Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

GEMELO DIGITAL DE LA ESTACIÓN 4 DE CÉLULA DE FABRICACIÓN FLEXIBLE FMS200

Alumno: María Álvarez Perales

Tutor: Prof. Dª. Elisabet Estévez Estévez
Prof. D. Alejandro Sánchez García
Dpto: Ing. Electrónica y Automática

Noviembre, 2021

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Electrónica y Automática

Doña Elisabet Estévez Estévez, tutora del Proyecto Fin de Carrera titulado: Gemelo Digital de la Estación 4 de Célula de Fabricación Flexible FMS200, que presenta María Álvarez Perales, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, NOVIEMBRE de 2021

El alumno:

María Álvarez Perales

Los tutores:

ESTEVEZ ESTEVEZ ELISABET - 72578245C

Firmado digitalmente por ESTEVEZ ESTEVEZ ELISABET - 72578245C
Fecha: 2021.11.19 09:13:55 +01'00'

Elisabet Estévez Estévez

SANCHEZ GARCIA ALEJANDRO - 77321466C

Firmado digitalmente por SANCHEZ GARCIA ALEJANDRO - 77321466C
Fecha: 2021.11.19 09:35:16 +01'00'

Alejandro Sánchez García

RESUMEN

En este proyecto se ha desarrollado un gemelo digital de la cuarta estación de la Célula de Fabricación Flexible FMS200.

En primer lugar, se han modelado los elementos de la estación haciendo uso de un software de diseño asistido. Este modelo virtual, se ha exportado y representado, utilizando el lenguaje Processing, en un espacio de trabajo, que simula a la perfección la estación que se pretende clonar.

En segundo lugar, se ha dotado al modelo representado de movimiendo, de forma que simula todas y cada una de las acciones de la estación.

Por último, se ha establecido una comunicación entre el modelo y un PLC virtual de Siemens, consiguiendo así controlar el gemelo modelado a través de la programación del autómata.

Como resultado, se obtiene una representación de la estación FMS-204 totalmente funcional que refleja el funcionamiento de la estación real, siendo esta la causa por la que se le conoce por el nombre de Gemelo Digital.

ABSTRACT

In this project, a digital twin of the fourth station of the FMS200 Flexible Manufacturing Cell has been developed.

First of all, the elements of the station have been modeled using assisted design software. This virtual model has been exported and represented, using Processing language, in a workspace that perfectly simulates the station to be cloned.

Secondly, the represented model has been provided with movement, so that it simulates each and every action of the station.

Finally, a communication has been established between the model and a Siemens virtual PLC, thus managing to control the modeled twin through the programming of the automaton.

As a result, a fully functional model of the FMS-204 station is obtained, which reflects the operation of the real station, being this the reason why this model is known by the name of digital twin.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Motivación

En un contexto en el que la presencialidad en las aulas, empresas e incluso centros de salud y hospitales se convierte en un lujo, nace la motivación de este proyecto.

La crisis provocada por la Covid-19 ha generado un gran impacto en todos los aspectos. En el ámbito tecnológico, ha supuesto un punto de partida para el desarrollo de disciplinas que se han convertido casi en una necesidad.

Algunas de estas disciplinas, han surgido durante el transcurso de esta nueva situación que nos ha tocado vivir. Un ejemplo claro de esto ha sido la adaptación de la docencia, de un formato totalmente presencial, a otro totalmente virtual.

Algunas otras, ya habían salido a la luz con anterioridad, pero sin lugar a dudas, esta crisis ha fomentado y acelerado su desarrollo. Un ejemplo de esto es lo que se conoce como laboratorios virtuales, muy útiles y ahora necesarios en la educación. Otro ejemplo clave son las llamadas fábricas digitales.

Los laboratorios constituyen una de las piezas más importantes en instituciones como universidades e institutos. Por una parte, es una herramienta necesaria para el estudiantado, sobre todo en el estudio de las disciplinas incluidas en lo que se conoce como STEM (acrónimo de las siglas en inglés de Ciencia (Science), Tecnología (Technology), Ingeniería (Engineering) y Matemáticas (Mathematics)) [1] y, por otra parte, una herramienta vital en el ámbito del desarrollo e investigación. Sin embargo, equipar un laboratorio con todo lo necesario para desarrollar estas actividades supone un coste difícil de afrontar, más aún en el caso de la enseñanza, ya que son muchos los alumnos que tienen que hacer uso de las instalaciones. Es por esto por lo que, en muchas ocasiones, los laboratorios constituyen un recurso escaso que termina generando cuellos de botella y, sobre todo, generando un alto costo de oportunidad, ante la imposibilidad de realizar todos los estudios que los estudiantes y los investigadores proponen. Ante esta necesidad surgen los laboratorios virtuales [2], una tecnología que permite reducir costos y ampliar, de manera ilimitada, la capacidad de los laboratorios.

Por su parte, las fábricas digitales [3] también han sido una herramienta que ha jugado un papel clave en los últimos años. Este sistema, es un gran ejemplo del resultado de la actual revolución industrial, conocida como Industria 4.0. Una fábrica digital o virtual es un entorno informático integrado para el diseño, simulación y optimización de una fábrica completa, incluyendo sus líneas de producción y sus procesos. Un gran ejemplo de fábrica digital es La Fábrica Digital para la Industria de Procesos de Siemens [4]. La empresa alemana presenta esta solución como una verdadera revolución que permitirá a los usuarios integrar la automatización, el software y la tecnología de vanguardia para conseguir grandes ventajas como una producción más flexible y una mayor productividad. Además, presenta el concepto de gemelo digital como una representación virtual de un sistema real, que permite representar y optimizar todo el ciclo de vida de una planta industrial.

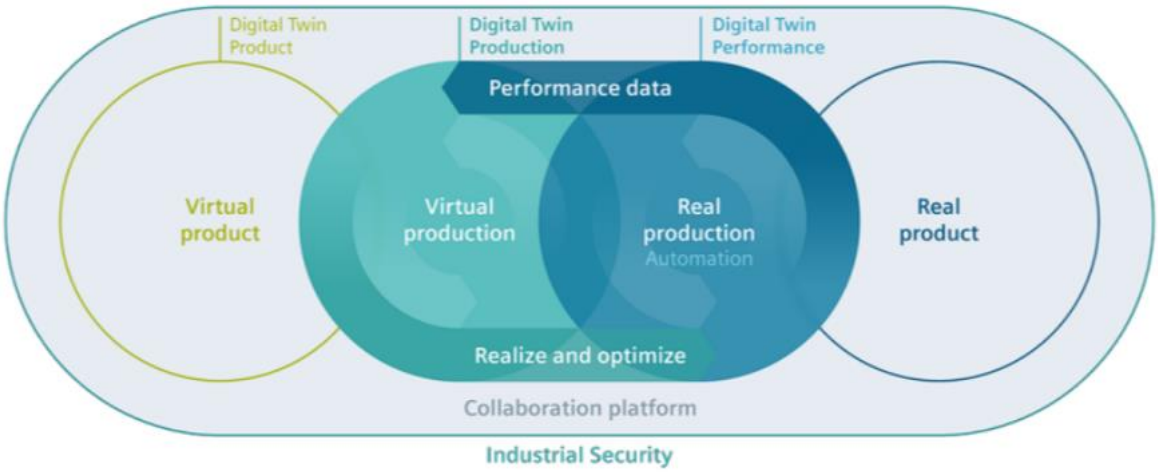


Figura 1 Gemelo digital presentado por Siemens

Todo esto ha servido de inspiración para realizar el presente proyecto, que consiste en el diseño de un gemelo digital de una de las estaciones que integran la Célula de Fabricación Flexible FMS200 que sirve como modelo de automatización de procesos industriales en el aprendizaje de un gran número de universidades de todo el mundo.

1.2. Objetivos

El objetivo principal de este proyecto es modelar, en un entorno virtual la estación cuarta del conjunto completo de la Célula de Fabricación Flexible FMS200 para desarrollar así un gemelo virtual de la misma.

Con esto, se persigue brindar al alumnado de la asignatura de Automática Avanzada una herramienta de apoyo con la que practicar y reforzar los conocimientos adquiridos sin necesidad de visitar de forma presencial el laboratorio. Esto permite una mayor disponibilidad horaria para practicar y/o realizar prácticas, solventando muchos de los inconvenientes anteriormente mencionados. Además, esta solución ofrece una captación profesional más acorde con la realidad en cuanto al desarrollo de la Industria 4.0 y la digitalización de los procesos que se encuentra actualmente en auge.

Por otra parte, se pretende que este gemelo digital sirva, de cara a la industria, no solo para tareas de ingeniería propiamente dichas, como pueden ser las simulaciones de planta antes de la puesta en marcha, sino también en procesos posteriores de la fabricación industrial, manteniendo informados a los operarios del estado de la planta en todo momento mientras se encuentra en funcionamiento, o facilitando las tareas de mantenimiento.

1.3. Estructura

Este documento se ha elaborado con la finalidad de recoger todos los conocimientos adquiridos durante la elaboración del proyecto, así como para servir de guía para futuras mejoras y para que los estudiantes que pretendan usar este gemelo puedan recurrir a este informe y obtener información sobre cómo utilizarlo y conocer las posibilidades que ofrece. Para conseguirlo, se ha separado la información en distintos epígrafes que se explican a continuación.

El capítulo titulado *SOFTWARE* se ha dedicado a describir todos y cada uno de los software que han sido necesarios para el desarrollo del gemelo digital, explicando las características más significativas de cada uno de ellos y las

posibilidades que ofrecen. Estas descripciones se han realizado de manera superficial, por lo que se recomienda al lector que consulte las referencias que se han ido introduciendo a lo largo de este informe para profundizar, si así lo desea, en los aspectos tratados.

A continuación, en el epígrafe *CÉLULA DE FABRICACIÓN FLEXIBLE,* se realiza una descripción, en primer lugar, de la Célula de Fabricación Flexible FMS200 completa y, posteriormente, en el subcapítulo *Estación FMS-204*, como su propio nombre indica, se detallan las características de la estación cuarta, que será la protagonista en este proyecto.

Seguidamente, en el epígrafe *GEMELO DIGITAL,* se ha tratado de explicar cómo se ha organizado el código fuente responsable del funcionamiento del gemelo, con la finalidad de poder reproducir este trabajo si alguien así lo quiere.

El siguiente capítulo, *GUÍA PARA EL USUARIO,* está dirigido a los estudiantes que pretendan utilizar esta herramienta en el aprendizaje de la automatización. Se pretende que sirva como guía de uso, por lo que se explican los procedimientos a seguir para ejecutar el programa, así como las características que este ofrece.

Por último, se incluyen dos anexos. El primero de ellos contiene los planos eléctricos de la estación, facilitados por el fabricante, y que son muy útiles y necesarios para comprender el cableado de la estación y poder identificar las entradas y salidas del PLC. El segundo anexo contiene el informe generado por NetBeans en el que se detalla la funcionalidad de cada clase y método que forman parte del código fuente del proyecto.

2. SOFTWARE

En este capítulo se pretende describir, de forma general, cada uno de los softwares empleados en el desarrollo de este proyecto.

Antes de profundizar en cada uno de ellos, es importante tener una visión del conjunto en general y del papel de cada software en particular.

El IDE utilizado para la programación ha sido NetBeans, y en él se han importado dos librerías, Moka7, que pertenece a Snap7, y Processing, cuyo lenguaje se ha utilizado para la programación del gemelo digital. Por su parte, Moka7 junto a NetToPLCsim permiten realizar la comunicación entre el Gemelo Digital y TIA Portal. Para conseguir que el gemelo sea completamente virtual, en lugar de conectar un PLC real, se conecta un PLC virtual, esto gracias a S7-PLCSIM. En la Figura 2 aparece un esquema de esto.

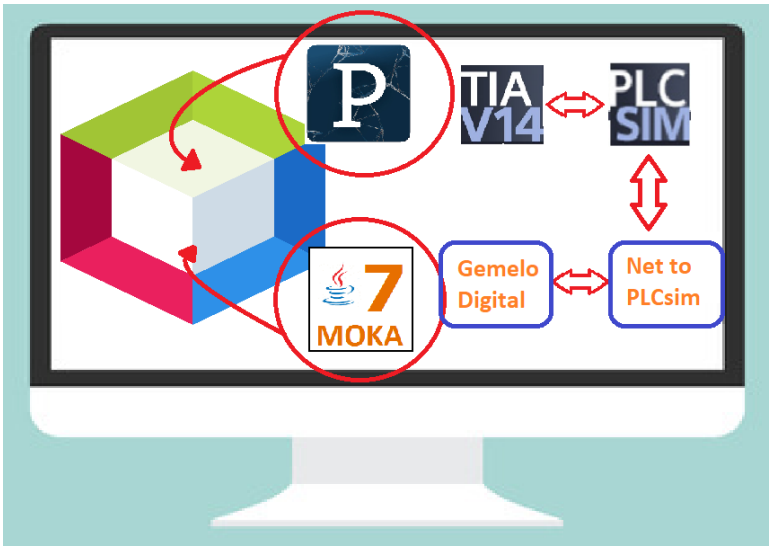


Figura 2 Relación entre los softwares empleados

2.1. NetBeans

NetBeans es lo que se conoce como entorno de desarrollo integrado o entorno de desarrollo interactivo, más conocido por sus siglas en inglés; IDE (Integrated Development Environment) [5]. Estos entornos facilitan en gran medida el Desarrollo de Aplicaciones Web (DAW), el Desarrollo de Aplicaciones Multiplataforma (DAM), desarrollo de juegos etc. ya que ofrecen herramientas como compiladores,

depuradores o bibliotecas. Además, todos los IDE deben cumplir una serie de características, las cuales garantizarán que la experiencia del usuario sea satisfactoria. Todo IDE debe contar con las siguientes herramientas:

- **Editor de código**. Un editor de texto en el que se trabaja con el código fuente de programas informáticos.
- **Compilador**. Un programa informático que traduce las instrucciones escritas en un lenguaje de programación, a código objeto, es decir, un lenguaje que el ordenador es capaz de procesar.
- **Depurador**. También conocido como debugger, es un programa que examina el código fuente de otro programa con la finalidad de probar y depurar los errores que el programador pudiera haber cometido.
- **Linker**. Herramienta con la que se combinan diferentes archivos de código fuente y se convierten en un único fichero ejecutable.
- **Refactorización de código**. Proceso en el que se recurre a funciones como el reformateo o la encapsulación para mejorar el código fuente.

Apache NetBeans [6] (Figura 3) es uno de los IDE más utilizados actualmente. Fue lanzado en el año 2000 por Sun MicroSystems. En 2010, el software NetBeans pasó a ser propiedad de la compañía Oracle Corporation, hasta 2018, cuando Oracle donó el software a la fundación Apache. Fue entonces cuando pasó de llamarse NetBeans, a llamarse Apache NetBeans, nombre que todavía conserva en la actualidad.

Figura 3 Apache NetBeans IDE

Al ser un desarrollador de código abierto y gratuito se convierte en una herramienta muy accesible. Proporciona editores, asistentes y plantillas para crear aplicaciones en Java, PHP, JavaScript, HTML5, CSS y varios lenguajes más.

Además, puede ejecutarse en todos los sistemas operativos que admitan Java, es decir, Windows, Linux, Mac OSX y BSD. Entre sus características más destacables aparece la posibilidad de programar en Framework de Java Swing; esto facilita el desarrollo de aplicaciones en entornos gráficos haciendo uso de interfaces gráficas.

NetBeans ofrece a sus usuarios algunas funciones y características que lo convierten en uno de los IDE más atractivos de la actualidad. Entre estas características encontramos la gestión de la interfaz de usuario y ventana, que permite al programador acomodar la interfaz como desee, añadiendo o eliminando menús, barras de herramientas, tipografías, etc.) Gestión de almacenamiento, ofreciendo la posibilidad de guardar o cargar datos. Marco asistente, que brinda al programador la opción de recibir soporte. Librerías visuales y herramientas de desarrollo integrado, entre otras.

2.2. Processing

Processing [7] es un lenguaje de programación y entorno de desarrollo integrado (IDE) [5] de código abierto [8]. Se dice que es un dialecto de Java, ya que está basado en este lenguaje, lo que lo convierte en una herramienta multiplataforma. Fue desarrollado por el departamento Media Lab del MIT (Massachusetts Institute of Technology) y lanzado en 2001. Sus creadores fueron Casey Res y Ben Fry.

Figura 4 Processing

Processing es un lenguaje sencillo y más asequible que la mayoría de los entornos gráficos, pero igualmente potente. Una de las ventajas que más destaca frente a sus competidores es la escalabilidad. Es posible combinar Processing con aplicaciones Java. Además, ofrece la posibilidad de portar los proyectos a la web gracias a Processing.js [9]

Existen tres niveles para programar en esta plataforma:

- El nivel más básico, tipo Ensamblador. Se programa sentencia a sentencia, empleando variables globales.
- Empleando la programación procedural o estructurada. Algo más compleja que la anterior pero más clara y limpia.
- Utilizando la programación orientada a objetos. Más compleja que las anteriores, pero más potente.

Processing ofrece la posibilidad de generar un ejecutable para las diferentes plataformas; Mac OS, Windows y Linux. También se pueden desarrollar aplicaciones móviles gracias a la SDK para Android.

El entorno de desarrollo propio de Processing se llama PDE (Processing Development Enviroment) y se desarrolló en Java. Si el proyecto que se pretende realizar es muy complejo, existe la posibilidad de utilizar otros entornos más potentes como Eclipse o NetBeans.

El lenguaje Processing está basado en la versión 1.4.2 de Java, por lo que no se podrá utilizar en las versiones más actuales. Para solucionar este inconveniente se utiliza otro entorno de desarrollo, como los comentados en el párrafo anterior. En este caso se importaría una librería gráfica al proyecto.

Aunque, como se ha comentado en varias ocasiones, Processing esté basado en Java, hay diferencias entre ambos lenguajes:

- Processing presenta una sintaxis más relajada. También es una plataforma plug and play [10], es decir, no es necesario realizar configuraciones ni instalar software adicional para empezar a trabajar.
- Java no ofrece el nivel básico comentado anteriormente, gracias al cual, Processing ofrece la opción de realizar una programación simple y fácil, sin emplear funciones ni clases.
- Al programar de forma procedural, como en C, es posible definir funciones propias, que serán llamadas y ejecutadas en cualquier momento de la programación.

2.3. TIA Portal

TIA Portal (Totally Integrated Automation Portal) [11] es la plataforma de ingeniería de Siemens. Ofrece un entorno unificado para el control, visualización y accionamiento, integrando así todas las tareas de automatización de un proceso industrial. Además, al tratarse de una aplicación modular, se pueden añadir funcionalidades según las necesidades que se vayan demandando en cada proyecto.

Figura 5 Totally Integrated Automation Portal

TIA Portal incluye:

- Lenguaje de programación para autómatas programables (PLC) [12] de Siemens.
- Software para control de periféricos.
- Software WINCC, que permite la visualización de procesos y dispositivos.
- Start Drive, útil para la configuración e integración de accionamientos SINAMICS G de Siemens.
- Scout TIA, herramienta de Motion Control que permite acometer el control del movimiento de la máquina, pudiendo así realizar el control de la velocidad, posicionamiento, sincronismo relativo o absoluto etc.

Aunque, sin duda, lo más conocido de TIA Portal es la programación de PLC Simatic de Siemens.

La plataforma desarrollada por Siemens ofrece varias ventajas frente a sus competidores más cercanos. Una de ellas, la más importante, ya se comentó anteriormente; TIA Portal integra distintos software para procesos de producción industriales en un mismo interfaz, lo que facilita el aprendizaje, la interconexión y la operación. Esta integración permite ahorros de hasta el 30% durante la vida útil del ciclo de producción. Otra de las ventajas más distintivas que presenta, en la fácil detección de errores en la programación y la gestión de estos. Esta rápida respuesta minimiza los tiempos de parada de líneas de producción, aumentando la disponibilidad de la instalación y, consecuentemente, aumentando la rentabilidad de la planta. Además, la programación de controladores a través de TIA Portal, se lleva a cabo mediante lenguaje KOP. Este lenguaje de programación está basado en un esquema de contactos, similar a los de un esquema eléctrico. Al ser una programación gráfica, se convierte en un sistema intuitivo y cómodo, aunque sin dejar de ser una herramienta muy potente y versátil. Como resultado se obtiene una optimización de la calidad, la eficiencia y la consistencia de todo el proceso de producción.

2.4. S7-PLCSIM

S7-PLCSIM [13] es una herramienta ofrecida por Siemens que permite simular un PLC sin necesidad de disponer de ningún hardware. La simulación puede realizarse tanto en un PC como en una unidad de programación (como, por ejemplo, la PG 740 [14]). De esta forma, se puede simular completamente cualquier proyecto que se esté creando y ver cómo se comporta. Esto es muy útil a la hora del reconocimiento de errores y la depuración de programas.

Figura 6 S7-PLCSIM

Además, incorpora una interfaz de usuario sencilla, que permite visualizar y modificar parámetros utilizados por el programa, como por ejemplo la activación y desactivación de entradas. Las funciones que más destacan son las siguientes:

- Comando Pausa. Detiene la CPU simulada y permite reanudar la ejecución del programa en la operación donde se detuvo.
- Cualquier cambio que se haga en una subventana actualiza inmediatamente el contenido de la correspondiente dirección en la memoria. La CPU no espera hasta el final o el comienzo del ciclo para actualizar los datos que se hayan modificado
- Las opciones de control de ejecución permiten elegir cómo la CPU deberá ejecutar el programa. El Ciclo Individual ejecuta un ciclo del programa y espera a que el usuario solicite ejecutar el siguiente ciclo. Por el contrario, el Ciclo Continuo ejecuta el programa como un PLC

real, es decir, inicia un nuevo ciclo inmediatamente después de haber finalizado el que estaba ejecutando.

Aunque también existen otras muchas muy interesantes como la función “Grabar/reproducir”. Pese a que existen algunas diferencias entre trabajar con un PLC real y con un PLC simulado, esta herramienta ofrece un amplio abanico de posibilidades que hace que, para la mayoría de los casos, sea más que suficiente.

2.5. NetToPLCsim

NetToPLCsim [15] es una extensión de red para el simulador S7-PLCSIM comentado anteriormente. Esta extensión permite acceder al simulador desde su red a través de una comunicación TCP/IP (Iso-On-TCP), utilizando la interfaz de red del PC en el que se ejecuta la simulación.

2.6. Snap7

Snap7 [16] es un software multiplataforma con el que se consigue “imitar” las comunicaciones tal y como lo harían de forma nativa los PLC de Siemens, empleando tecnología Ethernet y protocolo S7.

Figura 7 Snap7

Entre sus características más destacables se encuentran las siguientes:

- Arquitectura diseñada tanto para 32 como para 64 bits.
- Se trata de software abierto.

- Disponible para varios sistemas operativos: Windows, Linux, MacOSX, etc.
- Soporte para varios tipos de CPU.
- Su funcionamiento no depende de librerías de terceros.
- La instalación no es necesaria para su correcto funcionamiento.
- Modelos de transmisión de información síncrona y asíncrona.

La implementación del protocolo S7, se soporta sobre una ampliación del protocolo TCP recogida en la RFC 1006 y titulada como “ISO Transport Service on top of TCP”. Esto es necesario ya que TCP está orientado al envío de datos no transmitiendo información sobre la longitud de la información contenida o sobre cuándo empiezan o terminan. Sin embargo, en aplicaciones de automatización, es indispensable operar orientado a mensajes. Esto es, enviar bloques de datos en los que el receptor sea capaz de reconocer dónde empiezan y terminan cada uno de ellos. Con RFC 1006, se logra esto ya que se incluye una cabecera que lo define, y, por tanto, garantiza el proceso.

Sin embargo, en este caso, para la resolución del proyecto, se ha utilizado Moka7 [16] en lugar de Snap7 para compatibilizarlo con el uso de NetBeans. Moka7 no tiene un código de interfaz que cargue Snap7, pero es una implementación Java pura del protocolo S7. Se implementa como un conjunto de clases de código fuente que se puede utilizar en un proyecto Java para comunicarse con un PLC real o simulado. Moka7 es parte de Snap7, por lo que comparten la misma licencia. El uso de esta variante se debe a que Moka7 es una biblioteca de clases que, por conveniencia, incluye los archivos fuente en dos proyectos: un proyecto de NetBeans 4.7 llamado Moka7-NetBeans y otro proyecto, en este caso de Eclipse Kepler, llamado Moka7-Eclipse. Los proyectos son absolutamente iguales y contienen los mismos archivos fuente, pero, en este caso, se trabajará con el proyecto de NetBeans como ya se ha comentado.

Figura 8 Moka7

2.7. Autodesk Inventor

Para el modelado 3D de los elementos que integran la estación se ha empleado el software Autodesk Inventor [17], más en específico la versión Autodesk Inventor Professional 2022.

Figura 9 Autodesk Inventor Professional 2022

Este software es un paquete de modelado paramétrico de sólidos 3D desarrollado por la empresa Autodesk [18]. Se trata de un entorno para diseño asistido lanzado por primera vez el 20 de septiembre de 1999. Con una larga

trayectoria en el mercado, compite con programas como SolidWorks, CATIA o Solid Edge. Autodesk Inventor ofrece gran facilidad para el modelado tridimensional a través de un entorno cercano al usuario.

A la larga trayectoria de desarrollo del software mencionado, se suma la gran expansión mundial de los productos de SMC, haciendo posible encontrar muchos de los componentes que integran la estación ya modelados y disponibles para descargar. Parte de los componentes utilizados para modelar la estación se han obtenido en las siguientes bibliotecas y bases de datos:

- SMC [19]
- Omron Industrial Automation [20]
- Mitsubishi Electric [21]
- 2D & 3D CAD MODELS MANUFACTURER CATALOGS
- Tracerparts Product Content Everywhere [22]

2.8. Sublime Text y extensión .yml

Sublime Text [23] (Figura 10) es un editor de texto que, aunque su versión completa es de pago, existe una versión de evaluación totalmente completa y funcional que está disponible para descargarla de forma gratuita y sin fecha de expiración.

Figura 10 Sublime Text

Se trata de un IDE muy liviano y fácil de instalar. Es compatible con un gran número de lenguajes de programación (C++, Java, JavaScrippt, PHP, Python, SQL, HTML, entre muchos otros), lo que le permite implementar una función de coloreado

y envoltura de sintaxis que resalta las expresiones propias de la sintaxis del lenguaje que se está utilizando, facilitando su lectura y comprensión.

Para desarrollar este proyecto, se ha utilizado Sublime Text para crear archivos de extensión .yml. Esta extensión resulta muy útil y cómoda porque utiliza un lenguaje de alto nivel. En esta ocasión, dichos archivos se utilizarán para almacenar variables de forma accesible para que el usuario pueda modificarlas fácilmente en cualquier momento.

3. GEMELO DIGITAL ESTACIÓN FMS 204

3.1. Estación FMS 204

Una célula de fabricación flexible, también conocida como FMS (Flexible Manufacturing System) por sus siglas en inglés, es un sistema formado por un conjunto de estaciones de trabajo. Cada estación trabaja de forma autónoma y está interconectada con el resto de las estaciones que integran la célula mediante un sistema de transporte de material. A esto se suma la alta automatización de los procesos productivos llevados a cabo con estos sistemas, brindando una amplia flexibilidad; tanto en variedad de piezas, como en volumen de producción. El resultado es un sistema de producción que permite una capacidad de fabricación o montaje equiparable a la obtenida en líneas de ensamblaje a la vez que ofrece piezas y/o ensambles de mayor complejidad que los que se obtienen en un montaje manual. En definitiva, estos sistemas de fabricación nacen con la intención de paliar las desventajas que presentan, por una parte, la producción de series unitarias y, por otra parte, la producción masiva de piezas idénticas. Esta idea se ve reflejada en la Figura 11, donde se representan distintos sistemas de producción, en función de la variedad de piezas que se pueden conseguir y el número de piezas por hora que se puede fabricar, de esto, dependerá que el sistema sea más o menos flexible, productivo y que suponga un mayor o menor coste.

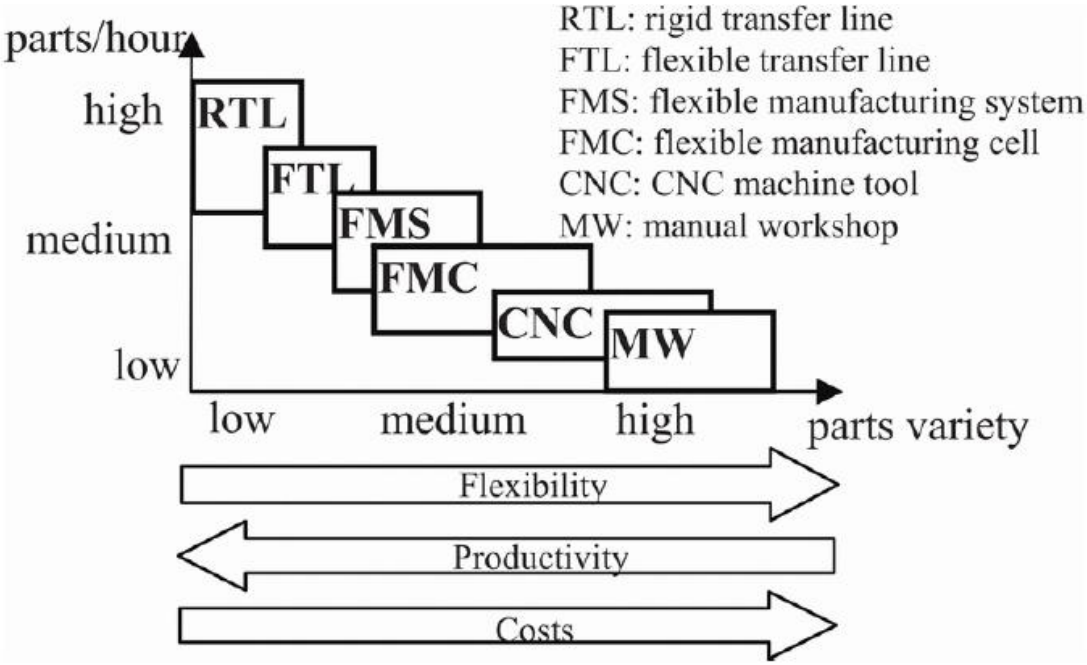


Figura 11 Variación de la flexibilidad, productividad y coste en distintos sistemas de fabricación. [24]

La célula FMS-200, mostrada en la Figura 12, Figura 12 Célula de fabricación flexible FMS-200es un sistema didáctico desarrollado por la compañía SMC [19] basándose en los sistemas de fabricación flexible descritos anteriormente.

Figura 12 Célula de fabricación flexible FMS-200

Esta célula está integrada por una serie de estaciones, cada una de la cual, realiza el proceso de inserción de un componente distinto. El resultado es un mecanismo de giro totalmente ensamblado (ver Figura 13 y Figura 14)

Figura 13 Elementos del sistema de giro

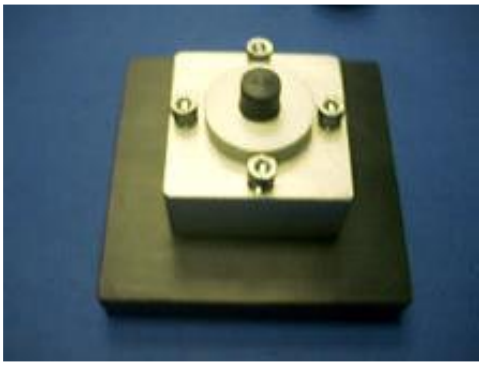


Figura 14 Sistema de giro ensamblado

Las estaciones que componen la célula se disponen alrededor de un sistema de transferencia, el cual, mediante cintas transportadoras, se encarga del transvase de las piezas entre estaciones. En el sistema de ensamblaje flexible, las estaciones que lo integran son las siguientes:

- Alimentación de la base
- Montaje del rodamiento
- Prensa hidráulica
- Inserción del eje
- Colocación de la tapa
- Montaje de tornillos
- Robot manipulador y atornillador
- Secado de pintura en horno
- Control de calidad por visión artificial
- Almacén de conjuntos terminados

Todas las estaciones anteriormente mencionadas tienen en común una botonera de mando situada en la parte frontal de las mismas, se puede ver en la Figura 15.

Figura 15 Vista frontal de la estructura de las estaciones

La botonera cuenta con pulsadores de marcha, paro y rearme, un selector de ciclo continuo/único, una seta de emergencia y pilotos luminosos de falta de material e indicador de error.

Sin embargo, el presente proyecto se desarrollará sobre la estación cuarta, encargada de la inserción del eje, la cual, también es nombrada como FMS-204.

Como se ha comentado anteriormente, la estación FMS-204, que se muestra en la Figura 16, es la encargada del montaje de un eje. Este eje se sitúa en el interior del rodamiento insertado en la estación anterior, la FMS-203.

Figura 16 Estación FMS-204. Inserción de ejes

Los componentes que integran la estación se distribuyen en torno a un plato giratorio que consta de ocho posiciones, a lo largo de las cuales se realizan distintas operaciones que se describen a continuación.

El plato giratorio se acciona por un motor paso a paso controlado por driver, de forma que, cada vez que gira, se produce un avance correspondiente a la división del plato entre el número de posiciones definidas, es decir, avanza una posición en sentido antihorario. Ver Figura 17 y Figura 18.

Figura 17 Plato giratorio

Figura 18 Servomotor para el giro del plato

En la primera posición del plato se depositan ejes que se encuentran almacenados en un alimentador tipo petaca. Esta alimentación se realiza por gravedad y siguiendo un sistema paso a paso, el cual se realiza mediante dos cilindros compactos neumáticos. Los cilindros se sitúan siempre en posiciones opuestas, de forma que, cuando el cilindro inferior libera el último eje del cargador, el cilindro superior sujeta al resto (Figura 19)

Figura 19 Posición de los cilindros de alimentación

El sistema de alimentación proporciona ejes de materiales distintos y en dos posiciones posibles.

En primer lugar, los ejes pueden ser de dos materiales distintos, de aluminio o de nylon. La diferencia entre uno y otro es notable a simple vista; los ejes de nylon son de color gris oscuro y mate, mientras que los de aluminio son de un gris más claro y brillante (Figura 20).

En segundo lugar, los ejes tienen dos alturas distintas; una correcta y otra incorrecta.

Figura 20 Ejes de nylon y aluminio

La segunda posición está destinada a la medición de la altura del eje. Como se ha comentado, los ejes no presentan una forma simétrica, por lo que es importante identificar la posición de cada eje para ensamblarlo en la posición correcta.

Para determinar la altura del eje se utiliza un cilindro neumático, que se muestra en la Figura 21.

Figura 21 Cilindro neumático para la medición de la altura del eje

El cilindro cuenta con un detector magnético, de esta forma distingue si, en su recorrido, se topa con el eje o, si por el contrario llega al final de carrera, indicando que el eje se ha colocado en la posición correcta.

La posición tercera es la responsable de darle la vuelta al eje en caso de que el cilindro neumático de la posición anterior detecte que el eje está colocado en la posición incorrecta.

Esto se lleva a cabo sujetando el eje con una pinza de dos dedos, elevándolo haciendo uso de un cilindro de vástagos paralelos para, seguidamente girarlo mediante un actuador de giro de 180º. Después el cilindro desciende y las pinzas se abren, colocando así el eje de nuevo en la mueca correspondiente del plato. Se puede ver a este tercer actuador en la Figura 22.

Figura 22 Actuador de giro de ejes

La siguiente operación que se realiza consiste en detectar el material del eje. Para esto se utilizan dos posiciones consecutivas de la estación.

En primer lugar, la posición cuarta cuenta con un sensor inductivo, de forma que, si el eje que se sitúa bajo él es de aluminio, el sensor se activará.

En segundo lugar, en la quinta posición se sitúa un sensor capacitivo, el cual se activa si el eje que se coloca debajo es de nylon. Sendos sensores se muestran en la Figura 23.

Figura 23 Sensores inductivo y capacitivo

La penúltima posición de la estación está destinada a la evacuación de ejes incorrectos. Determinar qué características de altura y material es correcto y cuál incorrecto, se deja a elección del usuario. Para llevar a cabo esta acción, la estación cuenta con un manipulador que, al recibir la orden correspondiente, extrae el eje del plato.

En la Figura 24 se observa que se trata de un manipulador de dos ejes, cuyo elemento terminal es una ventosa, que será la encargada de la sujeción del eje por su parte superior. Ambos ejes están compuestos por cilindros neumáticos de vástagos paralelos, mediante los cuales se realizan los movimientos de elevación del eje y conducción de este hasta una rampa de evacuación.

Como se ha comentado, la ventosa sujeta el eje mediante la técnica de vacío. Además de la ventosa, se utiliza un eyector para conseguir el vacío necesario y un vacuostato que suministrará una señal indicando que la sujeción es correcta.

Figura 24 Manipulador de expulsión de ejes incorrectos

Por último, y para concluir el ciclo de la estación FMS-204, en la séptima posición se realiza la inserción del eje. Para ello se utiliza un manipulador de tipo rotolineal, que puede observarse en la Figura 25.

Figura 25 Manipulador rotolineal para inserción de eje

Este manipulador se encarga de la recogida del eje, el desplazamiento del mismo hasta el punto de descarga y la inserción en el mecanismo de giro. El actuador puede desplazarse tanto en vertical como realizar un giro de unos 180º aproximadamente. Su función es descender hasta que la ventosa que posee en su extremo esté en contacto con el eje correspondiente del plato giratorio. A continuación, se lleva a cabo la sujeción mediante el mismo sistema de vacío

descrito en la posición anterior. Una vez sujeto el eje, el manipulador asciende y realiza el giro hasta colocarse sobre el mecanismo de giro. Por último, desciende y anula el efecto de vacío que sujetaba el eje, consiguiendo así insertarlo en el mecanismo y concluyendo, por tanto, el objetivo de la estación cuarta.

Para concluir con la descripción de la estación de inserción de ejes, se muestra, una recopilación de las entradas y salidas que ofrece.

Por un lado, en la Tabla 1 se muestran las entradas al PLC. Estas entradas son las señales generadas por los finales de carrera, detectores de vacío y sensores en general. En la tabla de entradas aparecen cuatro columnas, la primera de ellas, “nombre”, hace referencia al nombre con el que se identifica la variable. Cada actuador de la estación está etiquetado con una letra, por lo que el nombre de la variable de entrada hace referencia a esta letra. En la segunda columna aparece el tipo de variable, seguidamente, en la tercera columna se escribe el direccionamiento del PLC que coincide con la variable en cuestión, por último, se ha añadido una columna de comentarios en la que se hace una descripción de la variable para que el lector pueda identificar de qué se trata.

Tabla 1 Entradas de la estación cuarta al PLC

NOMBRE	TIPO DE DATO	DIRECCIÓN	COMENTARIO
Start	Booleano	%I2.0	Pulsador de inicio
Stop	Booleano	%I2.1	Pulsador de paro
Continuo/Único	Booleano	%I2.2	Selector de modo
Reset	Booleano	%I2.3	Pulsador de reinicio
a1	Booleano	%I2.4	Final de carrera superior del cilindro A
a0	Booleano	%I2.5	Final de carrera inferior del cilindro A
b0	Booleano	%I2.6	Final de carrera

			cilindro B girado sobre el plato
b1	Booleano	%I2.7	Final de carrera cilindro B girado sobre el mecanismo de giro
g1	Booleano	%I3.0	Final de carrera superior del cilindro G
g0	Booleano	%I4.3	Final de carrera inferior del cilindro G
j0	Booleano	%I3.1	Plato en reposo
dm	Booleano	%I3.2	Detector de metal
dp	Booleano	%I3.3	Detector de presencia/nylon
V1	Booleano	%I3.4	Vacío en ventosa 1
c0	Booleano	%I3.5	Cilindro C retraído
c1	Booleano	%I3.6	Cilindro C extendido
d0	Booleano	%I3.7	Final de carrera superior del cilindro D
d1	Booleano	%I4.0	Final de carrera inferior del cilindro D
V2	Booleano	%I4.1	Vacío en ventosa 2
f1	Booleano	%I4.2	Final de carrera cilindro F

Por otro lado, y de forma análoga a la tabla de entradas anterior, en la Tabla 2 aparecen recogidas las señales de salida que el PLC puede mandar a la estación.

Tabla 2 Salidas del PLC a la estación cuarta

NOMBRE	TIPO DE DATO	DIRECCIÓN	COMENTARIO
Alarma	Booleano	%Q2.0	LED de alarma
A+	Booleano	%Q2.1	Bajar cilindro A
B+	Booleano	%Q2.2	Rotar cilindro B
V1+	Booleano	%Q2.3	Activar ventosa 1
C+	Booleano	%Q2.4	Expandir cilindro C
C-	Booleano	%Q2.5	Retraer cilindro C
D+	Booleano	%Q2.6	Bajar cilindro D
V2+	Booleano	%Q2.7	Activar ventosa 2
E+	Booleano	%Q3.0	Expulsar un eje
F+	Booleano	%Q3.1	Bajar cilindro F
G+	Booleano	%Q3.2	Bajar cilindro G
H+	Booleano	%Q3.3	Girar eje
I+	Booleano	%Q3.4	Cerrar pinzas para sujeción de eje
J+	Booleano	%Q3.5	Girar plato
K+	Booleano	%Q3.6	
FM	Booleano	%Q3.7	LED falta material

3.2. Implementación del Gemelo Digital

La resolución del presente proyecto se ha desarrollado siguiendo los siguientes principios.

El primer paso consiste en el modelado 3D de la estación. Este modelado, como se ha comentado en epígrafes anteriores, se ha realizado en un software de modelado 3D. Posteriormente, para poder utilizar los modelos y representarlos mediante Processing, ha sido necesario exportarlos con extensión .obj. Además, es importante separar los modelos de los actuadores en dos, por una parte, se exportará la base, o parte fija del actuador, es decir, aquella que no tendrá movimiento. Por otra parte, se exporta la parte móvil, que será la encargada de representar el movimiento del actuador.

Una vez se han obtenido todos los objetos, se representarán en el espacio 3D de Processing respetando las posiciones de cada actuador en la estación real.

El siguiente paso será dotar de movimiento a las partes móviles de los actuadores. Esto se hará siguiendo las pautas correspondientes a la clase a la que pertenezca cada actuador. Esto se detallará en epígrafes posteriores.

Por último, se configurará la comunicación con el PLC, (o PLC virtual en este caso) para que, desde la programación realizada en TIA Portal se modifiquen las posiciones de los actuadores virtuales. En definitiva, conseguir un gemelo digital de la estación real totalmente funcional.

Al utilizar un lenguaje de programación orientado a objetos (Java), se ha resuelto creando una serie de clases y paquetes que se detallan a continuación, con el objetivo de que el lector obtenga una idea clara y esquemática del planteamiento y funcionamiento general de la programación que se presenta.

3.2.1. Paquetes

En primer lugar, se ha dividido la programación en dos paquetes bien diferenciados. En Java, un paquete no es más que un contenedor de clases, cuyo objetivo consiste en agrupar clases e interfaces dotando así al código de una modularidad que facilitará la comprensión y orden del programa.

Si se observa la Figura 26, en el árbol de proyecto existe un espacio dedicado a los paquetes que se han creado, este espacio aparece con el nombre *"Source Packages".* Figura 26 Árbol de proyecto

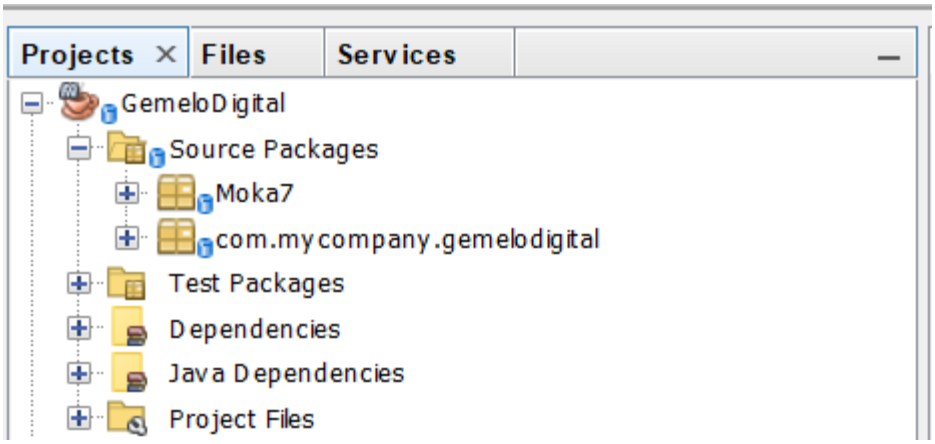


Figura 26 Árbol de proyecto

En este caso, el paquete Moka7 es el encargado de recoger todas las clases responsables de la comunicación entre el PLC virtual y la programación. En la Figura 27, al desplegar la rama de dicho paquete, se pueden observar las clases de comunicación.

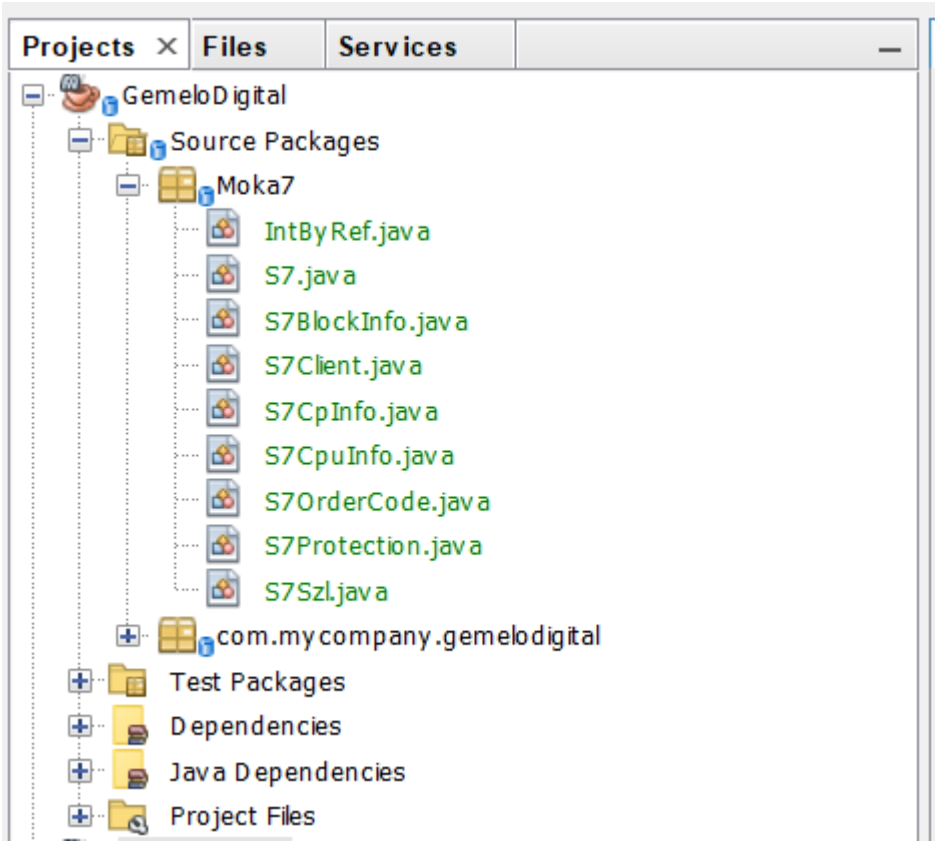


Figura 27 Paquete Moka7

Esta parte del código no es más que una importación de la librería Moka7 comentada en el epígrafe anterior sobre 12Snap7, por lo que su estudio y comprensión se deja en manos del lector.

El último paquete que forma parte del código es el llamado *"com.mycompany.gemelodigital"* (Figura 26). Es el paquete que contiene el resto de las clases. Estas clases, que aparecen en la Figura 28, se detallan en el siguiente epígrafe 5.2 Clases.

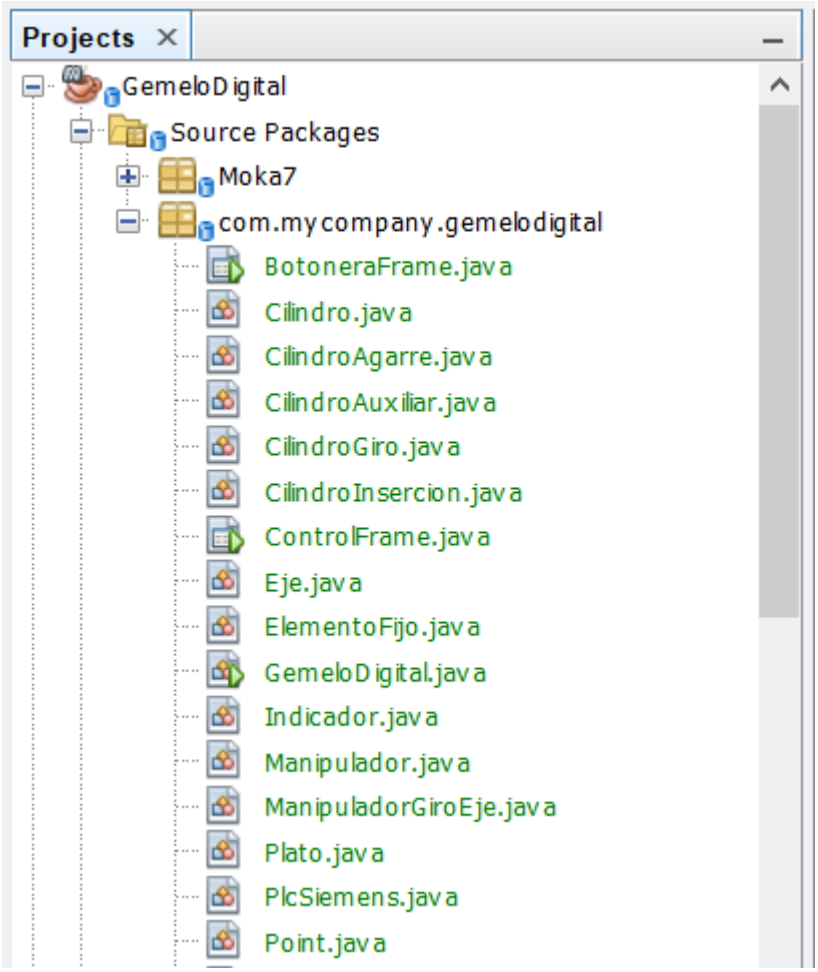


Figura 28 Paquete com.mycompany.gemelodigital

3.2.2. Clases

A continuación, se explicará brevemente cómo funcionan las clases que integran este proyecto.

Es importante conocer que, en Java, una clase se puede definir como una plantilla que define la forma de un objeto.

Un objeto, por su parte, es un espécimen de una clase, que normalmente se utiliza para modelar objetos del mundo real.

Una interfaz gráfica es una clase que, al ser ejecutada genera ventanas, cuadros de diálogo, barras de herramientas botones, etc.

Por último, es importante mencionar que el sistema de coordenadas seguido para la representación de los objetos que conforman la estación ha sido el que se muestra en la

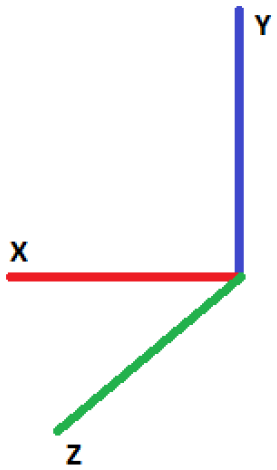


Figura 29 Sistema de coordenadas

3.2.2.1. Coordenadas

Point.java es una clase muy sencilla, que se utiliza para, como su propio nombre indica, recoger en un mismo objeto las tres coordenadas que delimitan un punto en el espacio. Es por esto por lo que sus tres únicos atributos (Figura 30), son X, Y y Z, que se corresponden con las respectivas coordenadas de un punto en el espacio.

Además, esta clase contiene constructores para 2D y 3D, es decir, se pueden crear puntos de 2 o 3 coordenadas para trabajar en espacios bidimensionales y tridimensionales.

```
public class Point {
    float x=0.0f;
    float y= 0.0f;
    float z=0.0f;
```

Figura 30 Atributos de la clase Point.java

3.2.2.2. Final de carrera

La clase Indicador es otra de las más básicas y elementales, y se utilizará en muchas otras clases. Los indicadores serán los encargados de denotar cuándo una señal está activa o inactiva. El uso más frecuente en este proyecto ha sido utilizar los indicadores como finales de carrera, es decir, no es más que un objeto que define un estado; activo o inactivo.

Los atributos de Indicador, que aparecen en la Figura 31 permiten, por una parte, controlar el estado de un objeto como se ha comentado anteriormente y, por otra parte, permiten dibujar unas señales redondeadas las cuales están programadas para que cambien de color cuando cambie le estado del objeto en cuestión. Es por esto por lo que aparece un atributo de tipo Point, que refleja las coordenadas donde se desea dibujar el indicador, y otros atributos de tipo entero que almacenan, en formato RGB, los colores que indicarán si el estado el objeto es activo o inactivo.

Para la función de dibujar los indicadores, se ha creado un método display cuya lógica es muy simple; comprueba el valor del atributo estado y, dependiendo de si está activo o inactivo, dibuja una elipse, en el punto marcado por el atributo de tipo Point, del color que corresponda.

```
public class Indicador {

    GemeloDigital root;
    Point pos;
    Boolean estado;
    int radio;
    Integer pulsado;
    Integer noPulsado;
    int pulsadoR;
    int pulsadoG;
    int pulsadoB;
    int noPulsadoR;
    int noPulsadoG;
    int noPulsadoB;
```

Figura 31 Atributos de la clase Indicador.java

3.2.2.3. Cilindros

Cilindro.java es una de las clases más básicas y robustas que se ha diseñado. A partir de ella se han ido desarrollando otras muchas siguiendo un planteamiento muy similar. Es por esto por lo que, con objetivo de que el lector comprenda el funcionamiento del código, se explicará más detalladamente las características de esta clase.

En primer lugar, las librerías. En la clase cilindro se han importado las librerías que aparecen en la Figura 32.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.Map;
import org.yaml.snakeyaml.Yaml;
import processing.core.*;
import static processing.core.PConstants.PI;
```

Figura 32 Librerías en la clase Cilindro.java

El paquete java.io está destinado a controlar los flujos de datos, es decir, se encarga de la lectura y escritura de estos. En este caso, solo se han importado las

clases necesarias de este paquete, pero, tal y como aparece en la Figura 33 existen muchas otras.

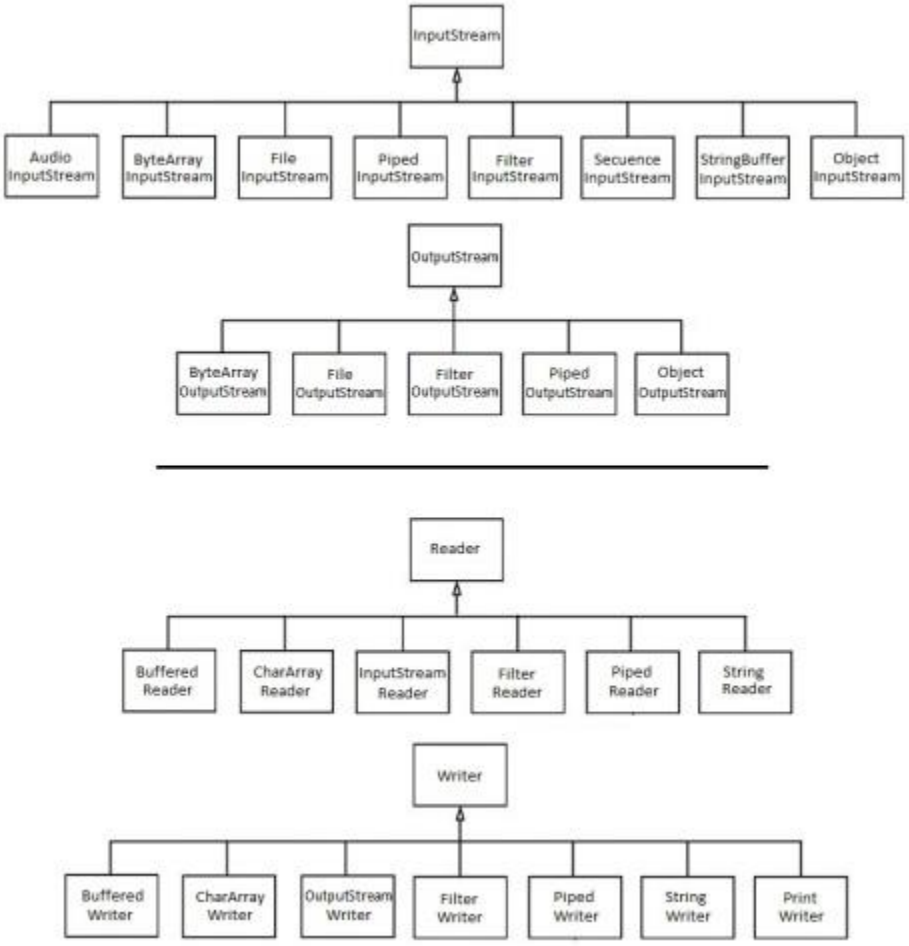


Figura 33 Esquema con las clases principales del paquete java.io [25]

Además de este paquete, se hace uso de otro llamado java.util. En este caso, se trata de un paquete formado por un gran conjunto de interfaces y clases. La interfaz Map, usada en este proyecto, es solo una de las tantas que incluye. Esta interfaz define el comportamiento de un objeto que mapea o asigna a una clave un determinado valor. En la Figura 34 aparecen algunas de las clases más utilizadas del paquete java.util. De forma análoga, en la Figura 35 aparecen algunas de las interfaces más importantes.

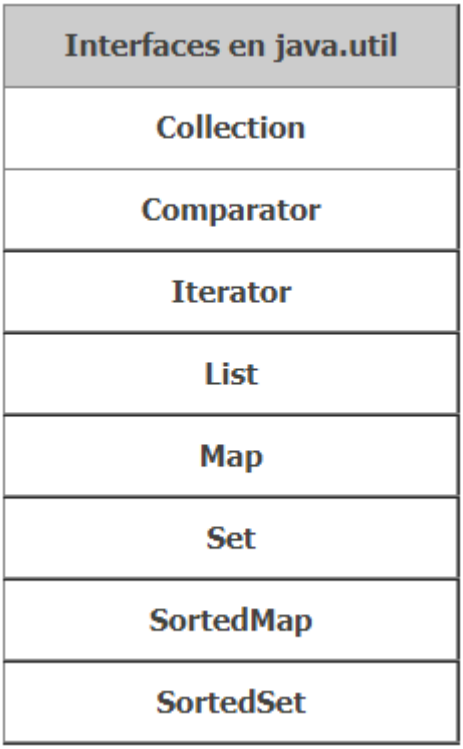


Figura 35 Interfaces principales del paquete java.util

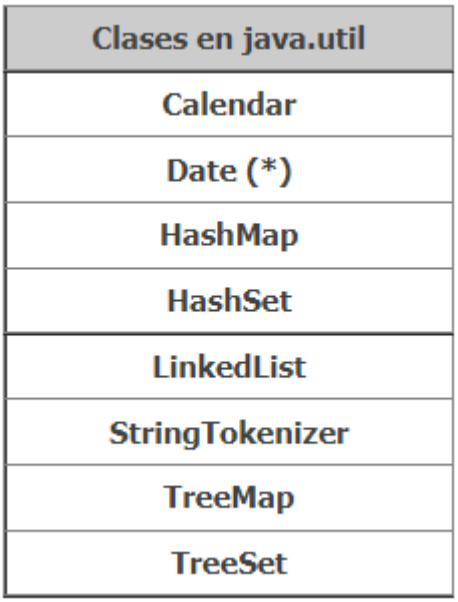


Figura 34 Clases principales del paquete java.util

La librería org.yaml.snakeyaml.Yaml pertenece al repositorio de Maven y permite la serialización de datos de forma que sea legible por humanos. Gracias a esta librería, se puede interpretar desde la programación, los archivos .yml comentados anteriormente, consiguiendo que el formato de comunicación entre el humano y el programa sea muy cómodo e intuitivo para el usuario final.

Las dos últimas librerías que aparecen en la Figura 32, pertenecen a Processing, y permiten implementar este lenguaje en la programación. Para más información, léase referencia [26].

En segundo lugar, aparecen en la Figura 36 los atributos de la clase Cilindro.java.

```
public class Cilindro {

    private GemeloDigital root;
    private Map<String, Object> params;
    private Point posBase;
    private Point rotBase;
    private Point posMovil;
    private Point rotMovil;
    private String nombreBase;
    private String nombreMovil;
    private PShape base;
    private PShape movil;
    private int escala;
    private int vel;
    private int vel_temp;
    private int posMax;
    private int posMin;
    private int pos;
    private int dirExpandir;
    private int dirRetraer;
    private int dirExtendido;
    private int dirRetraido;
    private int byteMemoriaExpandir;
    private int byteMemoriaRetraer;
    private int byteMemoriaExtendido;
    private int byteMemoriaRetraido;
    private Indicador indicadorRetraido;
    private Indicador indicadorExtendido;
```

Figura 36 Atributos de la clase Cilindro.java

El primer atributo se repetirá en todas las demás clases, y consiste en lo que en programación se conoce como “root”, que no es más que una forma de indicar que, en este caso, GemeloDigital posee los privilegios del sistema. A partir de ahora, cada vez que se necesite utilizar una función de Processing en el código fuente de una clase, se deberá escribir el nombre de la función precedida por “root.”, por ejemplo, en la Figura 37 se muestra cómo, para realizar traslaciones y rotaciones en la clase cilindro, se utiliza la sintaxis indicada.

```
root.pushMatrix();
root.translate(posBase.x, posBase.y, posBase.z);
root.rotateX(rotBase.x);
root.rotateY(rotBase.y);
root.rotateZ(rotBase.z);

root.shape(movil, 30, 0+pos);


root.popMatrix();
```

Figura 37 Ejemplo demostrativo del uso de root

Siguiendo con los atributos de la clase Cilindro.java, aparecen cuatro de tipo Point. Dos de ellos indican posición y, los otros dos, rotación. Además, se hace referencia a “Base” y “Móvil”, esto significa que los dos primeros atributos pertenecen a la posición y rotación de la parte fija del cilindro que se esté representando y, los dos restantes, se corresponden con la parte móvil del cilindro, es decir, el vástago.

Los siguientes atributos, de tipo String, serán donde se almacenen los nombres de los archivos de extensión .obj, tanto de la parte fija como de la parte móvil del cilindro, comentadas anteriormente. Es importante que el formato de estas cadenas de caracteres sea el siguiente: “NombreDelArchivo.obj”.

A continuación, dos atributos de tipo PShape que, de forma análoga, se corresponde con las partes fija y móvil del cilindro. El tipo de dato PShape se utiliza para almacenar formas [27].

Los siguientes atributos, por orden de aparición consisten en lo siguiente:

- escala: la escala que se aplicará a la representación del objeto
- vel: velocidad con la que se moverán las partes móviles de los cilindros, este valor se toma del archivo de configuración .yml que aparece en el ejemplo de la Figura 36.
- vel_temp: se utiliza para incrementar o disminuir el valor del atributo pos y conseguir que el objeto representado se vaya desplazando por el espacio.

- posMax: es la posición máxima hasta la que se desplazarán las partes móviles del cilindro. Se lee del archivo de configuración del cilindro correspondiente.
- posMin: análoga a la anterior, hace referencia a la posición mínima hasta la que se desplazará el objeto. Junto con la posición máxima, establecen los límites del movimiento de las partes móviles. Su valor se toma del archivo de configuración.
- pos: es la variable que se irá incrementando o disminuyendo marcando así el sentido del movimiento del cilindro, es decir, si se expande o se retrae.
- dirExpandir: también se lee del archivo de configuración y se trata de un número, escrito en hexadecimal, que representa la dirección de memoria relacionada con la expansión del cilindro.
- dirRetraer: análoga a la anterior, hace referencia a la dirección para retraer el cilindro.
- dirExtendido: también se toma del archivo de configuración y, siguiendo el mismo criterio que en los dos atributos anteriores, hace referencia a la dirección del final de carrera que indica que el cilindro está completamente extendido.
- dirRetraido: exactamente igual al anterior, pero para el final de carrera opuesto.
- byteMemoriaExpandir: este dato es un entero que se obtiene del fichero de configuración del cilindro en cuestión. Hace referencia al número de byte de salidas del PLC al que pertenece la dirección de la acción expandir, comentada anteriormente.
- byteMemoriaRetraer: funciona igual que el atributo anterior. Hace referencia al byte de salidas del PLC al que pertenece la dirección de la acción de retraer.
- byteMemoriaExtendido: análogo a los anteriores, pero en este caso, se hace referencia al byte de entradas al que pertenece la señal de cilindro extendido.

- byteMemoriaRetraido: igual al anterior, pero haciendo referencia al byte de entradas al que pertenece la dirección indicadora de que el cilindro se encuentra completamente retraído.
- indicadorRetraido: este atributo, de tipo Indicador, se utiliza como final de carrera. En el archivo de configuración aparece la ruta en la que se almacena el archivo de configuración del propio indicador. Se ha creado además un archivo de indicador Null (tal y como puede observarse en la Figura 38). Este indicador anula la funcionalidad de representarlo como una elipse que cambia de color al activarse.
- indicadorExtendido: totalmente análogo al atributo anterior. Indica cuándo el cilindro en cuestión se encuentra totalmente extendido.

```
posXFijo: -120
posYFijo: 135
posZFijo: -210
rotXFijo: 0
rotYFijo: -130
rotZFijo: 0
posXMovil: 0
posYMovil: 135
posZMovil: 0
rotXMovil: 0
rotYMovil: 0
rotZMovil: 0
escala: 1
carreraMax: 35
carreraMin: -20
velocidad: 5
byteExpandir: 3
dirExpandir: 0x01
byteRetraer: 0
dirRetraer: 0x08
byteExtendido: 4
dirExtendido: 0x02
byteRetraido: 0
dirRetraido: 0x20
ficheroFijo: "CilindroFFijo.obj"
ficheroMovil: "CilindroFMovil.obj"
indicadorRetraido: "config\\indicadorNull.yml"
indicadorExtendido: "config\\indicadorNull.yml"
```

Figura 38 Ejemplo de archivo de configuración de un cilindro de la clase Cilindro.java

Seguidamente, aparece el constructor de la clase Cilindro, que puede verse en la Figura 39.

```
Cilindro(GemeloDigital r, String nombreFichero) throws FileNotFoundException {
    root = r;
    InputStream inputStream = new FileInputStream(new File(nombreFichero));
    Yaml yaml = new Yaml();
    Map<String, Object> data = yaml.load(inputStream);
    System.out.println(data);
    this.pos = 30;
    this.posMax = (int) data.get("carreraMax");
    this.posMin = (int) data.get("carreraMin");
    posBase = new Point((int) data.get("posXFijo"),
            (int) data.get("posYFijo"), (int) data.get("posZFijo"));
    rotBase = new Point((int)data.get("rotXFijo")*PI/180,
            (int)data.get("rotYFijo")*PI/180,(int)data.get("rotZFijo")*PI/180);
    posMovil = new Point((int) data.get("posXMovil"),
            (int) data.get("posYMovil"), 0);
    rotMovil = new Point((int)data.get("rotXMovil")*PI/180,
            (int)data.get("rotYMovil")*PI/180,(int)data.get("rotZMovil")*PI/180);
    escala = (int) data.get("escala");
    posMin = (int) data.get("carreraMin");
    posMax = (int) data.get("carreraMax");
    vel = (int)data.get("velocidad");
    byteMemoriaExpandir = (int) data.get("byteExpandir");
    dirExpandir = (int) data.get("dirExpandir");
    byteMemoriaRetraer = (int) data.get("byteRetraer");
    dirRetraer = (int) data.get("dirRetraer");
    byteMemoriaExtendido = (int) data.get("byteExtendido");
    dirExtendido = (int) data.get("dirExtendido");
    byteMemoriaRetraido = (int) data.get("byteRetraido");
    dirRetraido = (int) data.get("dirRetraido");
    nombreBase = (String) data.get("ficheroFijo");
    nombreMovil = (String) data.get("ficheroMovil");
    indicadorRetraido = new Indicador(root, (String) data.get("indicadorRetraido"));
    indicadorExtendido = new Indicador(root, (String) data.get("indicadorExtendido"));
}
```

Figura 39 Constructor de la clase Cilindro.java

Este constructor tiene solo dos argumentos; la ya comentada anteriormente “root” y un argumento de tipo string. Este último hace referencia al nombre del archivo de extensión .yml que contiene todos los parámetros de configuración del cilindro (Figura 38). Además, se observa que se lanza la excepción “FileNotFoundException” [28], con el objetivo de que, si el archivo con nombre igual al insertado en el atributo, no se encuentra en la ruta especificada, se generará la excepción.

A continuación, puede observarse el método utilizado para la lectura del fichero de configuración, del cual se extraen los parámetros necesarios y se almacenan en variables. Para conseguir esto, en primer lugar se crea un objeto del tipo

InputStream [29], que es una clase abstracta que declara los métodos para poder leer datos de una fuente concreta. También se crea un objeto de tipo Yaml que permitirá acceder al archivo de configuración, que tendrá extensión .yml. Se escriben en pantalla los datos que se van almacenando gracias a la expresión System.out.println(data).

Lo siguiente será ir almacenando en los atributos descritos anteriormente, la información que contiene el archivo de configuración. Esto se hará de la siguiente utilizando el método get de la interfaz Map.

Esto irá seguido de los métodos get y set correspondientes a los atributos de la clase.

Lo siguiente que aparece en el código fuente de la clase Cilindro, es el método setup() que aparece en la Figura 40.

```
public void setup() {      //Lee el fichero .obj
    base = root.loadShape(nombreBase);
    base.scale(escala,escala,-escala);
    movil = root.loadShape(nombreMovil);
    movil.scale(escala,escala,-escala);

}
```

Figura 40 Método setup() de la clase Cilindro.java

Este método se utiliza para leer el archivo .obj. En primer lugar, utilizando el método loadShape() [30]se carga el fichero y seguidamente se escala con scale() [31].

El siguiente método que aparece es display(), de la Figura 41. Será el encargado de dibujar el cilindro en el espacio de representación.

```
public void display() {    //Dibuja el cilindro
    root.pushMatrix();
    root.translate(posBase.x, posBase.y, posBase.z);
    root.rotateX(rotBase.x);
    root.rotateY(rotBase.y);
    root.rotateZ(rotBase.z);

    root.shape(movil, 30, 0+pos);
    root.shape(base, 30, 0);


    root.popMatrix();

    pos += vel_temp;

}
```

Figura 41 Método display() de la clase Cilindro.java

Por primera vez aparecen las funciones pushMatrix() [32] y popMatrix() [33] de Processing. Con la primera de ellas, se guarda la matriz de coordenadas hasta que se ejecuta la segunda de ellas, es entonces cuando se reestablecen. Con esto se consigue que las transformaciones que se realicen entre las llamadas de las dos funciones no se sumen a las anteriores, si no que se efectúan desde la posición original, es decir, es una forma sencilla de rotar y/o trasladar un objeto sin que esto repercuta sobre el resto de los objetos representados.

Para finalizar, en la Figura 42 aparece el último método de esta clase, el cual se comentará de forma general debido a que se profundizará más en este tema en epígrafes posteriores.

```
public void update(byte[] entradasPLC,byte[]salidasPLC){
    if ((salidasPLC[byteMemoriaExpandir] & dirExpandir) > 0 & pos >= posMin) {
        vel_temp = -vel;
    } else if ((salidasPLC[byteMemoriaRetraer] & dirRetraer) > 0 & pos <= posMax) {
        vel_temp = vel;
    } else {
        vel_temp = 0;
    }

    if(pos>=posMax){
        indicadorRetraido.estado=true; //TODO Cambiar a funcion
        indicadorExtendido.estado=false; //TODO Cambiar a funcion
        entradasPLC[byteMemoriaExtendido] = (byte) (entradasPLC[byteMemoriaExtendido] | dirExtendido);
    }else if (pos <= posMin){
        indicadorRetraido.estado=false; //TODO Cambiar a funcion
        indicadorExtendido.estado=true; //TODO Cambiar a funcion
        entradasPLC[byteMemoriaRetraido] = (byte) (entradasPLC[byteMemoriaRetraido] | dirRetraido);
    }else{
        indicadorRetraido.estado = false;
        indicadorExtendido.estado = false;
        entradasPLC[byteMemoriaExtendido] = (byte) (entradasPLC[byteMemoriaExtendido] & ~dirExtendido);
        entradasPLC[byteMemoriaRetraido] = (byte) (entradasPLC[byteMemoriaRetraido] & ~dirRetraido);

    }
}
```

Figura 42 Método update() de la clase Cilindro.java

Este método se encarga de actualizar los valores de las entradas y salidas del PLC. Se ejecutará en bucle, constantemente, asegurando así que cada vez que se modifique alguna entrada o alguna salida, se actualicen estas en la comunicación con el PLC.

Por su parte, CilindroAgarre.java es una clase muy similar a la clase Cilindro.java explicada anteriormente. El planteamiento general de la clase es análogo; una clase cuyos atributos se obtienen de la lectura de un archivo de texto y con tres métodos significativos, setup(), display() y update().

El motivo de la creación de esta clase se debe a la necesidad de incluir en el fichero de configuración direcciones relativas a la apertura y cierre. Además, como esta clase se ha creado específicamente para las pinzas de sujeción de ejes del actuador encargado de girar los mismos, ha sido necesario incluir dos elementos de tipo PShape, uno por pinza, ya que cada pinza tendrá un movimiento distinto a la otra (misma dirección, pero sentido opuesto). Es por esto que el objeto pinza no se puede tratar como un único objeto, y es necesario separarlo en dos; pinza izquierda y pinza derecha. Todo esto se ve reflejado en la Figura 43.

```
CilindroAgarre(GemeloDigital r, String nombreFichero) throws FileNotFoundException  {
    root = r;
    InputStream inputStream = new FileInputStream(new File(nombreFichero));
    Yaml yaml = new Yaml();
    Map<String, Object> data = yaml.load(inputStream);
    System.out.println(data);

    this.pos = 0;
    rotMovil = new Point((int)data.get("rotX")*PI/180,(int)data.get("rotY")*PI/180,(int)data.get("rotZ")*PI/180);
    posMovil = new Point((int)data.get("posX"),(int)data.get("posY"),(int)data.get("posZ"));
    escala = (int) data.get("escala");
    this.carreraMax = (int) data.get("carreraMax");
    this.carreraMin = (int) data.get("carreraMin");
    velocidad = (int)data.get("velocidad");

    dirAbrir = (int) data.get("dirAbrir");
    byteAbrir = (int) data.get("byteAbrir");
    dirCerrar = (int) data.get("dirCerrar");
    byteCerrar = (int) data.get("byteCerrar");
    dirAbierto = (int) data.get("dirAbierto");
    byteAbierto = (int) data.get("byteAbierto");
    dirCerrado = (int) data.get("dirCerrado");
    byteCerrado = (int) data.get("byteCerrado");

    nombrePinza1 = (String) data.get("pinza1");
    nombrePinza2 = (String) data.get("pinza2");
    indicadorAbierto = new Indicador(root, (String) data.get("indicadorAbierto"));
    indicadorCerrado = new Indicador(root, (String) data.get("indicadorCerrado"));
}
```

Figura 43 Atributos de la clase CilindroAgarre.java

Con la clase CilindroGiro.java sucede lo mismo que en la anterior. Lo que la diferencia de la clase Cilindro.java es que la primera ha sido creada para poder simular un actuador que rota sobre sí mismo. En concreto, para la estación cuatro se ha utilizado en el actuador comentado anteriormente; el encargado de girar los ejes para colocarlos en la posición correcta. Para conseguir esto, si se observa la Figura 44, donde se pueden ver los atributos de la clase, se observa que, en lugar de trabajar con las acciones expandir y retraer, en esta ocasión se hace con Girar, volver y los respectivos Indicadores de final de carrera.

```
public class CilindroGiro {
    protected GemeloDigital root;
    protected Map<String, Object> params;
    protected Point posMovil;
    protected Point rotMovil;
    protected String nombreMovil;
    protected String ejeGiro;
    protected PShape movil;
    protected int escala;
    protected int velGiro;
    protected int velGiro_temp;
    protected int giroMax;
    protected int giroMin;
    protected int giro;
    protected int byteGirar;
    protected int dirGirar;
    protected int byteVolver;
    protected int dirVolver;
    protected int byteGirado;
    protected int dirGirado;
    protected int byteVuelto;
    protected int dirVuelto;
    protected Indicador indicadorGirado;
    protected Indicador indicadorVuelto;
```

Figura 44 Atributos de la clase CilindroGiro.java

Además, ahora el movimiento consistirá en una rotación y no en un movimiento lineal como en los casos anteriores. Para esto, en la Figura 45 se puede ver cómo, en el método display() de la clase, la variable que se incrementa y disminuye, llamada giro, se utiliza en la función RotateY() [34]. Esta función de Processing necesita que su argumento se introduzca en radianes, y es por esto por lo que se realiza la conversión de la Figura 45.

```
void display(Point posManip, int desplazamiento, Point rotManip) {
    root.pushMatrix();
    root.translate(posManip.x, posManip.y-desplazamiento, posManip.z);
    root.rotateX(rotManip.x);
    root.rotateY(rotManip.y+(giro*PI/180));
    root.rotateZ(rotManip.z);

    root.shape(movil, 0, 0);

    giro += velGiro_temp;
    root.popMatrix();
}
```

Figura 45 Método display de la clase CilindroGiro.java

Otra diferencia con respecto a la clase original Cilindro, reside en los argumentos del método display(). Si se observa la Figura 45, se aprecia que el método encargado de dibujar el cilindro de rotación necesita tres argumentos, mientras que en la clase cilindro no aparece ninguno.

Estos argumentos surgen de la necesidad de incluir este tipo de cilindro en un manipulador, es decir, un actuador que incorpora varios cilindros. En el caso del actuador mencionado anteriormente, encargado de girar los ejes para colocarlos en la posición correcta, se trata de un manipulador formado por tres actuadores, cuyos movimientos son propios de ellos mismos, pero repercuten en el resto.

Este manipulador está formado, en primer lugar, por un cilindro vertical común, que se expande y retrae como se ha explicado anteriormente. La diferencia reside en que, anclado a él se encuentran los dos actuadores restantes, que se comentan a continuación. Esto provoca que, al desplazarse el cilindro verticalmente, arrastre con él al resto de actuadores. En segundo lugar, se encuentra el cilindro de giro comentado anteriormente. Este verá modificado su posición vertical dependiendo del movimiento que describa el cilindro vertical y de forma totalmente independiente a si el cilindro de giro está o no rotando sobre sí mismo. En tercer y último lugar, a estos actuadores se encuentran ancladas las pinzas correspondientes al cilindro de agarre comentado en un epígrafe anterior. Estas pinzas, independientemente del movimiento de cierre y apertura, verán modificada su posición vertical por el cilindro simple, y su rotación en el eje Y por el cilindro de giro.

Esto se solventa añadiendo los siguientes atributos al método display():

- posManip: argumento de tipo Point, comentado anteriormente. Este argumento se utiliza para representar el cilindro de giro en la posición correcta. Es importante que a la hora de generar los ficheros .obj de los actuadores, se tenga en cuenta el origen de coordenadas de cada uno. Para que este método funcione sin problema, todo el manipulador debe mantener el mismo origen de coordenadas, de forma que, al extraer los actuadores individuales, el origen de cada uno de ellos coincida exactamente con el del manipulador completo y el de cada uno de los actuadores restantes. De esta forma, al utilizar la posición en los ejes x, y y z designados en el archivo de configuración del manipulador completo, para representar los actuadores individuales, obtendremos la representación del conjunto completamente coincidente.
- Desplazamiento: este argumento, de tipo entero, se utiliza para dotar, en este caso, al cilindro de giro, de movimiento vertical. Esta variable se extrae del cilindro vertical, y no es más que la variable comentada anteriormente, la cual incrementaba o disminuía su valor para retraer o extender el cilindro, creando así un movimiento ascendente o descendente.
- rotManipulador: este argumento, también de tipo Point, no es más que una analogía al primer argumento explicado en esta sección, posManip, pero en este caso es el responsable de la rotación. De esta forma, si para colocar el manipulador en su posición correcta alrededor del plato giratorio, se necesita rotarlo con respecto al eje Y (eje vertical), solo habrá que añadir la rotación que se desee al archivo de configuración del manipulador.

La clase CilindroAuxiliar.java se ha creado para ser utilizada en una tarea sencilla. No es más que una clase para cilindros auxiliares que serán empleados en los manipuladores. La diferencia frente a la clase Cilindro, reside de nuevo en los métodos display que implementa. La mecánica que se sigue es análoga a la comentada en el caso anterior. Esta clase implementa varios polimorfismos del método display().

En la Figura 46 se puede observar el método displayMovVert, que, como su propio nombre indica, será el utilizado para los manipuladores que implementen cilindros simples cuyo movimiento se desarrolle verticalmente y los cuales arrastren consigo a otros actuadores, como es el caso del actuador encargado de girar los ejes.

```
public void display(int desplazamiento,Point posManipulador, Point rotManipulador){ // VERTICAL
    /*
    *Polimorfismo del método display para poder interactuar con el manipulador.
    *
    */
    root.pushMatrix();
    root.translate(posManipulador.x, posManipulador.y+posBase.y, posManipulador.z);
    root.rotateX(rotManipulador.x);
    root.rotateY(rotManipulador.y);
    root.rotateZ(rotManipulador.z);

    root.shape(movil, 0+desplazamiento, 0-pos);
    root.shape(base, desplazamiento, 0);
    pos += vel_temp;

    root.popMatrix();

}
```

Figura 46 Método displayMovVert de la clase CilindroAuxiliar.java

En la Figura 47 aparece el polimorfismo del método display() ya comentado. En este caso, este polimorfismo se ha creado para realizar actuaciones horizontales, es decir, para cilindros que se expanden y retraen horizontalmente. Es por esto que la diferencia entre el polimorfismo para vertical y horizontal reside en el eje en el que se añade la variable pos, que como ya se ha comentado, será la encargada de aumentar o disminuir en función de la acción que tenga que desarrollar el cilindro en cuestión.

```
public void display(Point posManipulador, Point rotManipulador){ // HORIZONTAL
    /*
    *Polimorfismo del método display para poder interactuar con el manipulador.
    *
    */
    root.pushMatrix();
    root.translate(posManipulador.x, posManipulador.y, posManipulador.z);
    root.rotateX(rotManipulador.x);
    root.rotateY(rotManipulador.y);
    root.rotateZ(rotManipulador.z);

    root.shape(movil, 0+pos, 0);
    root.shape(base, 0f, 0f);
    pos += vel_temp;

    root.popMatrix();
}
```

Figura 47 Polimorfismo del método display para actuaciones horizontales de la clase CilindroAuxiliar.java

3.2.2.4. Manipuladores

Estas clases son el claro ejemplo de la combinación de varios cilindros como se ha explicado anteriormente.

En el caso de la clase Manipulador.java, se ha creado para representar el manipulador responsable de la expulsión de ejes incorrectos, que puede observarse en la Figura 24.

Este actuador está formado por dos cilindros simples, uno horizontal y otro vertical. Estos cilindros serán los atributos del manipulador y pertenecerán a la clase Manipulador, tal y como aparece en la Figura 48.

```
public class Manipulador {
    private GemeloDigital root;
    private Map<String, Object>params;
    private Point posBase;
    private Point rotBase;
    private int escala;
    private int posVert;
    private CilindroAuxiliar ManipVert;
    private CilindroAuxiliar ManipHoriz;
```

Figura 48 Atributos de la clase Manipulador.java

Esta clase que, en cierto modo unifica otras, también posee su propio método setup(), display() y update() que se muestran en la Figura 49.

```
public void setup(){
    ManipHoriz.setup();
    ManipVert.setup();
}

public void display(){
    ManipHoriz.display(posBase, rotBase);
    ManipVert.display(ManipHoriz.getPos(),posBase, rotBase);
}

public void update(byte[] entradasPLC, byte[]salidasPLC){
    ManipHoriz.update(entradasPLC, salidasPLC);
    ManipVert.update(entradasPLC, salidasPLC);

}
```

Figura 49 Métodos setup(), display() y update() de la clase Manipulador.java

Es importante cerciorarse de que estos métodos se emplean para llamar a los propios, con el mismo nombre, de los cilindros auxiliares.

De esta forma, y aunándolo al hecho, ya comentado, de que los cilindros auxiliares se representan en función de las coordenadas del fichero de configuración del actuador, a la hora de trabajar con manipuladores, en la clase principal se hará referencia a la clase general, es decir, a la clase Manipulador.

Por último, esta clase implementa un método llamado moverEje que se muestra en la Figura 50.

```
public void moverEje(Eje eje, Boolean mat, Boolean posicion){
            root.rotateY(rotBase.y+(PI));
            root.translate(-ManipHoriz.getPos(),-ManipVert.getPos(),0);
            eje.display(mat, posicion, 47+7, 113+40, -35, 0, 0, 0);
    }
```

Figura 50 Método MoverEje de la clase Manipulador.java

Este método se ha creado para representar gráficamente el movimiento del eje cuando es succionado por la ventosa del extremo del manipulador. De esta forma, cuando en la clase principal se llame a este método (se comentará posteriormente) será la clase del manipulador la encargada de dibujar el eje e ir variando su posición de la misma forma que varían las posiciones del cilindro vertical y horizontal. Se genera así un efecto que hace parecer que la ventosa realmente sujeta al eje mediante succión.

La arquitectura de la clase ManipuladorGiro.java, frente a la clase anterior, es totalmente análoga. Las diferencias más significativas aparecen en los atributos y la función encargada de dibujar el eje cuando este está sujeto por el actuador.

La diferencia en los atributos reside en que, en este caso, esta clase se utiliza para representar el actuador encargado de girar los ejes y colocarlos en la posición correcta. Este actuador puede observarse en la Figura 22. Es por esto que se tendrá un atributo de tipo CilindroAgarre, otro de tipo CilindroAuxiliar y otro de tipo CilindroGiro. Véase Figura 51.

```
public class ManipuladorGiroEje {
      private GemeloDigital root;
    private Map<String, Object>params;
    private Point posBase;
    private Point rotBase;
    private int escala;
    private int posVert;
    private CilindroAuxiliar cilindroVertical;
    private CilindroGiro cilindroGiro;
    private CilindroAgarre pinzas;
    boolean girarEje=false;
    private boolean ejeSujeto=false;
```

Figura 51 Atributos de la clase ManipuladorGiroEje.java

Además de esto, el método encargado de mover el eje también es distinto. Esto se debe a que, en este caso, además de mover el eje verticalmente, también habrá que rotarlo sobre sí mismo en el eje Z. Esto puede apreciarse en la Figura 52 que se muestra a continuación.

```
public void girarEje(Eje eje, Boolean mat, Boolean posicion){
            root.translate(50,-cilindroVertical.getPos()+240,0);
            root.rotateZ(-cilindroGiro.getGiro()*PI/180);
            eje.display(mat, posicion, 47-50, 113-190+60, -120, 0, 0, 0);
            if(cilindroGiro.indicadorGirado.estado && cilindroVertical.indicadorRetraido.estado){
                girarEje = true;
            }
    }
```

Figura 52 Método girarEje de la clase ManipuladorGiro.java

La clase CilindroInsercion.java ha sido especialmente diseñada para la representación del actuador encargado de insertar los ejes correctos en el ensamblaje. Este actuador, que puede verse en la Figura 25 presenta una singularidad; y es que, aunque al igual que el manipulador de desecho de ejes, presenta dos sistemas de movimiento distinto, pero con la diferencia, de que en este caso es una misma pieza la que realiza ambos movimientos. Es decir, en el caso del actuador aparecían dos cilindros independientes, uno se expandía y retraía verticalmente, y el otro lo hacía horizontalmente. En este caso, también existen dos movimientos, pero realizados por la misma pieza, que forma parte del conjunto completo del actuador en cuestión. Uno de los movimientos consiste en un desplazamiento vertical, mientras que el otro consiste en una rotación alrededor del eje Y. Al igual que en los casos anteriores, se codifica un método que permite que sea el actuador el que dibuje y desplace al eje correspondiente siguiendo el movimiento de este.

3.2.2.5. Ejes

Eje.java es otra de las clases más elementales que forman parte de este proyecto. Como su propio nombre indica, es la responsable de la actualización y representación de los ejes.

En la Figura 53 aparecen los atributos de la clase Eje. Como ya se comentó en el apartado relativo a la estación FMS204, hay que tener en cuenta que los ejes pueden ser de plástico, de metal, estar en la posición correcta o estar en una posición incorrecta (rotados 180º).

```
public class Eje {
    GemeloDigital root;
    private PShape ejePlastico;
    private PShape ejeMetal;
    private float escala = 0.0f;

    private boolean material; //  false:PLÁSTICO//true:METAL
    private boolean posicion; //  false:CORRECTO//true:INCORRECTO
    private float movx;
    private float movy;
    private float movz;

    public Eje (GemeloDigital r, boolean mat, boolean pos){
        root = r;
        this.material = mat;
        this.posicion = pos;
    }
```

Figura 53 Atributos de la clase Eje.java

Para representar todas estas características, cada objeto de tipo Eje tendrá un atributo de tipo booleano referente al material, que será false si el eje es de plástico, o true si es de metal. Tendrá también otro atributo de tipo booleano referente a la posición, que será false cuando sea correcta y true cuando sea incorrecta. Aunque esta última analogía puede resultar algo contradictoria, se ha definido así para solventar un inconveniente que se comentará en epígrafes posteriores.

Al igual que en clases anteriores, la clase eje también implementa un método display para representarlo en el espacio de trabajo. El código fuente de este método aparece en la Figura 54.

```
public void display(boolean mat, boolean pos, float x, float y, float z, float rotx, float roty, float rotz){
    root.pushMatrix();
    if (pos == true){
    root.translate(x+movx, y+20+movy, z+movz);
    root.rotateZ(rotz+PI);
    }else{
    root.translate(x, y, z);
    root.rotateZ(rotz);
    }
    root.rotateX(rotx);
    root.rotateY(roty);

    if (mat == false){
        root.shape(ejePlastico, 0, 0);
    }else{
        root.shape(ejeMetal, 0, 0);
    }

    root.popMatrix();
}
```

Figura 54 Método display() de la clase Eje.java

3.2.2.6. Elementos fijos

La clase Elementofijo.java, como su nombre indica, se utiliza para la representación de los elementos fijos que integran la estación FMS204.

Si se analizan los componentes que forman parte de la estación, se observa que gran parte de ellos son elementos estáticos que no están dotados de ningún tipo de movimiento.

Por una parte, están los elementos de soporte como la mesa sobre la que se encuentran anclados los actuadores de la estación. Pero, por otra parte, también existen actuadores que, aunque cumplen un papel muy importante, su funcionamiento no está ligado al movimiento. Ejemplos de este último caso pueden ser los detectores de metal y de presencia.

```
public class ElementoFijo {
    protected GemeloDigital root;
    protected Map<String, Object> params;
    protected Point pos;
    protected Point rot;
    protected String nombre;
    protected PShape elemento;
    protected int escala;
```

Figura 55 Atributos de la clase elementoFijo.java

Respecto a la clase, es muy similar a las anteriores, en sus atributos (Figura 55) encontramos variables de tipo PShape que almacenarán los .obj que se representarán posteriormente mediante el método display() () teniendo en cuenta los parámetros leídos del fichero de configuración correspondiente, siguiendo la metodología ya explicada.

```
public void setup(){
elemento = root.loadShape(nombre);
elemento.scale(escala);

}

public void display(){
    root.pushMatrix();
    root.translate(pos.x, pos.y, pos.z);
    root.rotateX(rot.x);
    root.rotateY(rot.y);
    root.rotateZ(rot.z);

    root.shape(elemento, 0, 0);

    root.popMatrix();
}
```

Figura 56 Métodos steup() y display() de la clase elementoFijo.java

3.2.2.7. Plato

La clase Plato.java es de gran importancia ya que contiene una de las metodologías clave para el funcionamiento de la estación. Tal y como indica su nombre, la clase Plato.java es la encargada de dibujar y controlar los movimientos del plato giratorio pero también la información relativa a los ejes que se encuentran en él, es decir, gracias a esta clase se lleva un control del número de ejes que hay sobre el plato, de las posiciones que estos ocupan, del tipo de material de cada eje y de la posición de los mismos (rotación correcta o incorrecta).

Para conseguir esto, se han creado tres array de booleanos que, junto a las demás variables típicas que se han ido explicando en clases posteriores, forman parte de los atributos de la clase, véase la Figura 57.

```
public class Plato {
    boolean posicionOcupada[]={false, false, false, false, false,
        false, false, false}; //  false:VACÍA//true:OCUPADA
    boolean material[]={false, false, false, false, false, false,
        false, false}; //  false:PLÁSTICO//true:METAL
    boolean posicionEje[]={false, false, false, false, false, false,
        false, false}; //  false:CORRECTO//true:INCORRECTO

    private GemeloDigital root;
    private Map<String, Object> params;
    private Point posPlato;
    private Point rotPlato;
    private String nombrePlato;
    private String ejeGiro;
    private PShape plato;
    private int escala;
    private int velGiro;
    private int velGiro_temp;
    private int giroMax;
    private int giroMin;
    private int giro;
    private int byteGirar;
    private int dirGirar;
    private int byteVolver;
    private int dirVolver;
    private int byteGirado;
    private int dirGirado;
    private int byteVuelto;
    private int dirVuelto;
    private Indicador indicadorGirado;
    private Indicador indicadorVuelto;
```

Figura 57 Atributos de la clase Plato.java

Cada uno de estos array es de dimensión ocho, esto se debe a que en el plato giratorio solo existen ocho posiciones posibles (ocho huecos donde colocar ejes). De esta forma, el primer elemento del array se corresponde con la posición 0, que es el último hueco del plato que no está asociado a ningún actuador. El segundo elemento del array se corresponde con el hueco referente al actuador de inserción de eje correcto, y así sucesivamente en sentido horario. En la Figura 58 se puede observar gráficamente qué actuador del plato, escrito en negro, coinciden con qué posición del vector, escrito en rojo. Se considera que las posiciones de los array siguen el siguiente orden:

material[] = {0,1,2,3,4,5,6,7}

Figura 58 Posiciones del plato correspondientes a cada elemento de los array de información

El nombre de cada uno de estos array hace referencia a la información que contiene, de esta forma, posicionOcupada registrará la información referente a qué cavidades del plato están ocupadas, material hace referencia al tipo de material de cada eje y posicionEje hace referencia a si el eje en cuestión está correctamente colocado o si, por el contrario, es necesario girarlo. De esta forma, y siguiendo la misma lógica que en los atributos de la clase Eje.java, un elemento true en el vector de posición ocupada significará que hay un eje en esa posición. Un elemento true en el array de tipo de material significará que el eje en esa posición es de metal, de lo contrario, será de plástico o nylon. Por último, si un elemento del array de posición del eje es false, significará que el eje se encuentra en la posición correcta y, si es false, será necesario girarlo 180º en el actuador encargado de esto. El motivo de esta decisión se ha basado en la funcionalidad de la estación; el hecho de que un eje correctamente posicionado se ligue a un false, se debe a que, si la posición 6 (comprobación de posición) no tiene un eje, y se activa el cilindro, este llegará al final de carrera indicando, en cierto modo, que el supuesto eje, si lo hubiera, está en la posición correcta. Como los array descritos se inicializan con todos sus elementos a false, es importante respetar esta analogía.

En esta clase vuelven a aparecer los métodos ya descritos anteriormente; setup(), display() y update(), cuya programación no difiere del resto. Sin embargo, la clase Plato.java implementa dos métodos únicos de esta clase y que tienen un papel muy importante.

En la Figura 59 aparece el método EcharEje(), que, como su nombre indica, es el que se encarga de generar aleatoriamente un eje y dibujarlo en la posición 7 (Figura 58).

```
public void echarEje(){

    boolean copiaPosOc[] = new boolean[8];
    boolean copiaMaterial[] = new boolean[8];
    boolean copiaPosEj[] = new boolean[8];

    int mat = (int)root.random(2);
    int pos = (int)root.random(2);

    copiaPosOc = Arrays.copyOf(this.getPosicionOcupada(),8);
    copiaMaterial = Arrays.copyOf(this.getMaterial(),8);
    copiaPosEj = Arrays.copyOf(this.getPosicionEje(),8);

    if ((mat == 0)&&(pos == 0)) {
        copiaMaterial[7] = false;
        copiaPosEj[7] = false;
    } else if ((mat == 0)&&(pos == 1)) {
        copiaMaterial[7] = false;
        copiaPosEj[7] = true;
    } else if ((mat == 1)&&(pos == 0)){
        copiaMaterial[7] = true;
        copiaPosEj[7] = false;
    } else{
        copiaMaterial[7] = true;
        copiaPosEj[7] = true;
    }

    copiaPosOc[7] = true;

    this.setPosicionOcupada(copiaPosOc);
    this.setMaterial(copiaMaterial);
    this.setPosicionEje(copiaPosEj);
```

Figura 59 Método EcharEje de la clase Plato.java

El otro método que se diferencia de los ya explicados es el que se muestra en la Figura 60. Es el encargado de actualizar los datos de los array cada vez que el plato gira ya que, cuando esto sucede, el eje que estaba en la posición siete pasa a

estar en la seis, el de la seis en la cinco y así sucesivamente. Esto provoca que sea necesario que los datos del array de material, posición ocupada y posición del eje también tengan que rotar.

```
public void girarPlato(){
    boolean copiaPosOc[] = new boolean[8];
    boolean copiaMaterial[] = new boolean[8];
    boolean copiaPosEj[] = new boolean[8];

    copiaPosOc = Arrays.copyOf(this.getPosicionOcupada(),8);
    copiaMaterial = Arrays.copyOf(this.getMaterial(),8);
    copiaPosEj = Arrays.copyOf(this.getPosicionEje(),8);

    for (int i=0;i<=6;i++){
        copiaPosOc[i] = copiaPosOc[i+1];
        copiaMaterial[i] = copiaMaterial[i+1];
        copiaPosEj[i] = copiaPosEj[i+1];
    }
    copiaPosOc[7] = false;
    copiaMaterial[7] = false;
    copiaPosEj[7] = false;

    this.setPosicionOcupada(copiaPosOc);
    this.setMaterial(copiaMaterial);
    this.setPosicionEje(copiaPosEj);
```

Figura 60 Método GirarPlato() de la clase Plato.java

3.2.2.8. Botonera

La clase BotoneraFrame.java es muy distinta a todas las presentadas anteriormente, ya que se trata de un JFrame Form.

Un JFrame en Java es una clase que pertenece a la librería gráfica Swing, y se utiliza para generar ventanas emergentes en las que se añaden objetos como pulsadores, etiquetar, botones de selección etc.

En este proyecto se han creado dos JFrame distintos. En primer lugar, se ha creado el llamado BotoneraFrame.java, que se muestra en la Figura 61.

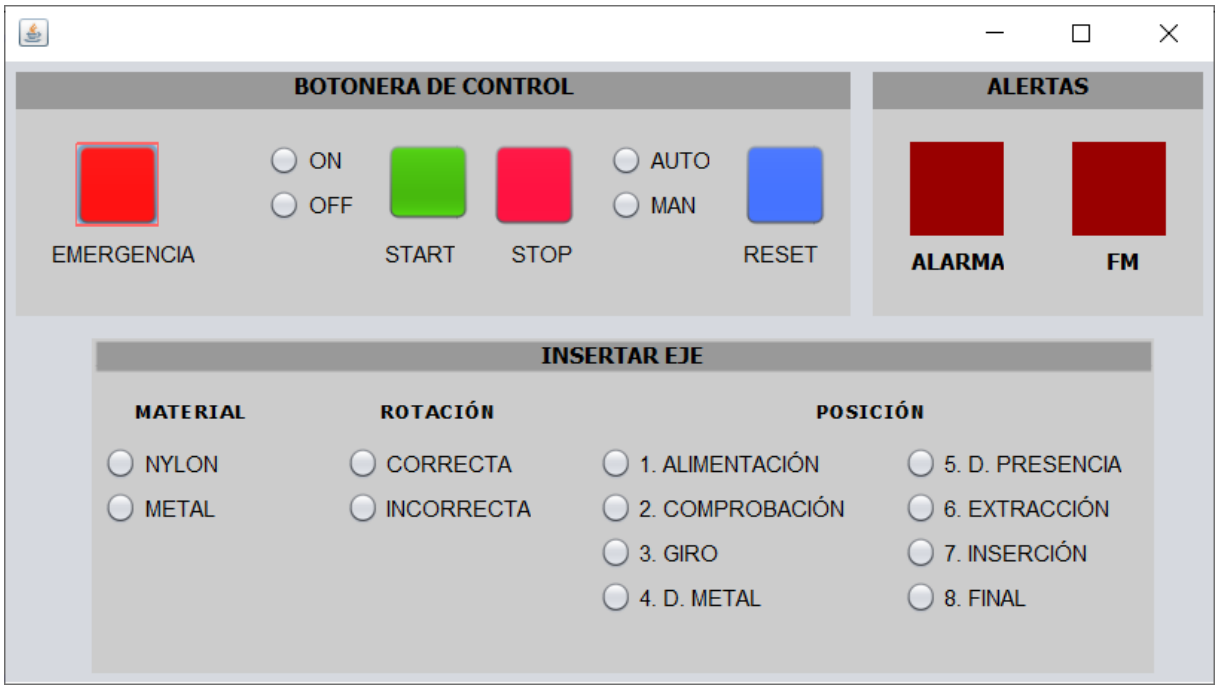


Figura 61 Botonera

El objetivo de esta ventana es, por una parte, simular la botonera que lleva incluida la estación, y que puede verse en la Figura 15 y, por otra parte, permitir al usuario introducir cualquier tipo de eje en cualquiera de las posiciones de la estación.

3.2.2.9. Panel de Control

El otro JFrame que se comentaba en el epígrafe anterior es el que aparece en la Figura 62; que se corresponde con la clase ControlFrame.java.

Esta ventana ha sido creada con la intención de permitir al usuario realizar pruebas independientemente de la programación del PLC. De esta forma, clicando sobre cualquier check box el actuador que se corresponda con esa letra, se moverá tal y como lo haría si se activara la entrada del PLC correspondiente.

Para identificar cada actuador con la letra que le corresponde se recomienda utilizar la Tabla 1 y la Tabla 2.

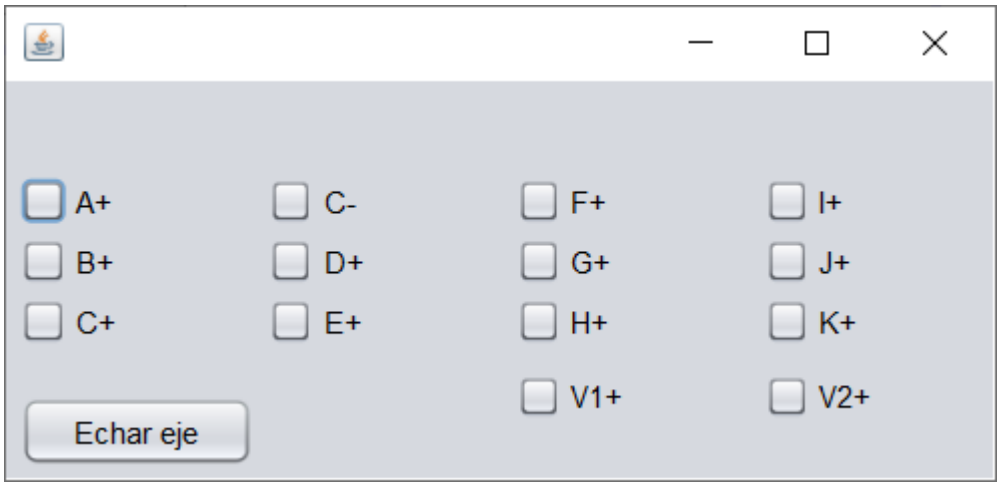


Figura 62 Panel de control

3.2.2.10. Gemelo Digital

La clase GemeloDigital.java es la clase principal del proyecto, por lo que es en ella donde se invocan, cuando procede, todos los métodos que se han ido explicando durante este capítulo.

En primer lugar, se declaran todas las variables que posteriormente serán necesarias para la programación de la estación.

```
public class GemeloDigital extends PApplet {
    private int ancho;
    private int alto;
    private String ipAddress;
    private List<String> nombreCilindros;
    private List<String> nombrePulsadores;
    private List<Cilindro> cilindros;
    private List<Pulsador> pulsadores;
    private List<String> nombreManipuladores;
    private List<Manipulador> manipuladores;
    private List<String> nombreCilindroInsercion;
    private List<CilindroInsercion> cilindroInsercion;
    private List<String> nombreManipuladoresGiroEje;
    private List<ManipuladorGiroEje> manipuladoresGiroEje;
    private List<String> nombrePlato;
    private List<Plato> plato;
    private List<String> nombreElementosFijos;
    private List<ElementoFijo> elementosFijos;

    private int dirBaseEntradas;
    private int offsetBaseEntradas;
    private byte entradasPLC[];
    private int dirBaseSalidas;
    private int offsetBaseSalidas;
    private byte salidasPLC[];
```

Figura 63 Declaración de variables en la clase GemeloDigital.java

Algunas de ellas se muestran en la Figura 63, y serán variables cuyo valor dependerá de la lectura del fichero config.yml, que aparece en la Figura 64.

```yaml
ancho: 1500
alto: 800
PLC: SIEMENS
ipAddress: 127.0.0.1
rack: 0
slot: 1
byteEntradas: 2
offsetEntradas: 5
byteSalidas: 2
offsetSalidas: 5

## Parametros Cilindros
nombreCilindros:
  - Config\\cilindroF.yml
nombreManipuladores:
  - Config\\manipulador.yml
nombreCilindroInsercion:
  - Config\\cilindroAB.yml
nombreManipuladoresGiroEje:
  - Config\\manipuladorGiroEje.yml
nombrePulsadores:
  - Config\\pulsadorA.yml
  - Config\\pulsadorB.yml
  - Config\\pulsadorCMas.yml
  - Config\\pulsadorCMenos.yml
  - Config\\pulsadorD.yml
  - Config\\pulsadorE.yml
  - Config\\pulsadorF.yml
  - Config\\pulsadorG.yml
  - Config\\pulsadorH.yml
  - Config\\pulsadorI.yml
  - Config\\pulsadorJ.yml
  - Config\\pulsadorK.yml
nombrePlato:
  - Config\\plato.yml
nombreElementosFijos:
  - Config\\mesa.yml
  - Config\\alimentacion.yml
  - Config\\detectorMetal.yml
  - Config\\detectorPresencia.yml
```

Figura 64 Fichero config.yml

Si se observa el fichero config.yml con atención, puede apreciarse cómo se han creado distintas listas, una para cada clase explicada anteriormente. Esto es así porque, si se sigue analizando la clase GemeloDigital.java, aparece una función, de nombre leerParametros, que será la encargada de crear un array list por cada tipo

de elemento existente y, posteriormente, leer la lista correspondiente, guardando cada una de las rutas de la Figura 64 como un elemento de ese array list. En la Figura 65 puede verse un ejemplo de esto.

```
nombreCilindros = (List<String>) data.get("nombreCilindros");
System.out.println(nombreCilindros);
cilindros =new ArrayList<>();
for (String str : nombreCilindros) {
    System.out.println(str);
    cilindros.add(new Cilindro(this, str));
}
```

Figura 65 Ejemplo de creación de un array list para cilindros

Una vez realizado esto para todas las listas del fichero de configuración, aparece la función settings() de la Figura 66. En ella se especifica la ruta donde se almacena el archivo config.yml. Además, se realiza un try catch [35] para que, en caso de no encontrar el archivo, se capture la excepción. Por último, se establecen los valores de alto y ancho que tendrá el espacio de trabajo en el que se representará la estación. Estos valores se obtienen del archivo de configuración.

```
@Override
public void settings() {
    try {
        leerParametros("C:\\Users\\Maria Alvarez\\Desktop\\TFG\\0000PROGRAMA\\digital-twin-master\\config\\config.yml");
    } catch (FileNotFoundException ex) {
        Logger.getLogger(GemeloDigital.class.getName()).log(Level.SEVERE, null, ex);
    }
    size(ancho, alto,P3D);// P3D); TODO explorar formato FX2D

}
```

Figura 66 Función settings() de la clase GemeloDigital.java

Después de esto, en la función setup() de la Figura 67, se utilizan varios ciclos for para recorrer cada uno de los array list creados, de forma que se llama a la función setup() de cada objeto de cada clase. Además, se llama al método Connect() de la clase PLCSiemens que permite la comunicación con el PLC virtual.

```
@Override
public void setup() {
    for (Cilindro cil : cilindros) {
        cil.setup();
    }

    for (Manipulador manip : manipuladores){
        manip.setup();
    }
    for (CilindroInsercion cilIn : cilindroInsercion){
        cilIn.setup();
    }

    for (ManipuladorGiroEje manipGE : manipuladoresGiroEje){
        manipGE.setup();
    }

    for (Plato manipPlato : plato){
        manipPlato.setup();
    }

    for (ElementoFijo manipElementoFijo : elementosFijos){
        manipElementoFijo.setup();
    }

    nuevoEje.setup();
    girox=0;
    giroy=0;

    plc.Connect();
}
```

Figura 67 Función setup() de la clase GemeloDigital.java

Por último, en la clase principal, se implementa la función draw(). En ella, se leen las salidas del OLC y se actualizan las entradas con las funciones de la Figura 68 y la Figura 69 respectivamente.

```
//Leer salidas PLC
plc.Client.ReadArea(Moka7.S7.S7AreaPA, 0, dirBaseSalidas,offsetBaseSalidas,salidasPLC );
```

Figura 68 Función para la lectura de las salidas del PLC

```
// Actualizar entradas PLC.
plc.Client.WriteArea(Moka7.S7.S7AreaPE, 0, dirBaseEntradas,offsetBaseEntradas,entradasPLC );
```

Figura 69 Función para la actualización de las salidas del PLC

También se actualizan las posiciones de los objetos. Esto se hace de forma análoga a la explicada en la función setup() pero, en este caso, se invoca a los métodos update() de los objetos, que ya fueron explicados. Se muestra un ejemplo para los cilindros en la Figura 70.

```
//Actualizar posiciones objetos
for (Cilindro cil : cilindros) {
    cil.update(entradasPLC,salidasPLC);
}
```

Figura 70 Actualización de la posición de los cilindros mediante los métodos udate()

Además, siguiendo la misma metodología, se dibuja el entorno de trabajo. Cabe destacar el método utilizado para dibujar los ejes correctos en el plato; para conseguirlo, se hace uso de un ciclo for que recorre los tres array de los atributos de la clase plato, de esa forma, va comprobando y dibujando posición por posición los ejes que corresponden. Se recomienda consultar el código fuente.

4. MANUAL PARA EL USUARIO

Este manual pretende ser una guía de uso donde se detallarán todos los pasos a seguir para hacer uso del gemelo digital y se explicarán todas las funciones que ofrece.

En primer lugar, se ejecutarán TIA Portal, NetBeans y NetToPLCsim. Es muy importante que este último se ejecute como administrador.

Una vez abierto TIA Portal, es elección del usuario abrir un proyecto existente o crear uno nuevo. Para que la comunicación funcione, es necesario permitir el acceso vía comunicación PUT/GET del interlocutor remoto. Esto se hace accediendo, desde la vista de proyecto, a la pestaña del árbol del proyecto de la izquierda llamada Dispositivos y redes y accediendo al PLC que se haya configurado (Figura 71).

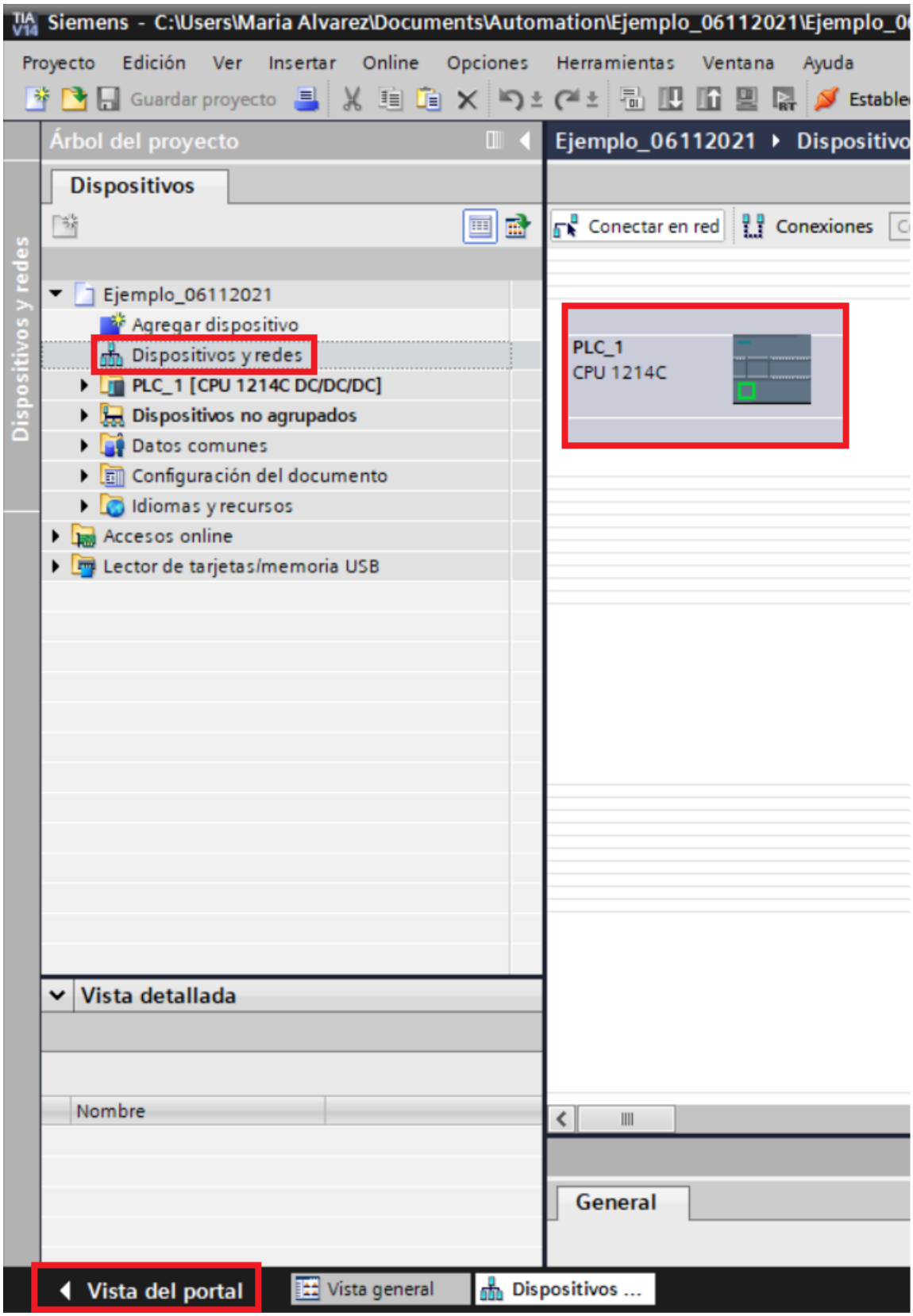


Figura 71 Configuración de TIA Portal

Una vez hecho esto, aparecerá la ventana que se muestra en la Figura 72.

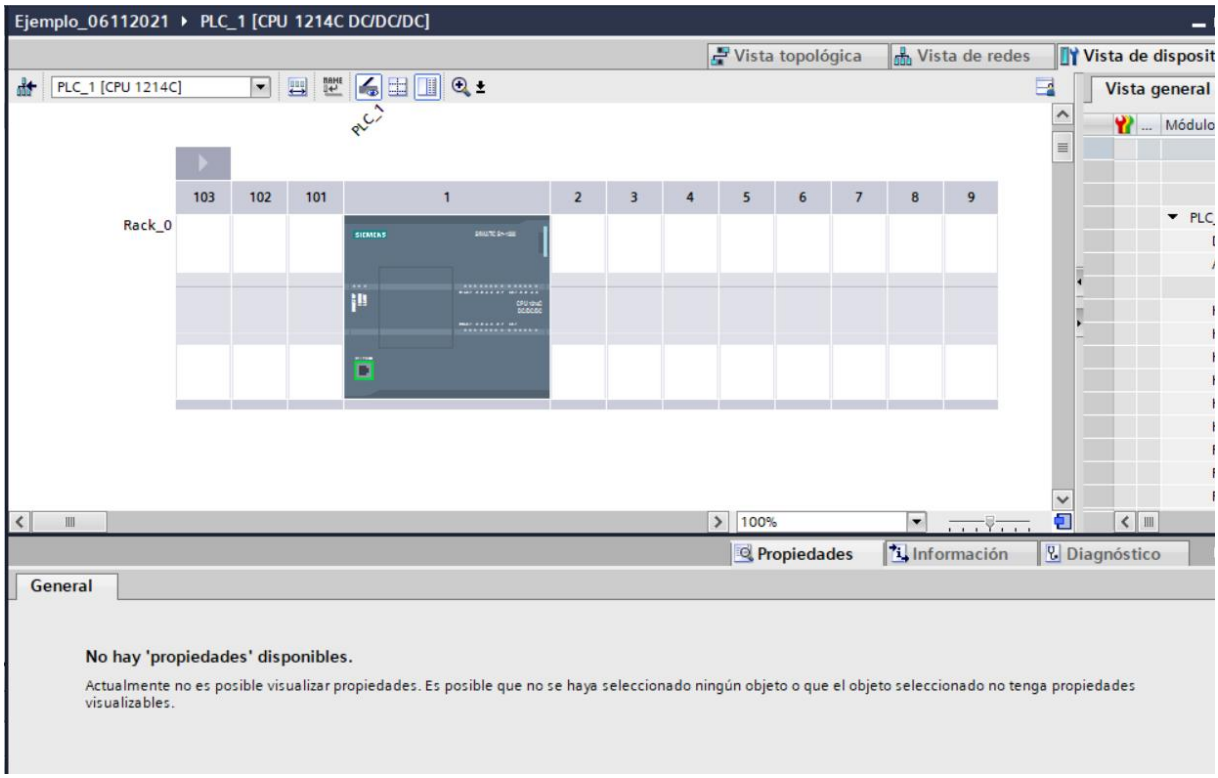


Figura 72 Dispositivos y redes en TIA Portal

Lo siguiente será clicar una vez sobre el PLC y se observará que la ventana inferior cambia. Por último, en la pestaña General a la izquierda de la ventana inferior y en Propiedades, a la derecha, se accede al submenú Protección y Seguridad, donde se encontrará la opción de permitir el acceso. Todo esto puede verse en la Figura 73.

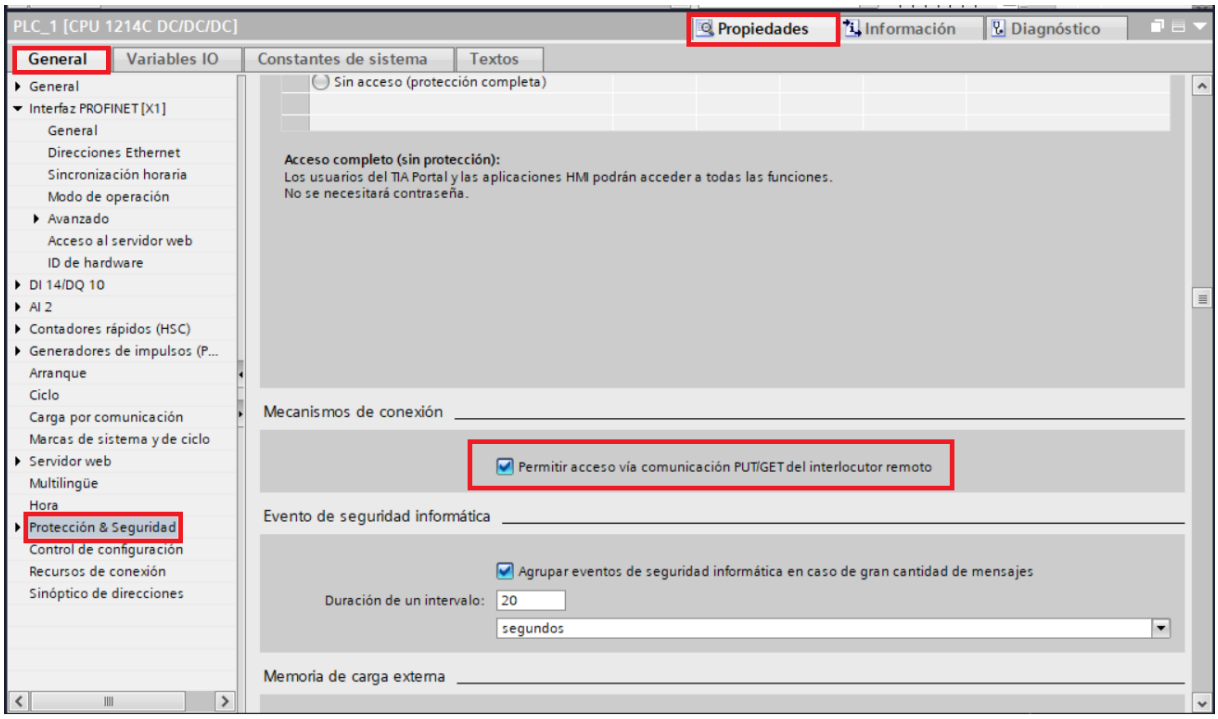


Figura 73 Configuración de acceso TIA Portal

Cuando se haya permitido el acceso, se procederá a iniciar el simulador S7-PLCSIM. Esto se hará desde la interfaz de TIA Portal, clicando sobre el símbolo de la barra de herramientas superior que aparece en la Figura 74.

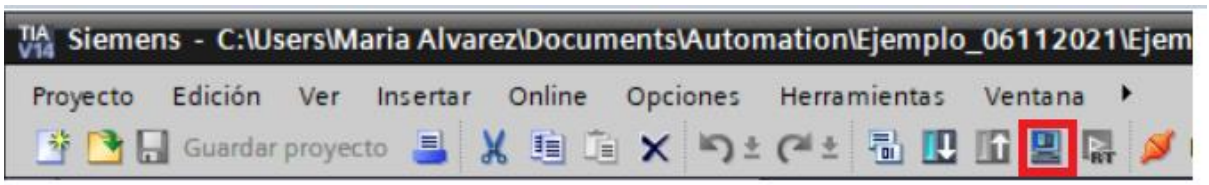


Figura 74 Iniciar simulación en TIA Portal

Después de esto se cargará el proyecto tal y como se haría con un PLC no virtual.

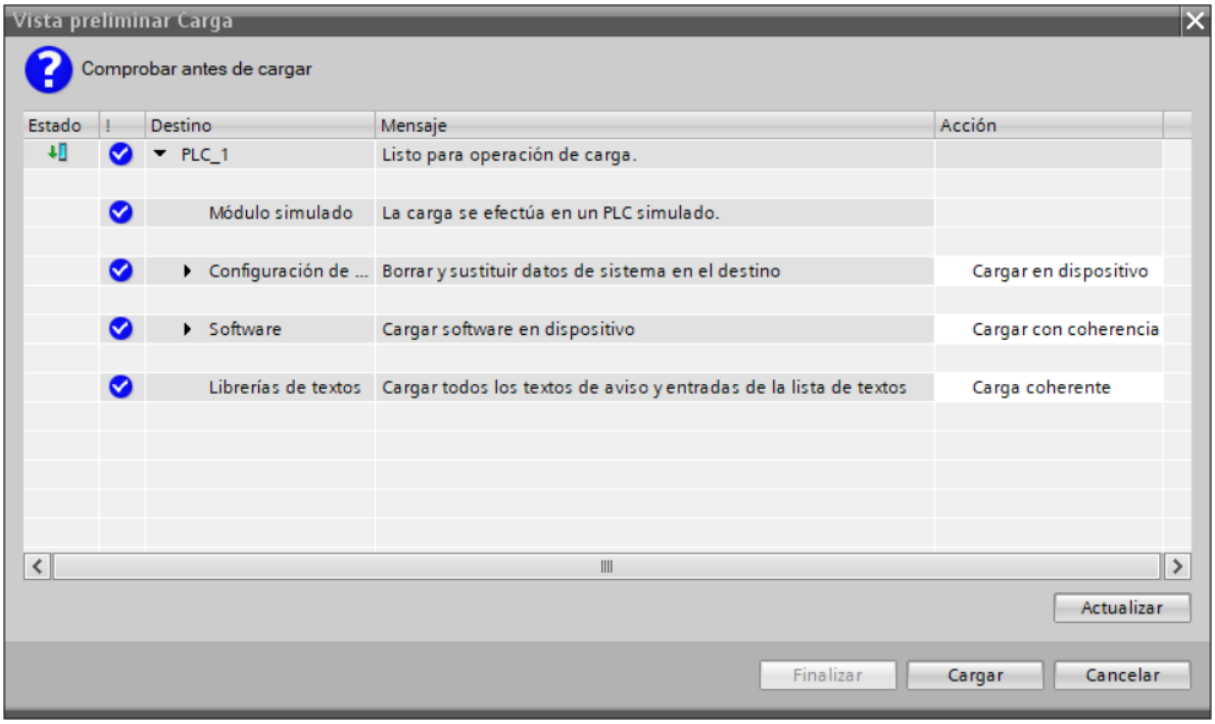


Figura 75 Cargar proyecto en PLC virtual

Aparecerá también una ventana del simulador que nos permitirá conocer el estado del PLC virtual en todo momento. Se muestra en la Figura 76

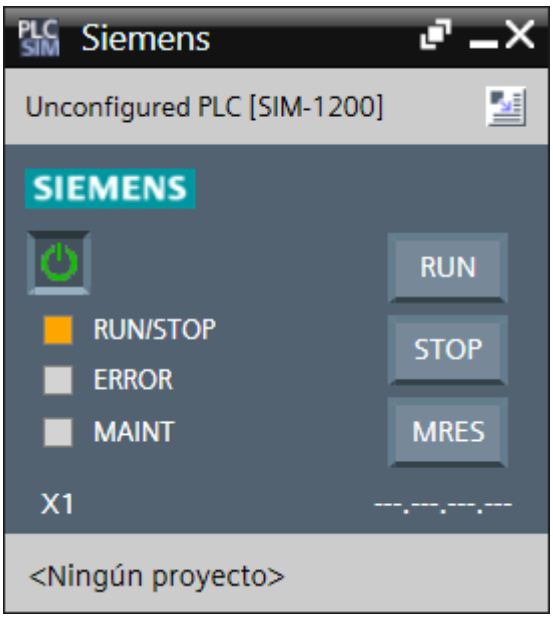


Figura 76 PLC virtual

El siguiente paso será configurar el software NetToPLCsim. Después de ejecutarlo como administrador aparecerá una ventana emergente de aviso, como la

de la Figura 77 ,indicando que el puerto 102 está en uso. Se para el servicio clicando en Sí.

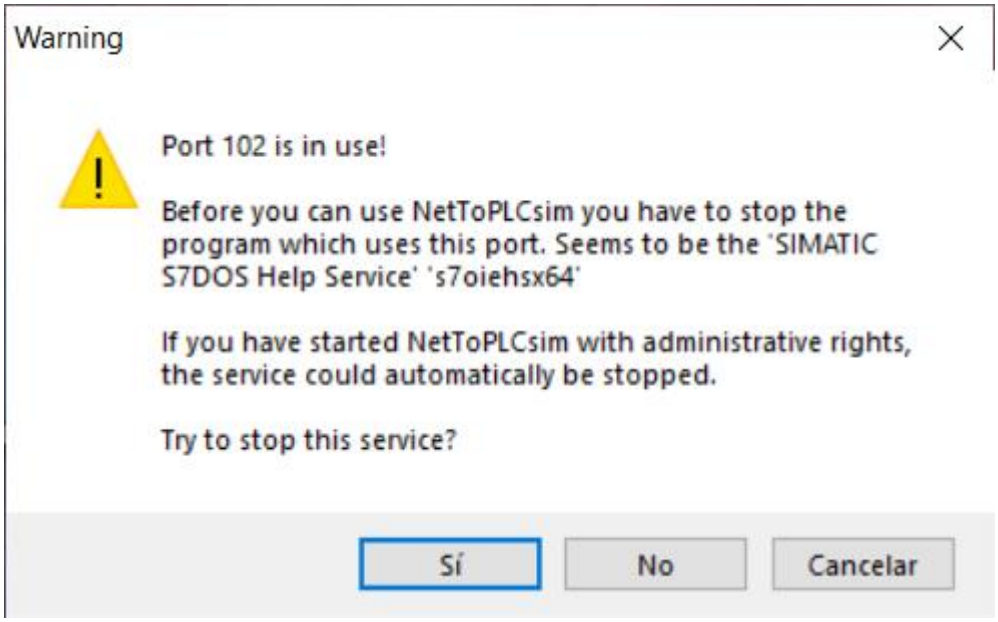


Figura 77 Ventana de alerta puerto 102 en uso

Una vez completado esto, se pulsa sobre add (Figura 78) para añadir un servidor.

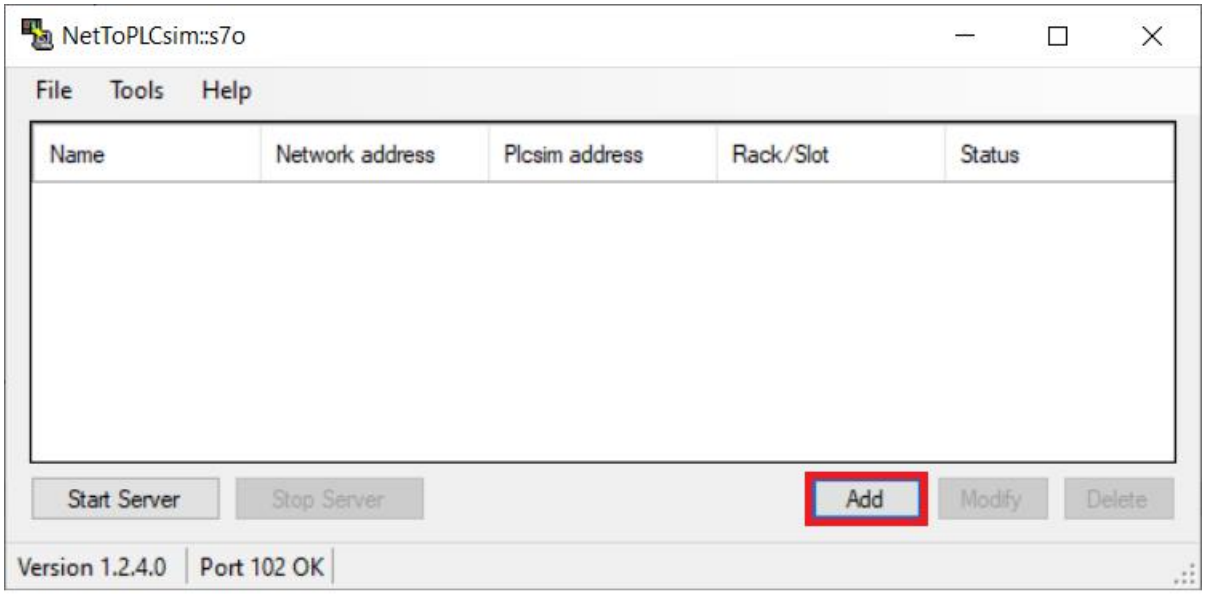


Figura 78 Método para añadir servidor en NetToPLCsim

Lo siguiente será completar los campos de la ventana emergente para configurar el servidor. Normalmente los parámetros a introducir serán los mostrados en la Figura 79 , aunque hay que tener en cuenta que si se introduce alguna

modificación en TIA Portal; nombre del PLC, número de rack y/o slot etc., deberá modificarse también el servidor.

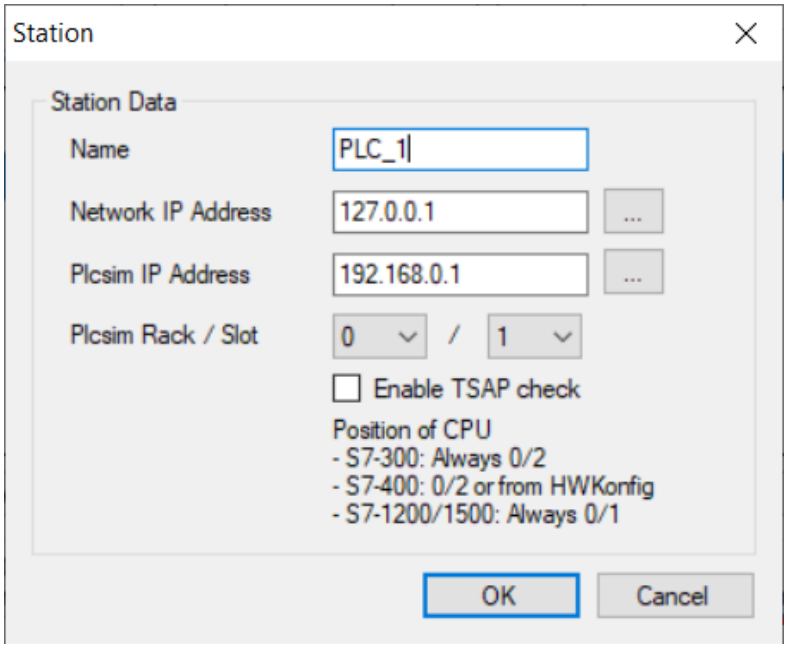


Figura 79 Configuración del servidor en NetToPLCsim

Después de esto se selecciona Ok y desparecerá la ventana emergente, por lo que la ventana de la Figura 78 quedará de forma similar a la mostrada en la Figura 80. Para finalizar se hará clic sobre Start Server para iniciar el servidor que se ha configurado.

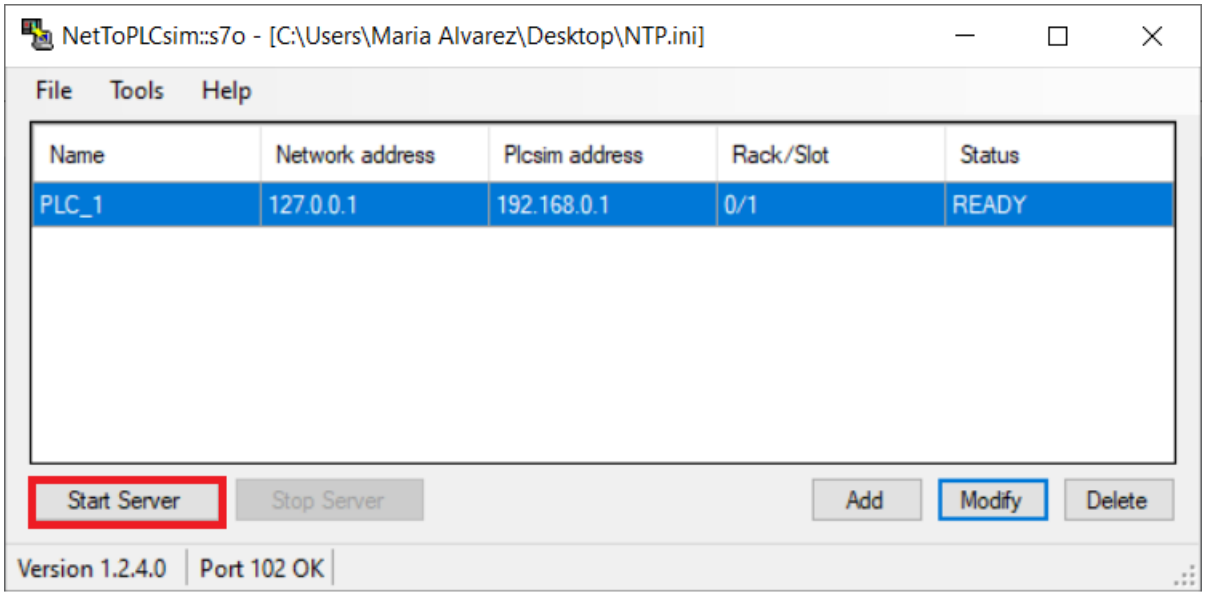


Figura 80 NetToPLCsim configurado

Para finalizar con la configuración del sistema, se ejecuta NetBeans y se abre el proyecto GemeloDigital adjunto en este proyecto. Para ejecutarlo, se pulsa F6 o, desde la barra de herramientas, se clica sobre el símbolo señalado en la Figura 81.

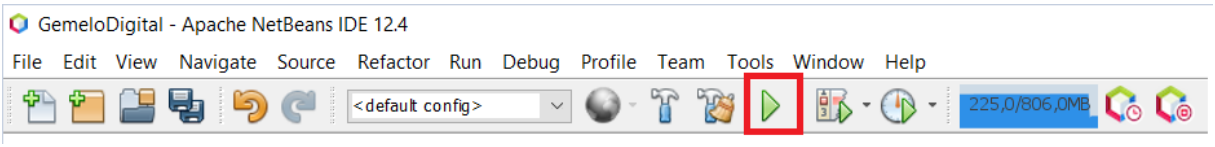


Figura 81 Ejecutar proyecto en NetBeans

Una vez hecho esto, el proyecto está listo para ser utilizado. Aparecerán tres ventanas emergentes al ejecutarlo.

En primer lugar, la botonera de la Figura 61. Esta interfaz simula la botonera incluida en la estación. Empezando por la botonera de control, de izquierda a derecha, aparece, en primer lugar, la seta de emergencia, seguidamente un selector On/Off, a continuación, dos pulsadores, start y stop, después de esto otro selector, en este caso Automático/Manual y, por último, otro pulsador, reset.

En la sección de alertas aparecen dos LED. En la Figura 81, el Led de alarma aparece encendido y el LED FM apagado, para apreciar la diferencia entre ambos.

Por último, en el panel de inserción de eje aparecen las opciones necesarias para permitir al usuario introducir cualquier tipo de eje en cualquiera de las posiciones predeterminadas de la estación.

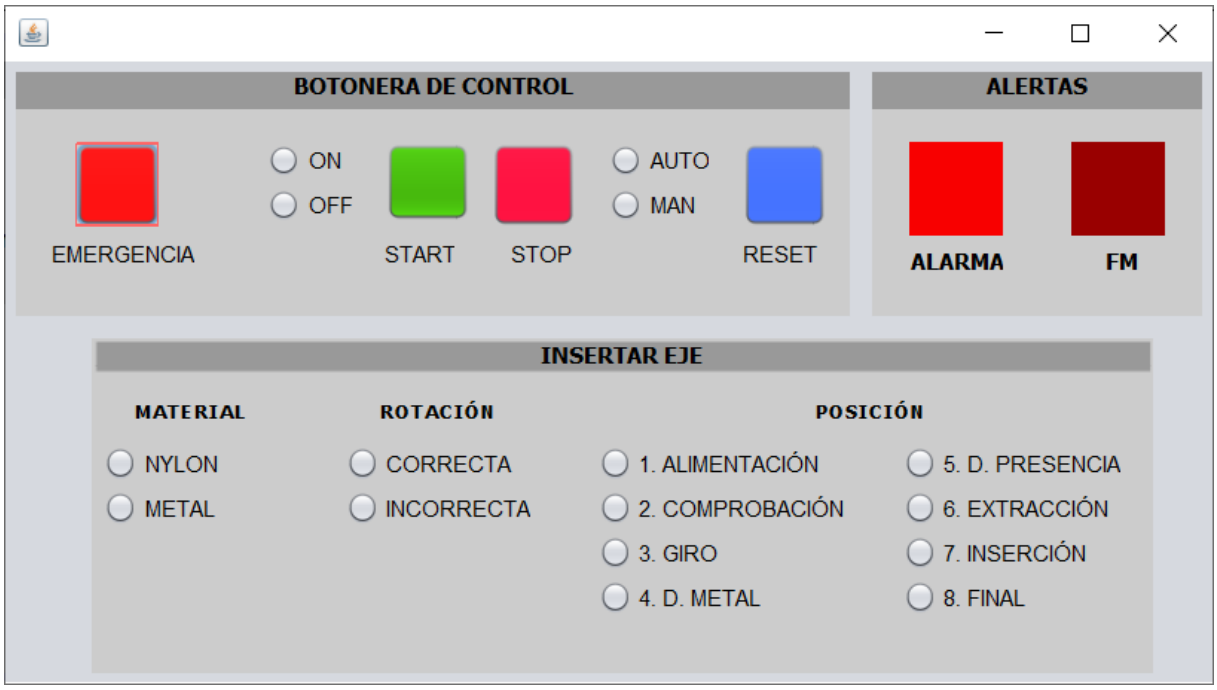


Figura 82 Botonera con el LED de alarma encendido

La segunda de las ventanas emergentes es la que se muestra en la Figura 62. Esta ventana permite al usuario realizar pruebas ya que, seleccionando las casillas, el actuador que se corresponda con la letra marcada se moverá tal y como lo haría al recibir la señal desde el PLC. De esta forma, la estación puede controlarse totalmente desde este panel de control y detectar posibles errores de comunicación o código. Para conocer la letra que corresponde a cada actuador se recomienda revidas la Tabla 1 y la Tabla 2 mostradas anteriormente.

La tercera y última ventana que aparece es el espacio de trabajo donde se simula la estación. En la Figura 83 y en la Figura 84 se puede ver el resultado de esta simulación. Para rotar la estación y cambiar el punto de vista, se coloca el ratón sobre esta ventana, se presiona el scroll del ratón y, manteniéndolo pulsado, se desplaza sobre la ventana para ir rotando la imagen.

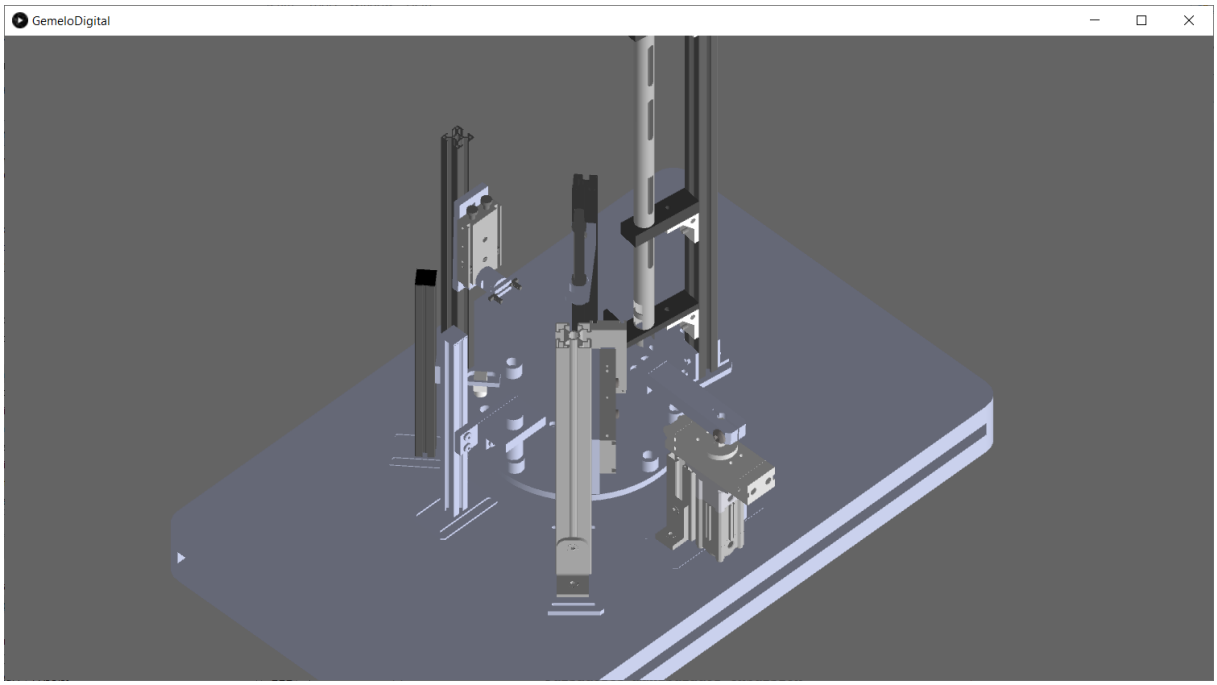


Figura 83 Imagen de la estación simulada

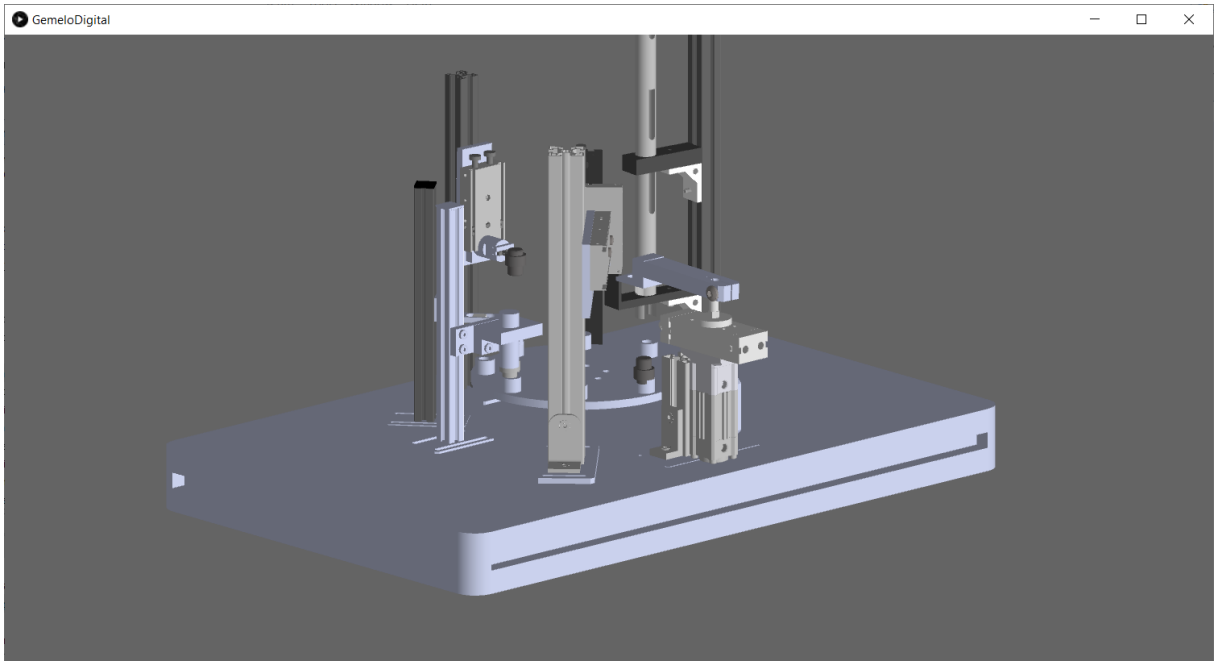


Figura 84 Imagen de la estación simulada

Por último, en el Anexo A de este proyecto, se adjunta el GRAFCET [36] de un ciclo de producción de la estación, entendiendo como ciclo todas las acciones descritas por la estación desde que un eje cae al plato giratorio desde el almacenamiento, hasta que este es insertado en el ensamble concreto, en caso de

que el eje sea correcto, o hasta que es expulsado por el manipulador en el caso de que el eje fuese incorrecto.

En el Anexo B, se adjunta el GRAFCET de la estación para un ciclo múltiple, es decir, que cada por cada vez que gira el plato, se añade un nuevo eje a la estación. De esta forma, todas las posiciones del plato estarán constantemente ocupadas.

El Anexo C incluye los planos eléctricos de la estación. Estos planos son muy útiles para conocer cómo se ha cableado la estación y, gracias a esto, saber qué señal se corresponde con cada entrada y salida del PLC.

Por último, en el Anexo D aparece un código QR que, al escanearlo, deriva en un vídeo en el que se muestra el Gemelo Digital en funcionamiento.

5. CONCLUSIONES

Una vez finalizado el proyecto, es inevitable pensar en la gran ventaja que suponen los sistemas de virtualización de plantas industriales. Tanto en la enseñanza, como en el ámbito laboral. Desgraciadamente, ha sido notorio para el desarrollo del proyecto la falta de material existente para aplicar estos nuevos sistemas emergentes en la automática industrial, lo que demuestra que es un campo de estudio que no ha hecho más que empezar.

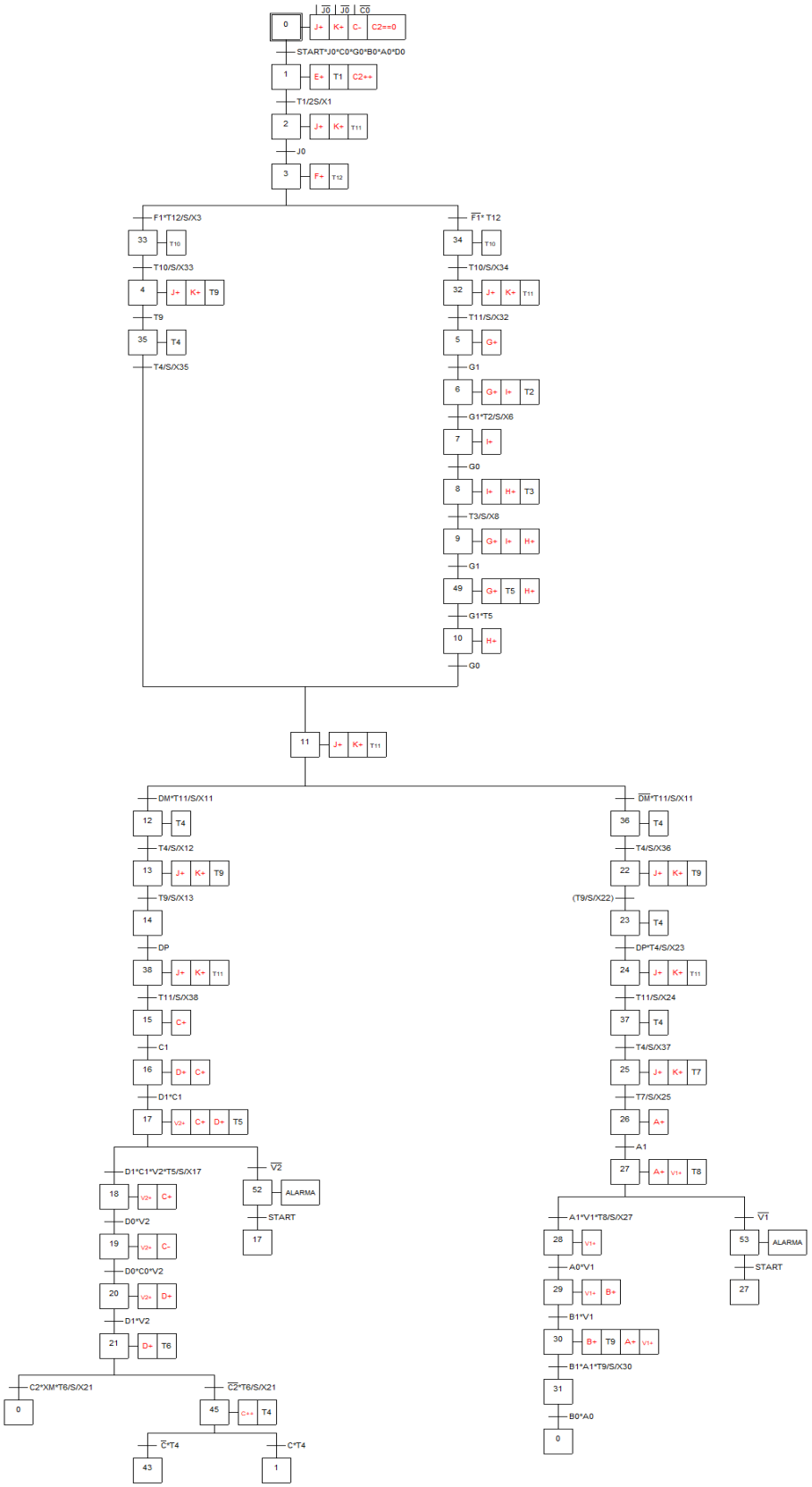
La virtualización de maquetas supone un punto de partida para un casi interminable ámbito de estudio y trabajo. Empezando por este proyecto, son muchísimas las mejoras que todavía pueden implementarse para conseguir que el sistema sea cada vez más robusto y eficiente, hasta llegar a una herramienta sólida.

Es importante desarrollar sistemas que sean altamente intuitivos para el usuario, consiguiendo así que su uso sea fácil y cómodo. Pensando en líneas futuras para mejorar este proyecto, sería interesante seguir trabajando para conseguir un sistema más robusto e intuitivo.

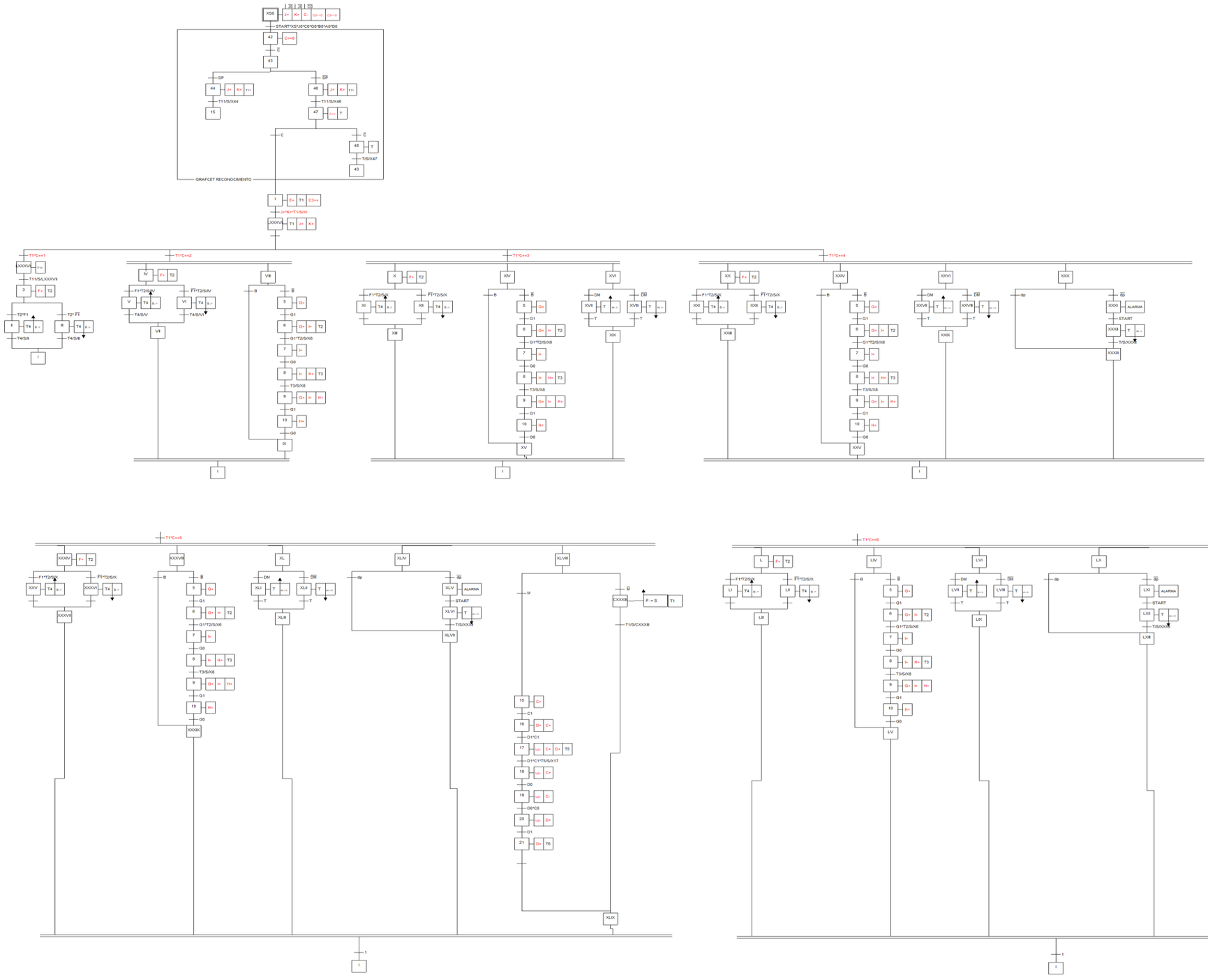
Para ello se propone dar al usuario más capacidad de decisión, dejándolo elegir, por ejemplo, las rutas de almacenamiento y lectura de archivos, o si desea trabajar mediante comunicación con el PLC o de forma manual. Además, realizar una captación de errores exhaustiva mejoraría la comprensión por parte del usuario ante cualquier posible problema que pudiera surgir. En cuanto a la robustez del código, sería interesante seguir trabajando en que las clases sean cada vez más generales, de forma que se pudiera utilizar una misma clase para más tipos de actuadores distintos.

En resumen, en un contexto en el que cada vez son más los dispositivos conectados a redes, la virtualización de maquetas y la creación de gemelos digitales aplicado a la industria y educación, puede suponer un pilar de gran importancia en la actual revolución industrial conocida como Industria 4.0.

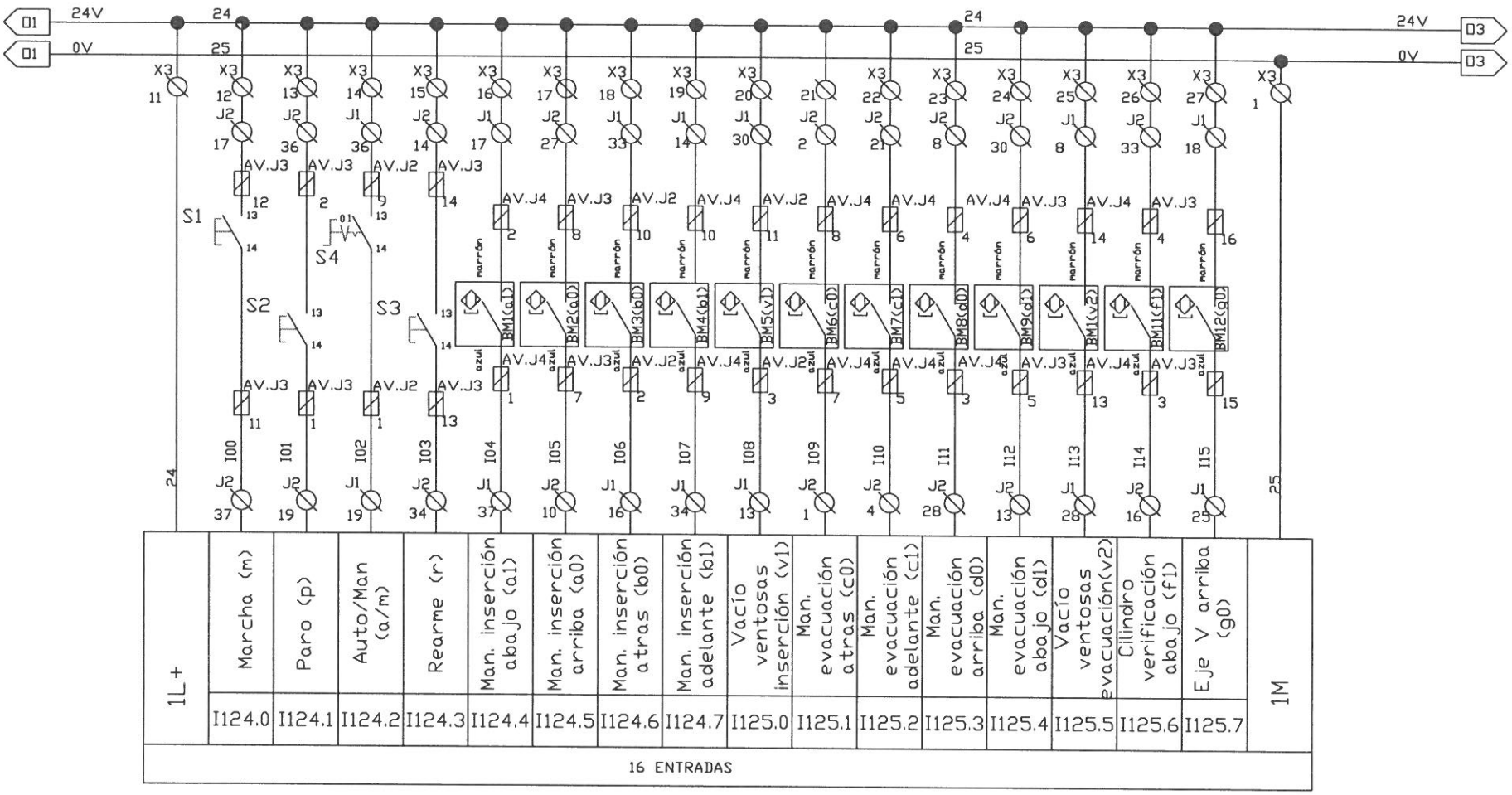
ANEXO A: GRAFCET CICLO BÁSICO



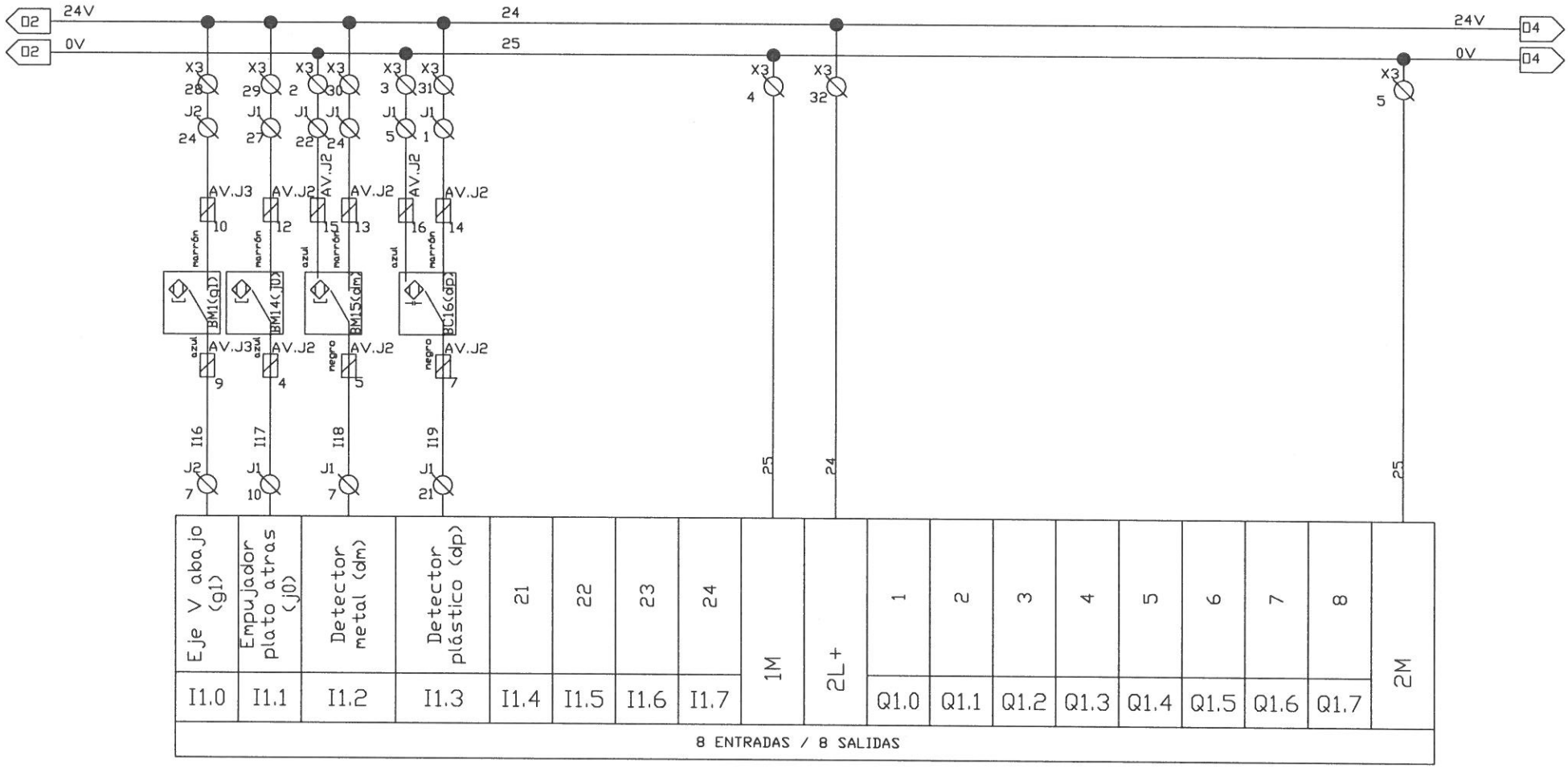
ANEXO B: GRAFCET CICLO MÚLTIPLE



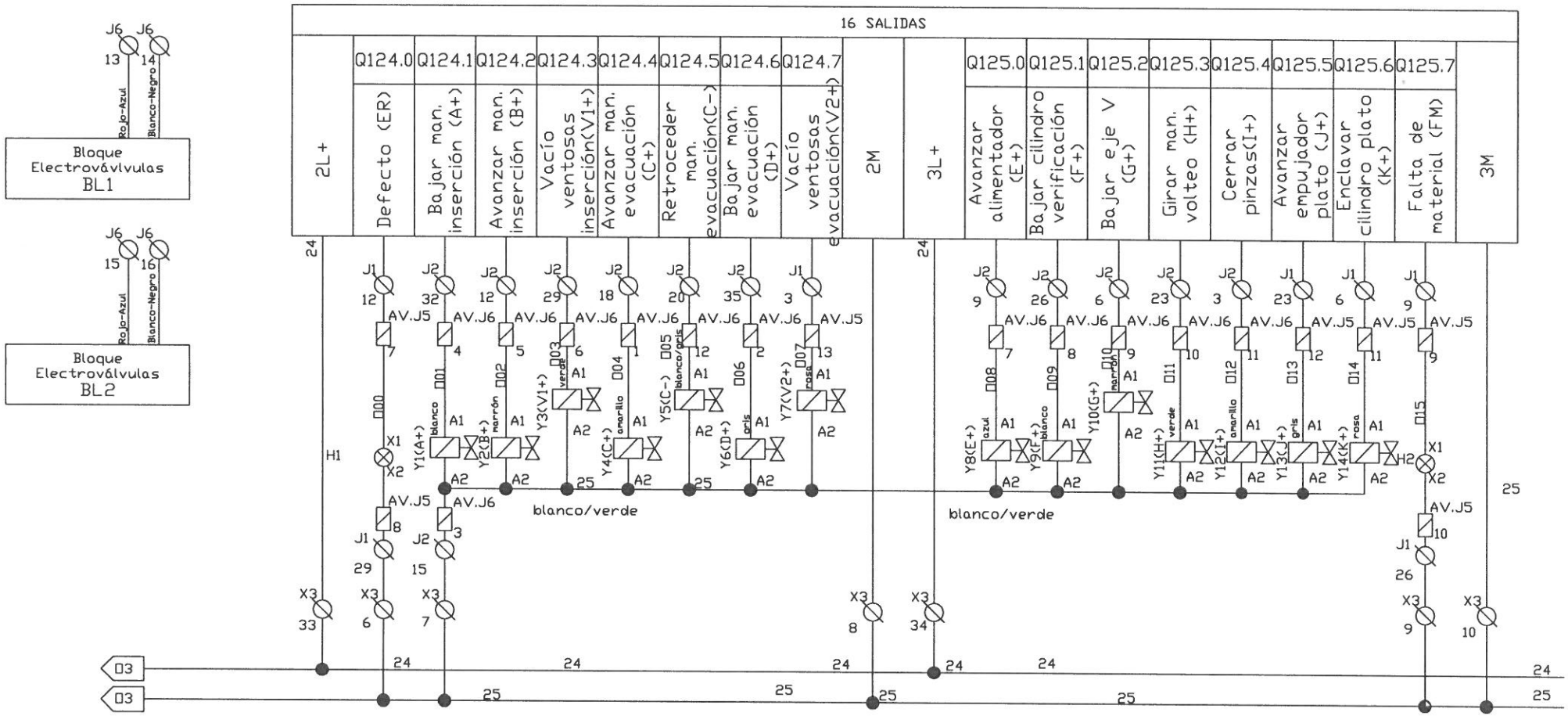
ANEXO C: PLANOS ELÉCTRICOS



Tolerancia General: Segun ISO-2768 m	Ref:	Fecha:	Nombre	Modificacion:	MATERIAL:		ESCALA:
					TRATAMIENTO	Fecha: 11-04-05	
						Dibujado: J. R. Barreal	
					CANTIDAD:	Comprobado: Fdz. de Antona	
SMC INTERNATIONAL TRAINING	FMS-200				DESIGNACION: CIRCUITO DE CONTROL ESTACIÓN EJE-ENTRADAS	Nº DE PLANO: S/4-2	



Tolerancia General: Segun ISO-2768 m	Ref:	Fecha:	Nombre	Modificacion:	MATERIAL:		ESCALA:
					TRATAMIENTO	Fecha: 11-04-05	
						Dibujado: J.R.Barreal	
					CANTIDAD:	Comprobado: Fdz. de Antona	
SMC INTERNATIONAL TRAINING	FMS-200				DESIGNACION: CIRCUITO DE CONTROL ESTACIÓN EJE-ENTRADAS/SALIDAS	Nº DE PLANO: S/4-3	



Tolerancia General: Segun ISO-2768 m	Ref:	Fecha:	Nombre	Modificacion:	MATERIAL:	ESCALA:
					TRATAMIENTO	Fecha: 11-04-05
					CANTIDAD:	Dibujado: J.R.Barreal
						Comprobado: Fdz. de Antona
SMC INTERNATIONAL TRAINING	FMS-200				DESIGNACION: CIRCUITO DE CONTROL ESTACIÓN EJE-SALIDAS	Nº DE PLANO: S/4-4

ANEXO D: VÍDEO DEMOSTRATIVO DEL GEMELO DIGITAL

A continuación, aparece un código QR que contiene el enlace de consulta del vídeo que se ha preparado con la intención de mostrar el funcionamiento del Gemelo Digital desarrollado en el proyecto.

Referencias

[1] Rodger W. Bybee, “What is STEM Education?”, Science Vol:329 (5995), pp:996. Accesible en: https://www.science.org/doi/full/10.1126/science.1194998. 2010.

[2] Edu4me, “LOS LABORATORIOS VIRTUALES: UNA ALTERNATIVA PARA BAJAR COSTOS Y AMPLIAR LA CAPACIDAD EN LAS UNIVERSIDADES” 2016. Accesible online: http://edu4.me/los-laboratorios-virtuales/ . Último acceso 17 de noviembre de 2021.

[3] Sothis, “FÁBRICAS DIGITALES: El primer paso hacia la Industria 4.0”, 2021. Accesible online: https://www.sothis.tech/fabricas-digitales-primer-paso-hacia-la-industria-4-0/ . Último acceso 17 de noviembre de 2021.

[4] Siemens news, “Fábrica Digital para la Industria de procesos”, 2021. Accesible online: https://new.siemens.com/es/es/empresa/temas-clave/fabrica-digital/industria-de-procesos.html?gclid=Cj0KCQiA4b2MBhD2ARIsAIrcB-Tqs1j_gQJiB379lRTSgO7NX8YAeMNVFh_2eoAx8HyPtqTshO4blrQaAnHTEALw_wcB . Último acceso 17 de noviembre de 2021.

[5] UNIR, “¿Qué es un IDE en programación?”, 2021. Accesible online: https://www.unir.net/ingenieria/revista/ide-programacion/ . Último acceso 17 de noviembre de 2021.

[6] NetBeans, 2021. Accesible online: https://netbeans.apache.org/

[7] Processing, 2021. Accesible online: https://processing.org/

[8] IBM, “¿Qué es el software de código abierto?”, 2021. Accesible online: https://www.ibm.com/ar-es/topics/open-source. Último acceso 17 de noviembre de 2021.

[9] Wikipedia, “Processing.js”, 2021. Accesible online: https://en.wikipedia.org/wiki/Processing.js. Último acceso 17 de noviembre de 2021.

[10] Wikipedia, “Plug and Play”, 2021. Accesible online: https://es.wikipedia.org/wiki/Plug_and_play . Último acceso 17 de noviembre de 2021.

[11] Siemens news, “Totally Integrated Automation Portal”, 2021. Accesible online: https://new.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html . Último acceso 17 de noviembre de 2021.

[12] Wikipedia, “Controlador Lógico Programable”, 2021. Accesible online: https://es.wikipedia.org/wiki/Controlador_l%C3%B3gico_programable. Último acceso 17 de noviembre de 2021.

[13] Siemens, “S7-PLCSIM. Manual de usuario”, 2001. Accesible online: https://automatasjwborjasunexpo.files.wordpress.com/2014/01/plcsim_s.pdf. Último acceso 17 de noviembre de 2021.

[14] Siemens, “Unidad de programación PG 740 PIII. Manual”, 1999. Accesible online: https://cache.industry.siemens.com/dl/files/117/1197117/att_43072/v1/740PIII_s.pdf.Último acceso 17 de noviembre de 2021.
[15] NetToPLCsim, 2021. Accesible online: http://nettoplcsim.sourceforge.net/
[16] Snap7, 2021. Accesible online: http://snap7.sourceforge.net/
[17] Autodesk Inventor, 2022. Accesible online: https://www.autodesk.es/products/inventor/overview?term=1-YEAR&tab=subscription
[18] Autodesk, 2021. Accesible online: https://www.autodesk.es/
[19] SMC, 2021. Accesible online: https://www.smc.eu/es-es
[20] Omron, 2021. Accesible online: https://industrial.omron.es/es/home
[21] Mitsubishi Electric, 2021. Accesible online: https://es.mitsubishielectric.com/es/
[22] TraceParts, 2021. Accesible online: https://www.traceparts.com/en
[23] Sublime Text, 2021. Accesible online: https://www.sublimetext.com/
[24] S. Herle, S. Man, G. Lazea, C. Marcu, and R. Robotin, "Exploiting parallel processing and optimal paths in a flexible manufacturing system," in *Automation Quality and Testing Robotics (AQTR), 2010 IEEE International Conference on*, 2010, pp. 1-6.
[25] Dark[byte], “Java-Paquete java.io-Introducción a los flujos de datos”, 2011. Accesible online: https://darkbyteblog.wordpress.com/2011/01/19/java-paquete-java-io-introduccion-a-los-flujos-de-datosio/. Último acceso 17 de noviembre de 2021.
[26] Github Processing Javadoc, 2021. Accesible online: https://processing.github.io/processing-javadocs/core/
[27] Processing, “PShape”, 2021. Accesible online: https://processing.org/reference/PShape.html. Último acceso 17 de noviembre de 2021.
[28] Oracle, “Class FileNotFoundException”, 2020. Accesible online: https://docs.oracle.com/javase/7/docs/api/java/io/FileNotFoundException.html. Último acceso 17 de noviembre de 2021.
[29] Oracle, “Class InputStream”, 2020. Accesible online: https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html. Último acceso 17 de noviembre de 2021.
[30] Processing, “LoadShape()”, 2021. Accesible online: https://processing.org/reference/loadShape_.html. Último acceso 17 de noviembre de 2021.
[31] Processing, “scale()”, 2021. Accesible online: https://processing.org/reference/scale_.html. Último acceso 17 de noviembre de 2021.

[32] Processing, “pushMatrix()”, 2021. Accesible online: https://processing.org/reference/pushMatrix_.html. Último acceso 17 de noviembre de 2021.

[33] Processing, “popMatrix()”, 2021. Accesible online: https://processing.org/reference/popMatrix_.html. Último acceso 17 de noviembre de 2021.

[34] Processing, “rotateY()”, 2021. Accesible online: https://processing.org/reference/rotateY_.html. Último acceso 17 de noviembre de 2021.

[35] Oracle, “The try Block”, 2021. Accesible online: https://docs.oracle.com/javase/tutorial/essential/exceptions/try.html. Último acceso 17 de noviembre de 2021.

[36] Automatas.org, “Introducción al GRAFCET”, 2006. Accesible online: https://www.automatas.org/redes/grafcet.htm. Último acceso 17 de noviembre de 2021.