realizar el desplazamiento de una pieza en el espacio (3D), se necesitan 3 articulaciones prismáticas, en las que se definen los diferentes ejes cartesianos. Además, como se tiene que permitir la rotación, se ha añadido una articulación rotacional.

Al igual que en el robot, para reconocer los valores introducidos con las posiciones deseadas, hay que realizar una doble derivación de este valor, antes de introducirlo en la articulación prismática/rotacional.

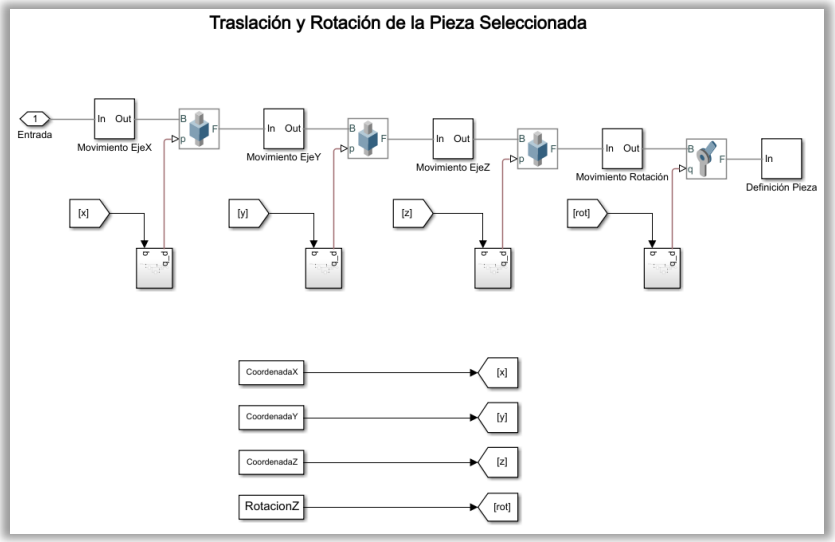


Figura 113. *Definición del movimiento de una pieza en Simulink (Rotación y Translación).*

Las conexiones entre estas articulaciones al no depender unas de otras, son más sencillas que en el robot y seguirán el siguiente modelo, en el que solo se define el origen de transformación y el propio eje.

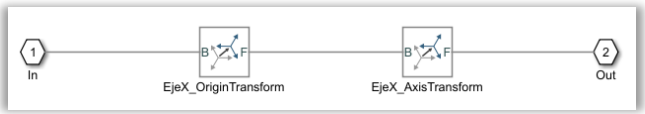


Figura 114. *Movimiento Eje X pieza Simulink.*

Tras definir todos los ejes y movimientos que se pueden realizar se asigna el objeto que va a ser desplazado en 'Definición Pieza', en el cual se define la ruta del objeto 3D a desplazar y los parámetros de inercia de este objeto.

CAPÍTULO 5.2.3. APP DESIGNER

Para finalizar la explicación de todo el desarrollo de software realizado, se ha dejado para este momento la programación de la interfaz de usuario. Ya que es el último componente de este puzle, con el cual el usuario puede manejar todo lo desarrollado anteriormente.

Gracias al uso de esta aplicación de MATLAB, la dificultad en el desarrollo de código, se ha reducido, ya que genera directamente unos fragmentos de código, simplemente al utilizar componentes de la librería. Por esto, no se mostrarán los fragmentos de código que se han autogenerado.

Aun así, todo el aspecto funcional ha tenido que ser desarrollado desde cero, y a pesar, de que es una aplicación que visualmente no permite grandes innovaciones, con un poco de creatividad e ingenio se ha conseguido desarrollar un aspecto semiprofesional para la interfaz.

Como en anteriores casos, antes de desarrollar un código lo primero es crear un diagrama de bloques en el que se muestre el funcionamiento de este. A continuación, se muestran el diagrama de bloques de las 4 ventanas de interfaz que se tienen.

La primera pertenece al menú principal, y los 3 restantes a una ventana secundaria, donde se muestran las diferentes imágenes obtenidas del algoritmo de visión artificial.

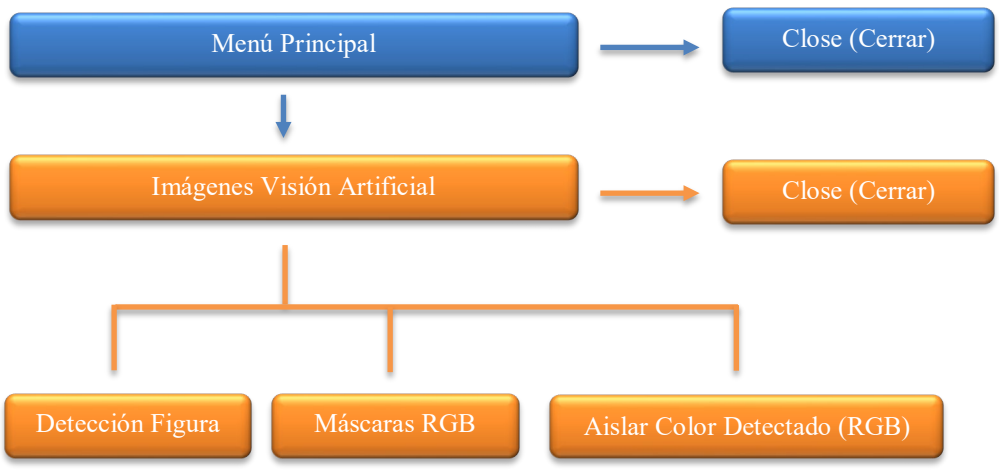


Figura 115. *Diagrama de bloques funcionamiento interfaz de usuario.*

Durante los siguientes puntos se irán desarrollando y explicando las funciones de cada interfaz.

Menú Principal

En este menú se muestran todas las opciones que el usuario puede modificar para realizar las simulaciones disponibles. Entre las opciones disponibles se encuentran:

- Interruptor para la activación de la interfaz
- Botón para abrir la ventana con las imágenes obtenidas con el procesamiento por visión artificial
- Botón de limpieza de variables (*Clean ALL*)
- Botones selectores entre las dos acciones disponibles
- Imágenes representativas de la acción seleccionada
- Menús despegables, para modificar los parámetros del cubo (tamaño y color)
- Slider para modificar la posición de origen del cubo
- Botón y campo rellenable para configurar la zona de interés de la visión artificial
- Botón para ejecutar la simulación
- Campos editables donde se indica los parámetros obtenidos por visión artificial y si la pieza es correcta o no.

La manera de utilizar esta interfaz por el usuario, a modo de manual introductorio se puede explicar con el siguiente diagrama de bloques:

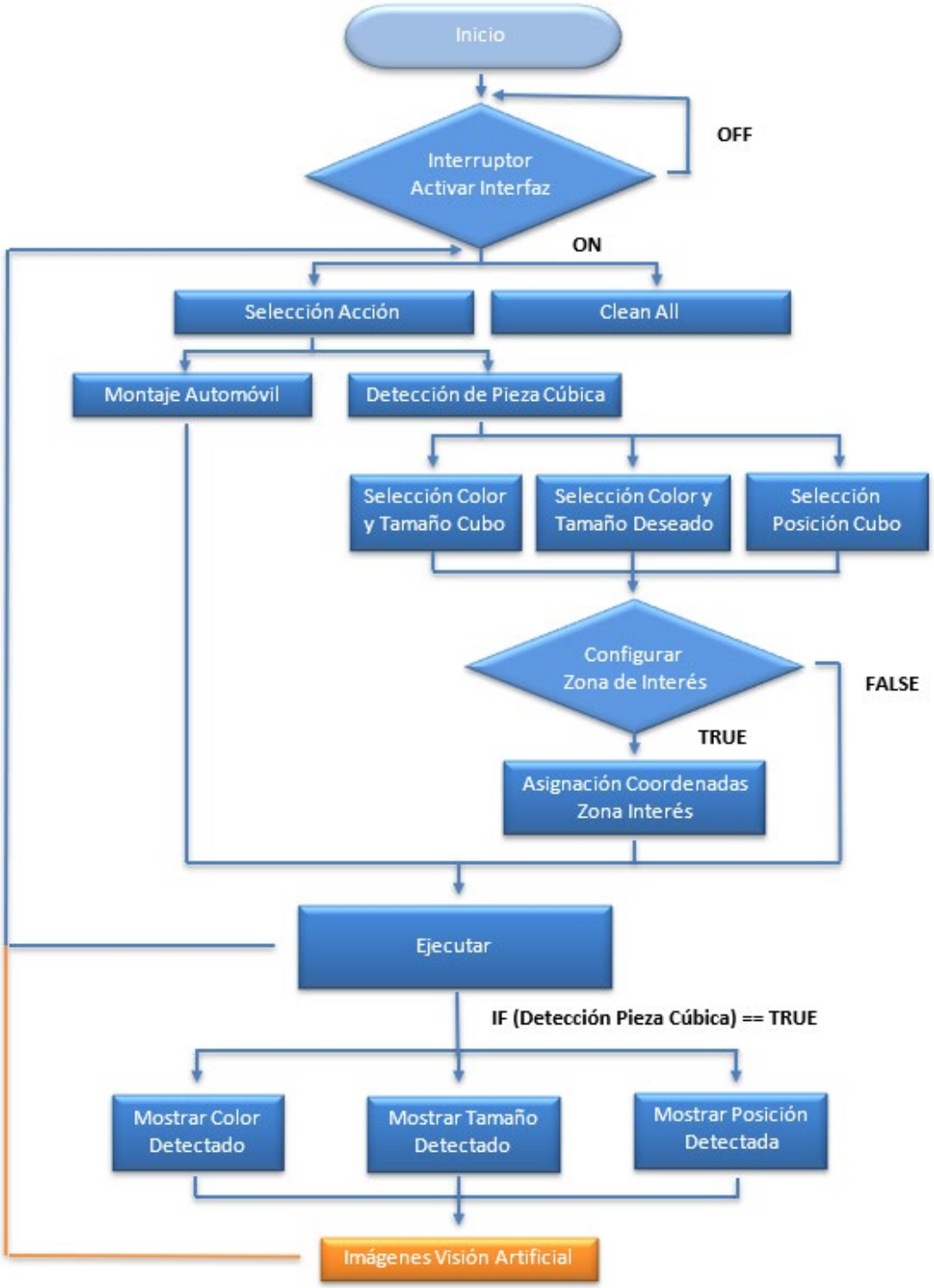


Figura 116. *Diagrama de bloques funcionamiento menú principal de la interfaz.*

Una vez se ha mostrado sus componentes y como ejecutarlo, se mostrará el proceso de programación realizado. Para visualizar el resultado de la interfaz, véase el siguiente *Capítulo 6.2 Resultados Matlab*.

Cuando se crea una interfaz, con esta aplicación, se está creando una clase predefinida, en la que se van añadiendo una serie de componentes. Por lo que la primera función de la clase que se comentará es la de inicio, la cual se ejecuta cada vez que se abre la interfaz.

StartupFcn(app)

La función encargada de iniciar la interfaz, tiene que ser capaz de desactivar todos los componentes, ya que se encuentra inicialmente apagada (por tanto, asignar a la lampara circular el color rojo). Estas características se modifican de la siguiente manera:

'app.NombreComponente.CaracteristicaCambiar'

Además, se inicializan las diferentes variables utilizadas a sus valores estándar. Esto es así, para el caso de que el usuario ejecute directamente la simulación sin seleccionar ningún parámetro, se pueda realizar esta simulación con los valores preestablecidos.

Posteriormente estos valores se copiarán al espacio de trabajo, que ya se comentó en anteriores capítulos, que servía como una especie de memoria, para las variables de MATLAB, utilizadas entre las distintas aplicaciones.

Todo esto se muestra en el siguiente fragmento de código:

```
function startupFcn(app)
    evalin('base', 'clear ALL')
    app.Lamp.Color='Red';
    app.ButtonGroup.Enable='Off';
    app.CleanAllButton.Enable='Off';
    app.ImagenesVisinArtificialButton.Enable='Off';
    app.SeleccionColor.Enable='Off';
    app.SeleccionSize.Enable='Off';
    app.SeleccionColorDeseado.Enable='Off';
    app.SeleccionSizeDeseado.Enable='Off';

                        ⋮

    Color='[255000,0,0]';
    Size=[0.15,0.15,0.15];
    Posicion='[1565,1000,200]';
    PosicionX=1590;
    assignin('base','Color',Color);
    assignin('base','Size',Size);
    assignin('base','Posicion',Posicion);
    assignin('base','SeleccionModo',1);
    assignin('base','PosicionX',PosicionX);
    assignin('base','ConfiguraEspacioVision',0);
    app.DeteccionFiguraButton.Value=false;
    assignin('base','ColorDeseado',app.SeleccionColorDeseado.Value);
    assignin('base','SizeDeseado',app.SeleccionSizeDeseado.Value);
end
```

El resto de funciones que se mostrarán a continuación, se definirán para cada componente utilizado.

Al utilizar los componentes de la librería, ya se definen de forma automática por lo que el programador tiene que escribir código únicamente en el 'callback' (función a ejecutar cuando se acciona ese componente), aunque ya se verá que hay diferentes funciones 'callback', en función del componente utilizado.

CallBack: Accionar Interruptor Encendido

Cada vez que se pulsa el interruptor se evalúa el estado en el que se encuentra con 'app.NombreComponente.Value' y con un 'switch' se realiza una acción u otra en función de si esta encendido o apagado.

Si está apagado, deshabilita todos los componentes y en caso contrario los habilita

```
% Value changed function: EncendidoSwitch
function EncendidoSwitchValueChanged(app, event)
    value = app.EncendidoSwitch.Value;
    assignin('base','Interruptor',value);
    switch value
        case 'On'
            app.Lamp.Color='Green';
            app.ButtonGroup.Enable='On';
            app.CleanAllButton.Enable='On';
            app.ImagenesVisinArtificialButton.Enable='On';

                        ⋮

        case 'Off'
            app.Lamp.Color='Red';
            app.ButtonGroup.Enable='Off';
            app.CleanAllButton.Enable='Off';
            app.ImagenesVisinArtificialButton.Enable='Off';
```

CallBack: Limpieza total (Clean All)

Cuando se acciona el botón de limpieza de variables, se activan las instrucciones encargadas de vaciar el contenido de todas estas variables y de borrarlas y posteriormente se reinicia la interfaz.

CallBack: Selección Acción

En este caso cada vez que se cambia entre los dos modos disponibles, con una sentencia condicional 'if' se guarda el valor seleccionado en la variable 'modo' y se remarca la acción seleccionada, para facilitar al usuario el manejo.

Como no existe la opción de remarcar una zona, hay que manipular dos fotos (uno con marca y otra sin ella), para dar la sensación de que se selecciona una de las acciones. En realidad, lo que se está haciendo es visualizar o esconder una imagen u otra.

```
% Selection changed function: ButtonGroup
function ButtonGroupSelectionChanged(app, event)
    %selectedButton = app.ButtonGroup.SelectedObject;
    assignin('base','Modo',app.MontajeCocheButton.Value);
    if app.MontajeCocheButton.Value==true
            assignin('base','SeleccionModo',1);
            app.CocheSeleccionado.Visible='On';
            app.CajasSeleccionado.Visible='Off';
    else
            assignin('base','SeleccionModo',2);
            app.CocheSeleccionado.Visible='Off';
            app.CajasSeleccionado.Visible='On';
    end
end
```

CallBack: Selección Color

Función encargada de actualizar el valor de la variable color con un formato comprendido por MATLAB, cada vez que se selecciona un color entre las distintas posibilidades de colores que ofrece la interfaz.

En el siguiente fragmento de cósigo se muestra como se ha realizado esto, en donde se ha utilizado un 'switch' para los distintos casos de colores, y con 'assignin' se asigna el valor a la variable en el espacio de trabajo.

```
% Value changed function: SeleccionColor
function SeleccionColorValueChanged(app, event)
    value = app.SeleccionColor.Value;
    switch value
        case 'Rojo'
            Color='[255000,0,0]';
        case 'Verde'
            Color='[0,255000,0]';
        case 'Azul'
            Color='[0,0,255000]';
        case 'Amarillo'
            Color='[255000,255000,0]';
        case 'Magenta'
            Color='[255000,0,255000]';
        case 'Cyan'
            Color='[0,255000,255000]';
        case 'Negro'
            Color='[0,0,0]';
    end
    assignin('base','Color',Color);
    assignin('base','SeleccionColor',app.SeleccionColor.Value);
end
```

CallBack: Selección Tamaño

En este caso la dinámica es muy similar al anterior callback, pero con dos ligeras diferencias. La primera es que la variable escrita tiene que tener un formato vectorial tridimensional, para determinar el tamaño (X, Y, Z) y la segunda es en relación del tamaño seleccionado por el usuario, ya que el 'slider' donde se determina la posición inicial del cubo, cambia de valores finales. Por ello se hace visible uno u otro slider (app.NombreSlider.Visible)

Esto se debe a que, en función del tamaño del cubo, en la cinta transportadora, se podrá colocar en mayor o menor número de posiciones.

Por ejemplo, el cubo grande ocupará mucho más espacio por lo que se puede posicionar en un menor número de sitios que el cubo pequeño o mediano.

En el siguiente fragmento solo se muestra una parte de la función para poder visualizar lo anterior.

```
% Value changed function: SeleccionSize
function SeleccionSizeValueChanged(app, event)
    value = app.SeleccionSize.Value;
    switch value
        case 'Pequeño'
            Size=[0.1,0.1,0.1];
            app.PosicionPequeno.Visible='On';
            app.PosicionMediano.Visible='Off';
            app.PosicionGrande.Visible='Off';
            Posicion='[1590,1000,200]';
        case 'Mediano'
            Size=[0.15,0.15,0.15];
            app.PosicionPequeno.Visible='Off';
            app.PosicionMediano.Visible='On';
            app.PosicionGrande.Visible='Off';
            Posicion='[1565,1000,200]';
```

CallBacks: Selección Color Deseado y Selección Tamaño Deseado

En este caso como no se escriben variables que modifican las características de un objeto 3D, no se requiere la conversión realizada en las anteriores funciones. Lo que supone que estas funciones simplemente se encargan de escribir en el espacio de trabajo de MATLAB el valor seleccionado por el usuario.

```
% Value changed function: SeleccionColorDeseado
function SeleccionColorDeseadoValueChanged(app, event)
    value = app.SeleccionColorDeseado.Value;
    assignin('base','ColorDeseado',value);
end

% Value changed function: SeleccionSizeDeseado
function SeleccionSizeDeseadoValueChanged(app, event)
    value = app.SeleccionSizeDeseado.Value;
    assignin('base','SizeDeseado',value);
end
```

CallBacks: Selección Posición (Tamaño Pequeño, Mediano y Grande)

Como se ha comentado previamente, la selección de la posición se realiza por medio de sliders, de los cuales existen 3 tipos, para los tres tamaños del cubo.

Las 3 funciones siguen la misma estructura, por lo que solo será necesario comentar uno de ellos (los demás cambian únicamente sus valores).

Cada vez que el usuario manipula el 'slider' se actualiza el valor de la variable que determina la posición, pero existe un inconveniente, ya que el valor que se obtiene es un número, y la variable leída por RobotStudio, tiene que ser una cadena de caracteres en la que se escribe un vector tridimensional.

Por tanto, en primer lugar, se convierte el número entero (al cual se suma una cierta cantidad para ajustarlo a los valores tratados en la simulación) en cadena de caracteres gracias al uso de la instrucción 'int2str'. Una vez se tiene el valor en cadena de caracteres, hay que añadirlo en la primera posición del vector tridimensional, para ello se utiliza 'strcat', instrucción para concatenar cadenas de caracteres.

Gracias a ese proceso, cada vez que el usuario cambia un valor, el valor guardado en el espacio de trabajo de MATLAB y que posteriormente será enviado a RobotStudio, tiene el formato adecuado para poder ser leído por estos programas.

```
% Value changing function: PosicionPequeno
function PosicionPequenoValueChanging(app, event)
    changingValue = event.Value;
    app.ValorPosicion.Value=changingValue;
    PosicionX=int2str(changingValue+1590);
    Posicion=strcat('[',PosicionX,',1000,200]');
    assignin('base','Posicion',Posicion);
    assignin('base','PosicionX',changingValue+1590);
    assignin('base','BarraPequena',changingValue);
end
```

CallBack: Botón Configurar Zona de Interés

Tiene el único objetivo de habilitar una variable, que permitirá en el algoritmo de visión artificial cambiar la zona de interés o mantener el valor preestablecido.

Para ello hay que determinar el estado del botón, si se acciona devolverá un valor TRUE, por lo que en ese momento hay que habilitar la variable.

```
% Value changed function:
% ConfigurarEspacioTrabajoVisinArtificialButton
function ConfigurarEspacioTrabajoVisinArtificialButtonValueChanged(app, event)
    value = app.ConfigurarEspacioTrabajoVisinArtificialButton.Value;
    if value==true
        assignin('base','ConfiguraEspacioVision',1);
    else
        assignin('base','ConfiguraEspacioVision',0);
    end
    assignin('base','BotonConfiguracion',app.ConfigurarEspacioTrabajoVisinArtificialButton.Value);
end
```

CallBack: Campo Editable Coordenadas Zona de Interés

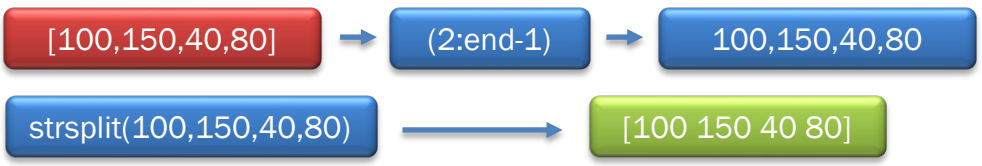
Si se ha seleccionado el botón para editar la zona de interés, en este campo editable se pueden copiar las coordenadas que se obtengan de elegir la nueva zona de interés.

Como el valor introducido esta entre corchetes y separados por comas, hay que seleccionar únicamente los valores numéricos, eliminando por tanto las comas, por ello se utilizan las siguientes funciones mostradas en el fragmento del código

```
% Value changed function: CoordenadasEditField
function CoordenadasEditFieldValueChanged(app, event)
    value = app.CoordenadasEditField.Value;
    value = value(2:end - 1);
    vec = str2double(strsplit(value));
    assignin('base','CoordenadasCorte',vec);
end
```

Para entender su funcionamiento se realizará un ejemplo:

Vector Introducido:



Cuando se tiene el valor deseado, se guarda en la variable correspondiente.

CallBack: Ejecutar Simulación

Uno de los botones más importantes de la interfaz, es el que ejecuta la simulación, ya que será el encargado de realizar la llamada a los 'scripts' que se definieron en anteriores capítulos, para poder ejecutar las acciones que hayan sido elegidas.

En este caso al pulsar el botón, se ejecuta el archivo OPC, en el que se desarrollaba toda la comunicación entre MATLAB y RobotStudio a la par que se ejecutaban en ambas plataformas la simulación.

Tras realizarse esta secuencia si la acción era la detección de pieza se llama a otra función (Detección) para mostrar en los campos habilitados para ello los datos reunidos del algoritmo de visión artificial.

Posteriormente, se invoca el segundo archivo OPC2, para finalizar la realización de la simulación, en la que se selecciona o se descarta la pieza.

```
% Button pushed function: EjecutarSimulacionButton
function EjecutarSimulacionButtonPushed(app, event)
    app.RedCross.Visible='Off';
    app.RedCross_2.Visible='Off';
    app.GreenCheck.Visible='Off';
    app.GreenCheck_2.Visible='Off';

    OPC;
    if app.DeteccionFiguraButton.Value==true
        Deteccion(app);
        OPC2;
    end
end
```

CallBack: Mostrar Parámetros Obtenidos por visión artificial

Como se comentaba en la anterior función, para mostrar los parámetros obtenidos por visión artificial, se realiza una llamada a la función 'Deteccion'.

En esta, se evalúan los parámetros del espacio de trabajo ('evalin'), y posteriormente se rellenan los campos con estos valores obtenidos.

Si el tamaño o el color son correctos o no, se asigna un verificado verde o una cruz roja respectivamente. Para ello se hace visible o no la imagen correspondiente en función de la validez de estos.

```
function Deteccion(app, event)
    ColorCorrecto= evalin('base','ColorCorrecto');
    ColorDetectado= evalin('base','ColorDetectado');

    SizeCorrecto= evalin('base','SizeCorrecto');
    SizeDetectado= evalin('base','SizeDetectado');

    PosicionDetectada= evalin('base','PosicionDetectada');

    app.ColorDetectadoEditField.Value=ColorDetectado;
    app.SizeDetectadoEditField.Value=SizeDetectado;
    app.PosicionDetectadaEditField.Value=PosicionDetectada;

    if ColorCorrecto==true
        app.GreenCheck.Visible='On';
    else
        app.RedCross.Visible='On';
    end

    if SizeCorrecto==true
        app.GreenCheck_2.Visible='On';
    else
        app.RedCross_2.Visible='On';
    end
end
```

CallBack: Abrir Nueva Ventana (Imágenes Visión Artificial)

Finalmente, para abrir una nueva ventana en la que se encuentran las imágenes procesas y obtenidas con el algoritmo, hay que hacer una llamada al archivo de AppDesigner en el que se ha definido esta nueva ventana.

```
% Button pushed function: ImagenesVisinArtificialButton
function ImagenesVisinArtificialButtonPushed(app, event)
    InterfazTFG_Imagenes;
end
```

Con esto finalizaría la explicación del código del menú principal y a continuación se mostrará cómo se han programado las restantes ventanas de las imágenes.

Ventana Secundaria (Imágenes Visión Artificial)

En todas las ventanas donde se muestran las imágenes, se tiene la posibilidad de cambiar el formato para mostrar todas las imágenes a la vez, o ir mostrando una a una.

Para pasar de una imagen a otra en este segundo formato, se hace uso de unas flechas.

Las ventanas como se mostró en la *Figura 115* son las siguientes:

1. **Detección Figura**

   Como en la anterior interfaz, el primer punto a definir es la función de inicio o 'startupFcn'. En este caso el procedimiento es bastante sencillo, ya que únicamente se asigna el valor cero al contador (encargado de la transición en el segundo formato con flechas), se predetermina una variable con el valor '1' para mostrar el segundo formato y se invoca la función en la que se muestran todas las imágenes.

```
% Code that executes after component creation
function startupFcn(app)
    assignin('base','Contador',0);
    app.VistaDoble.Value=1;
    MostrarImagenes(app);
end
```

Como se puede apreciar, en este caso, la función más importante de esta interfaz, es la denominada 'MostrarImagenes', en la que se realizan las siguientes tareas:

- Evaluar todas las imágenes del espacio de trabajo de MATLAB para poder introducirlas en la interfaz.

```
Imagen = evalin('base','Imagen');
ImagenGrises = evalin('base','ImagenGrises');
```

- Evaluar que botón indicador del formato esta pulsado y realizar una acción u otra en función de esto.

```
switch app.VistaDoble.Value
    case 1
```

- Si se eligen mostrar las 6 imágenes a la vez, el primer paso el vaciar el componente por si tuviera imágenes anteriores, quitar las flechas, ya que en este caso no son necesarias, borrar el título anterior que pudieran tener, y por último hacer el componente visible y asignar la imagen en este componente para mostrarla. Esta secuencia se realizaría con los siguientes comandos respectivamente

```
cla(app.UIAxes);
      ⋮
app.FlechaRetroceso.Visible='Off';
app.FlechaAvance.Visible='Off';
app.UIAxes.Title.String='';
      ⋮
app.UIAxes2.Visible='On';
      ⋮
imshow(Imagen,'Parent',app.UIAxes2);
```

- Si por el contrario se eligen mostrar las imágenes de dos en dos, hay que vaciar las posibles imágenes anteriores, los títulos, etc., y analizar el valor del contador (que indica al programa en qué posición de la interfaz se encuentra, al pulsar las flechas).

  Una vez se conoce el caso en el que se encuentra, se muestran las imágenes correspondientes, y se hacen visibles únicamente las flechas que correspondas. Por ejemplo, si está en la primera opción el usuario no podrá ir hacia atrás, por lo que solo se muestra la flecha de avance, y si está en el último caso, será lo mismo, pero al contrario con la flecha de retroceso. Finalmente se escriben los títulos de las imágenes que se están mostrando en ese momento.

  Como hay 6 imágenes y se muestran de 2 en 2, existen 3 casos, pero los 3 siguen la misma dinámica de funcionamiento.

En los siguientes fragmentos de código se muestran las instrucciones que hacen posible, la realización de la explicación anterior.

```
app.UIAxes2.Visible='Off';
        ⋮
cla(app.UIAxes2);
        ⋮
app.UIAxes2.Title.Visible='Off';
        ⋮
switch Contador
    case 0
        ⋮
    imshow(ImagenGrises,'Parent',app.UIAxes_2);
    app.FlechaRetroceso.Visible='Off';
    app.FlechaAvance.Visible='On';
    app.UIAxes.Visible='On';
        ⋮
    app.UIAxes.Title.String='Imagen Original';
    app.UIAxes_2.Title.String='Imagen Escala Grises';
```

Con esto finalizaría la explicación de la función 'MostrarImagenes'

Las siguientes funciones son más breves, ya que los componentes restantes no son tan complejos.

Los 'callbacks' de las flechas son los encargados de cada vez que se pulsa una de estas, actualizan la variable contadora a un número superior o inferior como se muestra en el siguiente fragmento de código. Tras esta actualización se vuelve a invocar la función de mostrar imágenes para actualizar la interfaz.

```
% Image clicked function: FlechaAvance
function FlechaAvanceClicked(app, event)
    Contador = evalin('base','Contador');
    assignin('base','Contador',Contador+1);
    MostrarImagenes(app);
end

% Image clicked function: FlechaRetroceso
function FlechaRetrocesoClicked(app, event)
    Contador = evalin('base','Contador');
    assignin('base','Contador',Contador-1);
    MostrarImagenes(app);
end
```

Otras funciones 'callbacks' que existen son las de los botones para cambiar el formato de la interfaz. Estas funciones se encargan de evitar que ambos pulsadores se puedan activar a la vez, es decir si se pulsa uno el otro se desactiva y viceversa. Tras su pulsación también se invoca la función 'MostrarImagenes' para actualizar la interfaz.

```
% Value changed function: VistaDoble
function VistaDobleValueChanged(app, event)
    value = app.VistaDoble.Value;
    switch value
        case 0
            app.Vista6.Value=1;
        case 1
            app.Vista6.Value=0;
    end
    MostrarImagenes(app);
end
```

Para navegar entre los 3 tipos de ventana que existen, se disponen una serie de botones en los que hay que definir sus funciones 'callback'. Para estos componentes la única función que desempeñan es borrar la ventana actual y abrir la nueva ventana solicitada con la nueva interfaz. En el siguiente fragmento de código se muestra como se ha realizado.

```
% Value changed function: MscarasRGBButton
function MscarasRGBButtonValueChanged(app, event)
    delete(app.UIFigure);
    InterfazTFG_Imagenes1;
end

% Value changed function: AislarColorDetectadoButton
function AislarColorDetectadoButtonValueChanged(app, event)
    delete(app.UIFigure);
    InterfazTFG_Imagenes2;
end
```

2. Máscaras RGB

El procedimiento de programación para esta ventana es muy similar a la anterior, por lo que no merece la pena volver a repetir lo mismo. Únicamente se centrará la atención a comentar que en este caso se muestran 3 imágenes en el primer formato y una sola en el segundo, por lo que existen un total de 3 casos para ese segundo formato.

La otra diferencia como es lógico, son las imágenes utilizadas, ya que en este caso se evalúan las imágenes en las que se muestran las máscaras.

```
% Value changed function: MscarasRGBButton
function MostrarImagenes(app, event)
    ImagenR = evalin('base','imagenR');
    ImagenG = evalin('base','imagenG');
    ImagenB = evalin('base','imagenB');
```

3. Aislar el color detectado RGB

En esta última ventana, se muestran 4 imágenes en el primer formato y una única en el segundo, por lo que en este segundo existen 4 posibles casos.

Las imágenes evaluadas son las siguientes:

```
% Value changed function: AislarColorDetectadoButton
function AislarColorDetectadoButtonValueChanged(app, event)
    ZonaInteres = evalin('base','ZonaInteres');
    AislaRojo = evalin('base','AislaRojo');
    AislaVerde = evalin('base','AislaVerde');
    AislaAzul = evalin('base','AislaAzul');
```

CAPÍTULO VI. RESULTADOS

Tras la explicación de lo que se quería desarrollar en el presente proyecto y la demostración de cómo se ha llevado a cabo, el lector va a leer uno de los capítulos más importantes del informe, ya que, en este, se muestra finalmente todo el resultado del trabajo realizado, y la ejecución de los programas.

En los siguientes capítulos se mostrarán algunas de las capturas más representativas obtenidas durante la ejecución del programa. Pero cabe destacar, que al ser una ejecución que se realiza de forma conjunta y sincronizada en 2 softwares diferentes, aunque en este informe se expliquen uno seguido del otro, en realidad funcionan de forma paralela.

CAPÍTULO 6.1. RESULTADOS EN MATLAB

CAPÍTULO 6.1.1. INTERFAZ DE USUARIO

El programa comienza con la manipulación de un usuario en la interfaz desarrollada, por lo que lo primero que se va a mostrar es el diseño final (previamente solo se ha explicado su programación).

El menú principal desde el que se controlan todos los parámetros de la simulación, tiene el siguiente aspecto:

Figura 117. *Menú Principal de la interfaz de usuario.*

En este menú se encuentra un interruptor, para habilitar la interfaz, y una serie de pulsadores o submenús para ir seleccionando las diferentes posibilidades que ofrece. Como ya se explicó en la programación, la opción seleccionada de la acción se denota con un circulo verde en las imágenes circulares.

Tras la selección de todos los parámetros que el usuario considere oportunos, se debe pulsar 'Ejecutar Simulación' y se realizarían las simulaciones correspondientes en los dos softwares utilizados.

Al ejecutar la simulación a modo de seguridad para evitar errores de mal uso por parte del usuario, se inhabilitan todos los pulsadores hasta que termine la simulación, de manera que no sea posible ejecutar otra simulación mientras se está realizando una previamente seleccionada.

Si el usuario selecciona el montaje del automóvil, la interfaz no volverá a tener relevancia hasta que acabe dicha acción. Pero si por otro lado se ha seleccionado la detección por visión artificial, en la interfaz se podrán visualizar ciertos aspectos de cómo ha ido la simulación.

En primer lugar, en los campos de los parámetros del cubo, se podrá visualizar cuales han sido estos parámetros detectados. A continuación, se muestran diferentes ejemplos de selección de cubos y los valores detectados por la visión artificial (véase Figura 118).

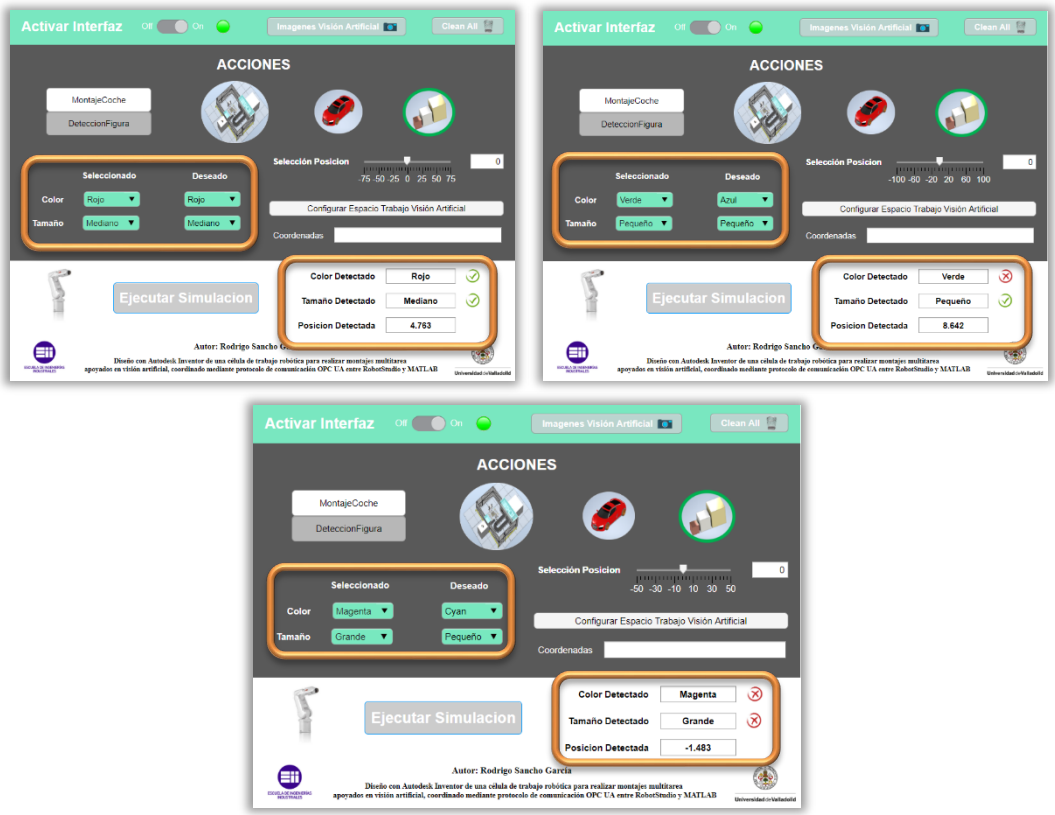


Figura 118. *Diferentes resultados en interfaz por detección de los parámetros del cubo.*

En segundo lugar, se habrán obtenido una serie de imágenes provenientes del algoritmo de visión artificial desarrollado. Estas imágenes se pueden visualizar pulsando el botón de la parte superior 'Imágenes Visión Artificial'. Al pulsarlo se abrirá esta nueva ventana:

Nota:

La figura que se seleccionó para demostrar las diferentes capturas que se muestran a continuación obtenidas por visión artificial, es un cubo de tamaño 'Mediano' y de color 'Rojo'.

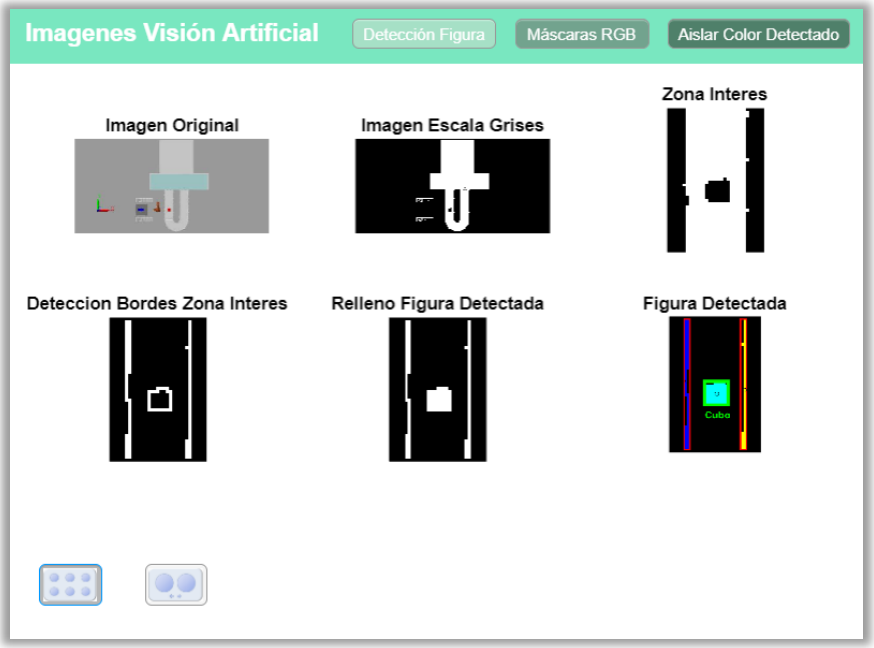


Figura 119. *Menú Secundario (Imágenes creadas por el algoritmo de visión artificial)*

En esta ventana se pueden visualizar todas las imágenes obtenidas y procesadas, pero también dispone de su versión en la que se muestran únicamente 2 imágenes por pantalla, y se puede navegar entre las 6 imágenes.

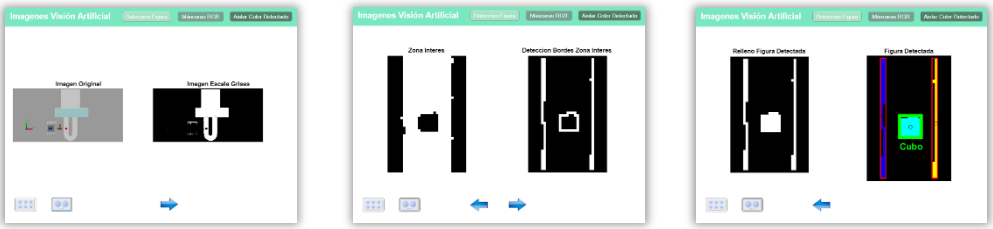


Figura 120. *Menú Secundario (Vista de 2 imágenes/página)*

Si, por otra parte, se desean visualizar el resto de las imágenes, hay que pulsar los botones de la parte superior, y el programa redireccionara al usuario hacia otra ventana, pero con la misma filosofía de funcionamiento que esta anterior.

Mascaras RGB

Figura 121. *Menú Secundario (Imágenes de las máscaras RGB)*

Como el color del cubo era el rojo, en este caso aparece un cuadrado blanco, representando la máscara que se aplica. En los demás colores el fondo será completamente negro, como se muestra en las anteriores imágenes.

Aislar Color Detectado

Para demostrar que funciona con otros colores, en este caso se ha sacado las capturas con un modelo de cubo mediano y de color verde.

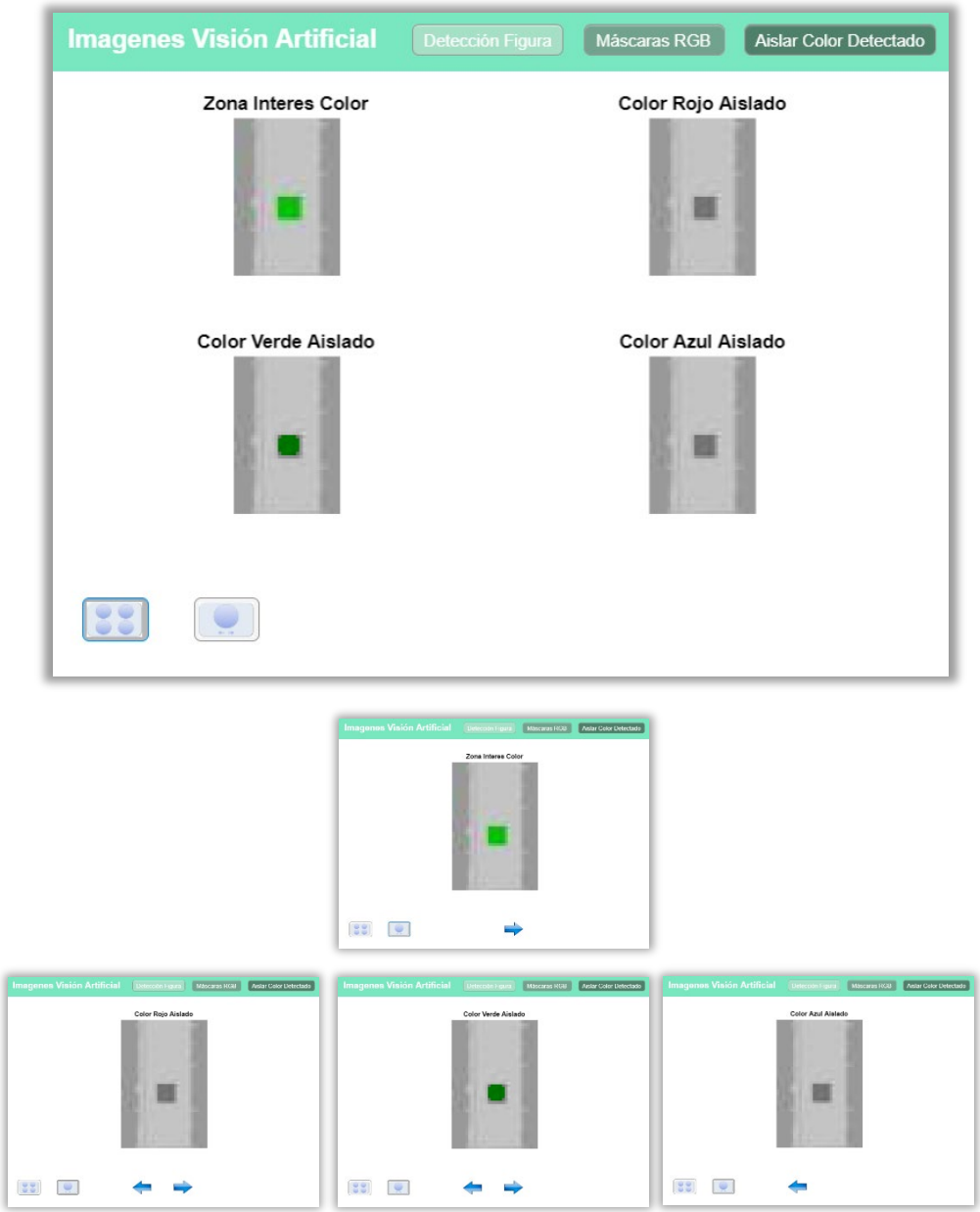


Figura 122. *Menú Secundario (Imágenes de los colores RGB aislados)*

Al ser una imagen procesada, en la que se ha aislado únicamente el color verde, como se puede apreciar no es tan nítido como en la imagen a color, pero si el lector visualiza con detalle la imagen del color verde aislado, podrá apreciar cómo no tiene el mismo color que las restantes imágenes de color rojo y azul.

CAPÍTULO 6.1.2. SIMULACIÓN EN MATLAB

El uso de Matlab para este proyecto, viene predominado por la necesidad de implementar la visión artificial. Por tanto, en el montaje del coche este programa carecía de sentido, lo que origina que no se realiza nada en esta acción.

Sin embargo, adquiere gran relevancia en la segunda acción de la detección de los parámetros del cubo. Ya que es donde se realiza el movimiento del cubo y se toma la captura de pantalla para su posterior procesamiento con el algoritmo desarrollado.

Como la mejor manera de mostrarlo es con un video, a continuación, se muestran algunas de las capturas obtenidas en este programa durante su ejecución.

Se van a distinguir los dos casos posibles, que la pieza detectada sea correcta, o que la pieza detectada no sea la deseada y por tanto incorrecta.

Cabe destacar que ambos casos se han realizado con los parámetros predeterminados (posición 0, tamaño mediano y color rojo).

Secuencia Pieza Correcta

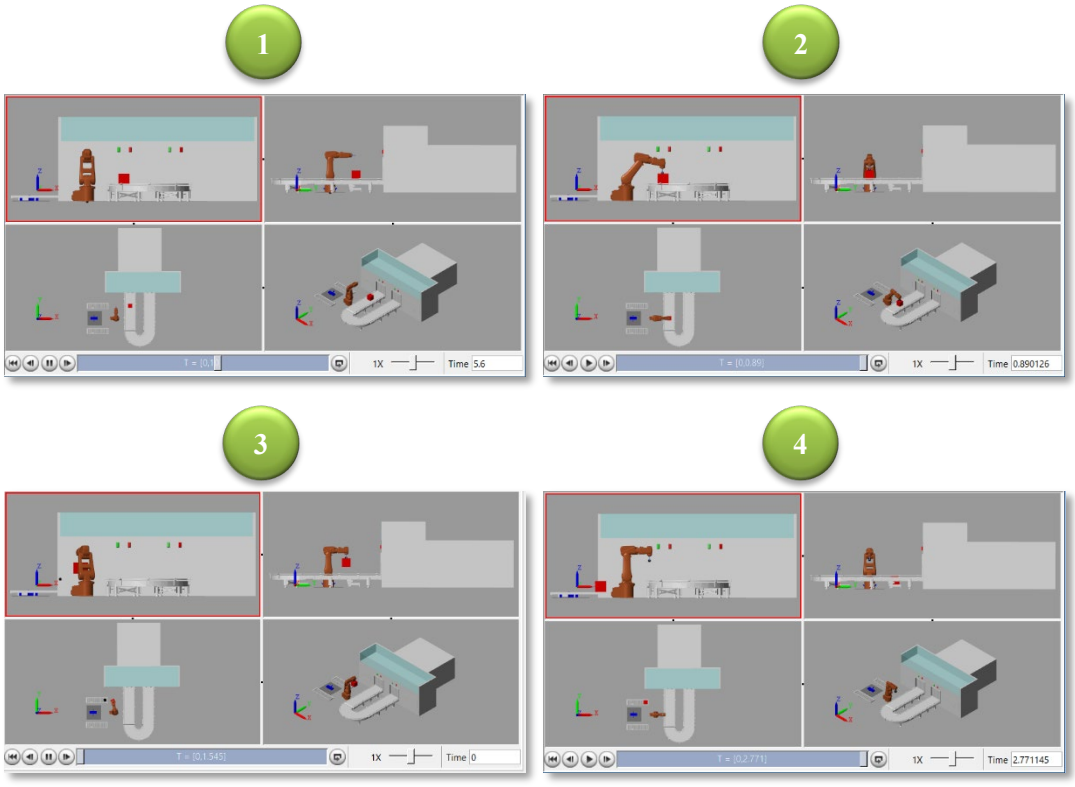


Figura 123. *Parte de la secuencia en MATLAB de tratamiento de pieza correcta.*

Secuencia Pieza Incorrecta

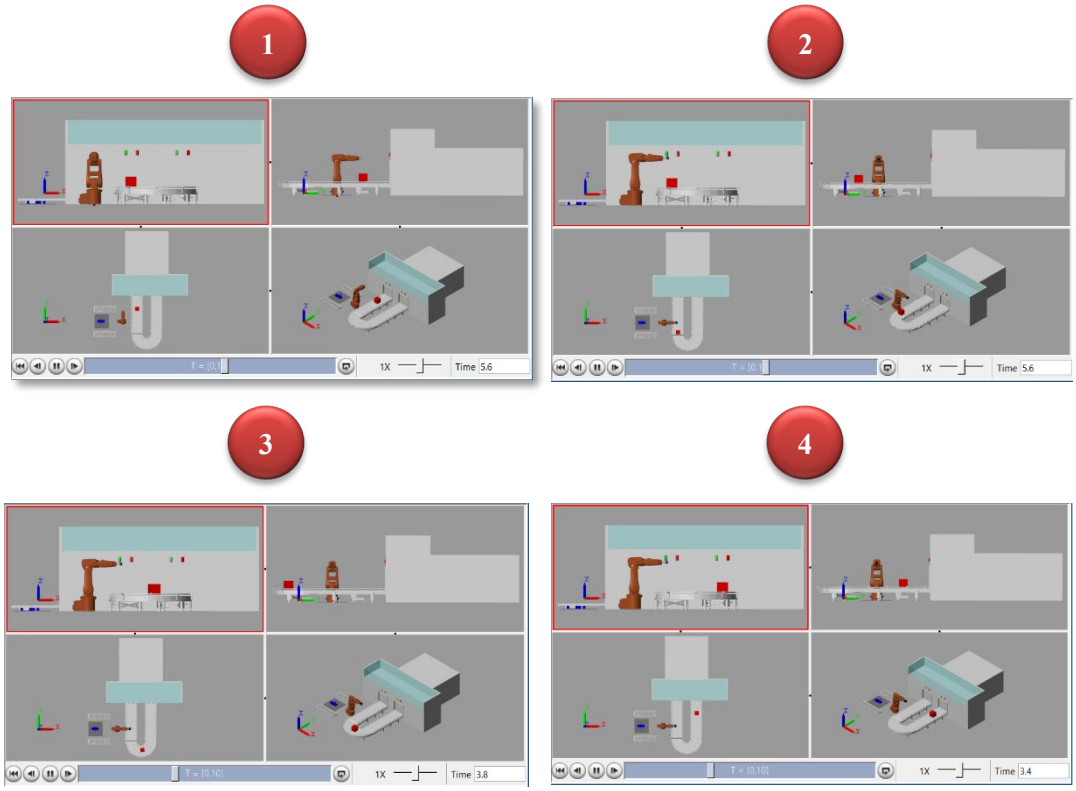


Figura 124. *Parte de la secuencia en MATLAB de tratamiento de pieza incorrecta.*

Esta secuencia puede variar en función del color, tamaño o posición elegido por el usuario. A continuación, se muestra únicamente la vista isométrica para diferentes valores de estos parámetros.

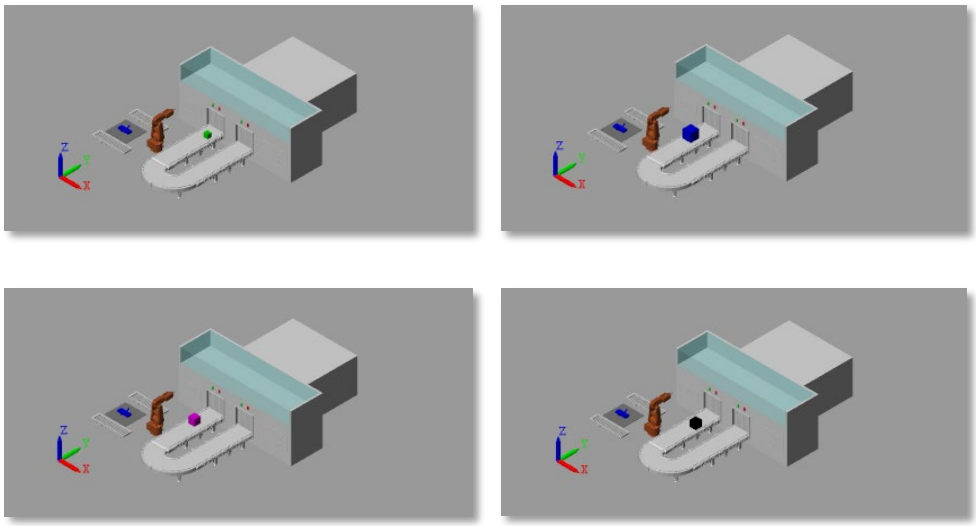


Figura 125. *Diferentes simulaciones del cubo con color, posición y tamaño modificados.*

CAPÍTULO 6.2. RESULTADOS EN ROBOTSTUDIO

Finalmente, se mostrarán los resultados de las simulaciones en el software RobotStudio. En este caso, sí que se ejecutan las dos tareas programadas, tanto la del montaje del automóvil, como la detección de pieza.

En la detección de pieza se adquieren los datos obtenidos de la simulación en Matlab, y en función de esos valores ejecuta una acción u otra.

El diseño final de la célula de trabajo ya se mostró en el *capítulo 4.2.11. Diseño Final*, por lo que ahora se centrara la atención en la realización de la simulación.

CAPÍTULO 6.2.1. MONTAJE AUTOMÓVIL

Al activar esta tarea, los componentes del automóvil comenzarán a salir desde el conveyor doble, y serán recogidos por el robot IRB120. Este se encargará de ir asignándolos al lado derecho o izquierdo del robot IRB14000. Donde este segundo robot realizara el montaje de todas las piezas y con la sincronización de ambos robots se finalizará el montaje y se enviara por el conveyor simple a otra sección de la fábrica.

Como es difícil mostrar este resultado sin un video, se han tomado a continuación al igual que en casos anteriores, una serie de capturas del funcionamiento.

Como se muestra en la siguiente ilustración, las piezas comienzan a salir en el sentido que indica la flecha naranja. Una vez alcancen la altura del robot se detienen para ser seleccionadas por este. Además, se puede mostrar cómo cambian algunos elementos en función del momento de la simulación. Las luces del conveyor, indicarán si este está en movimiento o está detenido (circulo amarillo), y las cámaras de vigilancia irán rotando constantemente (circulo naranja) abarcando todo el perímetro a vigilar.

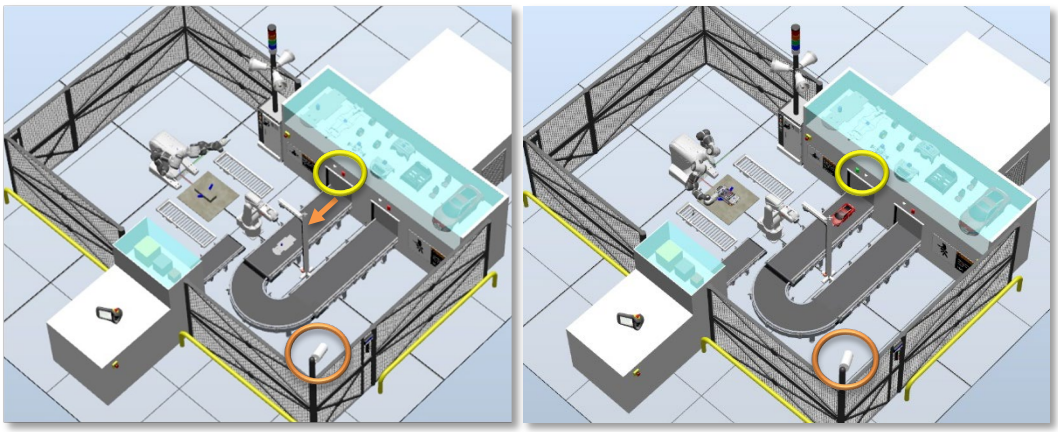


Figura 126. *Detalles de la simulación durante el montaje del automóvil en RobotStudio.*

En las siguientes ilustraciones se muestran algunos de los movimientos que tienen que realizar estos robots para completar el ensamblaje del automóvil.

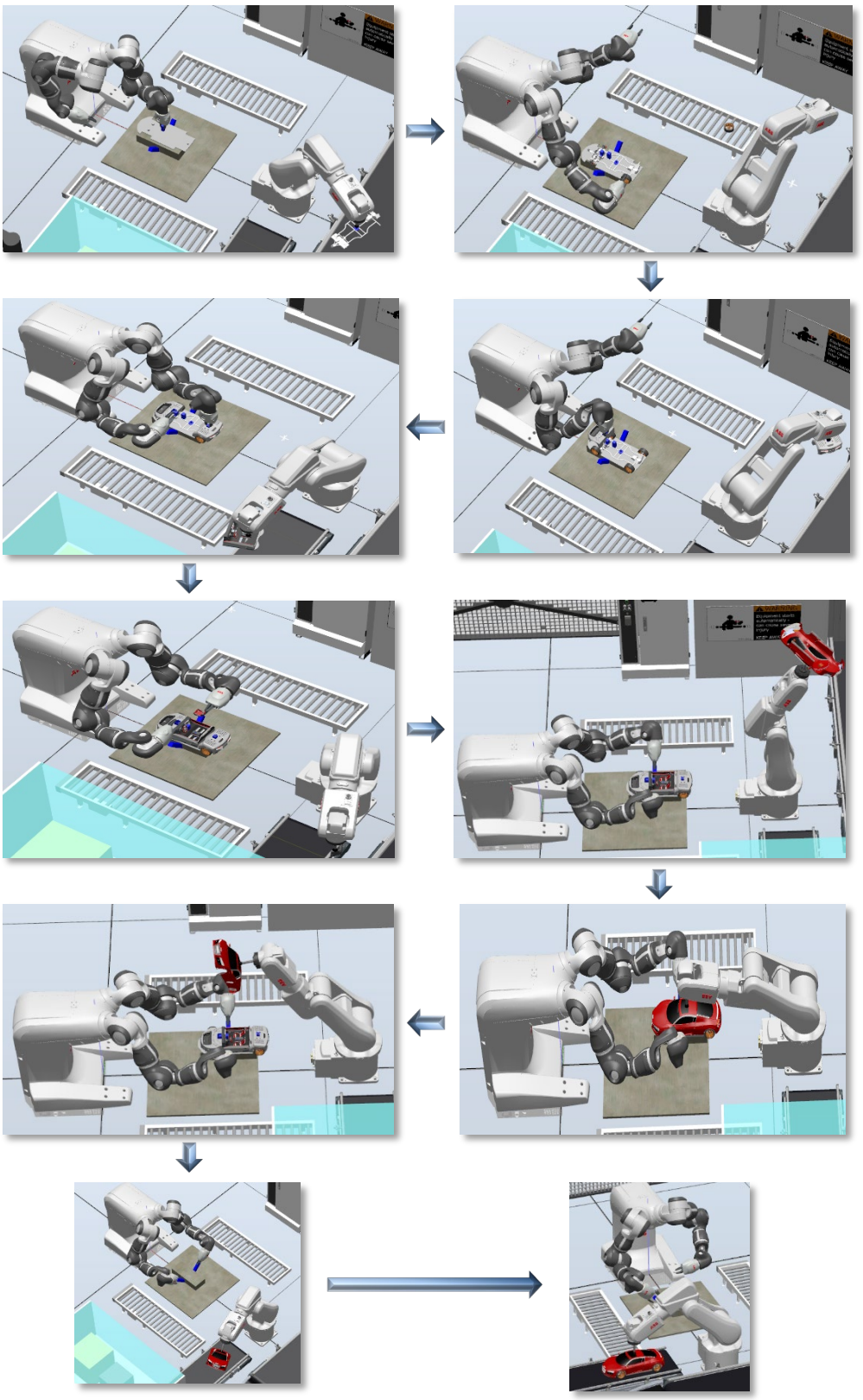


Figura 127. *Secuencias del montaje del automóvil en RobotStudio.*

Una vez montado, el automóvil se desplaza por el conveyor simple hasta alcanzar la siguiente etapa. Cada vez que se acciona una de las cintas transportadoras, sus luces cambian para indicar dicho movimiento.

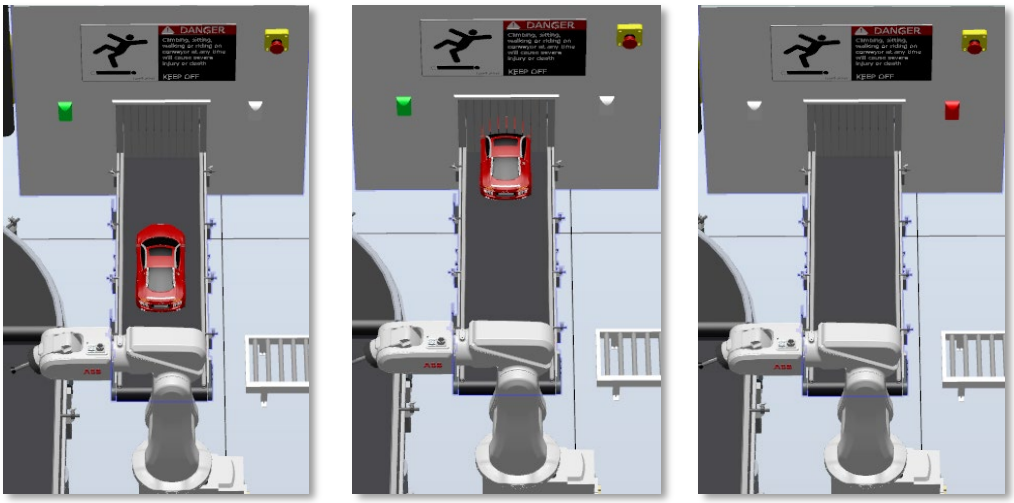


Figura 128. *Movimiento del automóvil montado por el conveyor de salida.*

A continuación, se mostrará el rango de movimiento que tienen que realizar las pinzas para poder manipular las diferentes piezas.

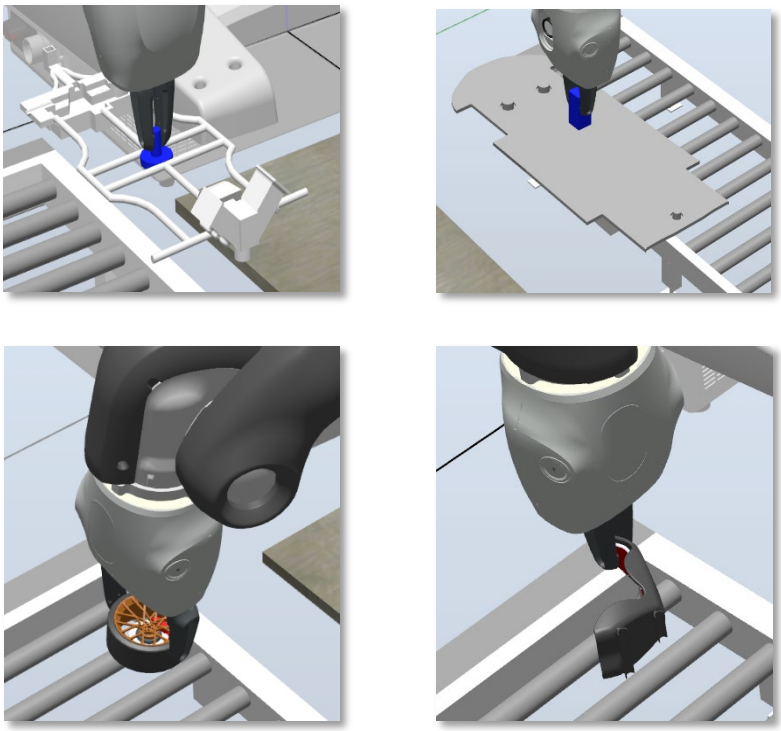


Figura 129. *Secuencias con algunos de los múltiples agarres que realizan los brazos robóticos.*

Finalmente, antes de proceder a mostrar la siguiente tarea, se va ha realizado un esquema de las fases del montaje del coche. Este ya se vio en la *Figura 72*, con el software de Autodesk Inventor, pero ahora se va a realizar con las imágenes tomadas durante la simulación, es decir este sería el montaje real que realizan los robots.

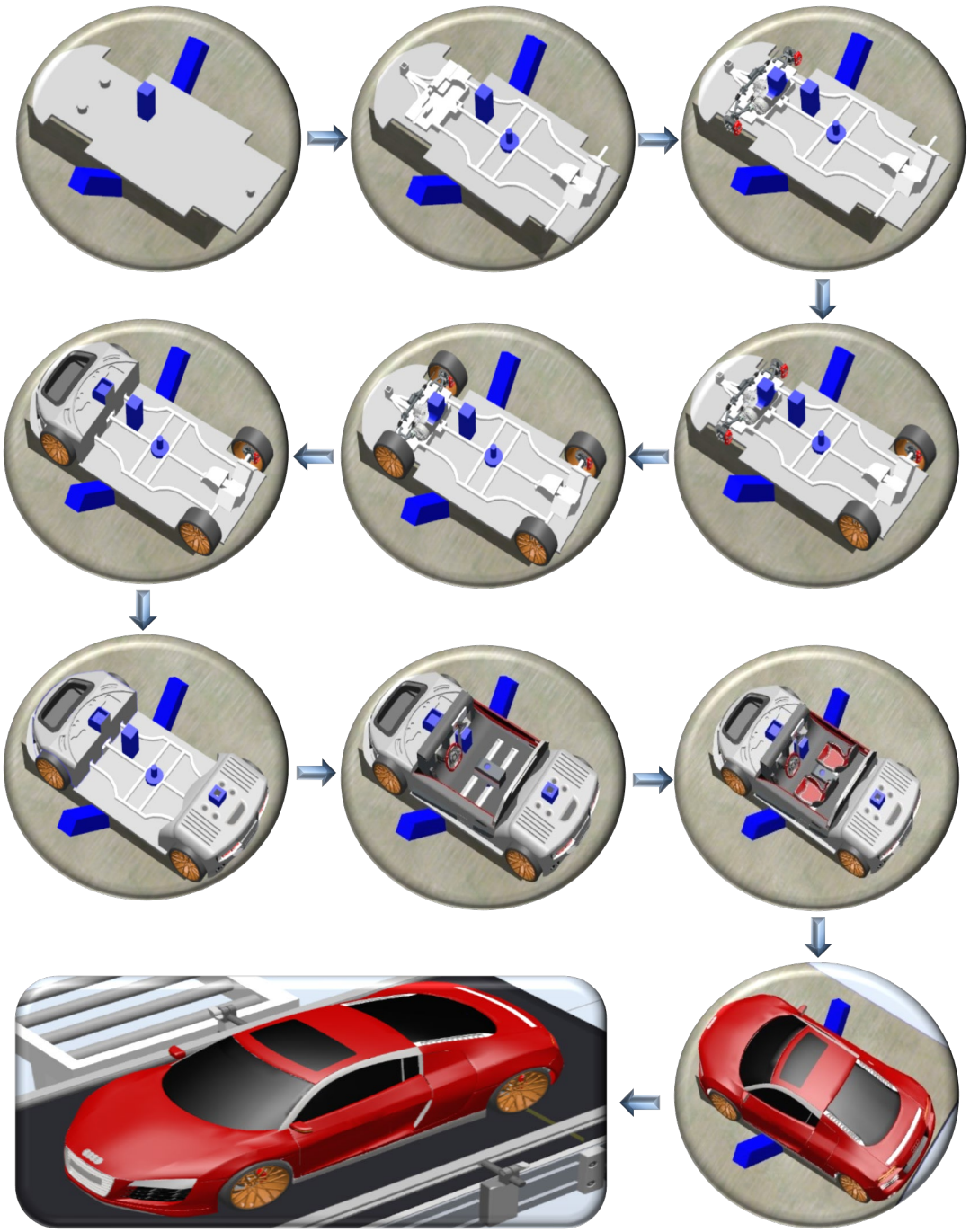


Figura 130. *Secuencia del montaje realizado por los robots del automóvil.*

CAPÍTULO 6.2.2. DETECCIÓN DE PIEZA

Al igual que se realizaba en Matlab, en RobotStudio también se muestra el movimiento del cubo. En este caso la situación es más realista, porque para la toma de decisiones se basa en los datos obtenidos de la visión artificial, es decir al igual que antes realizara un movimiento u otro en función de si es o no correcta la pieza, pero, además, la posición en la que coge el cubo depende de la posición detectada por visión artificial, con lo cual si se realiza una mala detección o el algoritmo no está implementado correctamente, la simulación fallaría.

En las siguientes ilustraciones se muestra una secuencia en la que los parámetros del cubo detectados eran los deseados y, por tanto, el robot IRB120 se dispone a dejar esta pieza en uno de los conveyors de YuMi. La posición inicial sería '0' y la detectada '4.7', a pesar de este ligero error producido por el algoritmo que no es capaz de detectar exactamente la posición, se puede ver cómo es un error pequeño y adquiere la pieza en su posición casi central.

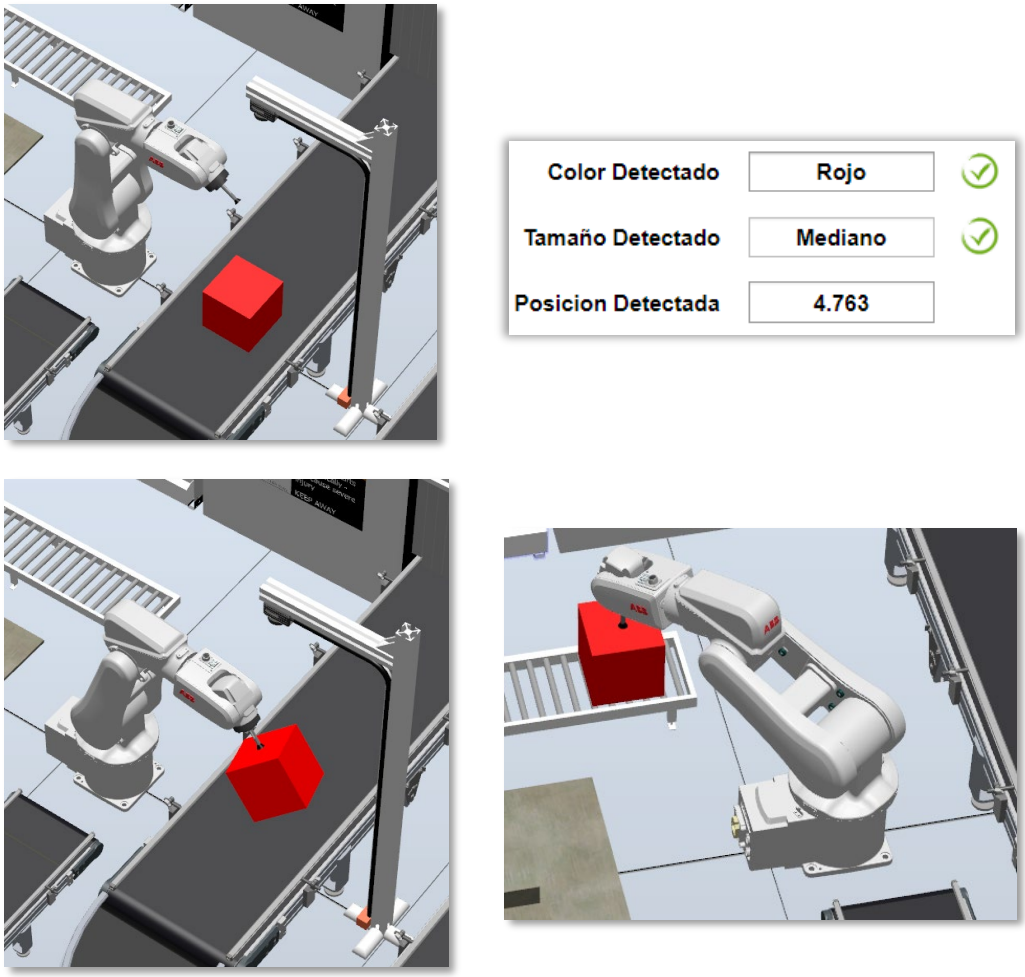


Figura 131. *Secuencia de manipulación de una pieza correcta.*

Por otro lado, se tiene el caso de que la pieza fuera incorrecta, en el cual la simulación hará regresar la pieza sin ser manipulada por los robots.

En este segundo ejemplo se han asignado como parámetros deseados los del anterior ejemplo, pero esta vez se introducen otros, originando que no coincidan los parámetros introducidos con los deseados, y por tanto obteniendo el siguiente resultado:

Figura 132. *Secuencia de manipulación de una pieza incorrecta.*

Debido a la infinidad de posibilidades que tiene el programa cambiando los diferentes parámetros, no se pueden enseñar todas, pero con las ilustraciones que se han mostrado a lo largo de este capítulo, se ha pretendido mostrar al lector el funcionamiento sobre el que se basarían todas estas posibilidades.

Con esto se da por finalizado el capítulo de resultados y se procede en el siguiente capítulo a analizar un breve estudio económico, teniendo en cuenta los costes que han derivado de la realización del presente proyecto.

CAPÍTULO VII. ESTUDIO ECONÓMICO

En este capítulo se muestra una estimación del coste total del proyecto. Para ello se han realizado una clasificación general en función de si se trata de un coste directo o indirecto. Una vez se han separado entre estas dos vertientes, se podrán encontrar otras clasificaciones como es la procedencia de estos costes.

CAPÍTULO 7.1. COSTES DIRECTOS

Se definen los costes directos como el gasto que guarda una relación estrecha con el producto, el servicio, o en este caso el proyecto. Es decir, son aquellos que se asociarían de una forma directa a la elaboración o y terminación de estos productos o servicios.

Costes asociados al equipo y software utilizado

Los costes de todo el material utilizado, los equipos requeridos, y el software con sus licencias de uso.

Costes Software			
Concepto		Coste real (€/Año)	Coste alumno UVA (€/Año)
Licencia RobotStudio		950,00	0,00
Licencias Matlab	MATLAB	800,00	0,00
	Image Processing Toolbox	400,00	0,00
	OPC Toolbox	460,00	0,00
Licencia Autodesk Inventor		2.886,00	0,00
Total		5.496,00 €	0,00 €

Tabla 12. *Costes debido a uso de software en el proyecto.*

Costes Equipo			
Concepto	Coste unitario (€)	Unidades	Coste Total
Ordenador Portátil (Hacer Aspire 7 AMD Ryzen 5 3550H, 8GB RAM, 512GB SSD, NVIDIA GTX1650-4GB)	899,00	1	899,00
		Total	899,00 €

Tabla 13. *Costes debido a uso de equipo físico (hardware) en el proyecto.*

Coste asociado a la mano de obra

Se realiza una estimación de las horas dedicadas a cada parte del proyecto y se tiene en cuenta que el salario promedio de un ingeniero en la especialidad de electrónica industrial y automática es de aproximadamente 16€/hora.

Aunque este valor puede variar en función de muchos aspectos, como el sitio, la oferta de trabajo, la experiencia…

Costes Mano de Obra			
Concepto	Coste por hora (€/Hora)	Unidades (Horas)	Coste Total (€)
Formación Proyecto	10	100	1000
Diseño y Propuesta del Proyecto	10	25	250
Diseño Piezas 3D	16	350	5600
Programación Matlab	16	225	3600
Programación RobotStudio	16	275	4400
	Total	975 horas	14850 €

Tabla 14. *Costes debido a la mano de obra en el proyecto.*

Costes directos totales

Reuniendo el total de todos los costes directos se tiene lo siguiente:

Costes Directos Totales	
Concepto	Coste real (€/Año)
Costes asociados al software	5.496,00
Costes asociados al equipo	899,00
Costes de la mano de obra	14.850,00
Total	21.245,00 €

Tabla 15. *Costes directos totales del proyecto.*

CAPÍTULO 7.2. COSTES INDIRECTOS

Se entiende como coste indirecto aquellos costes que no pueden ser atribuibles directamente al proyecto, pero que sí que pueden ser identificados por su sistema de contabilidad. Los más habituales suelen ser el alquiler, la luz, agua y otros suministros básicos, el internet y la limpieza y otros servicios esenciales.

No se va a realizar un estudio complejo en esta materia, por lo que la tabla de costes simplificados sería la siguiente:

Costes Indirectos Totales			
Concepto	Coste por unidad	Unidades	Coste Total (€)
Consumos Eléctricos (210kWh aprox.)	0.14 €/kWh	975 horas	29,4
Internet	26 €/mes	5 meses	130
		Total	159,4 €

Tabla 16. *Costes indirectos totales del proyecto.*

CAPÍTULO 7.3. COSTES TOTALES

Sumando los costes directos e indirectos que se han estimado en los anteriores capítulos, se tienen la siguiente tabla:

Costes Totales Proyecto	
Concepto	Coste real (€/Año)
Costes Directos Totales	21.245,00
Costes Indirectos Totales	159,40
Total	21.404,4 €

Tabla 17. *Costes indirectos totales del proyecto.*

CAPÍTULO VIII. CONCLUSIONES

En la realización de este proyecto se han desarrollado conocimientos en diferentes campos de la ingeniería, cumpliendo de esta manera uno de los principales objetivos requeridos en este trabajo.

Además, se tuvo que realizar un aprendizaje de nuevas herramientas que el autor no había visto durante los 4 años de duración en el grado. La visión artificial y el modelado de estaciones robóticas en MATLAB es un campo que se ha tenido que aprender de cero durante el desarrollo de este trabajo, lo que ha servido para aumentar más el rango de conocimientos adquiridos en esta etapa.

Realizando una revisión del trabajo realizado, se podría asumir que se han cumplido los objetivos propuestos de manera satisfactoria, ya que ha resultado un programa final muy dinámico e interactivo en el que se simula una serie de tareas en una célula robótica.

Esto origina nuevos campos de aplicación, ya que no solo servirá para que trabajen sobre esta célula otros estudiantes y puedan desarrollar sus habilidades de programación con los robots, sino que también, se puede expandir su uso con niños, para que jueguen con la aplicación desarrollada y puedan ver las diferentes simulaciones. De esta manera se les puede crear un interés en el fascinante mundo de la robótica desde temprana edad, sobre todo aquellos interesados por los temas tecnológicos.

Con todo esto se puede concluir que, a pesar de la complejidad de programación inicial incluida en este proyecto, ya que hay que comunicar varios programas de forma simultánea y la cantidad de horas de diseño que se han necesitado para el desarrollo de todos los componentes, este trabajo puede servir de punto de partida para infinidad de aplicaciones en un entorno más educativo, por lo que se invita al lector a analizar el siguiente capítulo (*IX. Trabajos Futuros*) por si estuviese interesado en ver la siguiente fase de desarrollo propuesta.

En contraposición al anterior párrafo, el punto menos positivo es que no se puede aplicar en la industria real, ya que ha sido un proyecto con otros fines, a pesar de ser un campo de investigación muy interesante debido a la revolución industrial actual de la Industria 4.0.

CAPÍTULO IX. TRABAJOS FUTUROS

Si se desea seguir en la línea de trabajo propuesta por el autor del presente proyecto, se proponen a continuación una serie de mejoras y posibles implementaciones con el objetivo de desarrollar aun con mayor profundidad esta idea.

1. Mejora del algoritmo de visión artificial

En primer lugar, se propone la mejora de este algoritmo, para poder realizar lo siguiente:

- Detectar mayor número de colores y mejorar la detección de las formas, para no depender de que el fondo y la cinta transportadora en la imagen sean blancos.
- Implementar la capacidad de analizar diversas formas, para realizar una detección de todas las piezas del montaje, y poder analizar posibles defectos en estas y descartarlas.
  Recuerde que se ha tomado como hipótesis en este proyecto, que las piezas del montaje eran todas correctas, pero en una situación real podría no ser así.

2. Añadir nuevos montajes con diferentes piezas

Para este proyecto, se seleccionó y diseño un automóvil de juguete, debido a que se consideró un elemento muy visual, y bonito de realizar su montaje. Además, otra de las características por la que se eligió este componente es por la complejidad que posee para realizar el montaje, de forma que se pudiese mostrar el gran rango de posibilidades que se tienen.
Pero si se desease realizar en un ámbito más real, tendrían que definirse las piezas industriales o los componentes existentes en la realidad.

3. Implementación del modelo en un sistema real

Como se ha ido explicando a lo largo del informe, este se trata de un proyecto simulado y ficticio, lo que implica que no existe en ninguna industria real.
A pesar de esto, cabe destacar, que siempre se ha buscado representar la mayor realidad en todo el diseño y en todos los programas utilizados, para que la implementación en una célula de trabajo real fuese lo más sencilla posible, teniendo todo este trabajo previo desarrollado.

Los programas servirían para los robots reales, los diseños están realizados con programas profesionales de diseño 3D, y la célula de trabajo se ha diseñado buscando cumplir toda la normativa vigente sobre espacios de trabajos robóticos.
Por lo que se propone como continuación de este proyecto, la implementación de este en una situación real.

REFERENCIAS

[1] Revista de Robots. (2019, julio). *Campos de conocimiento abarcados por la robótica* [Gráfico]. ¿Qué es la robótica? Recuperado el 20 de febrero de 2021 de https://revistaderobots.com/wp-content/uploads/2019/07/CUADRO-DE-INGENIER%C3%8DA-ROB%C3%93TICA-1-300x216.png

[2] Lie Yokuo (Autor de 'Lie Zin'). (2021, 26 marzo). [Ilustración]. Lie Yokuo. Recuperado el 5 de abril de 2021 de https://artemision.b-cdn.net/wp-content/uploads/2021/03/portada-artemision-blog-taoismo.jpg

[3] Blázquez Morales, L. F. (s. f.). [Ilustración]. *Museo Virtual*. Oficina Española de Patentes y Marcas. Recuperado el 5 de abril de 2021 de http://historico.oepm.es/museovirtual/galerias_tematicas.php?tipo=INVENTOR&xml=Her%C3%B3n%20de%20Alejandr%C3%ADa.xml

[4] *Elektro, el primer robot de la historia*. (2016, 29 agosto). [Fotografía]. Elektro, el primer robot de la historia y un símbolo del progreso. Recuperado el 5 de abril de 2021 de https://blog.todoelectronica.com/wp-content/uploads/2016/08/Joseph-Barnett-junto-a-Elektro-y-Sparko-imagen-history-computer.jpg

[5] *Falleció Joseph Engelberger, el padre de la robótica*. (2015, diciembre). Reportero Industrial. Recuperado el 7 de abril de 2021 de https://www.reporteroindustrial.com/temas/Fallecio-Joseph-Engelberger,-el-padre-de-la-robotica+109475

[6] *Unimate, el primer robot industrial*. (s. f.). [Fotografía]. Evolución de la robótica. Recuperado el 7 de abril de 2021 de https://assets.sutori.com/user-uploads/image/8547a8a0-f997-4aa7-9f4c-020f899ee9c5/b2-16-e1582228787654.jpeg

[7] ABB Robotics. (s. f.). *Área de Trabajo (Vista lateral) IRB120* [Ilustración]. Datos técnicos del robot industrial IRB 120. Recuperado 10 de marzo de 2021 de https://www07.abb.com/images/default-source/robotics/irb-120-working-range.jpg?sfvrsn=2

[8] ABB Robotics. (s. f.). *Área de Trabajo (Vista superior) IRB120* [Ilustración]. Datos técnicos del robot industrial IRB 120. Recuperado 10 de marzo de 2021 de https://www07.abb.com/images/default-source/robotics/irb-120-working-range06D9832C4AC278A0DCD2F4AA.jpg?sfvrsn=2

[9] ABB Robotics. (2019, 11 noviembre). *IRB 120 ABB's 6 axis robot – for flexible and compact production* [Archivo PDF]. Recuperado el 15 de marzo de 2021 de https://search.abb.com/library/Download.aspx?DocumentID=ROBO149EN_D&LanguageCode=en&DocumentPartId=2&Action=Launch

[10] ABB Robotics. (2020, 4 diciembre). *YuMi® IRB 14000* [Archivo PDF]. Recuperado el 15 de marzo de 2021 de https://search.abb.com/library/Download.aspx?DocumentID=ROBO149EN_D&LanguageCode=en&DocumentPartId=2&Action=Launch

[11] Gonzalo, M. (2015). *Simulación de una célula robotizada para la fabricación de piezas mecanizadas en un entorno industrial, utilizando RobotStudio* (trabajo final de grado). Universidad de Valladolid, Escuela de Ingenierías Industriales. Valladolid. Recuperado el el 15 de febrero de 2021 a partir de: http://uvadoc.uva.es/handle/10324/13173

[12] Juan Antonio, A. (2015). *Modelado de una célula robótica con fines educativos usando el programa Robot-Studio* (trabajo final de grado). Universidad de Valladolid, Escuela de Ingenierías Industriales. Valladolid. Recuperado el 15 de febrero de 2021 a partir de: http://uvadoc.uva.es/handle/10324/14441

[13] Álvaro, G. (2016). *Control remoto de una Célula robotizada mediante tecnología WI-FI* (trabajo final de grado). Universidad de Valladolid, Escuela de Ingenierías Industriales. Valladolid. Recuperado el 16 de febrero de 2021 a partir de: http://uvadoc.uva.es/handle/10324/17056

[14] Carlos, G. (2019). *Diseño de un sistema robótico educativo para jugar al ajedrez con robots industriales* (trabajo final de máster). Universidad de Valladolid, Escuela de Ingenierías Industriales. Valladolid. Recuperado el 16 de febrero de 2021 a partir de: http://uvadoc.uva.es/handle/10324/36877

[15] Víctor, L. (2020). *Diseño, simulación y fabricación de una estación robotizada universitaria* (trabajo final de máster). Universidad de Valladolid, Escuela de Ingenierías Industriales. Valladolid. Recuperado el 16 de febrero de 2021 a partir de: http://uvadoc.uva.es/handle/10324/41294

[16] Statista. (2020, 14 diciembre). *Principales empresas del sector de robótica industrial del mundo por ingresos* 2017. Recuperado el 10 de abril de 2021 de https://es.statista.com/estadisticas/601564/principales-empresas-del-sector-de-robotica-industrial-por-ingresos/

[17] ABB Robotics. (2021, 24 marzo). *Manual del Operador. RobotStudio* [Archivo PDF]. Recuperado el 1 de marzo de 2021 de https://search.abb.com/library/Download.aspx?DocumentID=3HAC032104-005&LanguageCode=es&DocumentPartId=&Action=Launch

[18] Autodesk. (2021, 21 abril). *Inventor Software | Autodesk*. Recuperado el 5 de marzo de 2021 de https://www.autodesk.com/products/inventor/overview?term=1-YEAR

[19] *Ensamblaje Modelo 3D*. (s. f.). [Ilustración]. 3DCadPortal. Autodesk Inventor. Recuperado el 5 de marzo de 2021 de https://www.3dcadportal.com/images/stories/autodesk/2015/ai2016-7.jpg

[20] The MathWorks, Inc. (s. f.). *MathWorks - Creadores de MATLAB y Simulink - MATLAB y Simulink*. MATLAB & Simulink. Recuperado 5 de marzo de 2021 de https://es.mathworks.com/

[21] The MathWorks, Inc. (s. f.-a). *Image Processing Toolbox*. MATLAB. Recuperado 5 de marzo de 2021 de https://es.mathworks.com/products/image.html

[22] The MathWorks, Inc. (s. f.-a). *OPC Toolbox*. MATLAB. Recuperado 5 de marzo de 2021 de https://es.mathworks.com/products/opc.html

[23] ABB Robotics. (s. f.-b). *Niveles Jerárquicos (Comunicación industrial)* [Gráfico]. Connect your robots to OPC UA. Recuperado 10 de marzo de 2021 de https://www07.abb.com/images/librariesprovider89/default-album/controllers/opc.jpg?sfvrsn=81df7217_1

[24] ABB Robotics. (2019, 11 noviembre). *Application manual IRC5 OPC UA Server* [Archivo PDF]. Recuperado el 10 de marzo de 2021 de https://robotapps.blob.core.windows.net/apps/ApplicationManual_ABB_IRC_OPC_UA_Server.pdf

[25] Rodríguez, M. (2016, 22 febrero). *Diferencias entre OPC clásico y OPC UA* [Gráfico]. Protocolo OPC UA. Características y aplicaciones en SCADA. https://revistadigital.inesem.es/gestion-integrada/files/2016/02/OPC-UA-1024x480.jpg

[26] Trilla, A, Peri, J. M, Violan, M, Pascual, E. V, & Rubinat, M. (2020, 12 marzo). *Medidas preventivas del Coronavirus SARS-CoV-2 | Hospital Clínic Barcelona.* Clínic Barcelona. Recuperado el 10 de marzo de 2021 de https://www.clinicbarcelona.org/asistencia/enfermedades/covid-19/prevencion

[27] ROBOTS INDUSTRIALES Y COBOTS (s. f.). *Robots industriales y cobots. en prevención de riesgos laborales* [Archivo PDF]. Recuperado el 15 de mayo de 2021 de https://www.femeval.es/dam/jcr:fd091e5f-3c97-42fd-9981-680e8232a645/GUIA_ROBOTS.pdf

[28] Relucio de la Fuente, R. (2005). *Miniproyecto automatización industrial (auti) (trabajo final de grado).* Universidad Politécnica de Cataluña, Barcelona. Recuperado el 15 de mayo de 2021 a partir de https://upcommons.upc.edu/bitstream/handle/2117/186037/40604-3452.pdf;jsessionid=801CE679B4E7D586BF49C35A9FF97F9E?sequence=1

[29] Mariscal Saldaña, M, García Herrero, S, & González, J. (2019). *Normativa de Robots Colaborativos (Cobots) y su influencia en la*. Prevención Integral & ORP Conference. Recuperado el 15 de mayo de 2021 de https://www.prevencionintegral.com/canal-orp/papers/orp-2019/normativa-robots-colaborativos-cobots-su-influencia-en-prevencion-riesgos-laborales

[30] Cepyme Aragón, Gobierno de Aragón (2012) *Guía técnica de seguridad en robótica* [Archivo PDF]. Recuperado el 15 de mayo de 2021 de https://www.femeval.es/dam/jcr:fd091e5f-3c97-42fd-9981-680e8232a645/GUIA_ROBOTS.pdf

[31] UNE Normalización Española. (s. f.). *UNE - Busca tu norma*. Busca tu norma (UNE). Recuperado 20 de mayo de 2021 de https://www.une.org/encuentra-tu-norma/busca-tu-norma/

[32] Plassa, J. (2009, 4 febrero). *Seguridad en una célula robótica* [Ilustración]. La protección segura de celdas robotizadas. Recuperado 25 de mayo de 2021 de https://img.interempresas.net/fotos/230050.gif

[33] Redacción AyJ Transmisiones. (2020, 26 octubre). *Materiales con los que se fabrican las bandas transportadoras*. Materiales de las bandas transportadoras | AyJ Transmisiones. Recuperado 25 de mayo de 2021 de https://www.ajtransmisiones.com/blog/materiales-de-bandas-transportadora

[34] ABB Robotics. (2014, 13 marzo). *ABB Robotics - Integrated Vision* [Video]. YouTube. Recuperado 25 de mayo de 2021 de https://www.youtube.com/watch?v=lYW0Sib1Er8

[35] AbrirArchivos. (s. f.). *¿Para qué son los archivos SAT?* Recuperado 1 de junio de 2021, de https://abrirarchivos.info/extension/sat

[36] Audi. (s. f.). Audi España Modelo R8. Recuperado 11 de marzo de 2021, de https://www.audi.es/es/web/es/modelos/r8/r8-coupe-v10-quattro.html

[37] Sanz, D. (s. f.). *Símbolos normalizados para los diagramas de bloques* [Ilustración]. JuegosRobotica. Diagrama de flujo. Recuperado 10 de mayo de 2021 de https://juegosrobotica.es/wp-content/uploads/diagrama-de-flujo-estandar.jpg

[38] Dia a dia en MME (s. f.). *Gama colores RGB y su código*. [Ilustración]. Recuperado el 2 de junio de 2021 de http://lh5.ggpht.com/_W8YChuXGYW8/SjQhxsBmm-I/AAAAAAAAAJ4/IxxzAQheAwo/image32.png?imgmax=800

ANEXOS

ANEXO A. CÓDIGO DESARROLLADO

Debido a la gran cantidad de código que se ha desarrollado, no tendría sentido incluirlo entero en el informe, debido a la extensión que adquiriría este. Además, teniendo en cuenta que la mayoría de código relevante se ha explicado previamente en diversos capítulos y que todo lo desarrollado se incluye en el fichero comprimido 'zip', el cual también se sube al repositorio documental de la UVA.

Por lo que únicamente se ha optado por mostrar la estructura de un código/programa, por software utilizado. Y si el lector está interesado en algún código completo en específico podrá visitar el contenido de los Anejos del fichero comprimido.

ANEXO A.1. EJEMPLO DE CÓDIGO EN RAPID

Maniobra IRB120

```
MODULE Maniobra_IRB120
   !***********************************************************
   !
   ! Módulo:  Maniobra_IRB120
   !
   ! Descripción:
   !   Este módulo contiene el código con el que se ejecutan
   ! los movimientos del robot IRB120.
   ! En el programa existen dos opciones:
   ! - Realización del montaje de un coche de juguete
   ! - Selección de un cubo en función de su validez
   !
   ! Los componentes del coche y los cubos se suministran
   ! a través de un Conveyor.
   !
   ! Autor: Rodrigo Sancho García
   !
   ! Versión: 1.0
   !
   !***********************************************************

   !***********************************************************
   !
   ! Declaración de Variables Utilizadas
   !
   !   En las siguientes líneas, se declaran los diferentes
   ! robtargets que necesita ejecutar el robot, para la
   ! realización del montaje del coche y seleccion del cubo.
   ! Además, se incluyen variables persistentes que serán
   ! modificadas durante la eejcución del programa
   ! (num,pos,bool,string...)
   !
   !***********************************************************

   !VARIABLES ROBTARGET
   CONST robtarget
HOME:=[[416.3153533,0,564],[0.5,0,0.866025404,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget CogerUPIzda:=[[442,0,400],[0,1,0,0],[0,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget CogerUPDcha:=[[442,0,400],[0,1,0,0],[-1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget Coger_ParteInferior:=[[487,15,210],[0,1,0,0],[0,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget Coger_Chasis:=[[485,0,223.625],[0,1,0,0],[-1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget Coger_Transmision:=[[484.611,-108.766,255.995],[0,1,0,0],[0,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    CONST robtarget Coger_Rueda:=[[485.037,-132.163,226.5],[0,1,0,0],[-
1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Coger_Capo:=[[483.55,-88.75,245.313],[0,1,0,0],[-
1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Coger_ParteTrasera:=[[485.758,-110.371,260.99],[0,1,0,0],[0,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Coger_Interior:=[[485,-77.5,233.031],[0,1,0,0],[-
1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Coger_Asiento:=[[505,-150,241],[0,1,0,0],[-1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Coger_ParteSuperior:=[[483,17.5,288.8],[0,1,0,0],[0,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget Dejar_ParteInferiorIzda:=[[-355,450,79.5],[0,-0.707106781,0.707106781,0],[1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_ChasisDcha:=[[-355,-450,100],[0,-0.707106781,0.707106781,0],[-
2,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_TransmisionIzda:=[[-235,450,128],[0,-
0.707106781,0.707106781,0],[1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_RuedaIzda:=[[-255,455,91.5],[0,1,0,0],[1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_RuedaDcha:=[[-255,-455,91.5],[0,1,0,0],[-
2,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_CapoDcha:=[[-280,-460,112],[0,-0.707106781,0.707106781,0],[-
2,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_ParteTraseraIzda:=[[-290,450,129.5],[0,0.707106781,0.707106781,0],[1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_InteriorDcha:=[[-290,-460,101.5],[0,-0.707106781,0.707106781,0],[-
2,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_AsientoDcha:=[[-255,-455,113],[0,0.707106781,0.707106781,0],[-
2,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_AsientoIzda:=[[-255,455,113],[0,0.707106781,0.707106781,0],[-
1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Dejar_ParteSuperior:=[[-546.25,0,275.5],[0,-0.707106781,0.707106781,0],[0,-1,-
1,7],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget DejarCocheMontado:=[[30,-500,243.4],[0,1,0,0],[-
1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    PERS robtarget CogerFigura:=[[512.222,-100,300],[0,1,0,0],[0,0,0,0],[9E+9,9E+9,9E+9,9E+9,9E+9,9E+9]];
    PERS robtarget DejarFigura:=[[-345,450,175],[0,-0.707107,0.707107,0],[1,0,-
1,0],[9E+9,9E+9,9E+9,9E+9,9E+9,9E+9]];

    !VARIABLES POS
    PERS pos PosInicial_Cubo:=[1609,1000,200];
    PERS pos Pos_Cubo:=[1609,-152,200];
    PERS pos Color_Cubo:=[0,255000,0];

    !VARIABLES NUM
    PERS num TamañoX:=0.1;
    PERS num TamañoY:=0.1;
    PERS num TamañoZ:=0.1;
    PERS num SeleccionAccion:=2;
    PERS num EjeX:=1612.22;
    PERS num Altura:=100;

    !VARIABLE STRING
    PERS string EstadoProceso:="Terminado";

    !VARIABLES BOOL
    PERS bool MontajeRealizado:=FALSE;
    PERS bool SeleccionFiguraRealizada:=FALSE;

    !*********************************************************
    !
    ! Procedimiento Main
    !
    !   Este es el programa principal que se ejecuta en el
    ! IRB120. Primeramente, se resetean todas las señales
    ! y variables involucradas en el proceso.
    ! Posteriormente se entra en un bucle infinito, en
    ! el que se decide la accion en función de la selección
    ! del usuario.
    !
    !*********************************************************
    PROC main()
```

```
    ResetSeñales;
    SeleccionAccion:=0;
    EstadoProceso:="Terminado";
    WaitUntil SeleccionAccion<>0;
    MontajeRealizado:=FALSE;
    SeleccionFiguraRealizada:=FALSE;
    WHILE TRUE DO
      TEST SeleccionAccion
      CASE 1:
        IF EstadoProceso="Terminado" AND MontajeRealizado=FALSE THEN
          MontajeCoche;
        ENDIF
      CASE 2:
        WaitDO PiezaAnalizada,1;
        IF PiezaIncorrecta=0 AND SeleccionFiguraRealizada=FALSE THEN
          ManejoPieza;
        ENDIF
      ENDTEST
    ENDWHILE
  ENDPROC

  !*********************************************************
  !
  ! Procedimiento ManejoPieza
  !
  !   Este es el procedimiento encargado de la configuración
  ! del movimiento del robot, para coger el cubo. Para ello,
  ! le han tenido que llegar los valores detectados por
  ! visión artificial del cubo. En función de estos se
  ! desplaza unas ciertas cantidades.
  !
  !*********************************************************
  PROC ManejoPieza()
    EstadoProceso:="Empezando";
    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    CogerFigura.trans.x:=EjeX+(Altura/2)-1150;
    CogerFigura.trans.y:=(Altura/2)-150;
    CogerFigura.trans.z:=Altura+200;
    MoveJ CogerFigura,v100,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO CogerPieza,1;
    WaitTime 5;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
    DejarFigura.trans.z:=Altura+75;
    MoveJ DejarFigura,v500,z100,TCP_Ventosa\WObj:=wobj0;
    WaitTime 1;
    Reset CogerPieza;
    WaitTime 1;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
    SeleccionFiguraRealizada:=TRUE;
    EstadoProceso:="Terminado";
  ENDPROC

  !*********************************************************
  !
  ! Procedimiento MontajeCoche
  !
  !   Este es el procedimiento encargado de la configuración
  ! del movimiento del robot, para coger las diferentes
  ! piezas que conforman el coche de juguete.
  ! Selecciona el lado derecho o izquierdo en función del
  ! último lado al que haya depositado una pieza.
  !
  !*********************************************************
  PROC MontajeCoche()
    SetDO ComienzaMontaje,1;
    EstadoProceso:="Empezando";
    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza1,1;
    WaitDI SensorPieza,1;
    Trayectoria_ParteInferior;

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza2,1;
```

```
    WaitDI SensorPieza,1;
    Trayectoria_Chasis;

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza7,1;
    WaitDI SensorPieza,1;
    Trayectoria_Transmision;

    WaitTime 10;
    FOR i FROM 1 TO 2 DO
      MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
      IF i=1 THEN
        SetDO NuevaPieza8,1;
        Reset NuevaPieza8;
      ELSEIF i=2 THEN
        SetDO NuevaPieza3,1;
        Reset NuevaPieza3;
      ENDIF
      WaitDI SensorPieza,1;
      Trayectoria_RuedasDcha;
      MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
      IF i=1 THEN
        SetDO NuevaPieza8,1;
        Reset NuevaPieza8;
      ELSEIF i=2 THEN
        SetDO NuevaPieza3,1;
        Reset NuevaPieza3;
      ENDIF
      WaitDI SensorPieza,1;
      Trayectoria_RuedasIzda;
    ENDFOR

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza9,1;
    WaitDI SensorPieza,1;
    Trayectoria_Capo;

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza10,1;
    WaitDI SensorPieza,1;
    Trayectoria_ParteTrasera;

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza11,1;
    WaitDI SensorPieza,1;
    Trayectoria_Interior;

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza4,1;
    WaitDI SensorPieza,1;
    Trayectoria_AsientoCopiloto;

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza5,1;
    WaitDI SensorPieza,1;
    Trayectoria_AsientoPiloto;

    MoveJ HOME,v1000,z100,TCP_Ventosa\WObj:=wobj0;
    SetDO NuevaPieza6,1;
    WaitDI SensorPieza,1;
    Trayectoria_FinalSincronizada;
    EstadoProceso:="Terminado";
    MontajeRealizado:=TRUE;
    Reset ComienzaMontaje;
  ENDPROC

  !Reseteo de Señales Implicadas
  PROC ResetSeñales()
    Reset NuevaPieza1;
    Reset NuevaPieza2;
    Reset NuevaPieza3;
    Reset NuevaPieza4;
    Reset NuevaPieza5;
```

```
    Reset NuevaPieza6;
    Reset NuevaPieza7;
    Reset NuevaPieza8;
    Reset NuevaPieza9;
    Reset NuevaPieza10;
    Reset NuevaPieza11;
    Reset CogerGrupo;
    Reset AñadirAGrupo;
    Reset CrearCubo;
    Reset CambiarColorCubo;
    Reset PiezaIncorrecta;
    Reset ComienzaMontaje;
    Reset PiezaAnalizada;
ENDPROC

!Trayectorias Implicadas en el Montaje del Coche
PROC Trayectoria_ParteInferior()
    MoveJ Coger_ParteInferior,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDO CogerPieza,1;
    MoveL CogerUPIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Dejar_ParteInferiorIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_ParteInferiorIzda,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
ENDPROC

PROC Trayectoria_Chasis()
    MoveJ Coger_Chasis,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    MoveJ CogerUPDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Offs(Dejar_ChasisDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveL Dejar_ChasisDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_ChasisDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
ENDPROC

PROC Trayectoria_Transmision()
    MoveJ Coger_Transmision,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    MoveL CogerUPIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Dejar_TransmisionIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_TransmisionIzda,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
ENDPROC

PROC Trayectoria_RuedasDcha()
    MoveJ Coger_Rueda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    MoveL CogerUPDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Offs(Dejar_RuedaDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveL Dejar_RuedaDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_RuedaDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
ENDPROC

PROC Trayectoria_RuedasIzda()
    MoveJ Coger_Rueda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    MoveL CogerUPDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Dejar_RuedaIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_RuedaIzda,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
ENDPROC

PROC Trayectoria_Capo()
    MoveJ Coger_Capo,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    MoveL CogerUPDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Offs(Dejar_CapoDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
```

```
    MoveL Dejar_CapoDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_CapoDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
  ENDPROC

  PROC Trayectoria_ParteTrasera()
    MoveJ Coger_ParteTrasera,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    MoveL CogerUPIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Dejar_ParteTraseraIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_ParteTraseraIzda,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
  ENDPROC

  PROC Trayectoria_Interior()
    MoveJ Coger_Interior,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    MoveL CogerUPDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Offs(Dejar_InteriorDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveL Dejar_InteriorDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_InteriorDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
  ENDPROC

  PROC Trayectoria_AsientoCopiloto()
    MoveJ Coger_Asiento,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    MoveL CogerUPDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Dejar_AsientoIzda,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_AsientoIzda,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
  ENDPROC

  PROC Trayectoria_AsientoPiloto()
    MoveJ Coger_Asiento,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    MoveL CogerUPDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ Offs(Dejar_AsientoDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveL Dejar_AsientoDcha,v500,fine,TCP_Ventosa\WObj:=wobj0;
    Reset CogerPieza;
    MoveL Offs(Dejar_AsientoDcha,0,0,100),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
  ENDPROC

  PROC Trayectoria_FinalSincronizada()
    MoveJ Coger_ParteSuperior,v500,fine,TCP_Ventosa\WObj:=wobj0;
    SetDo CogerPieza,1;
    SetDO Sincronizacion,1;
    WaitDI SincronizacionIN,1;
    Reset Sincronizacion;
    MoveJ Offs(Dejar_ParteSuperior,0,0,40),v500,fine,TCP_Ventosa\WObj:=wobj0;
    MoveL Dejar_ParteSuperior,v500,fine,TCP_Ventosa\WObj:=wobj0;
    WaitTime 1;
    Reset CogerPieza;
    SetDO AñadirAGrupo,1;
    SetDO CogerGrupo,1;
    WaitTime 1;
    MoveJ DejarCocheMontado,v500,z100,TCP_Ventosa\WObj:=wobj0;
    WaitTime 1;
    Reset CogerGrupo;
    MoveJ HOME,v500,z100,TCP_Ventosa\WObj:=wobj0;
  ENDPROC

ENDMODULE
```

ANEXO A.2. EJEMPLO DE CÓDIGO (SCRIPTS) EN MATLAB

Algoritmo desarrollado de visión artificial

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Función MATLAB con el algoritmo desarrollado para el tratamiento de una
% imagen. Mas en concreto, se encarga de detectar si hay una pieza en el
% conveyor, y en caso afirmativo, detectar su color, tamaño y posición.
% Datos de Entrada:
%   Imagen Capturada de la Simulación
% Operaciones Realizadas:
%   Deteccion de pieza, tamaño, color y posición
% Datos de Salida:
%   Color,Tamaño y Posición de la Pieza detectada
%
% Creado por:
%   - Rodrigo Sancho García
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [Color,Area,Posicion] = Algoritmo_VisionArtificial(Imagen)
ConfiguraEspacioVision = evalin('base','ConfiguraEspacioVision');
CoordenadasCorte=[187.5 127.5 30 45]; %Coordenadas de la zona de interes
% Método 1 de pasar la imagen a escala de grises
Img=rgb2gray(Imagen);
% Método 2 de pasar la imagen a escala de grises (con umbral)
umb=graythresh(Imagen);
ImgGray=im2bw(Imagen,umb);

% Configuración Zona a Detectar si el Usuario ha seleccionado esta opción
if ConfiguraEspacioVision==1
    imcrop(ImgGray);
    CoordenadasCorte = evalin('base','CoordenadasCorte');
end

%Recorte Zona interes
ImgGrayCut=imcrop(ImgGray,CoordenadasCorte);

% Deteccion de Bordes de la imagen con método 'Canny'
ImgBordes=edge(ImgGrayCut,'Canny');
% Método 2: Rastrear los límites de la región en la imagen binaria
[B,L]=bwboundaries(ImgGrayCut,'noholes');

% Rellenar huecos de la imagen anterior en la que se detectaron los bordes
ImgRellenada=imfill(ImgBordes,'holes');

% Tratamiento de la imagen, para marcar el cubo detectado y nombrarle
f=figure('visible','off');
S6=subplot(1,2,1);
set(S6,'Visible','off');

% Encuentra componentes conexas y las asigna etiquetas (1, 2, 3…)
% Devuelve una estructura con diversos campos, entre ellos:
%      cc.NumObjects:  número de objetos encontrados
CC = bwconncomp(ImgRellenada);

% L:  imagen de etiquetas  (0 fondo; 1, 2… las distintas comp. conexas)
% P.ej.: para separar objetos y rotarlos
L=labelmatrix(CC);

% Muestra imagen de etiquetas
% ('jet, 'k' para fondo negro)
RellenoColores=label2rgb(L, 'jet', 'k');
imshow(RellenoColores);

% Medir las propiedades de las regiones de la imagen procesada
propied=regionprops(ImgRellenada,'basic');
hold on

X=0;
Y=0;
AreaFigura=0;
```

```
% Bucle para detectar todos los objetos rectangulares de la imagen y
% marcarlos. Si detecta el cubo lo marca en verde y le asigna un nombre,
% los demás objetos serán marcados en rojo.
% Además, se obtiene el tamaño y posición (Centroide) del cubo detectado
for n=1:size(propied)
    rectangle('Position',propied(n).BoundingBox,'EdgeColor','r','LineWidth',2);
    centroide=propied(n).Centroid;
    Area=propied(n).Area;
    plot(centroide(1),centroide(2),'ko');
    if(centroide(1)<25 & centroide(1)>10 & centroide(2)<30 & centroide(2)>15 &
Area<110)

rectangle('Position',propied(n).BoundingBox,'EdgeColor','g','LineWidth',5);
        text(centroide(1)-
4,centroide(2)+7,'Cubo','Color','g','Fontsize',15,'Fontweight', 'bold');
        AreaFigura=propied(n).Area;
        X=centroide(1);
        Y=centroide(2);
    end
end

% Asignación de las imagenes obtenidas en el Workspace, para posteriormente
% mostrarlas en la Interfaz de App Designer
assignin('base','Imagen',Imagen);
assignin('base','ImagenGrises',ImgGray);
assignin('base','ImagenRecortada',ImgGrayCut);
assignin('base','ImagenBordes',ImgBordes);
assignin('base','ImagenRellenada',ImgRellenada);

F = getframe(S6);
assignin('base','FiguraDetectada',F.cdata);

% Tratamiento de los Colores de la Imagen
ZonaInteres=imcrop(Imagen,CoordenadasCorte);
assignin('base','ZonaInteres',ZonaInteres);
ZonaInteresGris=rgb2gray(ZonaInteres);

% Divide las 3 capas de una imagen RGB
imR=double(ZonaInteres(:,:,1));
imG=double(ZonaInteres(:,:,2));
imB=double(ZonaInteres(:,:,3));
% Se eliminan el resto de colores de la imagen excepto Red ó Green ó Blue
% Creando unas máscaras para esos colores
% Fondo: Negro y Color correspondiente: Blanco
imagenR=imR-imG-imB;
imagenG=imG-imR-imB;
imagenB=imB-imR-imG;
% Asignación de las anteriores imagenes
assignin('base','imagenR',imagenR);
assignin('base','imagenG',imagenG);
assignin('base','imagenB',imagenB);

imagen_binaria=imagenR>50;
imagen_binaria1=imagenG>50;
imagen_binaria2=imagenB>50;

% Se realiza el filtrado medio de la imagen en dos dimensiones
img_filt_bin=medfilt2(imagen_binaria);
img_filt_bin1=medfilt2(imagen_binaria1);
img_filt_bin2=medfilt2(imagen_binaria2);

% Se filtran las imagenes, para mantener unicamente el color buscado
% (R,G,B)
mascara=1-img_filt_bin;
mascara1=1-img_filt_bin1;
mascara2=1-img_filt_bin2;

imagen_roja=double(ZonaInteresGris)/255;
imagen_verde=double(ZonaInteresGris) .* mascara/255;
imagen_azul=double(ZonaInteresGris) .* mascara/255;
img_final=cat(3,imagen_roja,imagen_verde,imagen_azul);

imagen_roja=double(ZonaInteresGris) .* mascara1/255;
imagen_verde=double(ZonaInteresGris)/255;
```

```
imagen_azul=double(ZonaInteresGris) .* mascara1/255;
img_final1=cat(3,imagen_roja,imagen_verde,imagen_azul);

imagen_roja=double(ZonaInteresGris) .* mascara2/255;
imagen_verde=double(ZonaInteresGris) .* mascara2/255;
imagen_azul=double(ZonaInteresGris)/255;
img_final2=cat(3,imagen_roja,imagen_verde,imagen_azul);

assignin('base','AislaRojo',img_final);
assignin('base','AislaVerde',img_final1);
assignin('base','AislaAzul',img_final2);

ColorFigura='NULL';

% Algoritmo Detección Color
% Se recorre todos los pixeles de la imagen (Para las 3 capas, RGB) y
% detecta si existen pixeles de los colores buscados
[Filas,Columnas] = size(ZonaInteres);
for i=1:Filas
    for j=1:(Columnas/3)
        VectorColores=impixel(ZonaInteres,j,i);
        %%%%%ROJO%%%%%
        if VectorColores(1)>100 & VectorColores(2)<40 & VectorColores(3)<40
            ColorFigura='Rojo';
        %%%%%VERDE%%%%%
        elseif VectorColores(1)<40 & VectorColores(2)>100 & VectorColores(3)<40
            ColorFigura='Verde';
        %%%%%AZUL%%%%%
        elseif VectorColores(1)<40 & VectorColores(2)<40 & VectorColores(3)>100
            ColorFigura='Azul';
        %%%%%AMARILLO%%%%%
        elseif VectorColores(1)>100 & VectorColores(2)>100 & VectorColores(3)<40
            ColorFigura='Amarillo';
        %%%%%MAGENTA%%%%%
        elseif VectorColores(1)>100 & VectorColores(2)<40 & VectorColores(3)>100
            ColorFigura='Magenta';
        %%%%%CYAN%%%%%
        elseif VectorColores(1)<40 & VectorColores(2)>100 & VectorColores(3)>100
            ColorFigura='Cyan';
        %%%%%NEGRO%%%%%
        elseif VectorColores(1)<40 & VectorColores(2)<40 & VectorColores(3)<30
            ColorFigura='Negro';
        end
    end
end

% Asignación del Tamaño, Posición y Color detectado
Area=AreaFigura;
Posicion=X;
Color=ColorFigura;

fprintf('El area de la figura es: %d\r\n',AreaFigura);
fprintf('La posicion de la figura es: "X=%d":"Y=%d"\r\n',X,Y);
fprintf('El color de la figura es: %s\r\n',ColorFigura);

end
```

ANEXO A.3. EJEMPLO CÓDIGO DE UNA VENTANA DE LA INTERFAZ

Código Menú Principal de la interfaz

```
classdef InterfazTFG < matlab.apps.AppBase
% Properties that correspond to app components
properties (Access = public)
UIFigure matlab.ui.Figure
EjecutarSimulacionButton matlab.ui.control.Button
Image matlab.ui.control.Image
Image2 matlab.ui.control.Image
ActivarInterfazLabel matlab.ui.control.Label
EncendidoSwitch matlab.ui.control.Switch
Lamp matlab.ui.control.Lamp
CleanAllButton matlab.ui.control.Button
RedCross matlab.ui.control.Image
RedCross_2 matlab.ui.control.Image
GreenCheck matlab.ui.control.Image
GreenCheck_2 matlab.ui.control.Image
ColorDetectadoEditFieldLabel matlab.ui.control.Label
ColorDetectadoEditField matlab.ui.control.EditField
TamaoDetectadoEditFieldLabel matlab.ui.control.Label
SizeDetectadoEditField matlab.ui.control.EditField
Image3 matlab.ui.control.Image
ACCIONESLabel matlab.ui.control.Label
ButtonGroup matlab.ui.container.ButtonGroup
MontajeCocheButton matlab.ui.control.ToggleButton
DeteccionFiguraButton matlab.ui.control.ToggleButton
SeleccionadoLabel matlab.ui.control.Label
DeseadoLabel matlab.ui.control.Label
SeleccionColorDeseado matlab.ui.control.DropDown
SeleccionSizeDeseado matlab.ui.control.DropDown
ColorDropDownLabel matlab.ui.control.Label
SeleccionColor matlab.ui.control.DropDown
TamaoDropDownLabel matlab.ui.control.Label
SeleccionSize matlab.ui.control.DropDown
SeleccinPosicionSliderLabel matlab.ui.control.Label
PosicionMediano matlab.ui.control.Slider
SeleccinPosicionSlider_3Label matlab.ui.control.Label
PosicionGrande matlab.ui.control.Slider
ValorPosicion matlab.ui.control.NumericEditField
SeleccinPosicionSlider_2Label matlab.ui.control.Label
PosicionPequeno matlab.ui.control.Slider
ConfigurarEspacioTrabajoVisinArtificialButton matlab.ui.control.StateButton
CoordenadasEditFieldLabel matlab.ui.control.Label
CoordenadasEditField matlab.ui.control.EditField
AutorRodrigoSanchoGarcaLabel matlab.ui.control.Label
Image4 matlab.ui.control.Image
Image5 matlab.ui.control.Image
TITULOTFGBLABLABLALabel matlab.ui.control.Label
Coche matlab.ui.control.Image
Estacion matlab.ui.control.Image
Cajas matlab.ui.control.Image
CajasSeleccionado matlab.ui.control.Image
CocheSeleccionado matlab.ui.control.Image
PosicionDetectadaEditFieldLabel matlab.ui.control.Label
PosicionDetectadaEditField matlab.ui.control.NumericEditField
ImagenesVisinArtificialButton matlab.ui.control.Button
End
% Callbacks that handle component events
methods (Access = private)
% Code that executes after component creation
function startupFcn(app)
evalin('base', 'clear ALL')
app.Lamp.Color='Red';
app.ButtonGroup.Enable='Off';
app.CleanAllButton.Enable='Off';
app.ImagenesVisinArtificialButton.Enable='Off';
app.SeleccionColor.Enable='Off';
app.SeleccionSize.Enable='Off';
app.SeleccionColorDeseado.Enable='Off';
```

```
app.SeleccionSizeDeseado.Enable='Off';
app.ConfigurarEspacioTrabajoVisinArtificialButton.Enable='Off';
app.PosicionPequeno.Enable='Off';
app.PosicionMediano.Enable='Off';
app.PosicionGrande.Enable='Off';
app.ValorPosicion.Enable='Off';
app.CoordenadasEditField.Enable='Off';
app.EjecutarSimulacionButton.Enable='Off';
app.CocheSeleccionado.Visible='Off';
app.CajasSeleccionado.Visible='Off';
app.RedCross.Visible='Off';
app.RedCross_2.Visible='Off';
app.GreenCheck.Visible='Off';
app.GreenCheck_2.Visible='Off';
app.PosicionPequeno.Visible='Off';
app.PosicionMediano.Visible='On';
app.PosicionGrande.Visible='Off';
Color='[255000,0,0]';
Size=[0.15,0.15,0.15];
Posicion='[1565,1000,200]';
PosicionX=1590;
assignin('base','Color',Color);
assignin('base','Size',Size);
assignin('base','Posicion',Posicion);
assignin('base','SeleccionModo',1);
assignin('base','PosicionX',PosicionX);
assignin('base','ConfiguraEspacioVision',0);
app.DeteccionFiguraButton.Value=false;
assignin('base','ColorDeseado',app.SeleccionColorDeseado.Value);
assignin('base','SizeDeseado',app.SeleccionSizeDeseado.Value);
end

% Value changed function: EncendidoSwitch
function EncendidoSwitchValueChanged(app, event)
value = app.EncendidoSwitch.Value;
assignin('base','Interruptor',value);
switch value
case 'On'
app.Lamp.Color='Green';
app.ButtonGroup.Enable='On';
app.CleanAllButton.Enable='On';
app.ImagenesVisinArtificialButton.Enable='On';
app.SeleccionColor.Enable='On';
app.SeleccionSize.Enable='On';
app.SeleccionColorDeseado.Enable='On';
app.SeleccionSizeDeseado.Enable='On';
app.ConfigurarEspacioTrabajoVisinArtificialButton.Enable='On';
app.PosicionPequeno.Enable='On';
app.PosicionMediano.Enable='On';
app.PosicionGrande.Enable='On';
app.ValorPosicion.Enable='On';
app.CoordenadasEditField.Enable='On';
app.EjecutarSimulacionButton.Enable='On';
if app.MontajeCocheButton.Value==true
app.CocheSeleccionado.Visible='On';
app.CajasSeleccionado.Visible='Off';
else
app.CocheSeleccionado.Visible='Off';
app.CajasSeleccionado.Visible='On';
end
case 'Off'
app.Lamp.Color='Red';
app.ButtonGroup.Enable='Off';
app.CleanAllButton.Enable='Off';
app.ImagenesVisinArtificialButton.Enable='Off';
app.SeleccionColor.Enable='Off';
app.SeleccionSize.Enable='Off';
app.SeleccionColorDeseado.Enable='Off';
app.SeleccionSizeDeseado.Enable='Off';
app.ConfigurarEspacioTrabajoVisinArtificialButton.Enable='Off';
app.PosicionPequeno.Enable='Off';
```

```
app.PosicionMediano.Enable='Off';
app.PosicionGrande.Enable='Off';
app.ValorPosicion.Enable='Off';
app.CoordenadasEditField.Enable='Off';
app.EjecutarSimulacionButton.Enable='Off';
app.CocheSeleccionado.Visible='Off';
app.CajasSeleccionado.Visible='Off';
end
end

% Value changed function: SeleccionColor
function SeleccionColorValueChanged(app, event)
value = app.SeleccionColor.Value;
switch value
case 'Rojo'
Color='[255000,0,0]';
case 'Verde'
Color='[0,255000,0]';
case 'Azul'
Color='[0,0,255000]';
case 'Amarillo'
Color='[255000,255000,0]';
case 'Magenta'
Color='[255000,0,255000]';
case 'Cyan'
Color='[0,255000,255000]';
case 'Negro'
Color='[0,0,0]';
end
assignin('base','Color',Color);
assignin('base','SeleccionColor',app.SeleccionColor.Value);
end

% Value changed function: SeleccionSize
function SeleccionSizeValueChanged(app, event)
value = app.SeleccionSize.Value;
switch value
case 'Pequeño'
Size=[0.1,0.1,0.1];
app.PosicionPequeno.Visible='On';
app.PosicionMediano.Visible='Off';
app.PosicionGrande.Visible='Off';
Posicion='[1590,1000,200]';
case 'Mediano'
Size=[0.15,0.15,0.15];
app.PosicionPequeno.Visible='Off';
app.PosicionMediano.Visible='On';
app.PosicionGrande.Visible='Off';
Posicion='[1565,1000,200]';
case 'Grande'
Size=[0.2,0.2,0.2];
app.PosicionPequeno.Visible='Off';
app.PosicionMediano.Visible='Off';
app.PosicionGrande.Visible='On';
Posicion='[1540,1000,200]';
end
assignin('base','Size',Size);
assignin('base','SeleccionSize',app.SeleccionSize.Value);
assignin('base','Posicion',Posicion);
end

% Button pushed function: EjecutarSimulacionButton
function EjecutarSimulacionButtonPushed(app, event)
app.RedCross.Visible='Off';
app.RedCross_2.Visible='Off';
app.GreenCheck.Visible='Off';
app.GreenCheck_2.Visible='Off';
app.ButtonGroup.Enable='Off';
app.CleanAllButton.Enable='Off';
app.ImagenesVisinArtificialButton.Enable='Off';
app.SeleccionColor.Enable='Off';
```

```
app.SeleccionSize.Enable='Off';
app.SeleccionColorDeseado.Enable='Off';
app.SeleccionSizeDeseado.Enable='Off';
app.ConfigurarEspacioTrabajoVisinArtificialButton.Enable='Off';
app.PosicionPequeno.Enable='Off';
app.PosicionMediano.Enable='Off';
app.PosicionGrande.Enable='Off';
app.ValorPosicion.Enable='Off';
app.CoordenadasEditField.Enable='Off';
app.EjecutarSimulacionButton.Enable='Off';
app.CocheSeleccionado.Visible='Off';
app.CajasSeleccionado.Visible='Off';
OPC;
if app.DeteccionFiguraButton.Value==true
Deteccion(app);
OPC2;
end
app.ButtonGroup.Enable='On';
app.CleanAllButton.Enable='On';
app.ImagenesVisinArtificialButton.Enable='On';
app.SeleccionColor.Enable='On';
app.SeleccionSize.Enable='On';
app.SeleccionColorDeseado.Enable='On';
app.SeleccionSizeDeseado.Enable='On';
app.ConfigurarEspacioTrabajoVisinArtificialButton.Enable='On';
app.PosicionPequeno.Enable='On';
app.PosicionMediano.Enable='On';
app.PosicionGrande.Enable='On';
app.ValorPosicion.Enable='On';
app.CoordenadasEditField.Enable='On';
app.EjecutarSimulacionButton.Enable='On';
end

% Value changing function: PosicionMediano
function PosicionMedianoValueChanging(app, event)
changingValue = event.Value;
app.ValorPosicion.Value=changingValue;
PosicionX=int2str(changingValue+1565);
Posicion=strcat('[',PosicionX,',1000,200]');
assignin('base','Posicion',Posicion);
assignin('base','PosicionX',changingValue+1590);
assignin('base','BarraMediana',changingValue);
end

% Selection changed function: ButtonGroup
function ButtonGroupSelectionChanged(app, event)
%selectedButton = app.ButtonGroup.SelectedObject;
assignin('base','Modo',app.MontajeCocheButton.Value);
if app.MontajeCocheButton.Value==true
assignin('base','SeleccionModo',1);
app.CocheSeleccionado.Visible='On';
app.CajasSeleccionado.Visible='Off';
else
assignin('base','SeleccionModo',2);
app.CocheSeleccionado.Visible='Off';
app.CajasSeleccionado.Visible='On';
end
end

% Value changing function: PosicionPequeno
function PosicionPequenoValueChanging(app, event)
changingValue = event.Value;
app.ValorPosicion.Value=changingValue;
PosicionX=int2str(changingValue+1590);
Posicion=strcat('[',PosicionX,',1000,200]');
assignin('base','Posicion',Posicion);
assignin('base','PosicionX',changingValue+1590);
assignin('base','BarraPequena',changingValue);
end

% Value changing function: PosicionGrande
```

```
function PosicionGrandeValueChanging(app, event)
changingValue = event.Value;
app.ValorPosicion.Value=changingValue;
PosicionX=int2str(changingValue+1540);
Posicion=strcat('[',PosicionX,',1000,200]');
assignin('base','Posicion',Posicion);
assignin('base','PosicionX',changingValue+1590);
assignin('base','BarraGrande',changingValue);
end

% Value changed function: SeleccionColorDeseado
function SeleccionColorDeseadoValueChanged(app, event)
value = app.SeleccionColorDeseado.Value;
assignin('base','ColorDeseado',value);
end

% Value changed function: SeleccionSizeDeseado
function SeleccionSizeDeseadoValueChanged(app, event)
value = app.SeleccionSizeDeseado.Value;
assignin('base','SizeDeseado',value);
end

% Button pushed function: CleanAllButton
function CleanAllButtonPushed(app, event)
evalin( 'base', 'clearvars *' );
delete(app.UIFigure); %close the app
InterfazTFG; %re-open it again
end

% Value changed function: CoordenadasEditField
function CoordenadasEditFieldValueChanged(app, event)
value = app.CoordenadasEditField.Value;
value = value(2:end - 1);
vec = str2double(strsplit(value));
assignin('base','CoordenadasCorte',vec);
end

% Value changed function:
% ConfigurarEspacioTrabajoVisinArtificialButton
function ConfigurarEspacioTrabajoVisinArtificialButtonValueChanged(app, event)
value = app.ConfigurarEspacioTrabajoVisinArtificialButton.Value;
if value==true
assignin('base','ConfiguraEspacioVision',1);
else
assignin('base','ConfiguraEspacioVision',0);
end
assignin('base','BotonConfiguracion',app.ConfigurarEspacioTrabajoVisinArtificialButton.Val
ue);
end

% Image clicked function: Image
function Deteccion(app, event)
ColorCorrecto= evalin('base','ColorCorrecto');
ColorDetectado= evalin('base','ColorDetectado');
SizeCorrecto= evalin('base','SizeCorrecto');
SizeDetectado= evalin('base','SizeDetectado');
PosicionDetectada= evalin('base','PosicionDetectada');
app.ColorDetectadoEditField.Value=ColorDetectado;
app.SizeDetectadoEditField.Value=SizeDetectado;
app.PosicionDetectadaEditField.Value=PosicionDetectada;
if ColorCorrecto==true
app.GreenCheck.Visible='On';
else
app.RedCross.Visible='On';
end

if SizeCorrecto==true
app.GreenCheck_2.Visible='On';
else
app.RedCross_2.Visible='On';
end
```

```
end

% Button pushed function: ImagenesVisinArtificialButton
function ImagenesVisinArtificialButtonPushed(app, event)
InterfazTFG_Imagenes;
end
end

% Component initialization
methods (Access = private)

% Create UIFigure and components
function createComponents(app)

% Create UIFigure and hide until all components are created
app.UIFigure = uifigure('Visible', 'off');
app.UIFigure.Color = [1 1 1];
app.UIFigure.Position = [100 100 725 533];
app.UIFigure.Name = 'MATLAB App';

% Create EjecutarSimulacionButton
app.EjecutarSimulacionButton = uibutton(app.UIFigure, 'push');
app.EjecutarSimulacionButton.ButtonPushedFcn = createCallbackFcn(app,
@EjecutarSimulacionButtonPushed, true);
app.EjecutarSimulacionButton.BackgroundColor = [0.8 0.8 0.8];
app.EjecutarSimulacionButton.FontSize = 20;
app.EjecutarSimulacionButton.FontWeight = 'bold';
app.EjecutarSimulacionButton.FontColor = [1 1 1];
app.EjecutarSimulacionButton.Position = [150 98 208 44];
app.EjecutarSimulacionButton.Text = 'Ejecutar Simulacion';

% Create Image
app.Image = uiimage(app.UIFigure);
app.Image.ImageClickedFcn = createCallbackFcn(app, @Deteccion, true);
app.Image.Position = [18 63 114 106];
app.Image.ImageSource = 'IRB120.jpg';

% Create Image2
app.Image2 = uiimage(app.UIFigure);
app.Image2.Position = [1 462 725 92];
app.Image2.ImageSource = 'Fondo1.png';

% Create ActivarInterfazLabel
app.ActivarInterfazLabel = uilabel(app.UIFigure);
app.ActivarInterfazLabel.FontSize = 20;
app.ActivarInterfazLabel.FontWeight = 'bold';
app.ActivarInterfazLabel.FontColor = [1 1 1];
app.ActivarInterfazLabel.Position = [18 496 150 25];
app.ActivarInterfazLabel.Text = 'Activar Interfaz';

% Create EncendidoSwitch
app.EncendidoSwitch = uiswitch(app.UIFigure, 'slider');
app.EncendidoSwitch.ValueChangedFcn = createCallbackFcn(app, @EncendidoSwitchValueChanged,
true);
app.EncendidoSwitch.FontColor = [1 1 1];
app.EncendidoSwitch.Position = [212 498 45 20];

% Create Lamp
app.Lamp = uilamp(app.UIFigure);
app.Lamp.Position = [297 498 20 20];

% Create CleanAllButton
app.CleanAllButton = uibutton(app.UIFigure, 'push');
app.CleanAllButton.ButtonPushedFcn = createCallbackFcn(app, @CleanAllButtonPushed, true);
app.CleanAllButton.Icon = 'Basura.png';
app.CleanAllButton.IconAlignment = 'right';
app.CleanAllButton.BackgroundColor = [0.7294 0.8314 0.8118];
app.CleanAllButton.FontWeight = 'bold';
app.CleanAllButton.FontColor = [1 1 1];
app.CleanAllButton.Position = [606 496 103 25];
app.CleanAllButton.Text = 'Clean All';
```

```
% Create RedCross
app.RedCross = uiimage(app.UIFigure);
app.RedCross.Position = [645 140 39 23];
app.RedCross.ImageSource = 'Red Cross.PNG';

% Create RedCross_2
app.RedCross_2 = uiimage(app.UIFigure);
app.RedCross_2.Position = [644 105 39 23];
app.RedCross_2.ImageSource = 'Red Cross.PNG';

% Create GreenCheck
app.GreenCheck = uiimage(app.UIFigure);
app.GreenCheck.Position = [645 140 39 23];
app.GreenCheck.ImageSource = 'Green Check.PNG';

% Create GreenCheck_2
app.GreenCheck_2 = uiimage(app.UIFigure);
app.GreenCheck_2.Position = [645 105 39 23];
app.GreenCheck_2.ImageSource = 'Green Check.PNG';

% Create ColorDetectadoEditFieldLabel
app.ColorDetectadoEditFieldLabel = uilabel(app.UIFigure);
app.ColorDetectadoEditFieldLabel.HorizontalAlignment = 'right';
app.ColorDetectadoEditFieldLabel.FontWeight = 'bold';
app.ColorDetectadoEditFieldLabel.Position = [427 140 98 22];
app.ColorDetectadoEditFieldLabel.Text = 'Color Detectado';

% Create ColorDetectadoEditField
app.ColorDetectadoEditField = uieditfield(app.UIFigure, 'text');
app.ColorDetectadoEditField.HorizontalAlignment = 'center';
app.ColorDetectadoEditField.FontWeight = 'bold';
app.ColorDetectadoEditField.Position = [540 140 100 22];

% Create TamaoDetectadoEditFieldLabel
app.TamaoDetectadoEditFieldLabel = uilabel(app.UIFigure);
app.TamaoDetectadoEditFieldLabel.HorizontalAlignment = 'right';
app.TamaoDetectadoEditFieldLabel.FontWeight = 'bold';
app.TamaoDetectadoEditFieldLabel.Position = [413 105 112 22];
app.TamaoDetectadoEditFieldLabel.Text = 'Tamaño Detectado';

% Create SizeDetectadoEditField
app.SizeDetectadoEditField = uieditfield(app.UIFigure, 'text');
app.SizeDetectadoEditField.Editable = 'off';
app.SizeDetectadoEditField.HorizontalAlignment = 'center';
app.SizeDetectadoEditField.FontWeight = 'bold';
app.SizeDetectadoEditField.Position = [540 105 100 22];

% Create Image3
app.Image3 = uiimage(app.UIFigure);
app.Image3.Position = [1 156 725 343];
app.Image3.ImageSource = 'Fondo2.PNG';

% Create ACCIONESLabel
app.ACCIONESLabel = uilabel(app.UIFigure);
app.ACCIONESLabel.FontSize = 20;
app.ACCIONESLabel.FontWeight = 'bold';
app.ACCIONESLabel.FontColor = [1 1 1];
app.ACCIONESLabel.Position = [298 443 111 24];
app.ACCIONESLabel.Text = 'ACCIONES';

% Create ButtonGroup
app.ButtonGroup = uibuttongroup(app.UIFigure);
app.ButtonGroup.SelectionChangedFcn = createCallbackFcn(app, @ButtonGroupSelectionChanged,
true);
app.ButtonGroup.BorderType = 'none';
app.ButtonGroup.TitlePosition = 'centertop';
app.ButtonGroup.BackgroundColor = [0.349 0.349 0.349];
app.ButtonGroup.FontWeight = 'bold';
app.ButtonGroup.FontSize = 20;
app.ButtonGroup.Position = [44 340 173 93];
```

```
% Create MontajeCocheButton
app.MontajeCocheButton = uitogglebutton(app.ButtonGroup);
app.MontajeCocheButton.Text = 'MontajeCoche';
app.MontajeCocheButton.BackgroundColor = [1 1 1];
app.MontajeCocheButton.Position = [11 47 150 34];
app.MontajeCocheButton.Value = true;

% Create DeteccionFiguraButton
app.DeteccionFiguraButton = uitogglebutton(app.ButtonGroup);
app.DeteccionFiguraButton.Text = 'DeteccionFigura';
app.DeteccionFiguraButton.Position = [11 13 150 35];

% Create SeleccionadoLabel
app.SeleccionadoLabel = uilabel(app.UIFigure);
app.SeleccionadoLabel.FontWeight = 'bold';
app.SeleccionadoLabel.FontColor = [1 1 1];
app.SeleccionadoLabel.Position = [105 283 86 25];
app.SeleccionadoLabel.Text = 'Seleccionado';

% Create DeseadoLabel
app.DeseadoLabel = uilabel(app.UIFigure);
app.DeseadoLabel.FontWeight = 'bold';
app.DeseadoLabel.FontColor = [1 1 1];
app.DeseadoLabel.Position = [263 282 69 25];
app.DeseadoLabel.Text = 'Deseado';

% Create SeleccionColorDeseado
app.SeleccionColorDeseado = uidropdown(app.UIFigure);
app.SeleccionColorDeseado.Items = {'Rojo', 'Verde', 'Azul', 'Amarillo', 'Magenta', 'Cyan',
'Negro', ''};
app.SeleccionColorDeseado.ValueChangedFcn = createCallbackFcn(app,
@SeleccionColorDeseadoValueChanged, true);
app.SeleccionColorDeseado.BackgroundColor = [0.4706 0.9098 0.749];
app.SeleccionColorDeseado.Position = [251 250 81 22];
app.SeleccionColorDeseado.Value = 'Rojo';

% Create SeleccionSizeDeseado
app.SeleccionSizeDeseado = uidropdown(app.UIFigure);
app.SeleccionSizeDeseado.Items = {'Pequeño', 'Mediano', 'Grande', ''};
app.SeleccionSizeDeseado.ValueChangedFcn = createCallbackFcn(app,
@SeleccionSizeDeseadoValueChanged, true);
app.SeleccionSizeDeseado.BackgroundColor = [0.4706 0.9098 0.749];
app.SeleccionSizeDeseado.Position = [251 216 81 22];
app.SeleccionSizeDeseado.Value = 'Mediano';

% Create ColorDropDownLabel
app.ColorDropDownLabel = uilabel(app.UIFigure);
app.ColorDropDownLabel.HorizontalAlignment = 'right';
app.ColorDropDownLabel.FontWeight = 'bold';
app.ColorDropDownLabel.FontColor = [1 1 1];
app.ColorDropDownLabel.Position = [42 250 37 22];
app.ColorDropDownLabel.Text = 'Color';

% Create SeleccionColor
app.SeleccionColor = uidropdown(app.UIFigure);
app.SeleccionColor.Items = {'Rojo', 'Verde', 'Azul', 'Amarillo', 'Magenta', 'Cyan',
'Negro', ''};
app.SeleccionColor.ValueChangedFcn = createCallbackFcn(app, @SeleccionColorValueChanged,
true);
app.SeleccionColor.FontColor = [0.149 0.149 0.149];
app.SeleccionColor.BackgroundColor = [0.4706 0.9098 0.749];
app.SeleccionColor.Position = [107 250 81 22];
app.SeleccionColor.Value = 'Rojo';

% Create TamaoDropDownLabel
app.TamaoDropDownLabel = uilabel(app.UIFigure);
app.TamaoDropDownLabel.HorizontalAlignment = 'right';
app.TamaoDropDownLabel.FontWeight = 'bold';
app.TamaoDropDownLabel.FontColor = [1 1 1];
app.TamaoDropDownLabel.Position = [28 216 51 22];
```

```
app.TamaoDropDownLabel.Text = 'Tamaño';

% Create SeleccionSize
app.SeleccionSize = uidropdown(app.UIFigure);
app.SeleccionSize.Items = {'Pequeño', 'Mediano', 'Grande', ''};
app.SeleccionSize.ValueChangedFcn = createCallbackFcn(app, @SeleccionSizeValueChanged,
true);
app.SeleccionSize.FontColor = [0.149 0.149 0.149];
app.SeleccionSize.BackgroundColor = [0.4706 0.9098 0.749];
app.SeleccionSize.Position = [106 216 81 22];
app.SeleccionSize.Value = 'Mediano';

% Create SeleccinPosicionSliderLabel
app.SeleccinPosicionSliderLabel = uilabel(app.UIFigure);
app.SeleccinPosicionSliderLabel.HorizontalAlignment = 'right';
app.SeleccinPosicionSliderLabel.FontWeight = 'bold';
app.SeleccinPosicionSliderLabel.FontColor = [1 1 1];
app.SeleccinPosicionSliderLabel.Position = [373 304 115 22];
app.SeleccinPosicionSliderLabel.Text = 'Selección Posicion';

% Create PosicionMediano
app.PosicionMediano = uislider(app.UIFigure);
app.PosicionMediano.Limits = [-75 75];
app.PosicionMediano.ValueChangingFcn = createCallbackFcn(app,
@PosicionMedianoValueChanging, true);
app.PosicionMediano.FontColor = [1 1 1];
app.PosicionMediano.Position = [509 313 124 3];

% Create SeleccinPosicionSlider_3Label
app.SeleccinPosicionSlider_3Label = uilabel(app.UIFigure);
app.SeleccinPosicionSlider_3Label.HorizontalAlignment = 'right';
app.SeleccinPosicionSlider_3Label.FontWeight = 'bold';
app.SeleccinPosicionSlider_3Label.FontColor = [1 1 1];
app.SeleccinPosicionSlider_3Label.Position = [373 304 115 22];
app.SeleccinPosicionSlider_3Label.Text = 'Selección Posicion';

% Create PosicionGrande
app.PosicionGrande = uislider(app.UIFigure);
app.PosicionGrande.Limits = [-50 50];
app.PosicionGrande.ValueChangingFcn = createCallbackFcn(app, @PosicionGrandeValueChanging,
true);
app.PosicionGrande.FontColor = [1 1 1];
app.PosicionGrande.Position = [509 313 124 3];

% Create ValorPosicion
app.ValorPosicion = uieditfield(app.UIFigure, 'numeric');
app.ValorPosicion.Position = [661 304 48 22];

% Create SeleccinPosicionSlider_2Label
app.SeleccinPosicionSlider_2Label = uilabel(app.UIFigure);
app.SeleccinPosicionSlider_2Label.HorizontalAlignment = 'right';
app.SeleccinPosicionSlider_2Label.FontWeight = 'bold';
app.SeleccinPosicionSlider_2Label.FontColor = [1 1 1];
app.SeleccinPosicionSlider_2Label.Position = [373 304 115 22];
app.SeleccinPosicionSlider_2Label.Text = 'Selección Posicion';

% Create PosicionPequeno
app.PosicionPequeno = uislider(app.UIFigure);
app.PosicionPequeno.Limits = [-100 100];
app.PosicionPequeno.ValueChangingFcn = createCallbackFcn(app,
@PosicionPequenoValueChanging, true);
app.PosicionPequeno.FontColor = [1 1 1];
app.PosicionPequeno.Position = [509 313 124 3];

% Create ConfigurarEspacioTrabajoVisinArtificialButton
app.ConfigurarEspacioTrabajoVisinArtificialButton = uibutton(app.UIFigure, 'state');
app.ConfigurarEspacioTrabajoVisinArtificialButton.ValueChangedFcn = createCallbackFcn(app,
@ConfigurarEspacioTrabajoVisinArtificialButtonValueChanged, true);
app.ConfigurarEspacioTrabajoVisinArtificialButton.Text = 'Configurar Espacio Trabajo
Visión Artificial';
app.ConfigurarEspacioTrabajoVisinArtificialButton.Position = [373 237 336 22];
```

```
% Create CoordenadasEditFieldLabel
app.CoordenadasEditFieldLabel = uilabel(app.UIFigure);
app.CoordenadasEditFieldLabel.HorizontalAlignment = 'right';
app.CoordenadasEditFieldLabel.FontColor = [1 1 1];
app.CoordenadasEditFieldLabel.Position = [373 199 78 22];
app.CoordenadasEditFieldLabel.Text = 'Coordenadas';

% Create CoordenadasEditField
app.CoordenadasEditField = uieditfield(app.UIFigure, 'text');
app.CoordenadasEditField.ValueChangedFcn = createCallbackFcn(app,
@CoordenadasEditFieldValueChanged, true);
app.CoordenadasEditField.FontColor = [1 1 1];
app.CoordenadasEditField.Position = [466 199 240 22];

% Create AutorRodrigoSanchoGarcaLabel
app.AutorRodrigoSanchoGarcaLabel = uilabel(app.UIFigure);
app.AutorRodrigoSanchoGarcaLabel.FontName = 'Times New Roman';
app.AutorRodrigoSanchoGarcaLabel.FontWeight = 'bold';
app.AutorRodrigoSanchoGarcaLabel.Position = [265 39 164 22];
app.AutorRodrigoSanchoGarcaLabel.Text = 'Autor: Rodrigo Sancho García';

% Create Image4
app.Image4 = uiimage(app.UIFigure);
app.Image4.Position = [14 10 74 52];
app.Image4.ImageSource = 'LogoEII.png';

% Create Image5
app.Image5 = uiimage(app.UIFigure);
app.Image5.Position = [638 -3 83 73];
app.Image5.ImageSource = 'LogoUVa.png';

% Create TITULOTFGBLABLABLALabel
app.TITULOTFGBLABLABLALabel = uilabel(app.UIFigure);
app.TITULOTFGBLABLABLALabel.HorizontalAlignment = 'center';
app.TITULOTFGBLABLABLALabel.FontName = 'Times New Roman';
app.TITULOTFGBLABLABLALabel.FontSize = 10;
app.TITULOTFGBLABLABLALabel.FontWeight = 'bold';
app.TITULOTFGBLABLABLALabel.Position = [102 10 519 26];
app.TITULOTFGBLABLABLALabel.Text = {'Diseño con Autodesk Inventor de una célula de trabajo
robótica para realizar montajes multitarea '; 'apoyados en visión artificial, coordinado
mediante protocolo de comunicación OPC UA entre RobotStudio y MATLAB'};

% Create Coche
app.Coche = uiimage(app.UIFigure);
app.Coche.Position = [429 355 86 63];
app.Coche.ImageSource = 'Coche.png';

% Create Estacion
app.Estacion = uiimage(app.UIFigure);
app.Estacion.Position = [275 333 97 105];
app.Estacion.ImageSource = 'Estacion.png';

% Create Cajas
app.Cajas = uiimage(app.UIFigure);
app.Cajas.Position = [568 326 67 120];
app.Cajas.ImageSource = 'Cajas.png';

            % Create CajasSeleccionado
            app.CajasSeleccionado = uiimage(app.UIFigure);
            app.CajasSeleccionado.Position = [564 345 75 81];
            app.CajasSeleccionado.ImageSource = 'Cajas1.png';

            % Create CocheSeleccionado
            app.CocheSeleccionado = uiimage(app.UIFigure);
            app.CocheSeleccionado.Position = [434 339 77 95];
            app.CocheSeleccionado.ImageSource = 'Coche1.png';

            % Create PosicionDetectadaEditFieldLabel
            app.PosicionDetectadaEditFieldLabel = uilabel(app.UIFigure);
            app.PosicionDetectadaEditFieldLabel.HorizontalAlignment = 'right';
```

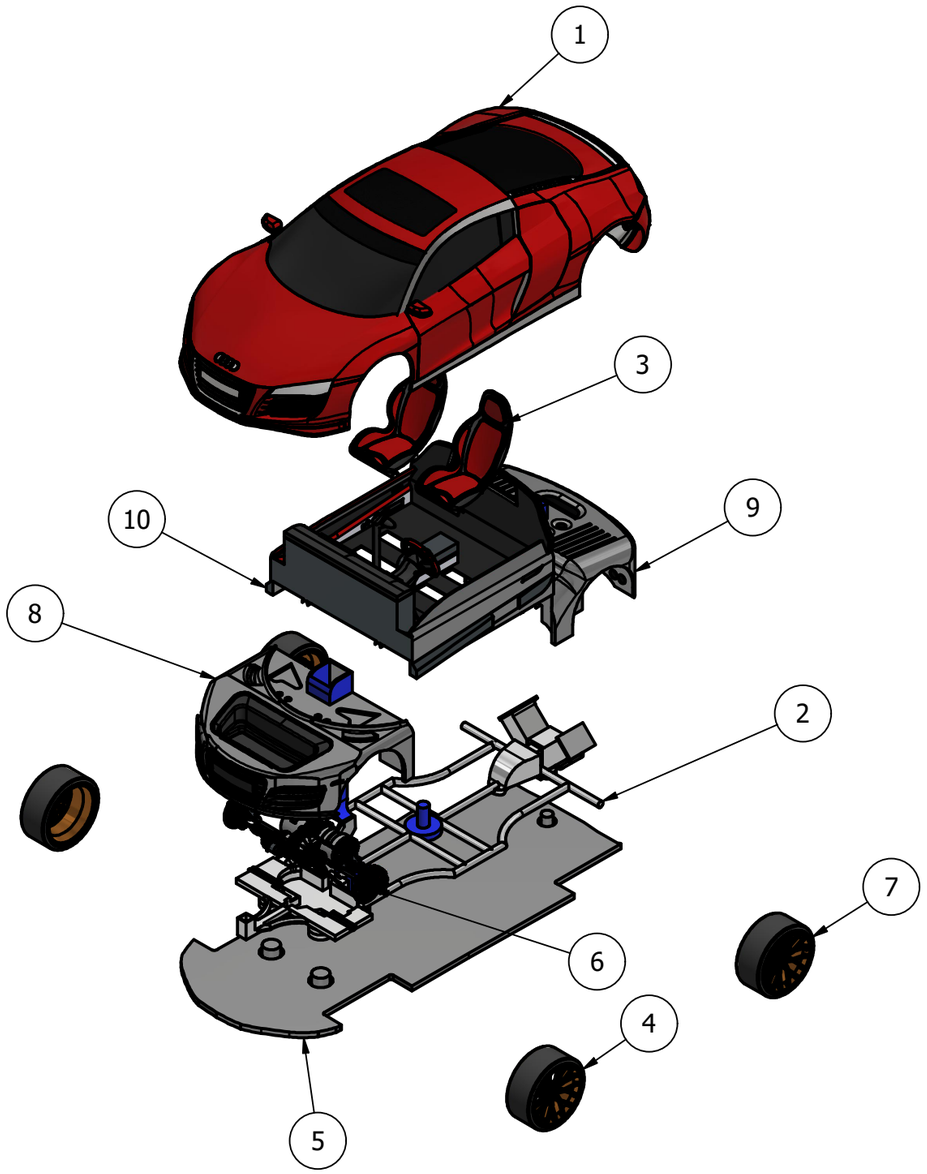
```
            app.PosicionDetectadaEditFieldLabel.FontWeight = 'bold';
            app.PosicionDetectadaEditFieldLabel.Position = [408 70 117 22];
            app.PosicionDetectadaEditFieldLabel.Text = 'Posicion Detectada';

            % Create PosicionDetectadaEditField
            app.PosicionDetectadaEditField = uieditfield(app.UIFigure, 'numeric');
            app.PosicionDetectadaEditField.HorizontalAlignment = 'center';
            app.PosicionDetectadaEditField.FontWeight = 'bold';
            app.PosicionDetectadaEditField.Position = [540 70 100 22];

            % Create ImagenesVisinArtificialButton
            app.ImagenesVisinArtificialButton = uibutton(app.UIFigure, 'push');
            app.ImagenesVisinArtificialButton.ButtonPushedFcn = createCallbackFcn(app,
@ImagenesVisinArtificialButtonPushed, true);
            app.ImagenesVisinArtificialButton.Icon = 'Emoji Camara.png';
            app.ImagenesVisinArtificialButton.IconAlignment = 'right';
            app.ImagenesVisinArtificialButton.BackgroundColor = [0.7294 0.8314 0.8118];
            app.ImagenesVisinArtificialButton.FontWeight = 'bold';
            app.ImagenesVisinArtificialButton.FontColor = [1 1 1];
            app.ImagenesVisinArtificialButton.Position = [370 495 199 26];
            app.ImagenesVisinArtificialButton.Text = 'Imagenes Visión Artificial';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = InterfazTFG

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

ANEXO B. PLANOS

En las siguientes hojas el lector puede encontrar algunos de los planos de las piezas diseñadas. Al igual que en el código no se muestran absolutamente todas las piezas, pero sí que se ha querido reflejar el despiece de algunos de los ensamblajes y las cotas más características de diferentes piezas diseñadas para la creación de la célula de trabajo.

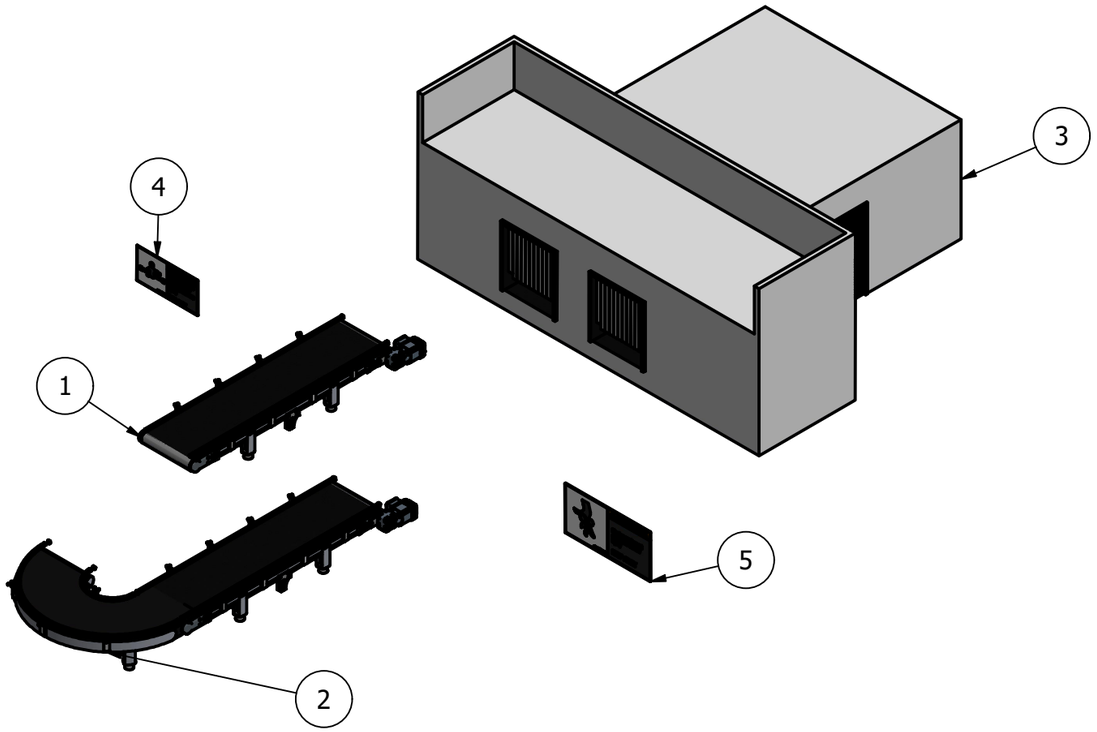
A continuación, se muestra el índice de los diferentes planos.

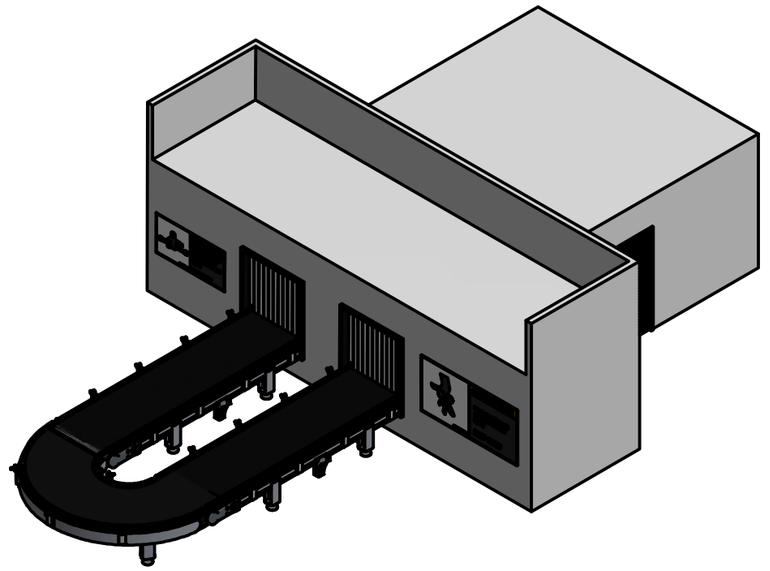
ÍNDICE



LISTA DE PIEZAS			
ELEMENTO	CTDAD	Nº DE PIEZA	DESCRIPCIÓN
1	1	Carroceria AUDI R8	
2	1	Chasis	
3	2	Asientos Coche	
4	2	Ruedas Delanteras	
5	1	Parte Inferior	
6	1	Transmision	
7	2	Ruedas Traseras	
8	1	Capo	
9	1	Parte Trasera	
10	1	Interior Coche	

DRAWN Rodrigo Sancho García	11/06/2021	Universidad de Valladolid
CHECKED		TITLE: Vista Explosionada del Automóvil
QA		
MFG		
APPROVED		
SIZE A3		DWG NO (1) Automóvil de juguete — REV
SCALE 1 / 50		SHEET 1 OF 1



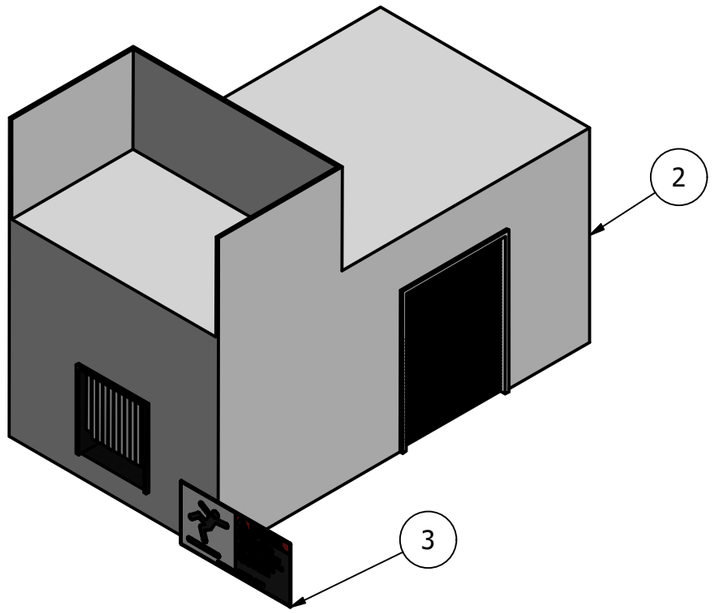


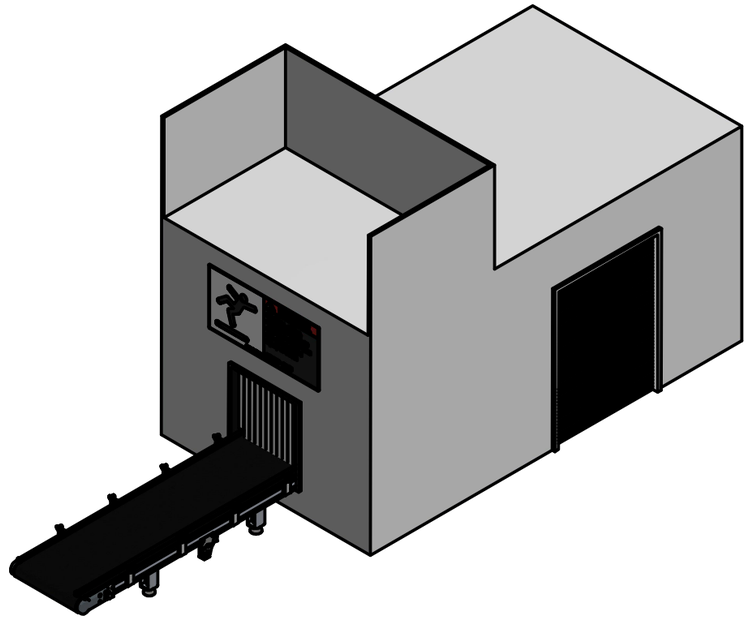
Nota:

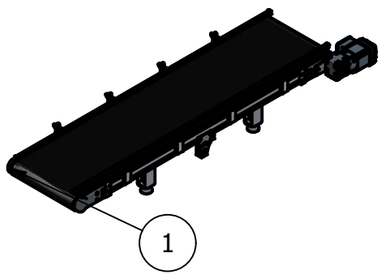
Este listado esta simplificado, ya que no se tienen en cuenta los cientos de piezas que conforman las cintas transportadoras.

LISTA DE PIEZAS			
ELEMENTO	CTDAD	Nº DE PIEZA	DESCRIPCIÓN
2	1	Cinta Transportadora Recta	
2	1	Cinta Transportadora Curva	
3	1	Edificio	
4	1	Cartel1	
5	1	Cartel2	

DRAWN Rodrigo Sancho García	11/06/2021	Universidad de Valladolid
CHECKED		TITLE: Vista Explosionada del Conveyor Doble
QA		
MFG		
APPROVED		
		SIZE A3 · DWG NO (2) Conveyor Doble · REV
		SCALE 1 / 45 · SHEET 1 OF 1





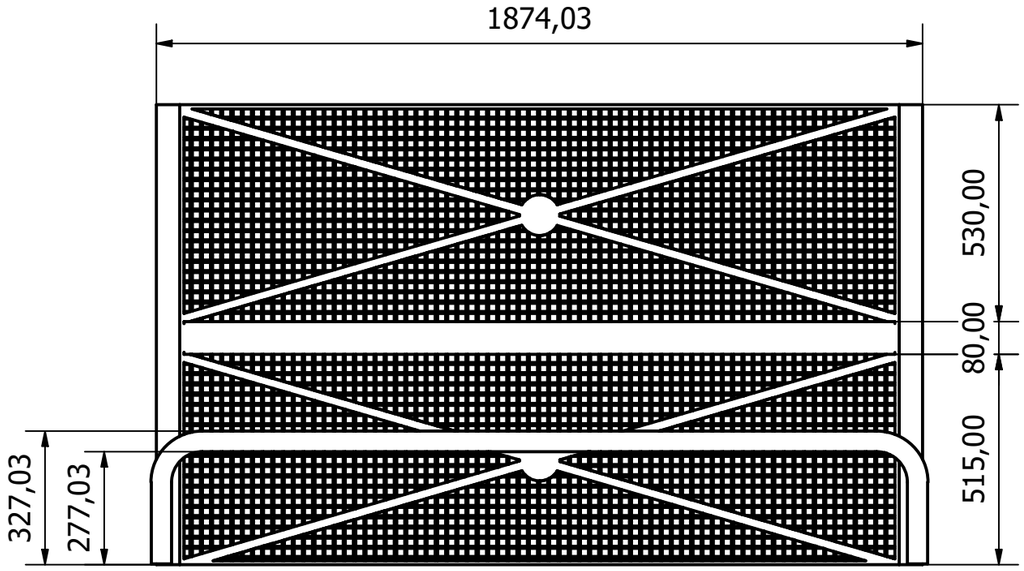


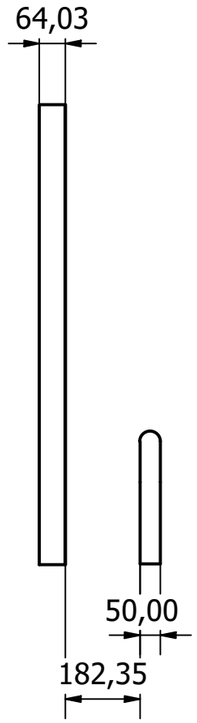
Nota:

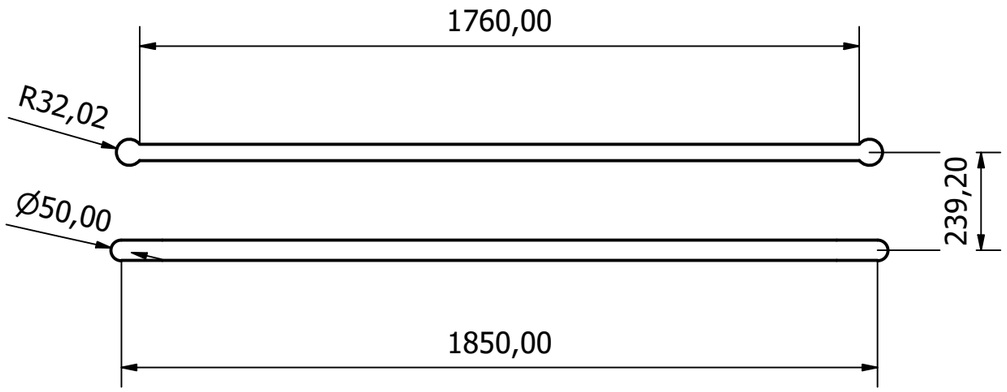
Este listado esta simplificado, ya que no se tienen en cuenta los cientos de piezas que conforman la cinta transportadora.

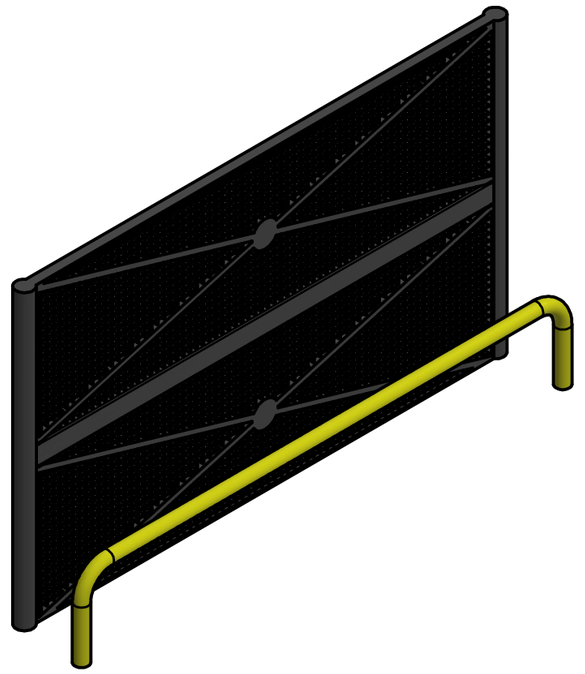
LISTA DE PIEZAS			
ELEMENTO	CTDAD	Nº DE PIEZA	DESCRIPCIÓN
1	1	Cinta Transportadora Recta	
2	1	Edificio	
3	1	Cartel	

DRAWN Rodrigo Sancho García	11/06/2021	Universidad de Valladolid
CHECKED		TITLE: Vista Explosionada del Conveyor Simple
QA		
MFG		
APPROVED		
		SIZE A3 | DWG NO (3) Conveyor Simple | REV
		SCALE 1 /35 | SHEET 1 OF 1



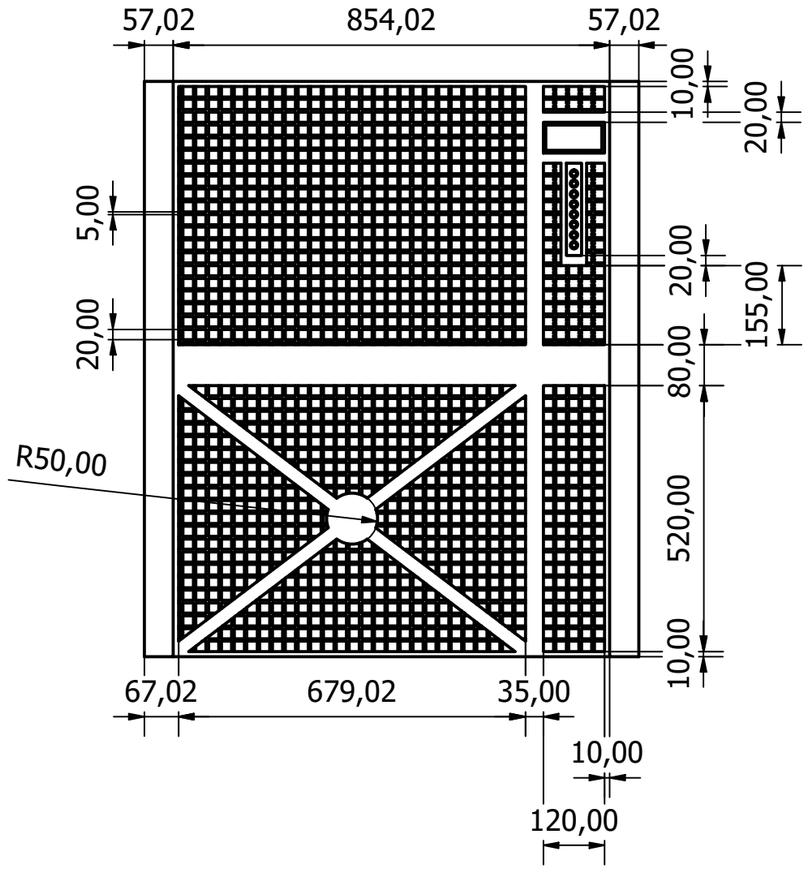


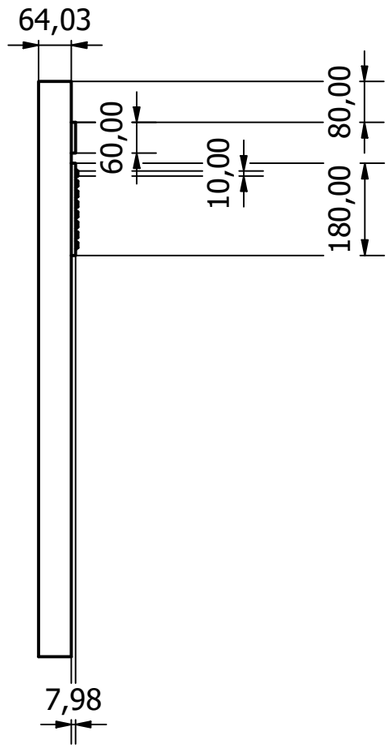


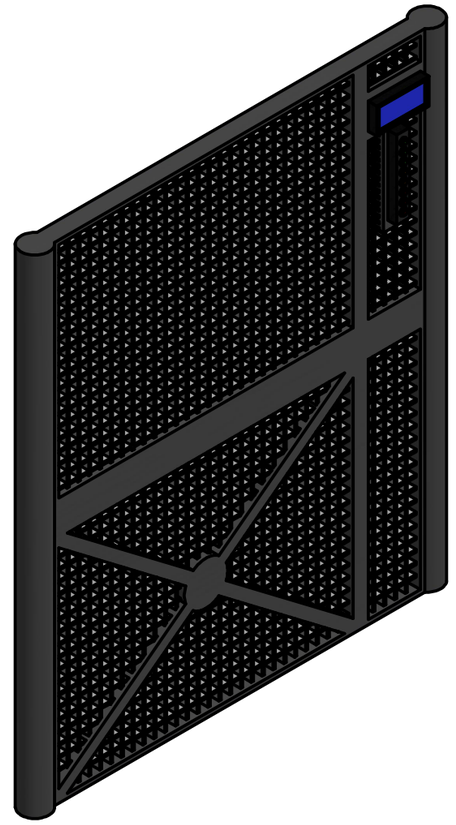


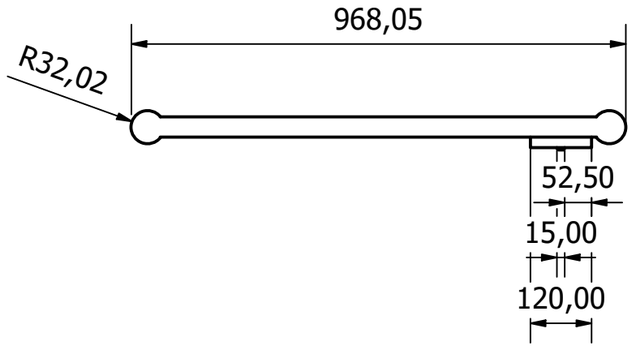
Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Rodrigo Sancho					11/06/2021	

Universidad de Valladolid	Modelo de Valla		
	(4) Valla	Edición	Hoja 1 / 1



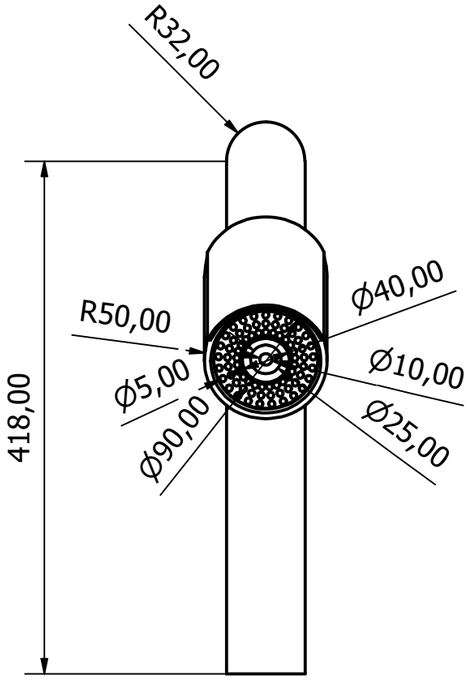


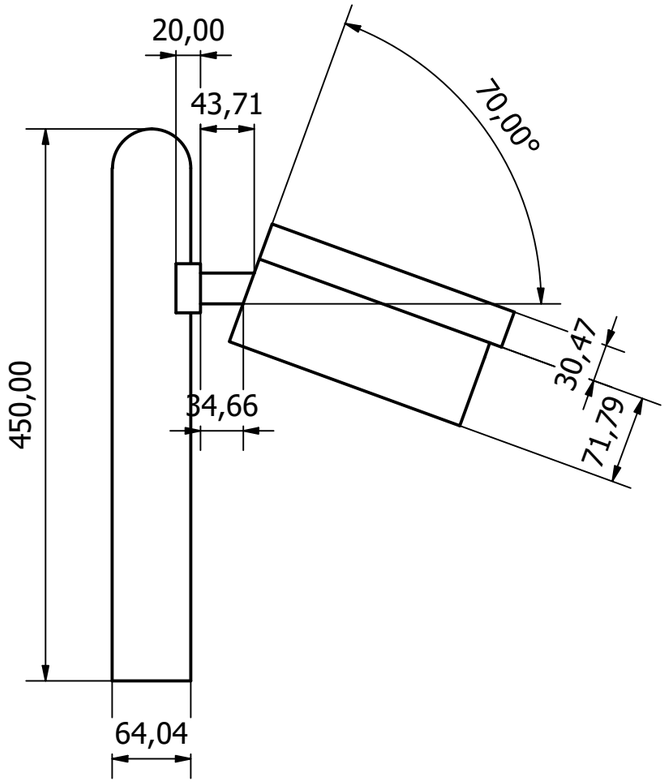


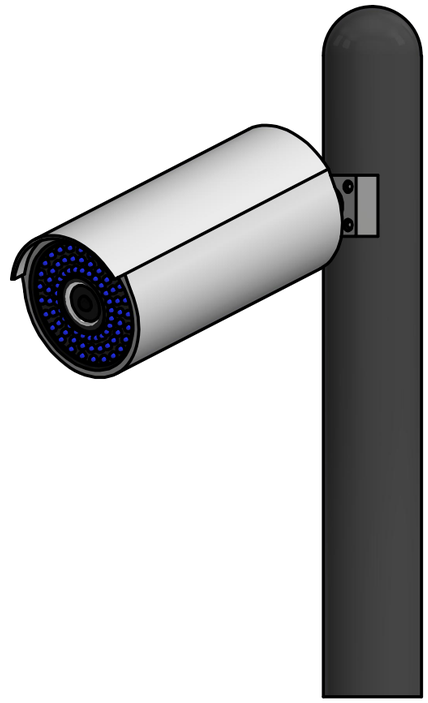


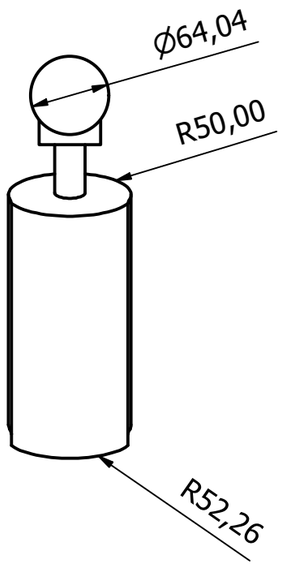
Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Rodrigo Sancho					11/06/2021	

Universidad de Valladolid	Puerta Acceso		
	(5) Puerta de Acceso	Edición	Hoja 1 / 1



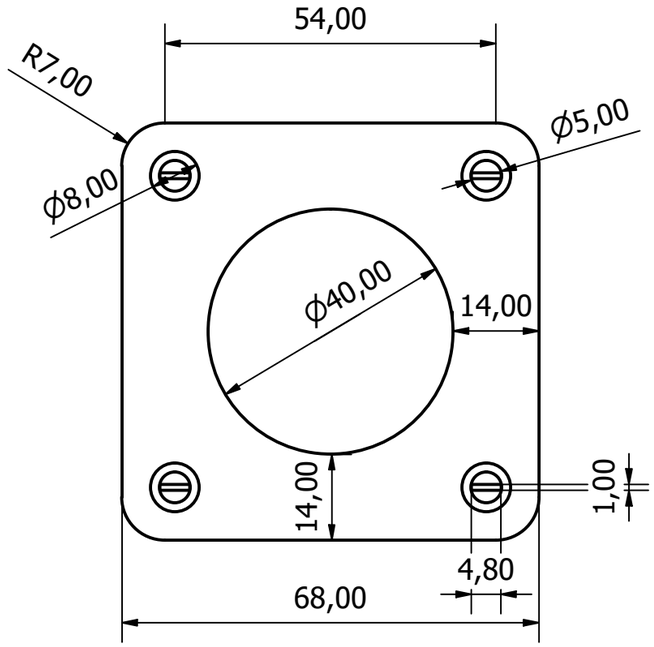


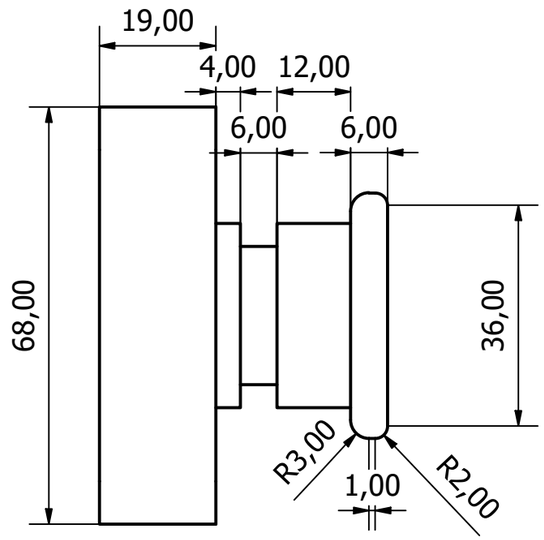


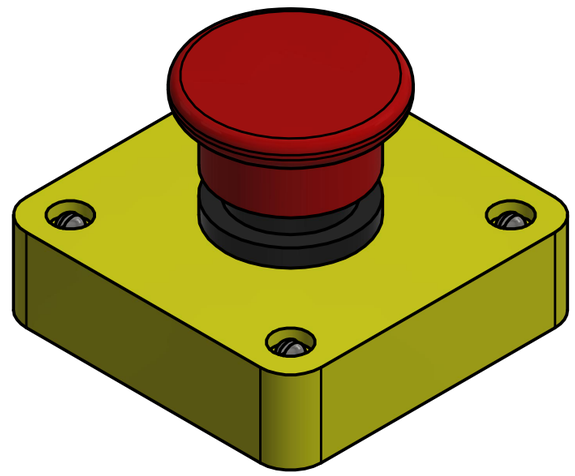


Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Rodrigo Sancho					11/06/2021	

Universidad de Valladolid	Camara de Vigilancia		
	(6) Camara de Vigilancia	Edición	Hoja 1 / 1

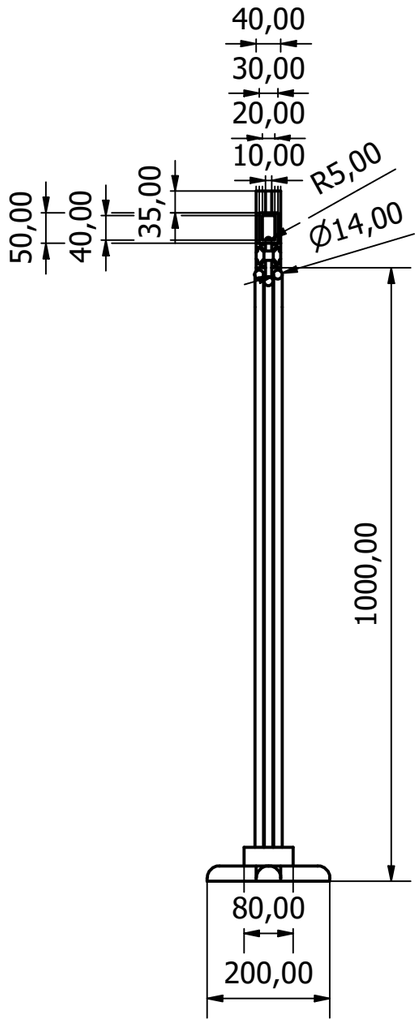


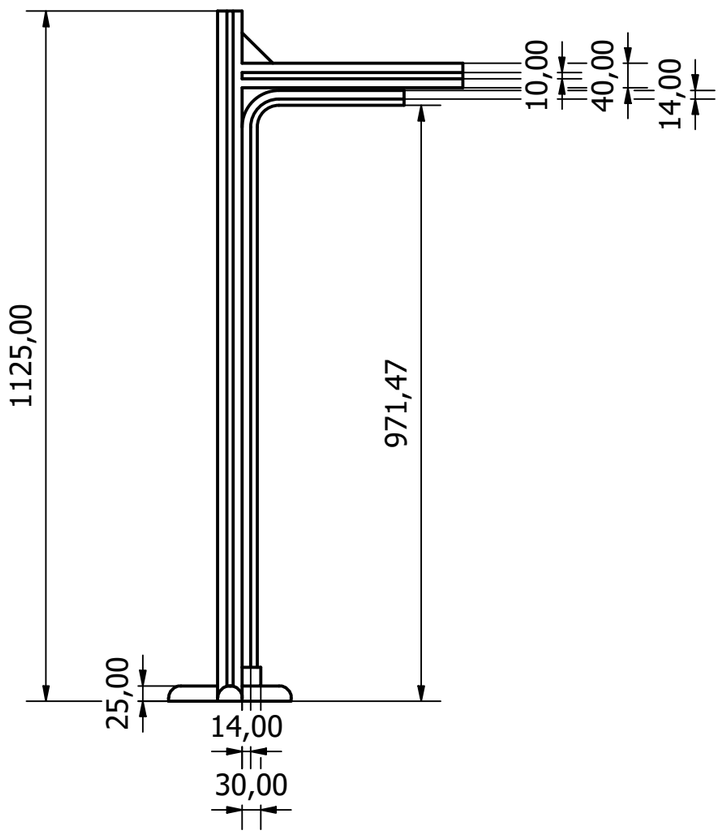


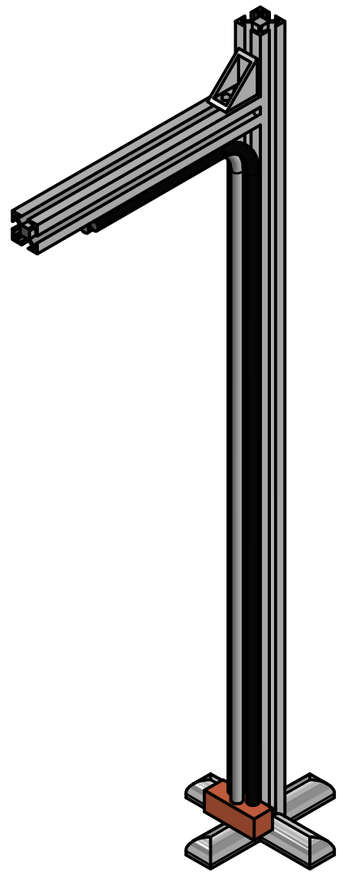


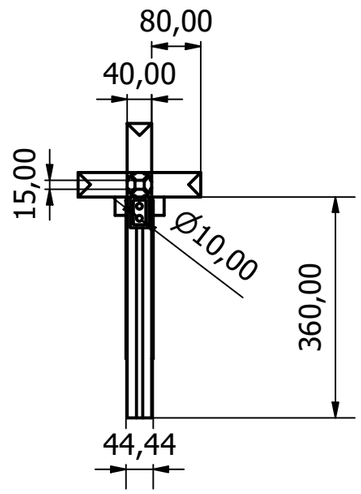
Diseño de	Revisado por	Aprobado por	Fecha	Fecha	
Rodrigo Sancho				11/06/2021	

Universidad de Valladolid	Seta de Emergencia		
	(7) Seta de Emergencia	Edición	Hoja 1 / 1

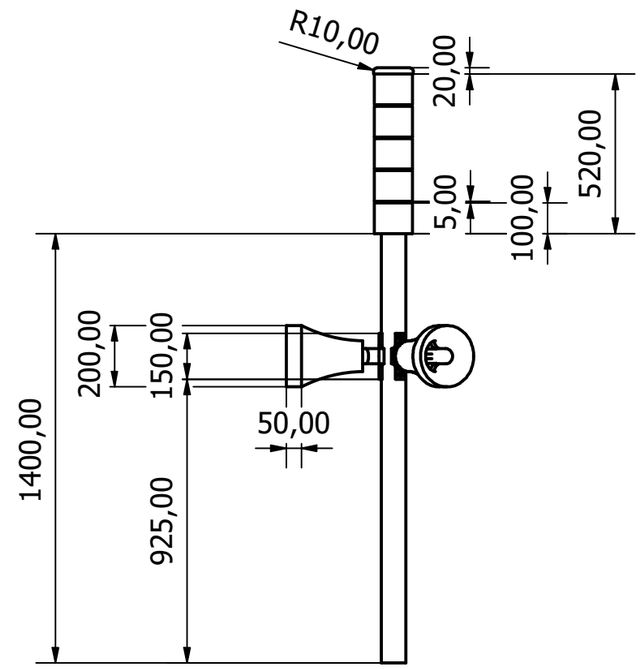


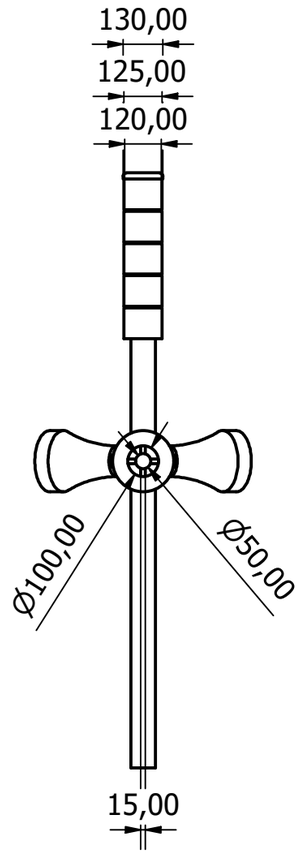


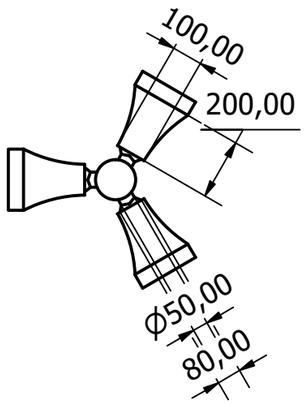


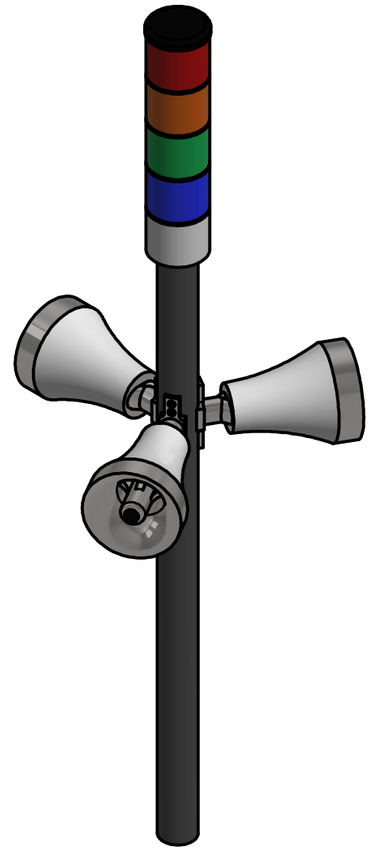


Diseño de: Rodrigo Sancho	Revisado por	Aprobado por	Fecha	Fecha: 11/06/2021	
Universidad de Valladolid	Soporte Cámara Visión Artificial				
	(8) Soporte Cámara		Edición	Hoja: 1 / 1	









Diseño de	Revisado por	Aprobado por	Fecha	Fecha	
Rodrigo Sancho				11/06/2021	

Universidad de Valladolid	Altavoces y Alarma Acústica		
	(9) Altavoz y Alarma	Edición	Hoja 1 / 1