GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Aplicación de la realidad virtual al diseño y programación de procesos industriales:

Versión simulación de procesos

Autor: Manuel Trabado de la Cruz

Director: José Antonio Rodríguez Mondéjar

Madrid

03/07/2020

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Aplicación de la realidad virtual al diseño y programación de procesos industriales: Versión simulación de procesos

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 20019/2020 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Manuel Trabado de la Cruz Fecha: 03 / 07 / 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: José Antonio Rodríguez Mondéjar Fecha: ……/ ……/ ……

Firmado por RODRIGUEZ MONDEJAR JOSE ANTONIO - 29057313X el día 03/07/2020 con un certificado emitido por AC FNMT Usuarios

APLICACIÓN DE LA REALIDAD VIRTUAL AL DISEÑO Y PROGRAMACIÓN DE PROCESOS INDUSTRIALES: VERSIÓN SIMULACIÓN DE PROCESOS

Autor: Trabado de la Cruz, Manuel
Director: Rodríguez Mondéjar, José Antonio.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto consiste en el desarrollo de un entorno que permita a un operador entrenarse de manera virtual en la operación de un proceso industrial, todo esto sin la necesidad de estar en su puesto de trabajo real.

Para llevar a cabo el proyecto se va a crear un modelo 3D con SolidEdge de una planta industrial, dicho modelo 3D se implementa en Unity para crear el entorno de la simulación. La simulación en Unity es controlada por un PLC virtual mediante PLCsim.

Para permitir una completa interacción entre la simulación y el PLC virtual se ha creado un servidor virtual, con NetToPLCsim, al cual se conecta el PLC virtual y permite crear un canal de comunicación entre la simulación en Unity y el PLC.

En este proyecto se ha diseñado un entorno que consiste en una planta industrial que consta de cuatro cintas transportadoras, tres de ellas se encargan de colocar un tipo de objeto distinto sobre la misma cinta, la cinta principal. También se han añadido tres brazos robóticos que se encargan de retirar los objetos de la cinta transportadora. Todo este proceso es monitorizado mediante sensores situados a lo largo de la cinta transportadora principal que permiten reconocer el tipo de objeto que tiene que coger cada brazo.

La interacción del usuario con el entorno consiste en un panel de control que permite controlar la planta, todos los cambios realizados en el panel de control se envían al PLC y este actúa en función de las ordenes enviadas por el usuario y los cambios que se producen son enviados al panel de control y de ahí transmitidos al resto de elementos de la simulación.

Mediante el panel de control se puede encender y apagar cintas, cambiar de control manual a control automático, cambiar los tiempos con los que los objetos son puestos en las cintas y reiniciar el sistema en caso de que se haya detectado un error y se paren las cintas.

IMPLEMENTATION OF VIRTUAL REALITY IN THE DESIGN AND PROGRAMMING OF INDUSTRIAL PROCESSES: PROCESSES SIMULATION VERSION

Author: Trabado de la Cruz, Manuel.
Supervisor: Rodríguez Mondéjar, José Antonio.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project consists in the development of an environment that allows a worker to be virtually trained in the operation of an industrial process, this system allows the worker to train without being in the real workplace.

To complete this project a 3D model of the industrial plant is going to be made, and then implemented in Unity to create the environment of the simulation. The simulation is controlled by a virtual PLC through PLCsim.

To allow the interaction between the simulation and the virtual PLC a virtual server has been created through NetToPLCsim, which allows to create a link between the simulation in Unity and the PLC.

In this project the environment is composed of four belt conveyors, three of them take charge of putting the different objects on the main belt. There are three robotic arms that take the products from the main belt. This process is controlled by sensors placed throughout the main belt that allow each robotic arm which object it must take.

The interaction between the user and the environment consist in a control desk that allows the user to control the industrial plant, all the changes made in the control desk are sent to the PLC and it executes the orders from the user and send the changes made in the PLC back to the control desk and then transmitted to the different elements of the simulation.

Through the control desk the user can turn on and off the conveyor belts, switch from the manual control to automatic and vice versa, change the delivery time of each product and restart the process if there is an error that stops the belts.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Este proyecto consiste en la elaboración de una aplicación basada en el uso de la realidad virtual para simular el entorno de trabajo en una planta industrial. Se pretende dotar al usuario de la mayor libertad posible y de la capacidad de interactuar con los elementos de su entorno para poder llevar a cabo una simulación de su trabajo en un entorno real.

Este proyecto está orientado en el entrenamiento del personal de un entorno industrial en el cual sea necesario el uso de un PLC sin la necesidad de estar en el puesto de trabajo real. Mediante el uso de un modelo 3D y el uso de la realidad virtual se puede simular el entorno de trabajo y así entrenar al personal sobre cómo funciona y comprobar el correcto funcionamiento del PLC.

En los últimos años la industria relacionada con el desarrollo de tecnología de realidad virtual ha experimentado un gran crecimiento gracias al sector de los videojuegos, que ha visto en esta tecnología una nueva forma de desarrollar videojuegos. Los sistemas de realidad virtual más usados son: HTC Vive [1] , Oculus Rift [2] y PlayStation VR [3]. Todos estos sistemas se utilizan también en la industria para el desarrollo de aplicaciones relacionadas con la realidad virtual para la simulación de distintos tipos de entornos, desde el entrenamiento de pilotos hasta el adiestramiento de soldados y policías. La realidad virtual posé una gran versatilidad que permite ser utilizada en un gran rango de aplicaciones.

1.1 OBJETIVO

El objetivo principal de este proyecto es la creación de un entorno de simulación en el cual, mediante uso de la realidad virtual se pueda simular que el usuario se encuentra en una planta industrial y tenga la capacidad de interactuar con los elementos de su entorno.

La aplicación desarrollada debe resultar intuitiva para quien lo usa debido a la gran cantidad de parámetros que se pueden controlar en ella.

1.2 FUNCIONAMIENTO

El funcionamiento del proyecto se basa en la comunicación entre la simulación en Unity y la simulación de un PLC virtual en PLC Sim. Esta comunicación se realiza a través de NeToPLCsim [4] mediante el protocolo de comunicación S7, el cual permite acceder a los datos de la simulación del PLC a través de la IP de este y así poder leer y cambiar los valores de las distintas variables del PLC.

El funcionamiento se basa en la capacidad de poder acceder a las variables del PLC en un determinado DataBlock desde Unity y así poder actuar según dicten las variables del PLC.

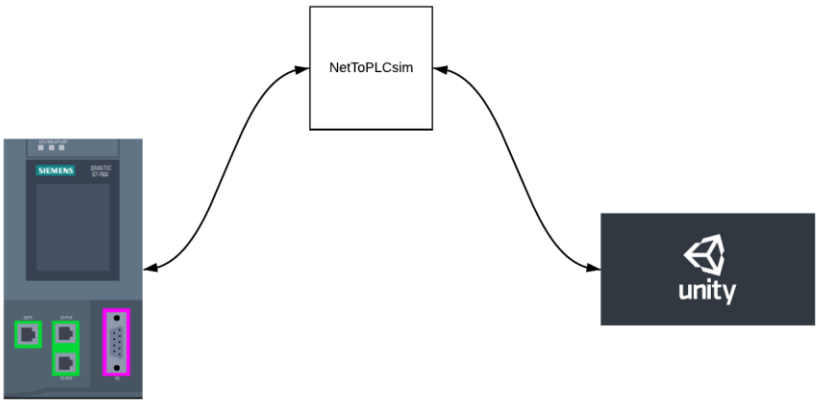


Figura 1 Esquema de conexión

Durante la ejecución de la simulación todas las instrucciones son ejecutadas desde Unity, la función del PLC es la de proporcionar a Unity la información del estado de las variables a controlar. En función del estado de las variables de control Unity ejecutará unas instrucciones u otras.

En el ANEXO IV: Variables De Control se adjunta una tabla con las variables que se van a controlar, su posición en el DataBlock y una descripción de para qué sirven.

Durante todo el proceso de la simulación Unity estará leyendo los valores del PLC ya actuará en función del estado de estas variables, las acciones que se lleven a cabo provocarán que

determinadas variables de control tengan que cambiar de estado, por ello es importante que cada vez que se produce un cambio Unity se lo comunica al PLC cambiando el estado de sus variables. Esto se resume en una comunicación de ida y vuelta que permite una mayor interacción entre Unity y el PLC, esta mayor interacción favorece la creación de sistemas más complejos.

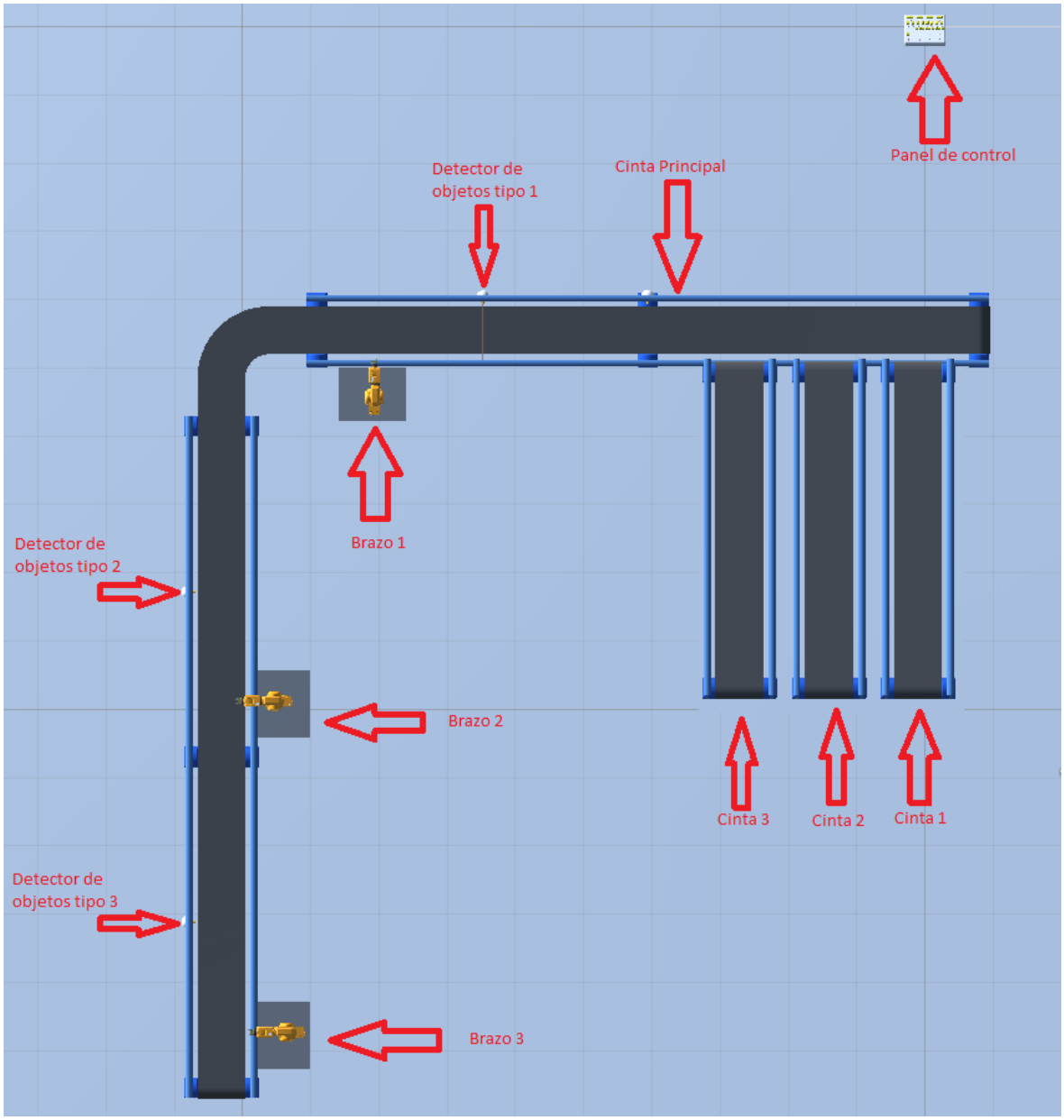


Figura 2 Vista cenital de la planta

Como se muestra en la Figura 2 Vista cenital de la planta la planta consiste en 3 cintas transportadoras que se encargan de colocar sobre la cinta principal cada uno de los tres tipos de objeto distintos. En la simulación los objetos aparecen sobre las cintas y caen sobre ellas, una vez en las cintas se mueven a lo largo de ellas hasta llegar a la cinta principal.

A lo largo de la cinta principal se han colocado una serie de sensores para detectar el tipo de objeto que pasa, esto ayuda seguir el recorrido que llevan los objetos por la cinta, y un sensor que detecta si el objeto que pasa se considera defectuoso.

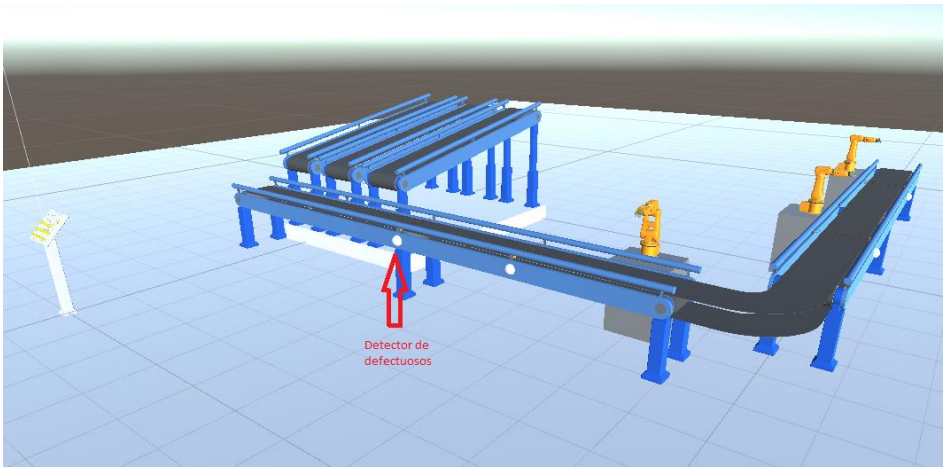


Figura 3 Vista general de la planta

Cuando se crea un objeto en la simulación se le define como defectuoso siguiendo el valor tomado por una variable pseudoaleatoria. El sensor de defectuosos permite al usuario ver si el ultimo objeto que ha pasado por el sensor es defectuoso o no.

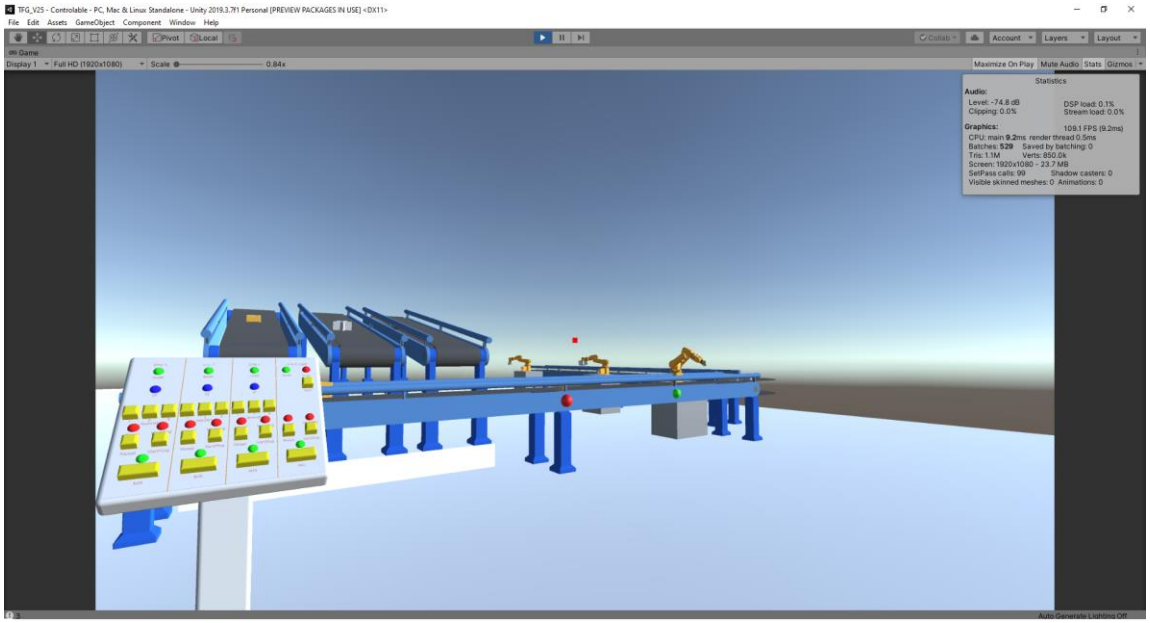


Figura 4 Simulación en Unity

La planta está diseñada de modo que desde la posición del panel de control se pueda ver el funcionamiento de toda planta, así se puede ver de forma rápida si se ha dejado de dispensar algún producto, si hay algún defectuoso o si alguno de los brazos se ha parado.

Mediante el uso del panel de control el operario puede controlar el funcionamiento de la planta. Además, también existe la posibilidad de que mientras el operario está usando la simulación otra persona puede cambiar parámetros de la simulación y así simular errores inesperados para que el operario sepa cómo actuar en caso de que algún error suceda.

Desde el panel de control el operario puede controlar:

- Tiempos de creación de los objetos.
- Encender y apagar cintas.
- Cambiar entre modo manual y automático de las cintas.
- Reiniciar la planta en caso de un error que haga que se pare.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

2.1 RECURSOS A UTILIZAR

2.1.1 UNITY

Unity es un motor gráfico y entorno de desarrollo orientado principalmente a la creación de videojuegos [5]. En este proyecto se va a utilizar Unity para crear el entorno en el que se ejecutará la simulación y se programará la interacción con el usuario.

Una de las principales ventajas de Unity frente a otros entornos de desarrollo es la existencia de una licencia gratuita que no tiene ninguna limitación funcional frente a las licencias de pago y además Unity permite instalar paquetes externos para facilitar el desarrollo de aplicaciones.

2.1.2 TIA PORTAL

Total Integrated Automation Portal, TIA Portal, por sus siglas en inglés es un software desarrollado por Siemens para la programación de PLC's. En este proyecto se usará para programar el PLC que controlará el comportamiento de toda la planta diseñada.

El PLC controlará las cintas transportadoras, los brazos robóticos, los tiempos en los que se pone un producto en la cinta y si los equipos están encendidos o apagados.

Todo esto será controlado a través de un panel de control, accesible desde la simulación, el cual permitirá al usuario cambiar los estados de las variables a controlar del PLC.

2.1.3 SOLID EDGE

En la planificación inicial del proyecto se pretendía usar NX [6], pero debido a la cuarentena por el COVID-19 fue imposible conseguir una licencia para el uso de NX. Por lo tanto, se

ha utilizado en su lugar Solid Edge [7]. Al igual que NX Solid Edge es un programa de CAD para el diseño de piezas 3D en el entorno de la ingeniería.

En Solid Edge se han diseñado las cintas transportadoras, el panel de control y los modelos de los objetos que se van a colocar en las cintas transportadoras.

2.1.4 AUTODESK FUSION 360

Este software de diseño 3D [8] ha sido necesario únicamente para poder exportar los archivos desde SolidEdge a un tipo de archivo que Unity pueda utilizar. Debido a que Unity necesita archivos OBJ para sus modelos y Solid Edge no tiene la opción de exportar a este tipo de archivo, el proceso de exportado de los modelos 3D hasta Unity ha sido el siguiente:

- Exportar en Solid Edge los modelos al formato STEP.
- Abrir los archivos STEP en Fusion 360.
- Exportar los archivos desde Fusion 360 a OBJ.
- Importar los archivos OBJ en Unity.

2.1.5 NETTOPLCSIM

NetToPLCsim será el programa utilizado para actuar como interfaz entre el PLC virtual y Unity. Este software permite crear un servidor virtual, dentro del mismo ordenador que se está utilizando para realizar la simulación, al cual se conectan tanto Unity mediante el uso de scripts como TIA Portal. Esto permite crear una comunicación mediante dirección IP entre Unity y el PLC virtual.

2.1.5.1 Pasos para configurar NetToPLCsim

1. Una vez abierto el programa pulsar sobre el botón *add* que aparece en la ventana para añadir un nuevo servidor.

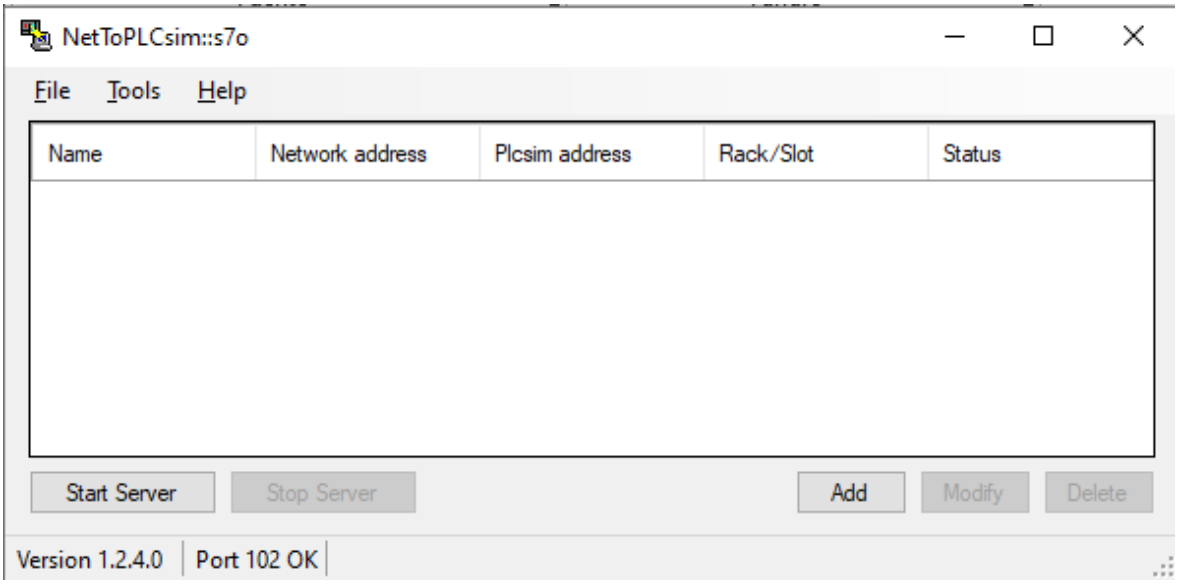


Figura 5 Menú principal de NetToPLCsim

2. Dentro de la con figuración para añadir un nuevo servidor se deben abrir los desplegables de las opciones *Network IP Adress* y *Plcsim IP Adress.* Una vez dentro del despegable se debe seleccionar la dirección IP correspondiente al PC que se va a usar para la simulación y la dirección IP de PLCsim respectivamente.

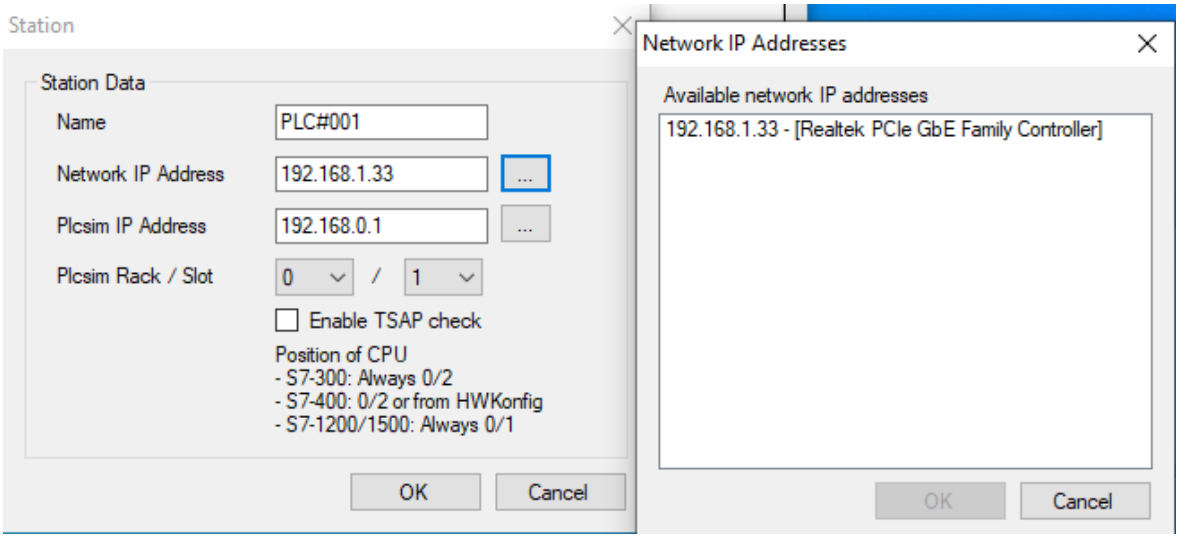


Figura 6 Selección de IP del PC de trabajo

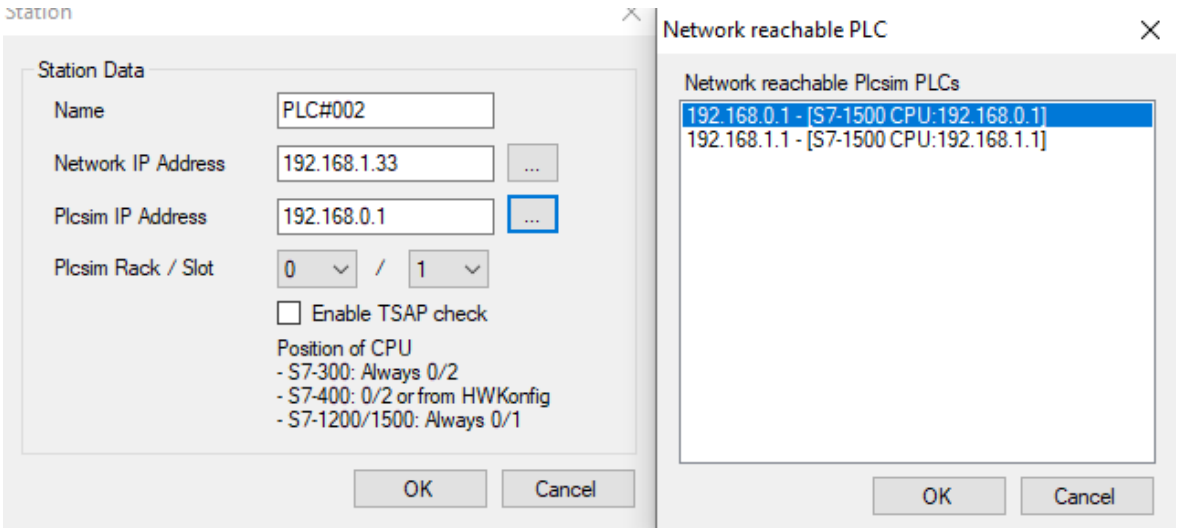


Figura 7 Selección de la IP de PLCsim

3. Seleccionar el rack y el slot correspondientes a los asignados en TIAPortal al PLC. En este proyecto los valores de rack y slot usados son 0 y 1 respectivamente.
4. Por último, cerrar la venta de configuración del nuevo servidor. Una vez en la ventana principal del programa seleccionar el nuevo servidor y pulsar el botón *Start Server*.

Capítulo 3. ESTADO DE LA CUESTIÓN

En el ámbito de la realidad virtual aplicada a la industria existen proyectos y estudios para distintos enfoques, sin embargo, en lo que concierne al tema de este proyecto no hay la misma variedad de estudios y proyectos que en otros campos de la industria. La realidad virtual ha tenido un gran desarrollo en los últimos años y podría posicionarse como una de las principales herramientas de desarrollo en el ámbito de la ingeniería gracias a la versatilidad que ofrece.

Entre los principales proyectos en este campo se destacan:

- Designing virtual reality approach for PLC applications development [9], llevado a cabo por Alexandru Avram, Liliana Samolia, Florin Samolia y Danait Afewerki. Este Proyecto pretende abordar una forma más inmersiva la enseñanza y el entorno de entrenamiento mediante el uso de la realidad virtual aplicada a la simulación de controladores lógicos programables (PLC). Este proyecto guarda gran similitud en el objetivo que se pretende conseguir.
- Virtual reality – An approach to improve the generation of fault free software for programmable logic controllers [10], llevado a cabo por D.Sapth y U.Osmers. En este proyecto se tiene el objetivo de simplificar la programación de PLC mediante el uso de una herramienta grafica interactiva mediante realidad virtual para tratar de reducir los problemas y fallos que se pueden cometer con el método de programación tradicional basado en un sistema booleano o basado en tiempos.

Capítulo 4. DESARROLLO EN TIA PORTAL

4.1 CONFIGURACIÓN

Para poder llevar a cabo una simulación que permita interacción con elementos exteriores al PLC es necesario realizar una configuración para poder realizar la simulación conjunta.

En este caso se ha utilizado un PLC S7-1516-13, la configuración necesaria podría cambiar para distintos modelos.

- En la ventana *Configuración de dispositivos* debemos hacer clic derecho sobre el PLC que se vaya a usar e ir a *Propiedades.*
- Una vez dentro de la opción de *Propiedades* acceder a la sección *Protección & Seguridad* y seleccionar la opción *Acceso Completo* dentro de la sección *Nivel de acceso* y después en *Mecanismos de conexión* seleccionar la opción *Permitir acceso vía comunicación PUT/GET del interlocutor remoto*.

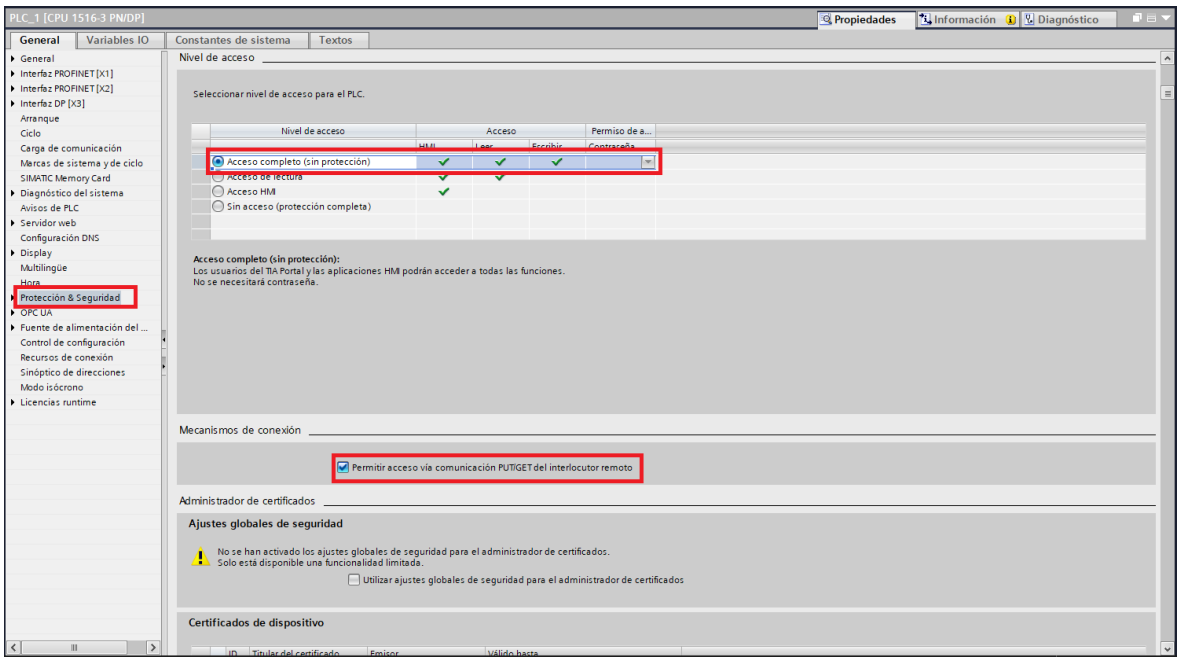


Figura 8 Configuración del nivel de acceso al PLC

- Dentro de la pestaña *Bloques de programa* hacer clic derecho sobre el/los Datablock al que se quiere acceder durante la simulación y acceder a *Propiedades* y en la pestaña de *Atributos* desmarcar la opción *Acceso optimizado al bloque.*

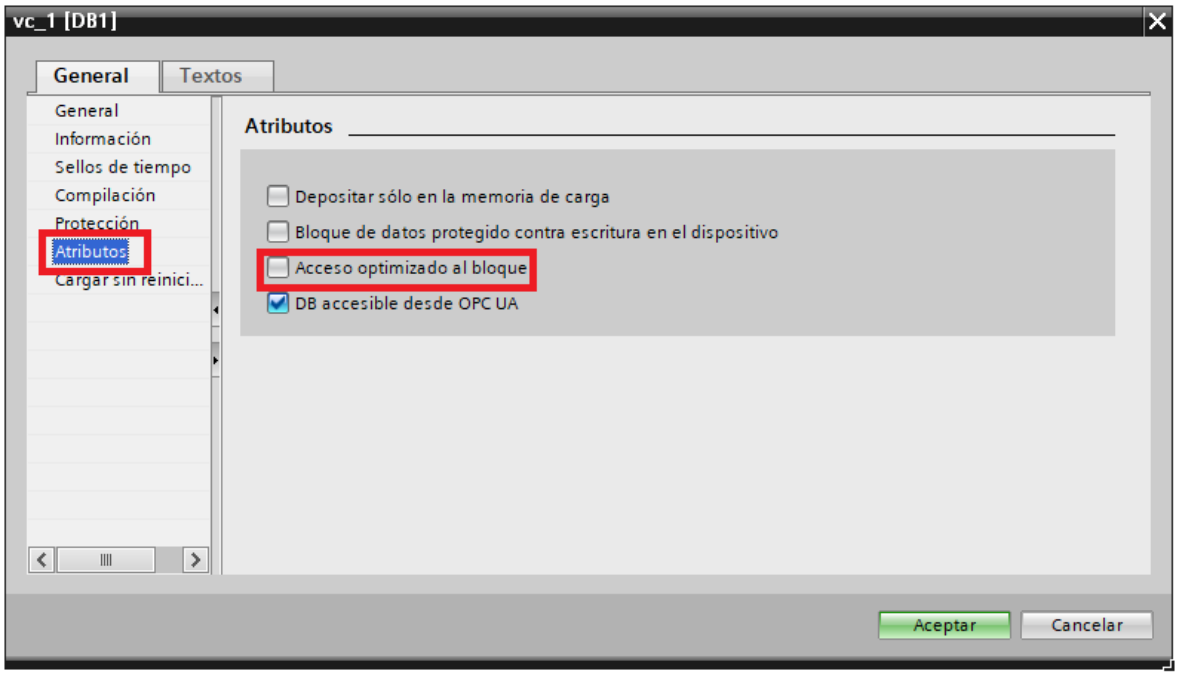


Figura 9 Configuración de acceso a un DataBlock

Con esta configuración el PLC ya debería de estar listo para permitir el acceso a sus datos. Es importante remarcar que en el caso en que haya determinados datos que no se quieran poder acceder desde el exterior, estos deberían se almacenados en un DataBlock distinto a los demás datos y no se le aplicara la configuración mostrada en Figura 9 Configuración de acceso a un DataBlock.

4.2 BLOQUES DE PROGRAMA

En esta sección se va a explicar el funcionamiento del programa creado para el PLC, para una mayor facilidad a la hora de ver el programa se han creado distintos Function Block con una función específica cada uno.

Para una mayor comprensión de la lógica seguida para el desarrollo del programa se recomienda leer ANEXO I: Diagramas Lógicos del PLC, en este anexo se muestran diagramas lógicos de los bloques mostrados en esta sección.

Para facilitar la elaboración del programa se ha decidido que todas las variables de control sean variables booleanas para que así sea más fácil representar el estado de una determinada variable.

4.3 CINTA PRINCIPAL

Este bloque se encarga de controlar el funcionamiento de la cinta principal, este bloque dice si la cinta está encendida o apagada.

Tiene como entradas:

- manual: esta señal indica si la cinta se controla en modo manual (*true*) o no (*false*). Si se encuentra en modo manual el operario puede decidir si enciende o apaga la cinta.
- start_stop: esta variable sirve para que el usuario pueda encender (*true*) o apagar (*false*) la cinta cuando está funcionando en modo manual. Si está en modo automático esta señal es ignorada independientemente del estado en el que se encuentre.
- error: esta señal representa si la planta se encuentra en un estado de error (*true*) o no (*false*). La señal procede de la salida de un biestable Flip-Flop, la señal se activa cuando se detecta un error y solo se desactiva cunado no hay un error y si se pulsa el reset del biestable, esto asegura que no se pueda volver a iniciar la planta sin que se cumplan con los requisitos para su correcto funcionamiento.
- arranque: se utiliza para poder volver a iniciar la planta después de haber detectado un estado de error. Durante el funcionamiento de la planta sin estado de error la señal permanece apagada (*false*), solo se enciende para volver a iniciar la planta.

- auto: indica si la cinta funcionará en modo automático (*true*) o no (*false*). Durante el funcionamiento en modo automático la cinta solo se para si se detecta un estado de error.

Salidas:

- on_off: esta señal indica si la cinta principal está en funcionamiento (true) o apagada (false).

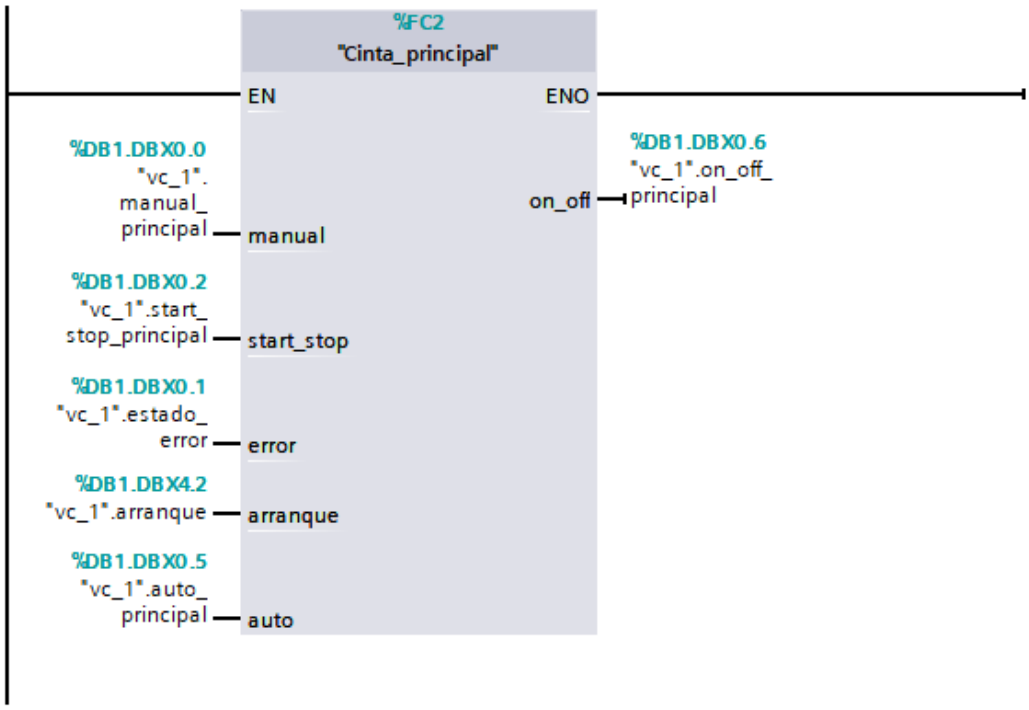


Figura 10 Bloque Cinta Principal

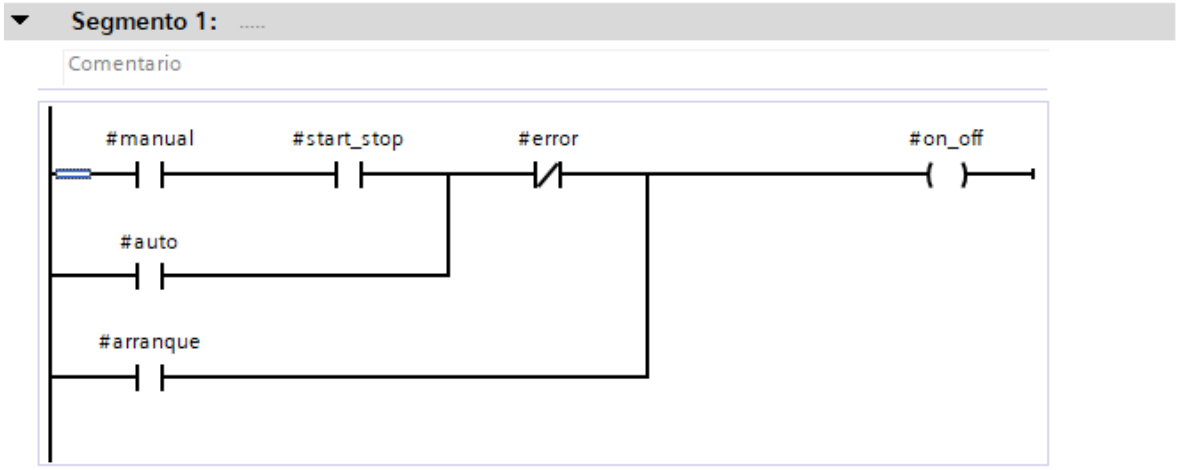


Figura 11 Lógica del bloque Cinta Principal

4.4 ERROR

Este bloque tiene la función de determinar cuándo se está en un estado de error. En este proyecto el estado de error no indica que haya habido un fallo durante la simulación o que algo no funcione, el estado de error indica que no se cumplen las condiciones necesarias establecidas para que siga en marcha la planta y esta se para, pero tanto la simulación en Unity como en PLCsim siguen funcionando, este estado se puede considerar como que la instalación que se está simulando está apagada.

Entradas:

- a_on_1: representa el estado en el que se encuentra el actuador número 1, *true* si está activo y *false* si está apagado.
- a_on_2: representa el estado en el que se encuentra el actuador número 2, *true* si está activo y *false* si está apagado.
- a_on_3: representa el estado en el que se encuentra el actuador número 3, *true* si está activo y *false* si está apagado.
- on_off_1: indica el estado en el que se encuentra la cinta número 1, *true* si esta encendida y *false* si está apagada.

- on_off_2: indica el estado en el que se encuentra la cinta número 2, *true* si esta encendida y *false* si está apagada.
- on_off_3: indica el estado en el que se encuentra la cinta número 3, *true* si esta encendida y *false* si está apagada.

Salida:

- error: si hay un error estará a *true* y si no hay error a *false*.

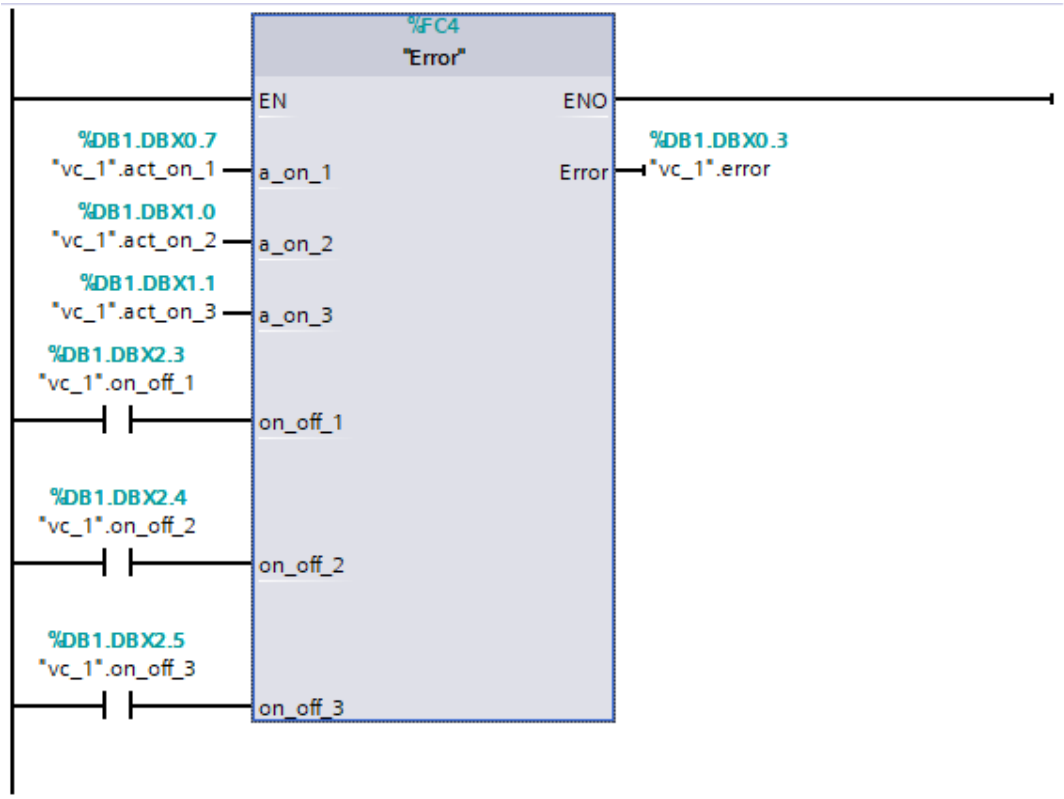


Figura 12 Bloque Error

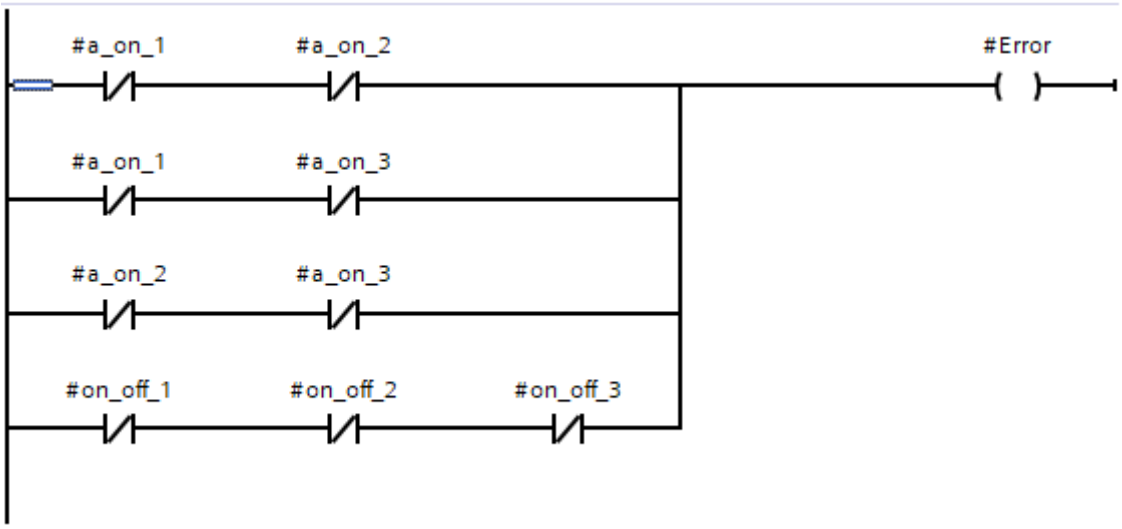


Figura 13 Lógica del bloque Error

4.5 CINTA DE DISTRIBUCIÓN

Este bloque se encargar de controlar las cintas en las que se colocan los diferentes objetos y son las encargadas de llevarlos hasta la cinta principal. El control de estas contas es muy similar al de la cinta principal, con la diferencia de que estas cintas no se paran por un estado de error, sino que se paran cada vez de que la cinta principal se para o el usuario hace que se paren en modo manual.

Entradas:

- on_off_principal: indica el estado de la cinta principal, encendida (*true*) o apagada (*false*).
- manual: *true* si se controla en modo manual y *false* si se controla en modo automático.
- start_stop: controla el encendido y apagado de la cinta en modo manual, *true* si se quiere encender y *false* para apagarla.
- auto: controla si la cinta se va a operar en modo automático (*true*) *o* manual (*false*).

Salidas:

- on_off: indica el estado de la cinta, *true* si esta encendida y *false* si está apagada.

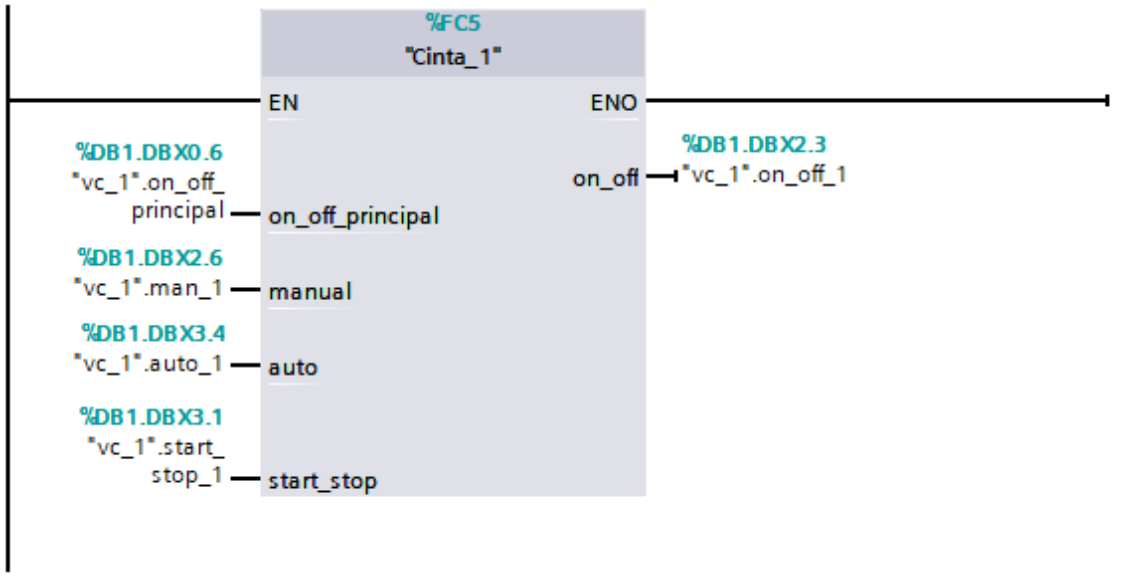


Figura 14 Bloque Cinta_1

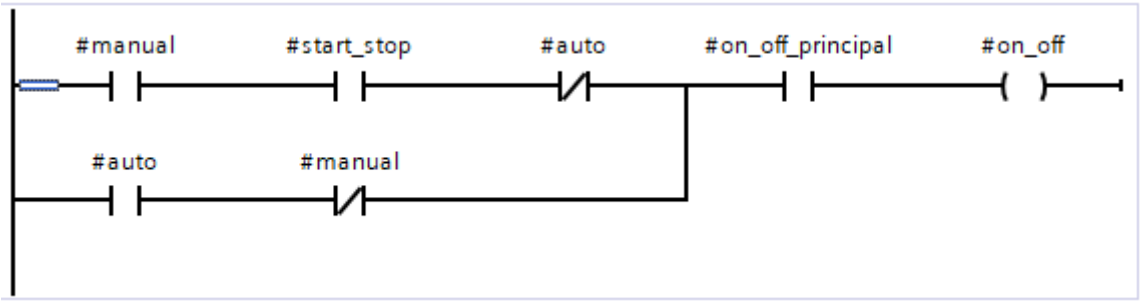


Figura 15 Lógica del bloque Cinta_1

Se repite la misma estructura para las cintas 2 y 3.

4.6 TEMPORIZADOR

Este bloque se encarga de controlar los intervalos de tiempo en los que se dejan los objetos en la cinta.

La cinta consta de tres velocidades, la velocidad se elige usando los distintos selectores, por defecto los selectores tienen valor *false.*

Cada temporizador tiene sus propios selectores:

- Temporizador 1:
  - sel1_1
  - sel2_1
- Temporizador 2:
  - sel1_2
  - sel2_2
- Temporizador 3:
  - sel1_3
  - sel2_3

A continuación, se muestra una tabla con las combinaciones de los selectores para cambiar los tiempos en cada temporizador.

sel1_1	sel2_1	Tiempo de cuenta (ms)
0	0	2500
1	0	4500
1	1	6500
sel1_2	sel2_2	Tiempo de cuenta (ms)
0	0	4500
1	0	6500
1	1	8500
sel1_3	sel2_3	Tiempo de cuenta (ms)
0	0	6500
1	0	8500
1	1	10500

Tabla 1 Tabla de tiempos de liberación de los temporizadores

Entradas:

- on_off: indica si la cinta principal está activa (*true*) o si está apagada (*false*). Si la cinta principal está apagada no se dispensan más objetos para evitar que se amontonen a la salida de la cinta de distribución.
- sel1: primer bit para la selección del tiempo de cuenta. Activo *true*, inactivo false.
- sel2: segundo bit para la selección del tiempo de cuenta. Activo *true*, inactivo false.
- entable: controla la activación del contador, se envía un pulso para iniciar la cuenta. Activo *true*, inactivo *false.*

Salida:

- libera: se activa para indicar la liberación de un nuevo objeto (*true*), se mantiene inactivo (*false*) mientras que no se cumplen las condiciones para la liberación de un producto nuevo.

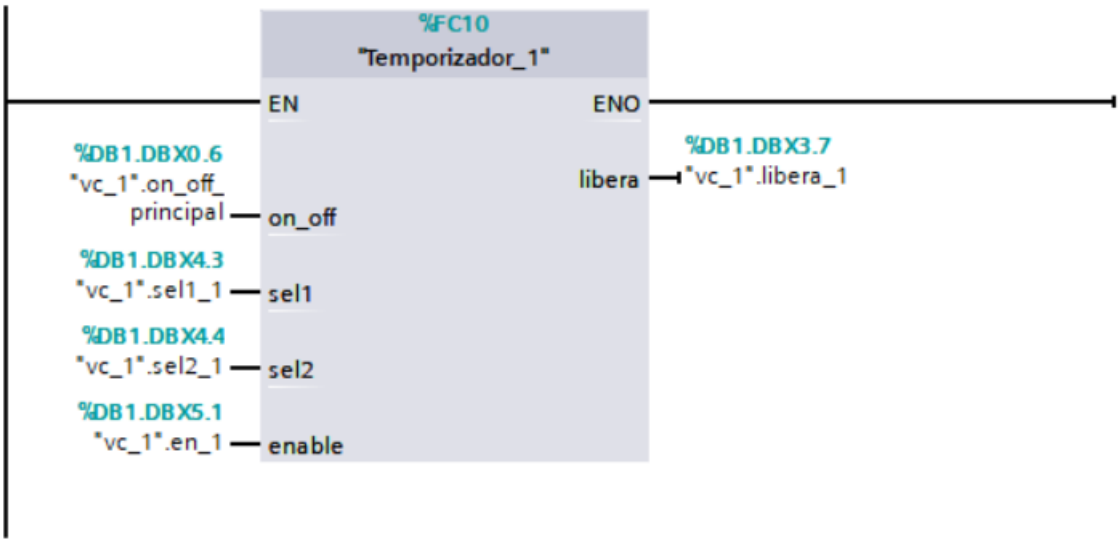


Figura 16 Bloque Temporizador_1

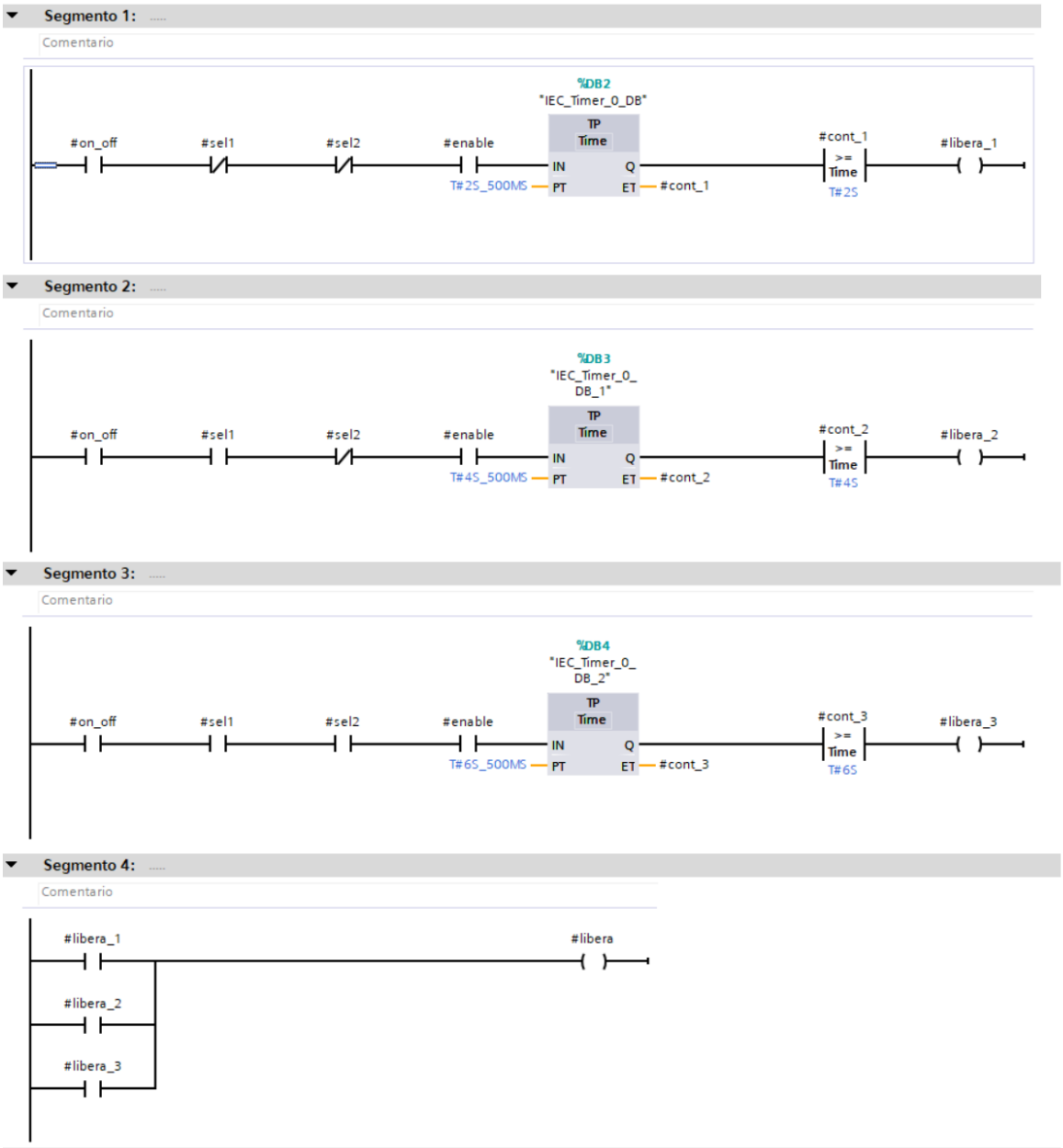


Figura 17 Lógica del bloque Temporizador_1

4.7 ACTUADOR

Este bloque se encarga de controlar cuando se tiene que activar un actuador para coger un objeto de la cinta. Para este bloque se han utilizado dos sensores para detectar los objetos,

aunque reamente solo necesitaría un sensor. Se ha decidido poner dos para dar más robustez al sistema y así en el caso en que uno falle el sistema seguiría funcionando.

El caso en que uno de los sensores falle no se da en una simulación porque no influyen factores como el paso del tiempo o los errores en la fabricación de componentes, pero con esto se pretendía dar una imagen más real de lo que puede ser un sistema como el usado en el proyecto para un caso real.

Los objetos solo se retiran de la cinta si el actuador está activo, la cinta principal esta activa y si al menos un sensor detecta el objeto que tiene que coger.

Entradas:

- ac_on: indica si el actuador está encendido (*true*) o si está apagado (*false*).
- sens1: esta señal indica si el sensor número 1 del actuador detecta un objeto (*true*) o no (*false*).
- sens2: esta señal indica si el sensor número 2 del actuador detecta un objeto (*true*) o no (*false*).
- on_off: indica si la cinta principal está activa (*true*) o si está apagada (*false*).

Salida:

- coge: indica si el actuador tiene que retirar un producto de la cinta (*true*) o no (*false*).

El PLC solo indica a los brazos robóticos si tienen que actuar o no, el control de los brazos robóticos forma parte de la programación en Unity.

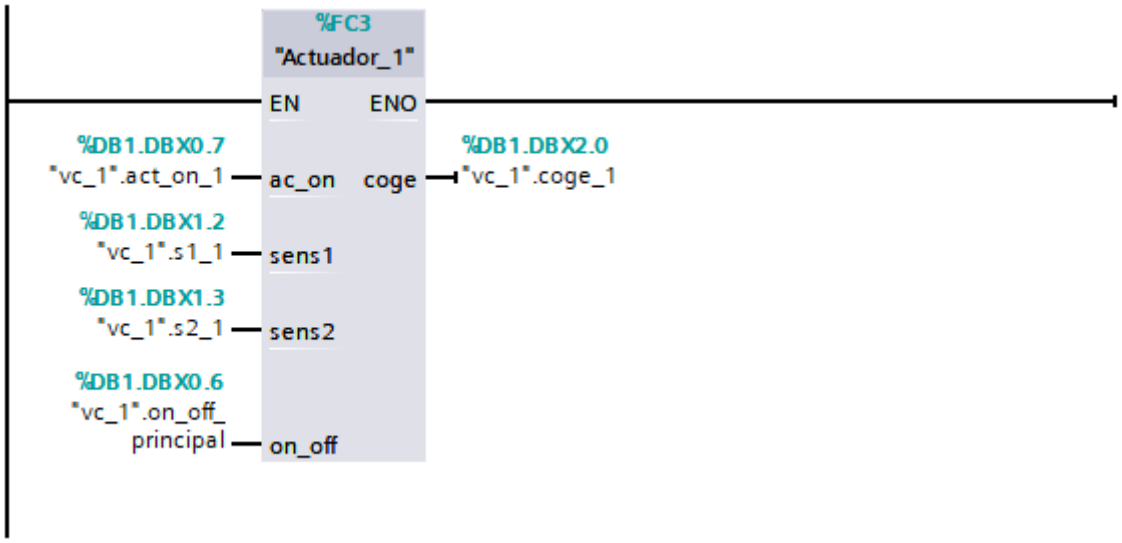


Figura 18 Bloque Actuador_1

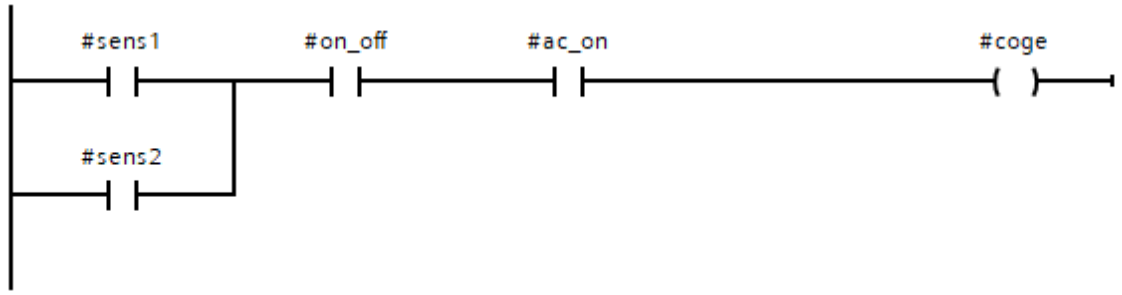


Figura 19 Lógica del bloque Actuador_1

4.8 SIMULACIÓN

Dentro del bloque *Main* se encuentran todos los *Function Block* mostrados en el apartado Bloques De Programa. En el caso de las cintas de distribución, los actuadores y los temporizadores se han repetido 3 veces para controlar las 3 cintas de distribución, los tres actuadores y los 3 brazos robóticos que se van a utilizar en el proyecto.

Los bloques se han repetido siguiendo siempre la misma estructura cambiando únicamente las variables que recibe cada bloque y las que salen para adaptarlas a cada elemento a

controlar. Las variables que se utilizan en la simulación se encuentran en ANEXO IV: Variables De Control.

Capítulo 5. DISEÑO 3D

En este capítulo se muestran los distintos conjuntos modelados en 3D usados en el proyecto.

No se van a adjuntar planos del diseño dado que es una simulación y los elementos diseñados no están planteados para ser utilizados más allá de en una simulación, excepto el brazo robótico y el sistema de agarre. El brazo robótico será el mismo que se utiliza en la minifábrica ICAI, se obtuvo del sitio web de ABB [11] y el sistema de agarre usado en el brazo está diseñado por SCHUNK [12].

5.1 PANEL DE CONTROL

El panel de control consta de una serie de botones para que el usuario pueda interactuar con el PLC y cambiar el valor de determinadas variables de control para poder controlar el funcionamiento de la planta. El panel cuenta también con unos indicadores para poder saber en todo momento el estado de los diferentes elementos del sistema.

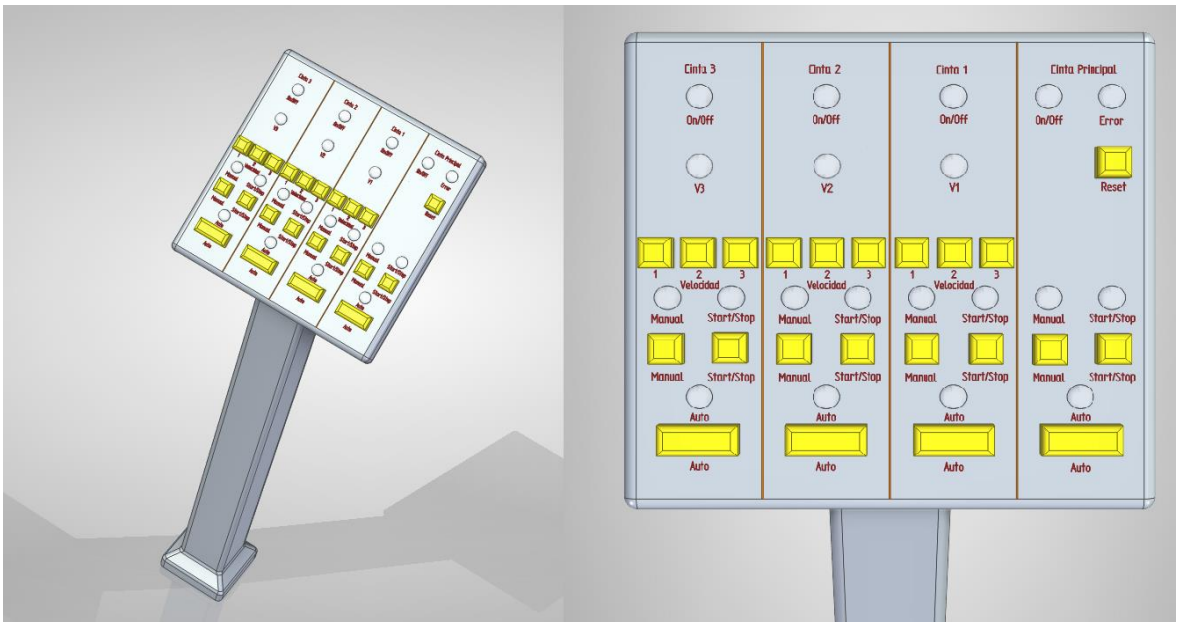


Figura 20 Panel de Control

5.2 CINTA PRINCIPAL

La cinta principal está formada por tres tramos distintos que le dan forma de "L". Consta de 2 tramos rectos y un tramo de curva para hacer un giro de 90 grados.

A lo largo de los dos tramos rectos se colocarán los brazos robóticos y los sensores para monitorizar el avance de los objetos en la cinta.

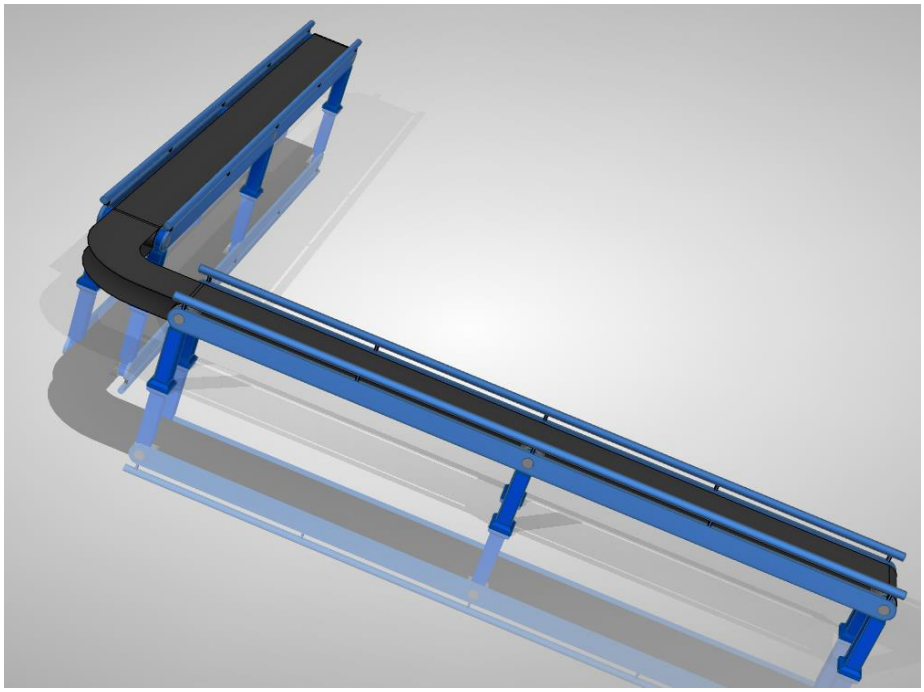


Figura 21 Cinta principal

5.3 CINTAS DE DISTRIBUCIÓN

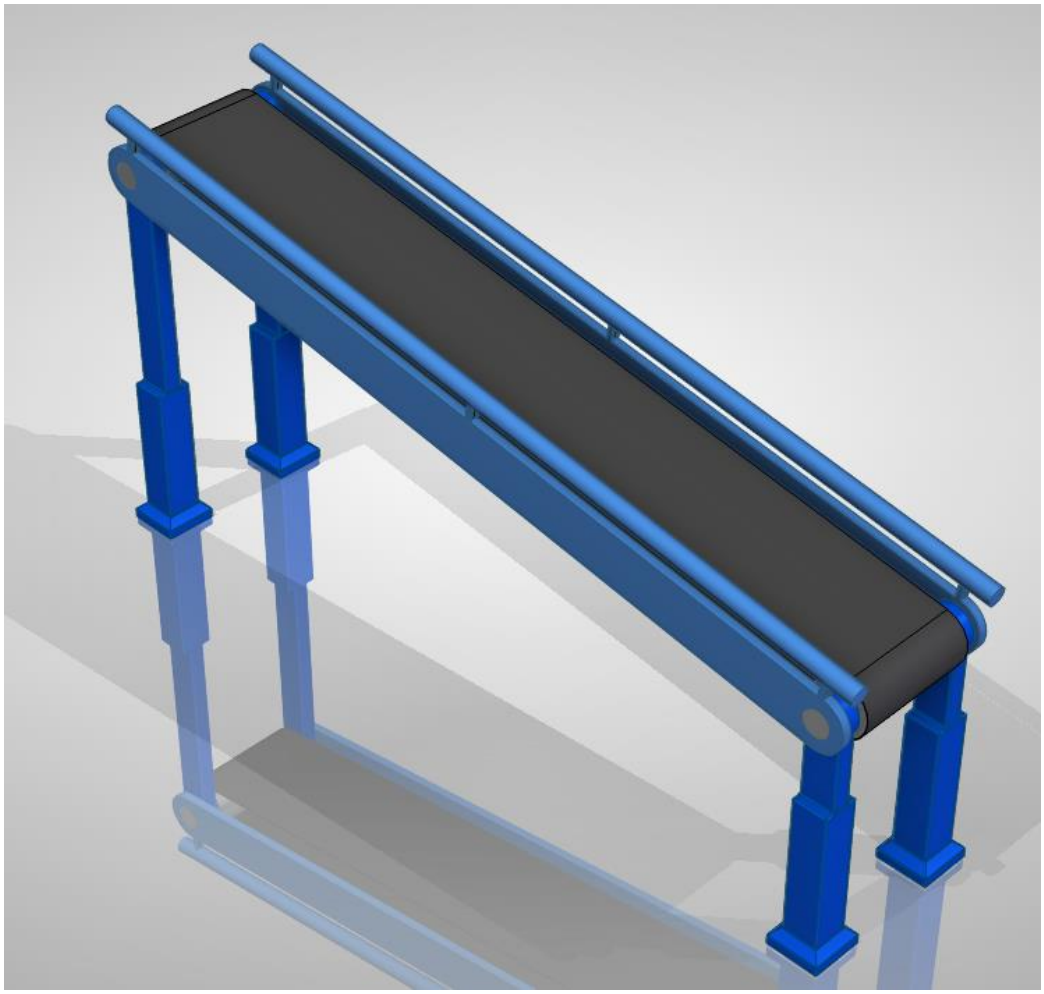


Figura 22 Cinta de distribución

Las cintas de distribución tienen un diseño similar a la cinta principal. Estas cintas están inclinadas y están a una altura ligeramente mayor a la cinta principal para que cuando los objetos caigan en la cinta principal caigan más centrados en la cinta.

5.4 OBJETOS PARA RECOGER DE LA CINTA: TIPO 1, TIPO 2 Y TIPO 3

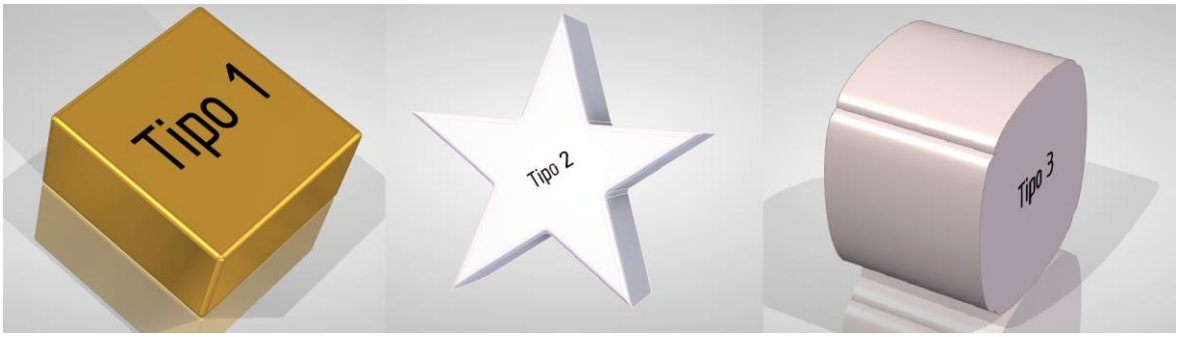


Figura 23 Objeto Tipo 1, Tipo 2 y Tipo 3

Se han creado tres tipos de objeto distintos para poder dar una mayor flexibilidad a la planta y dar más complejidad a la hora de añadir elementos a controlar.

Los objetos creados se han hecho con diseños bastante diferentes y llamativos para que el usuario pueda identificarlos rápidamente.

5.5 BRAZO ROBÓTICO Y PINZAS

El brazo utilizado es un modelo IRB120_3-58 IRC5 fabricado por ABB [13]. A continuación, se muestran algunas de sus características técnicas más relevantes [14]:

- Alcance máximo: 0.58 m.
- Número de ejes: 6.
- Máxima capacidad de carga: 3 Kg.
- Tiempo de aceleración 0 – 1 m/s: 0.07 s.
- Peso: 25 Kg.

Figura 24 Brazo robótico

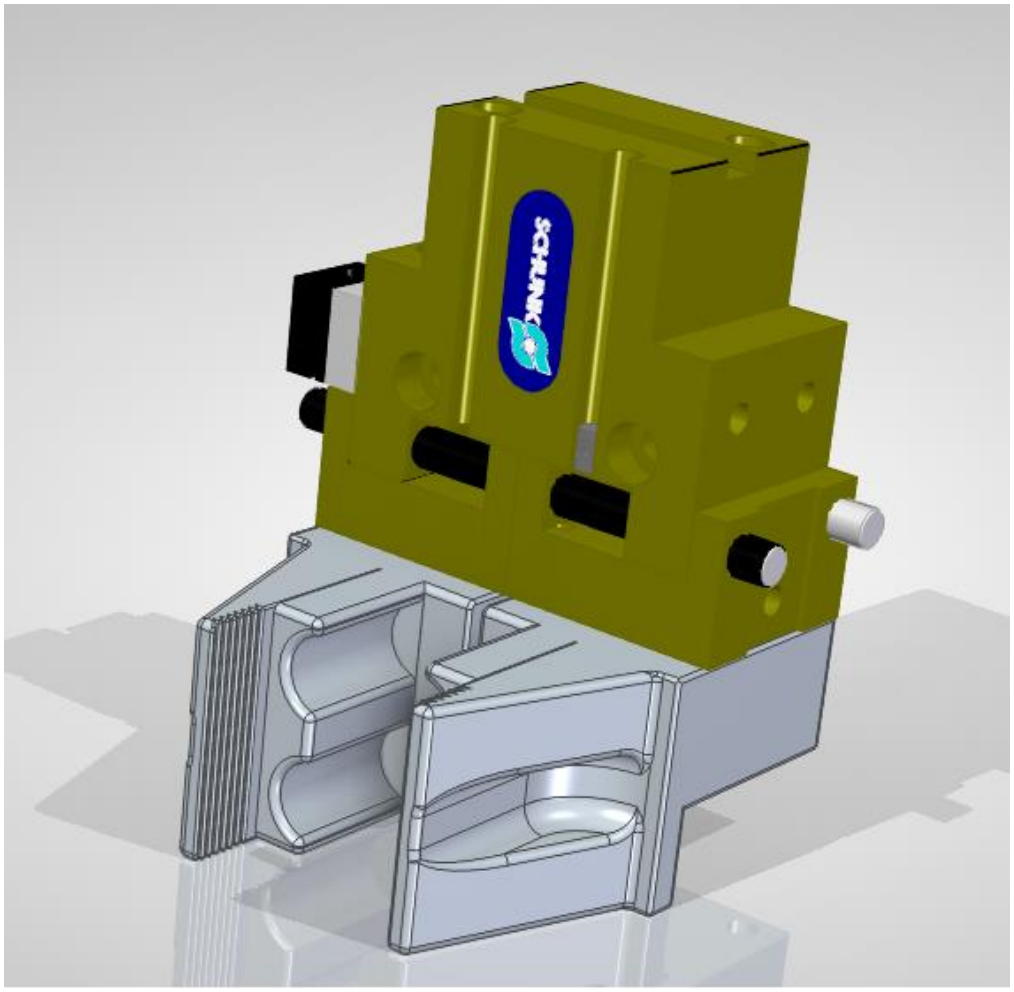


Figura 25 Sistema de agarre

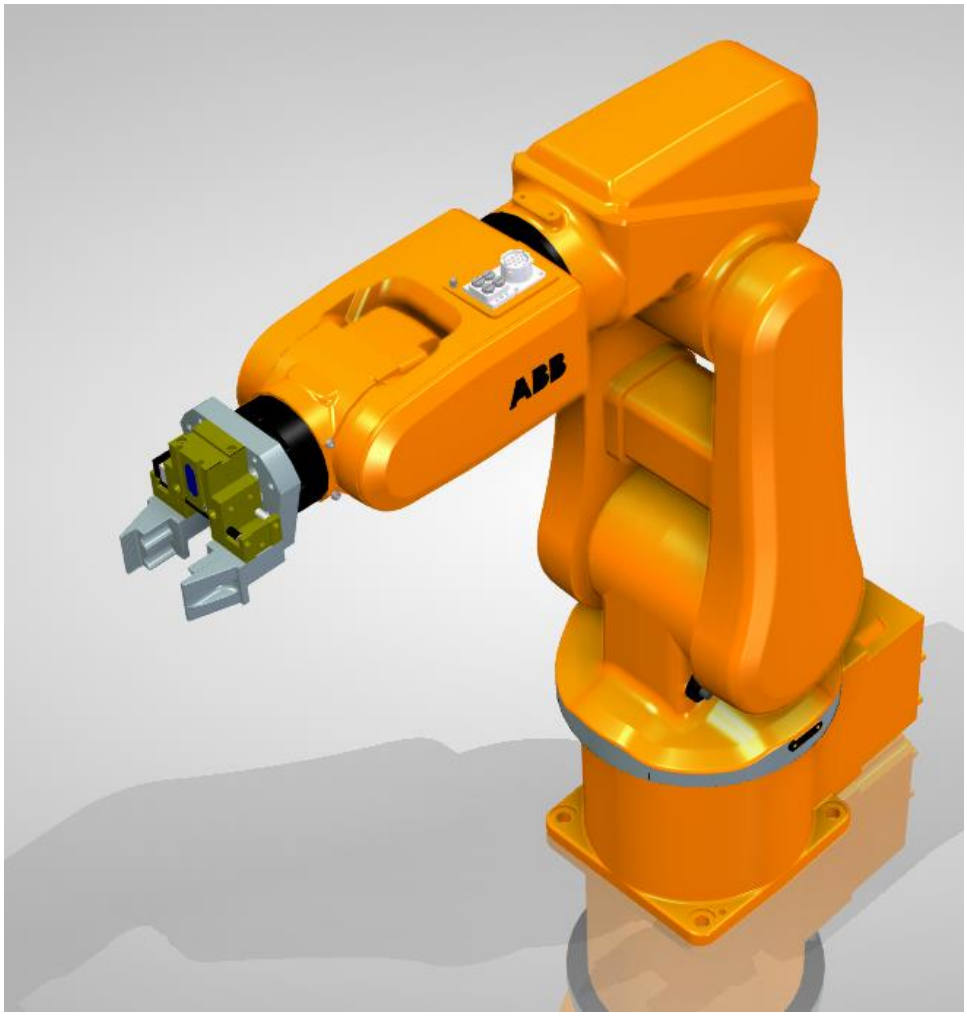


Figura 26 Brazo robótico con sistema de agarre

Capítulo 6. DESARROLLO EN UNITY

Todos los códigos creados para el proyecto se encuentran en ANEXO V: Códigos Fuente.

6.1 COMUNICACIÓN CON PLCSIM

Para realizar la comunicación con el PLC se ha utilizado la librería Sharp7 [15], [16]. La librería Sharp7 permite abrir comunicación con NetToPLCsim mediante el protocolo de comunicación S7, mediante este protocolo se puede acceder a los valores de los datos guardados en los bloques de datos del PLC y cambiar sus valores, así como leerlos.

Para facilitar la configuración de la comunicación con el PLC cuando se cambia de ordenador o de PLC, se ha permitido que quien utilice este proyecto pueda cambiar la información de la dirección IP, el rack y el slot del PLC sin necesidad de abrir los scripts del proyecto. Para esto se han definido las variables ip, rack y slot como parámetros serializados, esto permite a quien use el proyecto cambiar el valor de variables del código desde el editor de Unity.

```
[Header("Parámetros del PLC")]
[SerializeField] public string ip;
[SerializeField] public int rack;   //0;
[SerializeField] public int slot;   //1;
```

Figura 27 Serialización de parámetros del PLC

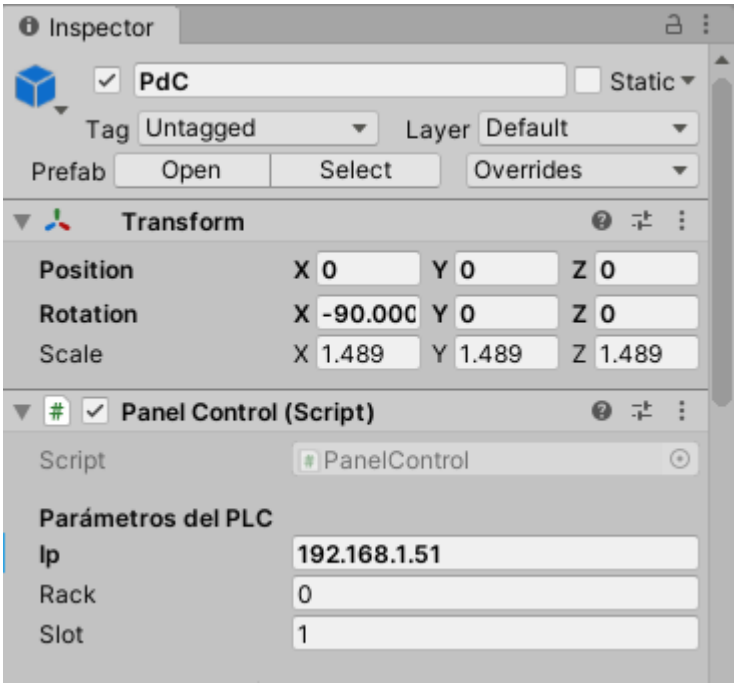


Figura 28 Acceso a parámetros serializados de la comunicación desde el editor

6.1.1 CREACIÓN DEL CLIENTE PARA LA COMUNICACIÓN:

```
        public S7Client client = new S7Client();
```

Figura 29 Cliente S7

Se crea una variable *client* del tipo S7Client. Las variables del tipo S7Client son las utilizadas para acceder a las funciones e instancias de la librería Sharp7.

6.1.2 CONEXIÓN CON EL PLC

```
int connect = client.ConnectTo(ip, rack, slot);
```

Figura 30 Conexión con el PLC

La función ConnectTo() de la clase S7Client permite abrir la línea de comunicación con el PLC.

Recibe:

- IP: string el cual contiene la dirección IP a la cual se quiere conectar.

- Rack: int que indica el número de rack correspondiente al PLC.
- Slot: int que indica el número de slot correspondiente al PLC.

Devuelve un valor tipo int que indica el código de error tras realizar la operación [17], si el valor devuelto es distinto de 0 ha habido un error.

6.1.3 LECTURA Y ESCRITURA DE DATOS

Para la escritura y lectura de datos primero se necesita crear un vector que funciones como buffer en el cual se guardan los valores de las variables a las que se accede.

```
public byte[] buffer = new byte[54];
```

Figura 31 Buffer de datos

En este caso se ha definido el buffer como tipo byte porque todas las variables que se van a leer son booleanas.

Para facilitar el acceso a los valores de las distintas variables del PLC se ha creado una clase llamada *Datos* que almacena la información necesaria para un fácil acceso a los valores de las variables del PLC.

La clase *Datos* tiene los siguientes parámetros:

- pos: int, guarda la posición de la variable en el bloque de datos.
- bit: int, guarda el bit asociado a la variable en el bloque de datos.
- nomb: string, guarda el nombre de la variable.
- pos_vec: guarda la posición que va a tener la variable en el buffer de datos.

```
public Datos(int p, int b, string nombre, int pos_v)
{
    pos = p;
    bit = b;
    nomb = nombre;
    pos_vec = pos_v;
}
```

Figura 32 Clase Datos

Para facilitar todo lo relacionado con el acceso a los datos del PLC se ha creado la clase *Comunicacion*. Dentro de esta clase se han creado funciones para permitir la lectura y escritura de las variables del PLC mediante el uso de las variables del tipo *Datos*, a cada variable del PLC se le ha asignado una variable del tipo *Datos* y así es más fácil acceder a ellas.

6.1.3.1 Escritura

```
public bool setEstado(Datos variable, bool estado)
    {
        //Guarda los códigos de error del proceso de escritura
        // Si vale 0 la escritura es correcta
        int escritura;
        //Función que permite cambiar el valor de una variable
        S7.SetBitAt(ref buffer, variable.pos, variable.bit, estado);
        escritura = client.DBWrite(1, 0, buffer.Length, buffer);
        if (escritura == 0)
        {
            //Debug.Log(variable.nomb + ": " + S7.GetBitAt(buffer,
variable.pos, variable.bit));
            return true;
        }
        else
        {
            //Se indica si falla la escritura
            Debug.LogError("Error de escritura");
            //Muestra el código de error en la consola
            Debug.LogError(escritura);
            return false;
        }
    }
```

Figura 33 Función setEstado

La función es pública para que pueda ser accedida desde otros scripts y así no tener que definir más funciones de comunicación.

Recibe:

- variable: parámetro del tipo *Datos* asociado a la variable del PLC que se quiere cambiar.
- estado: booleana, valor *true* o *false* que se quiere dar a la variable.

La función devuelve *true* si la escritura se ha realizado con éxito o *false* si ha habido un fallo.

6.1.3.2 Lectura

El método de lectura es análogo al de escritura.

```
public bool getEstado(Datos variable)
    {
        //Guarda los códigos de error del proceso
        // de lectura
        //Si vale 0 la lectura es correcta
        int lectura;
        //Guarda el estado tomado del PLC
        bool estado;
        lectura = client.DBRead(1, 0, buffer.Length, buffer);
        if (lectura == 0)
        {
            //Función que permite leer el valor de una variable
            estado = S7.GetBitAt(buffer, variable.pos, variable.bit);
            //Debug.Log(variable.nomb + ": " + estado);
            return estado;
        }
        else
        {
            //Se indica si falla la lectura
            Debug.LogError("Error de lectura");
            //Muestra el código de error en la consola
            Debug.LogError(lectura);
            return false;
        }
    }
```

Figura 34 Función getEstado

Recibe:

- variable: parámetro del tipo *Datos* asociado a la variable del PLC que se quiere cambiar.

La función devuelve *true* si la lectura se ha realizado con éxito o *false* si ha habido un fallo.

6.2 INTERACCIÓN CON EL USUARIO

Para permitir que el usuario se pueda mover con libertad por toda la planta se ha creado un controlador en primera persona, la realidad virtual no se va a implementar debido a que por el periodo de cuarentena no se dispone de los medios.

Con el controlador en primera persona el usuario puede desplazarse e interactuar con su entorno. Los scripts asociados al controlador se encuentran en Código PlayerMovement y Código Mouse_Look.

Para que el usuario pueda interactuar con el entorno se ha usado el sistema RayCast de Unity [18], el cual consiste en un rayo que sale del centro de la cámara y detecta los collider de los objetos con los que impacta, este sistema también puede ser utilizado en realidad virtual y facilita mucho el sistema de agarre de objetos.

```
       //Rayo que sale del centro de la cámara para seleccionar los
objetos
        var ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit))
        {
            var select = hit.transform;
            if (select.CompareTag(seleccionable))
            {
                seleccion = select;
                if (select.GetComponent<Renderer>() != null)
                {
                    select.GetComponent<Renderer>().material =
mat_resaltado;
                    if (Input.GetMouseButtonDown(0))
                    {
                        nombre = select.gameObject.name;
                        pulsado = true;
                    }
                }
            }
        }
```

Figura 35 Sistema RayCast para detección de objetos

Mediante el uso del sistema RayCast se ha creado la clase *Selector*, la cual se encarga de detectar cuando se pulsan los botones para controlar la cinta. Para facilitar el saber qué botón

se va a pulsar se ha implementado un código dentro de la clase *Selector* que resalta el color de los objetos con los que se puede interactuar.

El funcionamiento se basa en RayCast, cuando el rayo impacta en un objeto de la escena el código de la clase *Selector* accede a la etiqueta que tiene ese objeto, y si el nombre de la etiqueta es "seleccionable" el material de ese objeto se resalta para que el usuario sepa que puede interactuar con ese objeto.

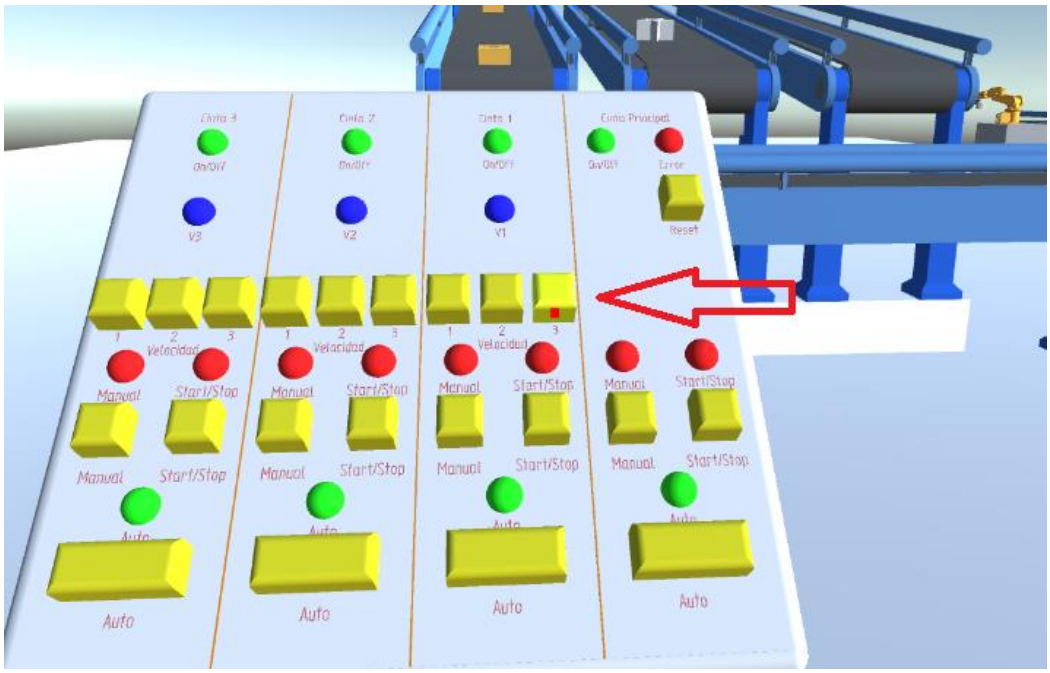


Figura 36 Material resaltado

Para facilitar que el usuario sepa dónde está el centro de la pantalla y le sea más fácil apuntar a los objetos con los que quiera interactuar, se ha puesto un cuadrado rojo a modo de mira para facilitar la selección de objetos.

El código de la clase *Selector* se encuentra en Código Selector.

6.3 CONTROL DE CINTAS TRANSPORTADORAS

Para simplificar el funcionamiento de las cintas transportadoras se ha decidido tomar una aproximación al problema que evite tener que mover la cinta y que con ella se mueva el objeto dado que la complejidad del problema es bastante elevada como. Se ha decidió poner un box collider en la superficie de la cinta transportadora y cuando un objeto entra en dicho collider adquiere una velocidad cuya trayectoria viene dada por dos empty gameobjects, uno en cada extremo de la cinta.

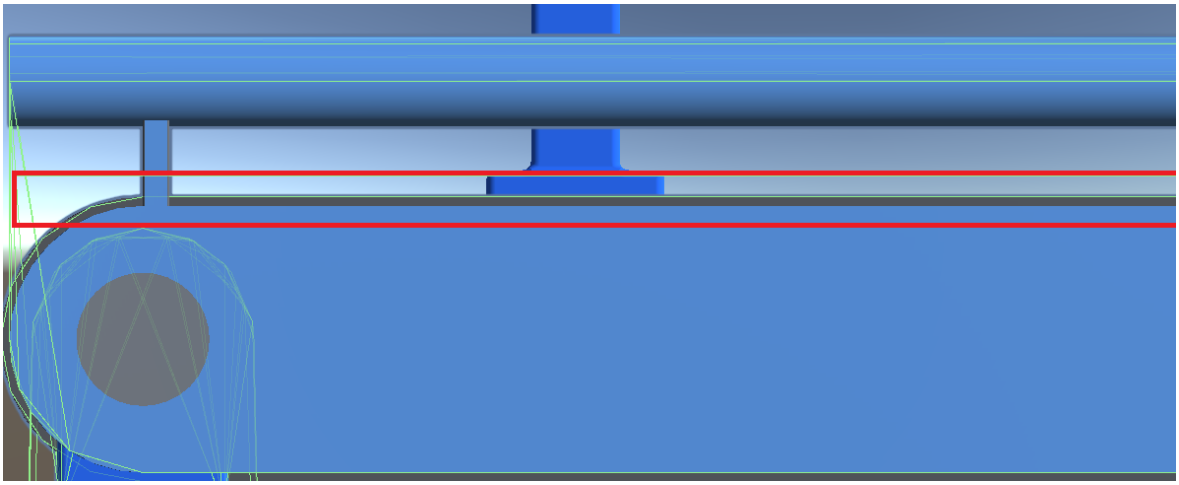


Figura 37 Detección de objetos sobre la cinta transportadora

El recuadro rojo mostrado en la Figura 37 Detección de objetos sobre la cinta transportadora es el box collider usado para detectar cuando hay un objeto en la cinta.

Este collider tiene la propiedad de trigger activada, esto permite ejecutar unas instancias determinadas cunado se detecta un objeto dentro del collider.

```
    private void OnTriggerStay(Collider other)
    {
        bool error = Color_Error.GetComponent<ColorPdC>().est_error;
        bool estado_1 = Color_Error.GetComponent<ColorPdC>().est_cinta_1;
        bool estado_2 = Color_Error.GetComponent<ColorPdC>().est_cinta_2;
        bool estado_3 = Color_Error.GetComponent<ColorPdC>().est_cinta_3;

        //Si está en error las cintas no se mueven
        if (error == false)
        {
```

```
            if (cinta_id == 1 && estado_1 == true)
                other.gameObject.transform.position += direccion *
velocidad * Time.deltaTime;
            if (cinta_id == 2 && estado_2 == true)
                other.gameObject.transform.position += direccion *
velocidad * Time.deltaTime;
            if (cinta_id == 3 && estado_3 == true)
                other.gameObject.transform.position += direccion *
velocidad * Time.deltaTime;
            if (cinta_id == 0)
                other.gameObject.transform.position += direccion *
velocidad * Time.deltaTime;
        }
    }
```

Figura 38 Movimiento de objetos en la cinta

La función OnTriggerStay permite detectar cuando un objeto permanece dentro de un collider.

La dirección de movimiento usada en la Figura 38 Movimiento de objetos en la cinta viene dada por el vector que une dos puntos en los extremos de la cinta.

```
        //Se define la trayectoria al iniciar la simulación
        float x = punto_1.transform.position.x -
punto_2.transform.position.x;
        float y = punto_1.transform.position.y -
punto_2.transform.position.y;
        float z = punto_1.transform.position.z -
punto_2.transform.position.z;
        direccion = new Vector3(x, y, z);
```

Figura 39 Dirección de movimiento en la cinta

Para tomar la curva en la cinta principal el planteamiento utilizado anteriormente no es válido porque solo permite trayectorias rectas. Para girar en lugar de usar dos puntos para definir una trayectoria recta se ha usado un punto situado en el centro de giro de la curva y se ha hecho que los objetos que entren en un collider que hay sobre la curva, igual que con los tramos rectos, roten sobre dicho punto y así parezca que giran.

```
    private void OnTriggerEnter(Collider other)
    {
other.gameObject.transform.RotateAround(Centro_Rotacion.transform.positio
n, Vector3.up, Velocidad_Rotacion);
    }
```

Figura 40 Rotación resoectoa un punto

Para el funcionamiento de las cintas transportadoras se han creado las clases *Mover* y *Girar_Curva*, las cuales se encargan de mover los objetos en los tramos rectos y de tomar la curva. Estas clases se encuentran en ANEXO V: Códigos Fuente en los apartados Código Mover y Código Girar_Curva.

6.4 CONTROL DE BRAZOS ROBÓTICOS

Para poder simular el movimiento del brazo robótico que hay disponible en la minifábrica de ICAI se tuvieron que añadir GameObject vacíos en las articulaciones del brazo para poder definir unos ejes de giro.

El sistema ideado para realizar los movimientos de los brazos consiste en usar vectores predefinidos que guarden los giros en ejes X, Y, Z cada una de las articulaciones. Para asegurarse de que esto funciona es importante tener una jerarquía en la cual se mantengan las relaciones entre hijos y padres de las distintas partes del brazo para que no se deforme el brazo o se produzcan giros imposibles. En este caso se ha decidido que lo mejor es que la base del brazo fuese el padre de todos los elementos y que cada articulación tenga como hijos los tramos de brazo que tiene conectados.

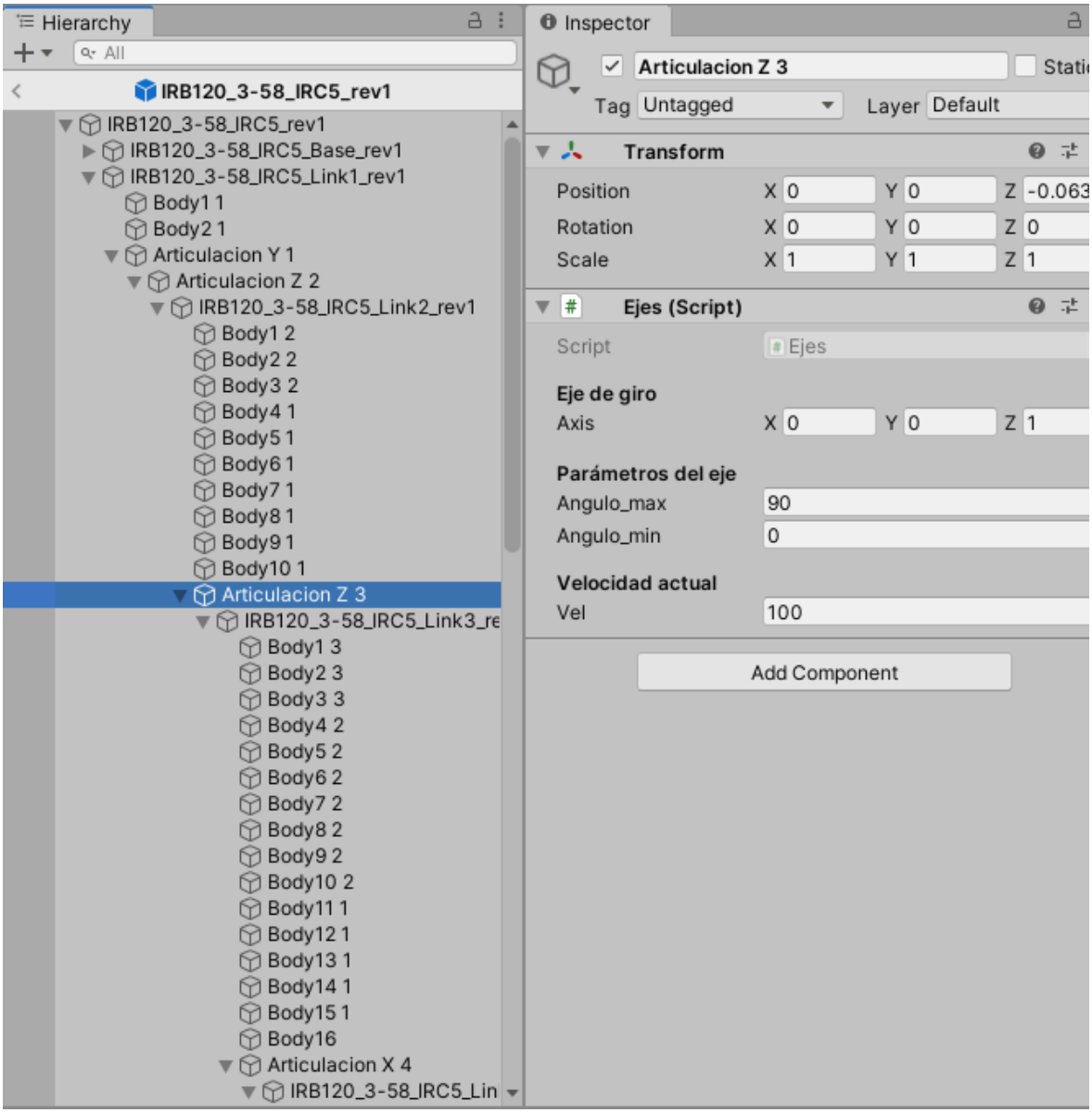


Figura 41 Jerarquía de los elementos del brazo robótico

La clase *Ejes* se encuentra en ANEXO V: Códigos Fuente en el apartado Código Ejes.

Para facilitar la implementación de los robots se ha creado la clase *Ejes*, esta clase se encarga de definir entorno a que eje del espacio gira cada articulación, la velocidad de giro y los ángulos máximos de giro.

```
[Header("Eje de giro")]
public Vector3 Axis;
```

```
[Header("Parámetros del eje")]
public float angulo_max = 90;
public float angulo_min = 0;

//Por defecto la velocidad de la articulacón vale 150
private static float velocidad = 150;
```

Figura 42 Variables de la clase Ejes

Esta clase también permite conocer el ángulo que esta girado un eje mediante la función *Get_Angulo()* para comprobar si un eje ha completado un giro.

```
public float Get_Angulo()
```

Figura 43 Función Get_Angulo

Para realizar los movimientos se ha utilizado la clase *Rotar*. Para los movimientos de los brazos se ha creado una máquina de estados y dependiendo del estado cada articulación del brazo girará un ángulo determinado.

Los estados definidos son:

- Reposo: el brazo se mantiene en la posición inicial a la espera.
- Mover: el brazo se mueve para colocarse en una posición en la cual puede coger el producto de la cinta.
- Coger: en este estado se coge el producto de la cinta cuando pasa por la pinza del brazo y se retira de la cinta.

Una vez se ha cogido el objeto de la cinta y se ha dejado el brazo vuelve al estado de reposo.

```
private void Update()
    {
        //movimientos para cada estado del actuador
        if (actuador.GetComponent<Actuador>().estado == "Reposo")
            Movimiento(0);
        if (actuador.GetComponent<Actuador>().estado == "Mover")
            Movimiento(1);
        if (actuador.GetComponent<Actuador>().estado == "Coger" &&
Comprueba_Movimiento(2) == false)
        {
            Movimiento(2);
        }
    }
```

Figura 44 Función Update de la clase Rotar

La función movimiento se encarga de hacer que cada eje gire el ángulo de tiene que girar.

```
private void Movimiento(int movimiento)
```

Figura 45 Función Movimiento

La función *Movimiento* recibe el numero correspondiente al movimiento que se va a ejecutar: 1, 2 o 3.

```
[Header("Parametros del brazo")]
[SerializeField] private Ejes[] ejes;
[SerializeField] private int num_ejes;
[SerializeField] private int num_movimientos;
[SerializeField] public float tolerancia;

[Header("Comprobar movimiento")]
public bool movimiento_comletado = false;

[Header("Movimientos programados")]
public float[] movimiento_1;
public float[] movimiento_2;
public float[] movimiento_3;
public float[,] movimientos;
```

Figura 46 Parámetros serializados en el script

Los ángulos que tiene que girar cada articulación son introducidos por el usuario a través del inspector de Unity mediante la serialización de las variables en el script.

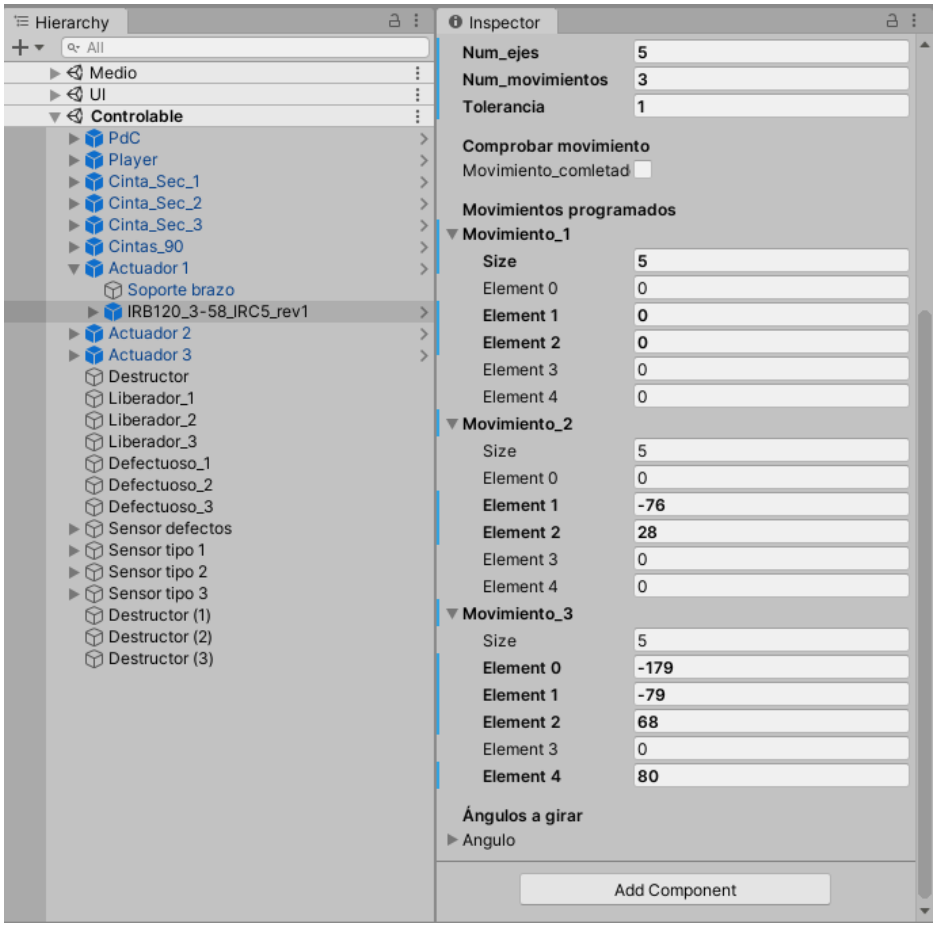


Figura 47 Serialización de parámetros

La clase *Rotar* se encuentra en ANEXO V: Códigos Fuente en el apartado Código Rotar.

6.5 PANEL DE CONTROL

El panel de control consiste en una serie de botones para cambiar parámetros del sistema y una serie de luces que indican el estado de los parámetros que se controlan.

Mediante el panel de control se puede cambiar el tiempo que tarda en ponerse un nuevo objeto en las cintas, el control automático o manual y encender a apagar las cintas. También se incluye un botón de reset que permite reiniciar el sistema después de que se haya detectado un error que hace que se paren las cintas y los brazos.

6.5.1 PULSADO DE BOTONES

Para controlar si se pula un botón o no se utiliza la variable booleana *pulsado* de la clase *Selector* la cual tiene valor *true* si se ha pulsado algún botón o *false* si no se pulsa ninguno.

Se realiza una comprobación aparte de si se pulsa el botón de reset, porque en caso de que se encuentre en estado de error da igual que se pulsen los demás botones ya que no va a cambiar nada del funcionamiento porque todo está parado. Si hay un estado de error y se pulsa el botón de reset se vuelve a poner todo en marcha y si no hay estado de error el botón de reset no produce no hace nada.

```
//Se comprueba si se pulsa el reset
if (reset_pulsado == true)
{
        pulsa_reset = false;
}
//Se detecta si se ha pulsado algún botón
if (selector.pulsado)
{
        estado = Control_Botones(selector.pulsado, selector.nombre);
        color.Color_pdc();
        color.Cambia_Vel(selector.nombre);
        estado = true;
}
```

Figura 48 Comprobación de pulsado

Cuando se pulsa un botón se ejecuta la función *Control_Botones*, esta función se encarga de cambiar los estados de las variables que se controlan desde el panel de control y de transmitir estos cambios al PLC para que se actualice el estado de las variables a controlar.

```
public bool Control_Botones(bool pulsa, string nombre)
```

Figura 49 Función Control_Botones

Recibe:

- pulsa: indica si se ha pulsado un botón (*true*) o no (*false*).
- nombre: el nombre del botón que se ha pulsado.

Siempre devuelve *true* para indicar que se han realizados los cambios correspondientes a la pulsación del botón.

La función *Control_Botones* se encuentra en ANEXO V: Códigos Fuente en Código Comunicación.

La función Sensores se encarga de comprobar el estado de los distintos sensores colocados en la cinta y de transmitir el valor de los sensores al PLC para asegurar de que el estado de los sensores es el mismo en el PLC que el que se muestra en la simulación en Unity.

6.6 CREACIÓN Y DETECCIÓN DE OBJETOS DEFECTUOSOS

6.6.1 CREACIÓN DE DEFECTUOSOS

Con el objetivo de dar un poco más de profundidad a la simulación se implementó un sistema que hace que algunos objetos sean clasificados como defectuosos.

La cantidad de objetos defectuosos puede ser definida por el usuario, permitiendo que no haya ningún defectuoso o que todos sean defectuosos. Para controlar el porcentaje de defectuosos se usa una variable pseudoaleatoria que varía entre 0 y 100, en el caso de que la variable pseudoaleatoria sea menor que un valor introducido por el usuario se cambia la etiqueta de ese objeto por “Defectuoso”.

```
public class Defectuoso : MonoBehaviour
{
    [Header("Porcentaje de defectuosos")]
    [Range(0f, 100f)]
    [SerializeField] private float defectuosos;
    private void OnTriggerStay(Collider other)
    {
        //variable psudo aleatoria con distribucion uniforme
        //puede tomar valores entre 0 y 100
        float random = Random.Range(0f, 100f);
        //Si la varaible random es menor a la tasa de defectuosos
definida
        // el objeto se etiqueta como defectuoso
        if(random < defectuosos)
        {
            other.gameObject.tag = "Defectuoso";
        }
    }
}
```

Figura 50 Clase Defectuoso

El valor introducido por el usuario para controlar el número de defectuosos es serializado para que pueda ser introducido desde el inspector de Unity sin necesidad de modificar nada del código.

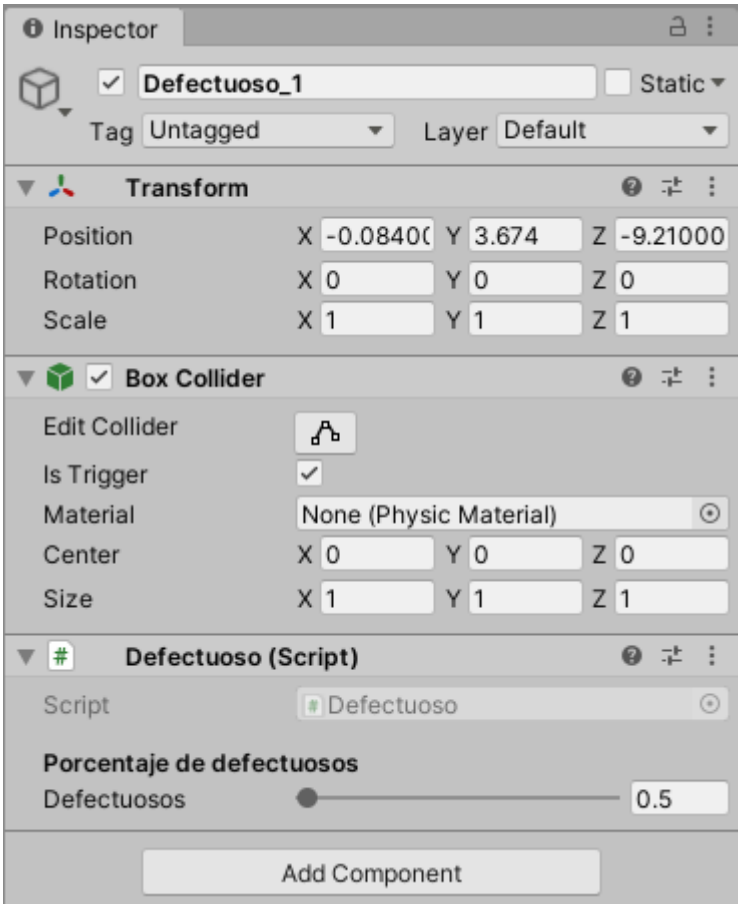


Figura 51 Serialización del porcentaje de defectuosos

Para abordar la creación de objetos defectuosos se ha creado la clase *Defectuoso*, se encuentra adjunta en Código Defectuoso.

6.6.2 DETECCIÓN DE DEFECTOS

Para que el usuario pueda ver que hay objetos defectuosos se ha creado un detector de defectos. Este detector se encuentra en la cinta principal antes de los brazos robóticos, tiene

un led el cual mostrará el color verde si el objeto que pasa por el sensor en ese momento no tiene un defecto y mostrará el color rojo si el objeto que pasa por el sensor es defectuoso.

Para detectar los defectos se ha creado la clase *Detector_Defectos*, la cual se encuentra adjunta en ANEXO V: Códigos Fuente en el apartado Código Detector_Defectos.

6.7 DETECTOR DE TIPO DE OBJETO

Dado que los brazos están programados para coger un solo tipo de objeto es necesario tener un sensor que diga al brazo si el objeto que va a pasar es del tipo que tiene que coger o no.

Para esto se ha implementado una solución parecida a la usada en **¡Error! No se encuentra el origen de la referencia.**. Se tiene un sensor el cual detecta si el objeto es del tipo correcto y pone de color verde un led, si el objeto es de otro tipo se pone de color rojo. Esto también vale para monitorizar por dónde van los objetos en la cinta, si un sensor tiene color rojo indica que el último objeto que ha pasado por la cinta no es del tipo correcto.

El funcionamiento del sensor se basa en un collider con trigger que lee la etiqueta que tiene el objeto que pasa por él y si la etiqueta coincide con el nombre de etiqueta de los objetos que tiene que coger ese brazo el sensor se muestra verde.

Las etiquetas utilizadas son:

- tipo 1
- tipo 2
- tipo 3

```
        //Se compureba si la etiqueta del objeto coincide con el tipo
buscado
        if(other.gameObject.CompareTag(tipo) == true)
        {
            luz.material.SetColor("_Color", Color.green);
            //Se cambia el color de la luz de cada tipo
            if(tipo == "tipo 1")
            {
                sensor_tipo_1 = true;
            }
```

```
            if (tipo == "tipo 2")
            {
                sensor_tipo_2 = true;
            }

            if (tipo == "tipo 3")
            {
                sensor_tipo_3 = true;
            }

        }
```

Figura 52 Detección de la etiqueta de un objeto

En este caso se ha creado la clase *Detector_Tipo*, se encuentra en ANEXO V: Códigos Fuente en el apartado Código Detector_Tipo.

Capítulo 7. PRESUPUESTO

En este capítulo se va a realizar una estimación del coste del desarrollo e implantación de este proyecto. En las cuentas del presupuesto no se tendrá en cuenta el coste de teclado, ratón y monitor por no considerarse gastos excepcionales relacionados con el proyecto.

Horas de trabajo	Coste(€/hora)	Coste total
500	20	10000

Tabla 2 Estimación de coste de horas trabajadas

Recursos	Cantidad (Uds.)	Coste Unitario (€)	Coste Total (€)
Solid Edge	1	0	0
Unity	1	0	0
NetToPLCsim	1	0	0
TIA Portal	1	3288	3288
PC Compatible con VR	1	1500	1500
Oculus Rift	1	450	450
		Total	5238

Tabla 3 Estimación de coste de los recursos

Por lo tanto, el coste total estimado del proyecto sería de 15238€.

Para las licencias de SolidEdge y Unity se han usado las licencias gratuitas. En el caso de que se use NX en lugar de SolidEdge el coste de la licencia seria de 11300€ más 2400€ anuales. Para el cálculo del presupuesto no se tendrá en cuenta el coste anual de NX porque el proyecto se ha realizado en menos de un año.

Recursos	Cantidad (Uds.)	Coste Unitario (€)	Coste Total (€)
NX	1	11300	11300
Unity	1	0	0
NetToPLCsim	1	0	0
TIA Portal	1	3288	3288
PC Compatible con VR	1	1500	1500
Oculus Rift	1	450	450
		Total	16538

Tabla 4 Estimación de coste de los recursos en caso de usar NX

En el caso de haber usado NX el coste del proyecto se incrementaría hasta 26538€.

Capítulo 8. ANÁLISIS DE RESULTADOS

8.1 FUNCIONAMIENTO FINAL

El funcionamiento final del proyecto ha sido el esperado, se ha conseguido ejecutar con éxito una simulación mediante la interacción entre Unity y PLCsim.

En la simulación se pueden controlar distintos sistemas, todos a través de el mismo PLC, mediante el uso de un panel de control. Este panel de control sirve de interfaz entre el usuario y el PLC. Con el panel de control se puede controlar:

- Tiempos de salida de cada producto.
- La velocidad de las cintas.
- Encender ya apagar las distintas cintas.
- Cambiar entre modo manual y modo automático en las cintas.
- Encender y apagar las cintas.

Adicionalmente se han añadido tres brazos robóticos los cuales se encargan de coger de la cinta el tipo de producto que tienen asignado. Estos brazos tienen la opción de apagarse y encenderse mediante el uso de un botón situado en la base de cada uno.

También se ha implementado la opción de que ciertos productos que son puestos en la cinta sean defectuosos. Cuando un producto es detectado como defectuoso el brazo al que le corresponde coger los objetos de su tipo no lo coge y llega hasta el final de la cinta donde es destruido.

8.2 PROBLEMAS ENCONTRADOS EN EL DESARROLLO

Los principales problemas encontrados durante el desarrollo del proyecto se han encontrado a la hora de establecer un sistema eficaz y fiable de comunicaciones. La comunicación entre Unity y PLCsim se ha comprobado con la práctica que es lenta en comparación con la

velocidad con la que se ejecutan los scripts en Unity, lo que provocaba que en ciertos momentos se enviasen instrucciones en varias líneas de código seguidas y PLCsim no ejecutase todas las instrucciones. Esto se ha solucionado espaciando el envío de instrucciones y creando funciones específicas para enviar las instrucciones que más se repetían.

Capítulo 9. CONCLUSIONES Y TRABAJOS FUTUROS

9.1 CONCLUSIONES

Durante el desarrollo del proyecto se ha visto que la tecnología de la realidad virtual tiene un gran potencial y mucho que ofrecer al campo de las simulaciones de procesos industriales. El uso de las técnicas de realidad virtual en este campo puede propiciar un gran avance en cómo se entrena al personal de diferentes instalaciones industriales, así como permitir un mejor entrenamiento del personal. Esto permite que aquellas personas que vayan a operar en los entornos simulados puedan conocer cómo será su lugar de trabajo y cómo funcionan las distintas herramientas con las que se tendrá que trabajar en el día a día antes incluso de que dicha instalación industrial este construida. Solo se necesita el modelo 3D de la instalación y conoce el funcionamiento de los sistemas.

Esto además puede permitir a cotar los tiempos de entrenamiento de personal, ya que se puede iniciar la formación mucho antes de la puesta en marcha de la planta industrial, lo cual también supone un ahorro para la empresa porque se puede reducir el tiempo de puesta en marcha.

9.2 TRABAJOS FUTUROS

Resulta fundamental dar el paso final en el proyecto e implementar la realidad virtual. No se ha podido implementar debido a la imposibilidad de acceder al equipamiento de realidad virtual de ICAI debido al estado de cuarentena por el COVID-19. Esta implementación supone un pequeño paso que consta de poca dificultad, pero causaría una gran diferencia en el resultado final.

Para posibles proyectos que se vayan a llevar a cabo en el futuro relacionados con el tema de este trabajo seria la implementación de un sistema multijugador. Con este sistema varias personas podrían estar en la misma simulación y así se podrían simular procesos en los cuales

sean necesarias más de una persona y en los que la coordinación y la interacción es necesaria. Esto también puede abrir la puerta simular sistemas más grandes y complejos con más de un PLC.

Capítulo 10. REFERENCIAS

[1] HTC, «VIVE Home Page,» [En línea]. Available: https://www.vive.com/eu/. [Último acceso: 1 Febrero 2020].

[2] Oculus Rift, «Oculus Rift Home Page,» [En línea]. Available: https://www.oculus.com/rift/?locale=es_LA#oui-csl-rift-games=mages-tale. [Último acceso: 1 Febrero 2020].

[3] PlayStation, «PlayStationVR,» Sony, [En línea]. Available: https://www.playstation.com/es-es/explore/playstation-vr/. [Último acceso: 1 Febrero 2020].

[4] NetToPLCsim, «NetToPLCsim - Network extension for Plcsim,» [En línea]. Available: http://nettoplcsim.sourceforge.net/. [Último acceso: 26 04 2020].

[5] Unity Technologies, «Unity Home Page,» [En línea]. Available: https://unity.com/es. [Último acceso: 26 04 2020].

[6] Siemens, «Siemens NX,» [En línea]. Available: https://www.plm.automation.siemens.com/global/es/products/nx/. [Último acceso: 26 04 2020].

[7] Siemens, «Solid Edge Home PAge,» [En línea]. Available: https://solidedge.siemens.com/es/. [Último acceso: 26 04 2020].

[8] Autodesk, «Fusion 360 for students and teachers,» [En línea]. Available: https://www.autodesk.com/products/fusion-360/students-teachers-educators. [Último acceso: 26 04 2020].

[9] A. Avram, L. Samoila, F. Samoila y D. Afewerki, «DESIGNING VIRTUAL REALITY APPROACH FOR PLC,» 2019. [En línea]. Available: https://www.upet.ro/annals/electrical/doc/2019/7%20Alexandru%20Avram.pdf. [Último acceso: 2 6 2020].

[10] U.Osmers y D.Spath, «Virtual reality - An approach to improve the generation of fault free software for programmable logic controllers (PLC),» 6 8 2002. [En línea]. Available: https://ieeexplore.ieee.org/document/558331/authors#authors. [Último acceso: 2 6 2020].

[11] ABB, «IRB 120 CAD Models,» 24 4 2020. [En línea]. Available: https://new.abb.com/products/robotics/industrial-robots/irb-120/irb-120-cad.

[12] SHUNK, «SHUCNK España Home Page,» [En línea]. Available: https://schunk.com/es_es/pagina-de-inicio/. [Último acceso: 27 04 2020].

[13] ABB, «ABB España Home Page,» [En línea]. Available: https://new.abb.com/es. [Último acceso: 27 04 2020].

[14] ABB, «Technical data for the IRB 120 industrial robot,» [En línea]. Available: https://new.abb.com/products/robotics/industrial-robots/irb-120/irb-120-data. [Último acceso: 27 04 2020].

[15] fbarresi, «Sharp7 GitHub page,» 17 Noviembre 2019. [En línea]. Available: https://github.com/fbarresi/Sharp7. [Último acceso: 24 Marzo 2020].

[16] D. Nardella, «Snap 7 Home Page,» [En línea]. Available: http://snap7.sourceforge.net/. [Último acceso: 24 Marzo 2020].

[17] D. Nardella, «S7 Reference Manual,» 9 Diceimbre 2019. [En línea]. Available: http://vps249990.ovh.net/grav/user/pages/16.logo-arduino/Settimino-refman.pdf. [Último acceso: 01 Mayo 2020].

[18] Unity Technologies, «Scripting Reference,» Unity Technologies, [En línea]. Available: https://docs.unity3d.com/ScriptReference/index.html. [Último acceso: 24 Marzo 2020].

[19] Unity Technologies, «Unity Manual,» Unity Technologies, [En línea]. Available: https://docs.unity3d.com/Manual/index.html. [Último acceso: 24 Marzo 2020].

[20] mycroes, «s7netplus GitHub Page,» 17 Julio 2019. [En línea]. Available: https://github.com/S7NetPlus/s7netplus. [Último acceso: 24 Marzo 2020].

[21] in2sight, «Game4Automation Digital Twin Professional - Unity Assets Store,» [En línea]. Available: https://assetstore.unity.com/packages/tools/utilities/game4automation-digital-twin-professional-143543. [Último acceso: 1 Febrero 2020].

[22] Automation24, «Siemens SIMATIC STEP 7 Prof. 2017/V15 - 6ES7810-5CC12-0YA5,» [En línea]. Available: https://www.automation24.es/siemens-simatic-step-7-prof-2017-v15-6es7810-5cc12-0ya5. [Último acceso: 6 6 2020].

ANEXO I: DIAGRAMAS LÓGICOS DEL PLC

LÓGICA DEL BLOQUE ERROR

A1	Si dos o más actuadores están a pagados hay un error
B1	si las cintas de distribución están apagadas hay un error

Tabla 5 Comentarios sobre el bloque Error

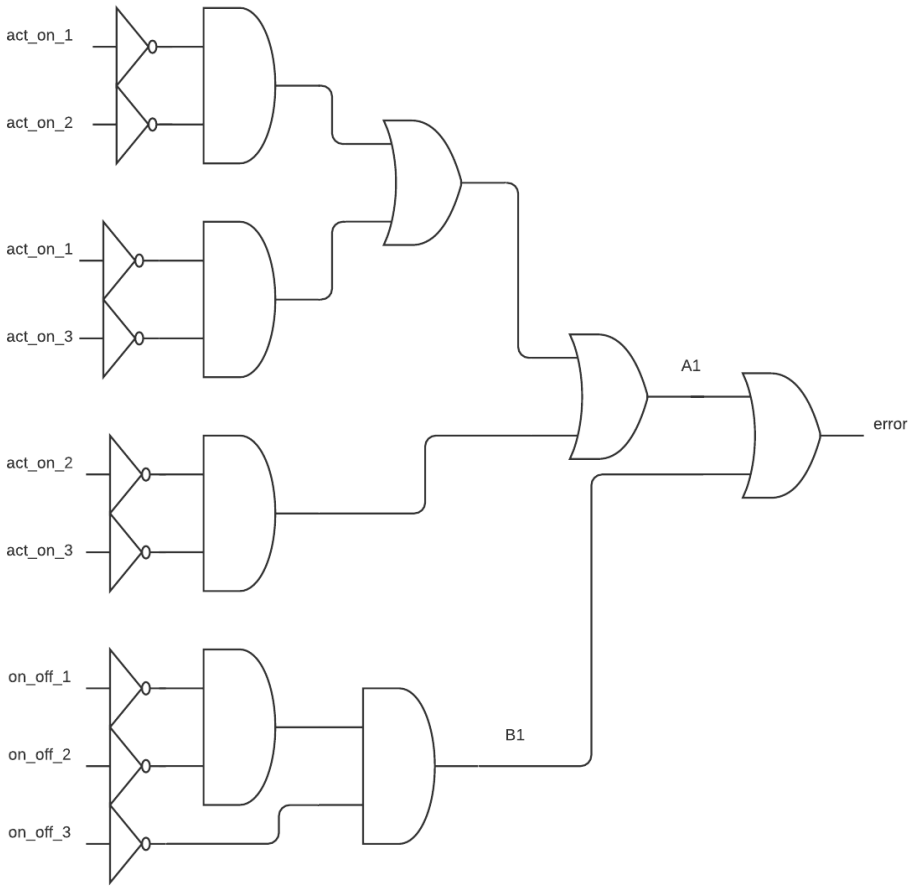


Figura 53 Diagrama lógico del bloque Error

LÓGICA DEL BLOQUE CINTA PRINCIPAL

A1	Para activar la cinta se necesita que el automático esté activo o que el manual y el start/stop estén activos
A2	Para que la cinta pueda funcionar no debe de haber un estado de error
B1	El arranque se utiliza como puente para poder arrancar con un estado de error y así poder desactivar el error

Tabla 6 Comentarios sobre el bloque Cinta Principal

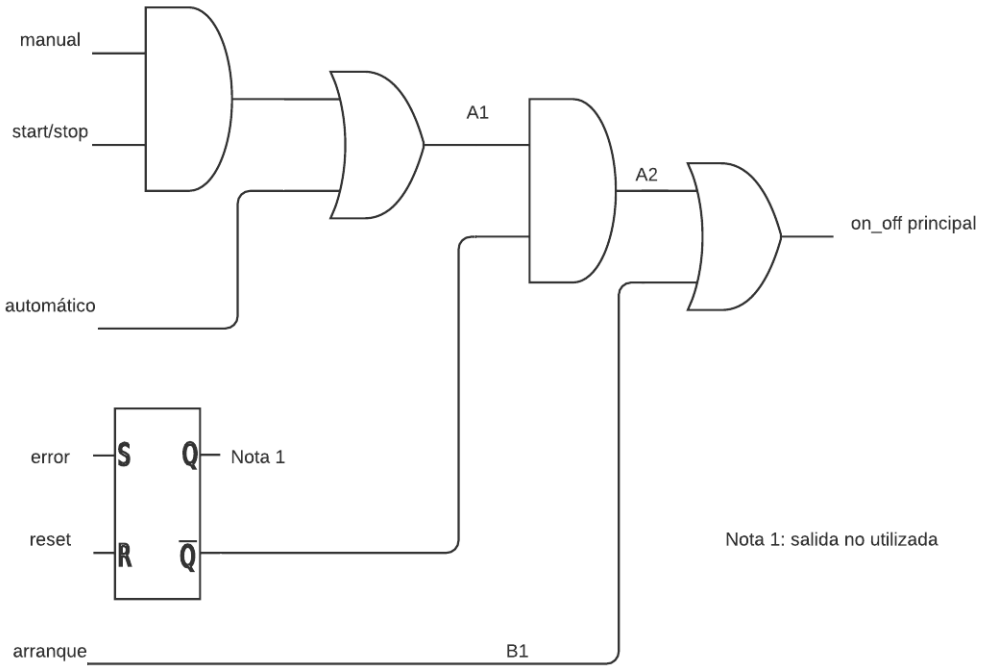


Figura 54 Diagrama lógico del bloque Cinta Principal

LÓGICA DEL BLOQUE CINTA 1

A1	Con el manual activo se puede encender y a apagar la cinta desde los botones del panel de control
A2	Solo se puede usar en modo manual con el modo automático en off
B1	Solo se puede usar en automático con el modo manual en off

Tabla 7 Comentarios sobre el bloque Cinta 1

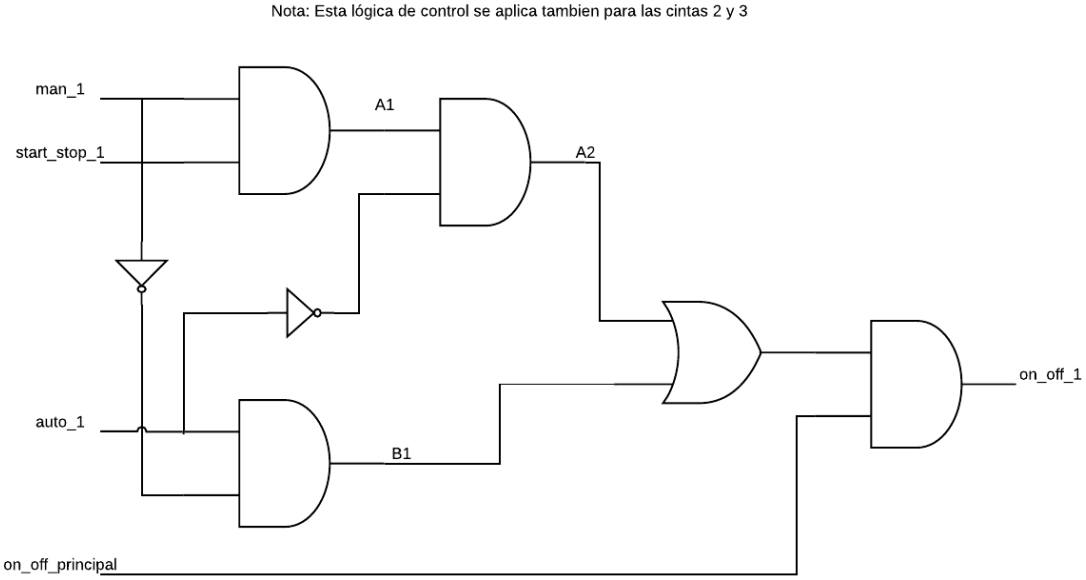
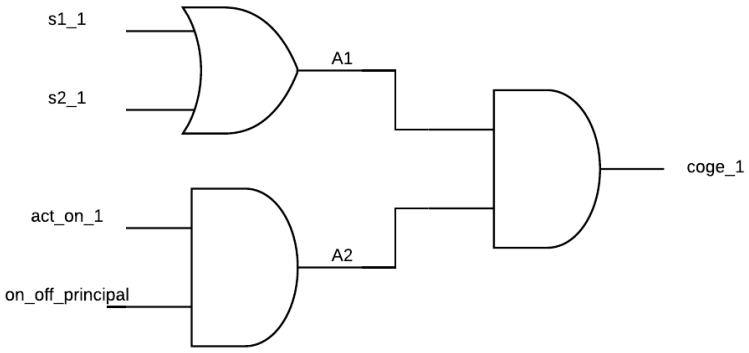


Figura 55 Diagrama lógico del bloque Cinta 1

LÓGICA DEL BLOQUE ACTUADOR 1

A1	Si uno o los dos sensores se activa, se ha detectado uno objeto para coger
A2	Si el actuador está encendido y la cinta principal está encendida se puede coger el objeto

Tabla 8 Comentarios sobre el bloque Actuador 1



Nota: esta lógica se aplica también para los actuadores 2 y 3

Figura 56 Diagrama lógico del bloque Actuador 1

LÓGICA DEL BLOQUE TEMPORIZADOR 1

A1	Para activar el temporizador se deben cumplir las condiciones de B1 y se debe seleccionar la correcta combinación de sel1_1 y sel2_1
A2	La señal de liberar estará activa durante 500 ms, cuando el tiempo de cuenta es mayor o igual al tiempo establecido en el comparador
B1	Para activar el temporizador la cinta sobre la que se deja el producto y el enable deben estar activos
B2	La señal para liberar se activa cuando uno de los tres temporizadores del bloque cumple con su condición

Tabla 9 Comentarios sobre el bloque Temporizador 1

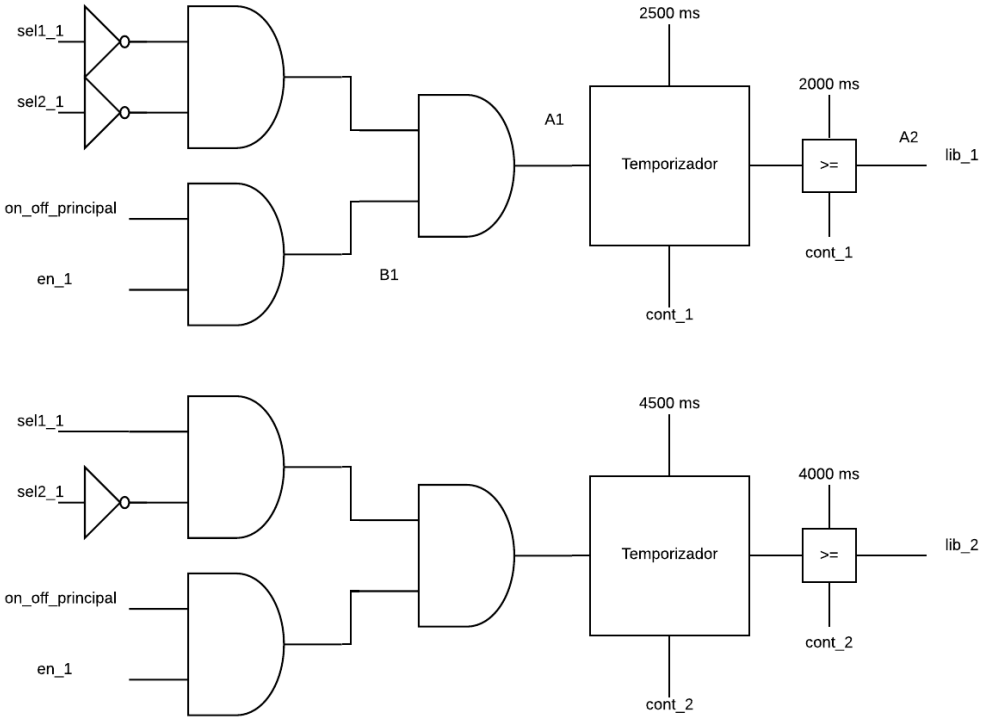


Figura 57 Diagrama lógico del bloque Temporizador 1, parte 1

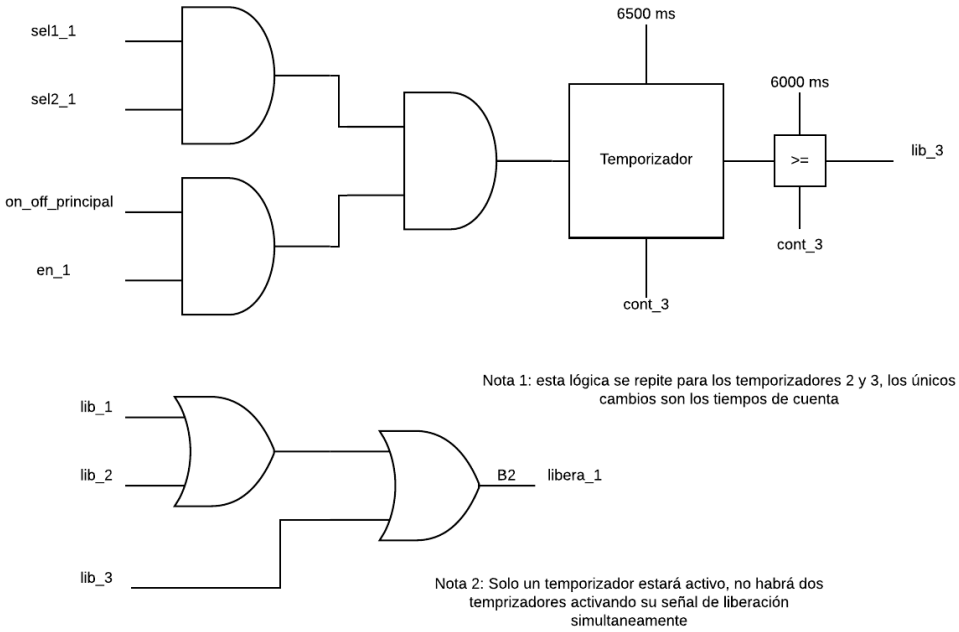


Figura 58 Diagrama lógico del bloque Temporizador 1, parte 2

ANEXO II: CONCEPTOS BÁSICOS DE TIA PORTAL

En este anexo se pretende explicar lo básico sobre TIA Portal y su entorno de desarrollo para una mejor comprensión del trabajo realizado.

PREPARACIÓN PARA SIMULAR EN PLCSIM

Para poder llevar a cabo una simulación en PLCsim es necesario marcar la opción *Permitir la simulación al compilar bloques* en la sección de *Protección* en las propiedades del proyecto.

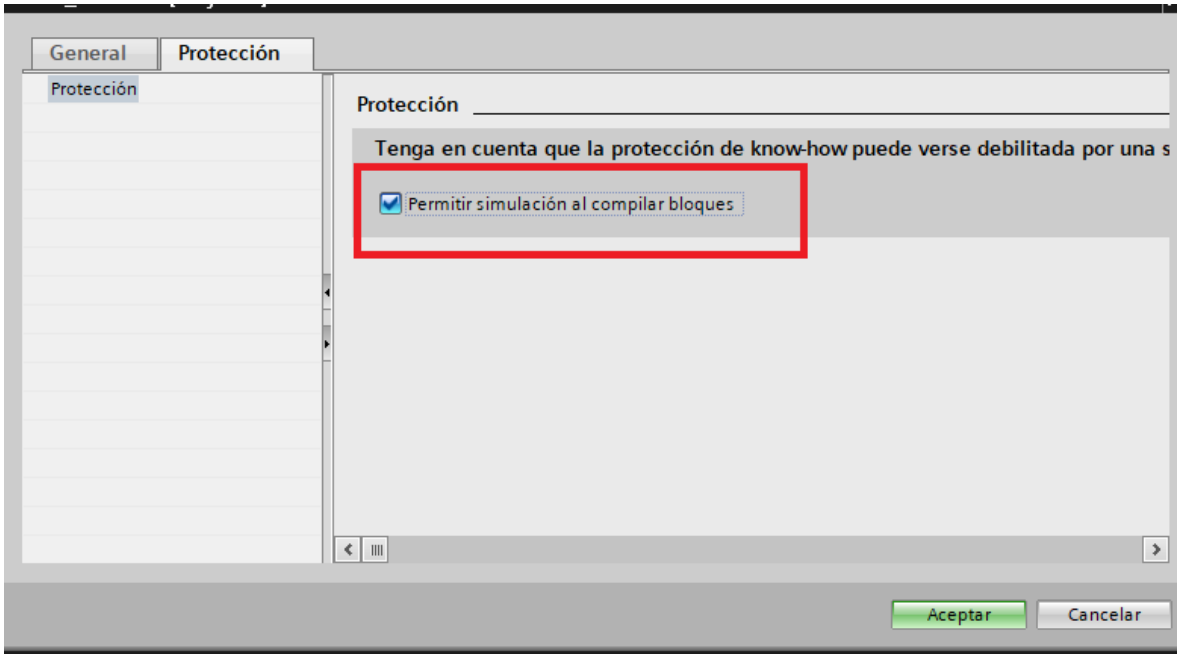


Figura 59 Configuración del proyecto para simular

BLOQUE DE ORGANIZACIÓN

Los bloques de organización (OB, por sus siglas en inglés) permiten estructurar la programación del autómata permitiendo separar el programa del autómata según distintos

comportamientos. Por ejemplo, a la hora de programar un autómata se pueden separar en distintos OB's los procesos de arranque y los procesos de control de motores.

Todo esto para tener una estructura más ordenada y que permita una mayor facilidad a la hora de comprender el funcionamiento de la programación.

Dentro de los OB's se pueden guardar otras estructuras de orden inferior como bloques de función o funciones.

BLOQUE DE FUNCIÓN

Son bloques lógicos con memoria, esto quiere decir que guardan de forma permanente los valores de las variables de las funciones que tienen definidas en bloques de datos propios. Dado que tienen memoria propia los valores de las variables son accesibles después de haber procesado el bloque.

FUNCIÓN

Las funciones se diferencian con los bloques de función en que no tienen una memoria propia, por lo tanto, las funciones no guardan los valores de las variables que utiliza.

Para poder acceder a dichas variables es necesario guardarlas en un bloque de datos externo a la función.

Las funciones pueden ser utilizadas para simplificar la estructura del programa, si se ve que una determinada lógica se está volviendo muy grande y difícil de seguir es recomendable dividirla en funciones y concatenar las salidas de unas funciones con las entradas de otras. Hacer esto no afecta al rendimiento del autómata y simplifica la visualización del programa.

BLOQUE DE DATOS

Permiten guardar variables (datos) del programa de forma permanente. Los bloques de datos son accesibles desde todos los bloques y son utilizados para guardar los valores de las variables de las funciones en el caso de que estos valores interesen que se guarden.

En este proyecto dentro del bloque de organización Main se ha dividido todo el programa en funciones y se han guardado todas las variables en un mismo bloque de datos para simplificar el acceso a las variables desde Unity.

HERRAMIENTAS DE DISEÑO

CONTACTO NORMALMENTE ABIERTO

Se le deben asignar variables booleanas. El funcionamiento de este elemento consiste en transmitir el estado de la variable a la que está asignado.

Si la variable tiene valor *true*, el contacto se cierra y si tiene valor *false* se abre.

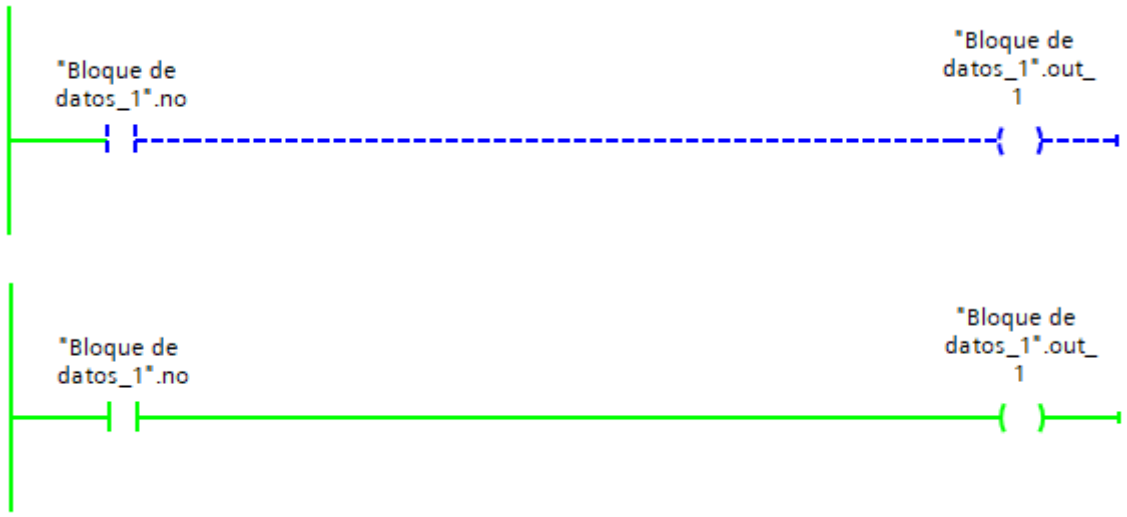


Figura 60 Funcionamiento de un contacto normalmente abierto

CONTACTO NORMALMENTE CERRADO

Su funcionamiento es al contrario que el contacto normalmente abierto, si la variable asignada tiene valor *true* se abre el contacto y si es *false* se cierra.

Este elemento resulta útil cuando se quiere utilizar el negado de una variable booleana.

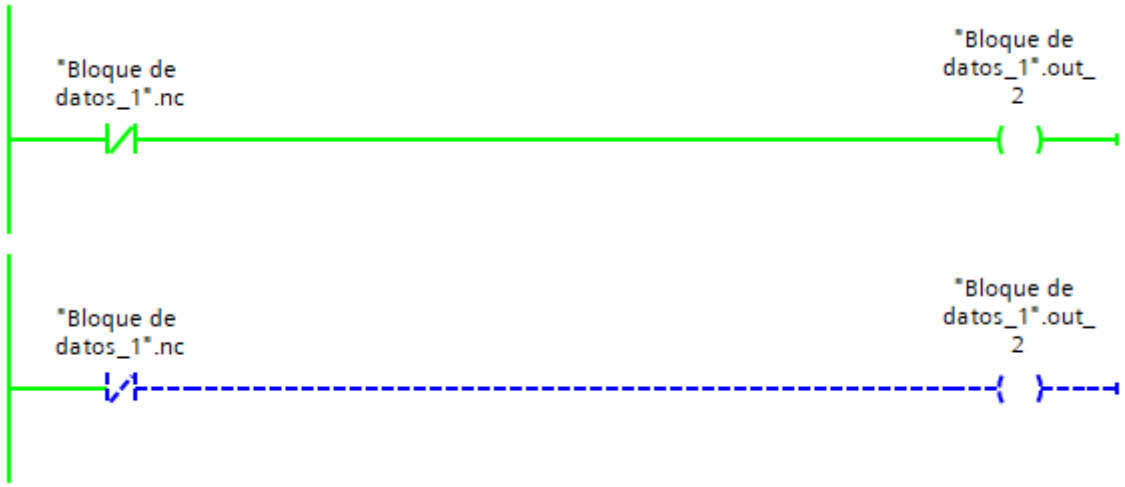


Figura 61 Funcionamiento de un contacto normalmente cerrado

ABRIR RAMA

Permite dividir una rama en dos y continuar operando por la nueva rama.

CERRAR RAMA

Permite unir dos ramas que se habían dividido anteriormente. Resulta muy útil a la hora de hacer la lógica de una puerta OR.

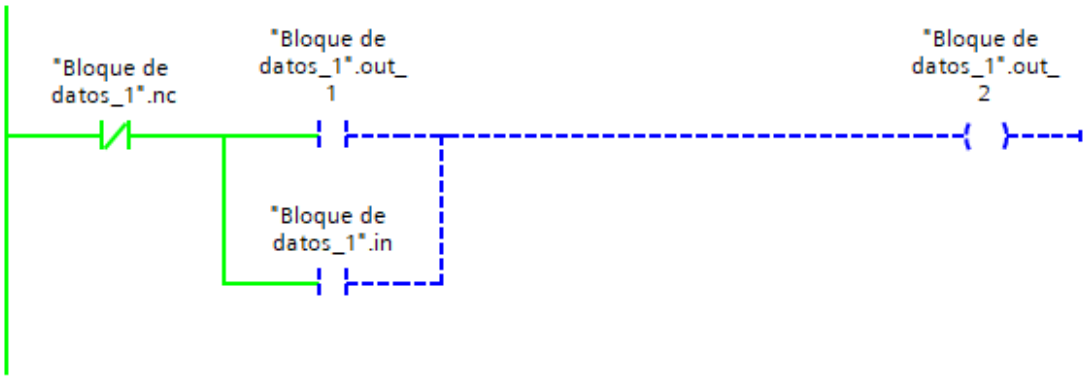


Figura 62 División de ramas en TIA Portal

ASIGNACIÓN

Permite dar valor a una variable, suele ser utilizado como la salida de una lógica.

En la Figura 62 División de ramas en TIA Portal el valor de *out_2* se da mediante una asignación.

ANEXO III: CONCEPTOS BÁSICOS DE UNITY

En este apartado se explican los conceptos básicos de Unity para comprender el funcionamiento de Unity y poder replicar el trabajo llevado a cabo en este proyecto [18] [19].

MONOBEHAVIOUR

MonoBehaviour es la clase base en la que se fundamenta el funcionamiento de los scripts que interactúan con el motor de Unity. Toda clase que se quiera utilizar para usar cualquier función de Unity debe ser hija de MonoBehaviour.

Todos los elementos explicados en este anexo pertenecen a la clase MonoBehaviour.

GAMEOBJECT

GameObject es la función base de todas las entidades de Unity, permite manipular y obtener la información de los elementos que se encuentran en la escena durante la simulación.

RIGIDBODY

Permite dar propiedades físicas a un gameobject (masa, aplicar fuerzas, rozamiento con el aire, centro de masa …).

VECTOR 3

Es la representación de un vector con tres componentes.

TRANSFORM

La propiedad *transform* permite conocer las transformaciones que se han aplicado y aplicar transformaciones a un gameobject. Las principales transformaciones usadas en el proyecto son las siguientes:

POSITION

Devuelve un vector3 que contiene las coordenadas en x, y, y z de un *gameobject*.

LOCALPOSITION

Posición de un *gameobject* relativa a su padre.

ROTATION

Devuelve un Quaternion que contiene la información de la rotación de un *gameobject*.

LOCALROTATION

Devuelve la rotación de un *gameobject* con respecto a su *gameobject* padre.

LOCALSCALE

Transformaciones en la escala de un *gameobject* relativas a las transformaciones del padre.

LOCALEULERANGLES

Devuelve los ángulos de rotación de Euler de un *gameobject* relativos a su padre.

PARENT

Permite acceder al *gameobject* que es el padre del gameobject al que pertenece el script.

ROTATEAROUND

Permite hacer rotar un *gameobject* con respecto a un eje en coordenadas no locales al propio *gameobject*.

ROTATE

Permite rotar un *gameobject* en torno a un eje del *gameobject* un cierto ángulo o a una velocidad determinada.

TAG

Devuelve la etiqueta que tiene asignada un *gameobject*.

NAME

Devuelve el nombre de un *gameobject*.

GETCOMPONENT

Permite acceder a cualquier tipo de componente asignada a un gameobject, se puede acceder a variables, funciones, propiedades de otro *gameobject* y acceder a scripts asociados a otro *gameobject*.

DESTROY

Permite destruir un *gameobject* y eliminarlo de la escena.

INSTANTIATE

Permite crear un clon de un *gameobject*.

RENDERER

Es lo que permite que un objeto se vea en pantalla. Permite acceder a propiedades como el color o la transparencia entre mucho otros.

START

Función que se ejecuta solo una vez durante la simulación, se ejecuta al inicio de la simulación y no vuelve a ejecutarse.

AWAKE

Se diferencia con *Start* en que *Awake* se ejecuta una vez que el objeto al que está asociado el script es instanciado.

UPDATE

Se ejecuta después de la función *Start* y se vuelve a ejecutar en cada frame.

FIXEDUPDATE

Se ejecuta cada cierto tiempo independientemente del framerate (tasa de cuadros por segundo). Por defecto se ejecuta 50 veces por segundo, por lo tanto, para aplicaciones que se ejecuten a más de 50 FPS (frames por segundo) es más lenta que la función *Update*.

COLLIDER

Permite crear una malla invisible entorno a un objeto que le da propiedades de un cuerpo rígido, por lo tanto, no puede ser atravesado.

TRIGGER

Permite que un *collider* pueda ser atravesado y además detecta cuando un objeto está atravesando el *collider*. Es muy útil para detectar cuando un objeto pasa por un determinado punto.

ANEXO IV: VARIABLES DE CONTROL

Nombre	Tipo de dato	Offset	Valor de arranque	Descripción
manual_principal	Bool	0.0	false	Entrada para elegir usar la cinta principal en modo manual (1) o en automático (0)
estado_error	Bool	0.1	false	Salida del flip-flop que controla el error
start_stop_principal	Bool	0.2	false	Entrada para arrancar (1) o parar (0) la cinta principal
error	Bool	0.3	false	Indica si se ha detectado algún error en el proceso
reset	Bool	0.4	false	Entrada para reiniciar el estado del error
auto_principal	Bool	0.5	true	Indica si la cinta principal opera en modo manual (0) o en automático (1)
on_off_principal	Bool	0.6	false	Indica si la cinta principal está en marcha (1) o no (0)
act_on_1	Bool	0.7	true	Indica si el actuador para el elemento 1 está encendido (1) o no (0)
act_on_2	Bool	1.0	true	Indica si el actuador para el elemento 2 está encendido (1) o no (0)
act_on_3	Bool	1.1	true	Indica si el actuador para el elemento 3 está encendido (1) o no (0)
s1_1	Bool	1.2	false	Indica si el sensor 1 para el elemento 1 detecta un objeto (1) o no (0)
s2_1	Bool	1.3	false	Indica si el sensor 2 para el elemento 1 detecta un objeto (1) o no (0)
s1_2	Bool	1.4	false	Indica si el sensor 1 para el elemento 2 detecta un objeto (1) o no (0)
s2_2	Bool	1.5	false	Indica si el sensor 2 para el elemento 2 detecta un objeto (1) o no (0)
s1_3	Bool	1.6	false	Indica si el sensor 1 para el elemento 3 detecta un objeto (1) o no (0)
s2_3	Bool	1.7	false	Indica si el sensor 2 para el elemento 3 detecta un objeto (1) o no (0)

coge_1	Bool	2.0	false	Si s1_1 o s2_1 están a 1 se recoge el producto de tipo de la cinta principal
coge_2	Bool	2.1	false	Si s1_2 o s2_2 están a 2 se recoge el producto de tipo de la cinta principal
coge_3	Bool	2.2	false	Si s1_3 o s2_3 están a 3 se recoge el producto de tipo de la cinta principal
on_off_1	Bool	2.3	false	Indica si la cinta 1 está activa (1) o no (0)
on_off_2	Bool	2.4	false	Indica si la cinta 2 está activa (1) o no (0)
on_off_3	Bool	2.5	false	Indica si la cinta 3 está activa (1) o no (0)
man_1	Bool	2.6	false	Indica si la cinta 1 está en modo manual (1) o automático (0)
man_2	Bool	2.7	false	Indica si la cinta 2 está en modo manual (1) o automático (0)
man_3	Bool	3.0	false	Indica si la cinta 3 está en modo manual (1) o automático (0)
start_stop_1	Bool	3.1	false	1 si la cinta 1 está en marcha o 0 si no
start_stop_2	Bool	3.2	false	1 si la cinta 2 está en marcha o 0 si no
start_stop_3	Bool	3.3	false	1 si la cinta 3 está en marcha o 0 si no
auto_1	Bool	3.4	true	Indica si la cinta 1 está en modo automático (1) o manual (0)
auto_2	Bool	3.5	true	Indica si la cinta 1 está en modo automático (1) o manual (0)
auto_3	Bool	3.6	true	Indica si la cinta 1 está en modo automático (1) o manual (0)
libera_1	Bool	3.7	false	1 si se pone un objeto en la cinta 1 y 0 cuando no
libera_2	Bool	4.0	false	1 si se pone un objeto en la cinta 2 y 0 cuando no
libera_3	Bool	4.1	false	1 si se pone un objeto en la cinta 3 y 0 cuando no
arranque	Bool	4.2	false	Entrada usada para arrancar la planta al inicio de la simulación: 1 activa (arrancar), 0 inactiva

sel1_1	Bool	4.3	false	Bit 1 del selector de velocidad 1
sel2_1	Bool	4.4	false	Bit 2 del selector de velocidad 1
sel1_2	Bool	4.5	false	Bit 1 del selector de velocidad 2
sel2_2	Bool	4.6	false	Bit 2 del selector de velocidad 2
sel1_3	Bool	4.7	false	Bit 1 del selector de velocidad 3
sel2_3	Bool	5.0	false	Bit 2 del selector de velocidad 3
en_1	Bool	5.1	false	Activador del temporizador 1
en_2	Bool	5.2	false	Activador del temporizador 2
en_3	Bool	5.3	false	Activador del temporizador 3

Tabla 10 Variables de control

ANEXO V: CÓDIGOS FUENTE

En este anexo se adjuntan los códigos utilizados para el desarrollo del proyecto. Cada código adjunto se corresponde a una clase determinada. Cada clase está contenida en un archivo .cs con el mismo nombre de la clase.

CÓDIGO ACTUADOR

```
using UnityEngine;

public class Actuador : MonoBehaviour
{
    [Header("Panel de Control")]
    [SerializeField] GameObject pdc;

    [Header("Brazo")]
    [SerializeField] GameObject brazo;

    [Header("Punto de agarre del brazo")]
    [SerializeField] GameObject cogedor;

    [Header("Objetivo")]
    [SerializeField] public string tipo;

    [Header("Estados")]
    public string estado = "Reposo";

    private bool[] estado_actuadores = new bool[3];
    private int actuador;
    private GameObject obj;

    private void Update()
    {
        //Se toman los estados de los actuadores de ColorPdC
        estado_actuadores[0] =
pdc.GetComponent<ColorPdC>().estado_actuador_1;
        estado_actuadores[1] =
pdc.GetComponent<ColorPdC>().estado_actuador_2;
        estado_actuadores[2] =
pdc.GetComponent<ColorPdC>().estado_actuador_3;

        //Se sekecciona el actuador en función del tipo de objeto que se
quiere comer
        if (tipo == "tipo 1")
            actuador = 0;
        if (tipo == "tipo 2")
```

```
            actuador = 1;
        if (tipo == "tipo 3")
            actuador = 2;

        //Máquina de estados
        switch (estado)
        {
            case "Reposo":
                break;
            case "Mover":
                //Se alinea con el objeto
                Mover(obj);
                break;
            case "Coger":
                //Coge el objeto de la cinta y lo mueve
                Coger(obj);
                break;
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        //Se comprueba si el esta actuador está encendido
        if (estado_actuadores[actuador] == true)
        {
            //Si la etiqueta del objeto coincide con la del objeto que
tiene asignado lo coge
            if (other.gameObject.CompareTag(tipo) == true && estado ==
"Reposo")
            {
                //Debug.LogError("Mover");
                obj = other.gameObject;
                estado = "Mover";
            }
        }
    }

    //Se mueve el brazo para alinearlo con el objeto en la cinta
    private void Mover(GameObject other)
    {
        if (cogedor.GetComponent<Cogedor>().agarrado == true)
            estado = "Coger";
    }

    //Coge el objeto de la cinta
    private void Coger(GameObject other)
    {
        if(other != null)
            other.gameObject.transform.position =
cogedor.gameObject.transform.position;

        if (brazo.GetComponent<Rotar>().movimiento_comletado == true)
            estado = "Reposo";
    }
}
```

CÓDIGO COLORPDC

```
using UnityEngine;

public class ColorPdC : MonoBehaviour
{
    [Header("Bombillas para indicar los estados")]
    public Renderer luz_Error;
    public Renderer luz_On_Off_prin;
    public Renderer luz_principal_auto;
    public Renderer luz_principal_manual;
    public Renderer luz_principal_stsp;
    public Renderer luz_1_manual;
    public Renderer luz_2_manual;
    public Renderer luz_3_manual;
    public Renderer luz_1_auto;
    public Renderer luz_2_auto;
    public Renderer luz_3_auto;
    public Renderer luz_1_onoff;
    public Renderer luz_2_onoff;
    public Renderer luz_3_onoff;
    public Renderer luz_1_stsp;
    public Renderer luz_2_stsp;
    public Renderer luz_3_stsp;
    public Renderer luz_act1;
    public Renderer luz_act2;
    public Renderer luz_act3;
    public Renderer vel_1;
    public Renderer vel_2;
    public Renderer vel_3;

    [Header("Estados controlados desde fuera de la clase")]
    public bool est_cinta_1;
    public bool est_cinta_2;
    public bool est_cinta_3;
    public bool est_error;
    public bool estado_actuador_1;
    public bool estado_actuador_2;
    public bool estado_actuador_3;

    //Cambia el valor de los selectores de velocidad del PLC
    // y cambia el color de los indicadores en la simulación
    public void Cambia_Vel(string nombre)
    {
        Comunicacion com = GetComponent<Comunicacion>();

        switch (nombre)
        {
            case "v1_1":
                com.setEstado(com.sel1_1, false);
                com.setEstado(com.sel2_1, false);
                Color_Vel(vel_1, 1);
```

```
                break;
            case "v2_1":
                com.setEstado(com.sel1_1, true);
                com.setEstado(com.sel2_1, false);
                Color_Vel(vel_1, 2);
                break;
            case "v3_1":
                com.setEstado(com.sel1_1, true);
                com.setEstado(com.sel2_1, true);
                Color_Vel(vel_1, 3);
                break;
            case "v1_2":
                com.setEstado(com.sel1_2, false);
                com.setEstado(com.sel2_2, false);
                Color_Vel(vel_2, 1);
                break;
            case "v2_2":
                com.setEstado(com.sel1_2, true);
                com.setEstado(com.sel2_2, false);
                Color_Vel(vel_2, 2);
                break;
            case "v3_2":
                com.setEstado(com.sel1_2, true);
                com.setEstado(com.sel2_2, true);
                Color_Vel(vel_2, 3);
                break;
            case "v1_3":
                com.setEstado(com.sel1_3, false);
                com.setEstado(com.sel2_3, false);
                Color_Vel(vel_3, 1);
                break;
            case "v2_3":
                com.setEstado(com.sel1_3, true);
                com.setEstado(com.sel2_3, false);
                Color_Vel(vel_3, 2);
                break;
            case "v3_3":
                com.setEstado(com.sel1_3, true);
                com.setEstado(com.sel2_3, true);
                Color_Vel(vel_3, 3);
                break;
            case null:
                break;
        }
    }

    //Cambia los colores de los indicadores del panel de control
    // en función del estado de sus variables correspondientes en el PLC
    public void Color_pdc()
    {
        Comunicacion com = GetComponent<Comunicacion>();
        Debug.Log("Cambia color");
        estado_actuador_1 = Cambiar_Color(luz_act1,
com.getEstado(com.act_off_1));
```

```
        estado_actuador_2 = Cambiar_Color(luz_act2,
com.getEstado(com.act_off_2));
        estado_actuador_3 = Cambiar_Color(luz_act3,
com.getEstado(com.act_off_3));
        est_error = Cambiar_Color(luz_Error, com.getEstado(com.error));
        Cambiar_Color(luz_On_Off_prin,
com.getEstado(com.on_off_principal));
        Cambiar_Color(luz_principal_manual,
com.getEstado(com.manual_principal));
        Cambiar_Color(luz_principal_auto,
com.getEstado(com.auto_principal));
        Cambiar_Color(luz_principal_stsp,
com.getEstado(com.start_stop_principal));
        Cambiar_Color(luz_1_auto, com.getEstado(com.auto_1));
        Cambiar_Color(luz_2_auto, com.getEstado(com.auto_2));
        Cambiar_Color(luz_3_auto, com.getEstado(com.auto_3));
        Cambiar_Color(luz_1_manual, com.getEstado(com.man_1));
        Cambiar_Color(luz_2_manual, com.getEstado(com.man_2));
        Cambiar_Color(luz_3_manual, com.getEstado(com.man_3));
        est_cinta_1 = Cambiar_Color(luz_1_onoff,
com.getEstado(com.on_off_1));
        est_cinta_2 = Cambiar_Color(luz_2_onoff,
com.getEstado(com.on_off_2));
        est_cinta_3 = Cambiar_Color(luz_3_onoff,
com.getEstado(com.on_off_3));
        Cambiar_Color(luz_1_stsp, com.getEstado(com.start_stop_1));
        Cambiar_Color(luz_2_stsp, com.getEstado(com.start_stop_2));
        Cambiar_Color(luz_3_stsp, com.getEstado(com.start_stop_3));
    }

    //Cambia el color de un indicador en función del estado que deba
representar
    // True: verde
    // False: rojo
    public bool Cambiar_Color(Renderer objeto, bool estado)
    {
        if (estado == true)
        {
            objeto.material.SetColor("_Color", Color.green);
            return true;
        }
        else
        {
            objeto.material.SetColor("_Color", Color.red);
            return false;
        }
    }

    //Cambia el color de los indicadores de velocidad
    //dependiendo de la velocidad elegida por el usuario
    // 1: azul
    // 2: amaraillo
    // 3: magenta
    public void Color_Vel(Renderer objeto, int vel)
    {
```

```
        if (vel == 1)
        {
            objeto.material.SetColor("_Color", Color.blue);
        }
        else
        {
            if (vel == 2)
            {
                objeto.material.SetColor("_Color", Color.yellow);
            }
            else
            {
                if (vel == 3)
                {
                    objeto.material.SetColor("_Color", Color.magenta);
                }
            }
        }
    }
}
```

CÓDIGO COMUNICACIÓN

```
using UnityEngine;
using Sharp7;

//Clase para almacenar las funciones que se van a usar para la
comunicación con el PLC
// solo una clase externa (hija o no) puede acceder a las funciones de
comunicación
public class Comunicacion : MonoBehaviour
{
    //Variables de control
    public Datos manual_principal = new Datos(0, 0, "manual_principal",
0);
    public Datos temp_out = new Datos(0, 1, "temp_out", 1);
    public Datos start_stop_principal = new Datos(0, 2,
"start_stop_principal", 2);
    public Datos error = new Datos(0, 3, "error", 3);
    public Datos reset = new Datos(0, 4, "reset", 4);
    public Datos auto_principal = new Datos(0, 5, "auto_principal", 5);
    public Datos on_off_principal = new Datos(0, 6, "on_off_principal",
6);
    public Datos act_off_1 = new Datos(0, 7, "act_off_1", 7);
    public Datos act_off_2 = new Datos(1, 0, "act_off_2", 8);
    public Datos act_off_3 = new Datos(1, 1, "act_off_3", 9);
    public Datos s1_1 = new Datos(1, 2, "s1_1", 10);
    public Datos s2_1 = new Datos(1, 3, "s2_1", 11);
    public Datos s1_2 = new Datos(1, 4, "s1_2", 12);
    public Datos s2_2 = new Datos(1, 5, "s2_2", 13);
    public Datos s1_3 = new Datos(1, 6, "s1_3", 14);
```

```
    public Datos s2_3 = new Datos(1, 7, "s2_3", 15);
    public Datos coge_1 = new Datos(2, 0, "coge_1", 16);
    public Datos coge_2 = new Datos(2, 1, "coge_2", 17);
    public Datos coge_3 = new Datos(2, 2, "coge_3", 18);
    public Datos on_off_1 = new Datos(2, 3, "on_off_1", 19);
    public Datos on_off_2 = new Datos(2, 4, "on_off_2", 20);
    public Datos on_off_3 = new Datos(2, 5, "on_off_3", 21);
    public Datos man_1 = new Datos(2, 6, "man_1", 22);
    public Datos man_2 = new Datos(2, 7, "man_2", 23);
    public Datos man_3 = new Datos(3, 0, "man_3", 24);
    public Datos start_stop_1 = new Datos(3, 1, "start_stop_1", 25);
    public Datos start_stop_2 = new Datos(3, 2, "start_stop_2", 26);
    public Datos start_stop_3 = new Datos(3, 3, "start_stop_3", 27);
    public Datos auto_1 = new Datos(3, 4, "auto_1", 28);
    public Datos auto_2 = new Datos(3, 5, "auto_2", 29);
    public Datos auto_3 = new Datos(3, 6, "auto_3", 30);
    public Datos libera_1 = new Datos(3, 7, "libera_1", 31);
    public Datos libera_2 = new Datos(4, 0, "libera_2", 32);
    public Datos libera_3 = new Datos(4, 1, "libera_3", 33);
    public Datos arranque = new Datos(4, 2, "arranque", 34);
    public Datos sel1_1 = new Datos(4, 3, "sel2_1", 35);
    public Datos sel2_1 = new Datos(4, 4, "sel1_2", 36);
    public Datos sel1_2 = new Datos(4, 5, "sel1_1", 37);
    public Datos sel2_2 = new Datos(4, 6, "sel2_2", 38);
    public Datos sel1_3 = new Datos(4, 7, "sel1_3", 39);
    public Datos sel2_3 = new Datos(5, 0, "sel2_3", 40);
    public Datos en_1 = new Datos(5, 1, "en_1", 41);
    public Datos en_2 = new Datos(5, 2, "en_2", 42);
    public Datos en_3 = new Datos(5, 3, "en_3", 43);

    // Se crea el cliente para la conexion
    public S7Client client = new S7Client();

    [Header("Parámetros del PLC")]
    [SerializeField] public string ip;
    [SerializeField] public int rack;   //0;
    [SerializeField] public int slot;   //1;

    //Variables auxiliares
    [Header("Estados de las variables")]
    //Vector para guardar el estado las variables de control
    public byte[] buffer = new byte[54];
    public int connect;
    //Indica al resto de los scrpits que se ha iniciado la simulación con
exito
    public bool iniciado;
    public bool reset_pulsado = true;
    public bool estado = false;
    public bool est_act_1 = true;
    public bool est_act_2 = true;
    public bool est_act_3 = true;

    //Detecta cuando se pulsa un botón y cambia el estado de la variable
    // correspondiente en el PLC
    public  bool Control_Botones(bool pulsa, string nombre)
```

```
{
    if (pulsa == true)
    {
        print(nombre);

        switch (nombre)
        {
            //Actuador 1
            case "act_off_1":
                if(getEstado(act_off_1) == true)
                {
                    setEstado(act_off_1, false);
                    est_act_1 = false;
                }
                else
                {
                    setEstado(act_off_1, true);
                    est_act_1 = true;
                }
                break;
            //Actuador 2
            case "act_off_2":
                if (getEstado(act_off_2) == true)
                {
                    setEstado(act_off_2, false);
                    est_act_2 = false;
                }
                else
                {
                    setEstado(act_off_2, true);
                    est_act_2 = true;
                }
                break;
            //Actuador 3
            case "act_off_3":
                if (getEstado(act_off_3) == true)
                {
                    setEstado(act_off_3, false);
                    est_act_3 = false;
                }
                else
                {
                    setEstado(act_off_3, true);
                    est_act_3 = true;
                }
                break;
            //Automático de la cinta principal
            case "auto_principal":
                if (getEstado(manual_principal) == true)
                {
                    setEstado(auto_principal, true);
                    setEstado(manual_principal, false);
                }
                break;
            //Manual de la cinta principal
```

```
            case "manual_principal":
                if (getEstado(auto_principal) == true)
                {
                    setEstado(manual_principal, true);
                    setEstado(auto_principal, false);
                }
                break;
            //Automático de la cinta 1
            case "auto_1":
                if (getEstado(man_1) == true)
                {
                    setEstado(auto_1, true);
                    setEstado(man_1, false);
                }
                break;
            //Manual de la cinta 1
            case "man_1":
                if (getEstado(auto_1) == true)
                {
                    setEstado(man_1, true);
                    setEstado(auto_1, false);
                }
                break;
            //Automático de la cinta 2
            case "auto_2":
                if (getEstado(man_2) == true)
                {
                    setEstado(auto_2, true);
                    setEstado(man_2, false);
                }
                break;
            //Manual de la cinta 2
            case "man_2":
                if (getEstado(auto_2) == true)
                {
                    setEstado(man_2, true);
                    setEstado(auto_2, false);
                }
                break;
            //Automático de la cinta 3
            case "auto_3":
                if (getEstado(man_3) == true)
                {
                    setEstado(auto_3, true);
                    setEstado(man_3, false);
                }
                break;
            //Manual de la cinta 3
            case "man_3":
                if (getEstado(auto_3) == true)
                {
                    setEstado(man_3, true);
                    setEstado(auto_3, false);
                }
                break;
```

```
//Start/Stop de la cinta principal
case "start_stop_principal":
    if (getEstado(start_stop_principal) == true)
    {
        setEstado(start_stop_principal, false);
    }
    else
    {
        setEstado(start_stop_principal, true);
    }
    break;
//Start/Stop de la cinta 1
case "start_stop_1":
    if (getEstado(start_stop_1) == true)
    {
        setEstado(start_stop_1, false);
    }
    else
    {
        setEstado(start_stop_1, true);
    }
    break;
//Start/Stop de la cinta 2
case "start_stop_2":
    if (getEstado(start_stop_2) == true)
    {
        setEstado(start_stop_2, false);
    }
    else
    {
        setEstado(start_stop_2, true);
    }
    break;
//Start/Stop de la cinta 3
case "start_stop_3":
    if (getEstado(start_stop_3) == true)
    {
        setEstado(start_stop_3, false);
    }
    else
    {
        setEstado(start_stop_3, true);
    }
    break;
//Botón de reset
case "reset":
    Arranque();
    setEstado(manual_principal, false);
    setEstado(man_1, false);
    setEstado(man_2, false);
    setEstado(man_3, false);
    set_enable();
    break;
case null:
    break;
```

```
            }
        }
        return estado = true;
    }

    //Función encargada de iniciar la comunicación con el PLC
    // cada vez que se arranca la simulación
    public bool Inico()
    {
        //Variable para acceder a las instancias publicas de ColorPdC
        ColorPdC colores = GetComponent<ColorPdC>();

        //Se conecta con el PLC
        connect = client.ConnectTo(ip, rack, slot);

        //Se comprueba si se ha podido conectar
        if (connect == 0)
        {
            Debug.Log("Conexion exitosa");
            //Se envia un pulso a los contadores
            set_enable();
            //Se realiza el proceso de arranque
            Arranque();
            //Se apaga la entrada de arranque
            setEstado(arranque, false);
            //Se incializan los colores de los indicadores
            colores.Color_pdc();
            Set_Colores_Vel();
            //Se indica que la comunicación con el PLC se ha realizado
con exito
            iniciado = true;
            return true;
        }
        else
        {
            //Si falla el inicio de la simulación se indica con un error
            Debug.LogError("Fallo de conexion");
            return false;
        }
    }

    //Inicializa los indicadores de velocidad
    // para indicar que están con velocidad 1
    private void Set_Colores_Vel()
    {
        ColorPdC color = GetComponent<ColorPdC>();

        color.Color_Vel(color.vel_3, 1);
        color.Color_Vel(color.vel_1, 1);
        color.Color_Vel(color.vel_2, 1);
    }

    //Envía un pulso a los temporizadores para que se activen
    public void set_enable()
    {
```

```
        setEstado(en_1, true);
        setEstado(en_2, true);
        setEstado(en_3, true);
        setEstado(en_1, false);
        setEstado(en_2, false);
        setEstado(en_3, false);
    }

    //Realiza el proceso de arranque
    public void Arranque()
    {
        //Se pone la variable arranque a true
        // esto permite que se desactive el estado
        // de error que hay al iniciar la simulación
        setEstado(arranque, true);
        //Se activan los automáticos de las cintas
        setEstado(auto_principal, true);
        setEstado(auto_1, true);
        setEstado(auto_2, true);
        setEstado(auto_3, true);
        //Se reinicia el flip-flop que indica el error
        // para que desaparezca la señal de error
        setEstado(reset, true);
        //Se encienden los actuadores
        setEstado(act_off_1, true);
        setEstado(act_off_2, true);
        setEstado(act_off_3, true);
        //Se comprueba si el proceso de arranque
        // ha tenido exito
        //Si sigue habeindo una señal de error el arranque ha fallado
        if (getEstado(error) == false)
        {
            //Se apaga la señal de arranque
            setEstado(arranque, false);
            //Se apaga el reset del flip-flop del error
            setEstado(reset, false);
            //Se indica que el arranque es correcto
            Debug.Log("Arranque correcto");
        }
        else
        {
            //Se indica si el arranque ha fallado
            Debug.Log("Fallo al arrancar");
        }
    }

    //Permite leer el estado de una variable del PLC
    public bool getEstado(Datos variable)
    {
        //Guarda los códigos de error del proceso
        // de lectura
        //Si vale 0 la lectura es correcta
        int lectura;
        //Guarda el estado tomado del PLC
        bool estado;
```

```
        lectura = client.DBRead(1, 0, buffer.Length, buffer);
        if (lectura == 0)
        {
            //Función que permite leer el valor de una variable
            estado = S7.GetBitAt(buffer, variable.pos, variable.bit);
            //Debug.Log(variable.nomb + ": " + estado);
            return estado;
        }
        else
        {
            //Se indica si falla la lectura
            Debug.LogError("Error de lectura");
            //Muestra el código de error en la consola
            Debug.LogError(lectura);
            return false;
        }
    }

    //Permite cambiar el valor de una variable del PLC
    public bool setEstado(Datos variable, bool estado)
    {
        //Guarda los códigos de error del proceso de escritura
        // Si vale 0 la escritura es correcta
        int escritura;
        //Función que permite cambiar el valor de una variable
        S7.SetBitAt(ref buffer, variable.pos, variable.bit, estado);
        escritura = client.DBWrite(1, 0, buffer.Length, buffer);
        if (escritura == 0)
        {
            //Debug.Log(variable.nomb + ": " + S7.GetBitAt(buffer,
variable.pos, variable.bit));
            return true;
        }
        else
        {
            //Se indica si falla la escritura
            Debug.LogError("Error de escritura");
            //Muestra el código de error en la consola
            Debug.LogError(escritura);
            return false;
        }
    }
}
```

CÓDIGO CREADOR

```
using UnityEngine;

public class Creador : MonoBehaviour
{
    //GameObject del que se quiere tener acceso a sus scrpits asociados
```

```
    [SerializeField] private GameObject Scripts;
    //Objeto que se quiere crear en pantalla durante la simulación
    [SerializeField] private Rigidbody obj;
    //Tipo del objeto a crear (1, 2 ó 3)
    [SerializeField] private int Tipo;

    void Update()
    {
        //Dependiendo del tipo de producto que sea se creará en una
determinada cinta
        if(Tipo == 1)
        {
            //Comprueba si la cinta 1 está encendida
            if (Scripts.GetComponent<Liberadores>().l_1 &&
Scripts.GetComponent<ColorPdC>().est_cinta_1 == true)
            {
                Rigidbody clon = (Rigidbody)Instantiate(obj,
transform.position, transform.rotation);
            }
        }
        else
        {
            if(Tipo == 2)
            {
                //Comprueba si la cinta 2 está encendida
                if (Scripts.GetComponent<Liberadores>().l_2 &&
Scripts.GetComponent<ColorPdC>().est_cinta_2 == true)
                {
                    Rigidbody clon = (Rigidbody)Instantiate(obj,
transform.position, transform.rotation);
                }
            }
            else
            {
                if(Tipo == 3)
                {
                    //Comprueba si la cinta 3 está encendida
                    if (Scripts.GetComponent<Liberadores>().l_3 &&
Scripts.GetComponent<ColorPdC>().est_cinta_3 == true)
                    {
                        Rigidbody clon = (Rigidbody)Instantiate(obj,
transform.position, transform.rotation);
                    }
                }
                else
                {
                    return;
                }
            }
        }
    }
}
```

CÓDIGO DATOS

```
//Clase para definir el tipo de objeto usado para asociarlo a las
// variables del PLC
public class Datos
{
    //Posición que guarda en el data block
    public int pos;
    //Bit asociado en el data block
    public int bit;
    //Nombre de la variable, es aconsejable que coincida
    // con el nombre que tiene en el PLC
    public string nomb;
    //Posición que guarda en el vector que almacena las variables
    public int pos_vec;

    //Consstructor de la clase
    public Datos(int p, int b, string nombre, int pos_v)
    {
        pos = p;
        bit = b;
        nomb = nombre;
        pos_vec = pos_v;
    }
}
```

CÓDIGO DEFECTUOSO

```
using UnityEngine;

//Se usa para crear objetos cdefetuosos
public class Defectuoso : MonoBehaviour
{
    [Header("Porcentaje de defectuosos")]
    [Range(0f, 100f)]
    [SerializeField] private float defectuosos;
    private void OnTriggerStay(Collider other)
    {
        //variable psudo aleatoria con distribucion uniforme
        //puede tomar valores entre 0 y 100
        float random = Random.Range(0f, 100f);
        //Si la varaible random es menor a la tasa de defectuosos
definida
        // el objeto se etiqueta como defectuoso
        if(random < defectuosos)
        {
            other.gameObject.tag = "Defectuoso";
        }
    }
```

```
}
```

CÓDIGO DESTRUCTOR

```
using UnityEngine;

//Destruye todos los objetos que entran en su collider y tienen
// las etiquetas indicadas
// La identificacion por etiqueta permite que no se destruya todo
// lo que toca el collider
public class Destructor : MonoBehaviour
{
    private void OnTriggerStay(Collider other)
    {
        if(other.gameObject.CompareTag("Defectuoso") == true ||
other.gameObject.CompareTag("tipo 1") == true ||
other.gameObject.CompareTag("tipo 2") == true ||
other.gameObject.CompareTag("tipo 3") == true)
            Destroy(other.gameObject);
    }
}
```

CÓDIGO DETECTOR_DEFECTOS

```
using UnityEngine;

public class Detector_Defectos : MonoBehaviour
{
    //Indica si el último objeto en pasar por el sensor era defectuoso
    [SerializeField] private Renderer luz_defecto;

    private void Start()
    {
        //Se incializa indicando que no ha pasado ningún defectuoso
        luz_defecto.material.SetColor("_Color", Color.red);
    }
    private void OnTriggerStay(Collider other)
    {
        if (other.gameObject.CompareTag("Defectuoso"))
        {
            //Se indica que el objeto es defectuoso
            luz_defecto.material.SetColor("_Color", Color.green);
        }
        else
        {
            luz_defecto.material.SetColor("_Color", Color.red);
        }
    }
}
```

CÓDIGO DETECTOR_TIPO

```
using UnityEngine;

public class Detector_Tipo : MonoBehaviour
{
    //Se indica que tipo de objeto se quiere reconocer (1, 2 ó 3)
    [SerializeField] string tipo;
    //Se usa para indicar si el último objeto en pasar era del tipo
correcto
    [SerializeField] Renderer luz;

    //Indican los estados de los sensores a otros Scrpits
    public bool sensor_tipo_1;
    public bool sensor_tipo_2;
    public bool sensor_tipo_3;

    private void Start()
    {
        //Se incicializan en rojo
        luz.material.SetColor("_Color", Color.red);
    }
    private void OnTriggerEnter(Collider other)
    {
        //Se compureba si la etiqueta del objeto coincide con el tipo
buscado
        if(other.gameObject.CompareTag(tipo) == true)
        {
            luz.material.SetColor("_Color", Color.green);
            //Se cambia el color de la luz de cada tipo
            if(tipo == "tipo 1")
            {
                sensor_tipo_1 = true;
            }

            if (tipo == "tipo 2")
            {
                sensor_tipo_2 = true;
            }

            if (tipo == "tipo 3")
            {
                sensor_tipo_3 = true;
            }

        }
        else
        {
            //Si no coincide se pone en rojo
            luz.material.SetColor("_Color", Color.red);
```

```
            if (tipo == "tipo 1")
            {
                sensor_tipo_1 =false;
            }

            if (tipo == "tipo 2")
            {
                sensor_tipo_2 = false;
            }

            if (tipo == "tipo 3")
            {
                sensor_tipo_3 = false;
            }
        }
    }
}
```

CÓDIGO ENABLES

```
using UnityEngine;

public class Enables : MonoBehaviour
{
    //Se usa para controlar la primera vez que se activan lo enbles
    private bool arrancado = false;
    //Estados anteriores y actuales de cada enable
    private bool estado_act_1;
    private bool estado_ant_1 = true;
    private bool estado_act_2;
    private bool estado_ant_2 = true;
    private bool estado_act_3;
    private bool estado_ant_3 = true;

    void Update()
    {
        ////Permite acceder a instancias de la clase Comunicacion
        Comunicacion com = GetComponent<Comunicacion>();
        ////Permite acceder a instancias de la clase Liberadores
        Liberadores liberador = GetComponent<Liberadores>();
        //Permite acceder a instancias de la clase PanelControl
        PanelControl pdc = GetComponent<PanelControl>();

        //Se incializan los enables
        if(pdc.iniciado == true && arrancado == false)
        {
            com.setEstado(com.en_1, true);
            com.setEstado(com.en_2, true);
            com.setEstado(com.en_3, true);
            com.setEstado(com.en_1, false);
            com.setEstado(com.en_2, false);
            com.setEstado(com.en_3, false);
```

```
            arrancado = true;
        }
        //Se guardan los estado actuales de cada enable
        estado_act_1 = liberador.estado_1;
        estado_act_2 = liberador.estado_2;
        estado_act_3 = liberador.estado_3;
        //Se comprueba si hay flancos de bajada
        if (estado_act_1 == false && estado_ant_1 == true)
        {
            com.setEstado(com.en_1, true);
            estado_ant_1 = estado_act_1;
            Debug.Log("1");
            com.setEstado(com.en_1, false);
        }
        else
        {
            estado_ant_1 = estado_act_1;
        }

        if (estado_act_2 == false && estado_ant_2 == true)
        {
            com.setEstado(com.en_2, true);
            estado_ant_2 = estado_act_2;
            Debug.Log("2");
            com.setEstado(com.en_2, false);
        }
        else
        {
            estado_ant_2 = estado_act_2;
        }

        if (estado_act_3 == false && estado_ant_3 == true)
        {
            com.setEstado(com.en_3, true);
            estado_ant_3 = estado_act_3;
            Debug.Log("3");
            com.setEstado(com.en_3, false);
        }
        else
        {
            estado_ant_3 = estado_act_3;
        }
    }
}
```

CÓDIGO GIRAR_CURVA

```
using UnityEngine;

public class Girar_Curva : MonoBehaviour
{
```

```
    //Objeto al rededor del que girarán los objetos al tomar la curva
    [SerializeField] public GameObject Centro_Rotacion;
    [Range(0, 1f)]
    //Variable para indicar la velocidad en la curva
    [SerializeField] private float Velocidad_Rotacion = 0.9f;

    private void OnTriggerEnter(Collider other)
    {

other.gameObject.transform.RotateAround(Centro_Rotacion.transform.positio
n, Vector3.up, Velocidad_Rotacion);
    }
    private void OnTriggerStay(Collider other)
    {

other.gameObject.transform.RotateAround(Centro_Rotacion.transform.positio
n, Vector3.up, -Velocidad_Rotacion);
    }
}
```

CÓDIGO LIBERADORES

```
using UnityEngine;

public class Liberadores : MonoBehaviour
{
    public bool l_1, l_2, l_3;
    private bool[] lib = new bool[3];
    private bool[] estado_act = new bool[3];
    private static bool[] estado_ant = { false, false, false };
    public bool activo;
    public bool estado_1, estado_2, estado_3;
    int cont = 0;

    void Update()
    {
        //Permite acceder a instancias de la clase Comunicacion
        Comunicacion com = GetComponent<Comunicacion>();

        int i;

        cont++;

        //En cada frame se comprueba un liberador porque si se comprueban
todos a la vez
        //baja mucho la tasa de fps
        switch (cont)
        {
            case 1:
                estado_act[0] = com.getEstado(com.libera_1);
                //cont++;
                break;
            case 2:
```

```
                estado_act[1] = com.getEstado(com.libera_2);
                //cont++;
                break;
            case 3:
                estado_act[2] = com.getEstado(com.libera_3);
                cont = 0;
                break;
        }

        //Se guardan por separado para poder acceder a ellos desde otro
script
        estado_1 = estado_act[0];
        estado_2 = estado_act[1];
        estado_3 = estado_act[2];

        //Se compureban los cambios de estado
        for (i = 0; i < lib.Length; i++)
        {
            if (estado_act[i] == true && estado_ant[i] == false)
            {
                estado_ant[i] = estado_act[i];
                lib[i] = true;
            }
            else
            {
                estado_ant[i] = estado_act[i];
                lib[i] = false;
            }
        }

        //Se indica el estado de cada liberador para que sea leído por
otro Script//Permite acceder a instancias de la clase PanelControl
        l_1 = lib[0];
        l_2 = lib[1];
        l_3 = lib[2];
    }
}
```

CÓDIGO MOUSE_LOOK

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Mouse_Look : MonoBehaviour
{
    [Header("Parámetros del ratón")]
    [SerializeField] private float mouseSensitivity = 100f;
    [SerializeField] private float up_max = 90f;
    [SerializeField] private float up_min = -90f;
    [Header("Bloquear ratón")]
```

```
    [SerializeField] private bool bloquear = true;
    private float xRotation = 0f;

    public Transform playerBody;

    void Start()
    {
        //Cursor.lockState = CursorLockMode.Locked;
    }

    void Update()
    {
        if (bloquear)
        {
            Cursor.lockState = CursorLockMode.Locked;
        }
        else
        {
            Cursor.lockState = CursorLockMode.None;
        }

        float mouseX = Input.GetAxis("Mouse
X")*mouseSensitivity*Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity *
Time.deltaTime;

        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, up_min, up_max);
        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        playerBody.Rotate(Vector3.up * mouseX);
    }
}
```

CÓDIGO MOVER

```
using UnityEngine;

public class Mover : MonoBehaviour
{
    //Puntos para definir la trayectoria de los objetos en la cinta
    [SerializeField] GameObject punto_1;
    [SerializeField] GameObject punto_2;
    //GameObject para ver si esta en estado de error
    [SerializeField] GameObject Color_Error;

    [Header("Velocidad de la cinta")]
    [Range(0f, 2f)]
    [SerializeField] private float velocidad = 0.3f;

    [Header("ID de la cinta")]
    [SerializeField] private int cinta_id;
```

```
    //vector que guarda la trayectoria de los objetos en la cinta
    private Vector3 direccion;

    void Start()
    {
        //Se define la trayectoria al iniciar la simulación
        float x = punto_1.transform.position.x -
punto_2.transform.position.x;
        float y = punto_1.transform.position.y -
punto_2.transform.position.y;
        float z = punto_1.transform.position.z -
punto_2.transform.position.z;
        direccion = new Vector3(x, y, z);
    }

    private void OnTriggerStay(Collider other)
    {
        bool error = Color_Error.GetComponent<ColorPdC>().est_error;
        bool estado_1 = Color_Error.GetComponent<ColorPdC>().est_cinta_1;
        bool estado_2 = Color_Error.GetComponent<ColorPdC>().est_cinta_2;
        bool estado_3 = Color_Error.GetComponent<ColorPdC>().est_cinta_3;

        //Si está en error las cintas no se mueven
        if (error == false)
        {
            if (cinta_id == 1 && estado_1 == true)
                other.gameObject.transform.position += direccion *
velocidad * Time.deltaTime;
            if (cinta_id == 2 && estado_2 == true)
                other.gameObject.transform.position += direccion *
velocidad * Time.deltaTime;
            if (cinta_id == 3 && estado_3 == true)
                other.gameObject.transform.position += direccion *
velocidad * Time.deltaTime;
            if (cinta_id == 0)
                other.gameObject.transform.position += direccion *
velocidad * Time.deltaTime;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        //Se da una fuerza cuando salen de la cinta para
        // no caigan sin velocidad
        float fuerza = 30 * velocidad;
        bool error = Color_Error.GetComponent<ColorPdC>().est_error;
        bool estado_1 = Color_Error.GetComponent<ColorPdC>().est_cinta_1;
        bool estado_2 = Color_Error.GetComponent<ColorPdC>().est_cinta_2;
        bool estado_3 = Color_Error.GetComponent<ColorPdC>().est_cinta_3;

        if (error == false)
        {
            if (cinta_id == 1 && estado_1 == true)
```

```
                other.attachedRigidbody.AddForce(direccion * fuerza,
ForceMode.Acceleration);
            if (cinta_id == 2 && estado_2 == true)
                other.attachedRigidbody.AddForce(direccion * fuerza,
ForceMode.Acceleration);
            if (cinta_id == 3 && estado_3 == true)
                other.attachedRigidbody.AddForce(direccion * fuerza,
ForceMode.Acceleration);
            if (cinta_id == 0)
                other.attachedRigidbody.AddForce(direccion * fuerza,
ForceMode.Acceleration);
        }
    }
}
```

CÓDIGO PANELCONTROL

```
using UnityEngine;
using UnityEngine.Profiling;

public class PanelControl : Comunicacion
{
    //Sensores de las cintas
    [SerializeField] GameObject sensor_1_1;
    [SerializeField] GameObject sensor_2_1;
    [SerializeField] GameObject sensor_1_2;
    [SerializeField] GameObject sensor_2_2;
    [SerializeField] GameObject sensor_1_3;
    [SerializeField] GameObject sensor_2_3;

    //Variables auxiliares
    public bool pulsa_reset = false;

    private static bool apagado11 = true;
    private static bool apagado21 = true;
    private static bool apagado12 = true;
    private static bool apagado22 = true;
    private static bool apagado13 = true;
    private static bool apagado23 = true;

    public bool sensor_tipo_1_1;
    public bool sensor_tipo_2_1;
    public bool sensor_tipo_1_2;
    public bool sensor_tipo_2_2;
    public bool sensor_tipo_1_3;
    public bool sensor_tipo_2_3;

    void Start()
    {
        //Se inicia la comunicación con el PLC
        Inico();
```

```
    }

    void Update()
    {
        Selector selector = GetComponent<Selector>();
        ColorPdC color = GetComponent<ColorPdC>();

        //Si se ha iniciado se ejecuta la simulación
        if (iniciado == true)
        {
            //Se comprueban los sensores
            Sensores();

            //Se comprueba si se pulsa el reset
            if (reset_pulsado == true)
            {
                pulsa_reset = false;
            }
            //Se detecta si se ha pulsado algún botón
            if (selector.pulsado)
            {
                estado = Control_Botones(selector.pulsado,
selector.nombre);
                color.Color_pdc();
                color.Cambia_Vel(selector.nombre);
                estado = true;
            }
        }
        else
        {
            Debug.LogError("Fallo de inicio");
        }
    }

    private void Sensores()
    {
        sensor_tipo_1_1 =
sensor_1_1.GetComponent<Detector_Tipo>().sensor_tipo_1;
        sensor_tipo_2_1 =
sensor_2_1.GetComponent<Detector_Tipo>().sensor_tipo_1;
        sensor_tipo_1_2 =
sensor_1_2.GetComponent<Detector_Tipo>().sensor_tipo_2;
        sensor_tipo_2_2 =
sensor_2_2.GetComponent<Detector_Tipo>().sensor_tipo_2;
        sensor_tipo_1_3 =
sensor_1_3.GetComponent<Detector_Tipo>().sensor_tipo_3;
        sensor_tipo_2_3 =
sensor_2_3.GetComponent<Detector_Tipo>().sensor_tipo_3;

        //Sensores del tipo 1
        if (sensor_tipo_1_1 == true)
        {
            if (apagado11 == true)
            {
                setEstado(s1_1, true);
```

```
            apagado11 = false;
        }
    }
    else
    {
        if (apagado11 == false)
        {
            setEstado(s1_1, false);
            apagado11 = true;
        }
    }

    if (sensor_tipo_2_1 == true)
    {
        if (apagado21 == true)
        {
            setEstado(s2_1, true);
            apagado21 = false;
        }
    }
    else
    {
        if (apagado21 == false)
        {
            setEstado(s2_1, false);
            apagado21 = true;
        }
    }

    //Sensores del tipo 2
    if (sensor_tipo_1_2 == true)
    {
        if (apagado12 == true)
        {
            setEstado(s1_2, true);
            apagado12 = false;
        }
    }
    else
    {
        if (apagado12 == false)
        {
            setEstado(s1_2, false);
            apagado12 = true;
        }
    }

    if (sensor_tipo_2_2 == true)
    {
        if (apagado22 == true)
        {
            setEstado(s2_2, true);
            apagado22 = false;
        }
    }
```

```
        else
        {
            if (apagado22 == false)
            {
                setEstado(s2_2, false);
                apagado22 = true;
            }
        }

        //Sensores del tipo 3
        if (sensor_tipo_1_3 == true)
        {
            if (apagado13 == true)
            {
                setEstado(s1_3, true);
                apagado13 = false;
            }
        }
        else
        {
            if (apagado13 == false)
            {
                setEstado(s1_3, false);
                apagado13 = true;
            }
        }

        if (sensor_tipo_2_3 == true)
        {
            //print("Dentro de 23");
            if (apagado23 == true)
            {
                //Debug.LogError("Enciende 23");
                setEstado(s2_3, true);
                apagado23 = false;
            }
        }
        else
        {
            if (apagado23 == false)
            {
                setEstado(s2_3, false);
                apagado23 = true;
            }
        }
    }
}
```

CÓDIGO PLAYERMOVEMENT

```
using UnityEngine;
```

```
public class PlayerMovement : MonoBehaviour
{

    public CharacterController controller;

    public float speed = 12f;
    public float gravity = -9.81f;
    Vector3 velocity;
    // Update is called once per frame
    void Update()
    {
        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");

        if (controller.isGrounded)
        {
            Vector3 move = transform.right * x + transform.forward * z;
            controller.Move(move * speed * Time.deltaTime);
        }
        else
        {
            velocity.y += gravity * Time.deltaTime;

            controller.Move(velocity * Time.deltaTime);
        }
    }
}
```

CÓDIGO SELECTOR

```
using UnityEngine;

public class Selector : MonoBehaviour
{
    //Material para resaltar los objetos seleccionables cuando
    // el cursor está sobre uno de ellos
    [SerializeField] private Material mat_resaltado;
    [SerializeField] private Material default_material;

    private Transform seleccion;

    //Etiqueta para comprobar si un objeto es seleccionable
    private string seleccionable = "seleccionable";
    //Nombre del objeto que se pulsa
    public string nombre;
    public bool pulsado = false;

    void Update()
    {
        PanelControl pdc = GetComponent<PanelControl>();

        if(pdc.estado == true)
```

```
        {
            pulsado = false;
        }

       if(seleccion!= null)
        {
            var selecRenderer = seleccion.GetComponent<Renderer>();
            selecRenderer.material = default_material;
            seleccion = null;
        }

       //Rayo que sale del centro de la cámara para seleccionar los
objetos
        var ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit))
        {
            var select = hit.transform;
            if (select.CompareTag(seleccionable))
            {
                seleccion = select;
                if (select.GetComponent<Renderer>() != null)
                {
                    select.GetComponent<Renderer>().material =
mat_resaltado;
                    if (Input.GetMouseButtonDown(0))
                    {
                        nombre = select.gameObject.name;
                        pulsado = true;
                    }
                }
            }
        }
    }
}
```

CÓDIGO EJES

```
using UnityEngine;

//Clase para definir los ejes de giro del brazo
public class Ejes : MonoBehaviour
{
    [Header("Eje de giro")]
    public Vector3 Axis;
    [Header("Parámetros del eje")]
    public float angulo_max = 90;
    public float angulo_min = 0;

    //Por defecto la velocidad de la articulacón vale 150
    private static float velocidad = 150;
```

```
    [Header("Velocidad actual")]
    public float vel = velocidad;

    public void v_mas()
    {
        //Velocidad en sentido positivo
        vel = velocidad;
    }

    public void v_menos()
    {
        //Velocidad en sentido negativo
        vel = -1 * velocidad;
    }

    public float Get_Angulo()
    {
        //Se coge el ángulo que está girado un eje
        if (this.Axis.x == 1)
        {
            return this.gameObject.transform.localEulerAngles.x;
        }
        else
        {
            if (this.Axis.y == 1)
            {
                return this.gameObject.transform.localEulerAngles.y;
            }
            else
            {
                if (this.Axis.z == 1)
                {
                    return this.gameObject.transform.localEulerAngles.z;
                }
                else
                {
                    return 0;
                }
            }
        }
    }


}
```

CÓDIGO ROTAR

```
using UnityEngine;
using Unity.Mathematics;
public class Rotar : MonoBehaviour
{
    [SerializeField] GameObject actuador;
```

```
    [Header("Parametros del brazo")]
    [SerializeField] private Ejes[] ejes;
    [SerializeField] private int num_ejes;
    [SerializeField] private int num_movimientos;
    [SerializeField] public float tolerancia;

    [Header("Comprobar movimiento")]
    public bool movimiento_comletado = false;

    [Header("Movimientos programados")]
    public float[] movimiento_1;
    public float[] movimiento_2;
    public float[] movimiento_3;
    public float[,] movimientos;

    [Header("Ángulos a girar")]
    public float[] angulo;

    private void Start()
    {
        angulo = new float[ejes.Length];
        movimientos = new float[num_movimientos, num_ejes];

        //Se rellena la matriz  de movimientos con los movimientos
predefinidos
        for (int k = 0; k < num_movimientos; k++)
        {
            for (int r = 0; r < num_ejes; r++)
            {
                if (k == 0)
                    movimientos[k, r] = movimiento_1[r];
                if (k == 1)
                    movimientos[k, r] = movimiento_2[r];
                if (k == 2)
                    movimientos[k, r] = movimiento_3[r];
            }
        }
    }

    private void Update()
    {
        //movimientos para cada estado del actuador
        if (actuador.GetComponent<Actuador>().estado == "Reposo")
            Movimiento(0);
        if (actuador.GetComponent<Actuador>().estado == "Mover")
            Movimiento(1);
        if (actuador.GetComponent<Actuador>().estado == "Coger" &&
Comprueba_Movimiento(2) == false)
        {
            Movimiento(2);
        }

    }

    //Se comprueba si un movimiento de ha completado
```

```
    private bool Comprueba_Movimiento(int mov)
    {
        int cont = 0;
        for(int i = 0; i < num_ejes; i++)
        {
            if (movimientos[mov, i] < 0)
                movimientos[mov, i] = movimientos[mov, i] + 360;

            if (math.abs(movimientos[mov, i] - ejes[i].Get_Angulo()) <
tolerancia)
                cont++;
        }

        //Se comprueba si todos los ejes están es su sitio
        if(cont == 5)
        {
            movimiento_comletado = true;
            return true;
        }
        else
        {
            movimiento_comletado = false;
            return false;
        }
    }

    //Recibe que movimiento tiene que hacer el brazo
    private void Movimiento(int movimiento)
    {
        float[] giro = new float[num_ejes];

        for (int t = 0; t < giro.Length; t++)
        {
            giro[t] = movimientos[movimiento, t];
        }

        Rota_Angulo(giro);
    }

    //Mueve cada eje a un ángulo predefinido
    private void Rota()
    {
        for (int i = 0; i < num_ejes; i++)
        {
            if (ejes[i].Get_Angulo() > 180)
                angulo[i] = ejes[i].Get_Angulo() - 360;
            else
                angulo[i] = ejes[i].Get_Angulo();

            if (ejes[i].angulo_min == ejes[i].angulo_max)
            {
                ejes[i].v_mas();
            }
            else
            {
```

```
                if (angulo[i] <= ejes[i].angulo_min)
                    ejes[i].v_mas();
                if (angulo[i] >= ejes[i].angulo_max)
                    ejes[i].v_menos();
            }

            ejes[i].transform.Rotate(ejes[i].Axis, ejes[i].vel *
Time.deltaTime);
        }
    }

    //Mueve el brazo a un conjunto de ángulos que recibe
    private void Rota_Angulo(float[] giros)
    {
        for (int i = 0; i < num_ejes; i++)
        {
            angulo[i] = ejes[i].Get_Angulo();

            if (giros[i] < 0)
            {
                giros[i] = giros[i] + 360;
            }

            if (math.abs(angulo[i] - giros[i]) > tolerancia)
            {
                if (giros[i] > angulo[i])
                {
                    if (math.abs(angulo[i] - giros[i]) > 180)
                    {
                        ejes[i].v_menos();
                    }
                    else
                    {
                        ejes[i].v_mas();
                    }
                }
                if (giros[i] < angulo[i])
                {
                    if (math.abs(angulo[i] - giros[i]) > 180)
                    {
                        ejes[i].v_mas();
                    }
                    else
                    {
                        ejes[i].v_menos();
                    }
                }
                ejes[i].transform.Rotate(ejes[i].Axis, ejes[i].vel *
Time.deltaTime);
            }
        }
    }
}
```

CÓDIGO COGEDOR

```
using UnityEngine;

public class Cogedor : MonoBehaviour
{
    [SerializeField] GameObject actuador;
    [SerializeField] GameObject cogedor;

    private bool agarrar = false;
    public bool agarrado = false;

    private void Update()
    {
        if (actuador.GetComponent<Actuador>().estado == "Mover")
        {
            agarrar = true;
        }
        else
        {
            agarrar = false;
            agarrado = false;
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (agarrar == true)
        {
            if
(other.CompareTag((actuador.GetComponent<Actuador>().tipo)) == true)
                agarrado = true;
        }
    }
}
```

ANEXO VI: OBJETIVOS DE DESARROLLO SOSTENIBLE

Figura 63 Objetivos de Desarrollo Sostenible

El objetivo de los ODS (Objetivos de Desarrollo Sostenible) es la proteccion del planeta y asegurar los derechos y calidad de vida para toda la población mundial.

Dado que en este proyecto se busca el desarrollo de una tecnología la cual facilite la formación de los trabajadores y por lo tanto mejore su seguridad en el trabajo, este proyecto guarda relacion con el putno número 9: Industria, innovación e infraestructura.