Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática.

Célula multi-robot para el montaje de una luminaria, basado en robots ABB controlados mediante un interface de Matlab, empleando el protocolo de comunicaciones OPC.

Autor:

Vidal del Cura, Raúl

Tutor:

Herreros López, Alberto.
Departamento de Ingeniería de Sistemas y Automática

Valladolid, junio, 2022.

Universidad de Valladolid

Trabajo de Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

ESCUELA DE INGENIERÍAS INDUSTRIALES

Autor: Raúl Vidal del Cura

Fecha: 2022

Agradecimientos

Primero de todo, agradecer a mis padres y a mi hermana el apoyo incondicional y la confianza recibidos durante esta y todas las etapas de mi vida, sin vosotros nada de esto habría sido posible.

También dar las gracias al resto de mi familia, a los que están y a los que se fueron, por todo el cariño y por haberme acompañado en esta etapa tan importante.

A todos los amigos que ha dado esta etapa académica, gracias por la ayuda y apoyo siempre en los momentos más difíciles y por ser los responsables de los buenos.

Por último, agradecer a mi tutor, Alberto Herreros López, por acompañarme y guiarme en este camino. La ayuda y dedicación brindadas durante fueron fundamentales para la realización de este proyecto.

Universidad de Valladolid

Trabajo de Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

ESCUELA DE INGENIERÍAS INDUSTRIALES

Autor: Raúl Vidal del Cura

Fecha: 2022

Autor: Raúl Vidal del Cura

Fecha: 2022

Resumen

En este Trabajo de Fin de Grado se plantea la creación de una célula robótica basada en una línea de montaje, compuesta por cuatro robots de la corporación multinacional tecnológica ABB que, sincronizados, permiten simular la creación de una luminaria.

La programación de estos robots se ha realizado con el *software* de ABB, Robotstudio. También se ha realizado una interfaz en Appdesigner de Matlab que permite la comunicación con Robotstudio, a través del protocolo de comunicaciones OPC, con la cual se controla el estado de los robots y el inicio y fin de la simulación.

Para el diseño de la mayoría de los componentes de la célula y de la propia luminaria, se ha utilizado Autodesk Inventor, que es un paquete de modelado paramétrico de sólidos en 3D creado por la empresa de *software* AutoDesk.

Palabras clave

- Robotstudio.
- Célula robótica.
- Proceso industrial.
- Matlab.
- Simulación.

Universidad de Valladolid

Trabajo de Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

ESCUELA DE INGENIERÍAS INDUSTRIALES

Autor: Raúl Vidal del Cura

Fecha: 2022

Autor: Raúl Vidal del Cura

Fecha: 2022

Abstract

In this End Degree Project, the creation of a robotic cell based on an assembly line is proposed, made up of four robots from the multinational technology corporation ABB that, synchronized, allow the creation of a luminaire to be simulated.

The programming of these robots has been done with the ABB *software*, Robotstudio. An interface has also been made in Matlab Appdesigner that allows communication with Robotstudio, through the OPC communications protocol, with which the state of the robots and the start and end of the simulation are controlled.

For the design of most of the components of the cell and of the luminaire itself, Autodesk Inventor has been used, which is a 3D parametric solid modeling package created by the *software* company AutoDesk.

Keywords

- Robotstudio.
- Robotic cell.
- Industrial process.
- Matlab.
- Simulation.

Universidad de Valladolid

Trabajo de Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

ESCUELA DE INGENIERÍAS INDUSTRIALES

Autor: Raúl Vidal del Cura

Fecha: 2022

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Autor: Raúl Vidal del Cura

Fecha: 2022

1. Introducción

El proyecto desarrolla la creación de una célula robótica y su simulación mediante el *software* de ABB, Robotstudio, el cual nos permite representar el comportamiento del robot para posteriormente poder trasladarlo a la realidad. Esto nos da la posibilidad de programar robots, diseñar el entorno y comprobar distintas posibilidades hasta hallar la opción que se ajuste a ciertas especificaciones de la forma más segura posible.

Para realizar correctamente un proceso de montaje es muy importante la coordinación de los diferentes pasos, esto se destacará en este proyecto donde se trabaja en la sincronización entre los movimientos de cuatro robots de ABB.

1.1 Objetivos

El objetivo principal del proyecto es ampliar los conocimientos en el *software* Robotstudio y su programación en RAPID *(Robotics Application Programming Interactive Dialogue*), que es un lenguaje de programación desarrollado por ABB, mediante la simulación del montaje en la que participan tres brazos robóticos y un robot paralelo, cada uno realizando una tarea diferente.

Para modelar correctamente el entorno de simulación se deberá diseñar en 3D diferentes componentes, así como la plataforma donde van a trabajar los robots, las herramientas usadas para el montaje, la luminaria, etc. Todos estos componentes serán mostrados con detalle más adelante. Una vez diseñados todos los componentes de la célula se deberá realizar la programación de cada robot.

Otro de los objetivos ha sido la aplicación de conocimientos adquiridos en el grado de Electrónica Industrial y Automática de la Universidad de Valladolid y la ampliación de estos sobre todo en el ámbito de la robótica en un entorno industrial.

Por último, se deseaba poder establecer comunicación mediante una interfaz usuario en Matlab con el objetivo de poder controlar la simulación a través de esta.

1.2 Motivación

Las amplias ramas que abarca la robótica en el mundo de la ingeniería, así como la mecánica, física, informática, control, etc., que son conocimientos adquiridos en el grado cursado, se quería demostrar el control que se tiene sobre ellos.

También se pretendía dar visibilidad a la diversa variedad de campos de aplicación de la robótica, centrando el proyecto en un entorno industrial.

Todo lo mencionado anteriormente unido al gusto por la robótica y la automatización de procesos industriales fueron factores que llevaron a desarrollar este TFG.

1.3 Estructura de la memoria

En esta memoria del TFG en primer lugar se aborda una breve introducción al mundo de la robótica. Después, se realizará un estudio del conocimiento acumulado dentro de esta área, es decir, se mostrarán los diferentes proyectos relacionados con la robótica que han sido realizados en la Universidad de Valladolid. Posteriormente, se realizará una discusión comentando las ventajas y limitaciones de la robótica en la industria y de cada uno de los proyectos para finalmente extraer una conclusión.

A continuación, se explicará el proyecto que se ha realizado y los diferentes *softwares* que han sido necesarios para poder implementarlo. Después, se explicará cómo se ha realizado el diseño de la célula y de todos los componentes que la componen. También se realizará una breve explicación sobre los robots que han sido utilizados y sus características más importantes.

Posteriormente, se mostrará cómo se ha realizado la programación de los robots, la creación de la interfaz Usuario-Máquina y la comunicación entre programas.

Para finalizar esta memoria se mostrará el funcionamiento de la aplicación y las conclusiones obtenidas del proyecto.

En la parte final de la memoria se encuentran la bibliografía y los anexos en los cuales se muestra parte del código utilizado tanto en Robotstudio como en Matlab y los planos de los componentes diseñados.

2. Introducción al mundo de la robótica

Son varias las ideas procedentes del desarrollo tecnológico que han logrado sobrepasar las barreras fijadas por el sector industrial, entre ellas cabe destacar el concepto de robot. [1]

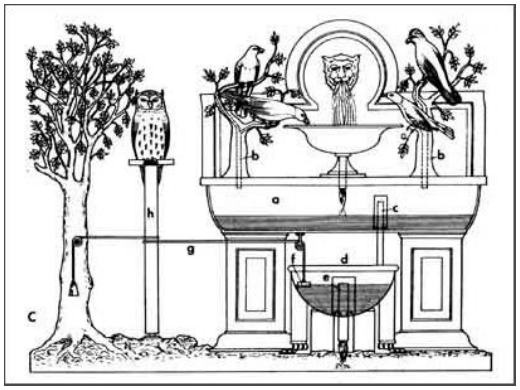
La robótica es un campo para la investigación, innovación y desarrollo de aplicaciones. Su empleo en la industria es cada vez más masivo y su uso en tareas de servicio y asistencia provoca la ampliación de los campos de aplicación, que pueden extenderse desde la rehabilitación para personas que posean una minusvalía hasta sofisticadas exploraciones espaciales. [2]

2.1 Origen de la robótica

La palabra robot fue usada por primera vez en el año 1921 cuando el escritor de origen checo, Karel Capek estrena su obra *Rossum's Universal Robot* (R.U.R). Su origen viene de la palabra eslava '*Robota*', cuyo significado es trabajo forzado. En la obra R.U.R. los robots eran máquinas androides fabricadas por un científico llamado Rossum, que servían a sus jefes humanos desarrollando diferentes trabajos físicos. [1]

Pero el origen de la robótica no empezó aquí, para ello debemos retroceder varios miles de años, concretamente a finales del siglo I (85 d.C), donde en la Antigua Grecia creaban diferentes máquinas y dispositivos capaces de imitar las funciones y movimientos de los seres vivos. A estas máquinas se las denominaba *automatos,* de la que deriva la palabra actual autómata. [3]

Herón de Alejandría (10d.C. - 70 d. C.) fue un ingeniero y matemático helenístico y el creador de uno de los primeros autómatas, los cuales describe en su libro "Autómata" en el año 62 d.C. Uno de los aparatos que lo componen es *Los pájaros de Herón,* que son aves con la capacidad de volar, gorjear y beber, cuyo movimiento se realizaba mediante dispositivos hidráulicos, palancas y poleas con un único fin de carácter lúdico. [2]



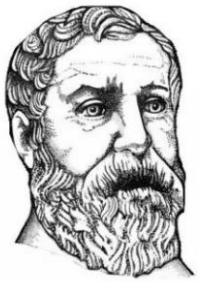


Ilustración 1. Los pájaros de Herón y su creador, Herón de Alejandría. [2]

La cultura árabe desde el siglo VIII hasta el XV heredó los conocimientos griegos y los empleó para darles una aplicación práctica, introduciéndolos en la vida cotidiana, sobre todo de la realeza. Un ejemplo de estos es la creación de diversos dispensadores automáticos de agua utilizados para beber o lavarse. Durante esta época también se desarrollaron otros autómatas como el *Hombre de Hierro* de Alberto Magno (1204-1282) o el *Gallo de Estrasburgo* (1352), el autómata más antiguo que aún se conserva en la actualidad cuyo autor es desconocido. Este formaba parte del reloj de la Catedral de Estrasburgo y su funcionamiento consistía en mover las alas y el pico para dar las horas. [1]

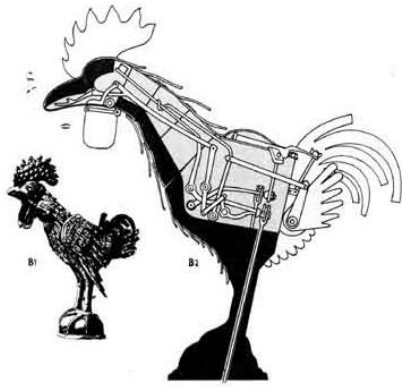


Ilustración 2. Gallo de Estrasburgo. [4]

A continuación, se muestra en la siguiente tabla una recopilación de los autómatas más importantes de los siglos XV al XVII.

Año	Autor	Autómata
1500	Leonardo Da Vinci	León mecánico
1738	Jacques Vaucanson	Flautista y Pato
1769	W.Von Kempelem	Jugador ajedrez
1770	Familia Droz	Muñeco capaz de escribir, dibujar y tocar melodías en un órgano
1805	Henri Maillardet	Muñeca mecánica capaz de dibujar

Tabla 1. Autómatas famosos. [3]

Ilustración 3. Pato de Vaucanson [5] y Muñeca de Henri Maillardet [6].

A partir del siglo XVIII, se comienza a emplear dispositivos automáticos en la producción, iniciándose la Automatización Industrial. [1]

En el año 1945 Isaac Asimov (1920-1992), considerado el mayor impulsor de la palabra robot, publicó en la revista *Galaxy Science Fiction* una novela de ciencia ficción en la que enunciaba las "Tres Leyes de la Robótica" [1]:

1. Un robot no hará daño a un ser humano ni, por inacción, permitirá que un ser humano sufra daño.
2. Un robot debe cumplir las órdenes dadas por los seres humanos, a excepción de aquellas que entren en conflicto con la primera ley.
3. 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.

Estas leyes, a pesar de su origen ficticio, en la actualidad son tenidas en cuenta en toda la comunidad científica dedicada a la robótica. [1]

2.2 Desarrollo de la robótica industrial

Una vez definidos los primeros autómatas en el apartado anterior, nos centraremos en el desarrollo de la robótica industrial.

El origen de esta se podría ubicar en el año 1948, cuando Raymond C. Goertz creó los primeros robots telemanipuladores en el *Argonne Nacional Laboratory*. El objetivo de estos era poder manipular elementos radiactivos evitando el riesgo para el operador. Consistía en un dispositivo mecánico maestro-esclavo, el manipulador maestro se encontraba en la zona segura y era movido por el operador, estos movimientos se transmitían mecánicamente al esclavo y este, que se encontraba en la zona radiactiva, los reproducía. El operador observaba el proceso a través de un grueso cristal y podía sentir las fuerzas que el esclavo aplicaba sobre el entorno. [3]

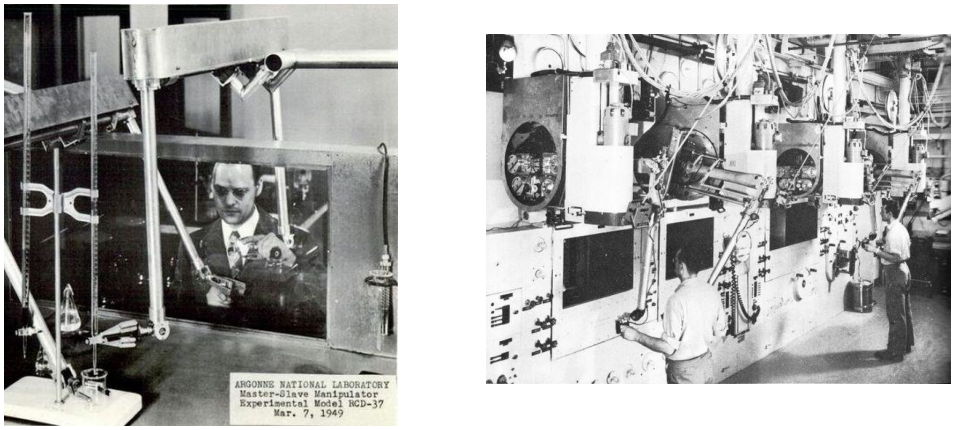


Ilustración 4. Primeros robots telemanipuladores. [7]

En 1954 Goertz, haciendo uso de la tecnología electrónica y el servocontrol, consiguió sustituir la transmisión mecánica por una eléctrica, creando el primer telemanipulador con servocontrol bilateral. [1]

Ralph Mosher, ingeniero de la empresa americana General Electric, desarrolló en 1958 un dispositivo formado por dos brazos mecánicos teleoperados por un maestro denominado exoesqueleto. Este dispositivo recibió el nombre de Handy-Man. [3]

La aplicación y evolución de los telemanipuladores quedó fijada a la industria nuclear, militar, espacial, etc., sin poder expandirse debido a que era necesario el mando continuo de un operador. Esto provocó la necesidad de sustituir el operador por un programa de ordenador que controlase los movimientos del manipulador, lo que dio paso al concepto de robot. [1]

La primera patente de un dispositivo robótico fue solicitada en 1954 por el británico C.E. Kenward. Sin embargo, fue el ingeniero norteamericano George C. Devol el que inventó y registró varias patentes, estableciendo las bases de los robots industriales. [1]

En 1956, Devol junto a Joseph F. Engelberger fundaron la *Consolidated Controls Corporation*, donde comenzaron a introducir sus máquinas en entornos industriales. Más tarde esta empresa se convertiría en *Unimation.* En 1960, se instala la primera máquina *Unimate* en la fábrica de *General Motors,* ubicada en Nueva Jersey para aplicaciones de fundición por inyección. [1]

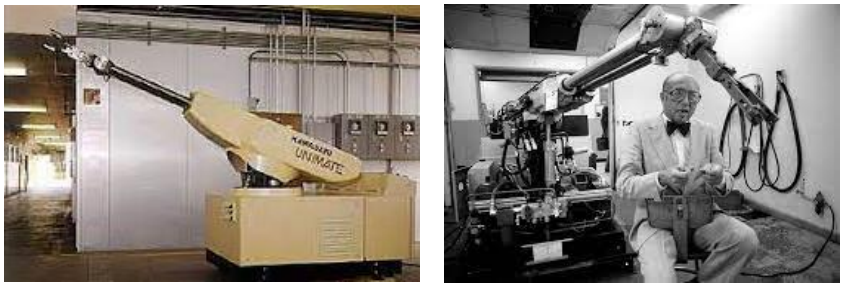


Ilustración 5. A la izquierda, el primer robot industrial [8]. A la derecha, Joseph F. Engelberger junto a uno de sus robots [9].

Años más tarde, concretamente en 1973, la compañía sueca ASEA desarrolló en Europa el primer robot con accionamiento totalmente eléctrico, el robot IRb6. En 1982 Makino, que era un profesor de la Universidad de Yamanashi de Japón, buscaba crear un robot simple, con 3 o 4 de grados de libertad y una configuración orientada al ensamblado de piezas, surgiendo el concepto de Robot SCARA (*Selective Compilant Assembly Robot Arm*). A partir de este momento surge un número interminable de proyectos que mejoran a los anteriores, llegando hasta la actualidad, momento en el que la robótica está completamente integrada en la industria y nos ofrece una elevada precisión y fiabilidad en tareas que antes era impensable. [1]

2.3 Definición y clasificación de los robots

Una vez visto el desarrollo y el origen en los apartados anteriores se procede a definir lo que es un robot y a realizar una clasificación de estos.

2.3.1 Definición

Encontramos varias definiciones sobre lo que es un robot. A continuación, se muestran las más representativas:

El Robot *Institute of America* (RIA) nos lo define de la siguiente manera*: "Un robot programable es un manipulador multifuncional reprogramable, capaz de mover materiales, piezas, herramientas o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas".* [1]

Mientras que la *International Federation of Robotics* (IFR) nos lo define como: "*Por robot industrial se entiende a una máquina de manipulación automática, reprogramable y multifuncional con tres o más ejes que puedan posicionar y orientar materias, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o en movimiento".* [1]

2.3.2 Clasificación

Resulta complicado hacer una clasificación de los diferentes tipos de robots que existen debido a la gran cantidad de factores sobre los que podría ser realizada. Primero realizaremos una clasificación según su cronología y después según su morfología. [1]

Según la Asociación Francesa de Robótica Industrial (AFRI), pueden ser clasificados en 3 generaciones según su cronología [1]:

- **1ª Generación:** robots que repiten la tarea programada de forma secuencial. No toma en cuenta las alteraciones de su entorno debido a que no se establece comunicación con él.
- **2ª Generación:** adquieren información limitada de su entorno y actúan en consecuencia. Son capaces de localizar, detectar piezas y esfuerzos y adaptan sus movimientos en consecuencia.
- **3ª Generación:** robots con programación mediante el lenguaje natural. Poseen la capacidad de planificar tareas de forma automática y tienen ciertas capacidades "inteligentes".

Basándonos en su morfología distinguimos cuatro tipos de robots [3]:

- **Robot Industrial o manipulador:** son los más usados y se componen de componentes mecánicos y electrónicos destinados a realizar procesos de fabricación en un entorno industrial de forma automática.
- **Robots móviles:** se componen de mecanismos que les permiten el desplazamiento autónomo por diferentes terrenos, como ruedas, orugas, etc. Están compuestos por sensores que son capaces de recibir información sobre el entorno. Son utilizados en entornos de muy difícil acceso como en exploraciones espaciales.
- **Androides o humanoides:** robots que intentan imitar la forma de actuar del ser humano. Destinados al marketing de las empresas que los desarrollan ya que no poseen una utilidad práctica debido a sus limitaciones y por el precio de fabricación.
- **Zoomórficos:** reproducen los sistemas de locomoción de diferentes seres vivos. El objetivo de estos es la creación de vehículos pilotados o autónomos, con capacidad de moverse en superficies muy accidentadas.

Los robots empleados en la creación de este proyecto son todos del tipo Robot Industrial. [3]

2.4 Proyectos relacionados con Robotstudio en la Universidad de Valladolid

A continuación, se va a hacer una recopilación de proyectos de robótica realizados con el *software* Robotstudio en la Universidad de Valladolid.

En el año 2015, Juan Antonio Ávila Herrero realizó su trabajo de fin de grado diseñando una célula robótica enfocada a fines educativos, con la cual los alumnos podrían realizar prácticas de robótica propuestas en la aplicación tanto de la célula real como en la simulada. Para ello diseñó diferentes componentes como una pinza como herramienta para mover objetos, una botonera para señales de entrada-salida, una caja de rotuladores para poder dibujar y dos mesas donde dibujar. [10]

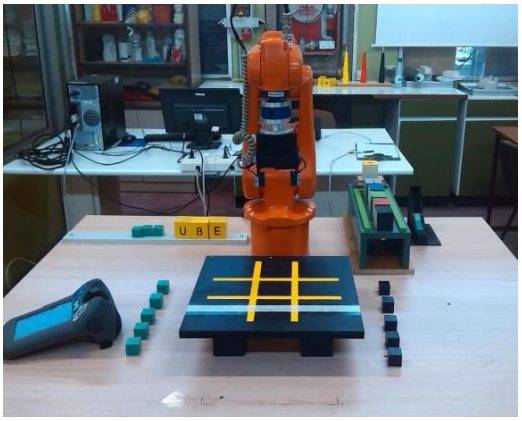


Ilustración 6. Célula robótica correspondiente al TFG de Juan Antonio Ávila Herrero. [10]

En este mismo año, Gonzalo Muinelo Garrido, en su trabajo de fin de grado, diseñó mediante Robotstudio una célula robotizada con el objetivo de que una pieza fuese tratada pasando por tres procesos diferentes. La célula estaba formada por tres máquinas diferentes y un robot central que debía ser programado para trasladar las piezas de una máquina a otra. [11]

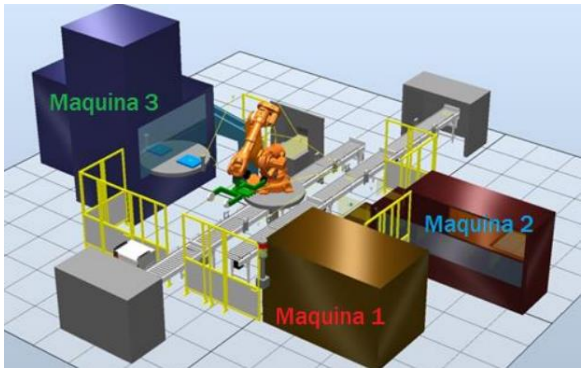


Ilustración 7. Célula robótica correspondiente al TFG de Gonzalo Muinelo Garrido. [11]

En 2016, Álvaro Galindo de Santos realizó su trabajo de fin de grado modelando la célula robotizada con la que se cuenta en el departamento. Estableció comunicación a través de *sockets* entre el robot y una *Tablet* para poder realizar una serie de juegos creados en proyectos anteriores y siendo controlados por el usuario. [12]

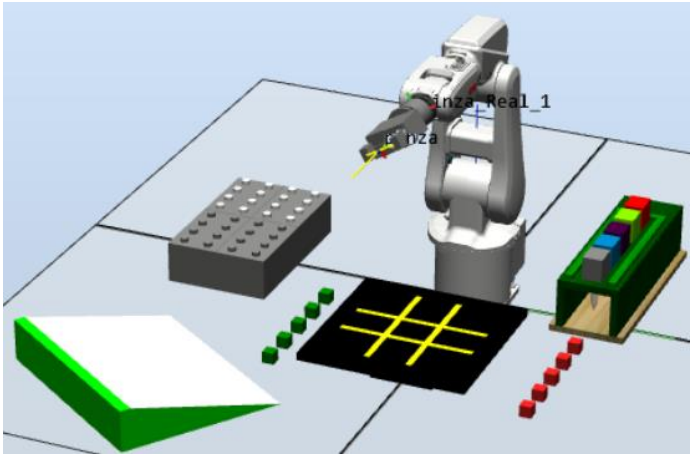


Ilustración 8. Célula robótica correspondiente al TFG de Álvaro Galindo de Santos. [12]

En el año 2021, Rodrigo Sancho García desarrolló en su proyecto de fin de grado la creación de una célula robótica compuesta por dos robots que, sincronizados, realizaban el montaje de un coche. También realizó una interfaz usuario para controlar la simulación mediante Matlab usando el protocolo de comunicaciones OPC. [13]

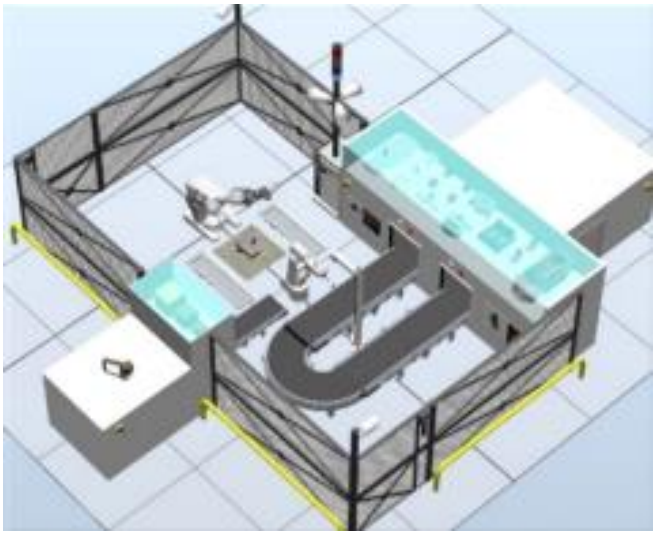


Ilustración 9. Célula robótica correspondiente al TFG de Rodrigo Sancho García. [13]

En el año 2022, Alejandro Sancho Gil abordó la creación de un entorno quirúrgico con el robot IRB 120, desarrollando una aplicación muy completa que permitía la simulación y planificación de los movimientos de las herramientas quirúrgicas con dicho robot, aplicado en operaciones de cirugía laparoscópica. [14]

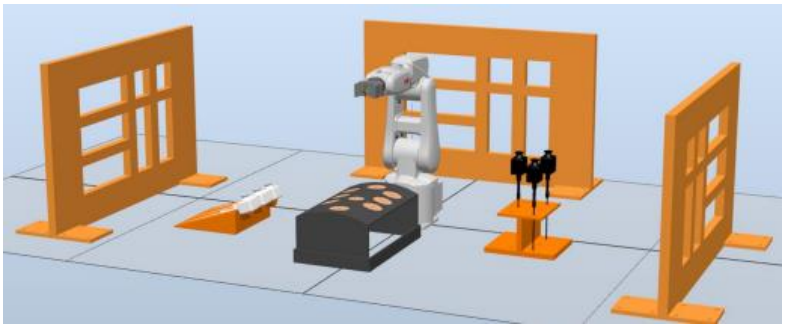


Ilustración 10. Célula robótica correspondiente al TFG de Alejandro Sancho Gil. [14]

2.5 Discusión sobre el estado del Arte

En este proyecto se ha expuesto la evolución histórica de la robótica, mencionando algunos de los robots con mayor relevancia. También se han comentado cinco proyectos realizados con el *software* Robotstudio en la Universidad de Valladolid.

Primeramente, se analizan las ventajas y desventajas de la robótica en un entorno industrial.

Ventajas	Desventajas
• Realizan tareas de forma más rápida y precisa que los humanos. • Pueden trabajar sin descanso. • Son automáticos y pueden realizar tareas repetitivas. • Pueden soportar ambientes hostiles	• Para el correcto funcionamiento necesitan energía y mantenimientos cada cierto tiempo. • La programación debe ser actualizada. • Coste elevado a corto plazo. • No mejoran con la experiencia como los humanos. • Por temas de seguridad, no todos los robots pueden compartir el espacio de trabajo con humanos.

Tabla 2. Ventajas y desventajas de la robótica en un entorno industrial.

A continuación, se realiza un análisis de los Trabajos de Fin de Grado realizados en la Universidad de Valladolid.

- En el TFG *Ávila Herrero* [10] se diseña una célula robótica enfocada a fines educativos. La realización de las prácticas de robótica propuestas en la aplicación es posible, tanto de la célula real como en la simulada, por los diferentes usuarios. Una limitación que presenta es la elevada carga computacional y, con el fin de disminuirla, se podría implementar la comunicación desde Arduino a Robotstudio sin la necesidad de usar Matlab.
- En el TFG *Muinelo Garrido* [11] se diseña una célula robotizada con el objetivo de programar un robot para que una pieza pudiese ser tratada por tres máquinas diferentes. Consigue la optimización de la distribución de las tres máquinas para que con un solo robot se pudiese realizar el proceso. Como desventaja, se podría destacar que el control del proceso por parte de los operarios no se encuentra facilitado con la posibilidad de controlar la célula desde un medio externo.
- En el TFG *Galindo de Santos* [12] modeló una célula robotizada para poder realizar distintos juegos controlados por el usuario. Debido a la comunicación mediante sockets, un gran beneficio será poder realizar el control del proceso mediante una *Tablet,* pudiendo ser utilizado por diferentes usuarios con fines didácticos. Lo descrito anteriormente presenta una desventaja debido a las continuas actualizaciones de *software* de Android, pudiendo provocar que la aplicación creada pudiese quedar obsoleta. Una propuesta sería realizar el control desde un *software* más potente, como sería Matlab.
- En el TFG *Sancho García* [13] se desarrolla la creación de una célula robótica compuesta por dos robots que realizan el montaje de un coche. Como principal ventaja destaca el control del proceso mediante una interfaz de usuario muy visual y fácil de utilizar, siendo posible su utilización por cualquier persona. El algoritmo de visión artificial utilizado a la hora de detectar las piezas requiere que el fondo y que la cinta transportadora donde se realiza la detección sean blancos, siendo una desventaja ya que no sería posible realizar el proceso en otro fondo.
- En el TFG *Sancho Gil* [14] se aborda la programación del robot IRB 120 con el fin de realizar operaciones de cirugía laparoscópica. Cabe destacar la aplicación completa desarrollada en el trabajo contemplando una gran cantidad de situaciones posibles, consiguiendo que la aplicación funcione de manera coordinada evitando que la simulación se pueda ver afectada por alguna de estas situaciones. Como limitación debería tenerse en cuenta que el robot IRB 120 no es un robot colaborativo, siendo necesario en las intervenciones quirúrgicas que el robot esté asistido por cirujanos. A mayores, se podría haber creado una interfaz de usuario para facilitar el uso de la aplicación

por el usuario, siendo necesario establecer comunicación con dos programas.

Una vez ha sido realizado el análisis de la robótica en la industria y de los diferentes Trabajos de Fin de Grado mencionados, se procede a realizar una conclusión general. Podemos deducir que la robótica es una de las ramas más completas del mundo de la ingeniería debido a la gran cantidad de campos que abarca y de las múltiples ventajas que presenta en la realización de los diferentes procesos orientados a la industria, medicina, etc. Aunque el avance de la robótica aumente de forma exponencial, aún queda mucho camino que recorrer, cuya meta se establece en conseguir una interacción segura entre robots y humanos y unos buenos resultados en los procesos a realizar. También se puede afirmar que, una vez analizados los trabajos realizados por otros compañeros, la robótica abarca un amplio abanico de posibilidades, pudiendo tener fines didácticos, médicos, industriales, etc., fines al servicio del ser humano que suponen un apoyo y una mejora de las acciones a realizar. Estas acciones podrán ser dirigidas, en caso de que sea requerido, por el usuario mediante diferentes protocolos de comunicación, que puede establecerse desde la sencillez de una aplicación hasta un *software* más potente.

3. Proyecto propuesto y software empleado

Tras realizar una introducción sobre la robótica y una vez se está informado sobre los anteriores proyectos que han sido realizados en la Universidad de Valladolid abarcando esta temática, unida con el deseo de innovar y abarcar el mayor número de campos de la ingeniería, se propone el siguiente proyecto.

Estos campos se mencionan a continuación:

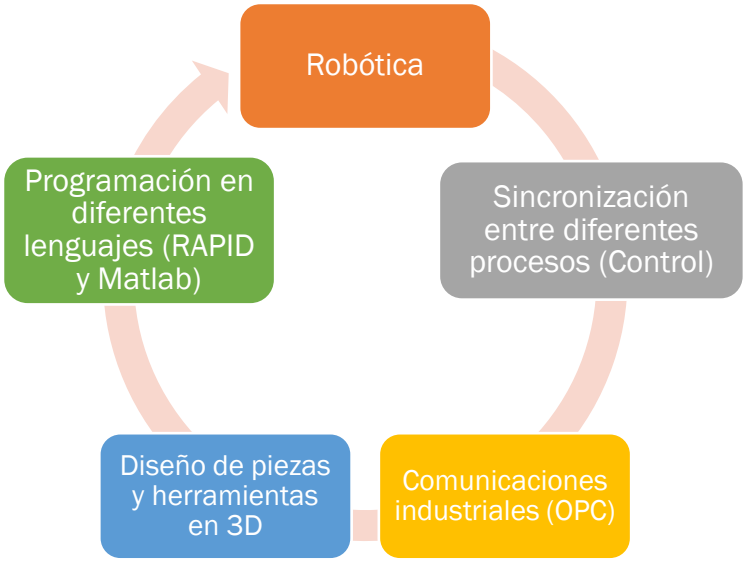


Ilustración 11. Campos de la ingeniería comprendidos en el proyecto.

3.1 Estudio de los objetivos

Una vez comentados los campos que se quieren abarcar en el desarrollo del siguiente proyecto, se procede a comentar un pequeño análisis sobre los objetivos que se tenían intención de cumplir. Es una buena práctica orientar el trabajo en torno a los objetivos establecidos antes de realizar cualquier proyecto. A continuación, se nombran los objetivos en cuestión:

- Crear una célula robótica en la que se realice un montaje mediante varios robots, en la cual cada robot realice una tarea diferente.
- Sincronización entre los robots y las tareas que debe desempeñar cada uno.
- Emplear los *Smart Components* que ofrece Robotstudio.
- Establecer comunicación de Robotstudio con un programa externo mediante OPC o TCP/IP.
- Creación de una interfaz de usuario para poder realizar el control de la simulación desde otra aplicación externa a Robotstudio.
- Crear una simulación dinámica y vistosa, pero a la vez realista, para que hubiese la opción de ser implementada en la realidad si se quisiese.

3.2 Célula robótica propuesta

Después de comentar los objetivos que se tenían se procede a describir la célula robótica. Esta se compone de tres brazos robóticos de la compañía ABB, siendo dos de ellos del modelo IRB 1600, el otro brazo robótico es el IRB 1660ID y el robot paralelo también perteneciente a ABB, cuyo modelo es el IRB 369C1, cuyas características serán descritas posteriormente. Cada uno de ellos realizará una tarea diferente para poder así completar el montaje de una luminaria.

Para poder entender correctamente el funcionamiento de la célula se va a utilizar un diagrama de flujo. En la Ilustración 12 se muestra la simbología de los diagramas de flujo junto con su significado.

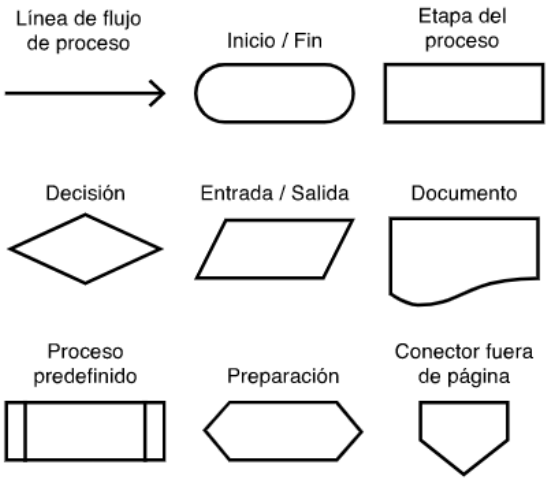


Ilustración 12. Símbolos diagrama de flujo. [15]

En el siguiente diagrama de flujo representa el funcionamiento de la célula robotizada con los cuatro robots trabajando de forma paralela.

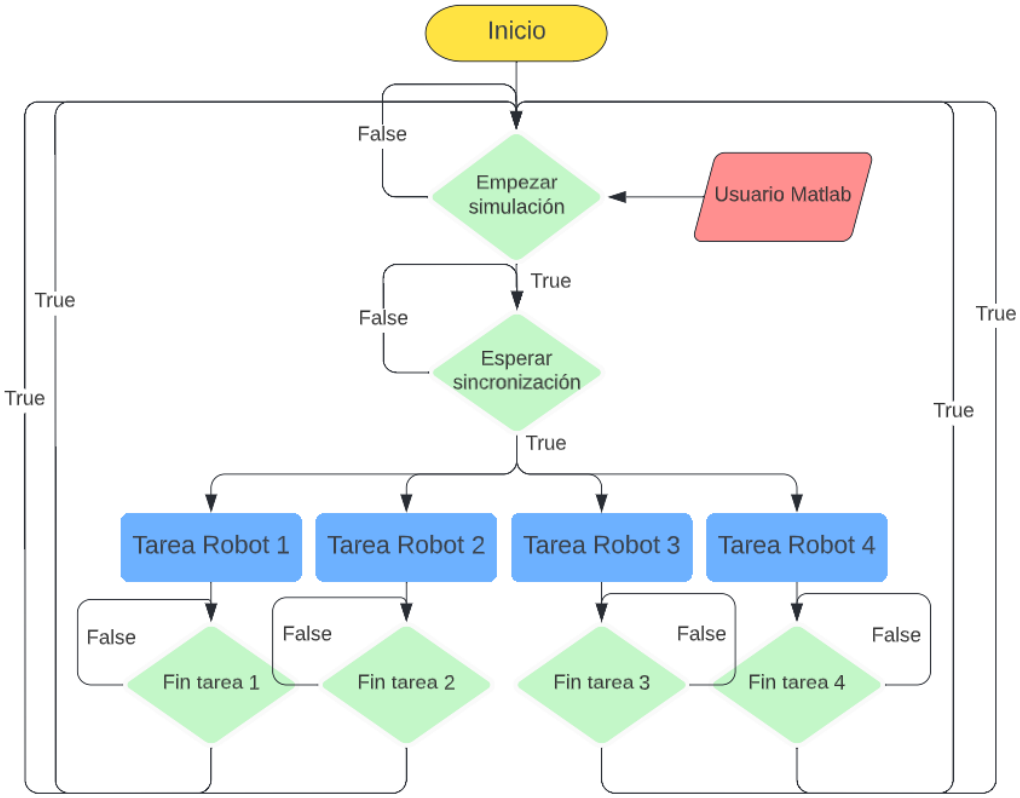


Ilustración 13. Diagrama de flujo que muestra el funcionamiento de la célula.

Hay que tener en cuenta el estado inicial ya que cuando se empieza a montar la primera luminaria no habrá piezas en todas las posiciones, por tanto, habrá robots que se encuentren estado ocioso. El proceso de montaje se podría resumir de la siguiente manera:

1º. Las piezas llegan en una bandeja mediante una cinta transportadora al primer robot (IRB 1600), este las coloca en una mesa giratoria y, una vez han sido colocadas, la mesa girará llevando las piezas al segundo robot. En este primer paso solo trabajaría el primer robot, los demás estarían ociosos esperando la llegada de piezas a su posición.
2º. Una vez hayan llegado las piezas a la segunda posición, el segundo robot (IRB 1600) realizará parte del montaje. Mientras tanto, una nueva bandeja entrará en el sistema y el primer robot volverá a colocar piezas en la mesa giratoria. Una vez que ambos robots hayan terminado, la mesa girará de nuevo.
3º. Cuando las piezas estén en la tercera posición comenzará el montaje el tercer robot (IRB 1600ID), mientras tanto los robots uno y dos seguirán con el montaje a la vez que este. Una vez hayan terminado las tareas cada uno de los robots, la mesa volverá a girar.

4º. Una vez llegue la luminaria completamente montada al cuarto paso, el robot paralelo (IRB 369C1) será el encargado de sacar la luminaria del sistema depositándola en una cinta transportadora, mientras los otros tres robots continúan con el montaje de más luminarias.
5º. A partir de este momento, los cuatro robots trabajarán en paralelo constantemente.

Una vez organizados todos los pasos del proceso de montaje, se deberá diseñar la propia luminaria, la mesa giratoria, las herramientas de los robots y los demás elementos que compondrán nuestra célula robótica. Para el correcto diseño de esta, deberemos informarnos sobre la normativa de robots industriales.

En cuanto a la interfaz, diseñada en AppDesigner deberá ser comunicada con Robotstudio mediante el protocolo de comunicaciones OPC. En esta se programará un estado de marcha/paro que parará o arrancará la simulación. También se indicará en esta, el estado de cada robot, es decir, si se encuentra funcionando o parado debido a que ha surgido algún problema.

A lo largo de los siguientes apartados se mostrarán todos los elementos y pasos necesarios para la creación de la célula robótica.

3.3 *Software* empleado

A continuación, se van a introducir todos los programas que han sido empleados para la creación del proyecto actual. Antes de ello, se realizará un breve esquema sobre lo que se pretende realizar y cómo cubre estas necesidades cada programa.

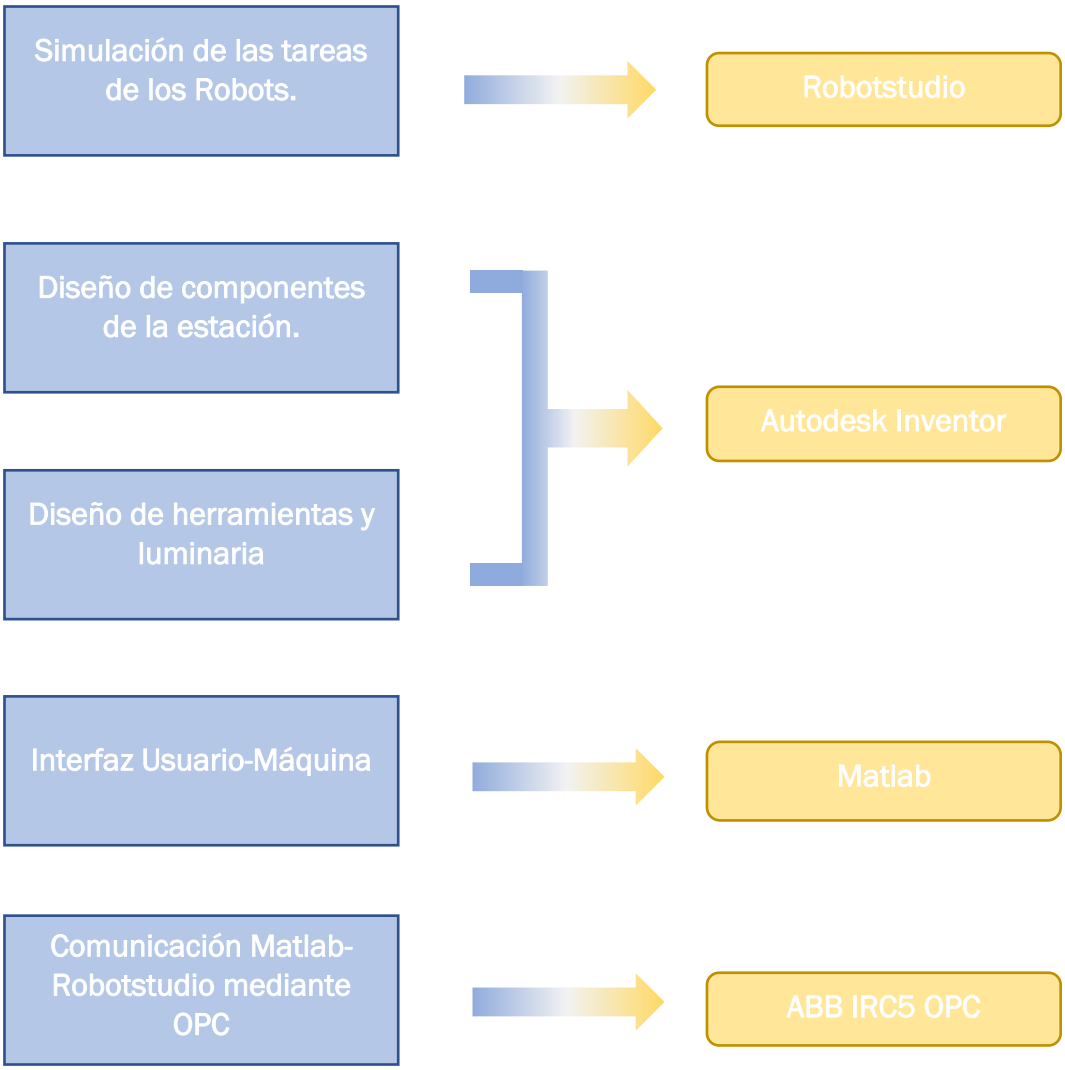


Ilustración 14. Esquema de las funciones del software empleado.

Tal y como se muestra en la Ilustración 14, han sido necesarias cuatro aplicaciones para poder realizar el proyecto, las cuales son explicadas a continuación.

3.3.1 Robotstudio

Al tratarse de la creación de una célula robótica, se necesitaba un entorno de simulación que permitiese la simulación y la programación de los movimientos de cada robot.

Ilustración 15. Logo RobotStudio.

A la hora de elegir el entorno y el fabricante de los robots se tuvo en cuenta la disponibilidad de Robotstudio en la Universidad de Valladolid, *software* que cubría perfectamente las necesidades del proyecto que se pretendía realizar. Este *software* pertenece a una de las empresas líderes en robótica industrial, denominada ABB.

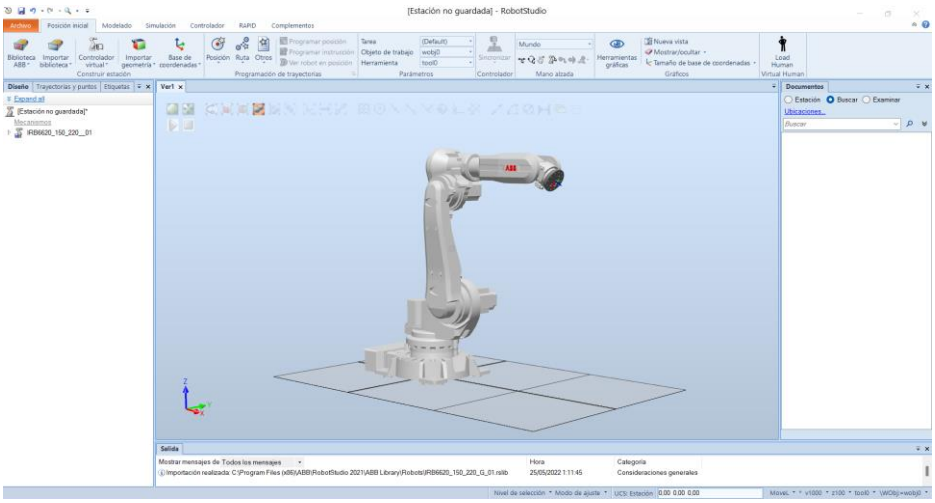


Ilustración 16. Entorno de trabajo en RobotStudio.

RobotStudio nos permite la simulación de células robóticas y la programación online y offline de los controladores de los robots. Como se acaba de comentar, existen dos modos de programación que se explican a continuación [16]:

- Modo *online,* en el cual el usuario se conecta a un controlador de robot, es decir, se programa con el robot físico. [16]
- Modo *offline,* el usuario está conectado a un controlador virtual que emula un controlador de robot en un PC. [16]

En nuestro caso, debido a la imposibilidad de realizar el proyecto de forma física, emplearemos un controlador virtual.

El controlador virtual es una copia del controlador que usan los robots de ABB en producción, esto quiere decir que cualquier movimiento que se programe en el modo offline conseguirá que el robot ejecute el mismo movimiento en la realidad. Esto supone una gran ventaja ya que la programación *offline* permite analizar y corregir el comportamiento mediante la comprobación de distintas posibilidades hasta hallar la opción que se ajuste a ciertas especificaciones de una manera mucho más segura. [16]

Cuando se abre la aplicación de Robotstudio nos aparece un menú (véase Ilustración 17) en el que nos da la posibilidad de elegir entre tres opciones: "Solución con estación vacía", "Solución con estación y controlador virtual" y "Estación vacía".

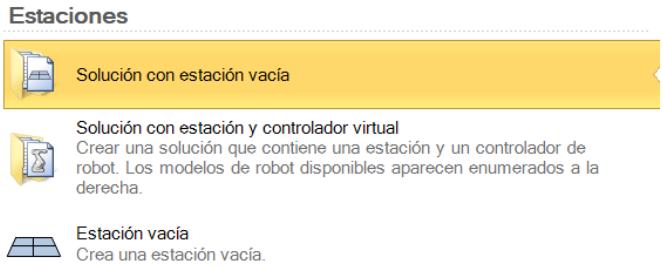


Ilustración 17. Selección del tipo de estación en Robotstudio.

En este proyecto se seleccionó la opción de estación vacía, por lo que posteriormente se tuvo que introducir y configurar el controlador.

Para poder añadir un robot seleccionamos, en el menú principal, "Posición inicial", nos saldrá la opción de "Biblioteca ABB" donde nos aparecerán los robots con los que se puede trabajar.

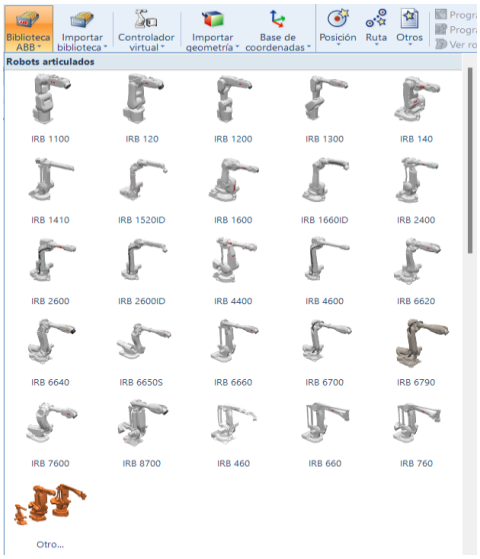


Ilustración 18. Robots disponibles en Robotstudio.

Para añadir un controlador a nuestra estación primero se deben añadir los robots necesarios, después seleccionamos las opciones “Controlador virtual” y “Crear controlador desde diseño”. A continuación, se nos abrirá una ventana donde daremos un nombre a nuestro controlador. Después debemos seleccionar los robots que queremos añadir, como se muestra en la Ilustración 19.

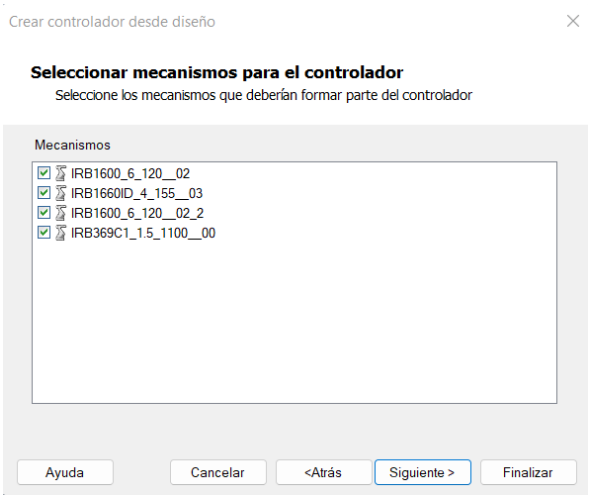


Ilustración 19. Selección de robots en el controlador.

Posteriormente, el programa asignará a cada robot una tarea de movimiento automáticamente, en este paso simplemente tenemos que seleccionar “Siguiente” e iremos a otra pestaña donde tendremos que dar a “Opciones” y se nos abrirá un menú con varias opciones.

El paso actual es esencial, debido a que dota al controlador de poder establecer comunicación mediante OPC, TCP/IP, entre otros, con otras aplicaciones. En “Communication” debemos elegir la opción de “PC Interface” y después le daremos a “Aceptar”, como se muestra en la Ilustración 20.

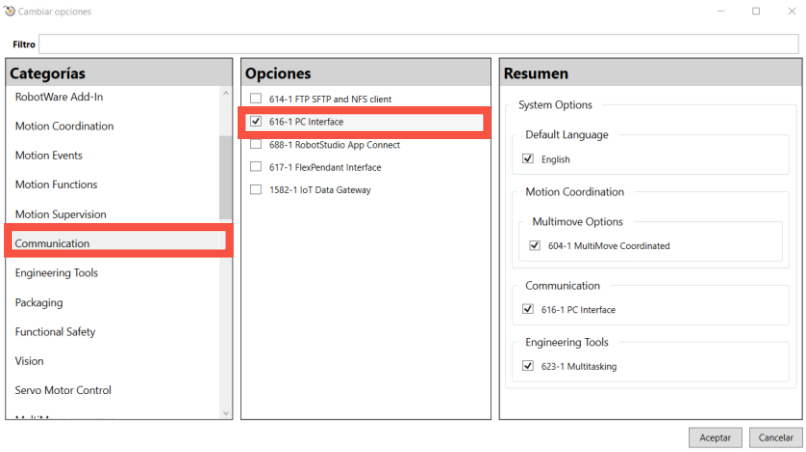


Ilustración 20. Menú de opciones del controlador.

Después seleccionaremos "Finalizar" y nuestro controlador estaría correctamente configurado. Debemos esperar un tiempo a que el controlador arranque y, una vez pasado este tiempo, nos saldrían disponibles las tareas de movimiento de cada robot y podríamos comenzar la programación de cada uno de ellos.

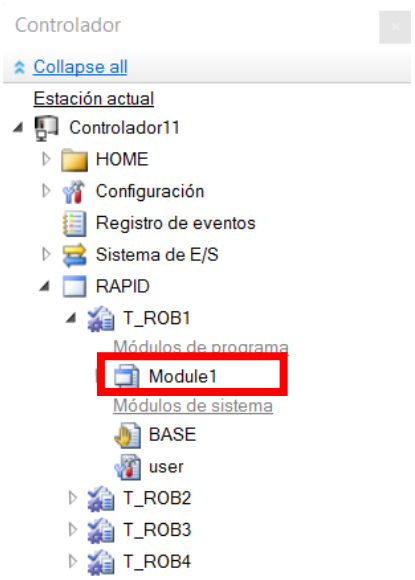


Ilustración 21. Visualización de las tareas de cada robot.

Para comenzar el programa de RAPID, lenguaje de programación de Robotstudio, simplemente tendremos que seleccionar el recuadro de color rojo señalado en la Ilustración 21. Por defecto sale el nombre de "Module1", pero se le puede cambiar el nombre al que se desee. La programación de cada robot será explicada en el capítulo 5.1.2 Programación en RAPID.

3.3.2 Matlab

Otra de las herramientas importantes para la creación de este proyecto ha sido Matlab plataforma de cálculo numérico y de programación utilizada por millones de ingenieros y científicos. Algunas de las utilidades de este programa se citan a continuación:

Utilidades de Matlab:
• Procesamiento de señales. • Visión artificial. • Comunicaciones. • Diseño de distemas de control. • Robótica. • Procesamiento de imágenes. • Análisis de datos. • Desarrollo de algoritmos. • Modelado de sistemas.

Tabla 3. Tabla sobre las utilidades de Matlab. [17]

Esta aplicación cuenta con un lenguaje de programación propio, el lenguaje M, siendo compatible con plataformas como Unix, Microsoft Windows, macOS y GNU/Linux. Las prestaciones más destacadas de este *software* son la representación de datos y funciones, la creación de interfaces de usuario, manipulación de matrices, la implementación de algoritmos y la comunicación con otros programas que poseen lenguajes diferentes. Estas prestaciones pueden ser ampliadas mediante la instalación de *toolbox,* que permiten aumentar el campo de aplicación de este *software*. [17]

Ilustración 22. Logo Matlab. [17]

En este proyecto se han utilizado una *toolbox* y una aplicación que incluye Matlab, las cuales, se mencionan a continuación:

La *toolbox* requerida para establecer comunicación con Robotstudio a través de OPC es '*OPC Toolbox',* que proporciona acceso a datos históricos de plantas industriales en tiempo real permitiendo la lectura, la escritura y el registro de datos OPC de diferentes dispositivos.

Permite trabajar con diferentes datos de servidores en tiempo real e historiadores de datos que cumplen con los siguientes estándares:

- OPC UA.
- OPC Data Access (DA).
- OPC Classic Historical Data Access (HDA).

También ofrece una conexión segura gracias a la utilización de algoritmos de cifrado y métodos de autenticación de usuarios. [18]

Para desarrollar la interfaz gráfica hombre-máquina se ha empleado una aplicación de Matlab denominada '*App Designer'*, en la que se pueden crear aplicaciones profesionales de una forma sencilla.

Para crear el diseño, esta aplicación cuenta con una biblioteca de elementos que el usuario puede emplear para el desarrollo de su aplicación. Una vez son seleccionados los elementos que forman la interfaz. La propia aplicación va generando automáticamente código orientado a objetos que especifica la distribución y diseño de la interfaz.

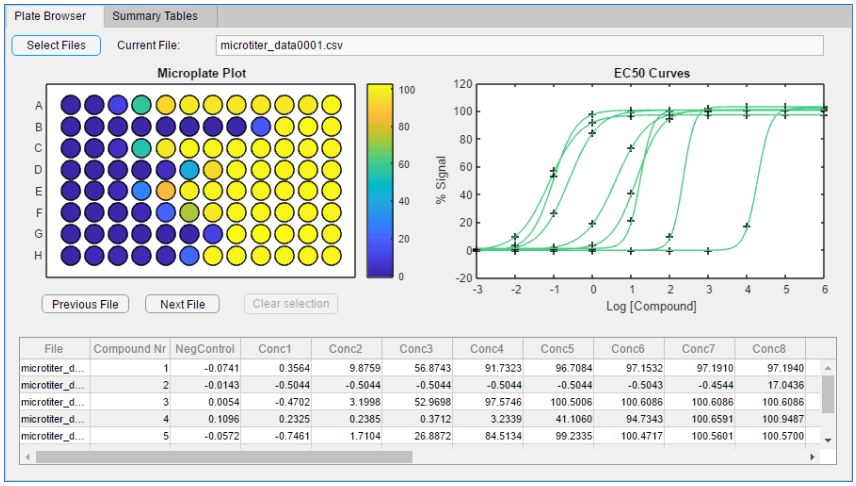


Ilustración 23. Ejemplo interfaz gráfica realizada con App Designer.

3.3.3 ABB IRC5 OPC

Como se ha comentado anteriormente, en este proyecto se debía establecer comunicación entre diferentes programas, por lo que se necesitó emplear este *software* perteneciente a ABB.

Antes de describir este programa, se va a realizar una explicación sobre que es OPC (*Open Protocol Communication*). OPC es una tecnología diseñada para comunicar datos de forma segura y operativa en diferentes sectores de la industria, sobre todo en la industria de la automatización [19] y está basado en una serie de normas industriales para la interconectividad de sistemas que proporciona una interfaz común para las comunicaciones entre diferentes productos de distintos proveedores. [20]

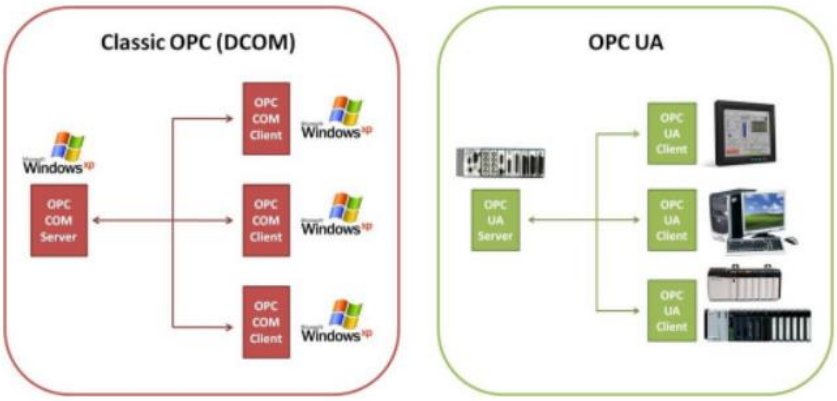


Ilustración 24. Compatibilidad de OPC clásico y OPC UA. [19]

OPC se creó para funcionar únicamente en Windows (véase Ilustración 24). Este estándar ha evolucionado y se ha sustituido por OPC UA (Arquitectura unificada) cuyo objetivo era mantener la funcionalidad de la OPC clásica sustituyendo tecnología COM/DCOM de Microsoft por la tecnología de servicios de la web y a mayores pudiéndose integrar en [20]:

- Sistemas de ejecución de fabricación (MES).
- Sistemas de planificación de recursos de la empresa (ERP).
- Controladores y dispositivos inteligentes con capacidad de funcionamiento en tiempo real.

Por lo tanto, su uso ya no es exclusivo de los entornos de Windows como se ha comentado anteriormente, sino que se adapta a la Fundación de Comunicación de Windows (WCF), que es un modelo de programación empleado para generar aplicaciones que se intercomunican. [20] El rango de aplicaciones de OPC UA aumentó considerablemente debido a que la compatibilidad y conectividad también aumentaron.

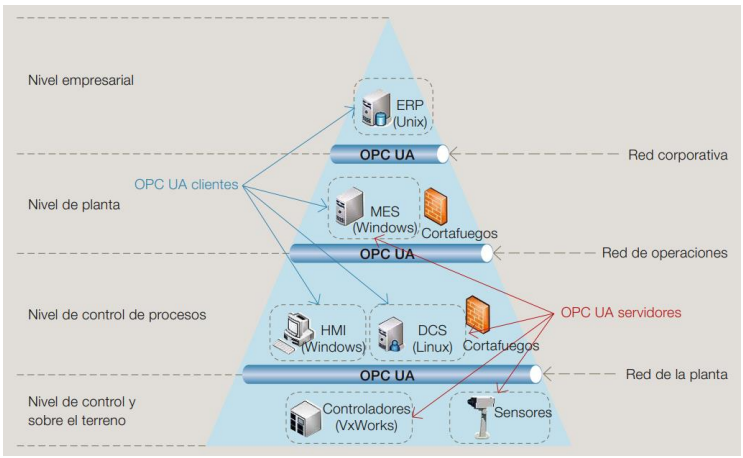


Ilustración 25. Aplicaciones de OPC UA en la pirámide de automatización. [20]

Una vez realizada la introducción a OPC, se realiza a continuación una explicación sobre la aplicación ABB IRC5 OPC.

La aplicación de configuración ABB IRC5 OPC UA se utiliza para crear y administrar alias para los controladores de robot IRC5 de ABB. Un alias es un descriptor fácil de usar que representa una interfaz de comunicaciones para un controlador de robot IRC5 de ABB. [21] Con este *software*, se pueden transmitir datos sin problemas desde la planta de la fábrica de máquina en máquina, al sistema de monitorización (MES) o a la nube, conectando robots ABB a dispositivos de otros proveedores al sistema ERP.

En el siguiente esquema se muestra cómo se realiza la comunicación en este proyecto:

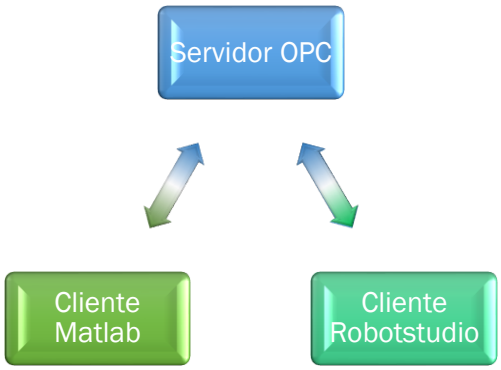


Ilustración 26. Esquema del modelo Cliente-Servidor.

Para crear el alias del controlador debemos seguir los siguientes pasos:

1º. Al abrir la aplicación nos aparecerá la siguiente pestaña donde deberemos pulsar a 'Add new Alias' (señalado en rojo en la Ilustración 27).

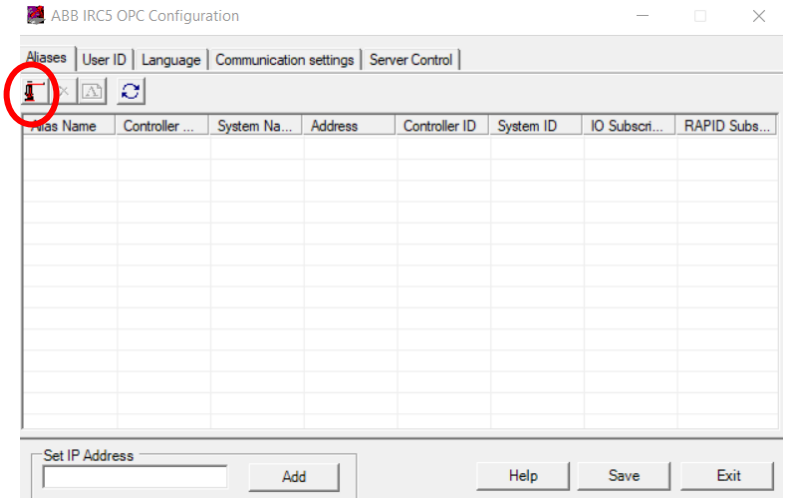


Ilustración 27. Configuración del alias del controlador.

2º. Una vez hecho esto se nos abre una ventana con la opción 'Scan'. Cuando se seleccione esta opción aparecerán todos los controladores disponibles. Se deberá seleccionar entonces el controlador deseado.

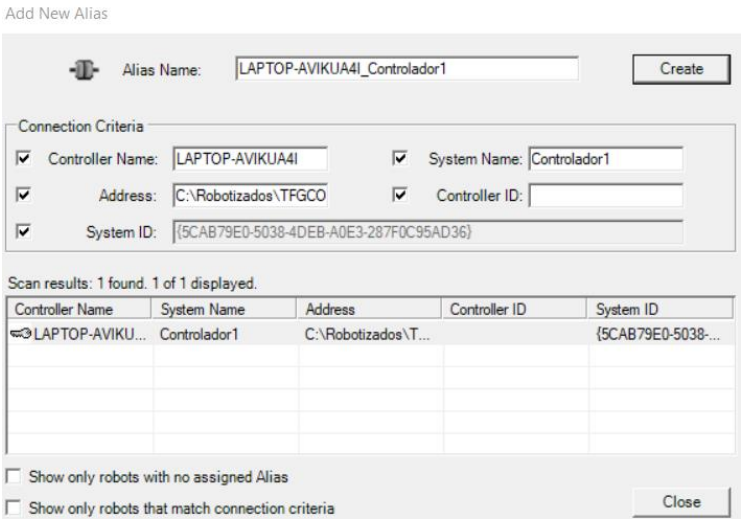


Ilustración 28. Selección del controlador.

3º. Una vez se ha seleccionado el controlador, se debe dar al botón 'Create' y el alias del controlador ya estará correctamente configurado.

3.3.4 Autodesk Inventor

En este proyecto hay una parte de diseño, que era necesaria para crear correctamente la estación y todos los elementos que la componen, incluyendo el diseño de la luminaria que posteriormente iba a ser montada por los robots. Para ello, se necesitaba un *software* que permitiese diseñar sin problemas todos estos componentes. La aplicación elegida para realizar estos diseños fue Autodesk Inventor, que es una herramienta potente con la que se puede realizar prácticamente cualquier tipo de pieza.

Ilustración 29. Logo Autodesk Inventor.

A continuación, se va a realizar una introducción a este programa.

Autodesk Inventor es un paquete de modelado paramétrico de sólidos en 3D creado por la empresa de *software* AutoDesk. Ofrece herramientas profesionales para el diseño mecánico en 3D, documentación y la posibilidad de simular productos, lo que facilita el trabajo de diseño de productos. Tuvo su inicio en septiembre de 1999, años después de otros *softwares* conocidos como:

- SolidWorks.
- Pro/ENGINEER.
- CATIA.
- Solid Edge.

Son muchas las posibilidades que nos ofrece este *software*. A continuación, se comentan las más importantes:

Utilidades Autodesk Inventor:
• Diseño de piezas. • Diseño de ensamblajes. • Diseño de tubos y tuberías. • Diseño de cableado. • Diseño de moldes y mecanizado. • Diseño de piezas de chapa. • Diseño de estructuras. • Análisis de tensiones • Creación de planos normalizados. • Documentación de diseño y fabricación.

Tabla 4. Utilidades más importantes de Autodesk Inventor.

En este proyecto no se han necesitado la mayoría de ellas ya que la aplicación se ha utilizado únicamente para el diseño, sin entrar en detalles técnicos de cada pieza.

Una vez se ha introducido los aspectos más importantes de la aplicación se van a mostrar las opciones que se tienen a la hora de crear una pieza en 3D. Al entrar en la aplicación se nos abrirá el siguiente menú (véase Ilustración 30), en este proyecto las opciones que se han utilizado son pieza y ensamblaje.

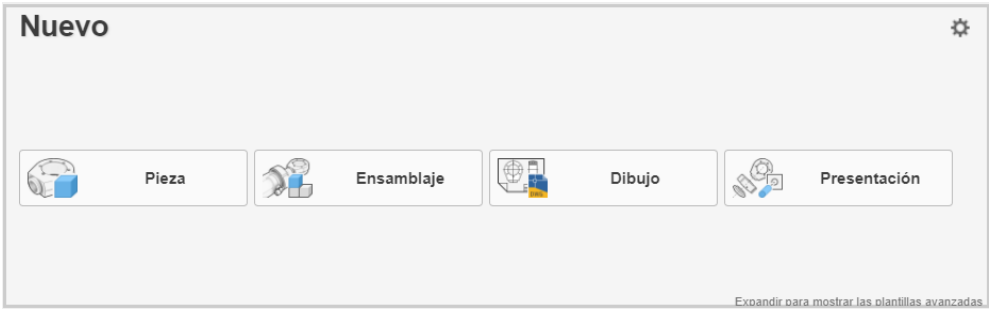


Ilustración 30. Menú de opciones Autodesk Inventor.

Después de seleccionar la opción "Pieza", seleccionaremos la opción de crear un nuevo boceto en 2D y se nos abrirá un menú donde elegiremos el plano donde queremos trabajar (véase Ilustración 31), este paso es muy importante debido a que después se van a importar las piezas a Robotstudio y si seleccionamos mal el plano de trabajo las piezas pueden importarse giradas o con el origen de la pieza colocado de forma incorrecta.

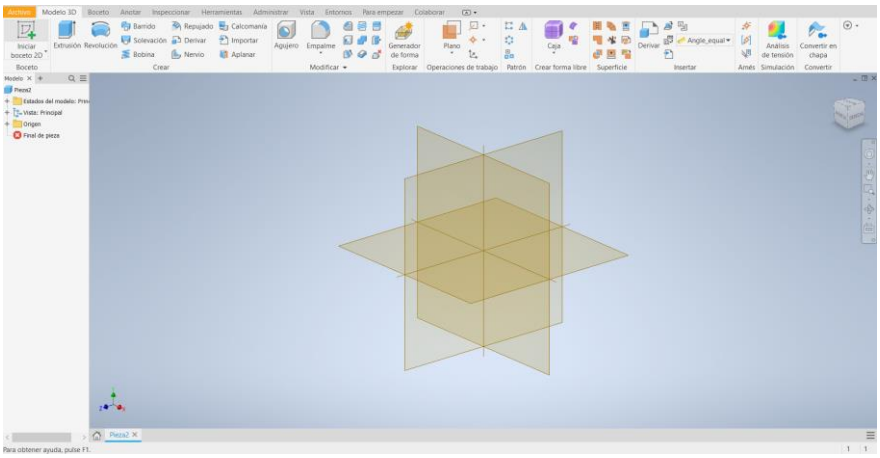


Ilustración 31. Selección de plano de trabajo en Autodesk Inventor.

Después de seleccionar el plano, se procedería a dibujar el boceto en 2D. Una vez terminado le daríamos a "Terminar boceto" y nos dejaría seleccionar una de las opciones que se muestran en la Ilustración 32:

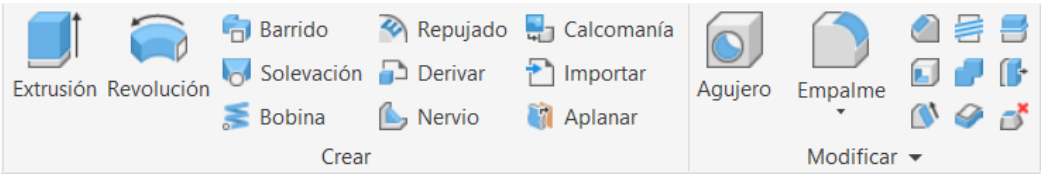


Ilustración 32. Opciones de crear una pieza 3D en Autodesk Inventor.

Las opciones más utilizadas en este proyecto se definen a continuación:

- Extrusión: crea un cuerpo añadiendo profundidad a un boceto.
- Revolución: crea un cuerpo mediante la revolución de un cuerpo alrededor de un eje.
- Barrido: recorre un boceto a lo largo de una trayectoria seleccionada para crear un cuerpo.
- Agujero: crea diferentes tipos de agujero basados en puntos de boceto.
- Empalme: añade un redondeo o empalme a una arista seleccionada.

Esta aplicación también nos permite definir el material del que están hechas nuestras piezas, para ello en el menú principal debemos seleccionar "Herramientas" y nos saldrá la opción de "Material", donde podremos elegir el tipo de material del que estará compuesta nuestra pieza. Por ejemplo, en la Ilustración 33 se ha elegido oro como material.

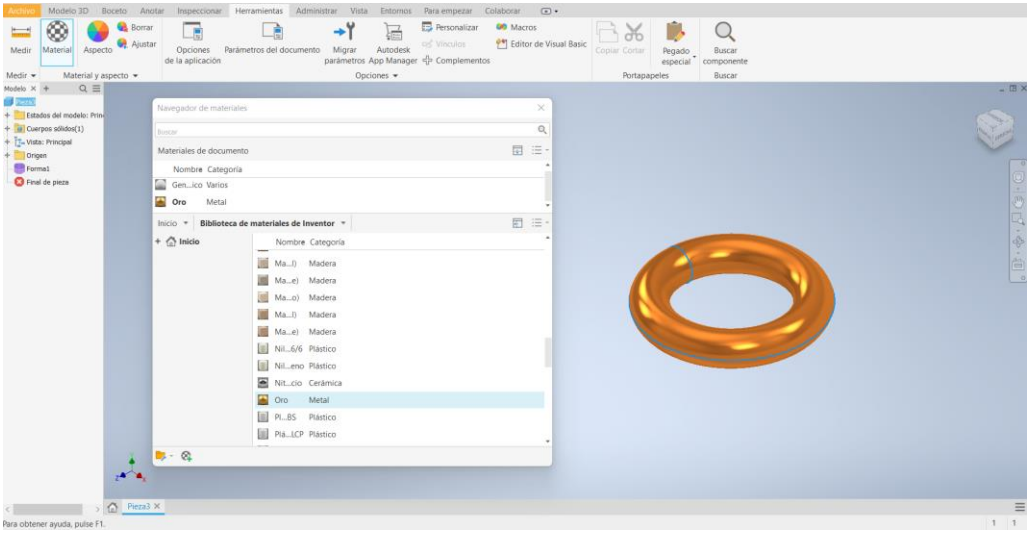


Ilustración 33. Selección de material en Autodesk Inventor.

Una vez están creadas las piezas que formarán un objeto, procedemos a crear un ensamblaje para poder juntarlas en una única pieza. Para ello, en el menú principal (véase Ilustración 30), seleccionamos 'Ensamblaje'.

Una vez estamos en la opción ensamblaje, debemos insertar las piezas que conformarán el objeto y después colocarlas aplicando restricciones. Para ello seleccionamos la opción 'Restringir' y se nos abrirá el siguiente menú (véase Ilustración 34) donde podremos añadir restricciones que controlan la posición y el comportamiento de los componentes que forman nuestro ensamblaje.

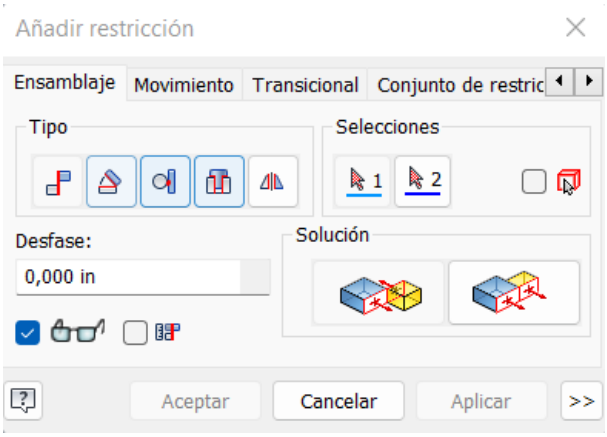


Ilustración 34. Añadir restricción en Autodesk Inventor.

4. Diseño de la estación

Antes de mostrar los componentes que forman la estación y el aspecto final de esta, se va a hacer una revisión sobre la normativa de seguridad de robots en entornos industriales ya que este aspecto ha influido a la hora de diseñar la célula robótica.

4.1 Seguridad en instalaciones robotizadas

Debido a las características de trabajo de los robots, no es necesaria la presencia humana para su funcionamiento. Este alejamiento entre robot y humano significa que el riesgo de accidente es menor, ya que el operario no está cerca del peligro, es decir, del robot. No se tienen garantías de que el operario se encuentre siempre alejado del entorno de trabajo del robot, ya que, por ejemplo, se deben hacer revisiones, limpieza del entorno, etc., por lo que el riesgo mencionado anteriormente no se elimina del todo. La forma de garantizar en el entorno de trabajo del robot la ausencia del hombre es instalar unos elementos que impidan el acceso del trabajador a la zona de peligro, o en su defecto, medios destinados a detectar la presencia dentro de la célula de un operario y proceder a detener al robot en su movimiento cuando el operario entre en esta. [22]

Los riesgos de los robots se pueden dividir en dos tipos:

Riesgos tradicionales:		
Factores físicos:	**Factores químicos:**	**Factores biológicos:**
Polvo, ruidos, vibraciones, temperatura, electricidad estática, humedad, etc.	Según como actúan en el organismo pueden ser tóxicos, irritantes, cancerígenos, etc.	Su acción sobre el organismo puede dar lugar a enfermedades profesionales.
Factores fisiológicos:	**Factores psicológicos:**	
Distinguimos dos tipos, sobrecarga estática (postura) y sobrecarga dinámica (esfuerzos)	En ciertas condiciones de trabajo se pueden producir sobrecargas neuropíquicas.	

Tabla 5. Riesgos tradicionales. [22]

Riesgos específicos:		
Debido a la automatización de los robots industriales en momentos determinados pueden ser impredecibles en sus acciones, por ello hay que tener en cuenta los siguientes riesgos:		
Riesgo de colisión entre hombre-máquina:	**Riesgo de proyección :**	**Riesgo de atrapamiento:**
Provocados por golpes causados por el movimiento del robot.	Debido al alcance de piezas que el robot deje caer o proyecte sobre un operario.	El robot al moverse puede atrapar a un trabajador.

Tabla 6. Riesgos específicos. [22]

A continuación, se nombran las causas que originan estos accidentes.

Ilustración 35. Causas de accidentes provocados por robots. [1]

Es importante destacar que, según estudios realizados por 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, etc., mientras que sólo el 10% ocurre durante el funcionamiento normal de la línea. [1]

Hasta principios de los años noventa la normativa legal relativa a la instalación y empleo de robots era muy escasa debido a varios factores:

- Necesidad de enfrentarse a los problemas técnicos y de ventas antes que con otros problemas.
- No se tenía una experiencia suficiente en materia de accidentes ocasionados por robots como para establecer una casuística válida.
- Demasiados inconvenientes a la hora de unificar criterios y niveles de seguridad entre diferentes compañías y países.
- La dificultad y tiempo necesario para preparar la documentación referente a la normativa.

Las primeras normativas, se crearon alrededor de los años 90 y se citan las más importantes:

Normativa internacional ISO 10218 :1992. [32]

Normativa americana ANSI/RIA R15.06-1992. [32]

Normativa europea EN 775 y española UNE-EN 775. [32]

Todas ellas al ser tan antiguas ya han sido anuladas por normativas más modernas. En concreto, la última citada ha sido sustituida por diferentes normativas hasta llegar a la actual:

UNE-EN ISO 10218-1:2012: Robots y dispositivos robóticos. Parte 1: Robots. Requisitos de seguridad para robots industriales. [32]

Esta será anulada en el futuro por la siguiente normativa:

PNE-prEN ISO 10218-1: Robótica. Requisitos de seguridad. Parte 1: Robots industriales. [32]

Medidas de seguridad a tomar en la fase de diseño de la célula robotizada.

Para el diseño de la célula se deben utilizar barreras de acceso y protecciones que minimicen el riesgo de aparición de accidentes, a continuación, se citan las siguientes medidas de seguridad [23]:

- **Barreras de acceso a la célula:** se restringirá la entrada a la célula colocando barreras que limitarán el entorno de trabajo impidiendo el acceso a personas. En caso de que algún operario o cualquier persona entre dentro de la célula deberá producirse parada inmediata.
- **Dispositivos de intercambio de piezas:** si el operador debe introducir o recoger piezas dentro de la célula se utilizarán dispositivos que permitan realizar estas operaciones a distancia, así como cintas transportadoras, mesas giratorias, etc.
- **Movimientos condicionados**: si el operador debe entrar en el interior de la célula mientras esta se encuentra en estado de funcionamiento, se debe programar al robot para que este no haga ningún movimiento mientras el operario este dentro de su campo de acción.
- **Zonas de reparación:** deberán existir zonas de reparación y mantenimiento. dentro del campo de la célula de trabajo, pero fuera del campo de acción del robot. Mientras el operario esté en las zonas de reparación, el robot no realizará ningún movimiento.
- **Condiciones adecuadas en la instalación auxiliar:** los sistemas eléctricos, neumáticos o hidráulicos deberán estar equipados correctamente con protecciones, aislamientos, etc., para tener un funcionamiento sin riesgo para el operario.
- **Mecanismos de acceso:** se debe proveer a la célula de trabajo de dispositivos con código para evitar la entrada a personal no autorizado, así como sensores de presencia o proximidad, sistemas de visión, barreras de seguridad fotoeléctricas, etc.
- **Parada de emergencia:** se debe instalar un número adecuado de botones o interruptores de parada de emergencia tanto en el interior como en el exterior de la célula, sobre todo en los puntos críticos.
- **Señalización adecuada:** se deben instalar tanto mecanismos audibles (alarmas) como semáforos que indiquen continuamente el estado de la célula.

Una vez definidas las medidas de seguridad necesarias para el diseño de la célula, se van a citar las características de sistemas de seguridad que deben ser empleados. [23]

Barreras materiales: la dimensión de estas debe estar relacionada con el riesgo existente y el robot instalado. Este sistema de protección se basa en la combinación de altura y distancia con el propósito de no acceder al punto peligroso. Estas distancias se establecen en las siguientes normas:

Ilustración 36. Normas de las dimensiones de barreras materiales. [23]

Observando la Ilustración 37, vemos que tenemos tres parámetros, "a", "b" y "c" y en función de los valores que tengamos deberemos de interpretarlos en la Tabla 7. Observando el punto más crítico de nuestra célula, se deduce que las barreras deberán tener una altura mínima entre 1800mm y 2000mm, por lo que se elige el valor que mayor nivel seguridad nos ofrezca, es decir, 2 metros.

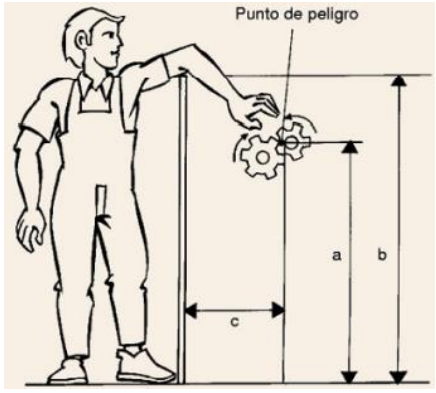


Ilustración 37. Parámetros que influyen en la altura de las barreras materiales. [23]

DISTANCIAS DE UN PUNTO DE PELIGRO DESDE EL SUELO a mm	ALTURA DEL BORDE DE LA BARRERA b mm							
	2400	2200	2000	1800	1600	1400	1200	1000
	DISTANCIA HORIZONTAL DESDE EL PUNTO DE PELIGRO c mm							
2400	100	100	100	100	100	100	100	100
2200	-	250	350	400	500	500	600	600
2000	-	-	350	500	600	700	900	1100
1800	-	-	-	600	900	900	1000	1100
1600	-	-	-	500	900	900	1000	1300
1400	-	-	-	100	800	900	1000	1300
1200	-	-	-	-	500	900	1000	1400
1000	-	-	-	-	300	900	1000	1400
800	-	-	-	-	-	600	900	1300
600	-	-	-	-	-	-	500	1200
400	-	-	-	-	-	-	300	1200
200	-	-	-	-	-	-	200	1100
0	-	-	-	-	-	-	200	1100

Tabla 7. Tabla en la cual se fija la altura de la barrera en función de tres parámetros. [23]

Barreras inmateriales: también llamadas cortinas fotoeléctricas, emplean elementos optoelectrónicos emisores y receptores, que forman una cortina de radiaciones ópticas que detectan la interrupción de estas acciona una alarma y que cortará el suministro eléctrico parando la célula completamente.

Deben ser colocadas de manera que sea imposible acceder al espacio de trabajo sin atravesarlas.

Cuando se instalan las barreras fotoeléctricas se debe tener en cuenta la distancia que será recorrida por la persona antes de que el robot se pare. Esta distancia viene dada por la siguiente fórmula. [23]

$$D \geq \mathrm{V} \cdot (t_1 + t_2) + C_3$$

Donde:

D = Distancia de protección

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

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.

4.2 Elementos que componen la célula robotizada

Una vez estudiada la normativa y las medidas a seguir se van a comentar todos los componentes por los que está formada la célula.

4.2.1 Plataforma giratoria

Como se ha comentado anteriormente, se van a utilizar cuatro robots para realizar el montaje. Con el objetivo de realizar una simulación interactiva, se diseñó una plataforma con forma de cruz (véase Ilustración 38), de tal forma que en cada extremo de esta pudiesen trabajar los robots. El área donde cada robot realizará sus tareas se señala en rojo en la Ilustración 38.

Esta plataforma cuenta con una base donde en el interior de ella se encuentra la electrónica y el motor que la permitirá girar.

Esta pieza, al ser importada desde Autodesk Inventor a Robotstudio, carece de la capacidad de girar, por lo que en Robotstudio se deberá crear un mecanismo

que aporte estas características a la plataforma. La creación de este mecanismo será explicada en el capítulo 5.1.1 Creación de mecanismo.

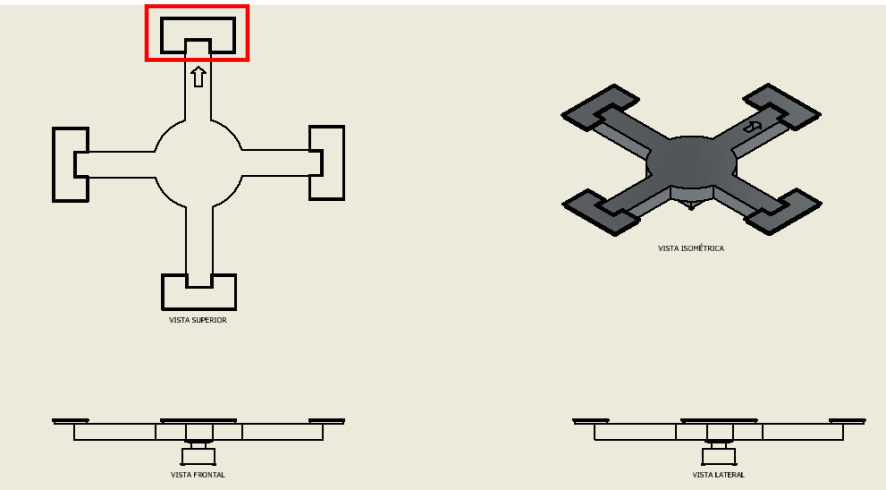


Ilustración 38. Plataforma giratoria diseñada por el autor del proyecto.

4.2.2 Soporte de herramientas

Debido a que dos de los robots realizan operaciones diferentes se necesitaban herramientas distintas para poder ejecutarlas. Es por ello que se diseñaron estas plataformas para permitir que el robot tome y deje las herramientas según la acción que vaya a realizar.

Se crearon dos tipos de soportes, uno con capacidad para dos herramientas para el segundo robot y otro con capacidad para tres herramientas para el tercer robot.

Como se puede observar en la Ilustración 39, estos cuentan con diferentes formas extruidas, para que cada herramienta se ajuste al hueco perfectamente.

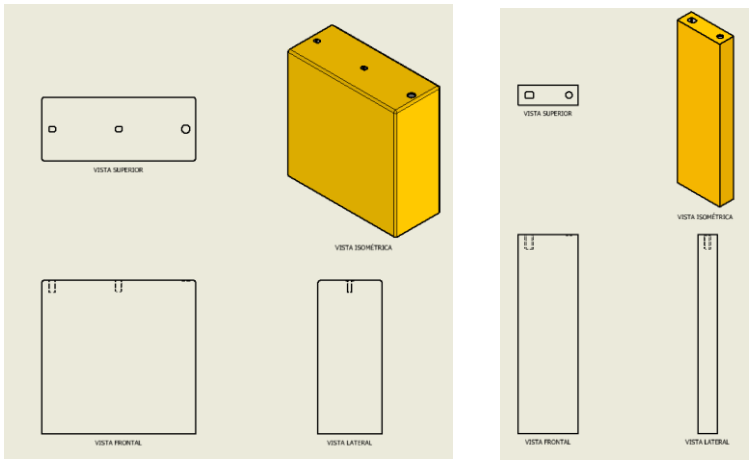


Ilustración 39. Soportes de herramientas diseñados por el autor del proyecto.

4.2.3 Soporte de tornillos

Este soporte se diseñó con el fin de que el robot tuviese más facilidades a la hora de coger los diferentes tornillos. Está compuesto por dos elementos: la base, que tiene forma de tubo y el propio soporte, que cuenta con una serie de agujeros donde se introducirán los diferentes tornillos y que posteriormente el robot cogerá para fijarlos en luminaria.

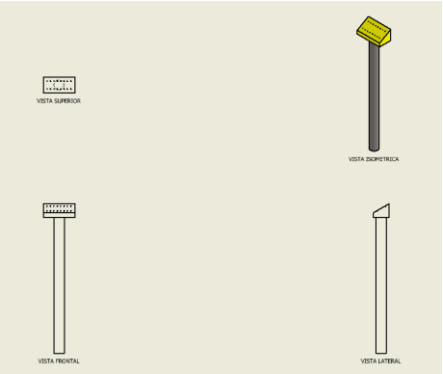


Ilustración 40. Soporte de tornillos diseñado por el autor del proyecto.

4.2.4 Plataforma intercambio de material

Se necesita abastecer a la célula de diferentes materiales necesarios para el montaje de la luminaria, pero, al tratarse de una célula que está compuesta por robots, los operarios no pueden entrar al interior de esta mientras se encuentre en estado de funcionamiento. Es por ello que surgió la necesidad de diseñar plataformas que permitan el intercambio de material del interior al exterior de la propia célula.

Como se puede apreciar en Ilustración 41, está compuesto por una base formada por cuatro cilindros, que sujetan un tubo hueco por el que el operario introducirá el cabezal del soporte de tornillos mencionado anteriormente (véase la Ilustración 40).

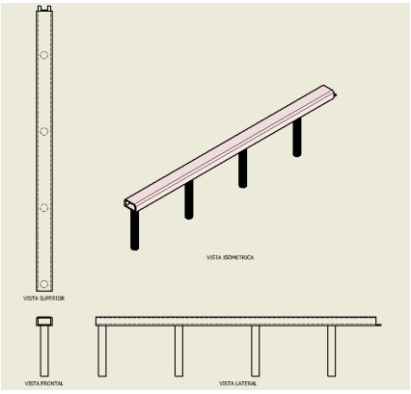


Ilustración 41. Plataforma de intercambio de material diseñado por el autor del proyecto.

4.2.5 Cinta transportadora

Esta cinta está ubicada en la mesa donde trabaja el operario y la necesidad de crearla fue para cumplir con la normativa de seguridad, ya que el operario no puede trabajar dentro del espacio de trabajo del robot. El funcionamiento de esta cinta será desplazar desde un extremo a otro la pieza que el robot deposite sobre ella para hacérsela llegar al operario y que, una vez el operario haya terminado con su tarea, la pieza volviese a ser transportada al otro extremo.

La cinta está formada por una serie de rodillos que permiten el giro de la banda transportadora, donde se apoyan las piezas que van a ser transportadas. Esta banda debe ser de un material resistente, por lo que se eligió el caucho como material.

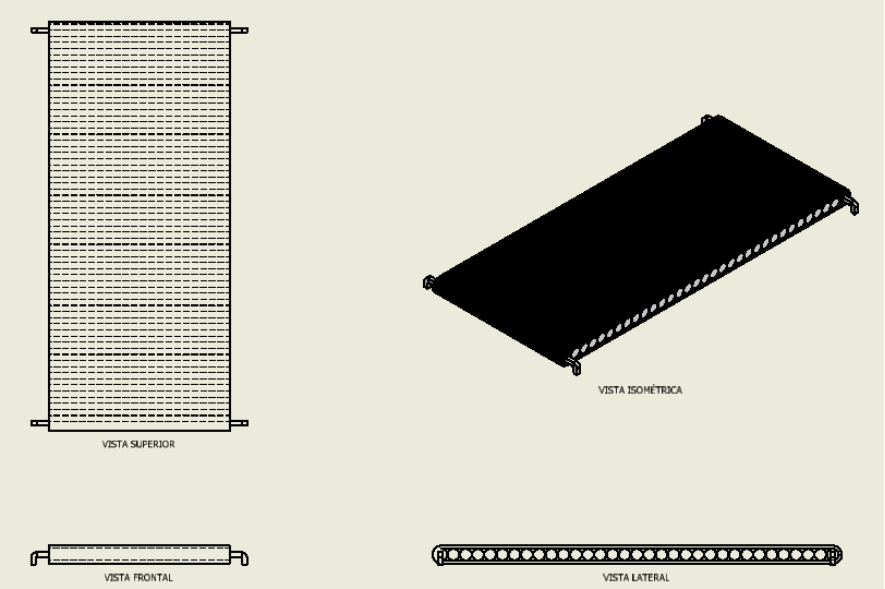


Ilustración 42. Cinta transportadora diseñada por el autor del proyecto.

4.2.6 Plataforma de sujeción robot paralelo

El robot paralelo, a diferencia de los demás que trabajan anclados al suelo en este proyecto, se encuentra sujeto a una plataforma que le permite trabajar en su posición natural, es decir, desde arriba.

Esta plataforma se compuso de diferentes vigas de acero cruzadas y a la vez fijadas a los pilares, aportando de esta forma estabilidad y fiabilidad a la hora de soportar el peso del robot.

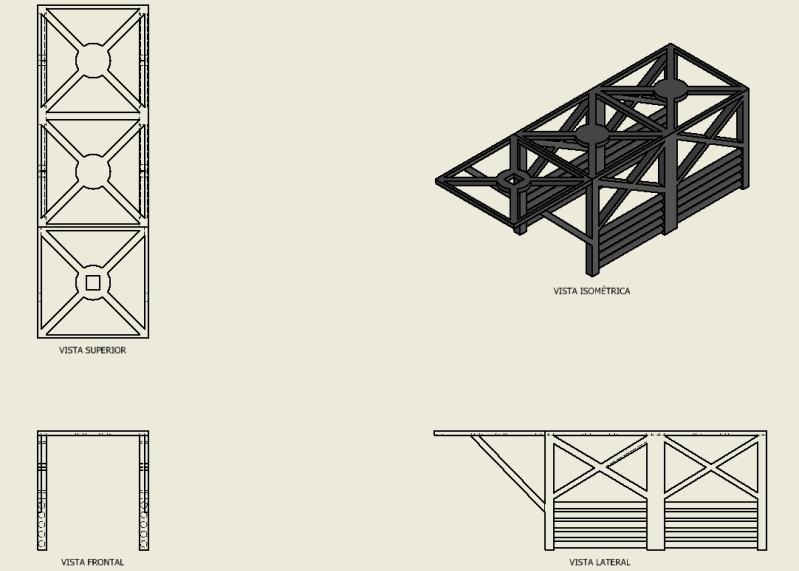


Ilustración 43. Plataforma de sujeción diseñada por el autor del proyecto.

4.2.7 Columna de señalización

Como se ha comentado al principio del capítulo actual, las instalaciones robotizadas deben estar equipadas con mecanismos que adviertan sobre el estado de la célula. Las condiciones del funcionamiento de la célula deben ser fácilmente reconocidas por cualquier operario, esto ayuda a que se pueda reaccionar ante perturbaciones de forma inmediata.

Como se puede observar, la columna de señalización diseñada se compone de cinco luces de diferentes colores, ya que cada una muestra un mensaje distinto.

Ilustración 44. Columna de señalización.

Como se puede observar en la Ilustración 44, también se ha diseñado un cartel para que cualquier tipo de persona pueda identificar el estado de la célula de forma inequívoca.

El material de recubrimiento de los diferentes leds es el plástico, mientras que la unión de cada luz y el soporte están hechos de aluminio.

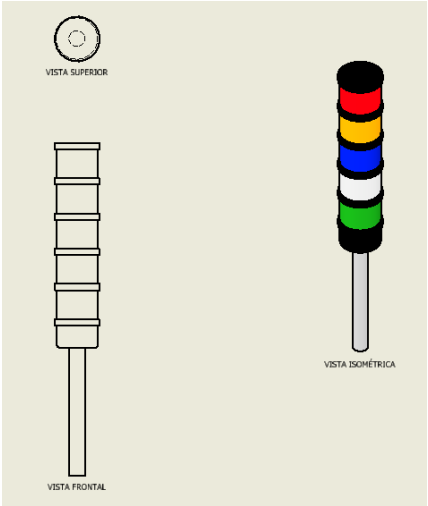


Ilustración 45. Columna de señalización diseñada por el autor del proyecto.

4.2.8 Setas de emergencia

Debido al riesgo que se tiene a la hora de trabajar instalaciones robotizadas, la célula debe estar equipada con interruptores de parada de emergencia (setas de emergencia) tanto en el interior, como en el exterior de la célula.

Es por ello que se han colocado varias setas de emergencia tanto en la zona de trabajo de los robots, como en el exterior de la célula.

En cuanto al diseño, se ha buscado que este sea vistoso con el objetivo de que pueda ser identificado fácilmente y de forma inequívoca por los operarios.

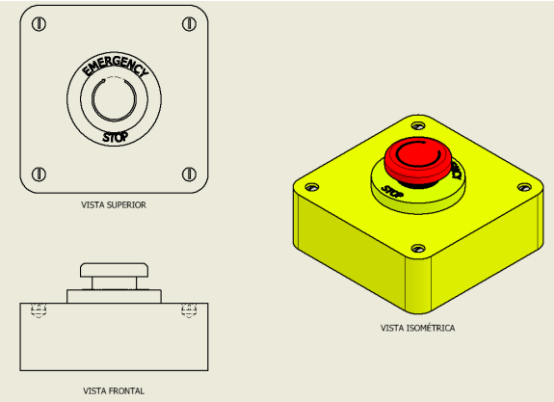


Ilustración 46. Seta de emergencia diseñada por el autor del proyecto.

4.2.9 Control de acceso

Para evitar la entrada de personal no autorizado en la célula, se debe proveer a esta de mecanismos de acceso, en este caso se ha decidido colocar en las dos puertas de acceso a la célula un dispositivo que permitirá abrir estas puertas al personal que cuente con la tarjeta de acceso.

El mecanismo acoplado en ambas puertas de acceso quedaría de la siguiente manera:

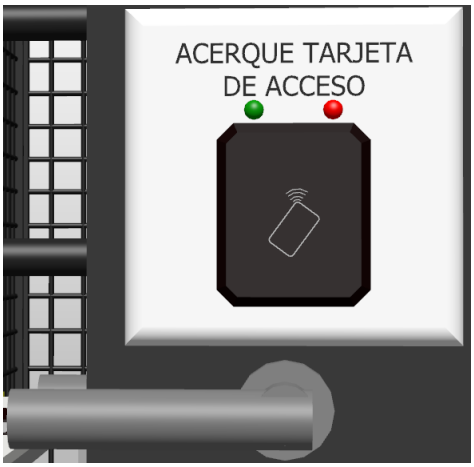


Ilustración 47. Mecanismo colocado en la puerta de acceso.

Como se puede comprobar este mecanismo está compuesto de dos leds que se iluminarán en función de si se permite el acceso (led verde) o si se deniega el acceso (led rojo). También cuenta con un lector de tarjetas sin contacto, en el cual el operario deberá acercar la tarjeta para poder acceder.

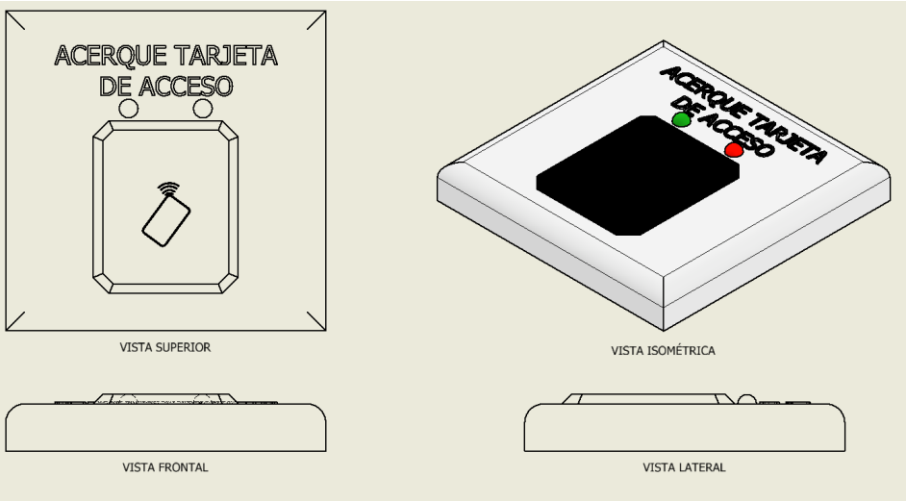


Ilustración 48. Control de acceso a la célula diseñada por el autor del proyecto.

4.2.10 Barreras protectoras

Para delimitar el área de trabajo de los robots se han creado barreras para impedir el acceso. Estas han sido diseñadas mediante una plataforma rectangular y hueca que en su interior se han colocado cilindros de forma horizontal y vertical.

Este diseño ofrece una alta seguridad y robustez a la vez que permite ver a través de la barrera lo que sucede en el interior de la célula, siendo esto último importante ya que ofrece al operario la posibilidad de controlar el proceso de una forma presencial. El material que se utilizó para el diseño de estas vallas es el acero.

Se diseñaron dos tipos de vallas, en lo único que se diferencian es en la dimensión.

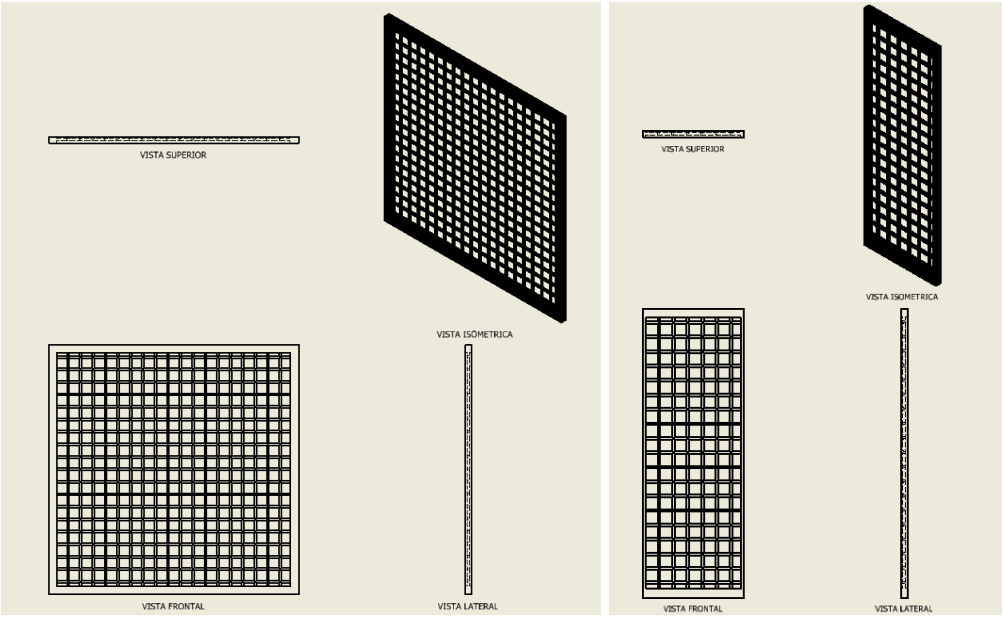


Ilustración 49. Barreras protectoras diseñadas por el autor del proyecto.

4.2.11 Barreras fotoeléctricas

En las zonas que no es posible colocar una barrera protectora, se han colocado barreras fotoeléctricas. Están dispuestas en la célula de manera que ninguna persona puede acceder al interior de esta sin haber activado el dispositivo y para que si se alcanza el espacio restringido se haya detenido parado antes el funcionamiento.

Estas barreras se colocan de dos en dos, siendo una de ellas el emisor y otra el receptor. La luz emitida desde el emisor se envía directamente al receptor, y cuando un objeto interrumpa el haz de luz entre el emisor y el receptor se active la señal de estado. Esto se explicará con más detalle en el capítulo 5.1.3 Programación de Smart components.

El diseño de estas se realizó de un color vistoso y con una forma que se pueda adaptar fácilmente a las vallas diseñadas. En el interior de estas se encuentran los sensores encargados de detectar los objetos que atraviesen el haz de luz creado entre emisor y receptor.

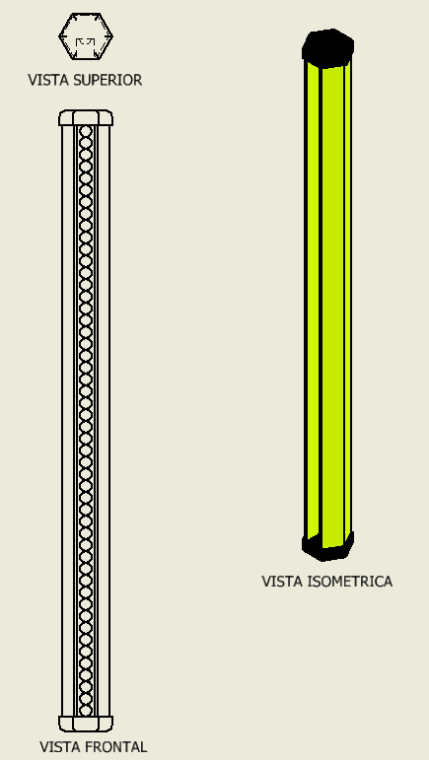


Ilustración 50. Barrera fotoeléctrica diseñada por el autor del proyecto.

4.2.12 Componentes proporcionados por RobotStudio

Los componentes mencionados anteriormente han sido diseñados en Autodesk Inventor, pero también se ha utilizado la librería que proporciona Robotstudio para añadir elementos que completen el diseño de la estación.

Se encuentran en la siguiente pestaña de la aplicación:

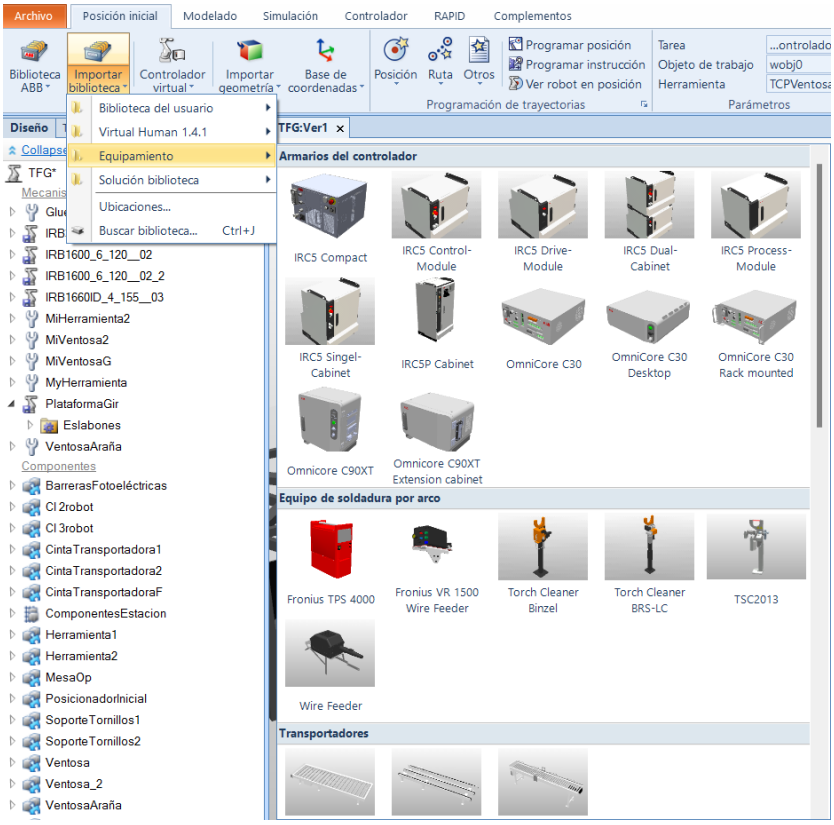


Ilustración 51. Biblioteca de componentes en Robotstudio.

Los elementos empleados de esta biblioteca son los siguientes:

- **Cintas transportadoras:** encargadas del transporte de las diferentes piezas necesarias para el montaje de la luminaria.

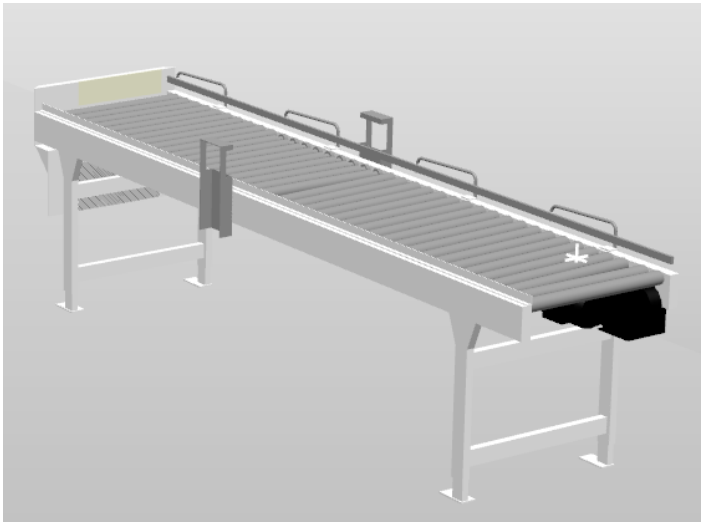


Ilustración 52. Cinta transportadora proporcionada por RobotStudio.

- **Controlador IRC5:** contiene los elementos electrónicos necesarios para controlar los diferentes robots y los equipos periféricos. Debido a que el proyecto solo se realiza en simulación (se emplea un controlador virtual), este elemento es puramente decorativo y se ha añadido para que la célula tenga el máximo parecido posible a la realidad.

Ilustración 53. Controlador IRC5 proporcionado por RobotStudio.

4.3 Robots utilizados en la célula robotizada

Una vez se han mostrado todos los componentes que forman la célula, se procede a comentar las características técnicas de cada robot empleado.

4.3.1 IRB 1600

En este proyecto se han utilizado dos brazos robóticos ABB IRB 1600, véase Ilustración 54, que poseen 6 ejes rotacionales. Las funciones predefinidas por el fabricante son la manipulación, ensamblaje de piezas, de empaque, de corte, de soldadura, distribución, etc. Las funciones que realizarán en este proyecto serán la manipulación de piezas, ensamblaje y fijación de piezas. [24]

Ilustración 54. IRB 1600. [24]

Este robot tiene un peso de 250 kg y la carga máxima que puede soportar es de 6kg, lo cual no va a suponer un problema debido a que en este proyecto las piezas no son pesadas por lo que nunca van a superar máximo. [24]

En la Ilustración 55, a la izquierda se pueden apreciar los diferentes ejes que tiene el robot y el sentido positivo y negativo de cada uno. A la derecha se muestran las dimensiones del ABB IRB 1600. [24]

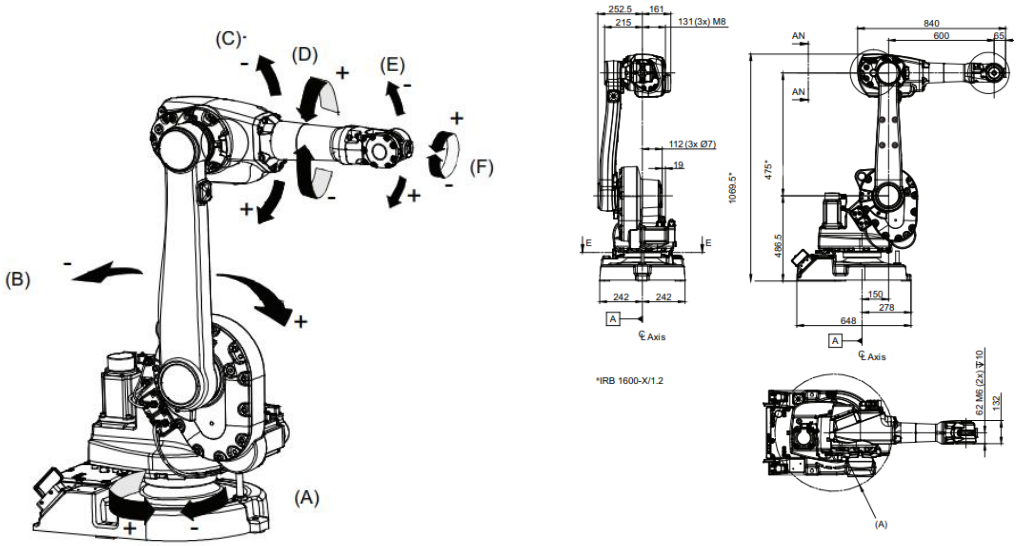


Ilustración 55. Ejes y dimensiones del IRB 1600. [24]

En la izquierda de la Ilustración 55, vemos que tenemos los parámetros A, B, C, D, E y F, que nos indican el eje correspondiente como se muestra en la siguiente tabla:

Pos	Descripción	Pos	Descripción
A	Eje 1	B	Eje 2
C	Eje 3	D	Eje 4
E	Eje 5	F	Eje 6

Tabla 8. Ejes del robot referenciados. [24]

A la derecha de la Ilustración 55, se tiene el parámetro A, que representa el radio mínimo de giro, cuyo valor es de 335 mm.

A continuación, se adjuntan los valores de la velocidad y rango de trabajo de cada eje:

IRB1600	Máxima velocidad eje (°/s)	Rango de trabajo del eje (°)
Eje 1	150	-180 a 180
Eje 2	160	-63 a 110
Eje 3	170	-235 a 55
Eje 4	320	-200 a 200
Eje 5	400	-115 a 115
Eje 6	460	-400 a 400

Tabla 9. Velocidad y rango de trabajo de cada eje del robot IRB1600. [24]

Como se puede observar en las siguientes ilustraciones, el área de trabajo de este robot es muy amplia, lo que nos permite la posibilidad de alcanzar un amplio rango de puntos.

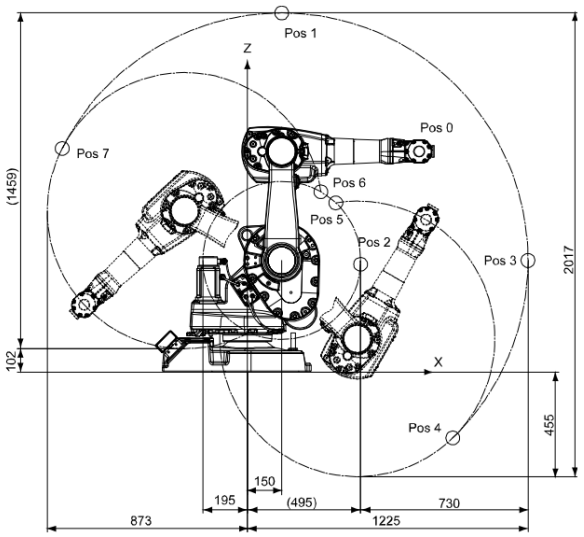


Ilustración 56. Área de trabajo IRB1600, vista lateral. [24]

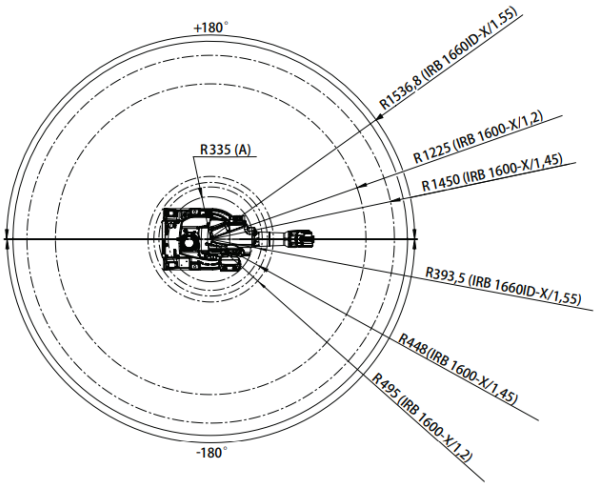


Ilustración 57. Área de trabajo de la familia IRB1600/1660, vista superior. [24]

Se muestran a continuación las especificaciones más importantes, así como el entorno de trabajo, conexiones, rendimiento, etc. Estas tablas han sido obtenidas del *datasheet* que proporciona ABB. [25]

Environment

Ambient temperature for mechanical unit	
During operation	+ 5°C (41°F) to + 45°C (113°F)
During transportation and storage	- 25°C (- 13°F) to + 55°C (131°F)
During short periods (max. 24 h)	up to + 70°C (158°F)
Relative humidity	Max. 95%
Safety	Double circiuts with supervisions, emergency stops and safety functions, 3-position enable device
Emission	EMC/EMI shielded

Specification

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

Tabla 10. A la izquierda, tabla del entorno de trabajo. A la derecha, tabla de especificaciones del IRB1600. [25]

Performance (according to ISO 9283)

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

—

Technical information

Electrical Connections	
Supply voltage	200-600 V, 50-60 Hz
Energy consumption	0.58 kW

Tabla 11. Datos del rendimiento del IRB1600. [25]

4.3.2 IRB 1660ID

El robot IRB 1660ID, véase la Ilustración 58, pertenece a la familia del robot descrito anteriormente, pero este resulta ideal para aplicaciones de soldadura al arco, servicio a máquinas, manejo de materiales, aplicación de adhesivo y fresado. Las funciones que realizará en este proyecto serán similares a los dos brazos robóticos mencionados anteriormente y a mayores realizará la aplicación de adhesivo sobre la luminaria. [26]

Ilustración 58. IRB 1660ID. [24]

A continuación, se muestran las dimensiones del IRB1660ID. Este modelo presenta un tamaño más grande que el anterior. Esto se eligió a consciencia debido a que se necesitaba un área de trabajo mayor.

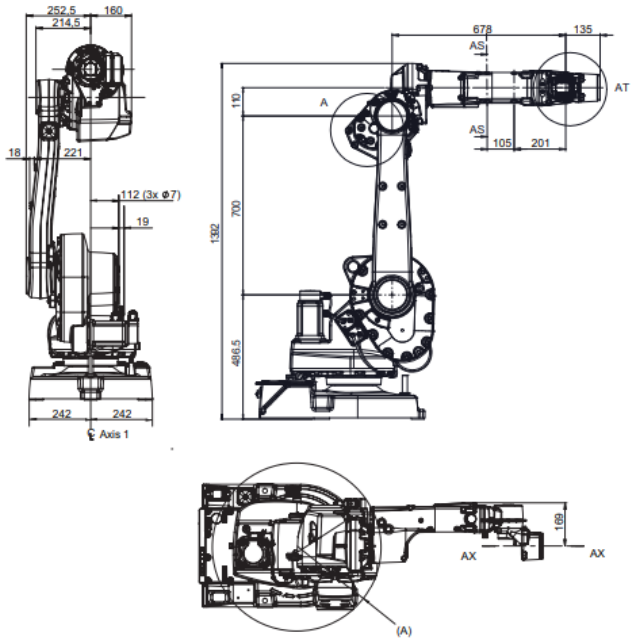


Ilustración 59. Dimensiones del IRB1660ID. [24]

Este robot tiene un peso de 257 kg y la carga máxima que puede soportar es de 6kg, como se ha comentado anteriormente esto no va a suponer un problema ya que las piezas que va a manejar no son muy pesadas. [26]

IRB1660ID	Máxima velocidad eje (°/s)	Rango de trabajo del eje (°)
Eje 1	180	-180 a 180
Eje 2	180	-90 a 150
Eje 3	180	-238 a 79
Eje 4	320	-175 a 175
Eje 5	360	-120 a 120
Eje 6	500	-400 a 400

Tabla 12. Velocidad y rango de trabajo de cada eje del robot IRB 1660ID. [26]

Del *datasheet* del fabricante se obtiene el área de trabajo del robot y se muestra en la siguiente imagen (Ilustración 60).

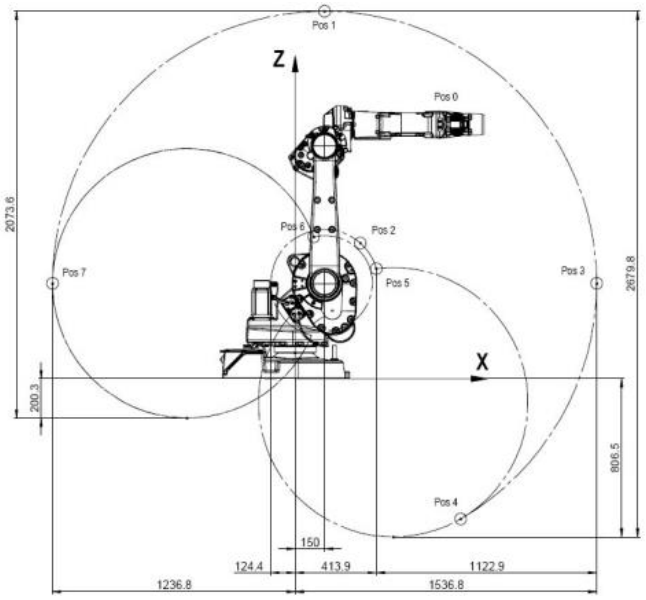


Ilustración 60. Área de trabajo IRB1660, vista lateral. [24]

Para visualizar la vista superior del área de trabajo del IRB1660ID véase la Ilustración 57.

Una vez han sido definidas las medidas, el volumen en el espacio de trabajo y demás características, se muestran las principales especificaciones y parámetros definidos en el *datasheet* que nos proporciona el fabricante, donde encontramos las siguientes tablas:

Specification

Robot version	Reach (m)	Payload (kg)	Armload (kg)
IRB 1660ID-6/1.55	1.55	6	15+15
IRB 1660ID-4/1.55	1.55	4	15+15
Number of axes	6		
Protection	IP67 (IP40 for axis 4)		
Mounting	Floor, tilted, inverted		
Controller	IRC5 Single Cabinet/IRC5 PMC Panel Mounted/IRC5 Compact		

Performance (according to ISO 9283)

	IRB 1660ID -6/1.55	IRB 1660ID -4/1.55
Path repeatability	0.05 mm	0.08 mm
Position repeatability	0.02 mm	0.02mm

Tabla 13. A la izquierda, tabla de especificaciones. A la derecha, tabla del rendimiento del IRB1660ID. [26]

Technical information

Electrical Connections	
Supply voltage	200-600 V, 50-60 Hz
Power consumption	0.62 kW, ISO-Cube at max. load and speed
Physical	
Robot base	484 x 648 mm
Robot height	1392 mm
Robot weight	257 kg
Environment	
Ambient temperature for mechanical unit	
During operation	+5° C (41° F) to + 45°C (113°F)
During transportation and storage	-25° C (-13° F) to +55° C (131° F)
During short periods (max. 24 h)	up to +70° C (158° F)
Relative humidity	Max. 95%
Noise level	< 70 dB(A)
Safety	Double circuits with supervisions, emergency stops and safety functions. 3-position enable device
Emission	EMC/EMI shielded

Data and dimensions may be changed without notice.

Tabla 14. Tabla de información técnica del IRB1660ID. [26]

4.3.3 IRB369C1

Este modelo pertenece a la clase de robots paralelos. Un robot paralelo es un mecanismo en el cual una plataforma móvil se encuentra unida a una base por varias cadenas cinemáticas independientes. Se trata de robots que son ligeros, rígidos y que poseen altas aceleraciones. Son muchas las aplicaciones de este tipo de robot. [27] A continuación, se citan algunas de ellas:

- Simuladores de vuelo.
- Posicionamiento de precisión.
- Transporte de objetos.
- Maquinado de piezas.
- Medicina.

En este proyecto solo se ha empleado para el transporte de objetos, concretamente es el encargado de trasladar la luminaria desde la plataforma giratoria hasta una cinta transportadora. [28]

A continuación, se muestra el área de trabajo del robot:

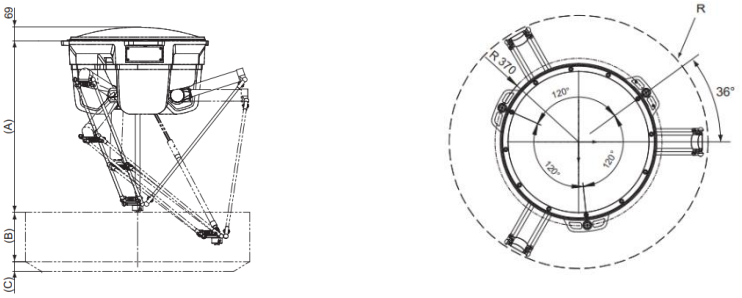


Ilustración 61. Área de trabajo del robot paralelo. [29]

Donde los parámetros mostrados en la Ilustración 61 tienen los siguientes valores [29]:

- A = 960mm.
- B = 200mm.
- C = 0mm.
- R= 400mm.

También se muestra la siguiente tabla donde se muestra la precisión, repetibilidad y estabilidad que tiene el robot con diferentes cargas [29]:

IRB 360-1/1130, IRB 360-3/1130 and IRB 360-8/1130	At 0.1 kg	At 1.0 kg	At 3.0 kg	At 8.0 kg
Pose accuracy, AP (mm)	0.01	0.04	0.10	0.04
Pose repeatability, RP (mm)	0.10	0.09	0.06	0.07
Pose stabilization time, Pst (s) within 0.2 mm of the position	i	0.03	0.05	0.05
Path accuracy, AT (mm)	0.51	0.52	1.00	2.32
Path repeatability, RT (mm)	0.30	0.21	0.14	0.10

Tabla 15. Tabla del rendimiento del robot paralelo. [29]

4.4 Diseño final

Una vez han sido mostrados y explicados todos los componentes que han sido diseñados y los robots que se han empleado, se procede a mostrar cómo se han distribuido todos y cada uno de ellos para la implementación de la célula robotizada.

A continuación, se mostrarán diferentes imágenes, con distintas perspectivas con el fin de mostrar los detalles de la célula diseñada.

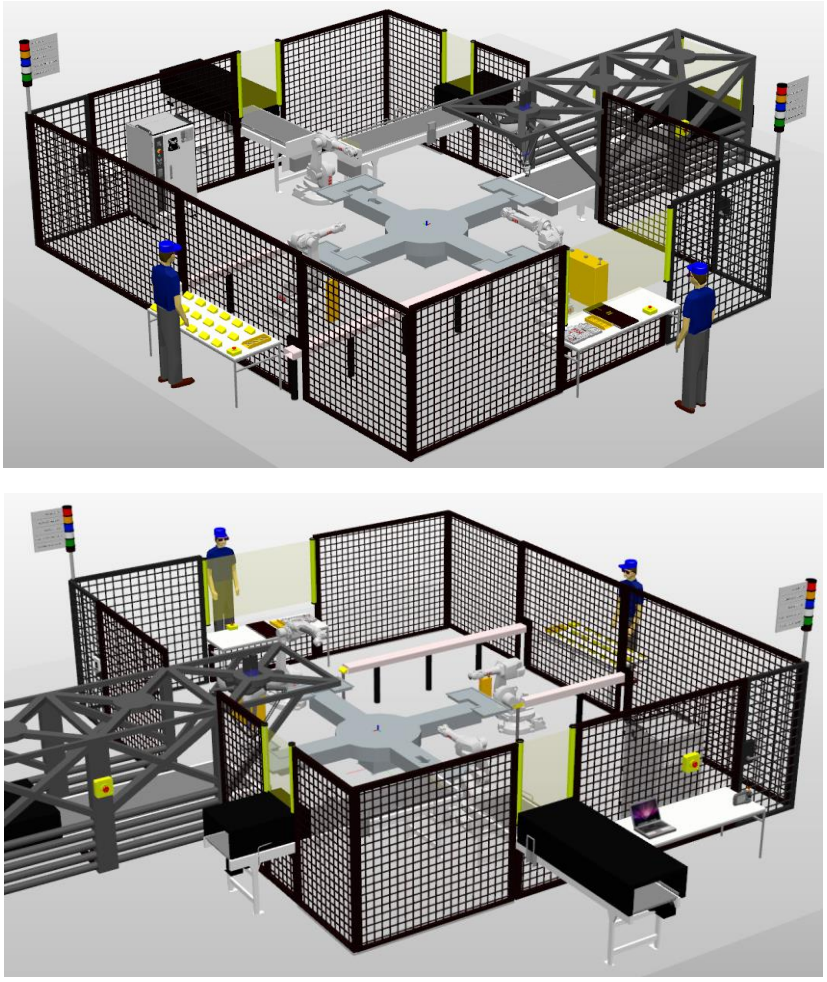


Ilustración 62. Diferentes perspectivas de la célula de trabajo diseñada.

El puesto de trabajo del operario se muestra en la siguiente ilustración. Como se puede observar, se han colocado barreras fotoeléctricas que, por seguridad, cuando detecten cualquier cuerpo en ellas pararán la célula de trabajo.

Ilustración 63. Puesto de trabajo del operario.

4.5 Herramientas y piezas utilizadas en la estación

Una vez han sido mostrados todos los elementos necesarios para la creación de la célula robotizada, se va a mostrar el diseño de la luminaria y el de las herramientas que utilizan los robots para completar el montaje.

4.5.1 Luminaria

A continuación, se van a comentar todos los componentes que forman la luminaria.

Base de la luminaria

El material del que se compone la luminaria es el aluminio. Uno de los factores que se tuvieron en cuenta en el diseño es la acumulación del calor dentro de una luminaria. Por ello se crearon canales en la parte de debajo de la base para ayudar a la evacuación del calor. Teniendo en cuenta esto y que el material del que está hecho es el aluminio, cuyo coeficiente de conductividad térmica es de 209,3 W/(m·K), se le otorga una gran disipación térmica a la luminaria. [30]

Esta parte de la luminaria deberá tener los mecanizados correspondientes para que se puedan fijar en ella la PCB, el *driver*, la caja de control y el propio marco.

También cuenta en la base con un mecanizado que permite la entrada de una manguera para que sea posible la alimentación de la luminaria, este es señalado en color rojo en la Ilustración 64.

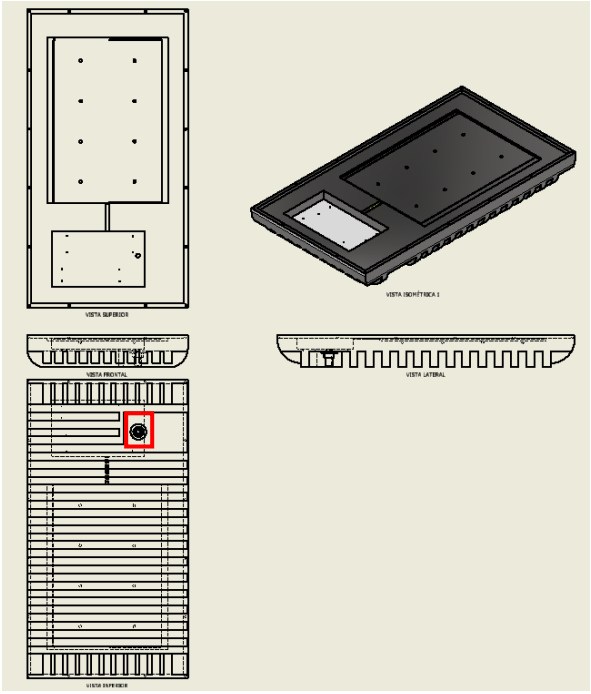


Ilustración 64. Base de la luminaria diseñada por el autor del proyecto.

Marco

Se muestra en la Ilustración 65 lo que sería la parte de arriba de la luminaria. Esta cuenta con mecanizados que permiten la fijación a la base y con un hueco en el medio, donde iría colocado el cristal.

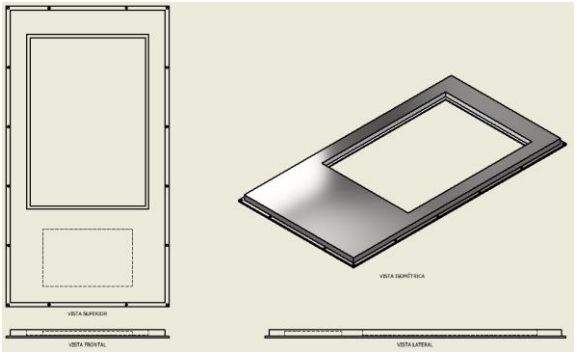


Ilustración 65. Marco de la luminaria diseñado por el autor del proyecto.

Driver

Diseño simple de una caja de metal con la que se pretende simular la parte de la luminaria encargada de la alimentación de la placa de leds (PCB). El *driver* iría fijado a la luminaria como se muestra en la Ilustración 74.

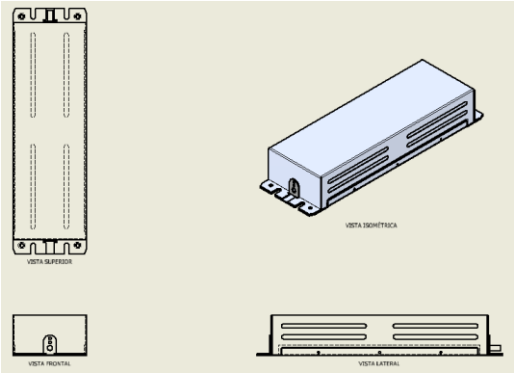


Ilustración 66. Driver de la luminaria.

Caja de control

Diseño muy parecido al anterior, pero este simula la parte de electrónica que llevaría una luminaria, es decir, controlaría el encendido y apagado, la intensidad que se suministra a los leds fijando la cantidad de luz que proporcionan, etc. Se muestra en la Ilustración 74 la posición que ocuparía en la luminaria.

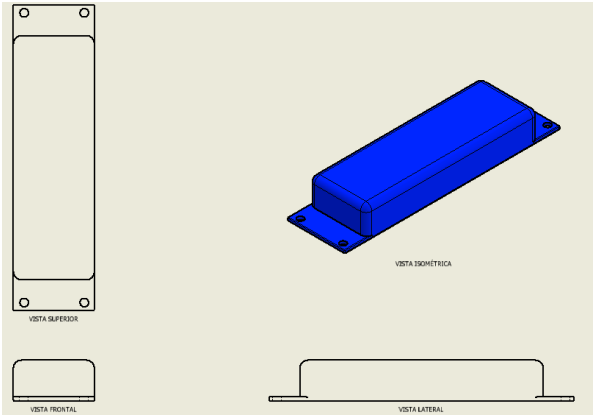


Ilustración 67. Caja de control diseñada por el autor del proyecto.

Placa de leds

A continuación, se muestra la PCB, que es el elemento encargado de la iluminación de la luminaria gracias a los 32 leds que posee. Este elemento está formado por una base en la que se han colocado 8 placas como la que se muestra en la Ilustración 68 marcada en color rojo. Como se puede observar está dotada de terminales que marcan la alimentación positiva y negativa, que es un factor a tener en cuenta a la hora de realizar la conexión.

La ubicación de esta en la luminaria se muestra en la Ilustración 73.

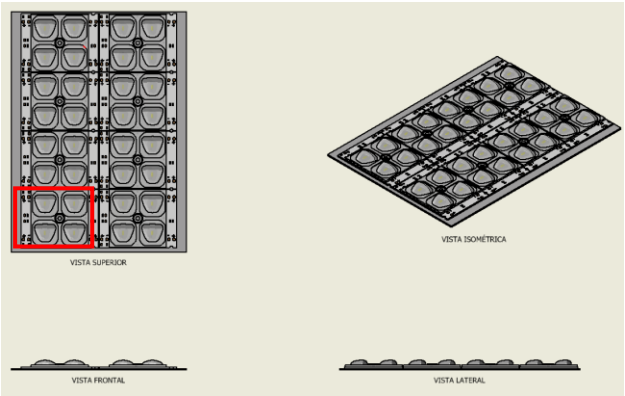


Ilustración 68. Placa de leds de la luminaria.

Cristal

Último elemento que compone la luminaria. La creación de este es muy simple, se trata de una placa rectangular al que se le asigna un material transparente (véase la Ilustración 69). La función de este es cerrar los departamentos ópticos de la luminaria, de manera que la luminaria quede protegida y garantizar la limpieza del reflector de elementos como el agua, polvo, etc.

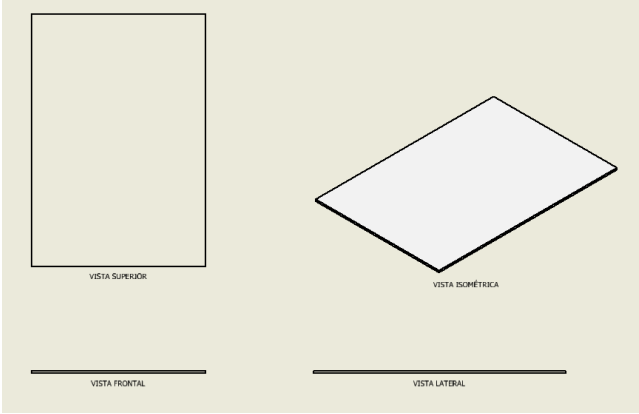


Ilustración 69. Cristal de la luminaria diseñado por el autor del proyecto.

A la hora de importar este objeto a Robotstudio no conserva las propiedades de transparencia que se le asignan en Autodesk Inventor.

Esta propiedad se consigue seleccionando el objeto en Robotstudio y nos saldrán las opciones que se ven en la Ilustración 70:

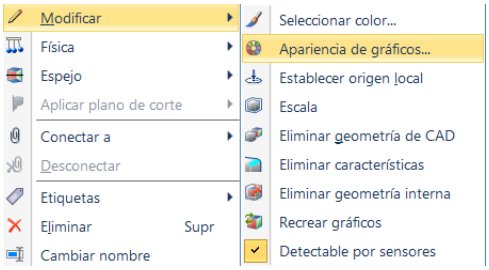


Ilustración 70. Apariencia de gráficos.

Seleccionaremos “Apariencia de gráficos” y se nos abrirá la siguiente ventana (Ilustración 71), donde daremos un valor del 60% a la opacidad del objeto.

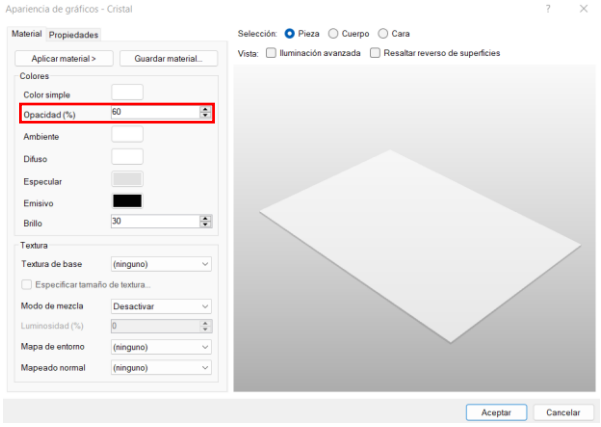


Ilustración 71. Seleccionar opacidad del objeto.

Tornillería

La función de estos es la fijación de los componentes. Se distinguen tres tipos de tornillos: para fijar el marco a la base, para fijar la PCB y para fijar el *driver* y la caja de control. De estos, solo se ha adjuntado el plano de uno de ellos ya que lo único en lo que difieren es en el tamaño del cuerpo.

El cabezal del tornillo tiene forma de hexágono y se ha diseñado para que este coincida perfectamente con la herramienta encargada de fijar todos los componentes, véase la Ilustración 77.

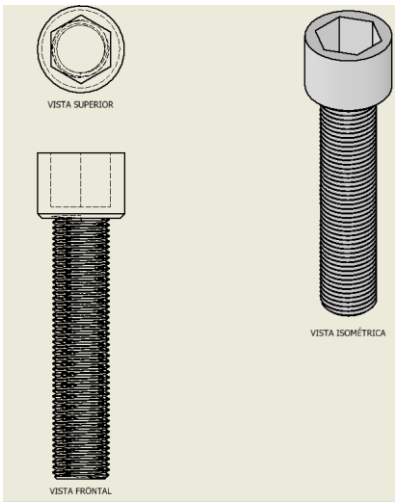


Ilustración 72. Tornillo de cabeza Allen diseñado por el autor del proyecto.

Diseño final de la luminaria

Una vez comentados todos los elementos que componen la luminaria se muestra a continuación el diseño completo en las siguientes ilustraciones.

Ilustración 73. Diseño completo de la luminaria diseñada por el autor del proyecto.

En la Ilustración 74 se ha suprimido el marco superior para poder apreciar la posición que ocupa cada elemento en el interior de la luminaria.

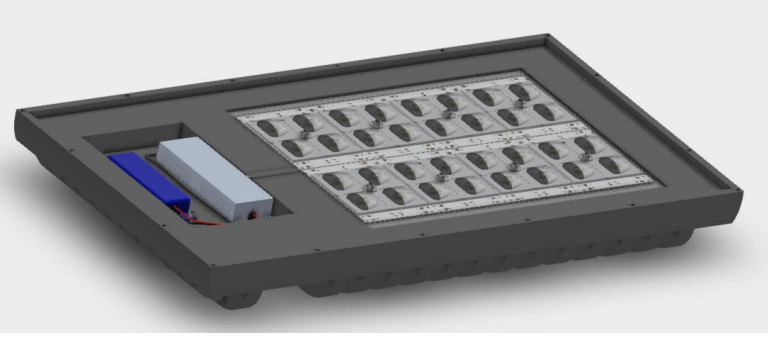


Ilustración 74. Luminaria sin el marco superior.

Como se muestra en la Ilustración 75, también se han tenido en cuenta las conexiones para la alimentación de la luminaria. Señalado en rojo se muestra la extrusión realizada para conectar el *driver* a la PCB

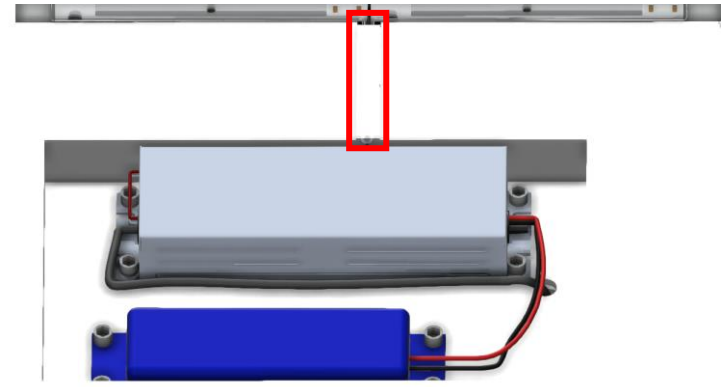


Ilustración 75. Detalle del cableado realizado en la luminaria.

4.5.2 Herramientas de los robots

A continuación, se van a mostrar el diseño de las herramientas que emplean los robots para completar el proceso de montaje.

Ventosa

Se decidió usar ventosas en vez de pinzas debido a la diferencia de tamaños existentes entre las piezas, ya que la ventosa permite al robot coger todo tipo de elementos sin importar el tamaño y con una pinza esto no sería posible. La función de esta es dar capacidad al robot para realizar traslaciones y poder colocar las piezas.

La función de agarre de las piezas debemos programarla mediante componentes inteligentes en Robotstudio, esto se explicará en el capítulo 5.1.3 Programación de Smart components.

El diseño de la ventosa se muestra en la Ilustración 76.

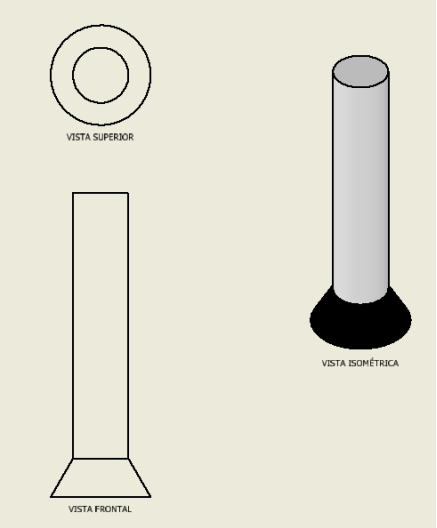


Ilustración 76. Herramienta del robot diseñada por el autor del proyecto.

Herramienta fijadora de tornillos

Para poder fijar los componentes con los tornillos descritos anteriormente se necesitaba el diseño de una herramienta que se muestra en la Ilustración 77. Está compuesta por una base (señalada con una flecha de color azul) que la permite adherirse al extremo del robot.

También cuenta con una pieza (señalada con la flecha roja), que permite que sea colocada por el robot en el soporte de herramientas mostrado en la Ilustración 39.

En el extremo de la herramienta se puede apreciar la forma hexagonal que coincide perfectamente con el tipo de tornillería mostrado anteriormente (véase Ilustración 72).

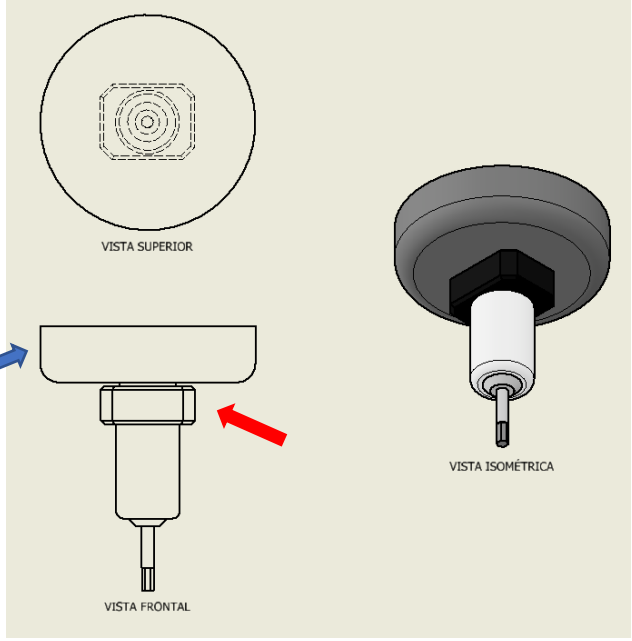


Ilustración 77. Herramienta del robot diseñada por el autor del proyecto.

Herramienta fijadora del cristal

El estilo de esta herramienta es prácticamente el mismo que el anterior, en lo único que difieren es en el extremo de la herramienta, que en este caso es redondo y con el que se pretende simular un dispensador silicona para fijar el cristal al marco de la luminaria.

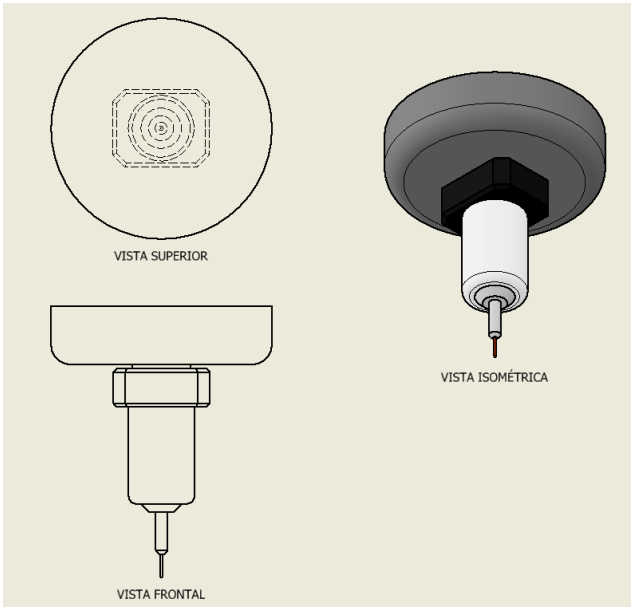


Ilustración 78. Herramienta del robot diseñada por el autor del proyecto.

5. Desarrollo y programación de la célula robótica

Como se ha comentado anteriormente, para la realización de las tareas se han usado los *softwares* Matlab y Robotstudio, que permiten realizar la simulación. En los siguientes capítulos se procede a explicar todas las tareas programadas.

5.1 Desarrollo en Robotstudio

Comenzaremos con el *software* donde se programan los robots utilizados, Robotstudio. En este capítulo hablaremos sobre la creación de la mesa giratoria, la programación en RAPID y de los Smart Components.

5.1.1 Creación de mecanismo

Como se ha comentado en el capítulo anterior, la mesa giratoria cuando se exporta desde Autodesk Inventor a Robotstudio no tiene la propiedad necesaria para poder girar. Asignar esta propiedad a este componente es posible gracias a la opción de crear mecanismos en Robotstudio.

Primero, en la pestaña “Modelado” debemos pulsar en la opción “Crear mecanismo”. Una vez hecho esto nos saldrá la siguiente ventana, donde tendremos que completar todos los campos para poder crear el mecanismo correctamente Aquí se le podrá dar un nombre a nuestro mecanismo, en el caso actual se le ha llamado “Plataforma Giratoria”. En tipo de mecanismo seleccionamos la opción por defecto “Robot”. A continuación, debemos especificar los eslabones que tendrá el mecanismo, obligatoriamente se debe especificar un eslabón base.

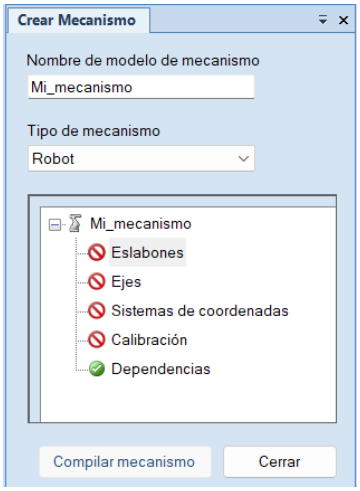


Ilustración 79. Configurar mecanismo.

Pulsamos en “Eslabones” y se nos abre la ventana mostrada en la Ilustración 80. En este caso se han declarado dos eslabones, el eslabón base, donde se encuentra el motor y la electrónica, que carecerá de movimiento y otro eslabón que será el que pueda girar. Para configurar cada eslabón primero se le debe dar un nombre y después elegir la pieza en “Componente seleccionado” a la que se va a asignar este eslabón. En caso de que el eslabón sea la base se debe marcar la opción “Establecer como eslabón base” señalada en rojo en la Ilustración 80. Después se debe añadir el eslabón pulsando en la flecha señalada en azul en la.

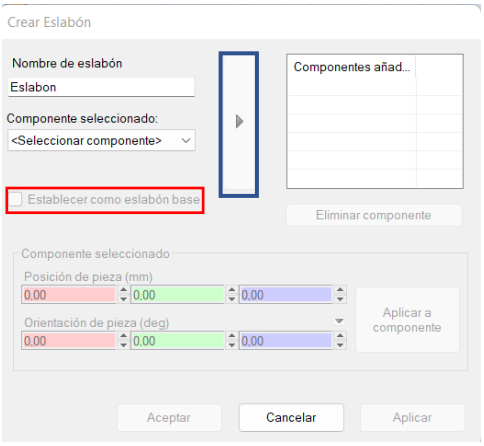
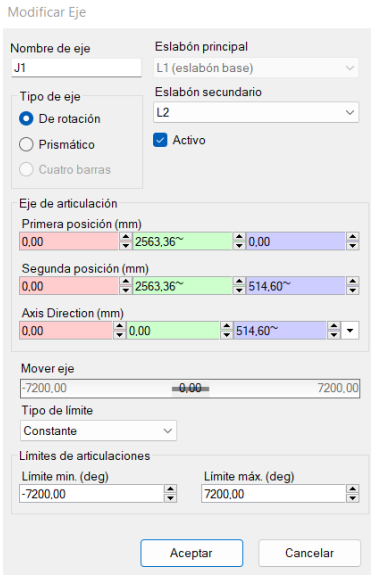


Ilustración 80. Configurar eslabón.

Una vez configurados los eslabones, se procede a configurar los ejes. Para ello, en la Ilustración 81 se muestra el apartado “Ejes”, que habría que configurarlo como se muestra a la izquierda de la siguiente imagen.



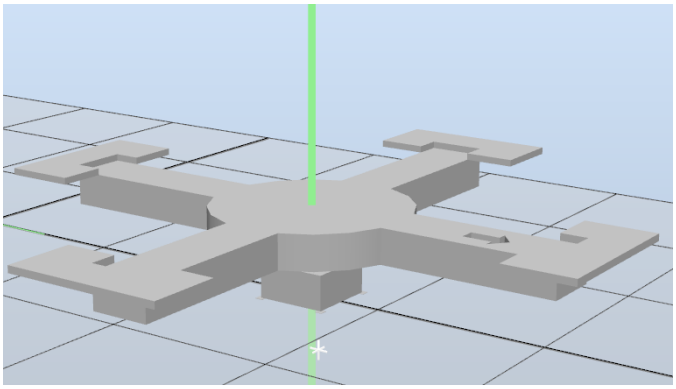


Ilustración 81. Configuración de los ejes del mecanismo.

El tipo de eje se debe configurar en modo de rotación. Después se debe configurar la posición del eje de la articulación, este ha sido configurado en el medio de la plataforma para que esta girase en torno a él. Para ello el primer punto se ha cogido en el centro de la base y el segundo punto en el centro de la parte de arriba de la plataforma, formándose el eje que se muestra en color verde en la parte de la derecha de la Ilustración 81.

El siguiente parámetro es fijar el límite de la articulación, este se configuró con un valor elevado para que nunca llegase al límite. Se fijó para que fuese de [-7200° a 7200°].

El tercer paso se trata de generar un sistema de coordenadas del mecanismo. Primero se le debe dar un nombre al sistema de coordenadas, en este caso se le llamó “Coordenadas_Plataforma”. El sistema de coordenadas era referido al segundo eslabón por lo que se posicionó en el centro de la parte de arriba de la plataforma.

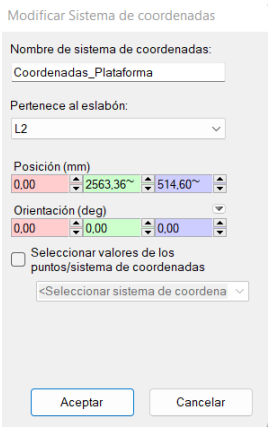


Ilustración 82. Sistema de coordenadas del mecanismo.

El cuarto paso es calibrar el eje para aumentar la precisión. Básicamente es repetir el segundo paso para ver si eje que se programa ahora coincide con el eje anteriormente programado.

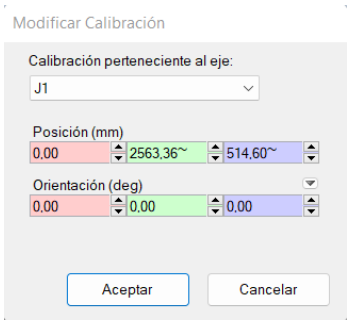


Ilustración 83. Calibración del eje del mecanismo.

Una vez configurados todos estos parámetros, faltaría definir las poses de la mesa giratoria. Estas poses van a ser configuradas en 0º, 90º, 180º, 270º y 360º. Estas se muestran en la Ilustración 84.

Poses

Nombre ...	Valores de pose
Pose inicial	[0,00]
Pose1	[90,00]
Pose2	[180,00]
Pose3	[270,00]
Pose4	[360,00]

Ilustración 84. Poses del mecanismo.

Para entender mejor este concepto de las distintas poses, se muestra a continuación en la Ilustración 85 la mesa giratoria en todas las poses definidas.

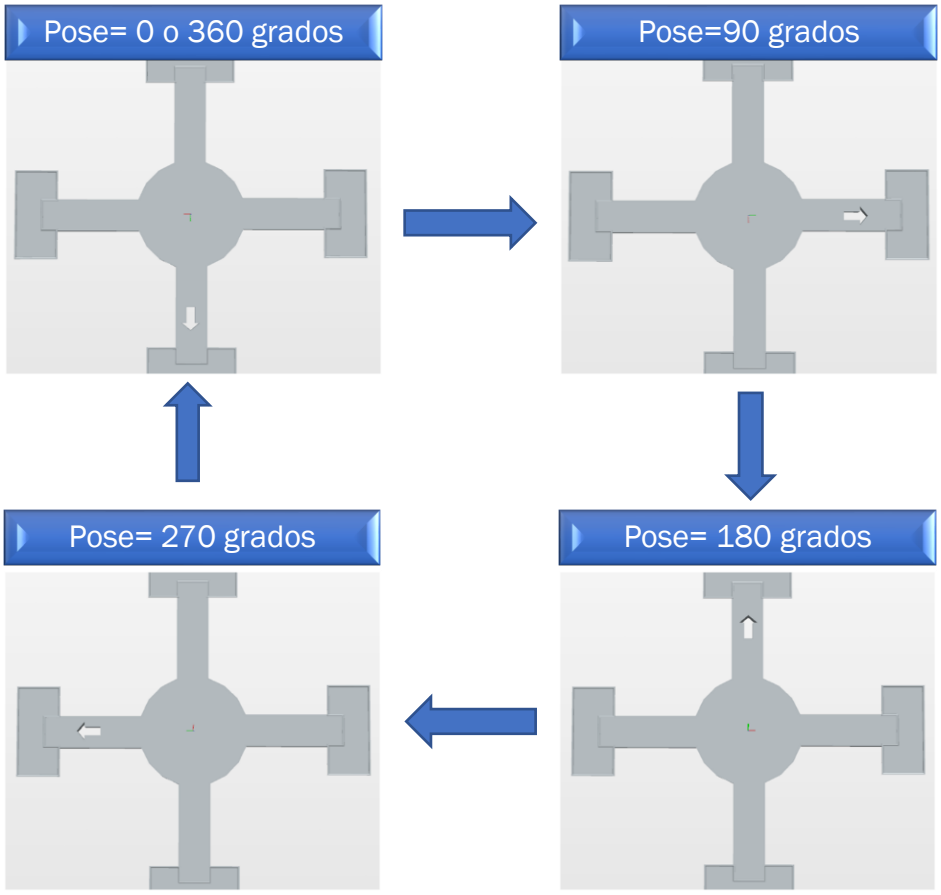


Ilustración 85. Demostración gráfica de las poses del mecanismo.

Con todo lo explicado anteriormente quedaría configurado el mecanismo completamente. En un capítulo situado más adelante se explicará cómo se realiza la programación mediante *Smart Components* para que este giré automáticamente.

5.1.2 Programación en RAPID

En el capítulo 3.2 Célula robótica propuesta, se realizó un diagrama de flujo donde se explicaban todos los pasos necesarios que debe hacer cada robot para el montaje completo de la luminaria. En este capítulo, se va a explicar cómo se ha hecho la programación de cada robot para poder realizar cada tarea.

Como se trata de una célula de cuatro robots, cada uno con una tarea diferente se van a necesitar cuatro códigos diferentes para controlar cada robot. Es importante destacar que todos los códigos van a estar relacionados entre sí, debido a que todas las tareas deben estar sincronizadas. A continuación, se van a explicar las funciones más importantes del código de cada tarea del robot.

Para poder entender correctamente todos los pasos realizados en la programación se van a utilizar diagramas de flujo siguiendo la simbología de la Ilustración 12.

Tarea 1, robot IRB 1600

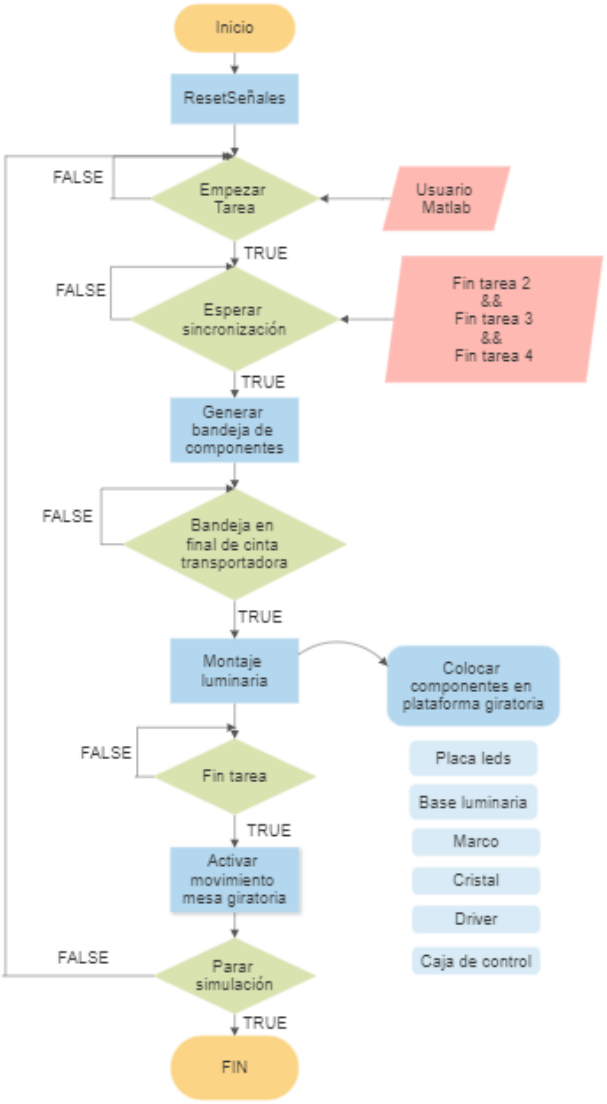


Ilustración 86. Diagrama de flujo correspondiente a la tarea del robot 1.

Una vez se ha mostrado el diagrama de flujo de la tarea 1, se procede a explicar la programación en el módulo de RAPID. Lo primero que hay que hacer a la hora de realizar un programa en este *software* es declarar las variables que van a ser utilizadas. Se distinguen tres tipos de variables:

- **Constantes (Const):** su valor no cambia durante la simulación.
- **Variables (Var):** su valor puede cambiar durante la simulación.
- **Persistentes (Pers):** deben ser inicializadas y si ese valor cambia en el programa, este se queda guardado de forma que no se pierde al finalizar.

Las variables usadas en este proyecto se muestran a continuación:

- **Tooldata:** un *tooldata* define la posición y orientación del TCP de la herramienta que vaya a tener el brazo robótico. En esta tarea solo se va a emplear una herramienta, que es una ventosa, por lo que solo se va a tener una variable de este tipo.

```
!Variable tooldata
PERS tooldata TCPVentosa:=[TRUE,[[0,0,54.763],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
```

Ilustración 87. Tooldata robot 1.

El *tooldata* de la herramienta ventosa en la estación de trabajo, se señala en la siguiente ilustración.

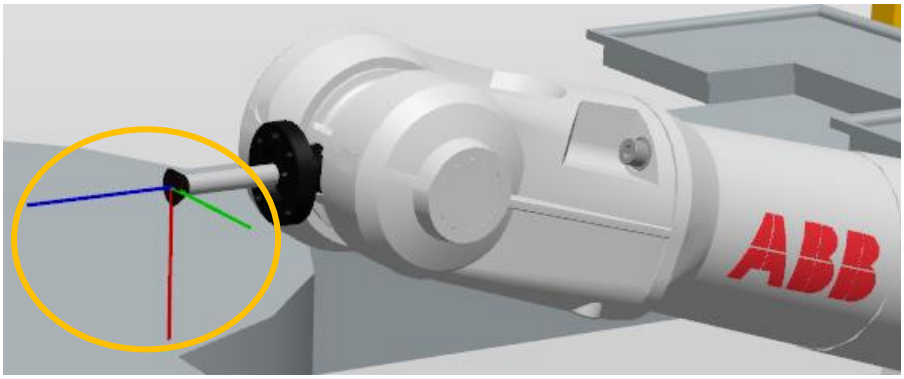


Ilustración 88. Tooldata de la herramienta en la estación de trabajo.

- **Robtarget:** un robtarget se utiliza para definir la posición (X, Y, Z) y la orientación (cuaternio: q1, q2, q3, q4). En este proyecto se han usado con el objetivo definir todas las posiciones a las que el robot debe llegar para realizar correctamente el montaje. En la siguiente ilustración se muestran algunos ejemplos de posiciones definidas.

```
!Variables robtarget
CONST robtarget AbajoLum:=[[-74.223,730.614,685.66],[0,0.707106781,0.707106781,0],[1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget ArribaLum:=[[97.966,732.494,702.251],[0,0.707106781,0.707106781,0],[0,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget BaseCaja:=[[-25.559,880.359,673.5],[0,0.707106781,0.707106781,0],[1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget CajaControl:=[[-25.339,968.947,677.17],[0,0.707106781,0.707106781,0],[1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Cristal:=[[89.532,1029.725,667.88],[0,0.707106781,0.707106781,0],[0,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Driver:=[[-24.06,1089.711,678.476],[0,0.707106781,0.707106781,0],[1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget PlacaLeds:=[[-131.769,1029.473,670.196],[0,0.707106781,0.707106781,0],[1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget PosIni:=[[806.295852188,0,929.002986412],[0.5,0,0.866025404,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Ilustración 89. Robtarget robot 1.

Para entender mejor este concepto se expone el siguiente ejemplo, en la Ilustración 90 se muestra un Robtarget en la estación de trabajo y en la Ilustración 91 se muestra al robot en la posición definida por este robtarget.

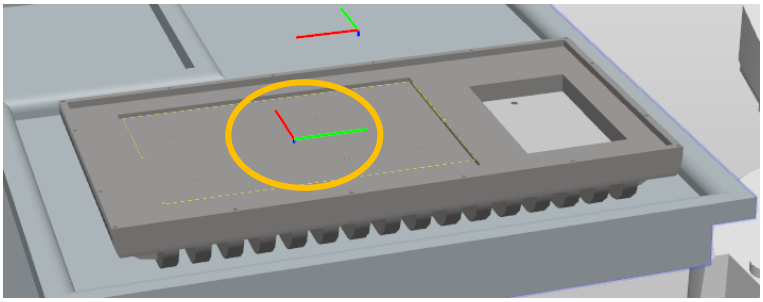


Ilustración 90. Robtarget en estación de trabajo.

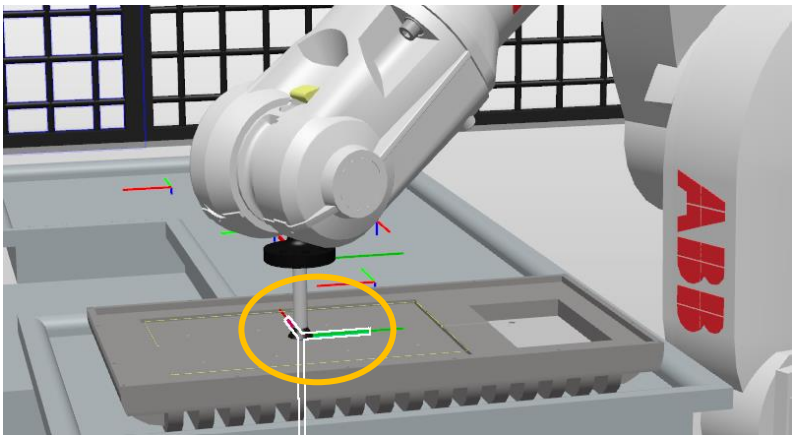


Ilustración 91. Robot en una posición definida por un robtarget.

- **Task:** con esta variable se declaran tareas del controlador. Esta variable nos será útil a la hora de sincronizar las tareas que controlan los movimientos de los robots.

- **Syncident:** se debe especificar el nombre del punto de sincronización. Se utiliza para identificar un punto de programa en el que la tarea del programa actual esperará a que las tareas de programa que se ejecuten en paralelo alcancen el mismo punto de sincronización. El nombre del tipo syncident debe ser el mismo en todas las tareas de programa cooperantes.

```
!Variables sincronizacion
PERS tasks Tarea1{2}:=[["T_ROB1"],["T_ROB2"]];
PERS tasks Tarea2{3}:=[["T_ROB1"],["T_ROB2"],["T_ROB3"]];
PERS tasks Tarea3{4}:=[["T_ROB1"],["T_ROB2"],["T_ROB3"],["T_ROB4"]];
PERS tasks TareaF{4}:=[["T_ROB1"],["T_ROB2"],["T_ROB3"],["T_ROB4"]];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
VAR syncident sync4;
VAR syncident sync5;
```

Ilustración 92. Variables necesarias para la sincronización.

- **Num:** variables del tipo numérico. Estas se han utilizado para crear contadores y condiciones.

```
!Variables numéricas
VAR num cond:=0;
VAR num cont:=0;
```

Ilustración 93. Variables numéricas.

- **Intnum:** se usa para identificar una interrupción. Esta variable ha sido utilizada en una función empleada para la simulación de una parada de emergencia.

```
!Variables interrupciones
Local VAR intnum Istop;
Local VAR intnum Parada;
```

Ilustración 94. Variable interrupción.

Una vez se han definido las variables empleadas en este módulo, se comentarán los procedimientos que han sido empleados. El programa se estructura de la siguiente manera, tenemos varios procedimientos secundarios que estos son llamados desde el procedimiento principal (main). Este último es mostrado en la Ilustración 95 y sigue la estructura del diagrama de flujo explicado en la Ilustración 86.

Main

El procedimiento empieza mandando al robot a su posición inicial (por si este estuviese descolocado) con la función 'MoveJ' (movimiento cómodo), después se hace una llamada al procedimiento 'ResetSeñales' cuyo objetivo es dar valor 0 a todas las señales utilizadas en el programa empleando la función 'Reset'. Con 'WaitDO' el puntero del programa no avanzará hasta que el valor de la señal 'Empezar' cambie de 0 a 1. El valor de esta señal cambiará cuando el usuario de Matlab lo indique desde la interfaz. Este valor será transmitido a Robotstudio a través de OPC.

Una vez el usuario ha decidido empezar el programa, se entra en un bucle infinito donde primero, se hace una llamada al procedimiento 'Interrupciones' que es el encargado de simular la parada de emergencia y después se tienen una serie de sentencias condicionales 'IF' y 'ELSEIF' con las que se programan los diferentes estados de la simulación. Estos estados se citan a continuación:

- **Estado 0 (IF cont=0):** llegada de piezas al sistema por primera vez, solo trabaja el robot 1. Los demás robots están ociosos, esperando la llegada de piezas a su posición.
- **Estado 1 (ELSEIF cont=1):** llegada de piezas a la posición dos, los robots 1 y 2 trabajan de forma sincronizada. Los robots 3 y 4 están ociosos esperando la llegada de piezas.
- **Estado 2 (ELSEIF cont=2):** llegada de piezas a la posición 3, los robots 1, 2 y 3 trabajan de forma sincronizada. El robot 4 está ocioso esperando la llegada de piezas.
- **Estado 3 (ELSEIF cont=3):** llegada de piezas a la posición 4, todos los robots trabajan de forma sincronizada.
- **Estado general (ELSEIF cont>=4):** a partir de este momento, se entra en otro bucle infinito en el que trabajan los cuatro robots a la vez ya que van a estar abastecidos de piezas constantemente.

Se consigue pasar de un estado a otro gracias a la creación de un contador que se incrementa cada vez que el robot termina su tarea.

```
PROC main()
    MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
    ResetSeñales;
    WaitDO Empezar,1;
    WHILE TRUE DO
        waittime 0.5;
        Interrupciones;
        IF cont=0 THEN
            Estado0;
        ELSEIF cont=1 THEN
            Estado1;
        ELSEIF cont=2 THEN
            Estado2;
        ELSEIF cont=3 THEN
            Estado3;
        ELSEIF cont>=4 THEN
            WHILE TRUE do
                PrimerPaso;
                SegundoPaso;
                TercerPaso;
                CuartoPaso;
            ENDWHILE
        ENDIF
        cont:=cont+1;
    ENDWHILE
ENDPROC
```

Ilustración 95. Procedimiento main tarea 1.

Estados tarea 1

Debido a que la estructura de programación de los estados es prácticamente igual, no se van a mostrar todos.

El funcionamiento del procedimiento que se muestra en la Ilustración 96 es el siguiente, primero se activa una señal 's5' que será la encargada de generar una bandeja con las piezas necesarias para el montaje. Esta bandeja avanzará por la cinta transportadora y cuando llegue al final de la cinta activará

la señal 'FCcinta1'. Con 'waitDi' se consigue que el puntero del programa no avance hasta que esta señal sea 1. Cuando la bandeja con las piezas llegue al final de la cinta se llamará al proceso 'MovimientoRobot1' encargado de colocar las piezas en la mesa giratoria. Cuando este procedimiento acabe se esperará a que el robot esté completamente parado (*Waitrob\Zerospeed*) y se activará la señal 's1' que permite que la plataforma giratoria se active. Este concepto se explicará con más detalle en el capítulo 5.1.3 Programación de Smart components.

```
PROC Estado0()
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.2;
    MovimientoRobot1;
    Waitrob\Zerospeed;
    setdo s1,1;
    reset s1;
ENDPROC
```

Ilustración 96. Procedimiento 'Estado0'.

Como se puede comprobar, el procedimiento mostrado en la Ilustración 97 es muy parecido al que se ha comentado justo antes. En lo único que difieren es que en este estado ya se debe tener en cuenta la sincronización entre los robots. Para ello se hace uso de la función WaitSyncTask, esta se utiliza para sincronizar varias tareas de programa en un punto especificado de cada uno de los programas. Cada tarea de programa espera hasta que todas las tareas de programa hayan alcanzado el punto de sincronización designado por la variable *syncident*. [31]

En este caso como se desea sincronizar las tareas 1 y 2, debe aparecer la misma instrucción en ambas. Para los demás estados sería exactamente igual, pero con más tareas que sincronizar, por lo que no se va a mostrar.

```
PROC Estado1()
    WaitSyncTask sync1,Tarea1;
    CogerParteAbajo;
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.5;
    MovimientoRobot1;
    CogerParteAbajo;
    WaitSyncTask sync2,Tarea1;
ENDPROC
```

Ilustración 97. Procedimiento 'Estado1'.

Movimientos robot 1

En el procedimiento 'MovimientoRobot1' se han programado todos los movimientos que debe realizar el primer robot. Sólo se muestra uno de los movimientos que realiza debido a que todos los movimientos tienen una estructura parecida y en lo que difieren es en el punto al que el robot es mandado.

En la Ilustración 98, se muestra la programación de movimientos del robot para coger de la bandeja la placa de los leds. Esto se consigue gracias a los robtarget declarados al inicio del programa y al uso de comandos como el MoveJ (movimiento cómodo) y MoveL (movimiento lineal). También se emplea el comando offs con el que se añade un offset a los robtarget declarados.

Para poder coger piezas con la herramienta 'Ventosa' se activa la señal 's6', que activa un componente inteligente que permite la adhesión (será explicado con detalle en el capítulo 5.1.3 Programación de Smart components).

Los argumentos que tienen los comandos MoveJ y MoveL se explican con el siguiente ejemplo:

```
MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
```

- **Robtarget (PosIni1):** posición a la que debe ir el robot.
- **Velocidad del robot (v1000):** esta ha sido cambiada dependiendo de la situación. Por ejemplo, a la hora de acercarse a un componente para cogerlo esta velocidad se redujo hasta v10 para evitar dañarlo.
- **Precisión del robot (z100):** para movimientos que no requerían precisión se emplea el comando 'z100' y para movimientos que requieren de precisión, como la colocación de piezas se emplea el comando 'fine'.
- ***Tooldata* (TCPVentosa):** se declara la posición y orientación del TCP del robot, que en este caso es el de la herramienta 'Ventosa'.
- **Wobjdata (wobj0):** describe el objeto de trabajo sobre el que se realizan las tareas. Los robtargets están referidos al Wobjdata. [16]

```
!Empezamos a coger placaLeds
MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
!Activar señal ventosa
SetDO S6,1;
MoveL offs(PlacaLeds,0,0,30),v500,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(PlacaLeds,0,0,5),v10,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(PlacaLeds,0,0,20),v100,fine,TCPVentosa\WObj:=wobj0;
MoveJ PosIni1,v500,z100,TCPVentosa\WObj:=wobj0;
MoveJ PosIni,v500,z100,TCPVentosa\WObj:=wobj0;
MoveJ offs(Target_140,0,0,25),v500,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(Target_140,0,0,4),v10,fine,TCPVentosa\WObj:=wobj0;
WaitRob\inpos;
reset S6;
waittime 1;
MoveJ offs(Target_140,0,0,25),v1000,fine,TCPVentosa\WObj:=wobj0;
```

Ilustración 98. Parte del código del procedimiento 'MovimientoRobot1'.

Interrupciones (Parada de emergencia)

A continuación, se va a explicar cómo se realiza la simulación de una parada de emergencia empleando interrupciones. Como se ha comentado antes el procedimiento 'Interrupciones' es llamado en el main del programa.

Lo primero que se debe hacer es conectar las interrupciones 'Istop' y 'Parada' (declaradas en la parte de arriba del código) con la rutina 'Trap' y posteriormente se declara cuando va a ser activada la interrupción.

En el caso de la interrupción 'Istop', cuando la señal 'Empezar' cuyo valor es modificado desde la interfaz de Matlab pase a valer 0, entonces se activa la interrupción.

En el caso de la interrupción 'Parada', cuando la señal digital de entrada 'SensorFotoelectrico' pase a valer 1, se activará la interrupción. Cuando se active una interrupción se va a ejecutar lo que indica la rutina 'trap_stop' o 'trap_parada'. En cualquier caso, se va a actuar de la misma manera, ya que ambas simulan una parada de emergencia:

- Sacará un mensaje por pantalla de "Parada de emergencia",
- La señal que indica que el robot se encuentra activo, pasará a tener valor cero.
- Se ejecutará la instrucción 'EXIT', que producirá el paro instantáneo del robot.

```
PROC Interrupciones()
    IDelete Istop;
    IDelete Parada;
    CONNECT Istop WITH trap_stop;
    CONNECT Parada WITH trap_parada;
    ISignalDO Empezar,0,Istop;
    ISignalDI SensorFotoelectrico,1,Parada;
ENDPROC

LOCAL TRAP trap_stop
    Tpwrite "Parada de emergencia";
    Reset Robot1ON;
    EXIT;
ENDTRAP

LOCAL TRAP trap_parada
    Tpwrite "Parada de emergencia";
    Reset Robot1ON;
    EXIT;
ENDTRAP
```

Ilustración 99. Interrupciones del programa para la simulación de parada de emergencia.

Este procedimiento se encuentra en cada módulo de programación de los cuatro robots, por lo que no se volverá a explicar.

Tarea 2, robot IRB 1600

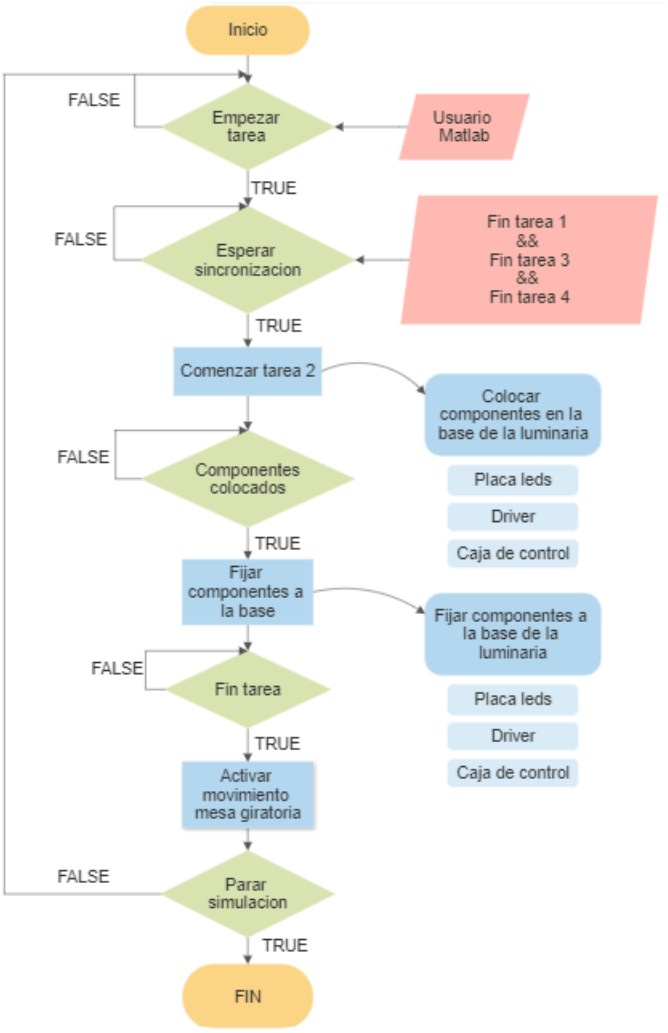


Ilustración 100. Diagrama de flujo correspondiente a la tarea del robot 2.

Lo primero, como se ha comentado anteriormente es declarar las variables. En la tarea del segundo robot, como es lógico los *robtarget* cambiarán respecto a la tarea anterior. Deben declararse también las variables de sincronización exactamente igual que en la tarea 1.

En este caso, al emplearse dos herramientas vamos a tener dos *Tooldata* diferentes que se muestran a continuación:

```
PERS tooldata TCPHerramienta1:=[TRUE,[[0,0,65],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
PERS tooldata TCPVentosa:=[TRUE,[[0,0,54.763],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
```

Ilustración 101. Variables tooldata de la tarea 2.

Después de declarar las variables y siguiendo la programación definida en el diagrama de bloques de la Ilustración 100 se procede a explicar las funciones más importantes de esta tarea.

Main

Como se puede comprobar, el procedimiento principal (main) tiene la misma estructura que en la tarea 1, pero cambian los procedimientos.

```
PROC main()
    MoveJ PosIni,v1000,z100,TCPHerramienta\WObj:=wobj0;
    waittime 2;
    WHILE TRUE DO
        Interrupciones;
        IF cont=0 THEN
            Estado1R2;
        ELSEIF cont=1 THEN
            Estado2R2;
        ELSEIF cont=2 THEN
            Estado3R2;
        ELSEIF cont>=3 THEN
            WHILE TRUE DO
                PrimerPasoR2;
                SegundoPasoR2;
                TercerPasoR2;
                CuartoPasoR2;
            ENDWHILE
        ENDIF
        cont:=cont+1;
    ENDWHILE
ENDPROC
```

Ilustración 102. Procedimiento principal (main) de la tarea2.

Igual que en la tarea anterior, la programación de los procedimientos para cada estado es muy parecida, por lo que no se van a explicar todos.

Estados de la tarea 2

El procedimiento empezaría con el robot cogiendo la herramienta 'ventosa'. Después con 'WaitDi' se espera a que se active la señal E1 (esta señal se va a activar cuando la plataforma giratoria gire hasta la posición dos).

Una vez la plataforma ha llegado a la posición, con el comando '*WaitSinkTask*' (explicado anteriormente) se espera a que los punteros de ambos programas lleguen a la misma línea para poder continuar. Como se puede ver en la Ilustración 97, el comando '*WaitSinkTask*' aparece exactamente igual que en la Ilustración 103, hasta que no se llegue en ambos programas a la línea citada el programa no avanzará, por lo que la tarea más rápida, tendrá que esperar hasta que la otra tarea alcance dicho punto.

Una vez ambos han llegado al punto de sincronización el robot empieza con el montaje colocando primero el *driver*, la caja de control y la placa de los leds en el interior de la luminaria. Posteriormente fijaría estos componentes. Una vez fijados los componentes, se activaría la señal 's9' que permite mediante un componente inteligente que los componentes se adhieran a la plataforma giratoria para que cuando esta girase, los componentes de la luminaria girasen con ella.

Cuando se active 's2' la plataforma girará hasta la siguiente posición.

```
PROC Estado1R2()
    CogerVentosa;
    WaitDI E1,1;
    WaitSyncTask sync1,Tarea1;
    ColocarComp;
    FijarComp;
    !Activamos sensor
    Setdo S9,1;
    waittime 1;
    reset s9;
    WaitRob\ZeroSpeed;
    WaitSyncTask sync2,Tarea1;
    SetDO S2,1;
    Reset S2;
    CogerVentosa;
ENDPROC
```

Ilustración 103. Procedimiento 'Estado1R2'.

Colocar componentes en luminaria

En el procedimiento 'ColocarComp' se colocan los componentes de la mesa giratoria (Origen) al interior de la luminaria (destino). Esto se consigue mediante los comandos MoveJ y MoveL como se ha explicado anteriormente. Este procedimiento se lleva a cabo con la herramienta 'ventosa'. La señal que permite que se adhieran o no componentes a esta herramienta se denomina s7.

Cuando se quiere coger un componente la señal 's7' se pondrá a 1 (SetDO s7,1) y cuando este si quiera soltar se pondrá a cero (Reset s7).

En el fragmento de código de la Ilustración 104 solo se muestra la colocación de la placa de los leds, para el *driver* y la caja de control la estructura sería exactamente igual variando únicamente la posición destino.

```
SetDO S7,1;
!Origen->Plataforma giratoria
MoveL Offs(PlacaLeds,0,0,30),v200,fine,TCPVentosa\WObj:=wobj0;
MoveL Offs(PlacaLeds,0,0,4),v10,fine,TCPVentosa\WObj:=wobj0;
MoveL Offs(PlacaLeds,0,0,30),v100,fine,TCPVentosa\WObj:=wobj0;
!Destino->Interior de luminaria
MoveJ Offs(Target_50,0,0,150),v200,fine,TCPVentosa\WObj:=wobj0;
MoveJ Offs(Target_50,0,0,20),v200,fine,TCPVentosa\WObj:=wobj0;
MoveL Offs(Target_50,0,0,7),v10,fine,TCPVentosa\WObj:=wobj0;
WaitRob\inpos;
Reset S7;
```

Ilustración 104. Fragmento de código del procedimiento 'ColocarComp'.

Cambio de herramienta

Para fijar los componentes a la luminaria se deben ejecutar varios procesos, el primero realizar el cambio de herramienta 'ventosa' a la herramienta encargada de fijar los tornillos. Este procedimiento se muestra en la Ilustración 105. Para llevar a cabo este proceso primero se debe de ir a la posición donde se debe dejar la ventosa y una vez en la posición se resetea la señal 'CogerVent', que es la que permite que esta se adhiera al robot. Una vez se ha dejado la ventosa se procede a coger la herramienta, para ello se activa la señal 'CogerHerr' y se lleva al robot a la posición dónde se encuentra esta herramienta. En el capítulo 5.1.3 Programación de Smart components se explicará con detalle cómo se produce la adhesión de la herramienta al robot.

```
!Ir a la posición donde dejamos la ventosa
MoveJ Offs(Ventosa,0,0,20),v1000,z100,tool0\WObj:=wobj0;
MoveL Ventosa,v10,fine,tool0\WObj:=wobj0;
WaitRob\inpos;
Reset CogerVent;
!Ventosa dejada
MoveJ Offs(Ventosa,0,0,50),v500,z100,tool0\WObj:=wobj0;
WaitTime 1;
!Cogemos herramienta para fijar tornillos
SetDO CogerHerr,1;
MoveJ Offs(Herramienta1,0,0,20),v500,z100,tool0\WObj:=wobj0;
MoveL Herramienta1,v10,fine,tool0\WObj:=wobj0;
MoveL Offs(Herramienta1,0,0,100),v300,z100,tool0\WObj:=wobj0;
MoveJ PosIni,v1000,z100,TCPHerramienta\WObj:=wobj0;
```

Ilustración 105. Fragmento de código que muestra cómo realizar el cambio de herramienta.

Coger tornillos del soporte

Una vez se ha producido el cambio de herramienta se procede a fijar los componentes. El procedimiento de fijación de tornillos es el siguiente, primero se coge el tornillo del soporte diseñado y después se coloca en la luminaria. El código de este se muestra en la Ilustración 106.

Para coger los tornillos del soporte se ha programado el procedimiento 'CogerTor' al que se le pasa por referencia un robtarget que indica la posición del tornillo que debe cogerse. El modo de operar es el siguiente, primero acercamos el robot a la posición y después se selecciona el tipo de tornillo que se debe escoger mediante sentencias condicionales IF y ELSEIF, ya que no todos los tornillos son iguales.

Después tenemos un bucle FOR que nos permite introducir la herramienta para coger los tornillos con precisión. Para poder realizar este movimiento se necesitan variables del tipo pose (Posición + Orientación). Inicialmente, hacemos que él *PosOr1* adquiera la posición y la orientación del robtarget 'aux1'. Después, el *PosOr2* nos indicará la distancia en Z con un valor que irá aumentando 0.5 milímetros por cada iteración que realice el bucle FOR hasta llegar a 2 milímetros de profundidad. Después, se calcula el *PosOr3* cómo la multiplicación del *PosOr1* y el *PosOr2*, obteniendo un nuevo punto con la posición y orientación del punto inicial pero desplazado en su eje Z un total de 2 milímetros. La señal 's8' se activa para que el tornillo quede adherido a la herramienta. El segundo bucle FOR es prácticamente igual que el que se acaba de comentar, pero en sentido contrario, es decir, para sacar el tornillo del agujero del soporte.

```
PROC CogerTor(robtarget aux1)
    !Primero acercamos el robot al punto
    MoveJ Offs(aux1,0,0,20),v1000,z100,TCPHerramienta1\WObj:=wobj0;
    MoveL Offs(aux1,0,0,5),v10,fine,TCPHerramienta1\WObj:=wobj0;
    !Dependiendo del tipo de tornillo la distancia a recorrer
    !será mas larga o más corta
    IF tipo=1 THEN
        dist:=2;
    ELSEIF tipo=2 THEN
        dist:=7;
    ENDIF
    ConFL\off;
    !Bucle que consigue coger con precision los tornillos
    FOR i FROM 0 TO 2 STEP 0.5 DO
        PosOr1.rot:=aux1.rot;
        PosOr1.trans:=aux1.trans;
        PosOr2:=[[0,0,i],[1,0,0,0]];
        PosOr3:=PoseMult(PosOr1,PosOr2);
        aux.rot:=PosOr3.rot;
        aux.trans:=PosOr3.trans;
        moveL aux,v10,fine,TCPHerramienta1\WObj:=wobj0;
    ENDFOR

  !Para que el tornillo quede adherido a la herramienta
  !se activa la señal s8
  SetDO s8,1;
  !Bucle para sacar el tornillo sin golpear el soporte
  FOR i FROM 0 TO dist STEP 0.5 DO
      PosOr1.rot:=aux1.rot;
      PosOr1.trans:=aux1.trans;
      PosOr2:=[[0,0,-dist],[1,0,0,0]];
      PosOr3:=PoseMult(PosOr1,PosOr2);
      aux.rot:=PosOr3.rot;
      aux.trans:=PosOr3.trans;
      moveL aux,v10,fine,TCPHerramienta1\WObj:=wobj0;
  ENDFOR
ENDPROC
```

Ilustración 106. Procedimiento 'CogerTor'.

En la Ilustración 107 se muestra gráficamente lo que se acaba de explicar. Como se puede apreciar, la forma de hexágono de la herramienta se ajusta perfectamente al hueco en forma de hexágono del tornillo.

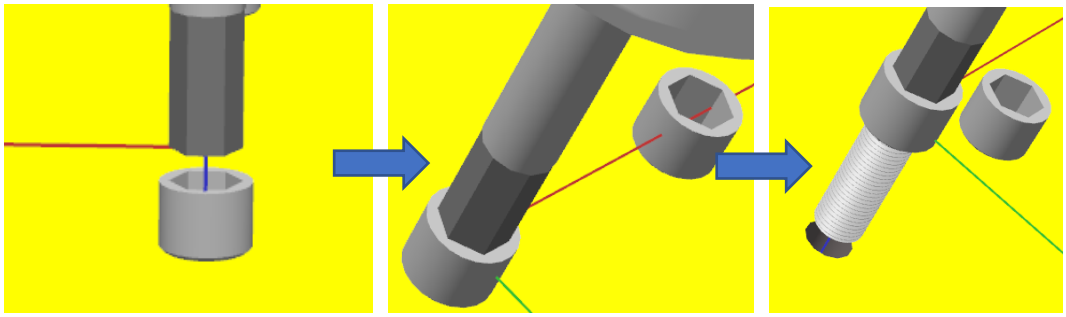


Ilustración 107. Secuencia de herramienta cogiendo tornillo.

Colocar tornillo en luminaria

A continuación, se va a explicar el proceso de cómo el robot coloca los diferentes tornillos en la luminaria. Lo primero que se hace es asignar un valor a la variable tipo, esto servirá para indicar al robot la dimensión del tornillo que se va a colocar. Después se llama a la función explicada anteriormente y se coge el tornillo. Una vez cogido se lleva este a la posición dónde va a ser colocado. Para la simulación de la fijación del tornillo se le manda al robot a dos posiciones diferentes que son las siguientes:

- Posición inicial: el robot coloca el tornillo justo al principio del agujero habilitado para colocarlo. Se muestra en la siguiente ilustración.

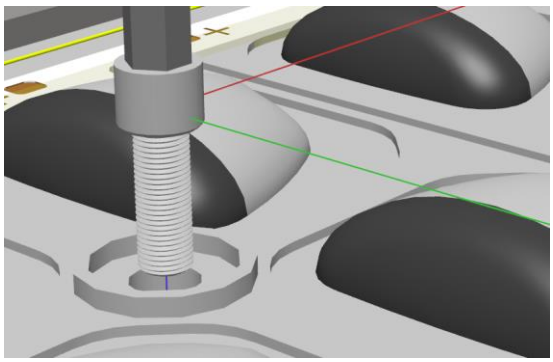


Ilustración 108. Tornillo en la posición inicial de fijación.

- Posición final: el robot coloca el tornillo en la posición final del agujero realizando una rotación de 180º respecto al eje Z y una traslación correspondiente a la dimensión del tornillo a lo largo del eje Z. Se muestra en la siguiente ilustración el eje de color rojo (eje X) ha rotado 180º respecto a la posición inicial antes comentada.

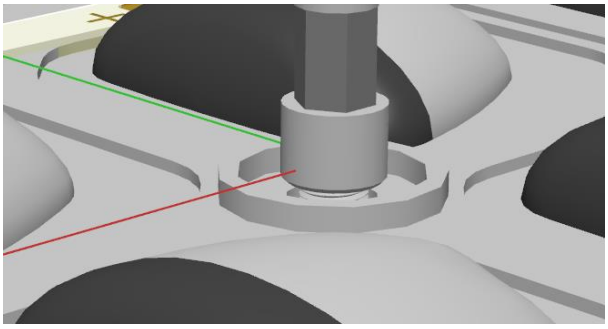


Ilustración 109. Tornillo en la posición final de fijación.

Una vez el tornillo está en la posición correcta se pone a cero la señal 's8', lo que permite que este se despegue de la herramienta y se quede fijada en la luminaria.

El proceso que se muestra en la Ilustración 110, se repite para las demás posiciones donde se debe fijar cada tornillo.

```
!1º Tornillo de la caja de control
Tipo:=1;
!Cogemos el tornillo del soporte
CogerTor(TornilloCC);
!Tornillo cogido se procede a llevar a la luminaria
MoveJ Offs(CC1I,0,0,40),v1000,z100,TCPHerramienta1\WObj:=wobj0;
MoveL Offs(CC1I,0,0,10),v100,fine,TCPHerramienta1\WObj:=wobj0;
MoveL Offs(CC1I,0,0,-0.8),v10,fine,TCPHerramienta1\WObj:=wobj0;
MoveL Offs(CC1F,0,0,-0.8),v10,fine,TCPHerramienta1\WObj:=wobj0;
!Reseteamos la señal para que la herramienta suelte el tornillo
reset s8;
```

Ilustración 110. Fragmento de código donde se fijan los tornillos a la luminaria.

Tarea 3, robot IRB 1600ID

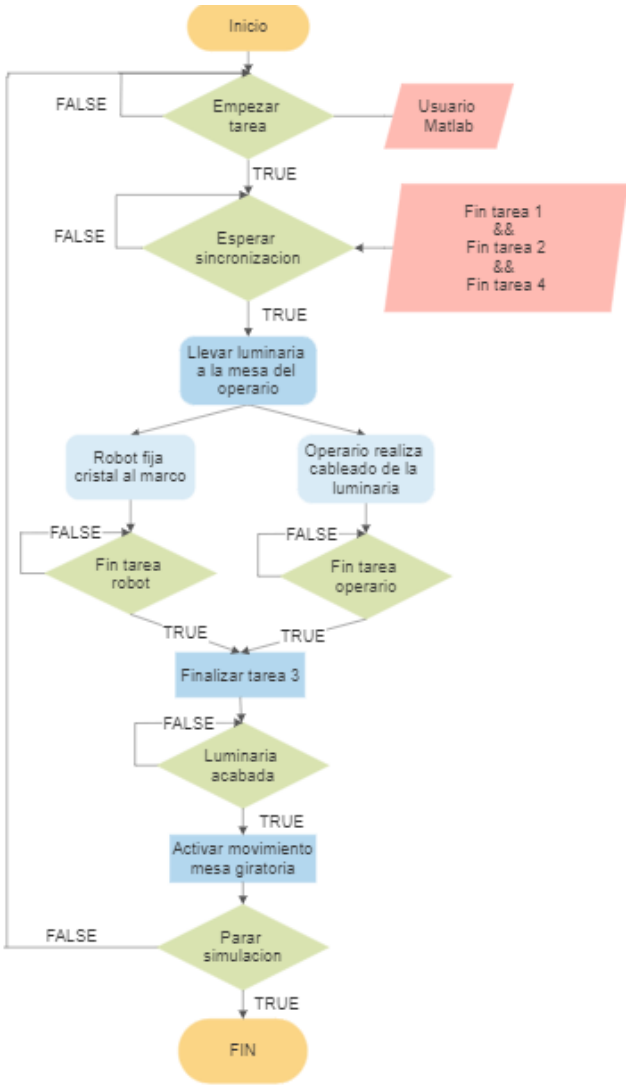


Ilustración 111. Diagrama de flujo correspondiente a la tarea del robot 3.

El procedimiento de declaración de variables es igual que en los casos anteriores, También se han de declarar las variables que permiten la sincronización. Para esta tarea cómo se han utilizado 3 herramientas diferentes se deben declarar tres *Tooldata,* uno para cada herramienta*:*

- Herramienta que fija el cristal al marco.
- Herramienta que fija los tornillos.
- Ventosa.

```
PERS tooldata TCPGlue:=[TRUE,[[0,0,70],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
PERS tooldata TCPHerramienta2:=[TRUE,[[0,0,70],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
PERS tooldata TcpVentosaG:=[TRUE,[[0,0,84.622],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
```

Main

La estructura del código main es muy parecida a la de las tareas anteriores y los procedimientos a los que llama son parecidos a la tarea dos (fijación de componentes, recoger y colocar componentes, etc.), por lo que solo se van a explicar las tareas que sean diferentes.

```
PROC main()
    !Robot colocado en la posicion inicial
    moveJ posIni,v1000,z100,TcpVentosaG\WObj:=wobj0;
    waittime 2;
    WHILE TRUE DO
        Interrupciones;
        IF cont=0 THEN
            Estado2R3;
        ELSEIF cont=1 THEN
            Estado3R3;
        ELSEIF cont>=2 THEN
            WHILE TRUE DO
                PrimerPasoR3;
                SegundoPasoR3;
                TercerPasoR3;
                CuartoPasoR3;
            ENDWHILE
        ENDIF
        cont:=cont+1;
    ENDWHILE
ENDPROC
```

Ilustración 112. Código main tarea 3.

Movimientos del tercer robot

El procedimiento en el que han sido programados los movimientos del robot se llama 'MovimientoRobot3' y a continuación se explican los movimientos más importantes. Lo primero que debe hacer el robot es llevar la luminaria al operario, una vez la haya dejado en la mesa del operario y se haya alejado lo suficiente para que el operario no esté en peligro se activará una señal llamada 'MoverOP', con la que se permitirá el movimiento del operario. A partir de este instante, el robot y el operario continuarán haciendo tareas en paralelo.

```
!Cogemos la luminaria para mandarla al operario
waitrob\InPos;
SetDO s10,1;
MoveJ offs(Lum,0,0,15),v1000,z100,TcpVentosaG\WObj:=wobj0;
MoveL Lum,v10,fine,TcpVentosaG\WObj:=wobj0;
MoveJ offs(Lum,0,0,40),v10,z100,TcpVentosaG\WObj:=wobj0;
MoveJ Help,v1000,z100,TcpVentosaG\WObj:=wobj0;
!Dejamos la Lum en la mesa del operario
MoveJ offs(Lmesa,0,0,40),v1000,z100,TcpVentosaG\WObj:=wobj0;
MoveL Lmesa,v10,fine,TcpVentosaG\WObj:=wobj0;
WaitRob\InPos;
reset s10;
MoveJ offs(Lmesa,0,0,40),v50,z100,TcpVentosaG\WObj:=wobj0;
MoveJ Help,v1000,z100,TcpVentosaG\WObj:=wobj0;
!Movemos al operario, para realizar comprobaciones y poner el cableado
setdo MoverOp,1;
reset MoverOp;
MoveJ PosIni,v1000,z100,TcpVentosaG\WObj:=wobj0;
```

Ilustración 113. Fragmento de código de la tarea Robot 3.

Mientras el operario está realizando diferentes operaciones en la base de la luminaria, el robot cambia de herramienta a la herramienta 'Glue', que es la encargada de fijar el marco del cristal.

Para la realización de este movimiento el robot debe recorrer con precisión los extremos del cristal para fijarlos al marco. El procedimiento creado para llevar a cabo esta tarea se muestra en la Ilustración 114, donde los puntos que debe recorrer el robot son llamados 'CR1', 'CR2', 'CR3', etc. Para que el robot haga un movimiento rectilíneo se ha utilizado el comando MoveL y para que recorra los puntos con precisión se ha utilizado el comando fine.

```
PROC FijarCristal()
    !Recorremos los extremos del cristal para fijarlos
    MoveL Cr1,v100,fine,TCPGlue\WObj:=wobj0;
    MoveL Cr2,v100,fine,TCPGlue\WObj:=wobj0;
    MoveL Cr3,v100,fine,TCPGlue\WObj:=wobj0;
    MoveL Cr4,v100,fine,TCPGlue\WObj:=wobj0;
    MoveL Cr5,v100,fine,TCPGlue\WObj:=wobj0;
    MoveL Cr6,v100,fine,TCPGlue\WObj:=wobj0;
    MoveL Cr7,v100,fine,TCPGlue\WObj:=wobj0;
    MoveL Cr8,v100,fine,TCPGlue\WObj:=wobj0;
    MoveL Cr1,v100,fine,TCPGlue\WObj:=wobj0;
    MoveJ offs(Cr1,0,0,30),v100,z100,TCPGlue\WObj:=wobj0;
    MoveJ PosIni,v1000,z100,tool0\WObj:=wobj0;
ENDPROC
```

Ilustración 114. Procedimiento 'FijarCristal'.

Una vez el robot ha terminado con la tarea esperaría en la posición inicial a que el operario mandarse una señal de que ha terminado ('FinTareaOperario') y de que se encuentre suficientemente alejado. Esto lo conseguimos con el comando 'WaitDi'.

```
MoveJ PosIni,v1000,z100,TcpVentosaG\WObj:=wobj0;
!Esperamos a que el operario acabe su tarea para que el robot
!pueda acercarse
WaitDI FinTareaOperario, 1;
```

Una vez el operario activa la señal, el robot cogerá la luminaria y completará el montaje poniendo el marco encima de la base para posteriormente fijarlo con tornillos. Cómo este proceso de fijación se ha explicado en la tarea dos, no se va a volver a explicar.

Tarea 4, robot paralelo

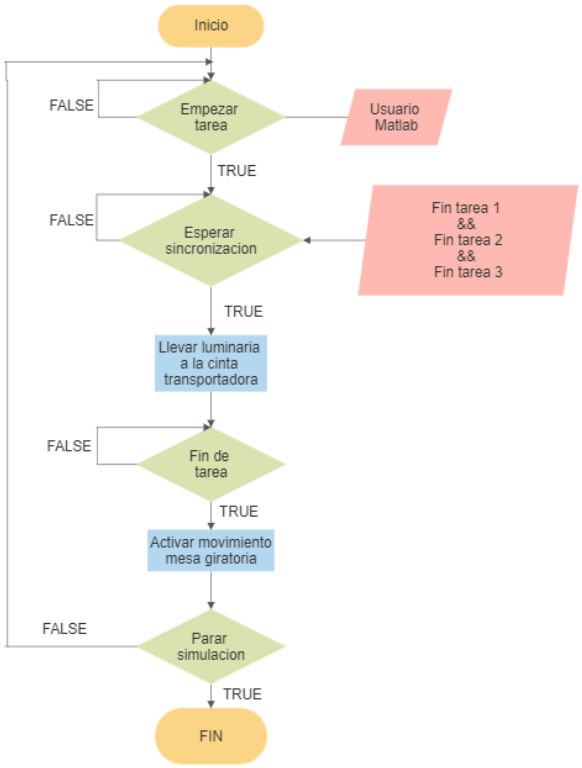


Ilustración 115. Diagrama de flujo correspondiente a la tarea del robot 4.

Como se puede observar en el diagrama de flujo, la tarea del robot cuatro va a ser la más simple de todas. El código main va a seguir la estructura mencionada en las tareas anteriores y la programación de los movimientos del robot va a ser mucho más simple.

```
PROC main()
    waittime 2;
    MoveJ PosIni,v1000,z100,TCPVentosaAraña\WObj:=wobj0;
    SetDO GenCaja,1;
    reset GenCaja;
    WHILE cond=0 DO
        Interrupciones;
        IF cont=0 THEN
           Estado3R4;
        ELSEIF cont>=1 THEN
            WHILE cond=0 DO
                PrimerPasoR4;
                SegundoPasoR4;
                TercerPasoR4;
                CuartoPasoR4;
            ENDWHILE
        ENDIF
        cont:=cont+1;
    ENDWHILE
ENDPROC
```

Ilustración 116. Main tarea Robot 4.

Para la realización de su tarea simplemente se ha tenido que programar el traslado de la luminaria completamente montada a una caja que se encuentra en la cinta transportadora, encargada de sacar la caja con la luminaria en su interior del sistema. Para ello primero se activa la señal de la ventosa 's12' y se manda al robot a recoger la luminaria. Una vez hecho esto, se traslada la luminaria a la cinta transportadora y se resetea la señal 's12' para que se desactive la ventosa. Después de esto, se activa la señal 'SalirSist' que es la encargada de mover la cinta transportadora para que la luminaria salga del sistema.

```
PROC MovimientoRobot4()
    MoveJ PosIni,v1000,z100,TCPVentosaAraña\WObj:=wobj0;
    !Activar ventosa
    SetDO s12,1;
    !Coger luminaria
    MoveJ offs(Lum,0,0,40),v500,z100,TCPVentosaAraña\WObj:=wobj0;
    MoveL Lum,v50,fine,TCPVentosaAraña\WObj:=wobj0;
    MoveL offs(Lum,0,0,40),v100,z100,TCPVentosaAraña\WObj:=wobj0;
    MoveJ PosIni,v500,z100,TCPVentosaAraña\WObj:=wobj0;
    !Dejar luminaria en la cinta transportadora
    MoveJ offs(LumCinta,-30,30,80),v500,z100,TCPVentosaAraña\WObj:=wobj0;
    MoveL offs(LumCinta,-30,30,0),v10,fine,TCPVentosaAraña\WObj:=wobj0;
    WaitRob\InPos;
    !Desactivar ventosa
    reset s12;
    waittime 1;
    MoveJ offs(LumCinta,0,0,80),v100,z100,TCPVentosaAraña\WObj:=wobj0;
    MoveJ PosIni,v500,z100,TCPVentosaAraña\WObj:=wobj0;
    !Activar señal que mueve la cinta transportadora
    SetDO SalirSist,1;
    waittime 0.5;
    Reset SalirSist;
ENDPROC
```

Ilustración 117. Procedimiento 'MovimientoRobot4'

5.1.3 Programación de Smart components

Los componentes inteligentes son objetos de RobotStudio con propiedades y lógicas integradas para simular componentes que no forman parte del controlador virtual. Algunos ejemplos de lo que se puede hacer con estos componentes son: movimientos de la pinza de un robot, movimiento de objetos en cintas transportadoras, generación de piezas, etc.

Dicho esto, para que los componentes diseñados e introducidos en Robotstudio tengan un funcionamiento real durante la simulación se hace uso de los componentes inteligentes. Para añadir un componente inteligente, en la pestaña de 'Modelado' se debe seleccionar 'Componente Inteligente'. una vez seleccionada nos saldrá la siguiente ventana:

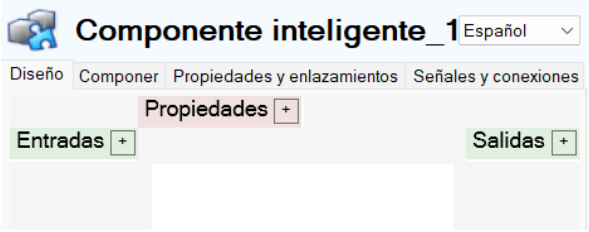


Ilustración 118. Ventana de diseño de Componente Inteligente.

A partir de esto, se empezarán a crear todos los componentes inteligentes que forman la estación. La programación de componentes inteligentes se basa en la unión de diferentes bloques que se añaden desde la ventana 'Componer'. Se distinguen entre varias categorías para clasificar estos bloques:

1. **Señales y propiedades:** donde se encuentran bloques que permiten operaciones lógicas (AND, OR, NOT, etc.), realizar expresiones matemáticas, contadores, medidores de tiempo, etc.
2. **Primitivos paramétricos:** permite la creación de cualquier tipo de pieza durante la simulación.
3. **Sensores:** encontramos bloques que contienen sensores de diferentes formas que activarán señales cuando un objeto los toque.
4. **Acciones:** este es uno de los tipos de bloques más utilizados debido a que se encuentran bloques que permiten conectar y desconectar objetos, crear copias de componentes gráficos y eliminar componentes gráficos.
5. **Manipuladores:** este tipo permite mover diferentes cuerpos formando trayectorias circulares y lineales. También permite mover los ejes de un mecanismo y posicionar objetos en una coordenada determinada.
6. **Controlador:** el bloque se puede obtener y puede cambiar valores de una señal de RAPID.
7. **Física:** permite asignar propiedades físicas a un objeto o controlar las posiciones de un eje físico.
8. **PLC:** permite establecer conexión con los PLCs de Siemens mediante OPC o SIMITConnection.
9. **Otros:** en este modo se encuentra una gran variedad de bloques que permiten, por ejemplo: cambiar el color de un objeto, reproducir sonido, general señales cuando se active o desactive la simulación o activar el seguimiento del TCP el robot.

Una vez se ha introducido la programación en los componentes inteligentes, se procede a mostrar la lógica de la estación, es decir, dónde se encuentra cada uno de los componentes inteligentes que han sido programados (véase la Ilustración 119). Como se puede ver, estos componentes están conectados al controlador mediante señales de entrada y salida que se activan y desactivan a través de RAPID.

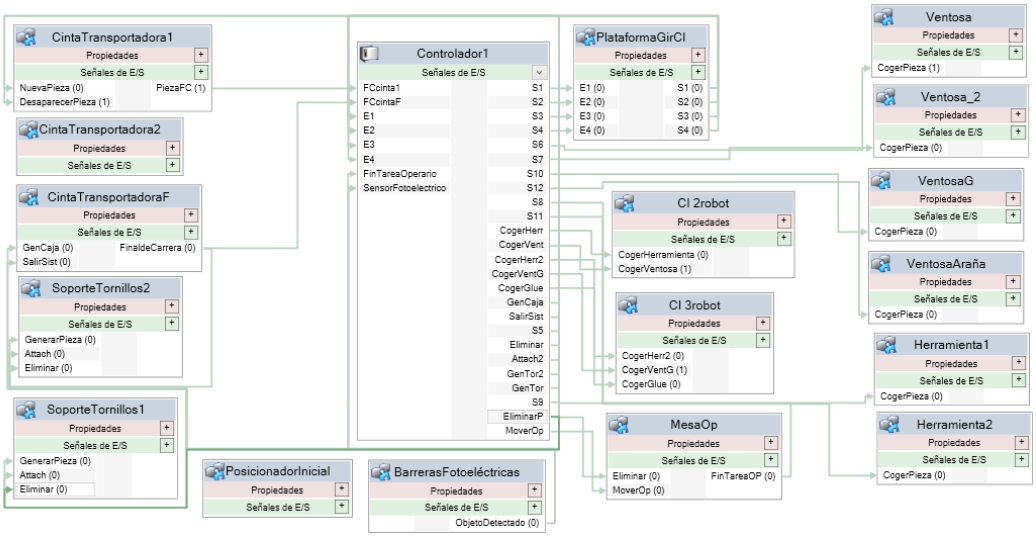


Ilustración 119. Lógica de la estación en RobotStudio.

A continuación, se procede a realizar la explicación de cada uno de ellos.

Cinta transportadora 1

Este componente ha sido creado para simular la creación de piezas y el transporte de estas mediante una cinta transportadora para hacérselas llegar al robot. Se cuenta con dos entradas digitales (‘NuevaPieza’ y ‘DesaparecerPieza’) y una salida digital (‘PiezaFC’).

Operaciones Lógicas del componente

Para que se puedan generar y mover piezas se deben cumplir una serie de condiciones que vienen dadas por una serie de puertas lógicas. Cuando se active la señal 'NuevaPieza' la puerta OR mandará un 1 lógico a la puerta AND con un segundo de retraso. La puerta AND estará activa a nivel alto siempre y cuando la puerta lógica NOT mande un 1 lógico.

El bloque 'FinaldeCarrera' es un sensor colocado al final de la cinta transportadora y se activará cuando detecte una pieza. La salida de este está unida a una puerta NOT, por lo que la salida será invertida, es decir, cuando se tenga un 1 lógico la puerta NOT provocará un 0 lógico y viceversa.

En resumen, cuando se active la señal 'NuevaPieza' y no haya ninguna pieza al final de la cinta transportadora se activarán los bloques que se explican a continuación.

Generación de piezas de la luminaria

Como en este proyecto se deben crear luminarias hasta qué el usuario decida parar la simulación, no se puede disponer de un único set de piezas, por lo que se deben crear copias de las piezas originales qué serán manipuladas por los robots y eliminadas después del montaje.

Para realizar esto, se usó el bloque 'Source', que cada vez que le llegue un 1 lógico, creará una copia de la pieza que se haya seleccionado en las propiedades del bloque. Se debe especificar la coordenada donde se quiera crear dicha copia, en este caso se crearán todas al inicio de la cinta transportadora. Debe existir un bloque 'Source' para cada copia que se quiera generar.

Para evitar problemas de memoria cuando finalice la simulación se ha activado en este bloque la opción 'Transient', que nos permite eliminar las piezas que han sido generadas una vez haya terminado la simulación.

Movimiento de las piezas

Una vez han sido generadas las copias de las piezas originales, la salida del bloque 'Source' estará activa a nivel alto (1) lo que provocará la activación de los bloques 'LinearMove', que son los encargados de mover las piezas hasta el final de la cinta transportadora dónde se encontrará el sensor comentado anteriormente y encargado de parar el movimiento de las piezas para que estas puedan ser cogidas por el robot. La velocidad con la que se mueven las piezas es de 400 mm/s (0.4 m/s).

Cada bloque 'LinearMove' debe estar conectado con el bloque 'Source', para que cada pieza generada pueda moverse

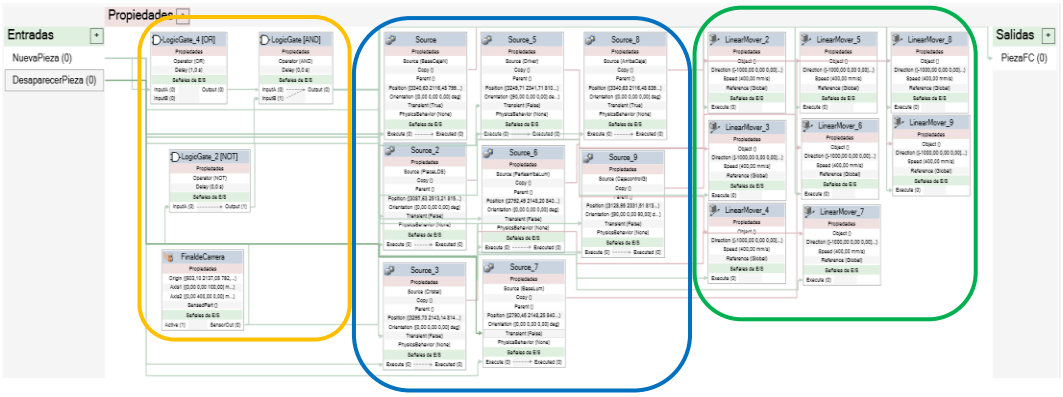


Ilustración 120. Componente Inteligente 'CintaTransportadora1'.

Cinta Transportadora 2

Este componente es el encargado de simular la salida del sistema de las bandejas dónde vienen las piezas.

Cuando el robot deposite la bandeja en la cinta transportadora, esta será detectada por un sensor ('PlaneSensor'), el cual mandará un 1 lógico al bloque OR, y con un retraso de 2 segundos activará el bloque 'LinearMover' provocando el movimiento por la cinta transportadora de una distancia de 2400 mm (longitud de la cinta transportadora) en un tiempo de 5 segundos, es decir, tendrá una velocidad de 480 mm/s (0.48 m/s). Cuando se haya ejecutado el bloque 'LinearMover' mandará una señal que activará el bloque 'Sink' que es el encargado de simular la salida del sistema del componente procediendo a su eliminación.

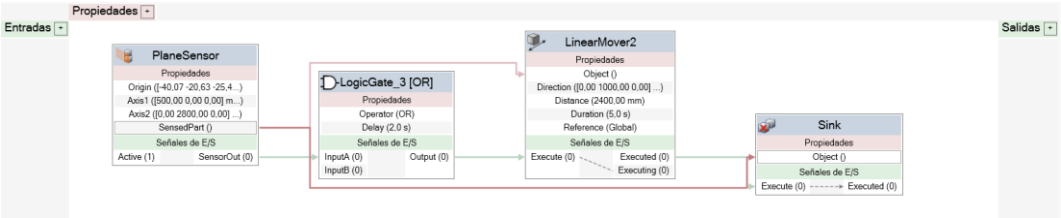


Ilustración 121. Componente Inteligente 'CintaTransportadora2'.

Cinta Transportadora Final

El objetivo de este componente inteligente es introducir la luminaria completamente montada en una caja para posteriormente poder sacarla del sistema a través de una cinta transportadora. Este componente inteligente cuenta con dos entradas digitales (‘GenCaja’ y ‘SalirSist’) y una salida digital (‘Final de carrera’).

Generación de caja y desplazamiento

Cuando se active la señal ‘GenCaja’ el bloque ‘Source’ generará una caja al inicio de la cinta transportadora, en el interior de esta se encuentra un sensor (‘PlaneSensor’) que detectará cuando introduce el robot la luminaria dentro de la caja.

Cuando el robot deposite la luminaria dentro de la caja se producirá la unión entre la caja y la luminaria gracias al bloque ’Attacher’. Cuando se produzca esta unión el bloque ’Attacher’ activará el bloque ‘LinearMove’ que permitirá el movimiento de ambos componentes hasta el final de la cinta transportadora con una velocidad de 600 mm/s (0.6 m/s).

Eliminación de componentes

Una vez han llegado los componentes al final de la cinta transportadora deben ser eliminados, ya que se supone que estos serían transportados a otra sección de la fábrica. Para la eliminación se cuenta con varios sensores que cuando detecten una pieza primero activarán el bloque 'Dettacher', encargado de deshacer la unión creada anteriormente entre los componentes de la luminaria y la caja, ya que si no se hace esto puede dar problemas en la simulación.

Después de deshacer la unión entre componentes, se manda un 1 lógico al bloque 'OR' y con un retraso de 200 milisegundos se activa el bloque 'Sink' que permitirá eliminar todos los componentes.

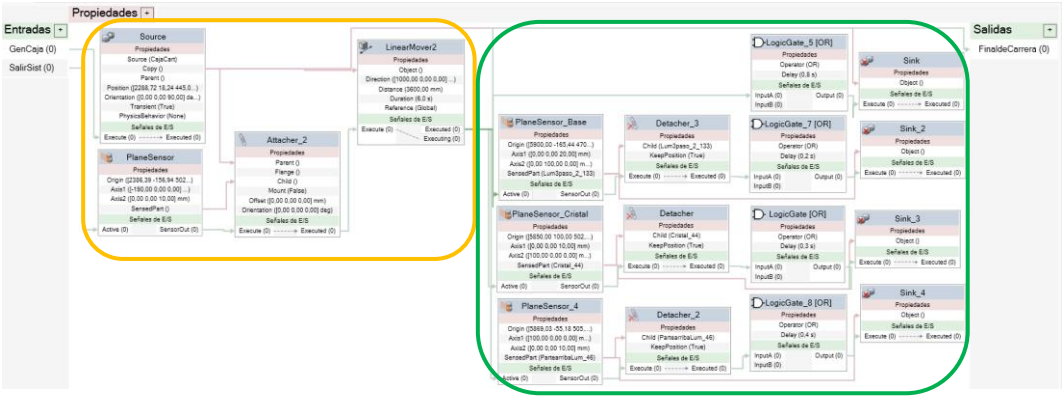


Ilustración 122. Componente Inteligente 'CintaTransportadoraF'.

Soporte de tornillos 1

En este componente inteligente se programa la generación de los tornillos necesarios y la unión posterior a los componentes de la luminaria. Se cuenta con 3 entradas digitales ('GenerarPieza', 'Attach' y 'Eliminar').

Generación de tornillos

Mediante el bloque 'Source' se generan copias de los tornillos originales en el soporte habilitado cada vez que se active la señal 'GenerarPieza'.

Cabe destacar que para generar los tornillos en este caso también debe tenerse en cuenta la orientación a la que se generan, ya que el soporte donde van colocados está inclinado.

Unión de componentes a la base de la luminaria

Una vez se han colocado los diferentes tornillos estos deben estar unidos a la luminaria. Para ello, se cuenta con un sensor que detectará la base de la luminaria que se pasará como parámetro al bloque 'Attacher' para indicarle la pieza con la que debe realizar esta unión. Se necesita un bloque 'Attacher' para cada componente.

En este componente inteligente también se ha programado la unión a la luminaria del *driver*, de la caja de control y de la PCB. Para realizar esto se han colocado 3 sensores que detectarán cada componente y posteriormente con el bloque 'Attacher' quedarán unidos a la luminaria.

Todos los bloques 'Attacher' serán activados mediante la señal de entrada 'Attach' controlada desde RAPID.

Eliminación de componentes

Todos los componentes generados serán eliminados por el bloque 'Sink' cuando se active la señal de entrada 'Eliminar'. Dicha señal es controlada desde RAPID.

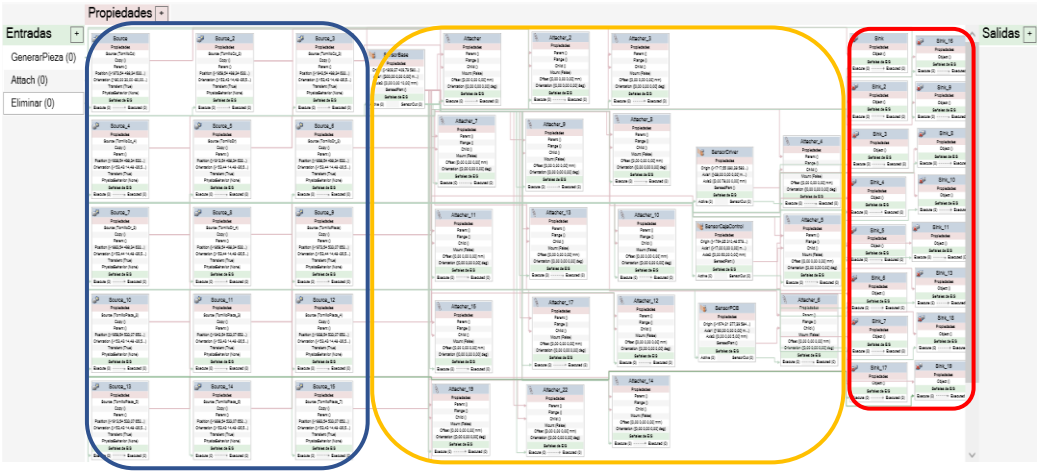


Ilustración 123. Componente Inteligente 'Soporte tornillos 1'.

Soporte de tornillos 2

El procedimiento de este componente inteligente es muy parecido al que se acaba de explicar. Se cuenta con 3 entradas digitales ('GenerarPieza', 'Attach' y 'Eliminar').

Primero, se generarán todos los tornillos en el soporte habilitado y con la orientación adecuada (señalado en azul), después cuando se active la señal 'Attach', se activará el bloque 'Attacher' y estos quedarán unidos a la luminaria (señalado en rojo). Para unir la iluminaria a la plataforma giratoria y que esta gire cuando la plataforma se mueva, se ha utilizado el bloque 'Attacher' señalado en color naranja. Para eliminar todos los componentes generados cuando la luminaria salga del sistema se utiliza el bloque 'Sink' (señalado en verde). Este actuará cuando se active la señal digital de entrada 'Eliminar'.

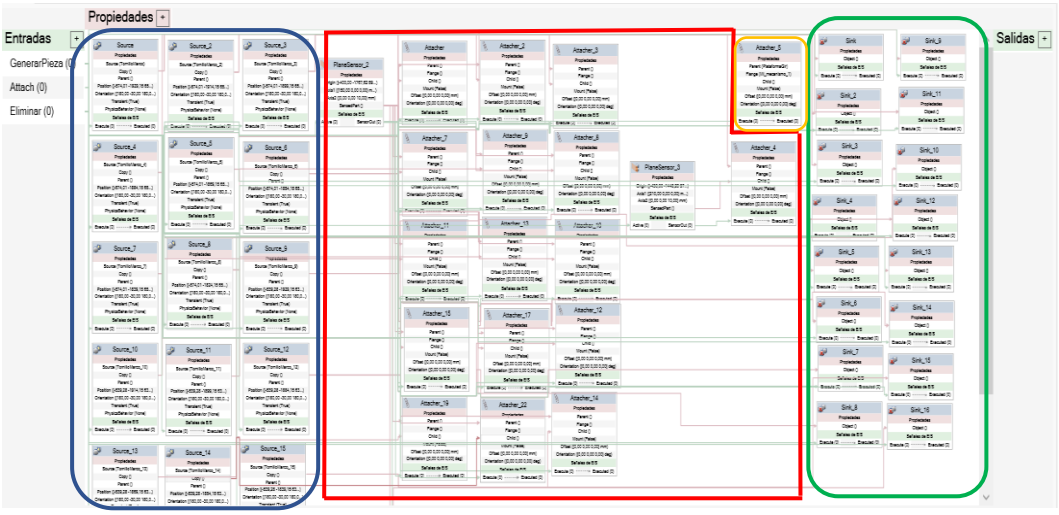


Ilustración 124. Componente Inteligente 'Soporte tornillos 2'.

Posicionador inicial

La función de este componente inteligente es la colocación de las herramientas de los robots para que cuando empiece la simulación estas estén en su lugar.

Para ello, se ha empleado el bloque 'SimulationEvents' que cuenta con dos salidas, una que se activará (emitirá una señal de un 1 lógico) cuando empiece la simulación y otra para cuando termine la simulación. Esta salida es conectada al bloque 'OR' para que se pueda mandar la señal cuando estemos en una situación u otra. La salida del bloque 'OR' es conectada a los diferentes bloques 'Positioner' a los que se tiene que definir la posición, la orientación y la herramienta que se debe posicionar.

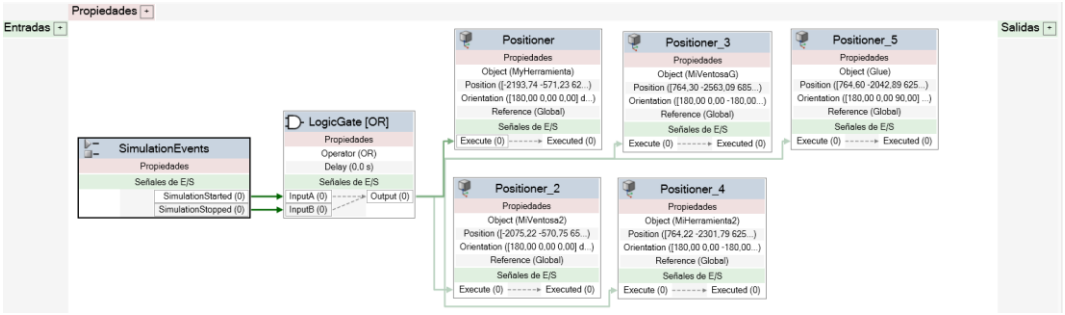


Ilustración 125. Componente Inteligente 'Posicionador Inicial'.

Plataforma giratoria

Este componente inteligente permite el movimiento del mecanismo 'Plataforma giratoria' por medio de señales activadas desde RAPID. Cuenta con cuatro entradas y cuatro salidas digitales.

Movimiento del mecanismo

Para programar el movimiento de la plataforma giratoria se ha necesitado hacer uso del bloque 'JointMover', al que se le debe indicar los grados que debe moverse el mecanismo y el tiempo que debe tardar en hacer este movimiento.

El funcionamiento es el siguiente: cada entrada digital ('E1', 'E2', 'E3' o 'E4') está asociada a un 'JointMover', entonces, cuando se active una de estas entradas activará un bloque 'JointMover' provocando un movimiento en el siguiente orden: 90º (E1), 180º (E2), 270º (E3) o 360º (E4) respecto de la posición inicial (0º). Cuando se ejecute el bloque que provoca un movimiento de 360º, se ejecutará otro 'JointMover' que llevará la plataforma a un ángulo de 0º en un tiempo de 0 segundos lo que permite repetir el proceso de manera ilimitada.

Cada vez que se ejecute completamente este bloque activará la correspondiente salida digital ('S1', 'S2', 'S3' o 'S4').

Unión de componentes a la plataforma

Para que los componentes que se colocan en la plataforma puedan girar cuando la mesa gire, se han colocado diferentes 'PlaneSensor' que cuando detecten piezas, activarán el bloque 'Attacher' (con un retraso de 6 segundos generado por el bloque 'OR') que les permitirá unirse a la plataforma giratoria.

Cuando se inicie la simulación, si la plataforma no está en su posición inicial (0º) se colocará en ella por medio de un 'JointMover' que será activado por el bloque 'SimulationEvents'.

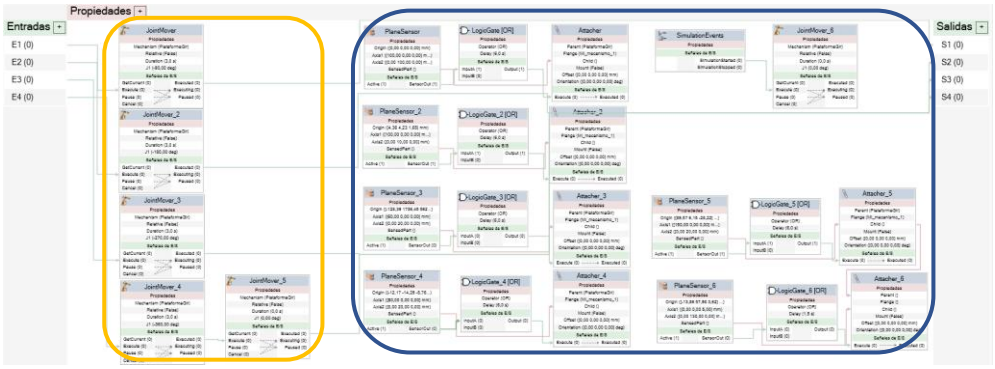


Ilustración 126. Componente Inteligente 'PlataformaGiratoria'.

Componente inteligente 2º robot

Este componente inteligente ha sido creado para poder realizar el cambio de herramienta, de manera que el robot pueda realizar operaciones con dos herramientas. Se cuenta con dos señales de entrada, una que se activará para poder coger la herramienta con la que se colocan los tornillos y otra con la que se podrá coger la ventosa.

El funcionamiento es el siguiente, ambas entradas irán conectadas al bloque 'OR', que tiene su salida conectada al bloque 'LineSensor'. Este sensor está ubicado en la posición del TCP del robot.

El bloque 'LineSensor' activará su salida cuando el sensor detecte una herramienta, esta salida es conectada al bloque 'AND'. Entonces, cuando en este bloque se detecte un 1 lógico por parte del 'LineSensor' y por parte de una de las entradas del componente inteligente, se activará el bloque 'Attacher' fijando la herramienta al extremo del robot.

Cuando se desactive cualquiera de las entradas del componente inteligente significa que el robot debe dejar la herramienta por lo que mediante el bloque 'Detacher' se eliminará la unión del robot a la herramienta.

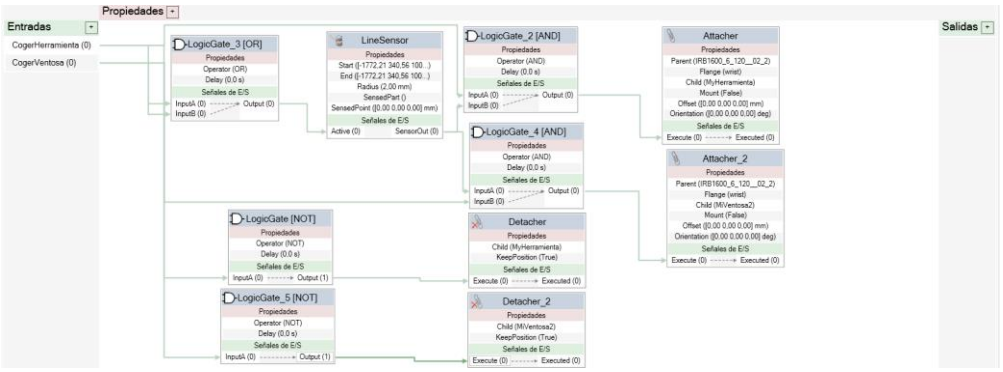


Ilustración 127. Componente Inteligente 'SegundoRobot'.

Componente inteligente 3º robot

El funcionamiento de este componente inteligente es exactamente el mismo que en el anterior, lo único que el tercer robot cuenta con 3 herramientas, por lo que se tendrá que programar un caso más.

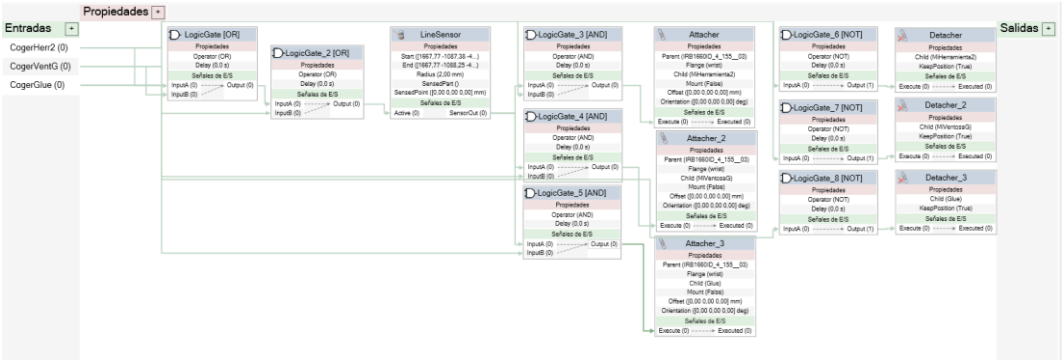


Ilustración 128. Componente Inteligente 'TercerRobot'.

Barreras fotoeléctricas

Con este componente inteligente se consigue parar la célula cada vez que los sensores de las barreras fotoeléctricas detecten un cuerpo. Se cuenta con una salida digital activa a nivel alto (1) que está conectada a una señal que activará una interrupción.

Para la creación de este componente se han necesitado cuatro 'PlaneSensor', que mediante puertas lógicas 'OR' conectan su salida a la salida digital del componente inteligente. Cuando cualquiera de los 'PlaneSensor' detecte un objeto, se mandará un 1 lógico a la salida del componente inteligente.

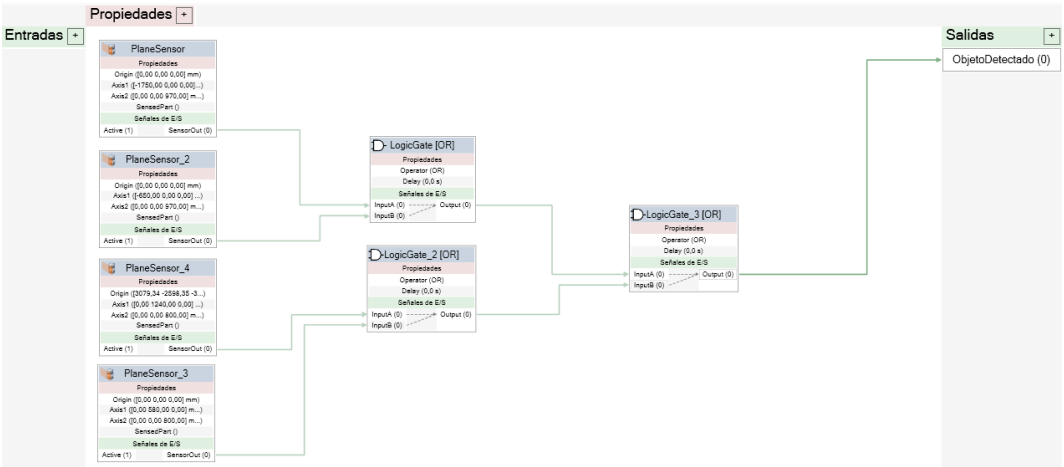


Ilustración 129. Componente inteligente 'Barreras Fotoeléctricas'.

Mesa operario

Mediante de este componente inteligente se simula la acción de un operario sobre la luminaria añadiéndola el cableado correspondiente. Este cuenta con dos señales digitales de entrada y una de salida. Para ello, cuando el robot deje la luminaria en la cinta transportadora que se ubica en la mesa del operario, esta se moverá hasta el final de la cinta donde un operario trabajará sobre ella.

El funcionamiento es el siguiente, la luminaria es movida por la cinta transportadora mediante el bloque 'LinearMove', cuando este se haya ejecutado, es decir, cuando la luminaria haya llegado al final de la cinta, se activará una señal que moverá al operario mediante otro bloque 'LinearMove'.

Una vez el operario ha llegado al lugar dónde debe trabajar con la luminaria se activa un contador de 10 segundos. Una vez ha transcurrido este tiempo el operario habrá terminado de realizar su trabajo y activará una señal ('FinTareaOP'), que moverá la luminaria hasta el inicio de la cinta transportadora, esta señal también será leída en RAPID donde se dará permiso al robot para poder recoger la luminaria de forma segura y poder continuar con el proceso.

La simulación que provoca el cambio de la luminaria sin cableado por una luminaria con cableado se realiza mediante sensores que detectan la pieza a eliminar, que a su vez activarán los bloques encargados de la eliminación ('Sink') y una vez eliminada, se generarán una pieza que cuenta con el cableado mediante el bloque 'Source'.

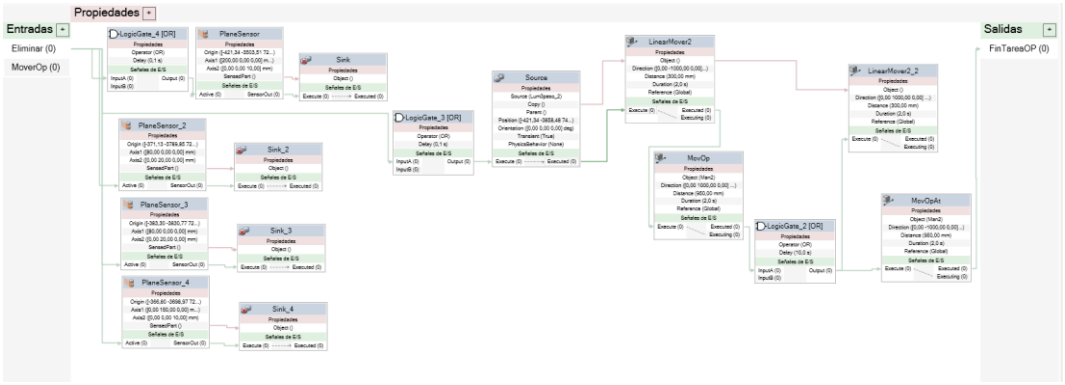


Ilustración 130. Componente Inteligente 'MesaOperario'.

Ventosa y Herramienta

El funcionamiento de los componentes inteligentes de todas las herramientas es el mismo. En el extremo de todas las herramientas se encuentra un sensor ('LineSensor') que podrá detectar piezas cuando se active la señal de entrada

del componente inteligente. Si esta señal de entrada está activada y el sensor detecta una pieza, se activará el bloque 'Attacher' creando una unión entre la herramienta y el elemento detectado.

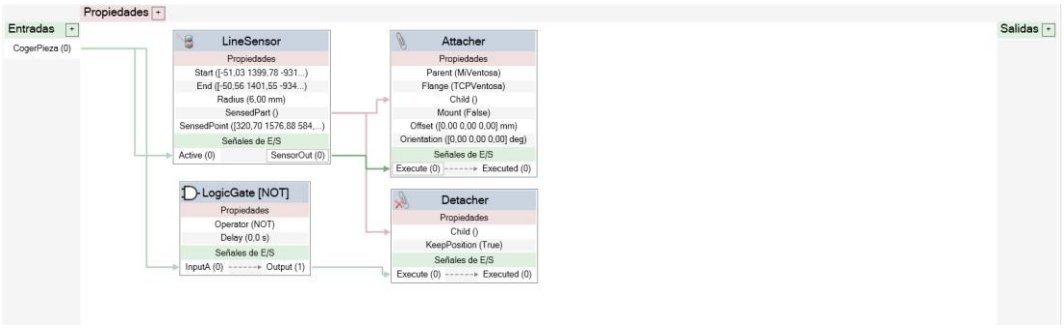


Ilustración 131. Componente Inteligente de las herramientas del robot.

5.2 Programación en Matlab

Una vez se ha explicado cómo se ha realizado la programación de Robotstudio, se procede a explicar la programación en Matlab. Como se ha comentado anteriormente, en este *software* se ha creado una interfaz en App Designer que se comunica mediante OPC con Robotstudio.

5.2.1 Creación del servidor

Para poder acceder a los datos de Robotstudio, primero se debe de crear y configurar el servidor correctamente. Para ello se deben seguir los siguientes pasos:

1º. En el '*Comand Window*' de Matlab debemos escribir 'OpcTool' y se nos abrirá la siguiente pestaña:

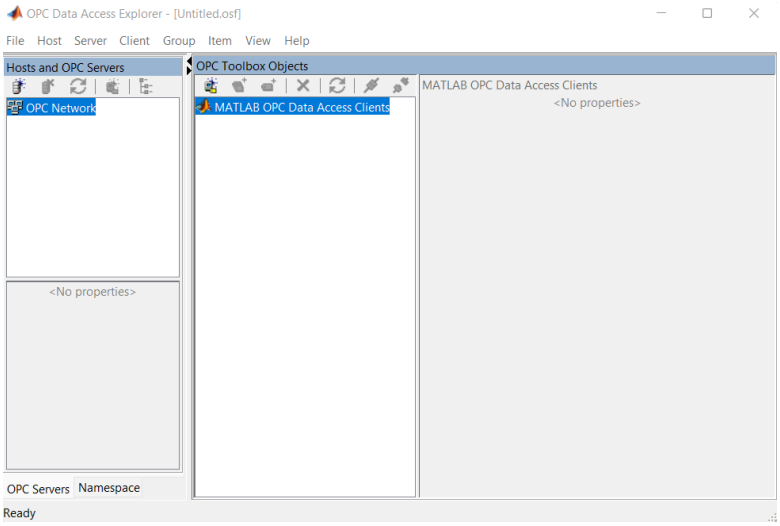


Ilustración 132. Configuración del servidor.

2º. Para este paso es necesario tener arrancada la aplicación 'ABB IRC5 OPC' con el alias del controlador correctamente configurado (como se explicó en el capítulo anterior 3.3.3 ABB IRC5 OPC). Después, debemos dar con el botón derecho en 'OPC Network' y seleccionar 'Add host'. Una vez creado el host nos saldrán los servidores disponibles:

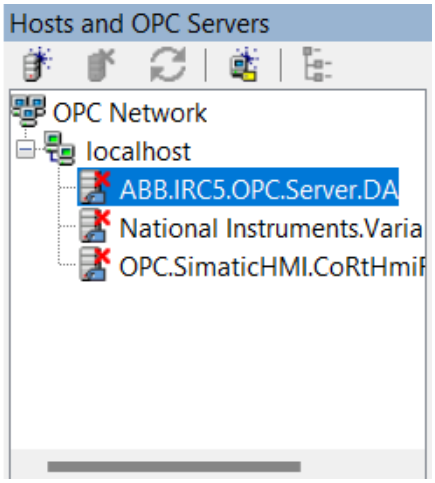


Ilustración 133. Servidores disponibles.

3º. En el servidor 'ABB.IRC5.OPC.SERVER.DA' se pulsa con el botón derecho y se selecciona la opción 'Create client'. Una vez hecho esto, se nos generará la siguiente ventana donde deberemos seleccionar 'Connect' y el proceso de configuración estaría terminado.

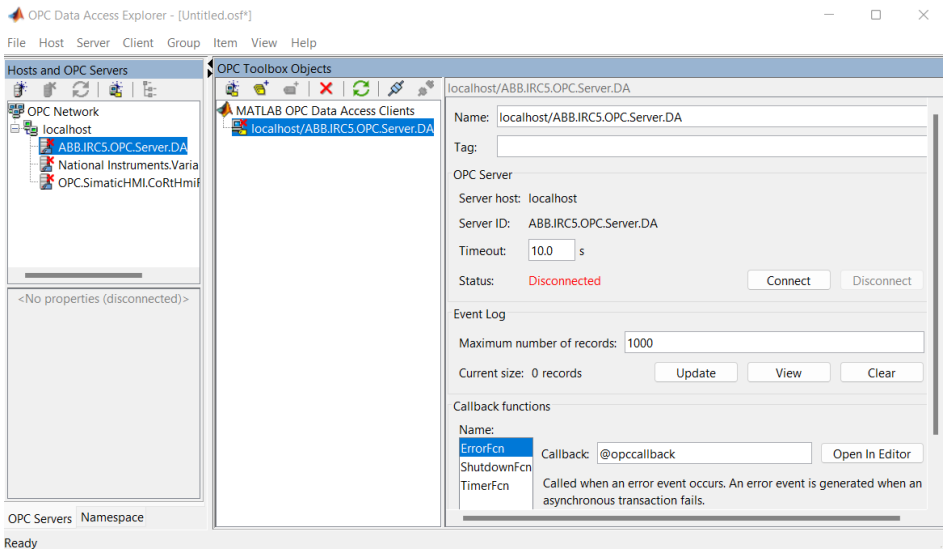


Ilustración 134. Servidor configurado.

Como se explicará más adelante, una vez estemos conectados, desde esta ventana se pueden observar todas las variables disponibles en Robotstudio. El objetivo es modificar el valor de estas variables desde Matlab, por lo que se necesita realizar una programación para poder llevarlo a cabo.

5.2.2 Código realizado en Matlab

Para poder establecer comunicación entre ambos programas se ha tenido que desarrollar un script que desarrolla la transmisión de información entre las variables de Matlab y las de Robotstudio.

A continuación, se procede a explicar el código de Matlab realizado:

Las siguientes líneas de código nos sirven para mostrar al usuario la información del Host y los servidores que hay disponibles utilizando los comandos 'opcserverinfo' 'ServerID'.

```
Host = opcserverinfo('localhost'); %Informacion del Host
ServDisp = Host.ServerID'; %Servidores Disponibles
```

Una vez se ha identificado el servidor al que se debe conectar, con el comando 'opcda' creamos el cliente y con comando 'connect' conectamos el cliente al servidor OPC. En este punto la conexión ya estará establecida, por lo que se podrá comenzar a transmitir información. Se deben especificar las variables que van a ser cambiar su valor, para ello se ha creado un conjunto de variables con el comando 'addgroup' que contiene todas las variables de Robotstudio que van a ser modificadas en Matlab.

```
UsuarioM = opcda('localhost', 'ABB.IRC5.OPC.Server.DA'); %Creamos un cliente
connect(UsuarioM); %Conecta el cliente creado al servidor OPC
Signals = addgroup(UsuarioM); %Grupo de señales que serán modificadas
```

Después, se declaran las variables y se añaden al grupo que se ha creado anteriormente ('Signals') especificando el ID de cada señal. En este punto ya estaría asignada cada variable de Robotstudio a una variable de Matlab.

```
%Señales de Robotstudio
Empezar=additem(Signals,'LAPTOP-AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Empezar');
Robot1On=additem(Signals,'LAPTOP-AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Robot1ON');
Robot2On=additem(Signals,'LAPTOP-AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Robot2ON');
Robot3On=additem(Signals,'LAPTOP-AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Robot3ON');
Robot4On=additem(Signals,'LAPTOP-AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Robot4ON');
```

Para conocer el ID de cada señal de una manera sencilla, se debe ir a la ventana en la que se ha configurado el servidor en el capítulo anterior, añadir la señal a un grupo de variables (señalado en verde) y después de esto nos aparecerá el ID de la señal (señalado en rojo).

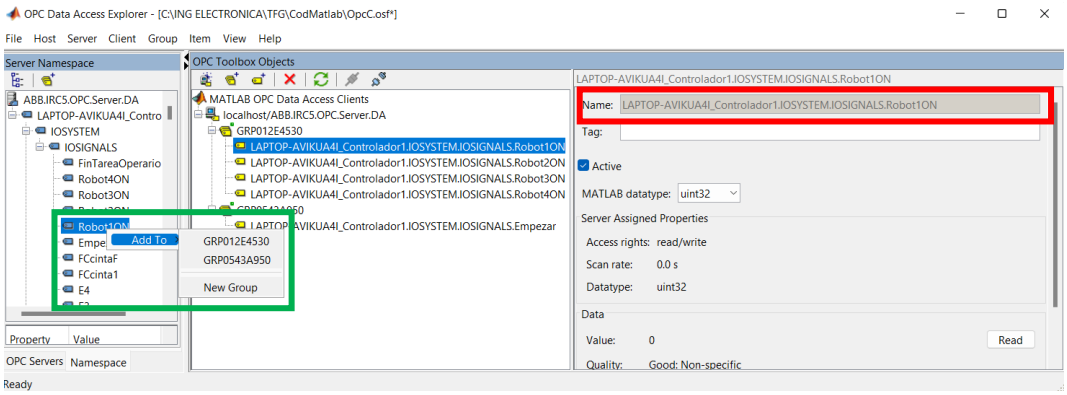


Ilustración 135. ID de cada señal.

El cambio de valor de una variable de Robotstudio desde Matlab se realiza de la siguiente manera:

1º. Con el comando 'evalin' se extrae el valor de la variable 'State' en el espacio de trabajo de Matlab (Workspace) y se guarda en una variable, en este caso en la variable 'Estado'. Esta variable actualiza su valor desde App Designer.
2º. Para escribir el valor de esta variable en Robotstudio se emplea el comando 'Write' que tiene como argumentos la variable que se quiere modificar ('Empezar') y el valor que se le quiere dar a la variable ('Estado').

```
%Escribimos sobre la variable de Robotstudio
Estado=evalin('base','State');
write(Empezar,Estado);
```

Para leer el valor de una variable de Robotstudio empleamos el comando 'read', esta devuelve la estructura de la señal que contiene información como el ID de la señal, el valor de la señal, la marca de tiempo, etc. Esta estructura es guardada en una variable auxiliar.

```
%Leemos la estructura de la señal
R1 = read(Robot1On);
R2 = read(Robot2On);
R3 = read(Robot3On);
R4 = read(Robot4On);
```

Para extraer el valor de la estructura empleamos el comando 'Value' que devolverá un tipo de número 'uint32'. Este valor será convertido a tipo 'double' para que sea más sencillo trabajar con él. Los valores de estas variables serán interpretados desde App Designer.

```
%Extraemos el valor de cada señal
Rob1=double(R1.Value);
Rob2=double(R2.Value);
Rob3=double(R3.Value);
Rob4=double(R4.Value);
```

5.2.3 App Designer

Como se ha comentado en capítulos anteriores la interfaz de usuario se ha realizado en App Designer, que es una aplicación de Matlab.

Para diseñar la interfaz, App Designer cuenta con un menú de elementos que se emplean para el desarrollo de la aplicación. Una vez son seleccionados los elementos que forman la interfaz, la propia aplicación va generando automáticamente código que se corresponde con la distribución y diseño de la interfaz. El código que se genera de forma automática no se va a mostrar.

Primero, se va a mostrar en un diagrama de bloques el funcionamiento de la aplicación.

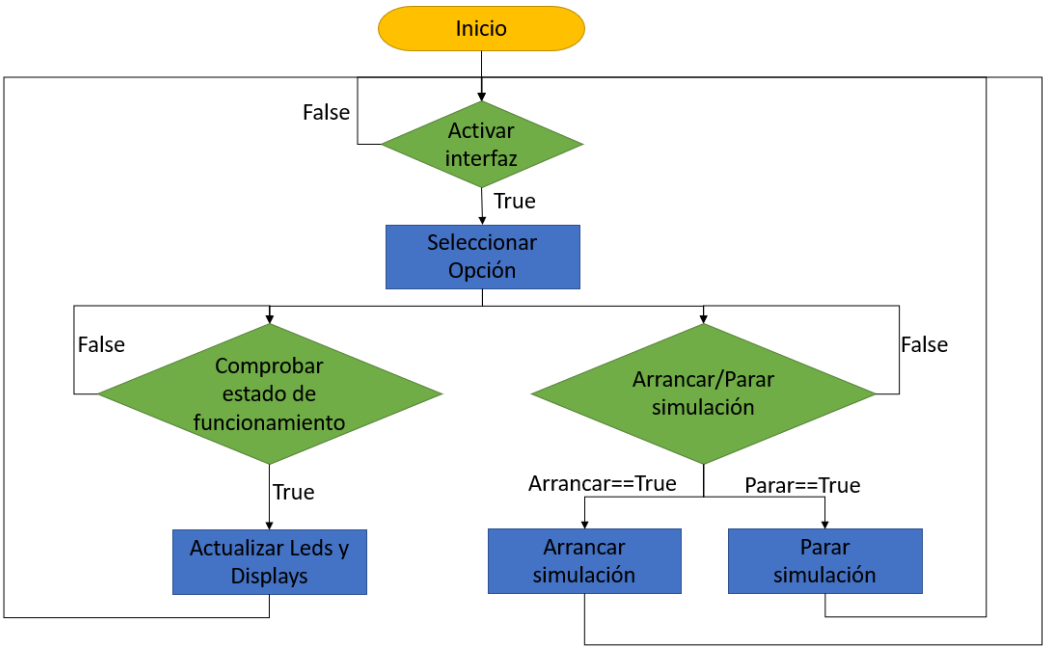


Ilustración 136. Diagrama de bloques interfaz de usuario.

Una vez ha sido mostrado el diagrama de bloques, se va a explicar el código desarrollado y las funciones que han sido creadas.

La interfaz creada está formada por:

- Un switch que nos permite activar la interfaz. Este se ha creado para evitar pulsaciones accidentales.
- Un interruptor que controla el inicio y el fin de la simulación.
- Un botón que cuando se pulsa muestra el estado de los robots, es decir, si están activos realizando una tarea o están inactivos.
- Led que muestra si la interfaz está activada (led en color verde) o desactivada (led en color rojo).
- Leds que muestran si los robots están activos (led en color verde), o no lo están (led en color rojo).
- Led que muestra si la simulación está en marcha (led en color verde) o está parada (led en color rojo).
- Varios *displays* que muestran también el estado de los robots mostrando si están activos o inactivos.
- Para que sea más fácil la identificación de cada robot, se han adjuntado imágenes en la interfaz de cada uno.

StartupFcn (app)

Esta función se va a ejecutar una sola vez, cuando se inicie la simulación. En esta función se añade al *workspace* de Matlab la variable 'State' (muestra el estado de la simulación).

También se encarga de desactivar todos los componentes ya que inicialmente la interfaz se encuentra apagada, por tanto, el Led indicador estará en color rojo, el switch y el botón estarán desactivados. Al representar el estado inicial de la simulación, el estado de los robots se declarará como inactivo.

```
function startupFcn(app)
    assignin('base','State',4);
    app.Lamp.Color='Red';
    app.Switch.Enable='Off';
    app.EJECUTARButton.Enable= 'Off';
    app.Label.Text= 'Inactivo';
    app.Label_2.Text= 'Inactivo';
    app.Label_3.Text= 'Inactivo';
    app.Label_4.Text= 'Inactivo';
end
```

Interruptor Activar/desactivar interfaz

Cuando se pulsa este interruptor se evalúa el estado en el que se encuentra y se guarda en la variable 'Value'. Mediante un switch se evalúa esta variable. Si el interruptor está apagado, el LED estará en color rojo, el switch

correspondiente a la marcha/paro estará deshabilitado y el botón que muestra el estado de los robots también estará deshabilitado.

En caso contrario, todas las opciones estarán habilitadas y el LED lucirá en color verde.

```
function ACTIVARINTERFACESwitchValueChanged(app, event)
    value = app.ACTIVARINTERFACESwitch.Value;
    switch value
        case 'Off'
            app.Lamp.Color='Red';
            app.Switch.Enable='Off';
            app.EJECUTARButton.Enable= 'Off';
        case 'On'
            app.Lamp.Color='Green';
            app.Switch.Enable='On';
            app.EJECUTARButton.Enable= 'On';
    end
end
```

Interruptor Marcha/Paro

En esta función se evalúan los casos posibles del switch marcha/paro. El valor que representa el interruptor se guarda en la variable 'Value' y mediante sentencias condicionales 'IF' y 'ELSEIF' se asignará a la variable 'State' un 1 o un 0 dependiendo si está en marcha o en paro. También se asignará el color verde sí se encuentra en estado de marcha y el color rojo si se encuentra en estado de paro.

Al finalizar estas sentencias condicionales, se hará una llamada a la función de Matlab 'ServOPC' que leerá del *workspace* el valor actualizado de la variable 'State', iniciando o finalizando en Robotstudio la simulación.

```
function SwitchValueChanged(app, event)
    value = app.Switch.Value;
    if strcmp(value, 'Marcha')
        assignin('base','State',1)
        app.LedLamp.Color = 'g';
    elseif strcmp(value, 'Paro')
        assignin('base','State',0)
        app.LedLamp.Color = 'r';
    end
    ServOPC;
end
```

Botón para visualizar estado de los robots

Como se puede observar esta función lo único que hace es que cuando se pulsa el botón, se hace una llamada a la función 'EstadoRobs (app)', la cual se explica a continuación.

```
function EJECUTARButtonPushed(app, event)
    EstadoRobs(app);
end
```

Función que evalúa el estado de los robots

En esta función se realiza una revisión a las variables que definen el estado de los robots. Lo primero que se hace es llamar a la función ServOPC, para que actualice el workspace de Matlab (comunicándose con Robotstudio) con los valores de las variables que se van a evaluar ('Rob1', 'Rob2', 'Rob3' y 'Rob4'), que si su valor es igual a 1 indicarán que el robot está funcionando y en caso contrario, indicarán que no estará funcionando.

Mediante sentencias condicionales 'IF' 'ELSE' se evalúa el valor de las diferentes variables, por ejemplo, si la variable 'Rob1' vale 1 se asigna el color verde al LED del robot 1 y en su display aparecerá la palabra 'Activo'. Sin embargo, si la variable 'Rob1' vale 0, el LED del robot 1 aparecerá en color rojo y en su display la palabra 'Inactivo'. Lo mismo sucederá para las demás variables.

```
function EstadoRobs(app)
    ServOPC;
    if Rob1==1
        app.IRB1600Lamp.Color = 'g';
        app.Label.Text= 'Activo';
    else
        app.IRB1600Lamp.Color = 'r';
        app.Label.Text= 'Inactivo';
    end
    if Rob2==1
        app.IRB1600_2Lamp.Color = 'g';
        app.Label_2.Text= 'Activo';
    else
        app.IRB1600_2Lamp.Color = 'r';
        app.Label_2.Text= 'Inactivo';
    end
    if Rob3==1
        app.IRB1600IDLamp.Color = 'g';
        app.Label_3.Text= 'Activo';
    else
        app.IRB1600IDLamp.Color = 'r';
        app.Label_3.Text= 'Inactivo';
    end
    if Rob4==1
        app.IRB369C1Lamp.Color = 'g';
        app.Label_4.Text= 'Activo';
    else
        app.IRB369C1Lamp.Color = 'r';
        app.Label_4.Text= 'Inactivo';
    end
end
```

6. Funcionamiento de la célula

Una vez se ha explicado cómo se ha desarrollado la célula, en este capítulo se va a mostrar el resultado final de todo el trabajo realizado, exponiendo el funcionamiento de cada aplicación. Para ello se han añadido las capturas más representativas de la ejecución en los diferentes *Softwares*. Primero se mostrará la interfaz de usuario creada en Matlab y posteriormente el resultado en Robotstudio.

6.1 Interfaz de usuario

La interfaz diseñada, en el estado inicial, se muestra en la Ilustración 137. Como se ha explicado anteriormente, el inicio del programa comienza cuando el usuario lo indica desde la interfaz y hasta que no se active la interfaz, permanecerán bloqueadas todas las acciones. Se cuentan con dos interruptores, uno de ellos activa la interfaz (señalado en rojo en la Ilustración 137) y otro pone en marcha o para la simulación (señalado en verde en la Ilustración 137). También se cuenta con un botón (señalado en azul en la Ilustración 137), con el que se actualiza el estado de funcionamiento de los robots. Para señalizar los diferentes estados también se cuentan con varios *displays* y varios leds. En el estado inicial, aunque esté bloqueada la interfaz, se mostrará el estado de funcionamiento, que lógicamente será con todos los robots inactivos.

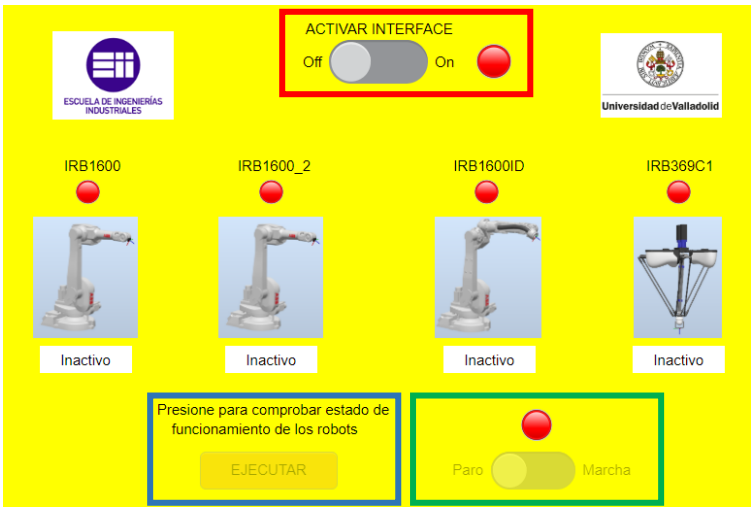


Ilustración 137. Interfaz de usuario en estado inicial.

Una vez se ha activado la interfaz y el usuario indica que comience el montaje, los *displays* y los diferentes leds irán cambiando en función del estado de la simulación. Por ejemplo, en una situación en la que solo estén activos los robots uno y dos y el usuario solicite comprobar el funcionamiento de los robots, la interfaz tendría el siguiente aspecto:

Ilustración 138. Interfaz usuario.

Como se puede comprobar, los leds de los robots 1 y 2 y de los interruptores, han cambiado al color verde y los *displays* muestran el estado 'Activo'.

6.2 Resultados en Robotstudio

Una vez se ha explicado la interfaz de usuario, se procede a mostrar la secuencia del montaje que realizan los robots mediante la captura de las situaciones del montaje más representativas. Como se ha comentado anteriormente, para el comienzo del montaje es necesario que el usuario haya puesto en marcha el proceso desde la interfaz. Una vez hecho esto, el proceso comienza y la bandeja con las piezas empieza a moverse por la cinta transportadora hasta el final de ella en la dirección que indica la flecha:

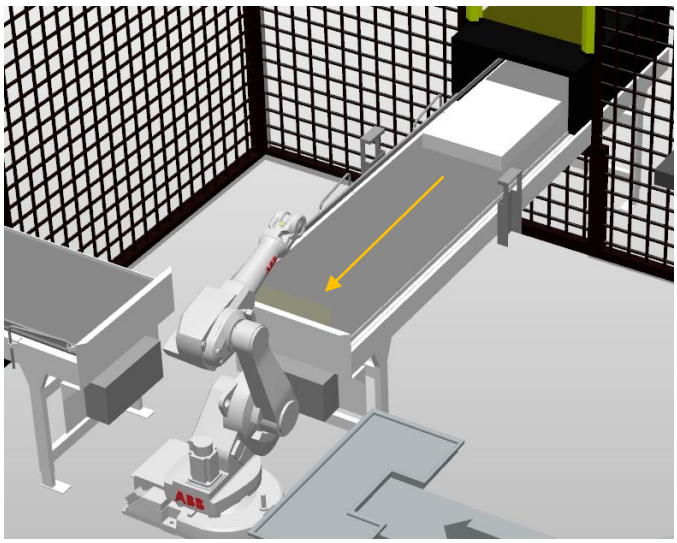


Ilustración 139. Llegada de la bandeja al robot 1.

Una vez está la bandeja en el final de la cinta transportadora, el robot 1 quitará la tapa de la bandeja y comenzará a poner las piezas en la mesa giratoria. Este proceso se muestra en las siguientes imágenes.

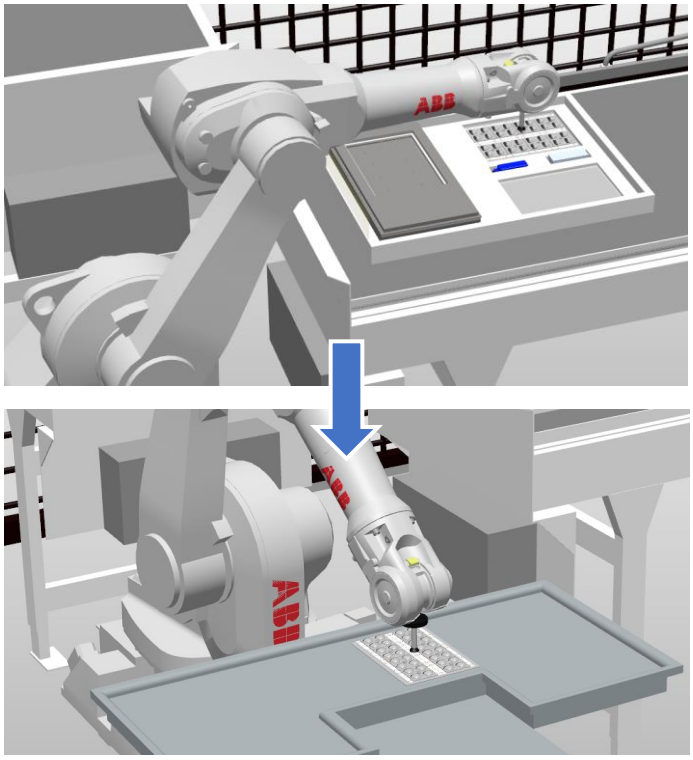


Ilustración 140. Secuencia de traslado de piezas de la cinta transportadora a la mesa giratoria.

El proceso se realizará de la misma forma para todas las piezas y quedarán dispuestas en la mesa giratoria de la siguiente manera:

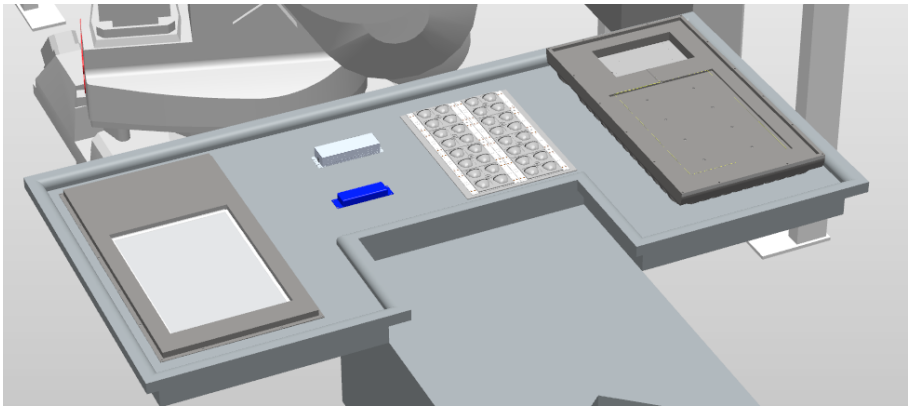


Ilustración 141. Componentes colocados en la mesa giratoria.

Una vez se hayan dejado todas las piezas, la mesa giratoria se moverá a la siguiente posición, donde el robot dos estará esperando para realizar el segundo paso del montaje. El proceso de montaje de este robot es el siguiente, primero colocará el *driver*, la caja de control y la PCB en la luminaria:

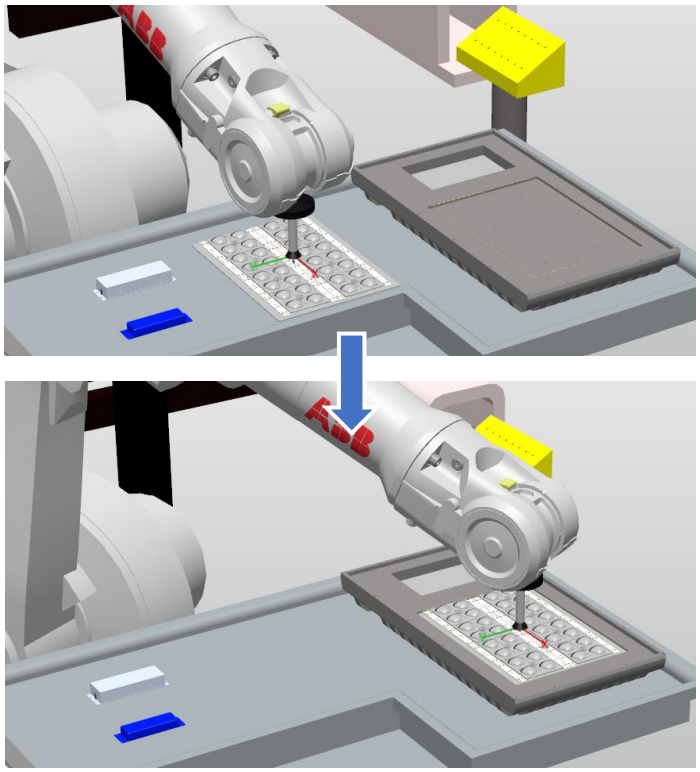


Ilustración 142. Proceso de colocación de componentes en la luminaria.

El proceso es el mismo para el *driver* y la caja de control, quedarían situados de la siguiente manera:

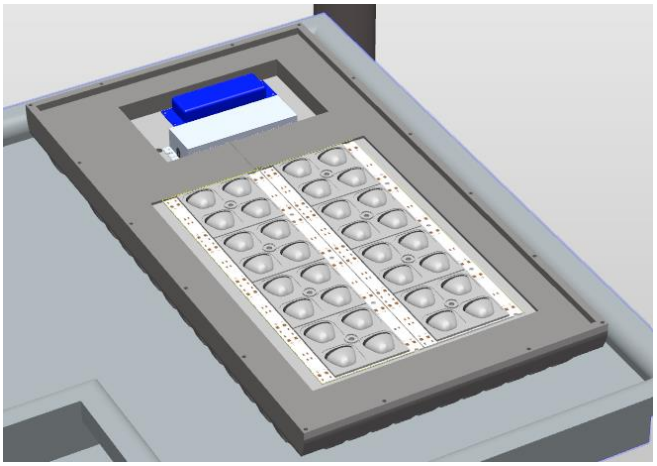


Ilustración 143. Componentes colocados en la base de la luminaria.

Después, fijará estos componentes a la base de la luminaria cogiendo los tornillos del soporte, para ello primero realizará el cambio de herramienta y después irá fijando todos los componentes:

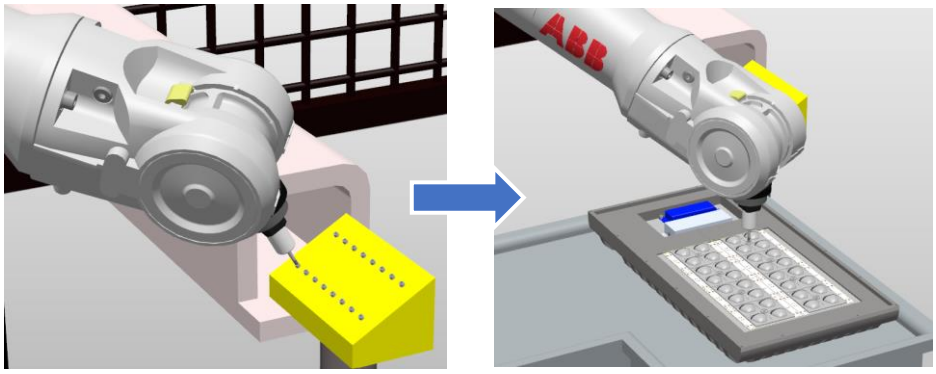


Ilustración 144. Secuencia de la fijación componentes a la luminaria.

Después de que se hayan fijado todos los componentes, la luminaria tendría el siguiente aspecto:

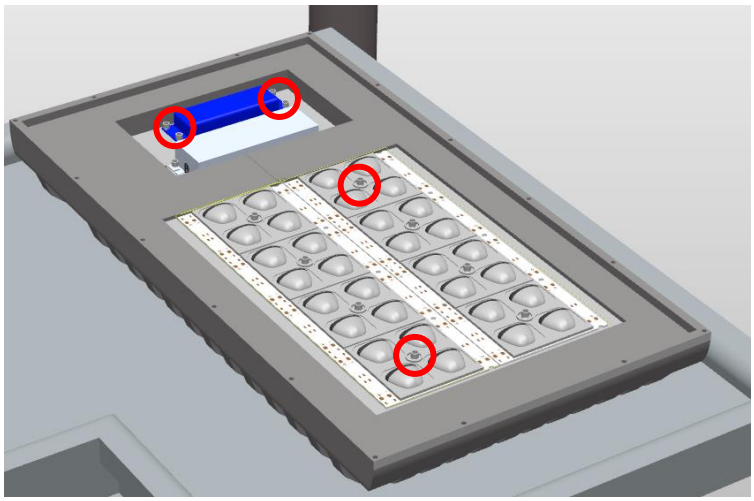


Ilustración 145. Luminaria con los componentes fijados mediante la colocación de tornillos (señalados en rojo).

Una vez el robot dos haya completado la tarea, la plataforma giratoria se moverá hasta la siguiente posición. Lo primero que hará el robot tres será mandar la luminaria a la mesa del operario donde se le añadirá el cableado. En la segunda imagen de la Ilustración 146 se encuentra señalado en rojo el cableado de la luminaria puesta por el operario.

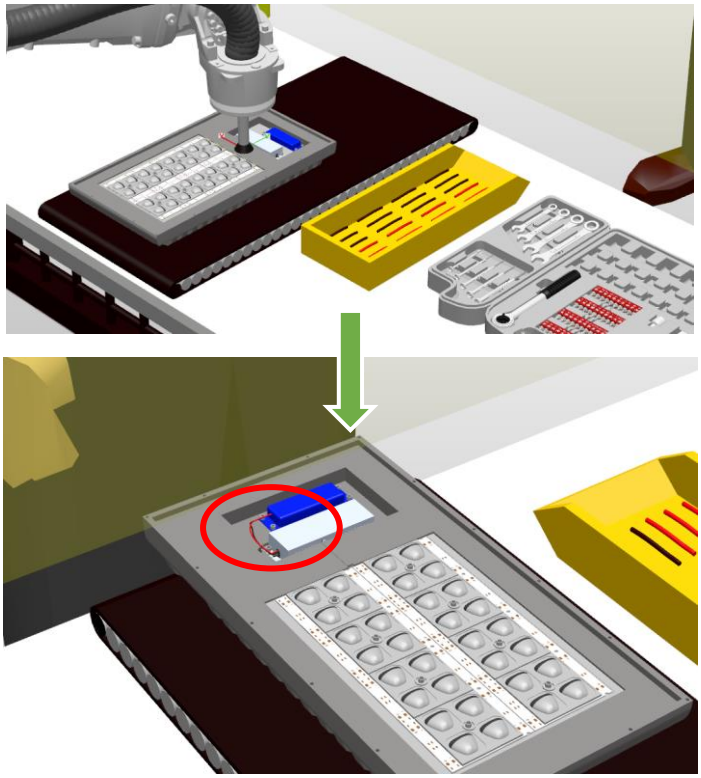


Ilustración 146. Luminaria en la mesa del operario.

Al mismo tiempo que el operario realiza su tarea, el robot fijará el cristal al marco por lo que antes de ello, deberá cambiar de herramienta. A continuación, se muestra el robot realizando esta tarea:

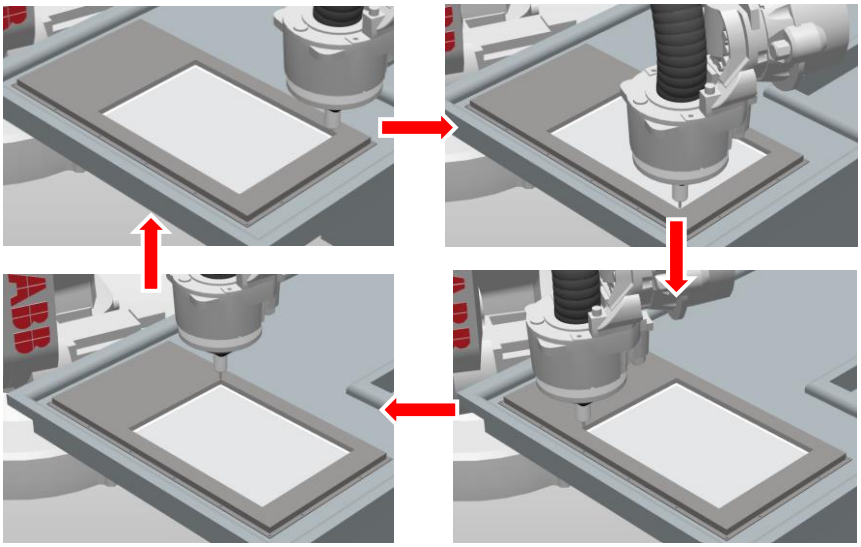


Ilustración 147. Robot fijando el marco a la base de la luminaria.

Cuando el robot termine de fijar el cristal al marco, esperará la señal activada por el operario que indique que ya ha terminado su tarea, e irá a recoger la luminaria para volver a dejarla en la mesa giratoria. Después de esto, cogerá el marco y lo colocará encima de la base.

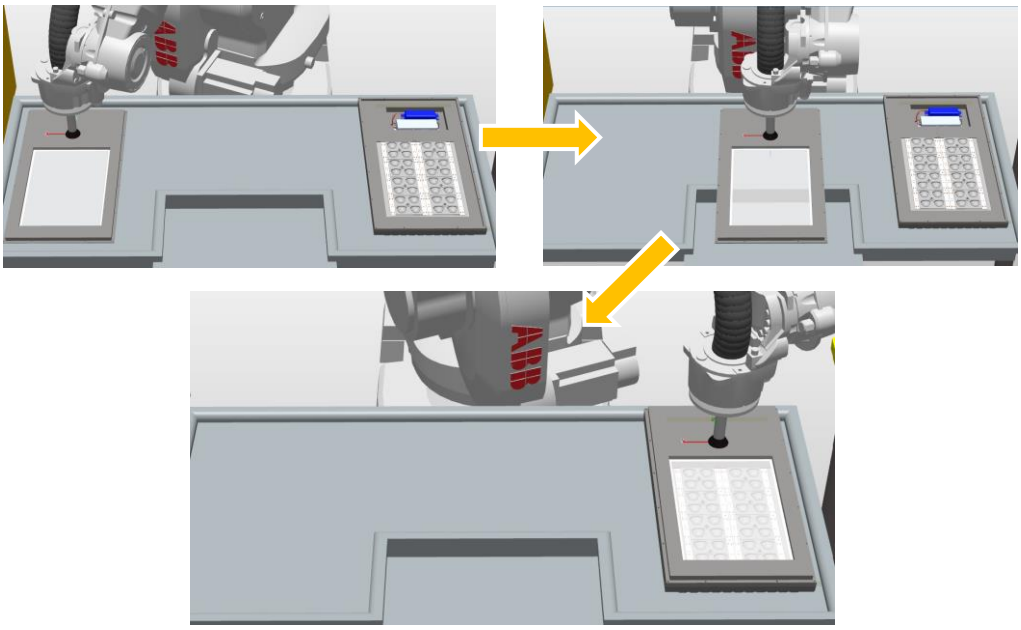


Ilustración 148. Secuencia de colocación del marco encima de la base de la luminaria.

Posteriormente a este paso se fijará el marco a la base. Después la luminaria estará completamente montada y solo faltaría sacarla del sistema.

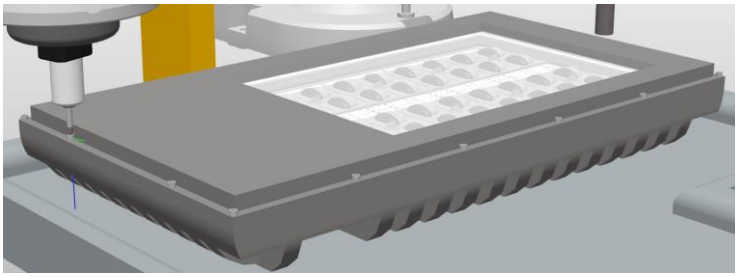


Ilustración 149. Luminaria montada.

Cuando el robot tres acabe su tarea, la mesa giratoria rotará hasta la siguiente posición. Cuando llegue la mesa a la posición cuatro, el robot paralelo trasladará la luminaria de la mesa giratoria hasta una caja situada en una cinta transportadora.

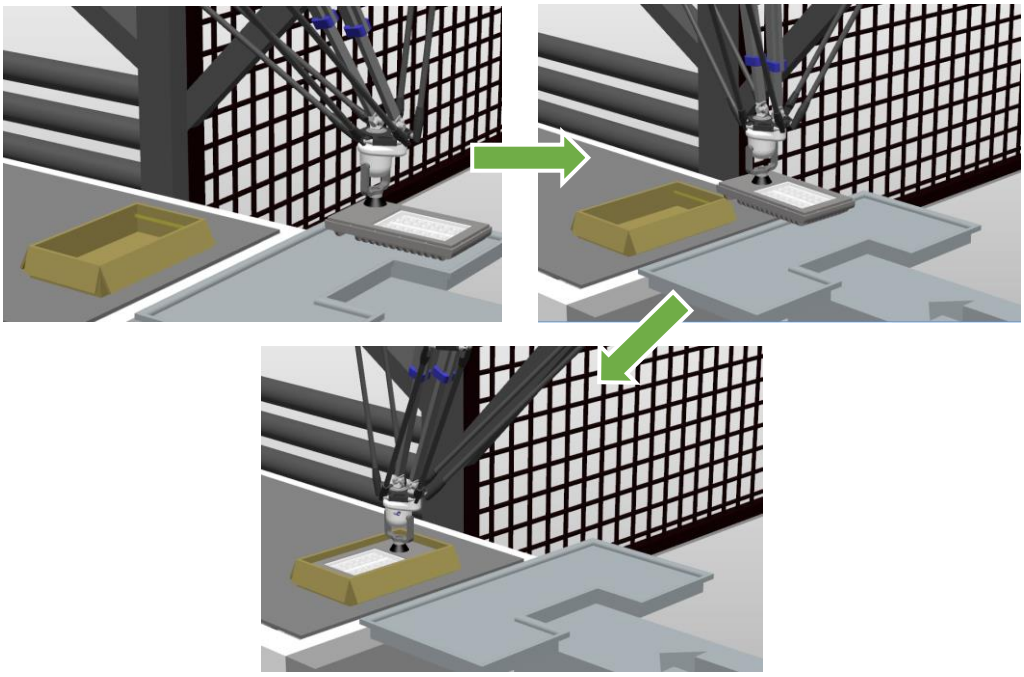


Ilustración 150. Transporte de luminaria desde la plataforma giratoria hasta la cinta transportadora.

Después de depositar la luminaria en la caja, se activará la cinta transportadora y sacará la luminaria del sistema en la dirección que se indica en la siguiente ilustración.

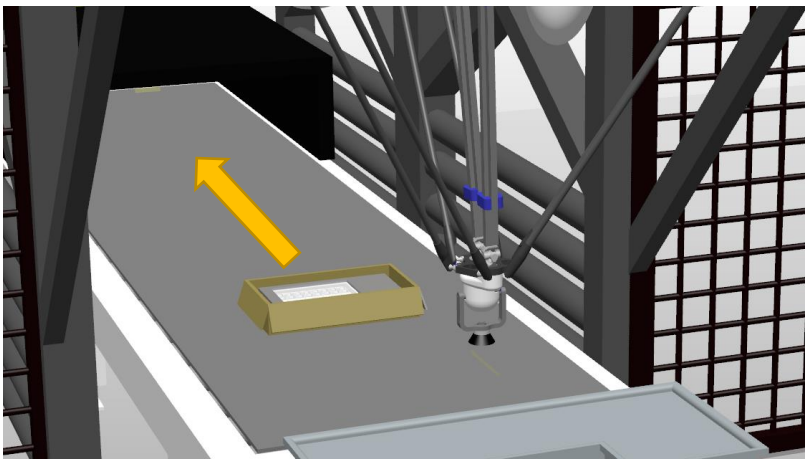


Ilustración 151. Salida del sistema de la luminaria.

Con todo esto, el montaje de una luminaria estaría listo y los robots continuarían trabajando en paralelo hasta que el usuario decidiese para la cadena de montaje.

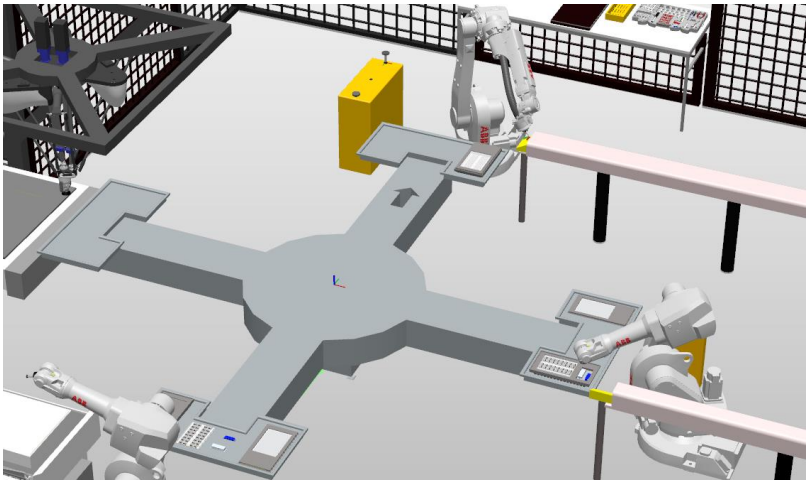


Ilustración 152. Robots realizando sus tareas en paralelo.

7. Conclusiones

Una vez finalizado el Trabajo de Fin de Grado, se extraen una serie de conclusiones relacionadas con la ampliación y aplicación de conocimientos en diferentes programas y en distintos campos de la ingeniería, sobre todo en el campo de la robótica.

En cuanto al *Software* Robotstudio, se ha podido comprobar que se puede realizar cualquier tipo de simulación sin la necesidad de usar robots de forma física, lo que nos ofrece una gran seguridad a la hora de implementar células robotizadas en entornos industriales. Se han aprendido una gran variedad de funciones de RAPID, que es el lenguaje de programación de Robotstudio, optimizando a su vez las que ya se conocían. También se ha tenido que aprender desde cero la programación de *Smart Components* de Robotstudio, ya que no se habían visto anteriormente. Como se ha mostrado este proyecto, los *Smart Components* son una potente herramienta con la que se puede simular cualquier tipo de acción real (movimiento de piezas por cintas transportadoras, ventosas, etc.).

Como se ha visto a lo largo de este proyecto, el diseño de objetos en 3D ha tenido un papel muy importante. Para el diseño de la célula y de todos los componentes se empleó Autodesk Inventor, *software* que se ha tenido que aprender a utilizar debido a que no se había empleado anteriormente. Una vez finalizado el proyecto, se puede afirmar que se ha logrado tener un elevado conocimiento sobre este *software* y que las posibilidades que ofrece en cuanto al diseño son muy amplias.

A la hora de realizar este proyecto se han afrontado una serie de tareas bastante laboriosas:

- El estudio de la colocación de todos los componentes diseñados en zonas que estuviesen al alcance de los robots (controlando su posición y orientación), consiguiendo que no tuviesen problemas a la hora de alcanzar estos componentes para poder manipularlos.
- La programación para sincronizar los cuatro robots.
- La programación de todas las tareas que deben realizar ya que cada robot ejecutaba movimientos diferentes.

Para la creación de la interfaz de usuario se empleó Matlab, software que se ha utilizado a lo largo de la carrera, y cuyos conocimientos se han ampliado descubriendo nuevas opciones que ofrece este potente software como, por ejemplo, la posibilidad de comunicarse con otros programas mediante el protocolo de comunicaciones OPC. Esto es una práctica muy útil en entornos industriales y que no se había tenido oportunidad de hacer hasta el momento.

Llegado a este punto, se puede afirmar que se han cumplido todos los objetivos que se habían propuesto al inicio del proyecto.

8. Líneas futuras

Para aquellos que estén interesados en el proyecto, se proponen las siguientes líneas futuras:

- **Añadir nuevos montajes a la célula robotizada:** para este proyecto se decidió realizar el montaje de una luminaria, pero aprovechando que la programación de la sincronización de los cuatro robots y que elementos como la plataforma giratoria ya están hechos y diseñados, se podrían implementar una gran variedad de montajes.
- **Mejora de la interfaz de usuario:** podrían añadirse nuevas funciones en la aplicación que se ha realizado.
- **Estudiar la comunicación TCP/IP:** realizar la comunicación entre programas empleando este protocolo y comparar los resultados conseguidos a través de la comunicación OPC.
- **Implementar la célula en un sistema real:** como se ha comentado a lo largo del proyecto, el diseño de esta célula se ha realizado con el fin de buscar el máximo parecido a un sistema real, estudiando la normativa actual sobre instalaciones robotizadas. Los códigos de programación de todos los robots servirían también para un entorno físico, por lo que, como línea futura de este proyecto se propone la implementación del entorno simulado a un entorno físico.

9. Bibliografía

[1] A. Barrientos, L. Felipe Peñín, C. Balaguer y R. Aracil, FUNDAMENTOS DE ROBÓTICA, Madrid: McGraw-Hill, 1997.

[2] «Temas para la educación: Herón de Alejandría. Un gran tecnólogo en la,» Mayo 2010. Available: https://www.feandalucia.ccoo.es/docu/p5sd7206.pdf. ISNN: 1989-4023. [Último acceso: 06 Marzo 2022].

[3] J. Ruiz-de-Garibay, «Robótica: Estado del arte,» ACADEMIA. Accelerating the world research. [Último acceso: 06 Marzo 2022].

[4] P. S. S. y. R. H. Alberto Brunete, Introducción a la Automatización Industrial, Madrid: Bookdown, 2020.

[5] «Desde el pato de Vaucanson a los robots de limpieza: así ha evolucionado la robótica desde sus inicios,» XATAKA, 01 Febrero 2018. Available: https://www.xataka.com/n/desde-el-pato-de-vaucanson-a-los-robots-de-limpieza-asi-ha-evolucionado-la-robotica-desde-sus-inicios. [Último acceso: 10 Marzo 2022].

[6] D. G. Guzmán, «Historia de la automatización,» Preceden. Available: https://www.preceden.com/timelines/717427-historia-de-la-automatizaci-n-brayan-daniel-guti-rrez-guzm-n-1682755. [Último acceso: 10 Marzo 2022].

[7] «La Robótica,» Timetoas. Available: https://www.timetoast.com/timelines/la-robotica-a0c6a9d4-f648-4394-bde4-9be7ca58dc42. [Último acceso: 10 Marzo 2022].

[8] «Guía de robótica,» Askix. Available: https://www.askix.com/guia-de-robotica_2.html. [Último acceso: 10 Marzo 2022].

[9] «FALLECIÓ JOSEPH ENGELBERGER, EL PADRE DE LA ROBÓTICA,» REPORTERO INDUSTRIAL, Diciembre 2015. Available: https://www.reporteroindustrial.com/temas/Fallecio-Joseph-Engelberger,-el-padre-de-la-robotica+109475. [Último acceso: 10 Marzo 2022].

[10] J. A. Ávila Herrero y A. Herreros López, «Modelado de una célula robótica con fines educativos usando el programa Robot-Studio (Trabajo de Fin de Grado),» Universidad de Valladolid, Escuela de Ingenierías Industriales, 2015. Available:

https://uvadoc.uva.es/handle/10324/14441. [Último acceso: 01 Abril 2022].

[11] G. Muinelo Garrido y J. C. Fraile Marinero, «Simulación de una célula robotizada para la fabricación de piezas mecanizadas en un entorno industrial, utilizando RobotStudio (Trabajo de Fin de Grado),» Universidad de Valladolid, Escuela de Ingenierías Industriales, 2015. Available: http://uvadoc.uva.es/handle/10324/13173. [Último acceso: 01 Abril 2022].

[12] Á. Galindo de Santos y A. Herreros López, «Control remoto de una Célula robotizada mediante tecnología WI-FI (Trabajo de Fin de Grado),» Universidad de Valladolid, Escuela de Ingenierías Industriales, 2016. Available: http://uvadoc.uva.es/handle/10324/17056. [Último acceso: 01 Abril 2022].

[13] R. Sancho García y A. Herreros López, «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 (Trabajo de Fin de Grado),» Universidad de Valladolid, Escuela de Ingenierías Industriales, 2021. Available: https://uvadoc.uva.es/handle/10324/47254. [Último acceso: 01 Abril 2022].

[14] A. Sancho Gil y J. C. Fraile Marinero, «Simulador de movimientos de herramientas quirúrgicas en operaciones de cirugía laparoscópica con robots (Trabajo de Fin de Grado),» Universidad de Valladolid, Escuela de Ingenierías Industriales, 2022. Available: https://uvadoc.uva.es/handle/10324/53083. [Último acceso: 01 Abril 2022].

[15] «Catálogo de Visualización de Datos,» Diagrama de Flujo. ID de documento: 3HAC032104-005. [Último acceso: 10 Junio 2022].

[16] «ROBOTICS. Manual del operador. RobotStudio. ABB,» 28 Febrero 2022. ID de documento: 3HAC032104-005. [Último acceso: 05 Abril 2022].

[17] «MathWorks,» MATLAB. Available: https://es.mathworks.com/products/matlab.html. [Último acceso: 10 Abril 2022].

[18] «Industrial Communication Toolbox,» MathWorks. MATLAB. Available: https://es.mathworks.com/products/industrial-communication.html?s_tid=srchtitle_opc%20toolbox_1. [Último acceso: 05 Abril 2022].

[19] M. Rodríguez, «Protocolo OPC UA. Características y aplicaciones en SCADA,» INESEM, 22 Febrero 2016. Available: https://www.inesem.es/revistadigital/gestion-integrada/protocolo-opc-ua-caracteristicas-y-aplicaciones/#:~:text=Actualmente%20este%20est%C3%A1ndar%20ha%20evolucionado,Unified%20Architecture%20(OPC%20UA).. [Último acceso: 07 Abril 2022].

[20] S.-H. L. Wolfgang Mahnke, «Arquitectura OPC unificada. Revista ABB,» Febrero 2009. Available: http://infoplc.net/files/documentacion/comunicaciones/infoplc_net_56_61_3M903_SPA72dpi.pdf. [Último acceso: 20 Abril 2022].

[21] «ROBOTICS. Application Manual. IRC5 OPC UA Server. ABB.,» 01 Marzo 2021. Document ID: 3HAC074394-001. [Último acceso: 20 Abril 2022].

[22] F. ARRIBAS, «NORMAS TÉCNICAS EN SEGURIDAD ROBÓTICA. UNE,» 28 Abril 2017. Available: https://issga.xunta.es/export/sites/default/recursos/descargas/documentacion/material-formativo/relatorios/2017_05_CO_Industria_4.0_Arribas.pdf. [Último acceso: 03 Abril 2022].

[23] «Guía Técnica de seguridad en Robótica,» CEPYME Aragón. Available: https://higieneyseguridadlaboralcvs.files.wordpress.com/2012/09/guc3ada-tc3a9cnica-de-seguridad-en-robc3b3tica.pdf. [Último acceso: 25 Abril 2022].

[24] «ROBOTICS. Product specification. IRB 1600/1660. ABB,» 01 Junio 2022. Document ID: 3HAC023604-001. [Último acceso: 04 Junio 2022].

[25] «ROBOTICS. IRB 1600. The highest performance 10 kg robot. ABB,» Document ID: PR10282EN. [Último acceso: 10 Junio 2022].

[26] «ROBOTICS. IRB 1660ID. High performance ID robot for arc welding and machine tending. ABB,» Document ID: ROB0337EN. [Último acceso: 10 Junio 2022].

[27] R. Aracil, R. J. Saltarén, J. M. Sabater y Ó. Reinoso, «Robots Paralelos: Máquinas con un Pasado para una Robótica del Futuro,» Univerdad Politécnica de Valencia, 27 Septiembre 2010. Available: http://ojs.upv.es/index.php/RIAI/article/view/8105. [Último acceso: 10 Junio 2022].

[28] H. A. M. Avalos, «Robots Paralelos, Conceptos y Aplicaciones. Universidad Politécnica de Madrid,» Slideshare, 12 Septiembre 2012. Available:

https://es.slideshare.net/htrmoreno/robots-paralelos . [Último acceso: 10 Junio 2022].

[29] «IRB 360 FlexPicker,» ABB. Available: https://new.abb.com/products/robotics/industrial-robots/irb-360. [Último acceso: 10 Junio 2022].

[30] W. A. B. Robert L. Powell, «Thermal Conductivity of Metals and Alloys at Low Temperatures: A Review of the Literature,» 01 Septiembre 1954. Available: https://books.google.es/books?hl=es&lr=&id=zVVIAQAAIAAJ&oi=fnd&pg=PA1&dq=thermal+conductivity+of+metals&ots=h0fUFTkxJE&sig=sK9lofkj6jrA9UrHhiPdOh8SW8o#v=onepage&q=thermal%20conductivity%20of%20metals&f=false. [Último acceso: 03 Mayo 2022].

[31] «ROBOTICS. Manual de referencia técnica. Instrucciones,funciones y tipos de datos de RAPID. ABB,» 28 Febrero 2022. ID de documento: 3HAC065038-005. [Último acceso: 11 Junio 2022].

[32] «UNE. Normalización española,» Available: https://www.une.org/encuentra-tu-norma. [Último acceso: 30 Abril 2022].

10. Anexos

En estos anexos se muestra parte del código utilizado tanto en Robotstudio como en Matlab y los planos de los componentes diseñados.

10.1 Código desarrollado en Robotstudio

A continuación, se muestra parte del código desarrollado en RAPID, lenguaje de programación de Robotstudio, representando la programación de uno de los robots. No se adjunta el código completo de los demás robots debido a la extensión de estos.

Tarea Robot 1

```
MODULE Tarea_1
   !Variable tooldata
   PERS tooldata
TCPVentosa:=[TRUE,[[0,0,54.763],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
   !Variables robtarget
   CONST robtarget AbajoLum:=[[-
74.223,730.614,685.66],[0,0.707106781,0.707106781,0],[1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget
ArribaLum:=[[97.966,732.494,702.251],[0,0.707106781,0.707106781,0],[0,0,1,0]
,[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget BaseCaja:=[[-
25.559,880.359,673.5],[0,0.707106781,0.707106781,0],[1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget CajaControl:=[[-
25.339,968.947,677.17],[0,0.707106781,0.707106781,0],[1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget
Cristal:=[[89.532,1029.725,667.88],[0,0.707106781,0.707106781,0],[0,0,1,0],[9E
+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget Driver:=[[-
24.06,1089.711,678.476],[0,0.707106781,0.707106781,0],[1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget PlacaLeds:=[[-
131.769,1029.473,670.196],[0,0.707106781,0.707106781,0],[1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget
PosIni:=[[806.295852188,0,929.002986412],[0.5,0,0.866025404,0],[0,0,0,0],[9E+
09,9E+09,9E+09,9E+09,9E+09,9E+09]];
   CONST robtarget Target_20:=[[-
25.31,890.625,733.5],[0,0.707106781,0.707106781,0],[1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    CONST robtarget PosIni1:=[[-
25.470508296,711.492872916,1083.659495596],[0.418768748,-
0.574027859,0.553842853,0.434030928],[1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_50:=[[-
691.145407322,170.831168165,1083.659495596],[0.072892902,-
0.791805564,0.096404551,0.598695863],[1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget
Target_60:=[[0.066478902,764.639320268,1068.733603874],[0.426486657,-
0.56400114,0.564050178,0.426449579],[0,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_70:=[[-948.345,241.829,690.383],[0,1,0,0],[1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_80:=[[555.009,-344.022,419.786],[0,-
0.707106781,0.707106781,0],[-1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_100:=[[729.286,320.703,440.66],[0,-
0.707106781,0.707106781,0],[0,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_110:=[[722.471,-346.421,414.66],[0,1,0,0],[-
1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_120:=[[662.309,-
110.059,417.991],[0,0.707106781,0.707106781,0],[-
1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_130:=[[565.39,-
109.438,420.757],[0,0.707106781,0.707106781,0],[-
1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_140:=[[614.903,79.957,412.312],[0,1,0,0],[0,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_150:=[[-
957.037,231.1,663.5],[0,0,1,0],[1,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+
09]];
    CONST robtarget
Target_10:=[[806.291651246,0,929],[0.5,0,0.866025404,0],[0,0,0,0],[9E+09,9E+
09,9E+09,9E+09,9E+09,9E+09]];
    !Variables sincronizacion
    PERS tasks Tarea1{2}:=["T_ROB1"],["T_ROB2"]];
    PERS tasks Tarea2{3}:=["T_ROB1"],["T_ROB2"],["T_ROB3"]];
    PERS tasks
Tarea3{4}:=["T_ROB1"],["T_ROB2"],["T_ROB3"],["T_ROB4"]];
    PERS tasks
TareaF{4}:=["T_ROB1"],["T_ROB2"],["T_ROB3"],["T_ROB4"]];
    VAR syncident sync1;
    VAR syncident sync2;
    VAR syncident sync3;
    VAR syncident sync4;
```

```
VAR syncident sync5;
VAR syncident sync6;
VAR syncident sync7;
VAR syncident sync8;
VAR syncident sync9;
VAR syncident sync10;
!Variables numéricas
VAR num cond:=0;
VAR num cont:=0;
!Variables interrupciones
Local VAR intnum Istop;
Local VAR intnum Parada;

PROC main()
   MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
   ResetSeñales;
   !Reset Eliminar;
   !Para eliminar después de la simulacion las piezas generadas
   !y evitar problemas de memoria
   SetDo Eliminar,1;
   ! Esperamos la señal de la interfaz de MATLAB a empezar el programa
   WaitDO Empezar,1;
   WHILE TRUE DO
      waittime 0.5;
      Interrupciones;
      IF cont=0 THEN
         Estado0;
      ELSEIF cont=1 THEN
         Estado1;
      ELSEIF cont=2 THEN
         Estado2;
      ELSEIF cont=3 THEN
         Estado3;
      ELSEIF cont>=4 THEN
         WHILE TRUE do
            PrimerPaso;
            SegundoPaso;
            TercerPaso;
            CuartoPaso;
            cont:=cont+1;
         ENDWHILE
      ENDIF
      cont:=cont+1;
   ENDWHILE
ENDPROC

PROC Interrupciones()
```

```
    IDelete Istop;
    IDelete Parada;
    CONNECT Istop WITH trap_stop;
    CONNECT Parada WITH trap_parada;
    ISignalDO Empezar,0,Istop;
    ISignalDI SensorFotoelectrico,1,Parada;
ENDPROC

LOCAL TRAP trap_stop
    Tpwrite "Parada de emergencia";
    Reset Robot1ON;
    EXIT;
ENDTRAP

LOCAL TRAP trap_parada
    Tpwrite "Parada de emergencia";
    Reset Robot1ON;
    EXIT;
ENDTRAP

PROC Estado0()
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.2;
    SetDo Robot1ON,1;
    MovimientoRobot1;
    Waitrob\Zerospeed;
    setdo s1,1;
    reset s1;
ENDPROC

PROC Estado1()
    WaitSyncTask sync1,Tarea1;
    CogerParteAbajo;
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.5;
    MovimientoRobot1;
    CogerParteAbajo;
    WaitSyncTask sync2,Tarea1;
ENDPROC

PROC Estado2()
    WaitSyncTask sync3,Tarea2;
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.5;
```

```
    MovimientoRobot1;
    CogerParteAbajo;
    WaitRob\inpos;
    WaitSyncTask sync4,Tarea2;
ENDPROC

PROC Estado3()
    WaitSyncTask sync5,Tarea3;
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.5;
    MovimientoRobot1;
    CogerParteAbajo;
    WaitRob\inpos;
    WaitSyncTask sync6,Tarea3;
ENDPROC

PROC PrimerPaso()
    waitdi e4,1;
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.5;
    MovimientoRobot1;
    CogerParteAbajo;
    WaitSyncTask sync7,TareaF;
ENDPROC

PROC SegundoPaso()
    setdo s1,1;
    reset s1;
    waitdi e1,1;
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.5;
    MovimientoRobot1;
    CogerParteAbajo;
    WaitRob\inpos;
    WaitSyncTask sync8,TareaF;
ENDPROC

PROC TercerPaso()
    waitdi e2,1;
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.5;
    MovimientoRobot1;
    CogerParteAbajo;
```

```
    WaitRob\inpos;
    WaitSyncTask sync9,TareaF;
ENDPROC

PROC CuartoPaso()
    waitdi e3,1;
    setdo s5,1;
    waitdi FCcinta1,1;
    waittime 0.5;
    MovimientoRobot1;
    CogerParteAbajo;
    WaitRob\inpos;
    WaitSyncTask sync10,TareaF;
ENDPROC

PROC MovimientoRobot1()
    waittime 0.5;
    reset s5;
    SetDO S6,1;
    MoveL offs(Target_20,0,0,10),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(Target_20,0,0,3),v10,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(Target_20,0,0,230),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL PosIni1,v500,z100,TCPVentosa\WObj:=wobj0;
    MoveL offs(Target_50,0,0,35),v500,z100,TCPVentosa\WObj:=wobj0;
    MoveJ offs(Target_70,0,0,150),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL Target_70,v100,z100,TCPVentosa\WObj:=wobj0;
    WaitRob\inpos;
    reset S6;
    waittime 1.5;
    MoveJ offs(Target_70,0,0,30),v100,fine,TCPVentosa\WObj:=wobj0;

    !Empezamos a coger placaLeds
    MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
    !Activar señal ventosa
    SetDO S6,1;
    MoveL offs(PlacaLeds,0,0,30),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(PlacaLeds,0,0,5),v10,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(PlacaLeds,0,0,20),v100,fine,TCPVentosa\WObj:=wobj0;
    MoveJ PosIni1,v500,z100,TCPVentosa\WObj:=wobj0;
    MoveJ PosIni,v500,z100,TCPVentosa\WObj:=wobj0;
    MoveJ offs(Target_140,0,0,25),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(Target_140,0,0,4),v10,fine,TCPVentosa\WObj:=wobj0;
    WaitRob\inpos;
    reset S6;
    waittime 1;
    MoveJ offs(Target_140,0,0,25),v1000,fine,TCPVentosa\WObj:=wobj0;
```

```
!Empezamos a coger parte de arriba
MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
WaitRob\inpos;
SetDO S6,1;
MoveL offs(ArribaLum,0,0,50),v500,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(ArribaLum,0,0,3),v100,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(ArribaLum,0,0,50),v500,fine,TCPVentosa\WObj:=wobj0;
MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
MoveJ PosIni,v1000,z100,TCPVentosa\WObj:=wobj0;
MoveJ offs(Target_80,0,0,25),v500,fine,TCPVentosa\WObj:=wobj0;
MoveL Target_80,v10,fine,TCPVentosa\WObj:=wobj0;
WaitRob\inpos;
reset S6;
waittime 1;

!Empezamos a coger Parte de abajo de la lum
MoveL PosIni,v1000,z100,TCPVentosa\WObj:=wobj0;
MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
WaitRob\inpos;
SetDO S6,1;
MoveL offs(AbajoLum,0,0,50),v1000,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(AbajoLum,0,0,3),v100,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(AbajoLum,0,0,50),v1000,fine,TCPVentosa\WObj:=wobj0;
MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
MoveJ PosIni,v1000,z100,TCPVentosa\WObj:=wobj0;
MoveJ offs(Target_100,0,0,25),v500,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(Target_100,0,0,0),v10,fine,TCPVentosa\WObj:=wobj0;
WaitRob\inpos;
reset S6;
!Empezamos a coger CajaControl
MoveL PosIni,v1000,z100,TCPVentosa\WObj:=wobj0;
MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
WaitRob\inpos;
SetDO S6,1;
MoveL offs(CajaControl,0,0,30),v1000,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(CajaControl,0,0,3),v100,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(CajaControl,0,0,30),v1000,fine,TCPVentosa\WObj:=wobj0;
MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
MoveJ PosIni,v1000,z100,TCPVentosa\WObj:=wobj0;
MoveJ offs(Target_120,0,0,25),v500,fine,TCPVentosa\WObj:=wobj0;
MoveL offs(Target_120,0,0,5),v10,fine,TCPVentosa\WObj:=wobj0;
WaitRob\inpos;
reset S6;
!Empezamos a coger Driver
MoveL PosIni,v1000,z100,TCPVentosa\WObj:=wobj0;
MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
WaitRob\inpos;
```

```
    SetDO S6,1;
    MoveL offs(Driver,0,-20,20),v1000,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(Driver,0,-20,3),v100,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(Driver,0,-20,20),v1000,fine,TCPVentosa\WObj:=wobj0;
    MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
    MoveJ PosIni,v1000,z100,TCPVentosa\WObj:=wobj0;
    MoveJ offs(Target_130,0,0,25),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(Target_130,0,0,5),v10,fine,TCPVentosa\WObj:=wobj0;
    WaitRob\inpos;
    reset S6;

    !Empezamos a coger Cristal
    MoveL PosIni,v1000,z100,TCPVentosa\WObj:=wobj0;
    MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
    WaitRob\inpos;
    SetDO S6,1;
    MoveL offs(Cristal,0,0,50),v1000,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(Cristal,0,0,3),v100,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(Cristal,0,0,50),v1000,fine,TCPVentosa\WObj:=wobj0;
    MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
    MoveJ PosIni,v500,z100,TCPVentosa\WObj:=wobj0;
    MoveJ offs(Target_110,0,0,25),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL Target_110,v10,fine,TCPVentosa\WObj:=wobj0;
    WaitRob\inpos;
    reset S6;
    waittime 2;
    MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
ENDPROC

PROC CogerParteAbajo()
    !Empezamos a coger Parteabajo de la caja
    SetDO S6,1;
    MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
    MoveL offs(BaseCaja,0,0,10),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL offs(BaseCaja,0,0,3),v10,fine,TCPVentosa\WObj:=wobj0;
    setdo s5,1;
    MoveL offs(BaseCaja,0,0,150),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL PosIni1,v500,z100,TCPVentosa\WObj:=wobj0;
    MoveL offs(Target_50,0,0,35),v500,z100,TCPVentosa\WObj:=wobj0;
    MoveJ offs(Target_150,0,0,230),v500,fine,TCPVentosa\WObj:=wobj0;
    MoveL Target_150,v100,z100,TCPVentosa\WObj:=wobj0;
    WaitRob\inpos;
    reset S6;
    MoveJ PosIni1,v1000,z100,TCPVentosa\WObj:=wobj0;
ENDPROC

PROC ResetSeñales()
```

```
        reset s1;
        reset s2;
        reset s3;
        reset s4;
        reset s5;
        reset s6;
        reset s7;
        reset s8;
        reset s9;
        reset s10;
        reset s11;
        Reset s12;
        reset CogerHerr;
        reset CogerVent;
        reset GenTor;
        Reset GenTor2;
        reset CogerGlue;
        reset CogerHerr2;
        Reset CogerVentG;
        Reset EliminarP;
        reset MoverOp;
        Reset Attach2;
        Reset SalirSist;
        Reset empezar;
        Reset Robot1ON;
        Reset Robot2ON;
        Reset Robot3ON;
        Reset Robot4ON;
    ENDPROC
ENDMODULE
```

10.2 Código desarrollado en Matlab

Script para la comunicación OPC 'ServOPC.m'

```
%Para obtener la informacion del Host
Host = opcserverinfo('localhost');%Informacion del Host
ServDisp = Host.ServerID'; %Servidores Disponibles

%Creacion y conexion del cliente
UsuarioM = opcda('localhost',
'ABB.IRC5.OPC.Server.DA'); %Creamos un cliente
connect(UsuarioM); %Conecta el cliente creado al
servidor OPC
Signals = addgroup(UsuarioM); %Grupo de señales que
serán modificadas

%Señales de Robotstudio
Empezar=additem(Signals,'LAPTOP-
AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Empezar');
Robot1On=additem(Signals,'LAPTOP-
AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Robot1ON');
Robot2On=additem(Signals,'LAPTOP-
AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Robot2ON');
Robot3On=additem(Signals,'LAPTOP-
AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Robot3ON');
Robot4On=additem(Signals,'LAPTOP-
AVIKUA4I_Controlador1.IOSYSTEM.IOSIGNALS.Robot4ON');

%Escribimos sobre la variable de Robotstudio
Estado=evalin('base','State');
write(Empezar,Estado);

%Leemos la estructura de la señal
R1 = read(Robot1On);
R2 = read(Robot2On);
R3 = read(Robot3On);
R4 = read(Robot4On);

%Una vez leída la señal, nos quedamos unicamente con el
valor de esta convertido a tipo double y lo guardamos
en una variable que será leída e interpretada en
Appdesigner
%Extraemos el valor de cada señal
Rob1=double(R1.Value);
Rob2=double(R2.Value);
Rob3=double(R3.Value);
Rob4=double(R4.Value);
```

Código de la interfaz

```
    methods (Access = public)
        function EstadoRobs(app)
            ServOPC;
            if Rob1==1
                app.IRB1600Lamp.Color = 'g';
                app.Label.Text= 'Activo';
            else
                app.IRB1600Lamp.Color = 'r';
                app.Label.Text= 'Inactivo';
            end
            if Rob2==1
                app.IRB1600_2Lamp.Color = 'g';
                app.Label_2.Text= 'Activo';
            else
                app.IRB1600_2Lamp.Color = 'r';
                app.Label_2.Text= 'Inactivo';
            end
            if Rob3==1
                app.IRB1600IDLamp.Color = 'g';
                app.Label_3.Text= 'Activo';
            else
                app.IRB1600IDLamp.Color = 'r';
                app.Label_3.Text= 'Inactivo';
            end
            if Rob4==1
                app.IRB369C1Lamp.Color = 'g';
                app.Label_4.Text= 'Activo';
else
app.IRB369C1Lamp.Color = 'r';
app.Label_4.Text= 'Inactivo';
end
end
end

% Callbacks that handle component events
methods (Access = private)

% Code that executes after component creation
function startupFcn(app)
assignin('base','State',4);
app.Lamp.Color='Red';
app.Switch.Enable='Off';
app.EJECUTARButton.Enable= 'Off';
app.Label.Text= 'Inactivo';
app.Label_2.Text= 'Inactivo';
app.Label_3.Text= 'Inactivo';
app.Label_4.Text= 'Inactivo';
end
```

```
% Value changed function: Switch
function SwitchValueChanged(app, event)
value = app.Switch.Value;
if strcmp(value, 'Marcha')
assignin('base','State',1)
app.LedLamp.Color = 'g';
elseif strcmp(value, 'Paro')
assignin('base','State',0)
app.LedLamp.Color = 'r';
end
ServOPC;
end

% Button pushed function: EJECUTARButton
function EJECUTARButtonPushed(app, event)
EstadoRobs(app);
end

% Value changed function: ACTIVARINTERFACESwitch
function ACTIVARINTERFACESwitchValueChanged(app, event)
value = app.ACTIVARINTERFACESwitch.Value;
switch value
case 'Off'
app.Lamp.Color='Red';
app.Switch.Enable='Off';
app.EJECUTARButton.Enable= 'Off';
case 'On'
app.Lamp.Color='Green';
app.Switch.Enable='On';
app.EJECUTARButton.Enable= 'On';
end
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 0];
app.UIFigure.Position = [100 100 639 425];
app.UIFigure.Name = 'MATLAB App';

% Create IRB1600LampLabel
app.IRB1600LampLabel = uilabel(app.UIFigure);
app.IRB1600LampLabel.HorizontalAlignment = 'right';
```

```
app.IRB1600LampLabel.Position = [54 278 52 22];
app.IRB1600LampLabel.Text = 'IRB1600';

% Create IRB1600Lamp
app.IRB1600Lamp = uilamp(app.UIFigure);
app.IRB1600Lamp.Position = [69 257 20 20];
app.IRB1600Lamp.Color = [1 0 0];

% Create IRB1600_2LampLabel
app.IRB1600_2LampLabel = uilabel(app.UIFigure);
app.IRB1600_2LampLabel.HorizontalAlignment = 'right';
app.IRB1600_2LampLabel.Position = [197 278 66 22];
app.IRB1600_2LampLabel.Text = 'IRB1600_2';

% Create IRB1600_2Lamp
app.IRB1600_2Lamp = uilamp(app.UIFigure);
app.IRB1600_2Lamp.Position = [220 257 20 20];
app.IRB1600_2Lamp.Color = [1 0 0];

% Create IRB1600IDLampLabel
app.IRB1600IDLampLabel = uilabel(app.UIFigure);
app.IRB1600IDLampLabel.HorizontalAlignment = 'right';
app.IRB1600IDLampLabel.Position = [375 278 64 22];
app.IRB1600IDLampLabel.Text = 'IRB1600ID';

% Create IRB1600IDLamp
app.IRB1600IDLamp = uilamp(app.UIFigure);
app.IRB1600IDLamp.Position = [398 257 20 20];
app.IRB1600IDLamp.Color = [1 0 0];

% Create IRB369C1LampLabel
app.IRB369C1LampLabel = uilabel(app.UIFigure);
app.IRB369C1LampLabel.HorizontalAlignment = 'right';
app.IRB369C1LampLabel.Position = [534 278 61 22];
app.IRB369C1LampLabel.Text = 'IRB369C1';

% Create IRB369C1Lamp
app.IRB369C1Lamp = uilamp(app.UIFigure);
app.IRB369C1Lamp.Position = [556 257 20 20];
app.IRB369C1Lamp.Color = [1 0 0];

% Create Switch
app.Switch = uiswitch(app.UIFigure, 'slider');
app.Switch.Items = {'Paro', 'Marcha'};
app.Switch.ValueChangedFcn = createCallbackFcn(app,
@SwitchValueChanged, true);
app.Switch.Position = [411 22 70 31];
app.Switch.Value = 'Paro';

% Create LedLamp
```

```
app.LedLamp = uilamp(app.UIFigure);
app.LedLamp.Position = [436 62 26 26];
app.LedLamp.Color = [1 0 0];

% Create Image
app.Image = uiimage(app.UIFigure);
app.Image.Position = [50 313 100 100];
app.Image.ImageSource = 'EII.png';

% Create Image2
app.Image2 = uiimage(app.UIFigure);
app.Image2.Position = [502 313 100 100];
app.Image2.ImageSource = 'UVa.png';

% Create EJECUTARButton
app.EJECUTARButton = uibutton(app.UIFigure, 'push');
app.EJECUTARButton.ButtonPushedFcn = createCallbackFcn(app,
@EJECUTARButtonPushed, true);
app.EJECUTARButton.BackgroundColor = [0.9294 0.6941 0.1255];
app.EJECUTARButton.Position = [171 22 114 31];
app.EJECUTARButton.Text = 'EJECUTAR';

% Create PresioneparacomprobarestadodeLabel
app.PresioneparacomprobarestadodeLabel = uilabel(app.UIFigure);
app.PresioneparacomprobarestadodeLabel.Position = [136 77 201 22];
app.PresioneparacomprobarestadodeLabel.Text = 'Presione para
comprobar estado de ';

% Create funcionamientodelosrobotsLabel
app.funcionamientodelosrobotsLabel = uilabel(app.UIFigure);
app.funcionamientodelosrobotsLabel.Position = [148 61 160 22];
app.funcionamientodelosrobotsLabel.Text = 'funcionamiento de los
robots';

% Create Image6
app.Image6 = uiimage(app.UIFigure);
app.Image6.Position = [515 147 101 100];
app.Image6.ImageSource = 'IRB369C1.png';

% Create Image5
app.Image5 = uiimage(app.UIFigure);
app.Image5.Position = [359 147 100 100];
app.Image5.ImageSource = 'IRB1600ID.png';

% Create Image4
app.Image4 = uiimage(app.UIFigure);
app.Image4.Position = [180 147 100 100];
app.Image4.ImageSource = 'IRB1600.png';

% Create Image3
```

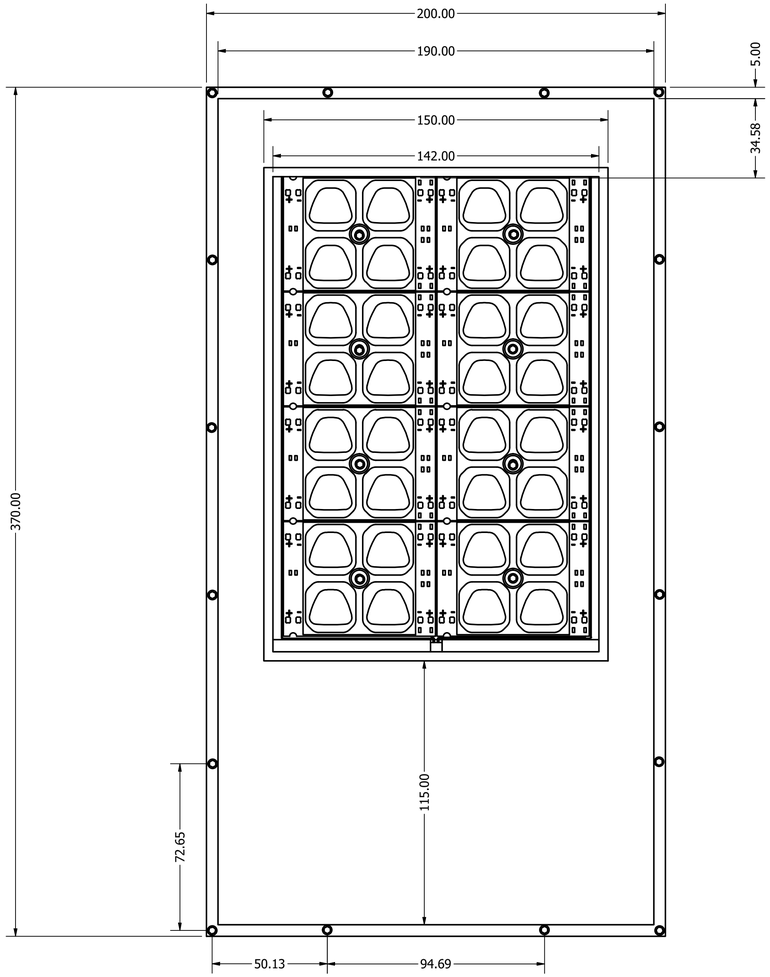
```
app.Image3 = uiimage(app.UIFigure);
app.Image3.Position = [27 147 100 100];
app.Image3.ImageSource = 'IRB1600.png';

% Create ACTIVARINTERFACESwitchLabel
app.ACTIVARINTERFACESwitchLabel = uilabel(app.UIFigure);
app.ACTIVARINTERFACESwitchLabel.HorizontalAlignment = 'center';
app.ACTIVARINTERFACESwitchLabel.Position = [256 391 127 22];
app.ACTIVARINTERFACESwitchLabel.Text = 'ACTIVAR INTERFACE';

% Create ACTIVARINTERFACESwitch
app.ACTIVARINTERFACESwitch = uiswitch(app.UIFigure, 'slider');
app.ACTIVARINTERFACESwitch.ValueChangedFcn = createCallbackFcn(app,
@ACTIVARINTERFACESwitchValueChanged, true);
app.ACTIVARINTERFACESwitch.Position = [278 356 81 36];

% Create Lamp
app.Lamp = uilamp(app.UIFigure);
app.Lamp.Position = [399 360 29 29];
app.Lamp.Color = [1 0 0];

% Create Label
app.Label = uilabel(app.UIFigure);
app.Label.BackgroundColor = [1 1 1];
app.Label.HorizontalAlignment = 'center';
app.Label.Position = [39 118 76 22];
app.Label.Text = '';

% Create Label_2
app.Label_2 = uilabel(app.UIFigure);
app.Label_2.BackgroundColor = [1 1 1];
app.Label_2.HorizontalAlignment = 'center';
app.Label_2.Position = [192 118 76 22];
app.Label_2.Text = '';

% Create Label_3
app.Label_3 = uilabel(app.UIFigure);
app.Label_3.BackgroundColor = [1 1 1];
app.Label_3.HorizontalAlignment = 'center';
app.Label_3.Position = [372 118 76 22];
app.Label_3.Text = '';

% Create Label_4
app.Label_4 = uilabel(app.UIFigure);
app.Label_4.BackgroundColor = [1 1 1];
app.Label_4.HorizontalAlignment = 'center';
app.Label_4.Position = [528 118 76 22];
app.Label_4.Text = '';

% 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 = InterfazM

% 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
```

10.3 Planos

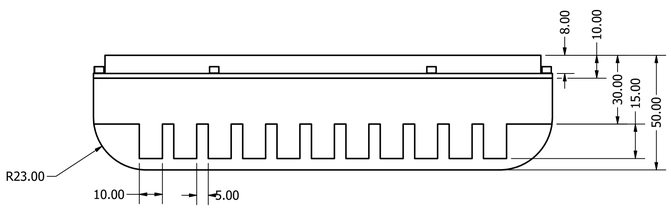
A continuación, se muestran los planos de los elementos más representativos que componen la célula de trabajo.

Índice de planos

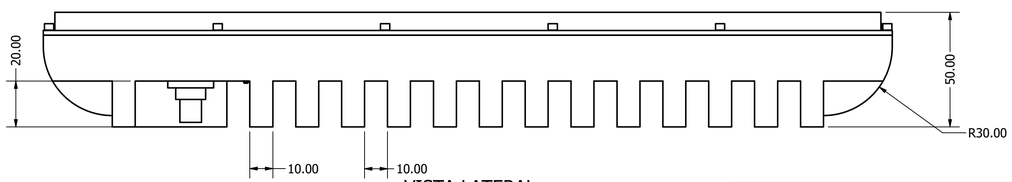


VISTA SUPERIOR

VISTA ISOMETRICA

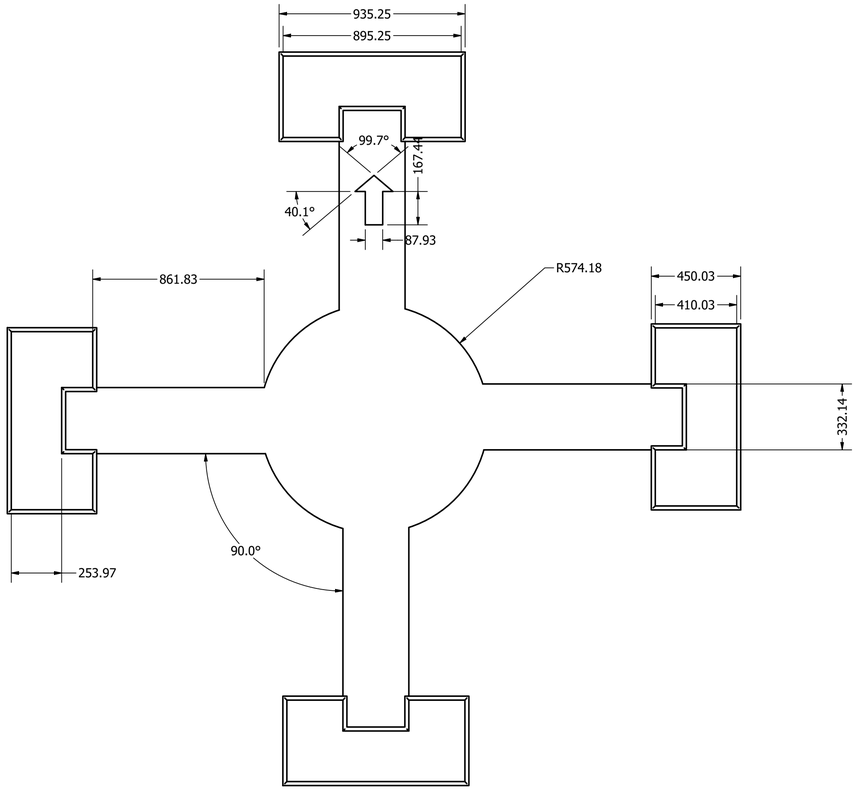


VISTA FRONTAL

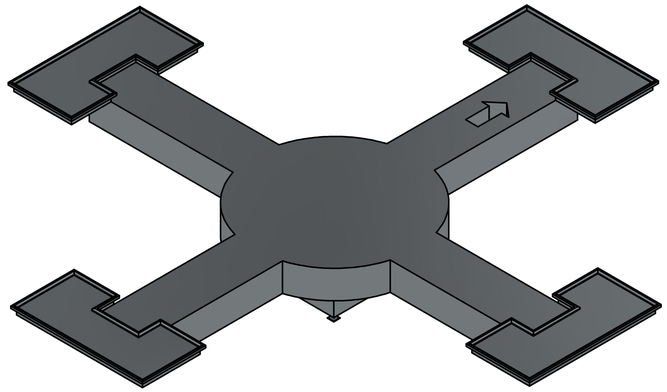


VISTA LATERAL

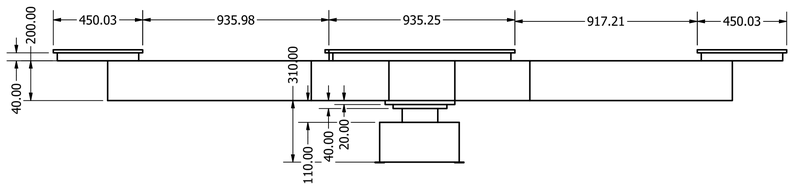
DRAWN Raúl Vidal	20/06/2022		
CHECKED		TITLE	
QA			
MFG			
APPROVED		SIZE D	DWG NO Luminaria / REV
		SCALE 1 : 1.3	SHEET 1 OF 1



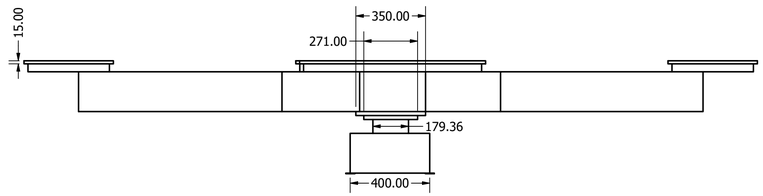
VISTA SUPERIOR



VISTA ISOMÉTRICA

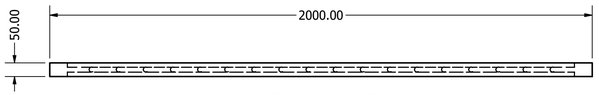


VISTA FRONTAL

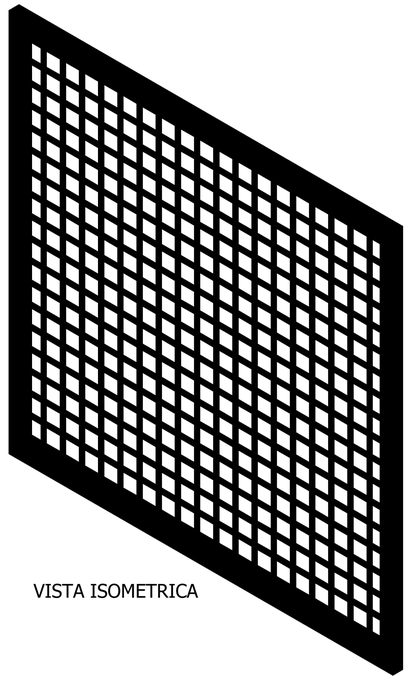


VISTA LATERAL

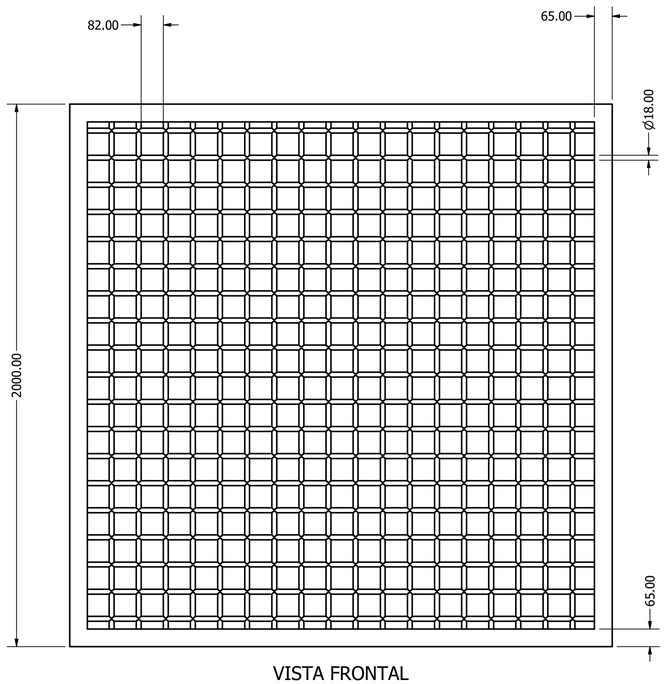
DRAWN Raúl Vidal	30/05/2022	
CHECKED		TITLE: Plataforma giratoria
QA		
MFG		
APPROVED		SIZE D · DWG NO PlanoMesaGir · REV
		SCALE 1 / 15 · SHEET 1 OF 1



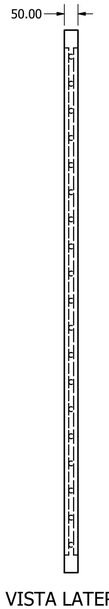
VISTA SUPERIOR



VISTA ISOMETRICA

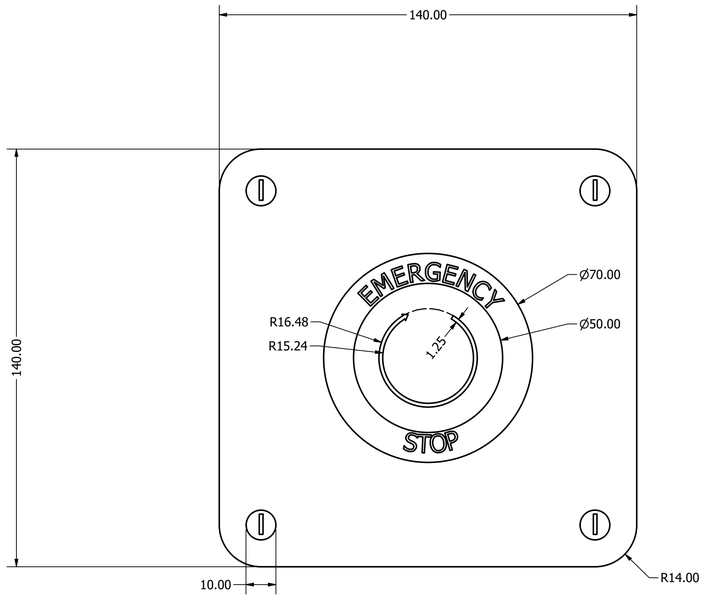


VISTA FRONTAL

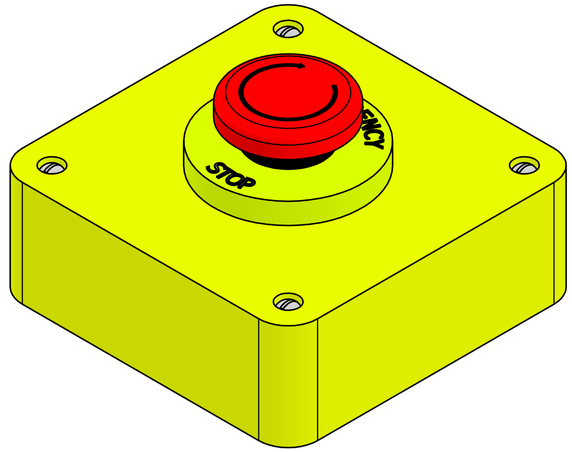


VISTA LATERAL

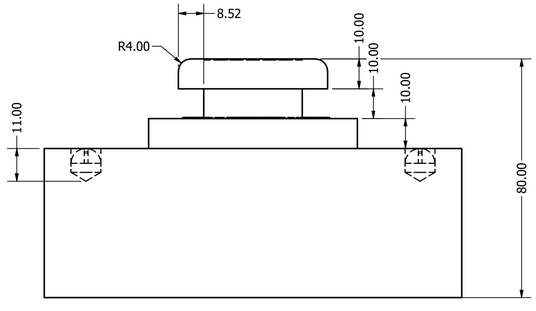
DRAWN Raúl Vidal	15/06/2022		
CHECKED			
QA		TITLE	Diseño de valla
MFG			
APPROVED		SIZE D	DWG NO BarreraProtectora
		SCALE 1 / 11	SHEET 1 OF 1



VISTA SUPERIOR

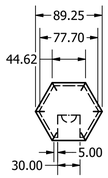


VISTA ISOMÉTRICA

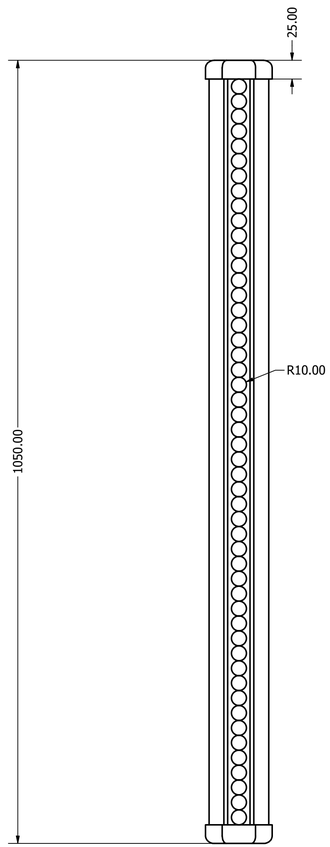


VISTA FRONTAL

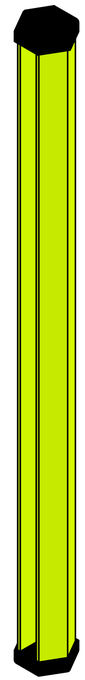
DRAWN Raúl Vidal	15/06/2022		
CHECKED			
QA		TITLE	Seta de emergencia
MFG			
APPROVED		SIZE D	DWG NO SetadeEmergencia
		SCALE 1 : 1	SHEET 1 OF 1



VISTA SUPERIOR

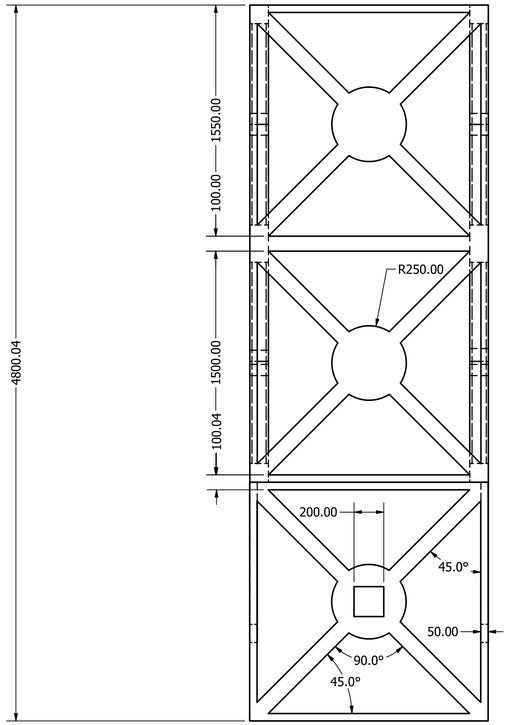


VISTA FRONTAL

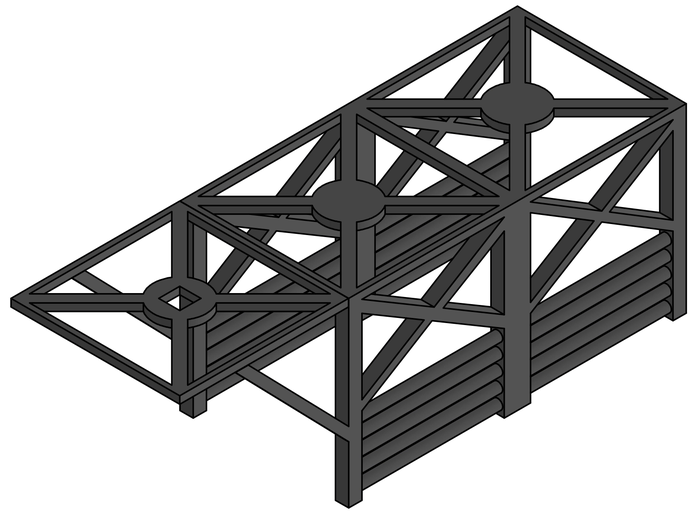


VISTA ISOMETRICA

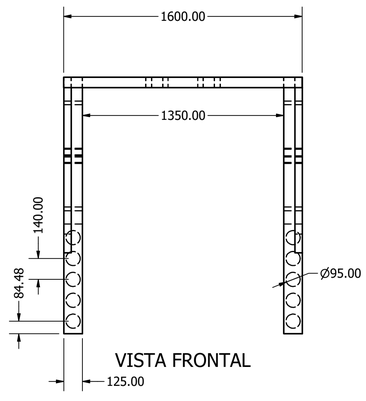
DRAWN Raúl Vidal	15/06/2022		
CHECKED		TITLE	Barrera Fotoeléctrica
QA			
MFG			
APPROVED		SIZE D	DWG NO Barrerafotoelectrica
		SCALE 1 : 4	SHEET 1 OF 1



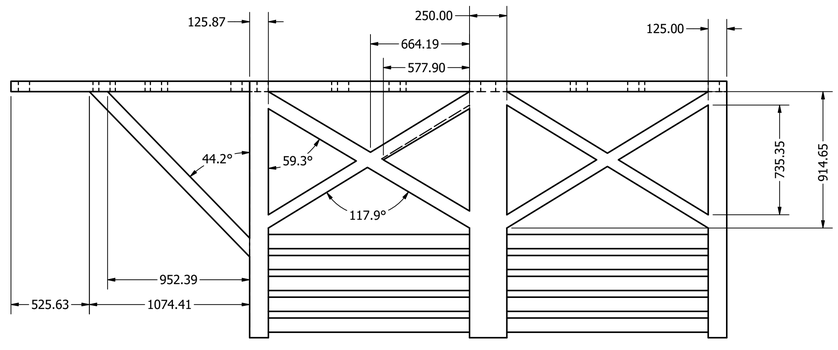
VISTA SUPERIOR



VISTA ISOMÉTRICA

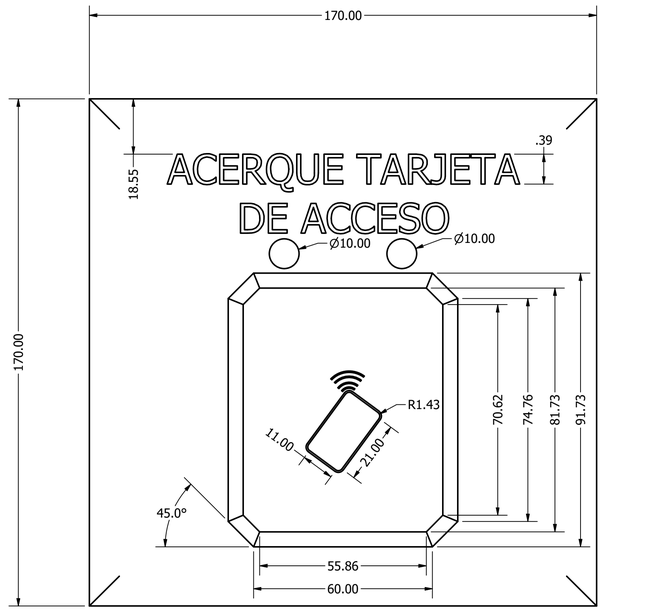


VISTA FRONTAL

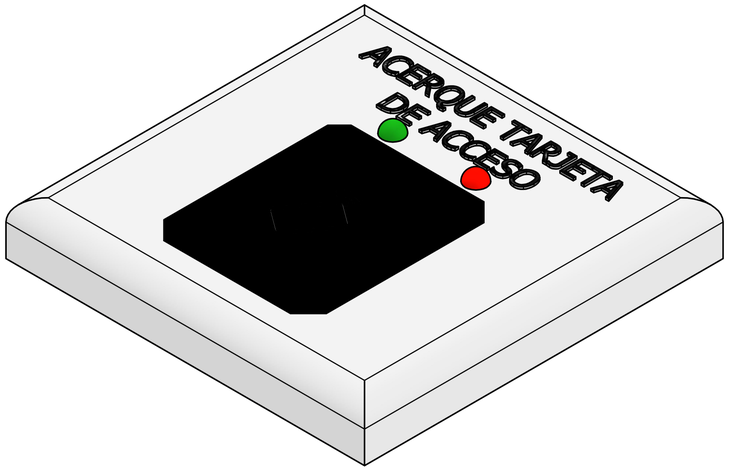


VISTA LATERAL

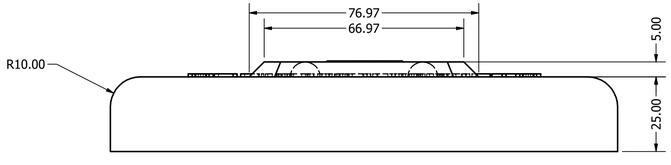
DRAWN Raúl Vidal	30/05/2022	
CHECKED		TITLE
QA		
MFG		
APPROVED		SIZE D / DWG NO Plataformaaraña / REV
		SCALE 1 / 20 / SHEET 1 OF 1



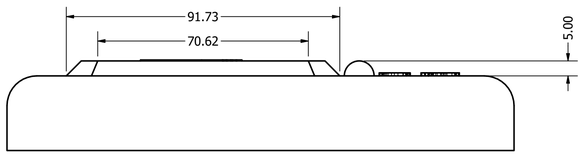
VISTA SUPERIOR



VISTA ISOMÉTRICA

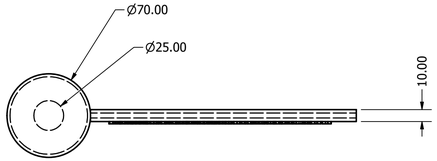


VISTA FRONTAL

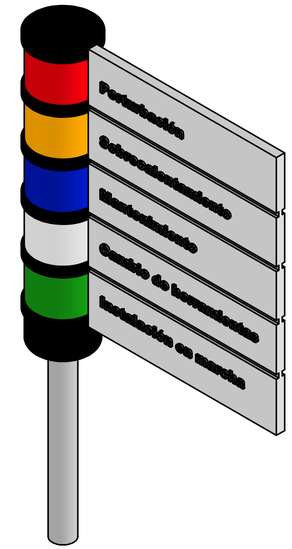


VISTA LATERAL

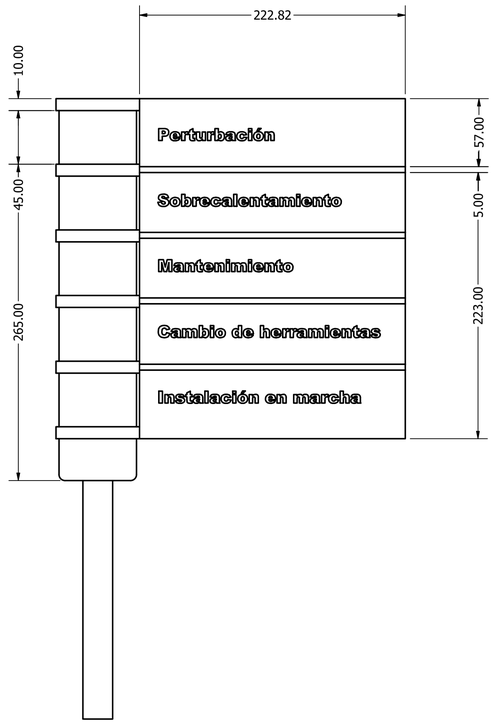
DRAWN Raúl Vidal	15/06/2022			
CHECKED				
QA		TITLE	Control de acceso	
MFG				
APPROVED		SIZE D	DWG NO Control de acceso	REV
		SCALE 1 : 1	SHEET 1 OF 1	



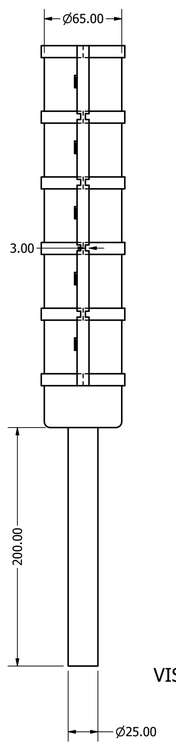
VISTA SUPERIOR



VISTA ISOMETRICA



VISTA FRONTAL



VISTA LATERAL

DRAWN Raúl Vidal	20/06/2022		
CHECKED			
QA			
MFG			
APPROVED			
		SIZE D	DWG NO ColumnaSeñalización
		SCALE 1 / 2.5	SHEET 1 OF 1

TITLE: Columna de señalización

REV