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 7 DE CÉLULA DE FRABRICACION FLEXIBLE FMS200

Alumno: Cristian Manuel Galán Machuca

Tutor: Prof. D. Alejandro Sánchez García
Prof. Dª. Elisabet Estévez Estévez

Dpto: Ing. Electrónica y Automática

Junio, 2023

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Alejandro Sánchez Garcia y Dª Elisabet Estévez Estévez, tutores del Proyecto Fin de Carrera titulado: Gemelo Digital de la Estación 7 de fabricación flexible FMS200, que presenta Cristian Manuel Galán Machuca, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Junio de 2023

El alumno:

Cristian Manuel Galán Machuca

Firmado digitalmente por Cristian Manuel Galán Machuca
Fecha: 2023.07.01 17:46:20 +02'00'

CRISTIAN MANUEL GALÁN MACHUCA

Los tutores:

SANCHEZ GARCIA ALEJANDRO - 77321466C

Firmado digitalmente por SANCHEZ GARCIA ALEJANDRO - 77321466C
Fecha: 2023.06.30 18:46:48 +02'00'

ALEJANDRO SÁNCHEZ GARCÍA

ESTEVEZ ESTEVEZ ELISABET - 72578245C

Firmado digitalmente por ESTEVEZ ESTEVEZ ELISABET - 72578245C
Fecha: 2023.06.30 13:58:11 +02'00'

ELISABET ESTÉVEZ ESTÉVEZ

Resumen

Este proyecto constituye una aportación a la aplicación de desarrollo de gemelos digitales, que se está llevando a cabo en el Departamento de Ingeniería Electrónica y Automática.

En primer lugar, como aporte a la aplicación se ha desarrollado la iluminación de una escena digital, con esta contribución se pretende dar un paso más para la aplicación, consiguiendo una configuración sencilla e intuitiva de la escena para el usuario, en la que tendrá la posibilidad de implementar distintos tipos de iluminación.

En segundo lugar, se ha desarrollado un gemelo digital, concretamente el de la estación FMS-210 de Siemens. La estación FMS-210 es la séptima en la célula flexible de fabricación, presente en los laboratorios de la Universidad de Jaén y es la encargada del control de calidad de la pieza. Para la implementación del gemelo digital se ha llevado a cabo el diseño 3D de sus componentes, cada uno de ellos se ha caracterizado según sus funciones y utilidades, incluyendo de manera simplificada el control de calidad de las piezas.

Por último, se ha creado un proceso demostrativo del funcionamiento del gemelo digital usando un PLC simulado.

Abstract

This project is a contribution to the application of digital twin development being carried out in the Engineering Department of Electronic and Automatic.

In the first place, as a contribution to the application, the lighting of a digital scene has been developed. This contribution is intended to take the application a step further, achieving a simple and intuitive configuration of the scene for the user, in which the user will have the possibility of implementing different types of lighting.

Secondly, a digital twin has been developed, specifically the Siemens FMS-210 station. The FMS-210 station is the seventh in the flexible manufacturing cell present in the laboratories of the University of Jaen and is responsible for the quality control of the part. For the implementation of the digital twin, the 3D design of its components has been carried out, each one of them has been characterised according to its functions, including, in a simplified way, the quality control of the parts.

Finally, a demonstration process has been developed through the simulation of a PLC to demonstrate the functionality of the digital twin.

Índice

INDICE DE FIGURAS

INDICE DE TABLAS

1. Introducción

Este proyecto está destinado a crear una plataforma para desarrollar gemelos digitales, principalmente enfocado a la célula flexible de fabricación de la serie FMS-200 de Siemens, presentes en las instalaciones de la Universidad de Jaén. La plataforma se encuentra en desarrollo por varios alumnos del grado de ingeniería electrónica industrial de la Universidad de Jaén.

Un gemelo digital (Pozas, 2022) es una réplica virtual de un objeto, sistema o proceso físico que se utiliza para simular, analizar y optimizar el comportamiento del modelo real. Los gemelos digitales son una tecnología emergente que está transformando la forma en que se diseña, opera y mantiene los sistemas tanto industriales, como de fabricación, generación de energía y salud.

La importancia de los gemelos digitales en la industria es que permiten a los ingenieros, diseñadores y operadores probar y optimizar sistemas en un entorno virtual antes de implementarlos en el mundo real. Esto reduce significativamente el riesgo de fallos y permite a las empresas realizar mejoras de manera más rápida y económica. Además, los gemelos digitales pueden ser utilizados para monitorizar el rendimiento de los sistemas en tiempo real, lo que permite una mejor planificación y gestión del mantenimiento y la optimización de los procesos.

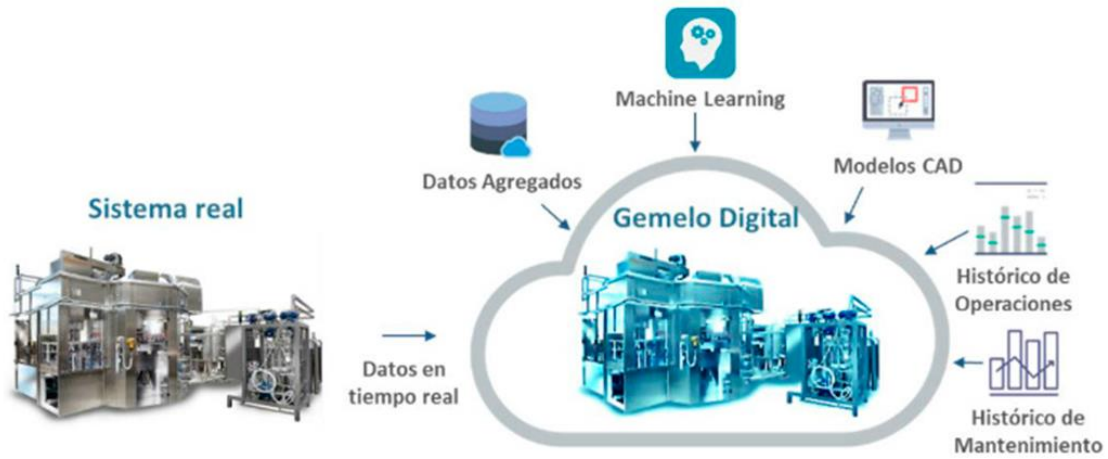


Figura 1. Características de un gemelo digital

En la Figura 1, (Mateo, 2020) se pueden observar las funcionalidades de un gemelo digital.

La capacidad del gemelo digital para simular diferentes escenarios de operación permite a los ingenieros y operadores de la planta probar y optimizar diferentes estrategias de control y operación antes de implementarlas en la planta real, lo que puede mejorar la eficiencia y reducir los costos operativos.

1.1. Motivación

Este proyecto está motivado por la necesidad de obtener una representación digital de la célula de fabricación flexible que se encuentra en los laboratorios de la Universidad de Jaén, destinadas al aprendizaje del alumnado.

En general la motivación de la creación de gemelos digitales es la de mejorar la eficiencia, calidad y productividad mediante la identificación y resolución de problemas del mundo real de una manera virtual.

Al crear un gemelo digital de un objeto físico, se puede simular su comportamiento en diferentes situaciones y condiciones, lo que permite a los usuarios evaluar diferentes escenarios y tomar decisiones informadas.

Los gemelos digitales también pueden utilizarse para desarrollar habilidades prácticas y experiencia en el mundo real en un entorno seguro y controlado. En un entorno educativo esto abre a los estudiantes la posibilidad de practicar en el gemelo digital antes de practicar con el modelo físico real, lo que reduce el riesgo de errores costosos y peligrosos.

En relación también con la enseñanza a distancia, si los estudiantes no tienen acceso a las instalaciones de la universidad, sea el caso de 2020 cuando la pandemia ocasionada por el virus COVID-19 dejó sin esta posibilidad a los alumnos, los gemelos digitales abren la posibilidad de un aprendizaje de calidad sin necesidad de presencialidad.

Un gemelo digital en la industria, representado en la Figura 2 (SIEMENS, s.f.), tiene ventajas como:

- ✓ Mejora de toma de decisiones, al poder simular los escenarios y ver cómo se comportaría el producto o sistema en ellos.

- ✓ Reducción de costos y tiempo, ya que los gemelos digitales pueden detectar problemas y fallos antes de que se produzcan en el mundo real, evitando de esta manera costos y ahorrando tiempo.
- ✓ Mejorar la eficiencia, identificando oportunidades de mejora y optimización en un entorno virtual.
- ✓ Innovación, el gemelo digital permite cambiar piezas o su funcionamiento para mejorar sus funcionalidades pudiendo de este modo innovar de forma segura.

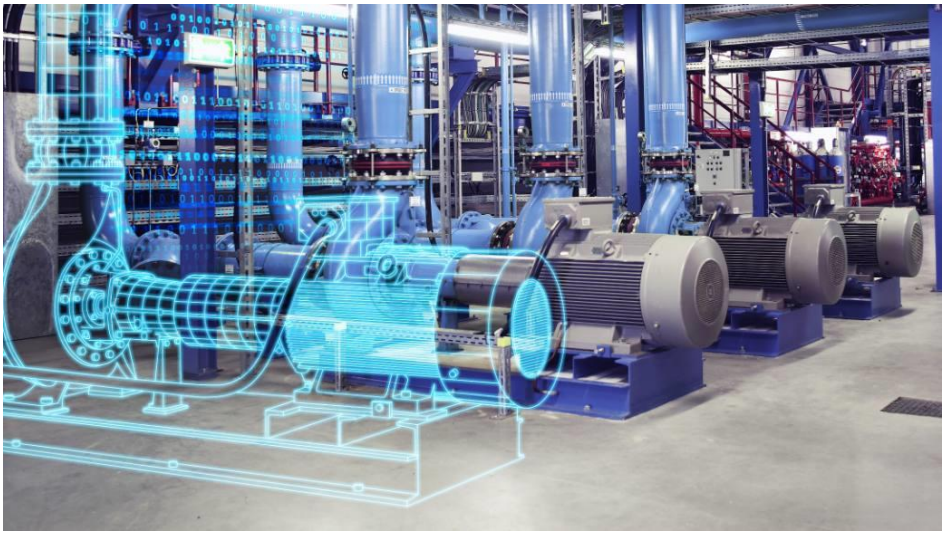


Figura 2. Representación de un gemelo digital en la industria

Los gemelos digitales (ITI, 2020) tienen una estrecha relación con la industria 4.0. La industria 4.0 se refiere a la integración de tecnologías avanzadas en la fabricación y la industria para crear una "fábrica inteligente" que es más eficiente, flexible y conectada. Los gemelos digitales son una herramienta clave en la industria 4.0 porque permiten la simulación, el análisis y la optimización de procesos y sistemas complejos en tiempo real.

Además, los gemelos digitales pueden ser utilizados para conectar y coordinar diferentes sistemas y procesos en una fábrica inteligente, lo que permite una mayor automatización y eficiencia en la producción.

1.2. Objetivos

El principal objetivo de este trabajo es dotar a la plataforma de gemelos digitales de una mayor capacidad de configuración, en lo que a la iluminación de la escena se refiere. Se pretende hacerlo para que la configuración de la iluminación sea rápida y sencilla de configurar y modificar para el usuario, consiguiendo de esta manera obtener representaciones gráficas que se vean más reales o que se adapten más a la visualización deseada.

Una iluminación adecuada es fundamental para crear una apariencia realista en los gemelos digitales. La forma en que la luz interactúa con los objetos virtuales puede simular sombras, reflejos y brillos, lo que contribuye a hacer que los gemelos digitales sean más creíbles y se integren de manera más efectiva en la escena virtual.

Otro objetivo a tener en cuenta es la realización de un gemelo digital de la estación FMS-210, presente en las instalaciones de la Universidad de Jaén. Con la creación de este gemelo digital se pretende conseguir replicar el comportamiento de la estación tanto a nivel visual, como a nivel de programación de un PLC., teniendo en cuenta sus movimientos y el control de calidad por visión que realiza de la pieza. Cumpliendo este objetivo se consigue que el alumnado pueda recrear a distancia las condiciones con las que trabajaría en las instalaciones de la universidad. Otra ventaja de la realización de este gemelo digital es que el alumnado evita la falta de tiempo de laboratorio, ya que en ocasiones los alumnos deben establecer turnos para trabajar con las estaciones. De este modo solo será necesario asistir de forma presencial al laboratorio para realizar comprobaciones.

Por último, se pretende realizar una simulación de demostración del proceso que realiza la estación FMS-210, para el análisis de calidad de las piezas fabricadas.

1.4 Estructura

Para que se pueda tener un breve conocimiento sobre los epígrafes del proyecto se desarrolla este apartado de estructura del mismo.

El capítulo *2 Software utilizado*, detalla el software que se ha utilizado para el desarrollo de la aplicación y el gemelo digital.

En el capítulo *3 Introducción a la aplicación*, desarrolla una explicación de las clases de la aplicación para tener una vista general de ella.

El capítulo *4 Sistema de iluminación de una escena digital*, detalla cómo se ha implementado la iluminación de la escena, explicando todas las clases implementadas, con ejemplos en Processing y en la aplicación.

El capítulo *5 Modelo Digital FMS-210*, muestra todos los modelos 3D desarrollados para crear el gemelo digital de la estación.

El capítulo *6 Modelo de información de FMS-210* detalla la forma en la que se desarrolla el gemelo digital de la estación a partir del fichero XML.

El capítulo *7 Simulación de control de calidad* recoge la parte de evaluación de las piezas del gemelo digital, explica cómo se ha llevado a cabo y qué clases participan.

El capítulo *8 Manual de usuario*, indica cómo poner en marcha el gemelo digital de la estación FMS-210, explicando cómo se realiza la conexión entre el gemelo digital y TIA Portal a través de NetToPLCSim.

El capítulo *9 Conclusiones*, pretende finiquitar el proyecto exponiendo los objetivos logrados y las líneas futuras.

Por último, figuran los ANEXOS:

ANEXO I: Esquema eléctricos de la estación FMS-210.

ANEXO II: XML completo del gemelo digital de la estación FMS-210.

2. Software utilizado

Este capítulo presenta de una manera general aquellas herramientas software necesarias para elaborar una plataforma que permita el diseño y desarrollo de gemelos digitales con finalidad de monitorizar estaciones de la célula de fabricación flexible FSM 200.

Nótese que todas ellas se tratan de herramientas de código abierto.

2.1. NetBeans

NetBeans (icono Figura 3) (NETBEANS, 2023) es un entorno de desarrollo integrado (IDE) de código abierto utilizado para desarrollar aplicaciones en Java y otros lenguajes de programación, como HTML5, PHP, C++, entre otros. Fue desarrollado originalmente por Sun Microsystems, y actualmente es mantenido por Apache Software Foundation.

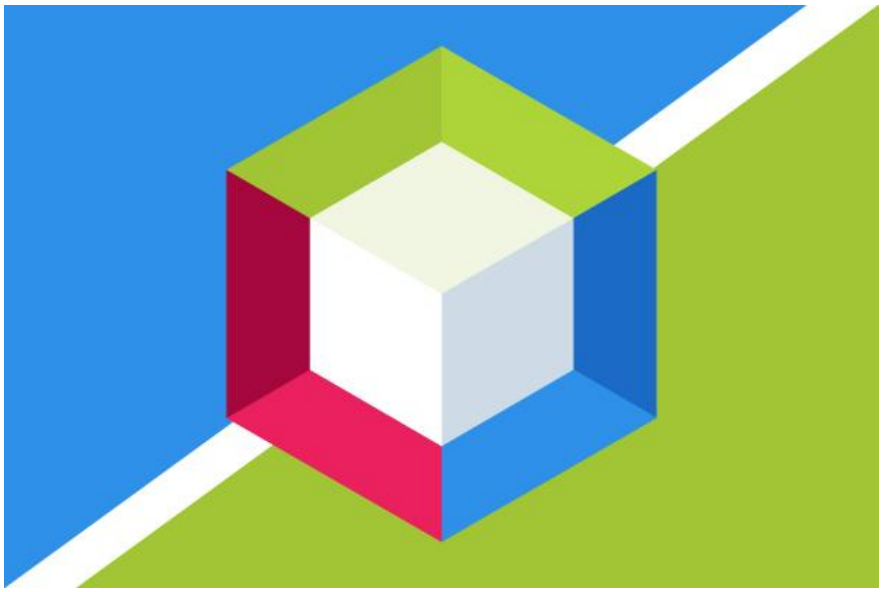


Figura 3. Apache NetBeans

Para el núcleo de la plataforma de desarrollo de gemelos digitales se ha hecho uso de NetBeans, a la cual se le ha añadido librerías para poder operar con la estructura de Processing. Para acceder a las entradas y salidas se hace uso de la librería Moka7.

Las principales ventajas de NetBeans son:

- Soporte para múltiples lenguajes de programación: NetBeans es compatible con varios lenguajes de programación, como Java, C/C++, PHP, HTML, JavaScript, entre otros.
- Edición de código avanzada: NetBeans proporciona una gran cantidad de características para facilitar la edición de código, como el resaltado de sintaxis, la auto-completación, la refactorización de código y la navegación de código.
- Depuración integrada: NetBeans ofrece un potente depurador integrado que permite a los desarrolladores detectar y corregir errores en el código de manera eficiente.
- Integración con sistemas de control de versiones: NetBeans ofrece soporte para sistemas de control de versiones como Git, SVN y Mercurial, lo que facilita el trabajo en equipo en proyectos de desarrollo de software.
- Diseñador de interfaces gráficas de usuario (GUI): NetBeans incluye un diseñador de GUI para crear interfaces gráficas de usuario fácilmente y sin necesidad de escribir código.
- Herramientas de análisis de código: NetBeans proporciona herramientas para analizar el código y detectar problemas como errores de sintaxis, errores de estilo de código.
- Integración con servidores de aplicaciones: NetBeans se integra con varios servidores de aplicaciones como Apache Tomcat, GlassFish y WildFly, lo que facilita el desarrollo de aplicaciones web y móviles.
- Compatibilidad con varias plataformas: NetBeans es compatible con Windows, Linux y macOS, lo que permite a los desarrolladores trabajar en diferentes sistemas operativos.

2.2. Processing

Processing (icono Figura 4) (Processing, s.f.) es una plataforma de programación creativa y de código abierto que se utiliza principalmente para crear visualizaciones interactivas, gráficos generativos y animaciones.

Processing es una herramienta muy útil para la educación, ya que permite a los estudiantes experimentar con la programación creativa y aprender a través de la exploración y la experimentación. También se utiliza en investigación, diseño de interfaces de usuario, robótica y muchas otras áreas que requieren la creación de visualizaciones interactivas y gráficos generativos.

Figura 4. Processing

Las ventajas del uso de Processing son:

- Fácil aprendizaje: Processing fue diseñado para ser un lenguaje accesible para cualquier persona, independientemente de su nivel de experiencia en programación.
- Orientación a la visualización: Processing se enfoca en la creación de proyectos visuales e interactivos, lo que lo hace ideal para artistas, diseñadores y personas interesadas en la visualización de datos.
- Gran comunidad: Processing cuenta con una gran comunidad de desarrolladores y usuarios que comparten conocimientos, herramientas y recursos para ayudar a los demás.
- Multiplataforma: Processing funciona en múltiples plataformas, incluyendo Windows, Mac OS X y Linux.
- Integración con otras tecnologías: Processing se integra fácilmente con otras tecnologías y lenguajes de programación, como Java, JavaScript y Python.
- Amplia variedad de bibliotecas: Processing tiene una amplia variedad de bibliotecas disponibles que permiten a los desarrolladores agregar fácilmente funcionalidades a sus proyectos, como la interacción con cámaras, la creación de gráficos 3D, la detección de gestos y mucho más

2.3. TIA Portal.V14

TIA Portal (icono Figura 5) (SIEMENS, 2023) es un software de ingeniería utilizado para programar y configurar dispositivos y sistemas de automatización industrial. TIA Portal es una abreviatura de "Totally Integrated Automation Portal" y es desarrollado por Siemens AG, una empresa alemana de tecnología industrial.

Figura 5. TIA Portal v.14

El software TIA Portal ofrece una interfaz de usuario integrada para configurar y programar dispositivos y sistemas de automatización industrial, incluyendo controladores lógicos programables (PLCs), dispositivos de entrada/salida, dispositivos de seguridad y sistemas de visualización. El software también ofrece herramientas de diagnóstico y simulación para ayudar en la resolución de problemas y en la mejora de la eficiencia de los sistemas.

Entre las ventajas de TIA Portal se encuentran su integración con otros productos de Siemens, su facilidad de uso y su eficiencia en el tiempo de desarrollo. Además, TIA Portal permite la programación y configuración de dispositivos de diferentes marcas y proveedores, lo que lo convierte en una herramienta versátil y flexible para la automatización industrial.

2.4. PLC SIM

PLC SIM (icono Figura 6) es un software de simulación de controladores lógicos programables que se utiliza para simular el comportamiento de un PLC sin la necesidad de hardware físico.

Figura 6. PLC SIM

Permite a los usuarios crear y ejecutar programas de PLC en un entorno virtual, lo que facilita el aprendizaje y la resolución de problemas. También se utiliza para probar y validar programas de PLC antes de su implementación en una aplicación real. PLC SIM es una herramienta útil para los programadores y diseñadores de sistemas de control, ya que les permite probar y ajustar el comportamiento de un sistema de control sin tener que realizar pruebas en el mundo real.

2.5. NetToPLCsim

NetToPLCsim (NetToPLCsim, 2022) es una solución de programación de controladores lógicos programables de Siemens que permite a los usuarios programar y controlar los sistemas de automatización industrial. Es una herramienta de programación basada en software que permite a los usuarios diseñar, simular y depurar programas de PLC para aplicaciones de automatización industrial. Sigue un esquema como el de la Figura 7 (infoPLC, 2021).

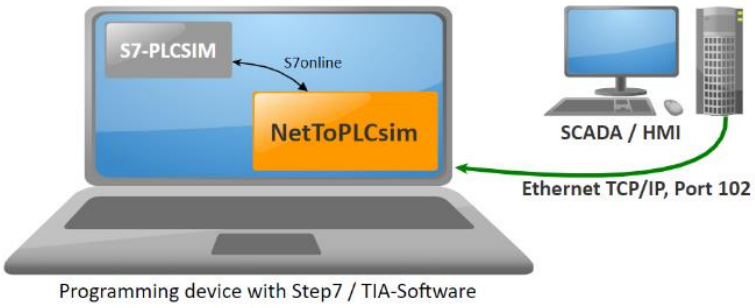


Figura 7. NetToPLCsim

NetToPLCsim proporciona un entorno de programación amigable para el usuario, que utiliza un lenguaje de programación de bloques de función (FBD) y gráficos de escalera (LAD) para programar el comportamiento del PLC. Además, también ofrece herramientas de simulación para probar el comportamiento del programa en un entorno virtual antes de su implementación en una aplicación real.

NetToPLCsim es una solución completa para la programación de PLC de Siemens, y se utiliza ampliamente en la industria para controlar procesos y sistemas automatizados en diferentes sectores, como la fabricación, el transporte, la energía y la construcción.

2.6 Autodesk Inventor Professional 2022

Autodesk Inventor 2022 (icono Figura 8) (AUTODESK, 2022) es un software de diseño asistido por computadora (CAD) desarrollado por Autodesk. Está diseñado para ayudar a los ingenieros y diseñadores a crear modelos 3D precisos y detallados de productos y piezas mecánicas.

Figura 8. Autodesk Inventor 2022

El software cuenta con herramientas avanzadas de modelado y simulación, que permiten a los usuarios crear y probar sus diseños en un entorno virtual antes de fabricar el producto final. También incluye herramientas de documentación y colaboración que facilitan la comunicación y el intercambio de datos entre equipos de diseño y producción.

Entre las principales características de Autodesk Inventor 2022 se incluyen:

- Modelado paramétrico y basado en funciones
- Simulación de movimiento y fuerzas
- Herramientas de visualización y renderizado avanzadas
- Funciones de ensamblaje y diseño de tuberías

- Capacidades de diseño de chapa metálica y soldadura
- Herramientas de creación de planos y documentación
- Integración con otras herramientas de Autodesk, como AutoCAD y Fusion 360.

3 Introducción a la Plataforma GRAV-DT

Para que sea más sencilla la comprensión del proyecto, en el capítulo tres se hace una breve explicación general de lo que ya se encuentra desarrollado en la aplicación, para así entender mejor el aporte realizado en este trabajo de fin de grado.

3.1 Clases creadas

La plataforma consta con anterioridad de dos package añadidos Moka7 y *.com.grav.gemelodigital.*

3.1.2 Moka 7

La biblioteca Moka7 (Rojko, 2017) es una biblioteca Java que se utiliza para comunicarse con dispositivos que usan el protocolo de comunicación S7 , como los controladores lógicos programables de Siemens. Esta biblioteca permite a los programadores Java leer y escribir datos desde y hacia dispositivos S7, lo que es muy útil en la automatización industrial.

En el contexto de Processing, la biblioteca Moka7 se puede utilizar para interactuar con dispositivos S7 y controlar procesos industriales en tiempo real. Esto permite a los desarrolladores de Processing crear proyectos de automatización industrial y control de procesos que se comunican con dispositivos S7.

3.1.2 .com.grav.gemelodigital

El package *.com.grav.gemelodigital*, es el package donde se desarrolla principalmente la plataforma de gemelos digitales. La clase principal dentro de este package, la cual sería la que se ejecuta para da lugar al gemelo digital es la clase GemeloDigital.

Para una mejor comprensión de la aplicación se presenta un diagrama de clases en la **¡Error! No se encuentra el origen de la referencia.**.

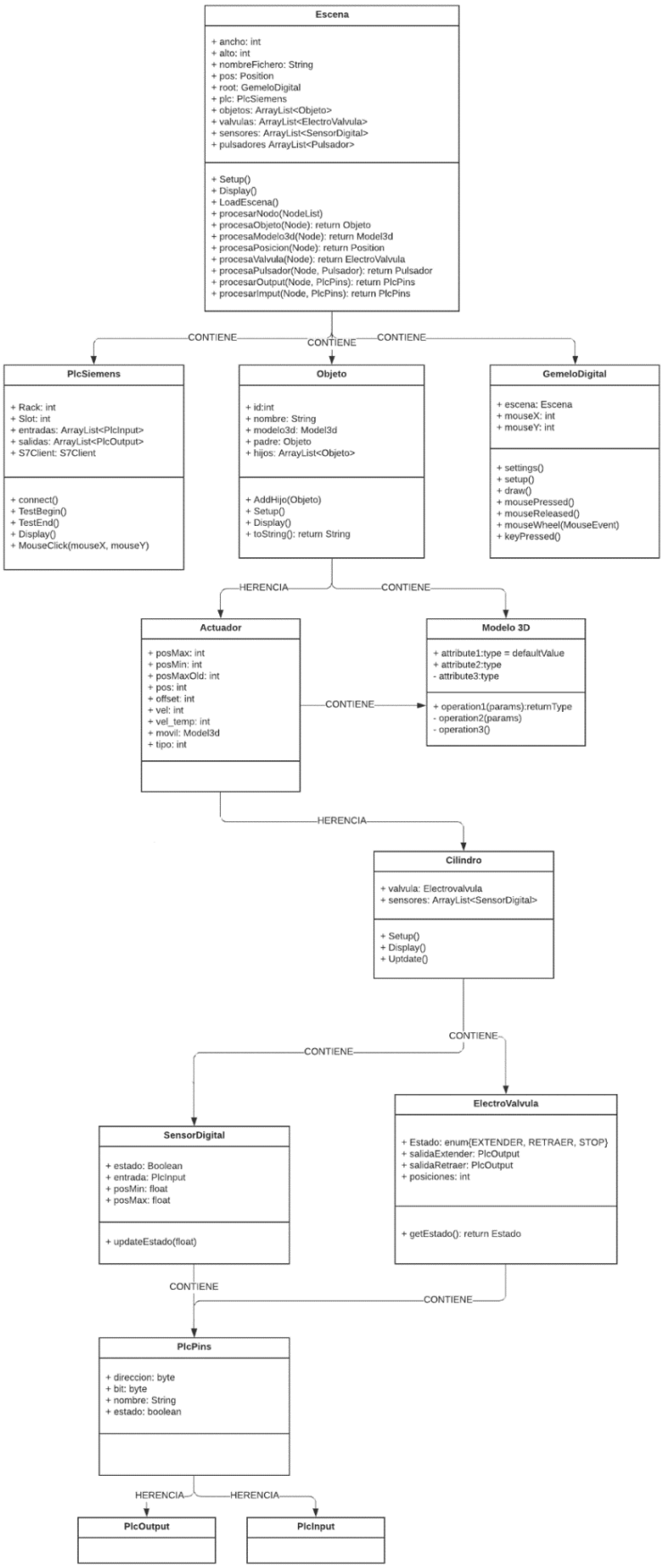


Figura 9. Diagrama de clases del GRAV-DT

La clase GemeloDigital, es la clase que se ejecuta y da lugar a la escena. Esta clase se compone principalmente de tres métodos que hace que funcione el gemelo digital:

- Void setting(): tiene como objetivo iniciar el entorno de Processing y se ejecuta solo una vez al comienzo del programa. Después de su ejecución, el programa tiene información sobre los componentes que componen el diseño técnico
- Void setup(): se encarga de iniciar todos los módulos identificados, se ejecuta una sola vez.
- Void draw(): es la encargada de representar y dar dinamismo al gemelo digital, está ejecutándose constantemente, lo que permite actualizar las entradas y salidas del gemelo digital y recargar gráficamente la escena.

La clase Escena es la encargada de procesar todos los elementos del gemelo digital de forma individual de manera que genera la escena, obtiene todos los parámetros de un fichero XML ligado a ella.

La clase Objeto se encarga de definir un objeto, el cual tiene un nombre, una representación 3D y un identificador, es la clase padre Actuator y crea un ArrayList en el cual se almacenan todas las clases hijas.

La clase model3D se encarga de representar gráficamente todos los objetos, esta clase se incluye como atributo en la clase objeto y en sus clases hijas. Como parámetros propios la clase model3D tiene el nombre, la posición del objeto, el nombre del archivo .obj a representar y la escala en la que debe ser representada.

La clase Position determina la posición en la que es representado el objeto. Como parámetros tiene las posiciones en los ejes “x”, “y” y “z” y la orientación del objeto “rx”, “ry” y “rz”.

La clase Actuator es hija de la clase Objeto, define un objeto que se comporta como un actuador. A los parámetros de la clase padre Objeto se le añade, un modelo 3D móvil, velocidad de movimiento, tipo de movimiento, posición máxima y posición mínima.

La clase cilindro es hija de la clase Actuator, esta clase define un actuador de tipo cilindro el cual hereda los parámetros de Objeto y Actuator. Añade además el parámetro Electrovalvula para gestionar el cilindro.

La clase Electrovalvula, gestiona el comportamiento de un cilindro, ya sea para extenderlo o retraerlo, tiene como parámetros un estado definido por un enum que dependiendo de si es 0 se extiende, 1 se retrae y 2 detiene su movimiento. Por último, el atributo numPos, define las posiciones de la válvula.

La clase PLCSiemenens es la encargada de simular el comportamiento del PLC, esta clase gestiona las entradas y las salidas del proceso.

La clase PlcPins es la que simula una dirección de una entrada o salida del PLC, esta clase contiene la dirección de la entrada o salida y su estado. También lleva asociado un nombre para diferenciarla. De esta clase se extienden dos clases hijas las cuales son PlcInput y PlcOutput, estas clases han sido creadas para diferenciar el procesamiento del pin del PLC según sea una entrada o una salida.

La clase Pulsador, se encarga de generar un pulsador para actuar sobre las entradas de tal tipo del gemelo digital, este pulsador lleva asociado un estado y una posición para ser representado en la escena. Puede ser de 2 tipos, pulsador o interruptor.

La clase SensorDigital es la encargada de representar los sensores del gemelo digital, lleva asociado un estado, una representación gráfica, un bit del PLC y una posición máxima y mínima para determinar el estado del sensor.

3.2 Clases importadas

Para entender cómo funciona la plataforma y su conexión con Processing, se presentan las siguientes librerías añadidas en algunas de las clases anteriormente mencionadas, que hacen posible que la aplicación funcione.

La clase processing.core.Papplet (Class PApplet, 2022) en Java es la clase principal del framework de Processing. Esta clase proporciona un entorno de programación para la creación de gráficos interactivos y multimedia, y contiene

métodos para dibujar formas, cargar y manipular imágenes y sonidos, y manejar eventos de usuario como clics de mouse y pulsaciones de teclas.

Para utilizar la clase processing.core.PApplet, se debe crear una subclase que extienda esta clase y proporcionar una implementación de los métodos settings(), setup(), y draw(). Estos métodos se llamarán automáticamente cuando se ejecute la aplicación y se utilizan para inicializar el estado de la aplicación, configurar la apariencia de la pantalla y dibujar gráficos en la pantalla, respectivamente.

Además de los métodos settings(), setup() y draw(), la clase processing.core.PApplet proporciona una amplia variedad de métodos para interactuar con la pantalla y el usuario. Algunos ejemplos de estos métodos son:

- background(): establece el color de fondo de la pantalla.
- fill(): establece el color de relleno para las formas dibujadas.
- stroke(): establece el color de borde para las formas dibujadas.
- ellipse(): dibuja una elipse en la pantalla.
- rect(): dibuja un rectángulo en la pantalla.
- image(): carga y dibuja una imagen en la pantalla.
- loadImage(): carga una imagen desde un archivo.

La librería processing.core. es un conjunto de clases y métodos que se utilizan en el framework Processing para crear aplicaciones gráficas interactivas. La biblioteca *processing.core* proporciona funcionalidades para la creación de gráficos en 2D y 3D, el manejo de imágenes y video, la entrada de datos de dispositivos como el mouse y el teclado, y la creación de interfaces gráficas de usuario (GUI).

La clase *processing.event.MouseEvent* en Java es una clase que representa un evento de ratón en el contexto del framework de Processing. Esta clase se utiliza para capturar información sobre eventos de ratón, como clics, movimientos y arrastres, y proporciona métodos para acceder a la posición del ratón y otros detalles relacionados con el evento.

Para capturar eventos de ratón en Processing, se utiliza el método mouseEvent() de la clase processing.core.PApplet. Este método se llama

automáticamente cada vez que se produce un evento de ratón, y como argumento recibe una instancia de processing.event.MouseEvent que contiene información sobre el evento.

java.awt.event.KeyEvent.VK_CONTROL es una clase que representa un evento generado por la interacción del usuario con el teclado. La constante VK_CONTROL es una constante estática que pertenece a la clase KeyEvent y representa la tecla "Control" del teclado.

VK_CONTROL se utiliza para representar la tecla de control en Java. Esta constante se utiliza comúnmente en combinación con otras teclas para realizar acciones específicas como cambiar la orientación de la escena.

La clase java.util.logging.Level en Java es una clase que define un nivel de registro (logging level) utilizado para clasificar mensajes de registro (logging messages). Los niveles de registro son una forma de filtrar mensajes de registro basados en su importancia o severidad.

La clase javax.xml.parsers.ParserConfigurationException en Java es una excepción que se lanza cuando ocurre un error al crear un nuevo analizador de documentos XML (XML document parser) utilizando la API de procesamiento de XML de Java.

La clase ParserConfigurationException es una subclase de la clase Exception y se utiliza para señalar errores durante la configuración de un analizador de documentos XML. Algunos de los errores comunes que pueden provocar esta excepción son la especificación de una característica no compatible o la falta de soporte para una función determinada por parte del analizador.

La clase java.io.File en Java es una clase que representa un archivo o un directorio en el sistema de archivos de la máquina en la que se está ejecutando una aplicación Java. Esta clase se utiliza para realizar operaciones relacionadas con archivos, como crear, leer, escribir o eliminar archivos o directorios.

La clase java.util.ArrayList, esta clase funciona como una estructura de datos en la que se pueden almacenar y manipular colecciones de objetos en Java. Se

utiliza cuando se necesita una estructura de datos que permita agregar, eliminar y acceder a elementos de manera dinámica y eficiente.

A diferencia de los vectores en Java, donde el tamaño del vector se establece en el momento de su creación y no se puede modificar, los objetos de la clase ArrayList pueden crecer o reducir su tamaño según se agreguen o eliminen elementos, esto nos permite añadir las iluminaciones deseadas sin tener que establecer un número limitado de ellas.

3.3 Tratamiento del fichero XML

Un fichero XML (XML, 2022) (eXtensible Markup Language) es un tipo de archivo que utiliza etiquetas para marcar y estructurar datos en un formato legible por máquina y por humanos. Es similar a un archivo HTML, pero mientras que HTML se utiliza principalmente para mostrar información en un navegador web, XML se utiliza para describir datos estructurados.

Para generar el gemelo digital se configura un fichero XML que está ligado a la clase Escena, esta clase Escena es la encargada de interpretarlo. Por último, la clase GemeloDigital es la encargada de representar los datos obtenidos del XML.

La estructura en el fichero XML cambia dependiendo de la clase o el parámetro que vaya a ser interpretado. Para ello se han creado diferentes métodos en la clase Escena capaces de interpretar todas las clases creadas.

Un ejemplo sencillo sería para definir el ancho y alto de una escena se representa en la Figura 10.

```
<ancho>1200</ancho>
<alto>800</alto>
```

Figura 10. Configuración de ancho y alto de una escena

Del mismo modo si se quiere añadir un pulsador se haría de la forma mostrada en la Figura 11.

```
<pulsador id="2">
    <nombre>pulsador</nombre>
    <modelo3d>
        <fileName>config/obj/zb5_aa6.obj</fileName>
        <pos>
            <x>200.0</x>
            <y>0.0</y>
            <z>0.0</z>
            <rx>0.0</rx>
            <ry>0.0</ry>
            <rz>0.0</rz>
        </pos>
        <escala>1</escala>
    </modelo3d>
    <input>
        <bit>2</bit>
        <byte>0</byte>
        <nombre>PB</nombre>
    </input>
</pulsador>
```

Figura 11. Configuración de un pulsador

Se puede observar cómo se añade un pulsador, para empezar se le asigna un identificador, después se añade la ruta a su modelo 3D, indicando también la posición en la escena que ocuparía, por último, se le asigna su dirección de entrada correspondiente.

De este modo mediante el XML se puede generar el gemelo digital añadiendo objetos, cilindros, sensores, entradas y salidas, pulsadores e iluminaciones.

4 Sistema de iluminación de una escena digital

Para seguir un progreso normal en el desarrollo de la aplicación de gemelos digitales, en este trabajo de fin de grado se ha hecho el aporte de la caracterización de la iluminación de la escena. Para ello se implementarán distintos tipos de iluminación que serán editables para el usuario según convenga a la estación a la que se le esté realizando el gemelo digital, o la forma en la que quiera que se vea la escena.

Con esta contribución se pretende que la simulación parezca más realista y creíble, al agregar distintos tipos de iluminación que pueden crear efectos que imitan las condiciones del mundo real. La iluminación adecuada puede mejorar la claridad de la simulación, por lo que puede ser utilizada para resaltar objetos críticos o partes del modelo que deban ser examinadas.

La iluminación puede ser utilizada para ayudar a los usuarios a orientarse dentro de la simulación. Por ejemplo, se pueden agregar focos, para guiar la iluminación donde el usuario desee.

Inicialmente el sistema de iluminación de la plataforma estaba implementado mediante la función lights(), esta función habilita la iluminación en una escena 3D. Al llamarla, se activa el modelo de iluminación por defecto de Processing. Esta función estaba definida en la clase GemeloDigital, en la función draw(), esta función la vemos representada en la Figura 12.

```
public void draw() {
    //Limpiar pantalla.
    this.background(100);
    this.ortho();
    translate(0,height);
    rotateY(PI);
    rotateZ(PI);
    escena.plc.Display(this, new Position(50.0f,50.0f,0.0f));
    lights();
```

Figura 12. Función lights() en el draw de la clase GemeloDigital

Si se quiere diseñar un tipo de iluminación distinta se debe acceder a otro tipo de funciones y no usar la función lights().

Para caracterizar el sistema de iluminación se profundizará en los tipos de iluminación que se pueden implementar a través de Processing los cuales son: iluminación ambiental, iluminación direccional, luz especular y caída de luz.

Otro punto a tener en cuenta es la posibilidad de poder destacar elementos, para ello también se ha implementado la posibilidad de añadir focos.

Para tener una idea de cómo se va a desarrollar el sistema de iluminación se proporciona el siguiente diagrama de clases en la Figura 13.

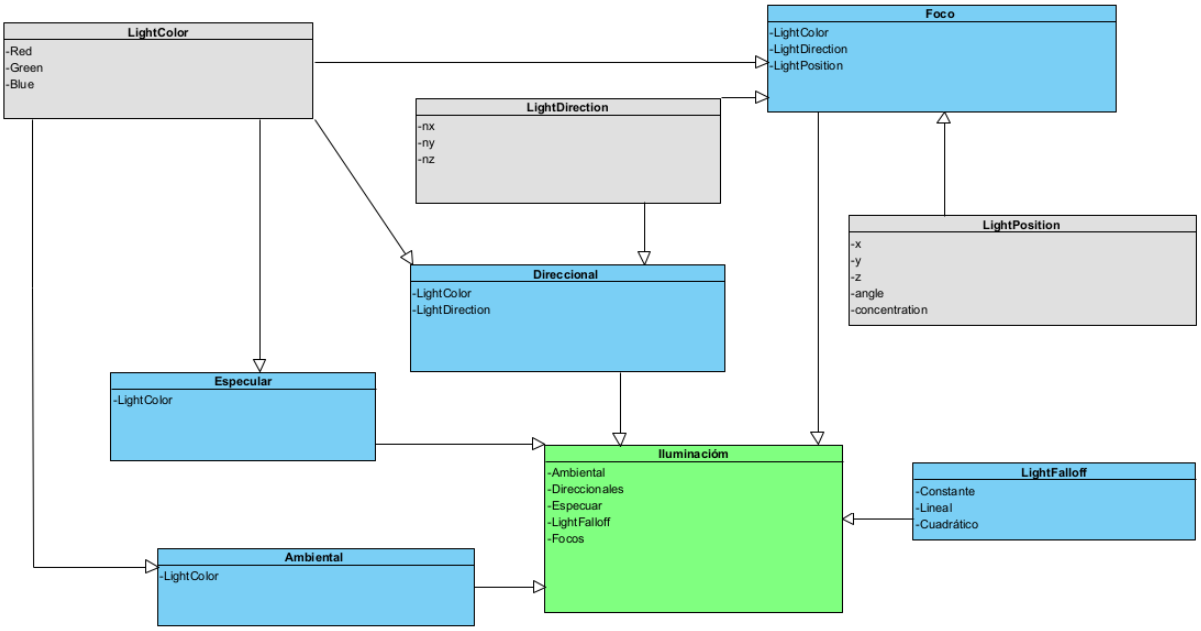


Figura 13. Diagrama de clases de iluminación de la escena

Las clases en color gris serían las auxiliares que ayudarían a crear los distintos tipos de iluminación. Los tipos de iluminación se representan con el color azul, por último, la clase iluminación representa la iluminación general de la escena representada de color verde.

4.1 Clases auxiliares para la iluminación

Han sido creadas 3 clases que, aunque por sí mismas no son capaces de generar la iluminación de la escena, sí que ayudan a otras clases a generarla. Se ha planteado hacerlo de esta forma ya que los distintos tipos de iluminación tienen parámetros en común que pueden ser recogidos por un objeto y reutilizados para varias iluminaciones distintas. Estas clases son LightColor, LightPosition y LightDirection. Con la creación de estas clases se consigue crear un objeto en las

iluminaciones de tipo color, dirección y posición, que evitan tener que procesar cada uno de los parámetros de estas clases en cada iluminación.

4.1.1 LightColor

LightColor es una clase que se utiliza para establecer el color del tipo de iluminación que se esté implementando. Esta clase tiene como propiedades los valores RGB (rojo, verde y azul), que se utilizan para definir el color de la luz.

Esta clase ha sido creada para poder ser reutilizada en todos los tipos de iluminaciones implementadas, en los cuales se defina el color del haz de luz. La ventaja de la creación de la clase LightColor es que, con tan solo instanciar un objeto de esta clase en las distintas iluminaciones, no hay que añadir los parámetros RGB, porque ya van intrínsecos en el objeto de la clase LightColor.

```
public class LightColor {
    private int red;
    private int green;
    private int blue;

   public LightColor() {
        red = 0;
        green = 0;
        blue = 0;
    }

    public LightColor(int red, int green, int blue) {
        this.red = red;
        this.green = green;
        this.blue = blue;
    }
```

Figura 14. Clase LightColor

En la Figura 14 se muestra la clase LightColor con los parámetros RGB definidos uno a uno, estos parámetros se definen como un valor entero y tienen el valor 0 por defecto, lo cual es la ausencia de color.

Los valores RGB de una determinada iluminación deben definirse en el fichero XML ligado a la clase Escena. Para poder interpretar estos valores se ha hecho una función llamada ProcesaColor dentro de la clase Escena, esta función (ver Figura 15 y Figura 16) es la encargada de leer todos los valores RGB del archivo XML y crear

el objeto de tipo LightColor con los correspondientes valores leídos. El método ProcesaColor es el siguiente:

```
LightColor procesaColor(Node nodeobjeto, String tabs){
    NodeList nodeList = nodeobjeto.getChildNodes();
    System.out.println(tabs+nodeobjeto.getNodeName());
    int red=0, green=0,blue=0;
    for (int i = 0; i < nodeList.getLength(); i++) {
        Node childNode  = nodeList.item(i);
        if(childNode.getNodeType() == Node.ELEMENT_NODE){
            switch(childNode.getNodeName()){
```

Figura 15. Procesamiento de color

```
case "red":
    red=(Integer.parseInt(childNode.getTextContent()));
    System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
    break;
case "green":
    green=(Integer.parseInt(childNode.getTextContent()));
    System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
    break;
case "blue":
    blue=(Integer.parseInt(childNode.getTextContent()));
    System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
    break;
default:
   System.out.println(tabs+"\tNodo no reconocido "+ childNode.getNodeName());
   break;
```

Figura 16. Casos de procesaColor

ProcesaColor acepta como parámetro un objeto de tipo Node y una cadena tabs como argumentos y devuelve un objeto de tipo LightColor. Este objeto crea la iluminación con el color deseado.

Cada color es identificado en el archivo XML con una etiqueta, en este caso las etiquetas que leen los valores RGB tomarían los nombres “red”, “blue” y “green”.

4.1.2 LightDirection

LightDirection (ver Figura 17) es una clase que se utiliza para determinar la dirección en la que se propaga el haz de luz de una iluminación. Esta clase toma como parámetros los valores “nx”, “ny” y “nz” que indican en coordenadas geométricas la dirección en la que se propaga la luz. Esta clase es muy útil cuando se crean luces direccionales o cuando se quiere determinar la dirección con la que

sale la luz de un foco. Al igual que LightColor, LightDirection es reutilizable para todos los tipos de iluminación que la requieran.

```
public class LightDirection {
    private int nx;
    private int ny;
    private int nz;

    public LightDirection(int nx, int ny, int nz) {
        this.nx = nx;
        this.ny = ny;
        this.nz = nz;
    }

    public LightDirection() {
        nx = 0;
        ny = 0;
        nz = 0;
    }
```

Figura 17. Clase LightDirection

Los parámetros anteriores tienen implementados los métodos get y set para hacerlos accesibles desde otras clases.

Los parámetros direccionales son procesados en la clase escena, que toma los valores del fichero XML, para ello ha sido creado una función llamada procesaLightDirection (Figura 18 y Figura 19) que es la encargada de ello.

```
LightDirection procesaLightDirection(Node nodeobjeto, String tabs){
    NodeList nodeList = nodeobjeto.getChildNodes();
    System.out.println(tabs+nodeobjeto.getNodeName());
    int nx=0, ny=0,nz=0;
    for (int i = 0; i < nodeList.getLength(); i++) {
        Node childNode  = nodeList.item(i);
        if(childNode.getNodeType() == Node.ELEMENT_NODE){
            switch(childNode.getNodeName()){
```

Figura 18. Procesamiento de dirección

```
        if(childNode.getNodeType() == Node.ELEMENT_NODE){
            switch(childNode.getNodeName()){
                case "nx":
                        nx=(Integer.parseInt(childNode.getTextContent()));
                        System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
                        break;
                case "ny":
                        ny=(Integer.parseInt(childNode.getTextContent()));
                        System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
                        break;
                case "nz":
                        nz=(Integer.parseInt(childNode.getTextContent()));
                        System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
                        break;
                default:
                    System.out.println(tabs+"\tNodo no reconocido "+ childNode.getNodeName());
                    break;
            }
        }
    }
    return new LightDirection(nx,ny,nz);
}
```

Figura 19. Casos de procesaLightDirection

Este método se encarga de leer y procesar las etiquetas de dirección las cuales serán “nx”, “ny” y “nz” que puede pertenecer tanto a luces direccionales como a focos.

4.1.3 LightPosition

LightPosition (ver Figura 20) es una clase creada para determinar el punto desde el que se empieza a iluminar una escena, el ángulo de salida del haz de luz y la concentración con la que ilumina. Esta clase ha sido creada para la creación de focos los cuales tienen un punto concreto desde el que se inicia el haz de luz a diferencia de la iluminación direccional que recorre todo un plano en la dirección indicada.

La clase LightPosition dispone de los siguientes atributos que la definen:

```
public class LightPosition {
    private int x;
    private int y;
    private int z;
    private float angle;
    private int concentration;

    public LightPosition(int x, int y, int z, float angle, int concentration) {
        this.x = x;
        this.y = y;
        this.z = z;
        this.angle = angle;
        this.concentration = concentration;
    }

    public LightPosition() {
        x = 0;
        y = 0;
        z = 0;
        angle = 0;
        concentration = 0;
    }
```

Figura 20. Clase LightPosition

Los parámetros “x”, “y” y “z” indican la posición en la que se empieza a crear el punto de luz, mientras que el ángulo indica, el ángulo con el que sale y la concentración la apertura de la luz.

Los parámetros de la función LightPosition son procesados por la clase Escena que los lee del fichero XML ligado a ella. Para procesar todos estos parámetros se ha creado una función llamada procesaLightPosition (Figura 21 y Figura 22) dentro de la clase Escena.

```
LightPosition procesaLightPosition(Node nodeobjeto, String tabs){
  NodeList nodeList = nodeobjeto.getChildNodes();
  System.out.println(tabs+nodeobjeto.getNodeName());
  int x=0, y=0,z=0, concentration=0;
  float angle =0;
  for (int i = 0; i < nodeList.getLength(); i++) {
      Node childNode  = nodeList.item(i);
      if(childNode.getNodeType() == Node.ELEMENT_NODE){
          switch(childNode.getNodeName()){
```

Figura 21. Procesamiento de posición

```
            case "x":
                x=(Integer.parseInt(childNode.getTextContent()));
                System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
                break;
            case "y":
                y=(Integer.parseInt(childNode.getTextContent()));
                System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
                break;
            case "z":
                z=(Integer.parseInt(childNode.getTextContent()));
                System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
                break;
            case "angle":
                angle=(Float.parseFloat(childNode.getTextContent()));
                System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
                break;
            case "concentration":
                concentration=(Integer.parseInt(childNode.getTextContent()));
                System.out.println(tabs+"\t "+childNode.getNodeName()+ "\t"+childNode.getTextContent());
                break;
            default:
                System.out.println(tabs+"\tNodo no reconocido "+ childNode.getNodeName());
                break;
        }
    }
}
return new LightPosition(x,y,z,angle,concentration);
```

Figura 22. Casos de procesaLightPosition

ProcesaLightPosition toma como parámetro un objeto de tipo node y una cadena de tabs como argumentos y devuelve un objeto de tipo LightPosition. Este objeto establece la posición, el ángulo y la concentración de un foco.

Cada uno de los parámetros son identificados con diferentes etiquetas, las cuales asignan el valor correspondiente a cada uno de los parámetros. En este caso las etiquetas tienen los nombres “x”, “y”, “z”, “angle” y “concentration”.

4.2 Tipos de iluminación implementados

Los tipos de iluminación que han sido implementados para configurar la escena son: iluminación ambiental de la escena, iluminaciones direccionales de la escena, iluminación especular de los objetos, desvanecimiento de la luz en la escena y focos. Se puede visualizar un diagrama con los tipos de iluminación en la Figura 23.

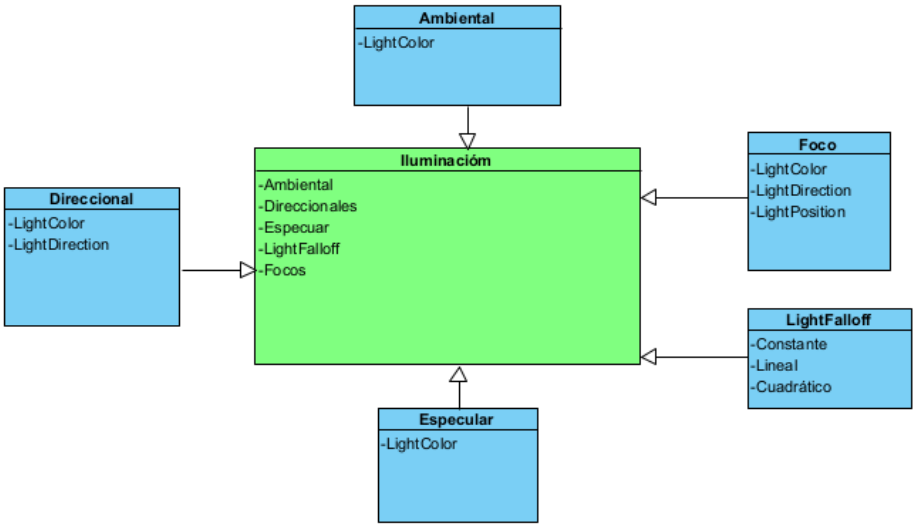


Figura 23. Diagrama de clases de tipos de iluminación

4.2.1 Iluminación ambiental

La iluminación ambiental (Processing, s.f.) es una fuente de luz omnidireccional que no tiene una ubicación, ni una dirección específica en la escena. Esta iluminación da un ambiente general a la iluminación de la escena.

En Processing, la luz ambiental se puede utilizar para agregar una iluminación de fondo, uniforme y suave a una escena.

Para utilizar la luz ambiental en Processing, primero hay que crear una instancia de la clase ambientLight. Esta es una función que se utiliza para establecer la iluminación ambiente en una escena 3D. Esta función toma tres argumentos: el valor rojo, verde y azul, los cuales son los valores RGB. Estos valores deben estar en el rango de 0 a 255. Es importante tener en cuenta que la luz ambiental no produce sombras o reflejos realistas.

Para ver con más detalle cómo funciona la iluminación ambiental se muestra un ejemplo en el entorno de Processing de su funcionamiento en la Figura 24.

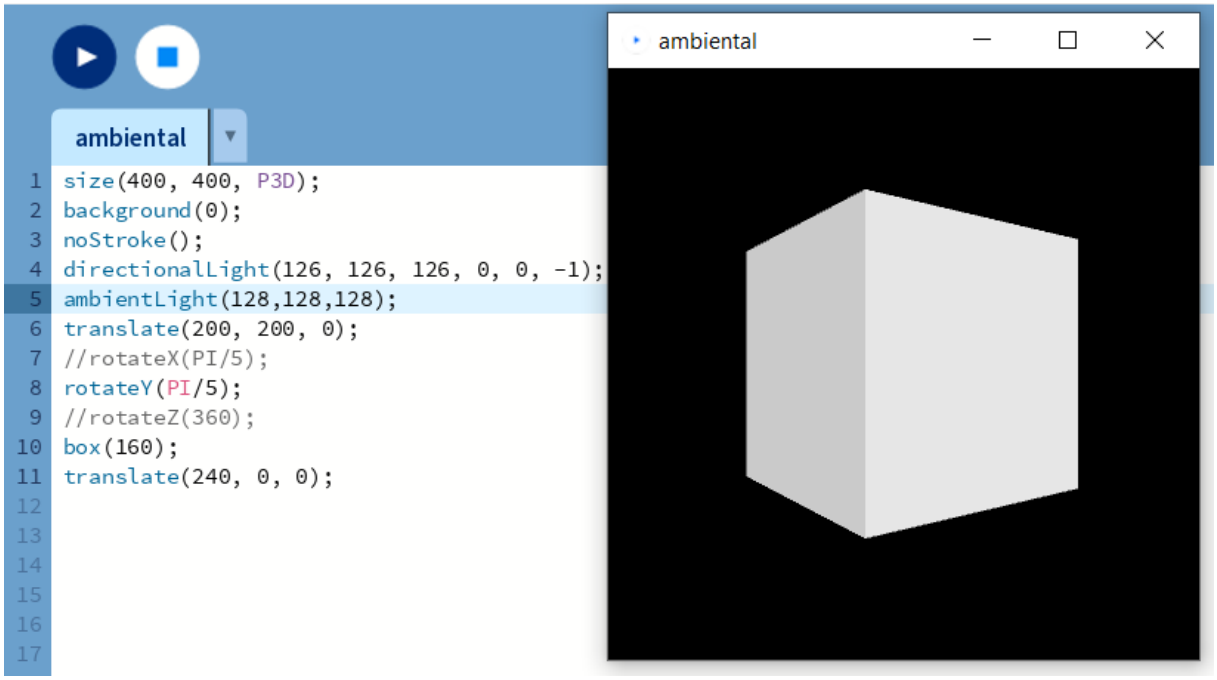


Figura 24. Ejemplo de Processing, luz ambiental blanca

Para comprender mejor cómo se trabaja en Processing, se explica el código de la Figura 24 línea a línea para su perfecta comprensión:

size(400, 400, P3D), establece el tamaño de la ventana en la que se mostrará la escena. En este caso, se ha establecido en 400 píxeles de ancho y 400 píxeles de alto, y se utiliza el modo de renderizado P3D para renderizar gráficos 3D.

background(0): Establece el color de fondo de la ventana en negro (0,0,0)

noStroke(): Establece que no se dibujen bordes para la caja.

directionalLight(126,126,126,0,0,-1): en este caso se ha añadido también una luz direccional para que sean perceptibles los bordes de a caja.

ambientLight(128,128,128): Crea una luz ambiental que ilumina la escena desde todas las direcciones. En este caso, se utiliza una luz gris claro.

translate(200, 200, 0): Cambia el origen de coordenadas a (200,200,0), lo que significa que los siguientes comandos de dibujo se realizarán en relación con ese punto en la escena.

rotateY(PI/5): Rota la caja alrededor del eje Y en un ángulo de 36 grados (PI/5 radianes).

box(160): Dibuja una caja de 160 píxeles de ancho, 160 píxeles de alto y 160 píxeles de profundidad en la posición actual del origen de coordenadas.

translate(240,0,0): Cambia el origen de coordenadas a (240,0,0), lo que significa que los siguientes comandos de dibujo se realizarán en relación con ese punto en la escena.

Una vez visto este ejemplo, para ver cómo se puede cambiar el color de la luz ambiental se hace lo siguiente:

Para cambiar el color de la luz ambiental se debe de cambiar los parámetros RGB, por ejemplo, se puede asignar un valor de 255 al color rojo y de 20 al verde y azul, de este modo se obtiene una luz ambiental roja (ver Figura 25).

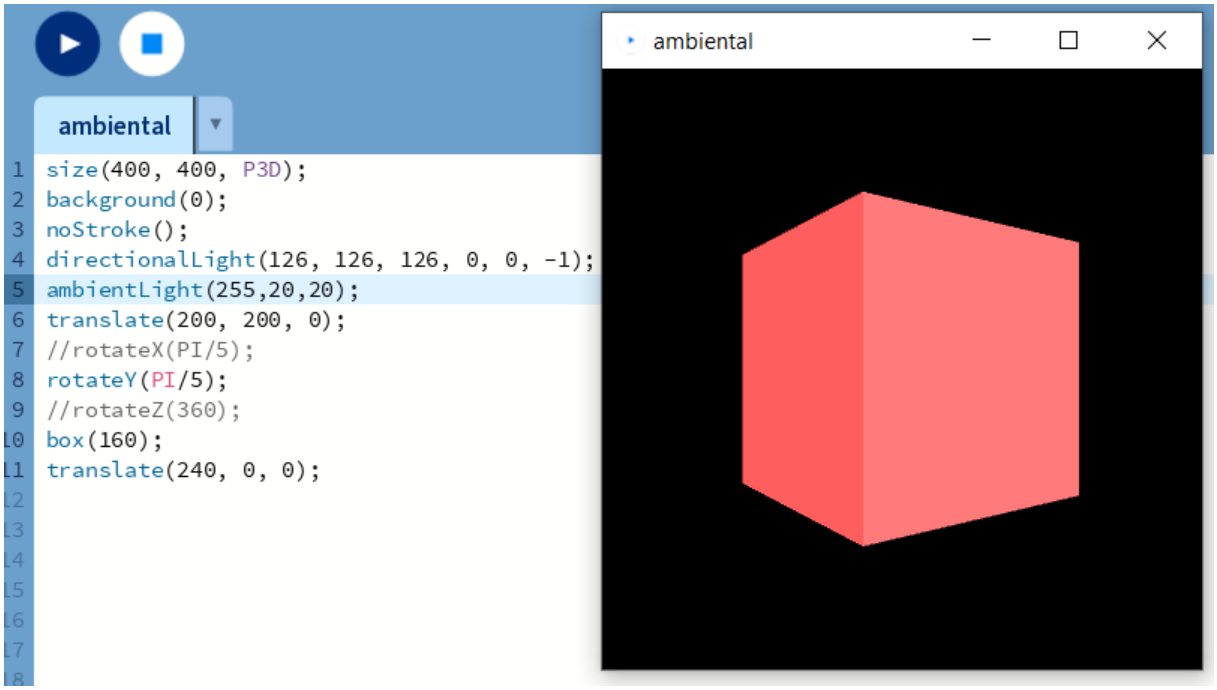


Figura 25. Ejemplo de Processing, luz ambiental roja

Del mismo modo si se quiere obtener una luz ambiental azul, se sube el parámetro de azul a 255 y el resto se sustituye por un valor de 20, (ver Figura 26).

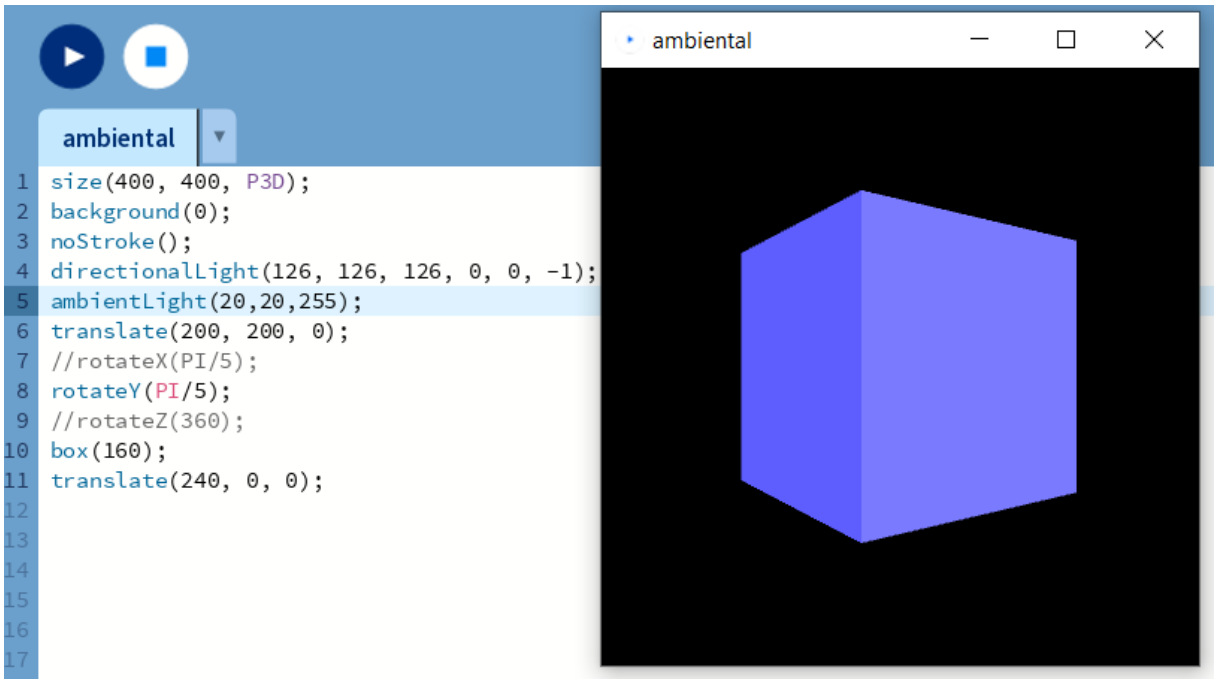


Figura 26. Ejemplo de Processing, luz ambiental azul

Sirva como introducción cómo se crea una iluminación ambiental en el entorno de Processing para ahora ver, como se ha adaptado para introducirla en la plataforma de gemelos digitales.

Para implementar una iluminación ambiental en la plataforma se ha creado la clase llamada Ambiental (Figura 27), esta clase será la encargada de crear la luz ambiental dentro la escena.

```
public class Ambiental {

    LightColor color;

    public Ambiental() {
        color = new LightColor();
    }

    public Ambiental(LightColor color) {
        this.color = color;
    }

    public int getRed() {
        return color.getRed();
    }

    public int getGreen() {
        return color.getGreen();
    }

    public int getBlue() {
        return color.getBlue();
    }
}
```

Figura 27. Clase Ambiental

La clase ambiental recibe un objeto de tipo LightColor, este objeto será el que proporciona el color a la luz ambiental que se añada a la escena. Para crear una iluminación ambiental se necesita tener acceso a los parámetros red, green y blue, por ello es que los métodos get los colores están implementados en la clase.

La iluminación es procesada en la clase Escena, por el método ProcesaIluminación que obtiene los valores del archivo XML. Estos valores serian introducidos de la forma que muestra la Figura 28.

```
<ambiental>
    <red>128</red>
    <green>128</green>
    <blue>128</blue>
</ambiental>
```

Figura 28. Luz ambiental blanca en XML

Dentro de la etiqueta <iluminacion> se debe de añadir la etiqueta <ambiental> para añadir dentro de ella a su vez las etiquetas de los parámetros RGB, en este ejemplo se han añadido los valores de 128 a cada uno de los parámetros para crear una luz uniforme blanca de una intensidad media. De este modo se obtiene una luz ambiental blanca de media intensidad (ver Figura 29).

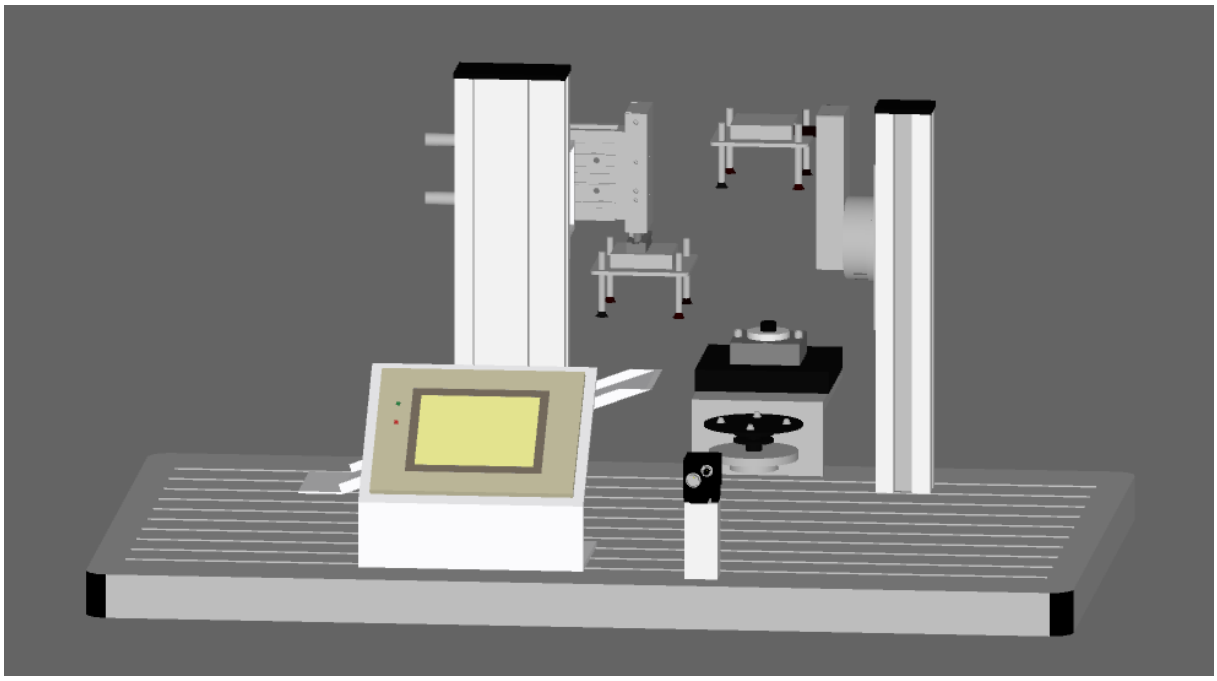


Figura 29. Luz ambiental blanca en escena

Para cambiar el color o la intensidad de la luz ambiental, se deben cambiar los valores en el archivo XML, subiendo o bajando los valores RGB proporcionalmente para una mayor o menor intensidad lumínica, o alterando el valor de los colores de manera que destaquen las tonalidades que se desean.

Para disminuir la intensidad lumínica, se añade una luz ambiental en la que todos los valores RGB son de un valor de 40, un valor bastante bajo comparado con el valor de 128 anterior. Se muestra en la Figura 30 cómo se debe modificar el archivo XML y el resultado en la escena en la Figura 31.

```
<ambiental>
    <red>40</red>
    <green>40</green>
    <blue>40</blue>
</ambiental>
```

Figura 30. Luz ambiental de brillo bajo, XML

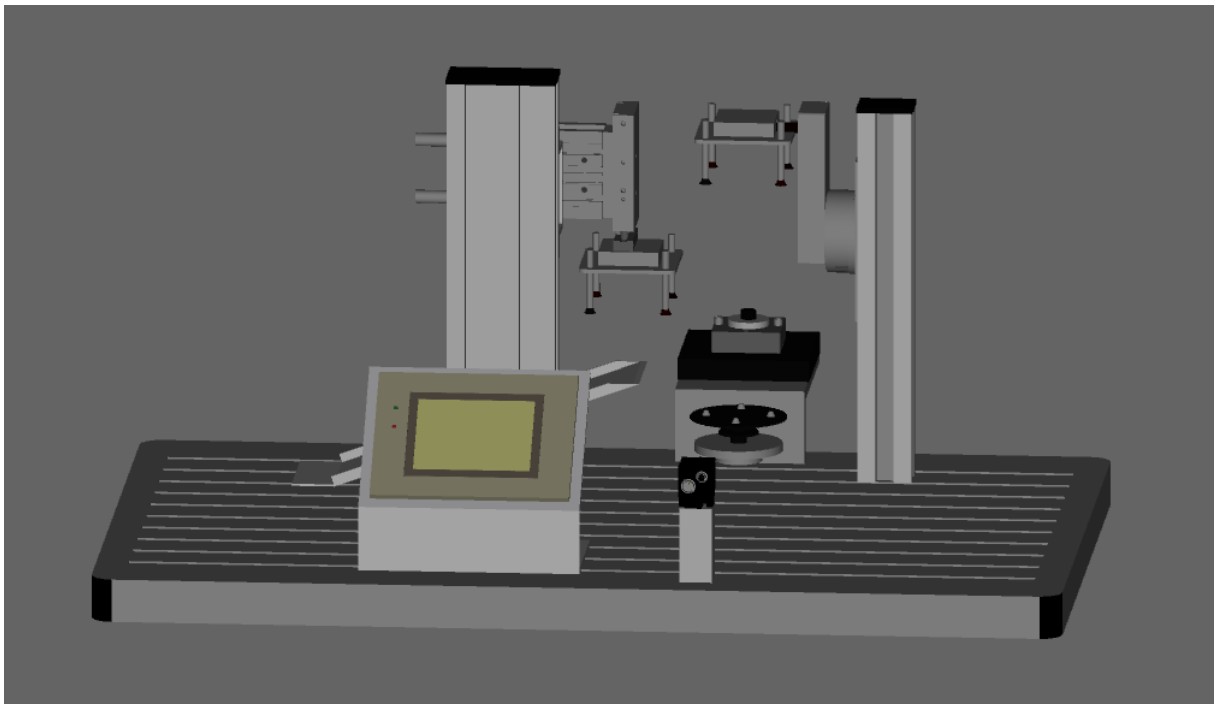


Figura 31. Luz ambiental de brillo bajo en escena

Si se quiere dar una tonalidad distinta a la escena, por ejemplo, con un haz de luz verde, se pude subir el valor del color verde a 240 y dejando el resto como al principio en 128 (ver Figura 32) y su correspondiente resultado en la aplicación en la Figura 33.

```
<ambiental>
    <red>128</red>
    <green>240</green>
    <blue>128</blue>
</ambiental>
```

Figura 32. Luz ambiental verde en XML

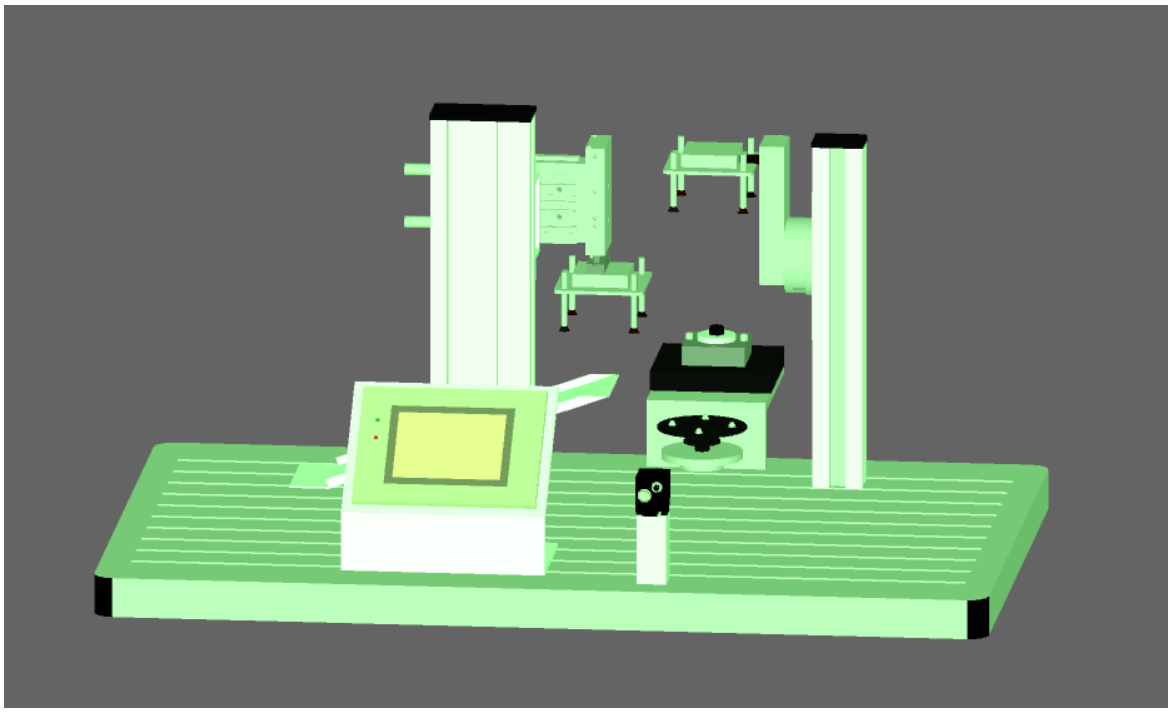


Figura 33. Luz ambiental verde, en escena

4.2.2 Iluminación direccional

La iluminación direccional es (Processing, s.f.), como su nombre indica un tipo de iluminación que proviene únicamente desde una dirección. Al proporcionar una dirección específica, se pueden resaltar texturas y detalles de los objetos 3D. También puede ser usada para crear un ambiente específico. En la plataforma de gemelos digitales las iluminaciones direccionales tienen un papel importante para crear el ambiente de la escena deseado.

Para crear una luz direccional en el entorno de Processing, hay que hacer una instancia a la clase directionalLight. Esta función toma seis parámetros para generar la luz direccional, los tres primeros son referidos a los valores RGB, los tres siguientes se refieren a la dirección tridimensional de la luz.

Para entender exactamente cómo funcionan la dirección del haz de luz según las coordenadas proporcionadas se exponen los siguientes ejemplos:

(1, 0, 0): la luz apunta hacia la derecha en la dirección positiva del eje x.

(-1, 0, 0): la luz apunta hacia la izquierda en la dirección negativa del eje x.

(0, 1, 0): la luz apunta hacia arriba en la dirección positiva del eje y.

(0, -1, 0): la luz apunta hacia abajo en la dirección negativa del eje y.

(0, 0, -1): la luz apunta hacia adelante en la dirección negativa del eje z.

(0, 0, 1): la luz apunta hacia atrás en la dirección positiva del eje z.

(1, 1, 0): la luz apunta hacia la esquina superior derecha en un ángulo de 45 grados en los ejes x e y.

(-1, -1, 0): la luz apunta hacia la esquina inferior izquierda en un ángulo de 45 grados en los ejes x e y.

En el entorno de Processing se puede generar una luz direccional de la forma en la que se representa en la Figura 34.

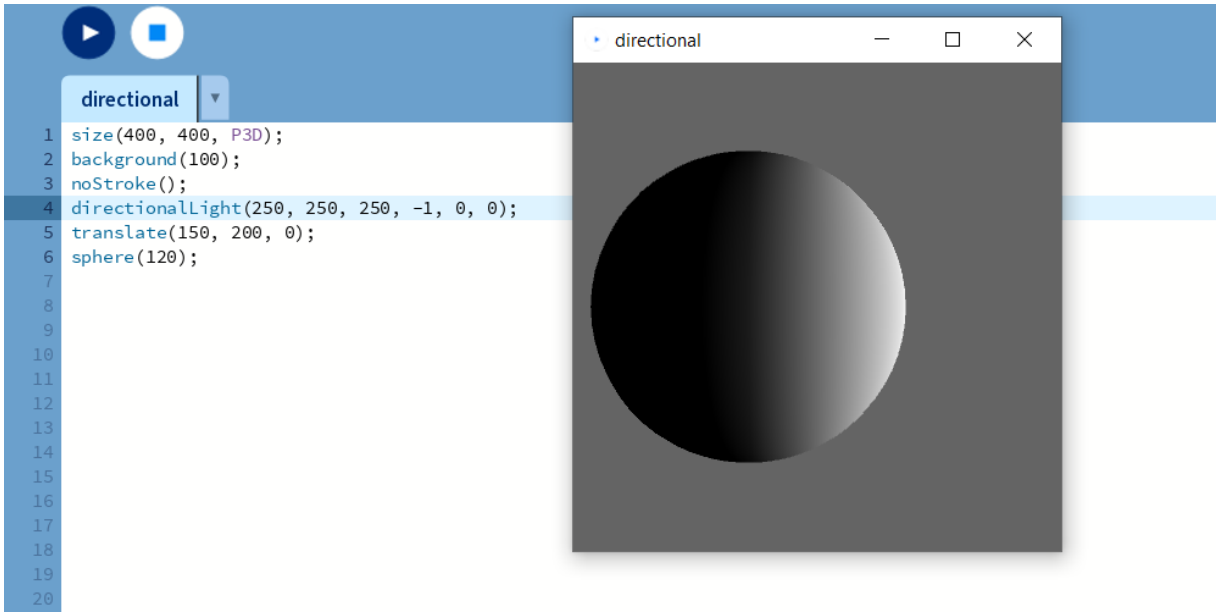


Figura 34. Ejemplo de Processing Luz direccional blanca dirección (-1,0,0)

Este código explicado línea a línea haría lo siguiente:

Size(400,400,P3D); define el tamaño de la ventana de dibujo de píxeles y el modo de renderizado en 3D. En esta ocasión la ventana es de 400x400 píxeles.

Background(100); define un fondo de escena de tono grisáceo

noStroke(0); establece que no haya borde en la esfera que se va a dibujar

directionalLight(250,250,250); establece un haz de luz blanca con una dirección (-1,0,0), es decir que apunta hacia la izquierda.

Translate(150,200,0); mueve la esfera de modo que se dibuje desplazada

Sphere(120); dibuja una esfera de radio 120 píxeles, que tendrá como centro las anteriores coordenadas

Si se cambian las coordenadas, estableciendo y=-1 y z=0, se obtiene un haz de luz blanco proveniente desde abajo (ver Figura 35).

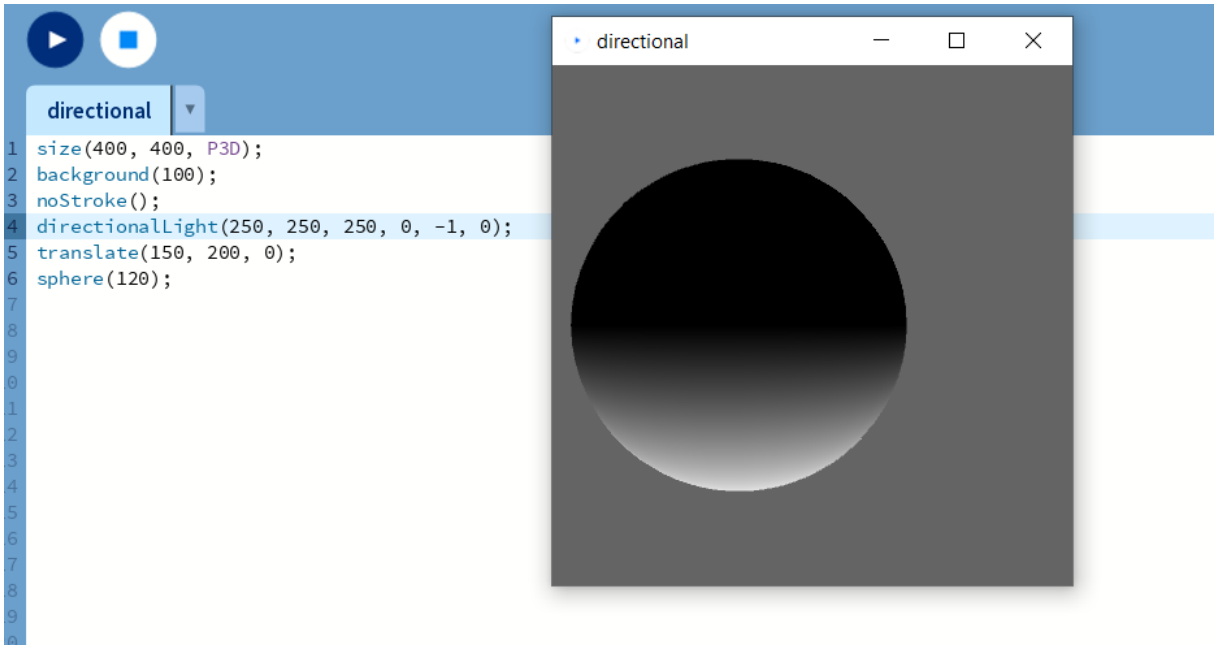


Figura 35. Ejemplo de Processing. Luz direccional blanca dirección (0,-1,0)

Del mismo modo que con la iluminación ambiental, se puede cambiar el color del haz de luz cambiando los valores RGB. Si se disminuyen los valores de verde y azul, pero se mantiene el de rojo, se obtiene un haz de luz roja. (ver Figura 36).

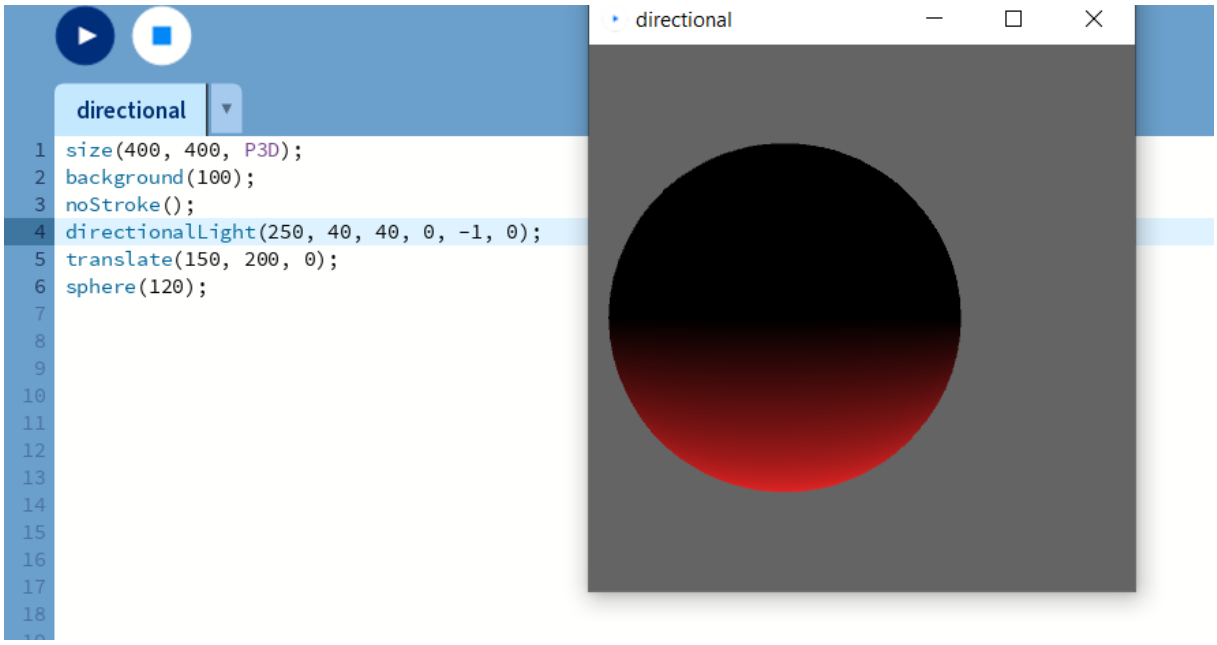


Figura 36. Ejemplo de Processing. Luz direccional roja dirección (0,-1,0)

Para implementar este tipo de iluminación a la plataforma de gemelos digitales se ha implementado una clase llamada LuzDireccional (Figura 37), esta clase utiliza las clases LightColor y LightDirection para crear una luz direccional.

```
public class LuzDireccional {

    LightColor color;
    LightDirection direccion;

    public LuzDireccional() {
        color = new LightColor();
        direccion= new LightDirection();
    }

    public LuzDireccional(LightColor color, LightDirection posicion) {
        this.color = color;
        this.direccion = posicion;
    }
```

Figura 37. Clase LuzDireccional

Se ha implementado los métodos get de los parámetros “red”, “blue”, “green”, “x”, “y” y “z” para que sea posible acceder a todos los parámetros de color y posición para el objeto de tipo LuzDireccional.

Como las clases anteriores, esta clase es procesada por la clase escena que va ligada al fichero XML, un ejemplo de creación de una luz direccional en la

plataforma de gemelos digitales se visualiza en la Figura 38 y su resultado en la Figura 39.

```
<luzDireccional>
    <red>200</red>
    <green>200</green>
    <blue>200</blue>
    <x>0</x>
    <y>1</y>
    <z>0</z>
</luzDireccional>
```

Figura 38. Ejemplo de luz direccional proveniente desde arriba, XML

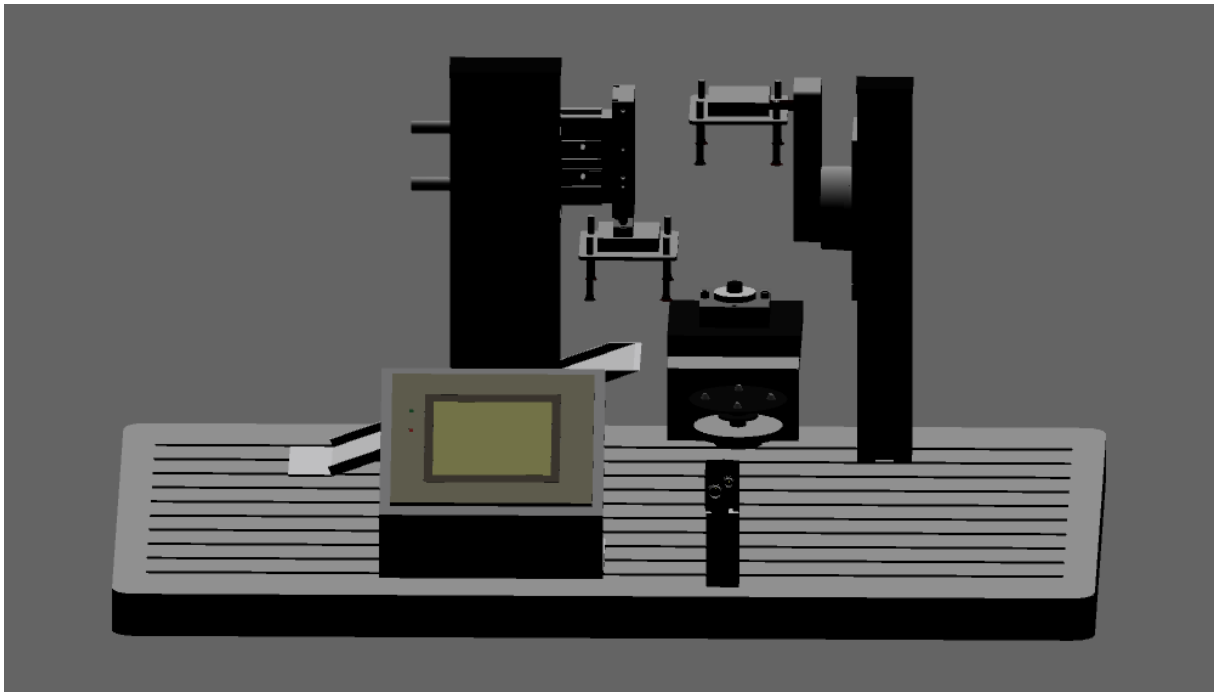


Figura 39. Ejemplo luz direccional proveniente desde arriba, escena

De este modo añadiendo los parámetros x=0, y=1 y z=0 se añade una luz direccional blanca, la cual sale desde arriba.

Si se cambian los parámetros para que el color del haz de luz sea azul, subiendo dicho parámetro a 200 y se cambia la dirección de la luz a (0,0,-1) se obtiene una iluminación frontal. Los datos XML se observan Figura 39 y el resultado en la escena en la Figura 40.

```
<luzDireccional>
    <red>20</red>
    <green>20</green>
    <blue>200</blue>
    <x>0</x>
    <y>0</y>
    <z>-1</z>
</luzDireccional>
```

Figura 40. Ejemplo de luz frontal azul, XML.

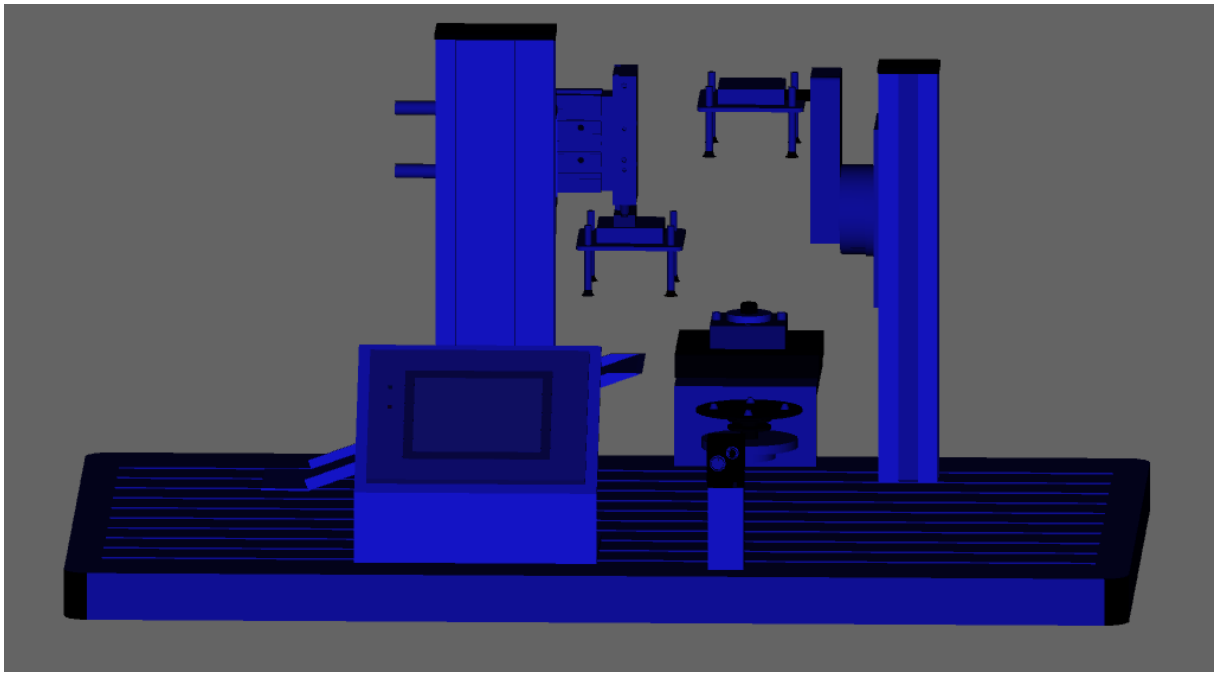


Figura 41. Ejemplo de luz frontal azul, escena

4.3.3 Iluminación especular

La luz especular es aquella que se refiere a la forma en la que la luz se refleja en la superficie de un objeto, creando un resplandor brillante y reflectante en la dirección de la luz incidente. La luz especular se llama así porque sigue las leyes de la reflexión especular, que establecen que el ángulo de incidencia de la luz es igual al ángulo de reflexión.

La luz especular es una técnica comúnmente utilizada en la renderización de gráficos en 3D para simular la apariencia de objetos metálicos, vidrio y otros materiales reflectantes.

En Processing, (Processing, s.f.) se puede utilizar la función LightSpecular para agregar luz especular a los objetos renderizados de una escena. A la función LightSpecular se le pasan 3 parámetros que son los valores RGB. Para poder ver cómo funciona se plantea el siguiente ejemplo en la Figura 42.

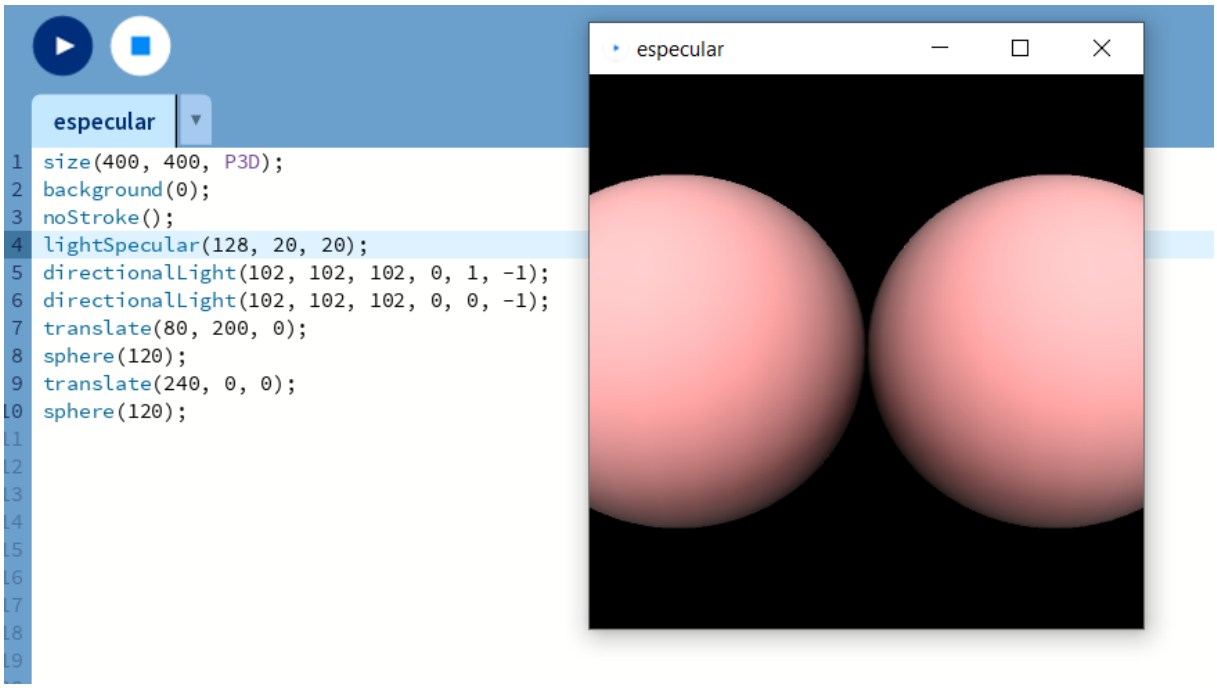


Figura 42. Ejemplo en Processing de luz especular rojiza

En este ejemplo se añade una luz especular con valores RGB que crean un reflejo rojizo. Para que pueda ser perceptible la luz especular se han añadido dos luces direccionales, una proveniente de frente y otra desde arriba de frente. Hay que resaltar que la función LightSpecular debe ser definida antes de las luces direccionales para que funcione correctamente.

En el siguiente ejemplo de la Figura 43 y Figura 44 se puede ver como aumentando proporcionalmente los valores de la función de luz especular se puede aumentar la intensidad de reflexión de la luz, se representa manteniendo los mismos valores de luz direccional para que quede constancia de que el valor que afecta en este caso a la iluminación de la escena es la luz especular.

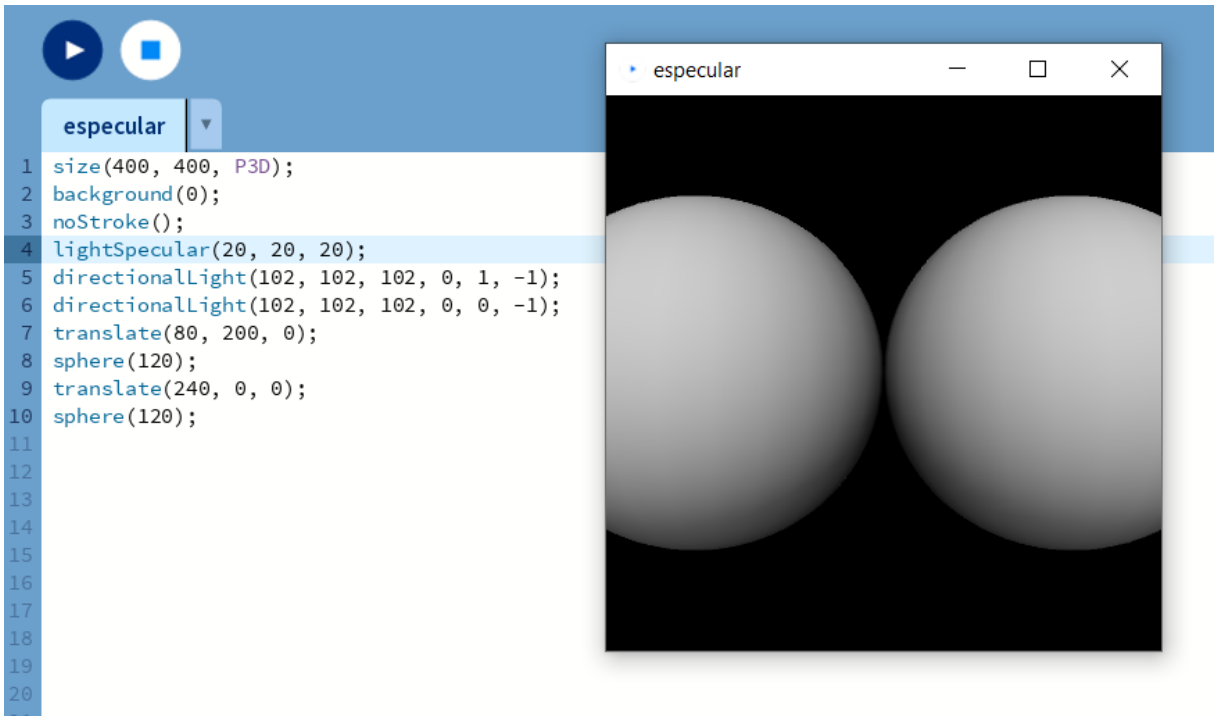


Figura 43. Ejemplo en Processing de luz especular de baja intensidad

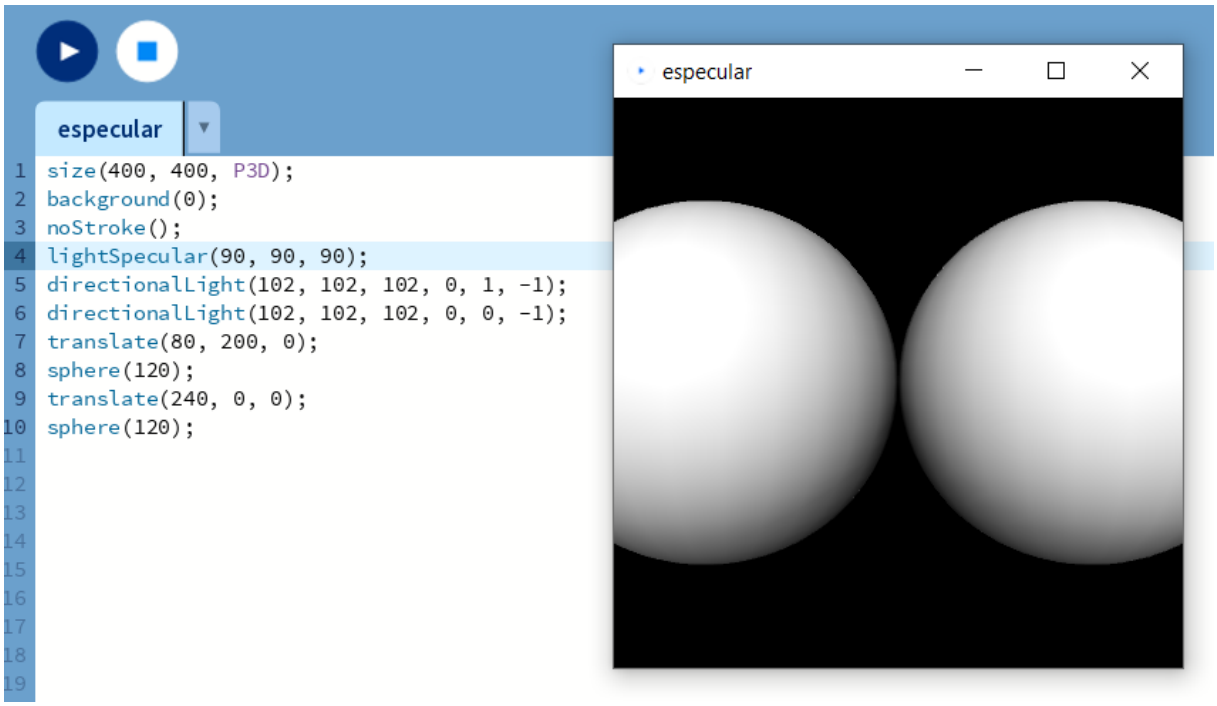


Figura 44. Ejemplo en Processing de luz especular de media intensidad

Una vez visto cómo funciona en Processing la luz especular, se puede ver cómo ha sido esta misma introducida en la plataforma de gemelos digitales.

Para la creación de la luz especular se ha creado una función llamada LuzEspecular (Figura 45), esta clase como la clase Ambiental, solo necesita un objeto de tipo LightColor para ser definida.

```
public class LuzEspecular {
    LightColor color;

    public LuzEspecular(){
        color = new LightColor();
    }

    public LuzEspecular(LightColor color) {
        this.color = color;
    }

    public int getRed() {
        return color.getRed();
    }

    public int getGreen() {
        return color.getGreen();
    }

    public int getBlue() {
        return color.getBlue();
    }

}
```

Figura 45. Clase LuzEspecular

La luz especular es procesada en la clase Escena, que toma los valores de los parámetros de LuzEspecular del fichero XML, siguiendo el mismo estilo de etiquetas que las clases anteriores. Siendo así si se quiere añadir una luz especular que cause reflejos de un tono morado los parámetros pueden tomar los valores observados en Figura 46 y su correspondiente efecto en la escena en la Figura 47.

```
<luzEspecular>
    <red>190</red>
    <green>30</green>
    <blue>190</blue>
</luzEspecular>
```

Figura 46. Luz especular morada, XML

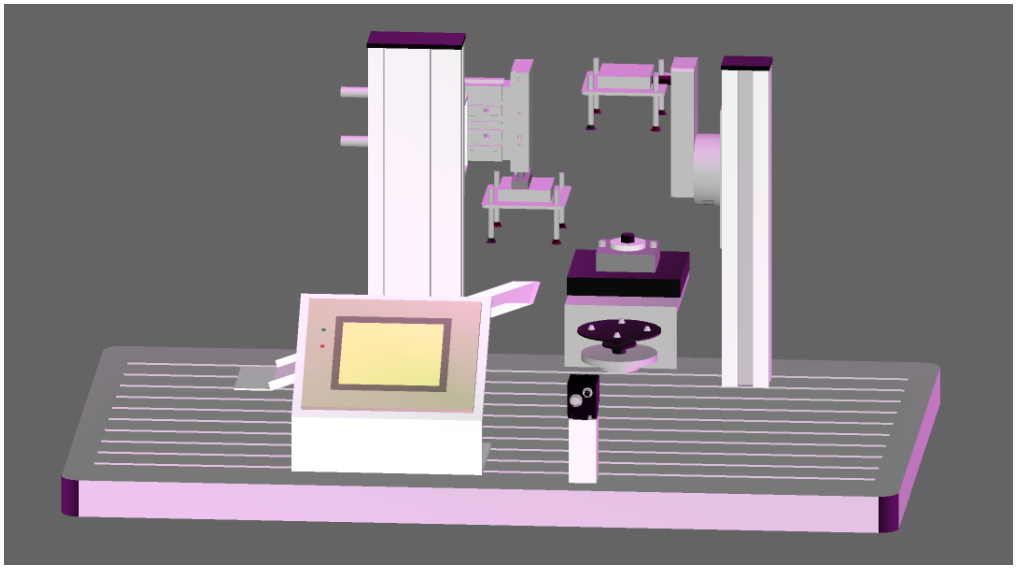


Figura 47. Luz especular morada, escena

Los reflejos de la luz especular no tienen por qué ser tan llamativos como en la Figura 47 en la que se ve una reflexión de tonalidad morada, de modo que, si solo se quiere aumentar la intensidad de la luz reflejada, se puede optar por añadir valores de colores que sean iguales y aumente de manera proporcional, de este modo si se le asigna un valor de 200 a los 3 parámetros RGB como se ve en la Figura 48 se obtenga una escena con un alto reflejo de luz especular.

```
<luzEspecular>
    <red>200</red>
    <green>200</green>
    <blue>200</blue>
```

Figura 48. Luz especular de alta intensidad, XML

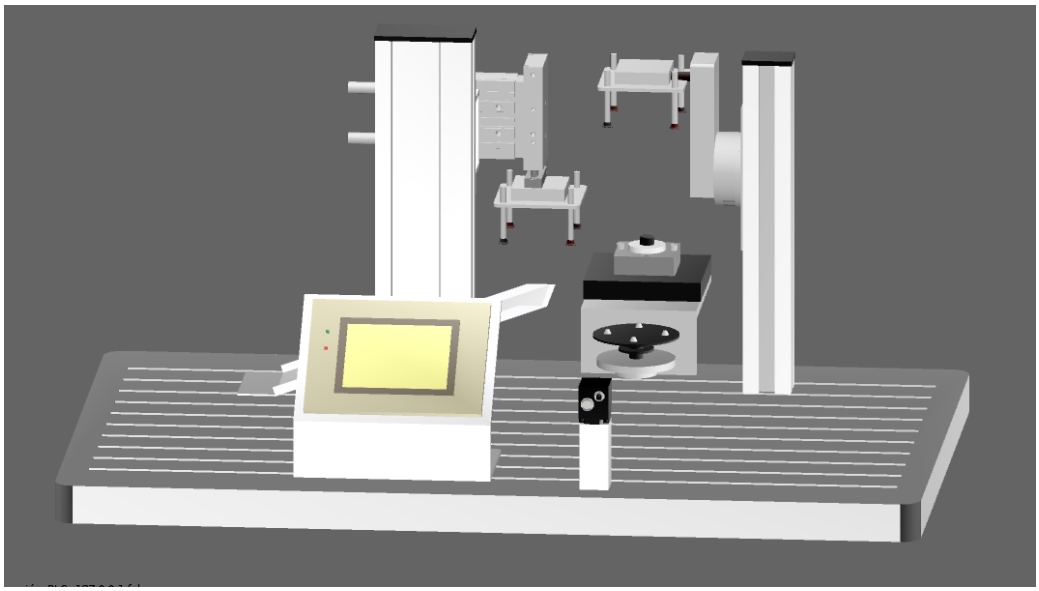


Figura 49. Luz especular de alta intensidad, escena

Se obtiene un brillo bastante alto y una gran cantidad de luz reflejada (ver Figura 49) por las superficies de la escena, un valor más apropiado para la escena podría ser un valor de 128 (Figura 50) para cada uno de los valores RGB, obteniendo con estos valores una cantidad de reflejo que no resulte demasiado alta (Figura 51).

```
<luzEspecular>
    <red>128</red>
    <green>128</green>
    <blue>128</blue>
</luzEspecular>
```

Figura 50. Luz especular de intensidad media

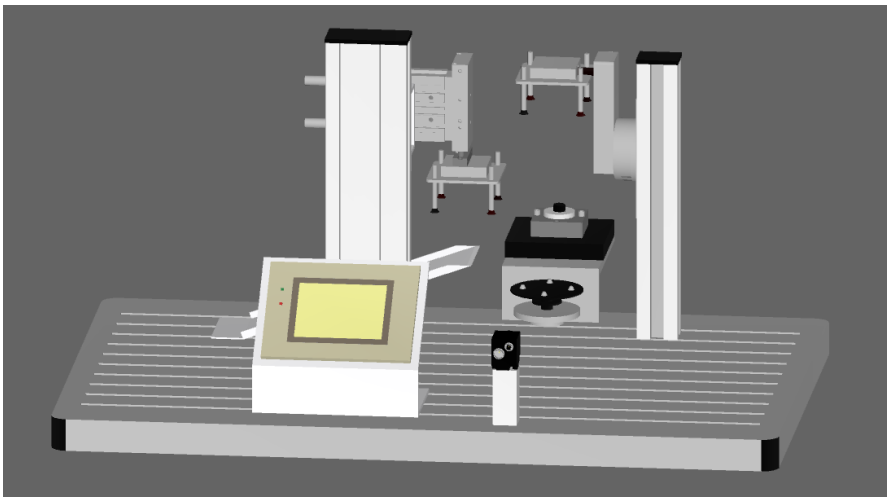


Figura 51. Luz especular de intensidad media, escena

De este modo se consigue una luz especular adecuada en primera instancia, para una visualización realista de la escena. Asignando uno valores iguales de 128 que proporcionan unos reflejos de luz idóneos para la iluminación de la estación.

4.2.4 Caída de luz

La caída de luz o generación de sombras es la encargada de que los objetos parezcan estar situados en la escena en un espacio tridimensional, para ello debe comportarse de manera coherente con la iluminación proporcionada. Se debe tener en cuenta que la caída de luz no afecta a la iluminación ambiental, ni a la especular.

En Processing (Processing, s.f.) la caída de luz se realiza con la función LightFalloff(), este método se basa en 3 parámetros para determinar la caída de luz de una escena. Los parámetros son constant, linear y quadratic.

El parámetro "constant" se refiere a la cantidad de luz que se mantiene constante en toda la escena, independientemente de la distancia a la fuente de luz. Este valor representa la cantidad de luz que llega directamente desde la fuente de luz y no se atenúa en absoluto. Un valor alto de "constant" significa que la luz será menos atenuada, lo que se traduce en una iluminación más uniforme en la escena. Normalmente, se utilizan valores entre 0 y 1 para este parámetro, donde un valor de 0 significa que no hay luz constante y un valor de 1 significa que la cantidad de luz es constante en toda la escena.

El parámetro "lineal" controla la cantidad de luz que se atenúa de forma lineal a medida que se aleja de la fuente de luz. Este valor representa la cantidad de luz que se refleja en las superficies de la escena y se dispersa en todas las direcciones, disminuyendo en intensidad a medida que se aleja de la fuente. Un valor alto de "linear" significa que la luz se atenuará rápidamente a medida que se aleja de la fuente, lo que puede resultar en sombras más oscuras y una iluminación menos uniforme. Normalmente, se utilizan valores pequeños entre 0 y 0.1 para este parámetro, dependiendo de la distancia de la fuente de luz y del tamaño del objeto que se esté iluminando.

El parámetro "quadratic" controla la cantidad de luz que se atenúa de forma cuadrática a medida que se aleja de la fuente de luz. Este valor representa la

cantidad de luz que se dispersa en todas las direcciones y se atenúa aún más a medida que se aleja de la fuente de luz. Un valor alto de "quadratic" significa que la luz se atenuará muy rápidamente a medida que se aleja de la fuente, lo que puede resultar en sombras muy oscuras y una iluminación muy desigual. Normalmente, se utilizan valores aún más pequeños entre 0 y 0.01 para este parámetro, dependiendo de la distancia de la fuente de luz y del tamaño del objeto que se esté iluminando.

Para añadir esta función al sistema de iluminación se ha creado una clase llamada LightFalloff (Figura 52), esta clase tiene como parámetros los parámetros ya mencionados constante, lineal y cuadrático.

```
public class LightFalloff {

    private float constante;
    private float lineal;
    private float cuadratico;



    public LightFalloff() {
        constante=1;
        lineal=0;
        cuadratico=0;
    }

    public LightFalloff(float constante, float lineal, float cuadratico) {
        this.constante = constante;
        this.lineal = lineal;
        this.cuadratico = cuadratico;
    }
```

Figura 52. Clase LightFalloff

En este caso los atributos deben de ser definidos como flotantes. Es importante tener en cuenta que como en Processing, la caída de luz debe ser procesada antes que las iluminaciones direccionales y los focos para que funcione correctamente. Los datos de valores constante, lineal y cuadrático se proporcionan en el archivo XML, por defecto se establece un valor constante=1, lineal=0 y cuadrático 0. Se introducen dichos valores como en la Figura 53 y se obtiene una escena como la de la Figura 54.

```
<lightFalloff>
    <constante>1</constante>
    <lineal>0</lineal>
    <cuadratico>0</cuadratico>
</lightFalloff>
```

Figura 53. Desvanecimiento de luz constante, XML

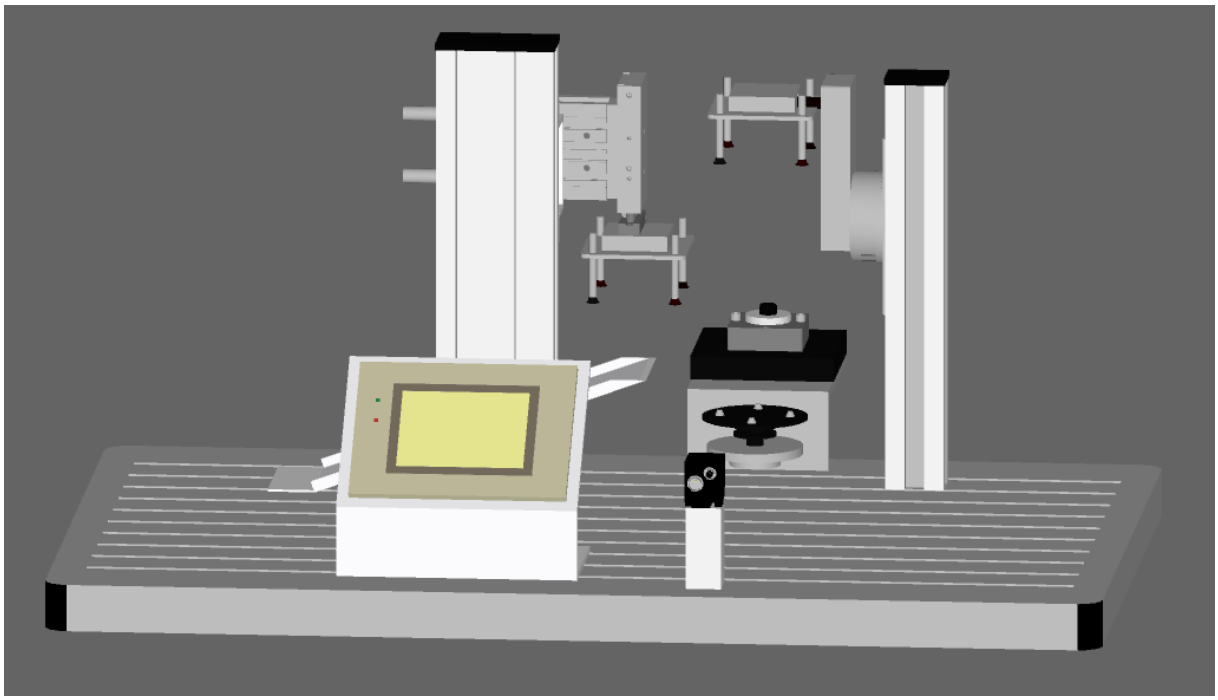


Figura 54. Desvanecimiento de luz constante, escena

Para aumentar la generación de sombras se puede añadir un valor de 0.1 al desvanecimiento de luz lineal (Figura 55), para obtener un desvanecimiento de luz más pronunciado observado en la Figura 56.

```
<lightFalloff>
    <constante>1</constante>
    <lineal>0.1</lineal>
    <cuadratico>0</cuadratico>
</lightFalloff>
```

Figura 55. Desvanecimiento de luz lineal, XML

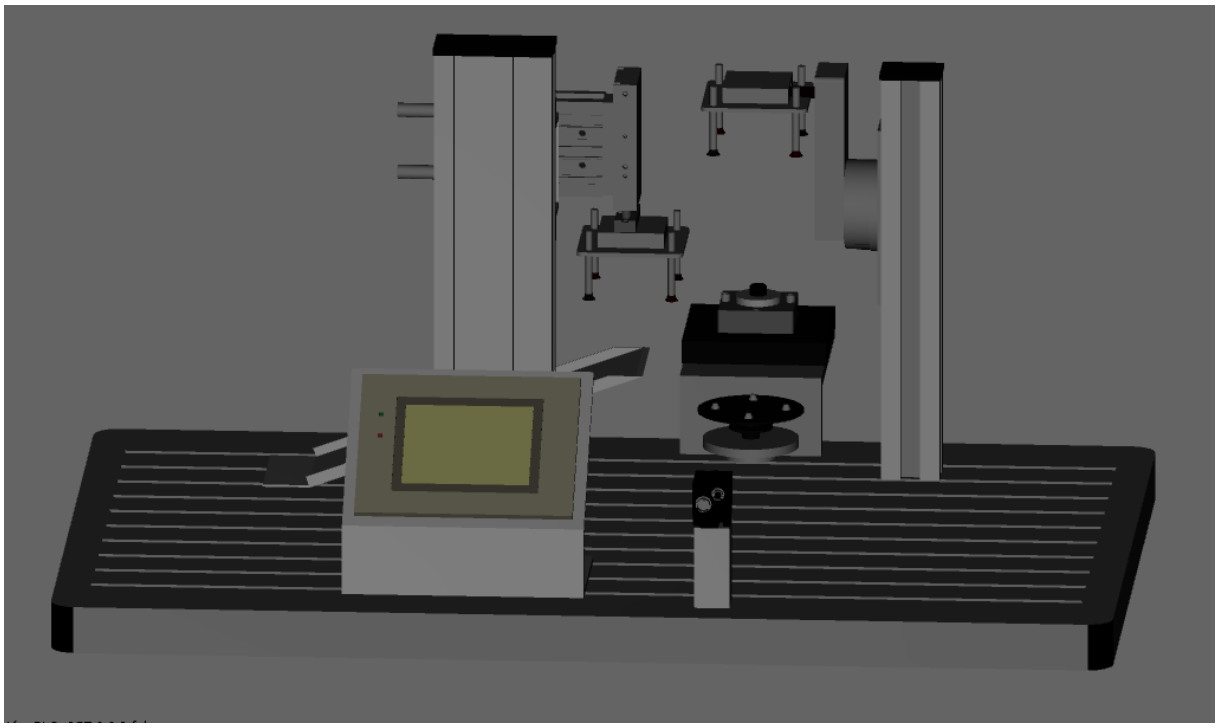


Figura 56. Desvanecimiento de luz lineal, escena

Lo que genera un notable aumento en la generación de sombras en la escena. Cuanto más grande sea el valor mayor será la tasa de aumento de generación de sombras respecto a la luz.

4.2.5 Focos

Los focos son una fuente de luz que se utiliza para iluminar una escena virtual, generalmente pueden ser utilizados para destacar elementos o iluminar desde una posición y con un ángulo concreto.

En Processing la función spotlight(), (Processing, s.f.) permite crear un foco de luz puntual. Este foco emite una luz desde un punto específico en el espacio.

La función spotlight consta de 11 parámetros para determinar las características del foco:

Los parámetros (1, 2 y 3) se refieren a los valores RGB de la luz emitida.

Los parámetros (4, 5 y 6) se refieren a los valores de la posición del foco en el espacio tridimensional. Esto define el punto de origen de la luz.

Los parámetros (7, 8 y 9) se refieren a la dirección del foco. Estos valores definen la dirección del eje central del cono de luz que emite el foco.

El parámetro 10 se refiere al ángulo de apertura del cono de luz que emite el foco, en grados. Este parámetro define la amplitud del cono de luz que emite el foco. Si el ángulo es pequeño, la luz será más concentrada en un área específica, mientras que, si es grande, la luz se extenderá más y abarcará un área más amplia.

El parámetro 11 se refiere a la concentración de la luz en el centro del cono. Este parámetro define el contraste entre el centro del cono de luz y su borde. Un valor alto de concentración hace que la luz esté más concentrada en el centro del foco, produciendo una iluminación más intensa en esa área y una disminución gradual de la intensidad hacia los bordes del foco. Por otro lado, un valor bajo de concentración produce una luz más uniforme en todo el foco.

Se puede ver cómo funcionan este tipo de focos en Processing observando la Figura 57.

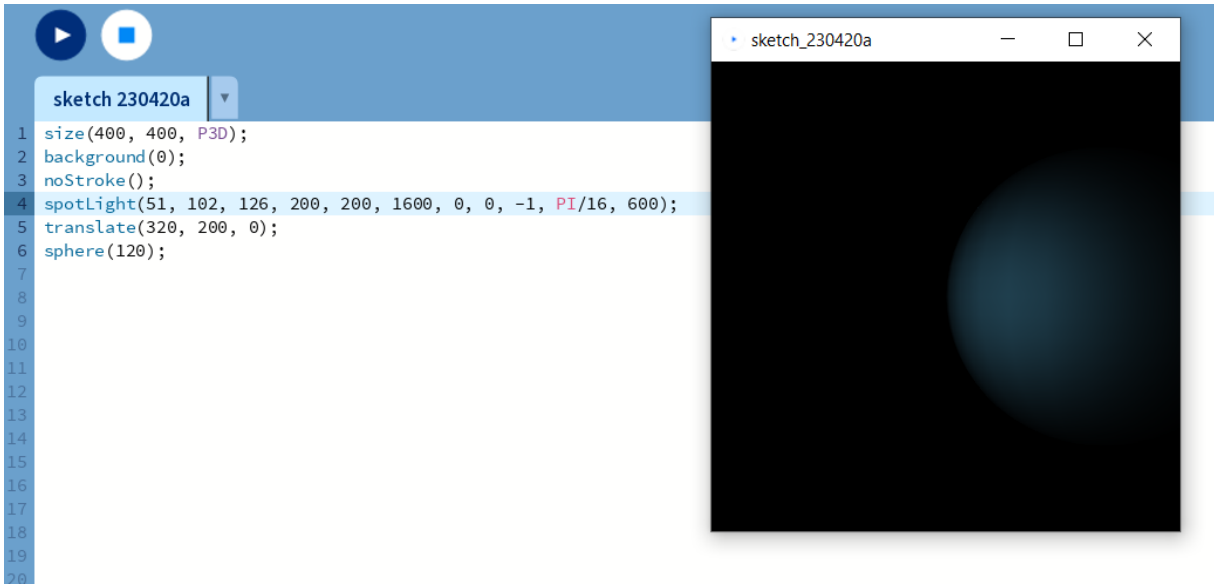


Figura 57. Ejemplo Procesing, foco de concentración alta

Analizando de manera simplificada el ejemplo, Size crea una escena de 400x400 píxeles, background(0), inicializa el color de fondo a negro y noStroke hace que las figuras siguientes no tenga borde.

Analizando la función que crea el foco spotLight(51, 102, 126, 200, 200, 1600, 0, 0, -1, PI/16, 600):

spotLight(r, g, b, nx,ny,nz,z,y,z, angle, concentration)

r,g y b crea una luz azul grisácea con los parámetros RGB.

Nx, ny y nz son las coordenadas 3D de la posición del punto de luz. Se utilizan los valores 200, 200, 1600, que corresponden a una posición en el espacio cerca del borde inferior de la pantalla.

X, y, z es la dirección del punto de luz, especificada como un vector normalizado. Se utiliza el vector (-1, 0, 0), que apunta hacia la izquierda.

Angle: es el ángulo del cono de luz, especificado en radianes. Se utiliza el valor PI/16, que corresponde a un cono de luz estrecho.

Concentration es la concentración de la luz en el centro del cono.

Para ver cómo afecta el ángulo y la concentración al foco se muestra los siguientes ejemplos:

Si se disminuye la concentración de 600 a 100 se pude ver en la Figura 58 cómo se reduce la intensidad del haz de luz del foco y se dispersa sobre toda la superficie de la esfera.

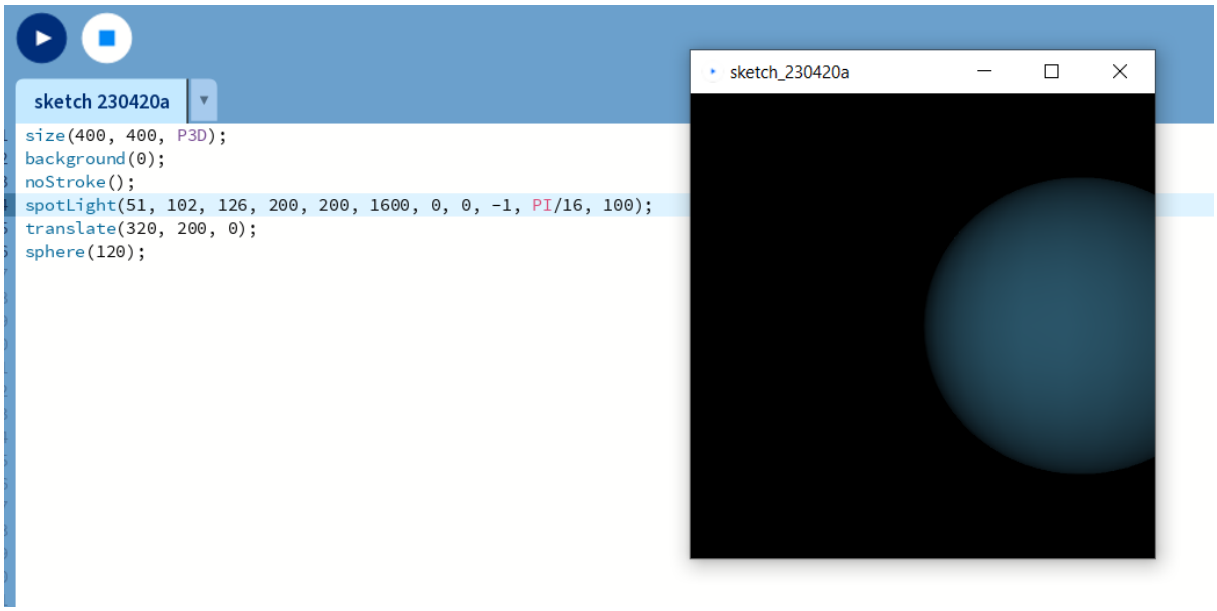


Figura 58. Ejemplo Processing, foco de contentración baja

Con un ángulo menor se puede ver en la Figura 59 como la luz se concentra en un área específica:

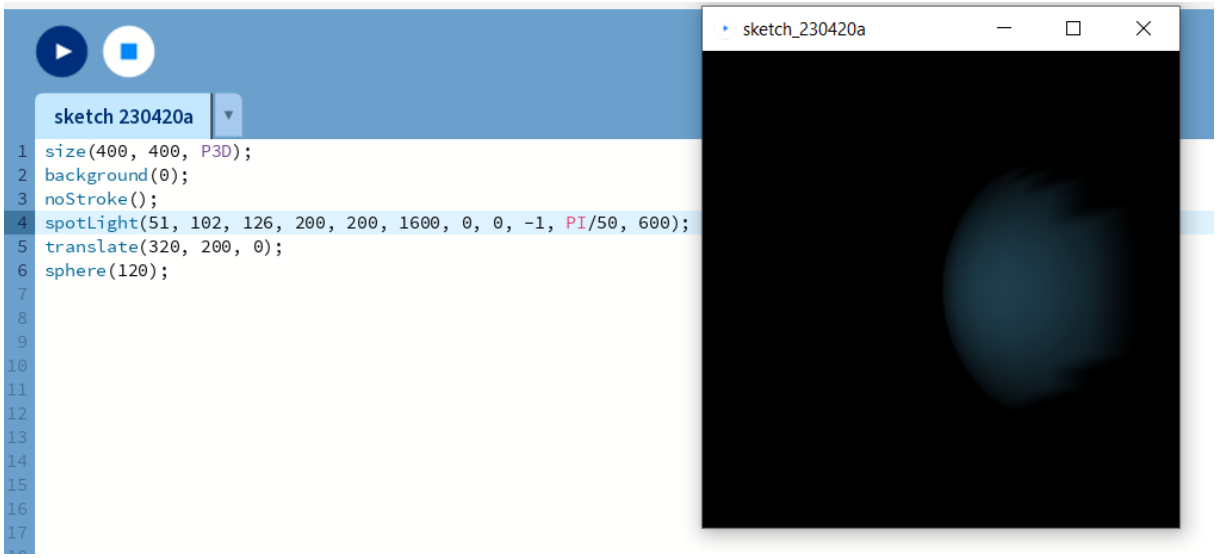


Figura 59. Ejemplo Processing, foco de ángulo de apertura bajo

Una vez visto cómo funciona en Processing la función spotLight se presenta la forma de implementar los focos en la plataforma de gemelos digitales. Se ha creado la clase Foco (Figura 60), esta clase es la encargada de crear los focos que se quieran añadir. Esta clase recibe 3 objetos para crear el foco, uno de tipo LightColor, otro de tipo LightPosition y otro de tipo LightDirection.

```
public class Foco {
    LightColor color;
    LightDirection direccion;
    LightPosition posicion;

    public Foco() {
        color = new LightColor();
        direccion = new LightDirection();
        posicion = new LightPosition();
    }

    public Foco(LightColor color, LightDirection posicion, LightPosition direccion) {
        this.color = color;
        this.direccion = posicion;
        this.posicion = direccion;
    }
```

Figura 60. Clase Foco

La cantidad de focos y sus características son definidos en el fichero XML (y procesados en la clase escena. Se define primeramente un foco situado en el punto (1500,500,600) (ver Figura 61). Este es un punto situado un poco a la derecha y cerca de la estación como se ve en la Figura 62.

```
<foco>
    <red>240</red>
    <green>240</green>
    <blue>240</blue>
    <x>1500</x>
    <y>500</y>
    <z>600</z>
    <nx>0</nx>
    <ny>0</ny>
    <nz>0</nz>
    <angle>0</angle>
    <concentration>0</concentration>
</foco>
```

Figura 61. Foco de luz blanca, situado en el punto (1500,500,600) XML

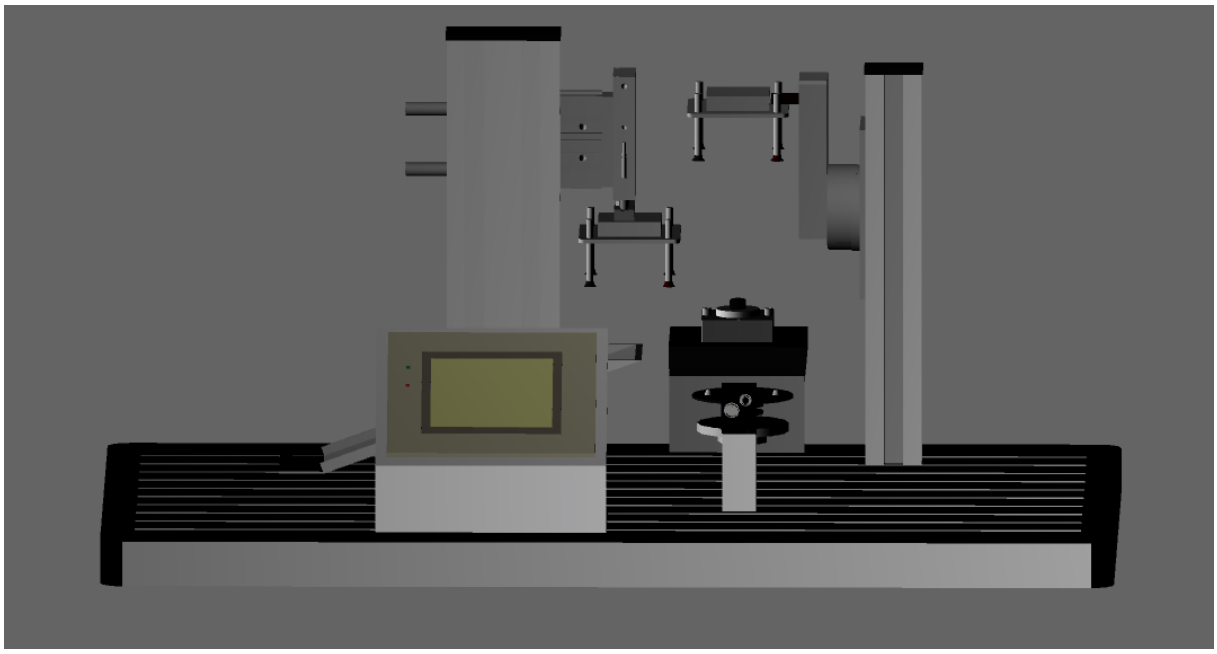


Figura 62. Foco de luz blanca, situado en el punto (1500,500,600) de la escena

Para ver cómo cambia de posición el punto de luz se cambia la coordenada en x a 200 (Figura 63), de este modo se observa como el punto de luz pasa a estar más centrado en la escena en la Figura 64.

```
<foco>
        <red>240</red>
        <green>240</green>
        <blue>240</blue>
        <x>200</x>
        <y>500</y>
        <z>600</z>
        <nx>0</nx>
        <ny>0</ny>
        <nz>0</nz>
        <angle>0</angle>
        <concentration>0</concentration>
    </foco>
```

Figura 63. Foco de luz blanca, situado en el punto (200,500,600) XML

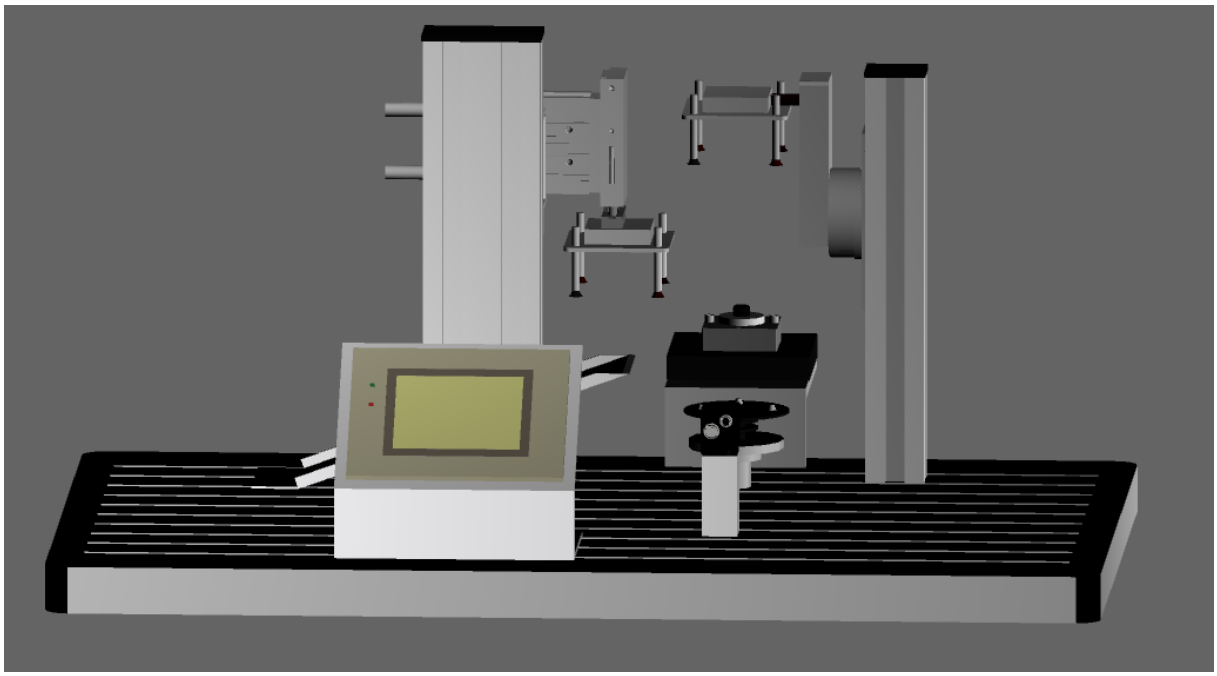


Figura 64. Foco de luz blanca, situado en el punto (200,500,600) de la escena

Si se cambia en este caso la coordenada “y” (Figura 65), se puede observar como el foco se sitúa por encima de la estación, iluminándola desde arriba (ver Figura 66).

```
<foco>
        <red>240</red>
        <green>240</green>
        <blue>240</blue>
        <x>200</x>
        <y>-500</y>
        <z>600</z>
        <nx>0</nx>
        <ny>0</ny>
        <nz>0</nz>
        <angle>0</angle>
        <concentration>0</concentration>
    </foco>
```

Figura 65. Foco de luz blanca, situado en el punto (200,-500,600) XML

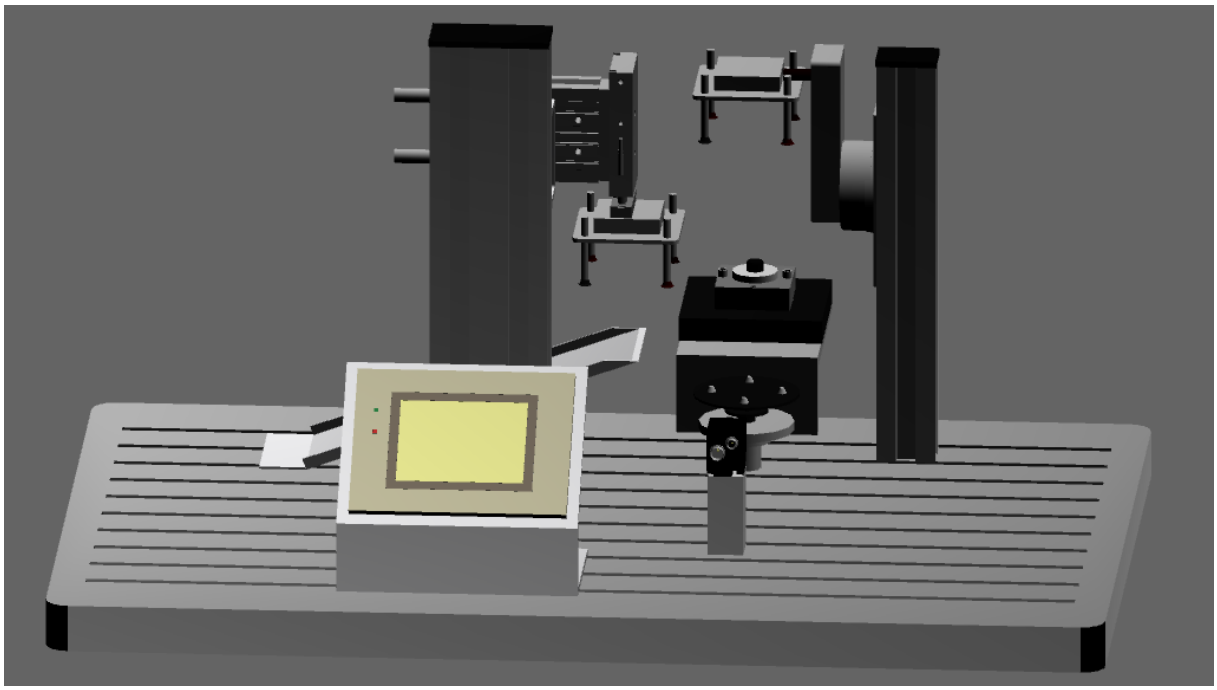


Figura 66. Foco de luz blanca, situado en el punto (200,-500,600) en la escena

4.4 Clase Iluminación

La clase iluminación (Figura 67) es la encargada de unificar todas las clases de iluminación anteriores en un único objeto de tipo Iluminación, este objeto es una iluminación completa de una escena, compuesto de la iluminación ambiental, iluminación especular, desvanecimiento de la luz, luces direccionales y focos.

```
public class Iluminacion {

    Ambiental ambiental;
    LuzDireccional direccional;
    LuzEspecular especular;
    Foco foco;
    LightFalloff caidaluz;
    ArrayList<LuzDireccional> direccionales;
    ArrayList<Foco> focos;

    Iluminacion() {
        ambiental = new Ambiental();
        direccionales = new ArrayList();
        especular = new LuzEspecular();
        caidaluz = new LightFalloff();
        focos = new ArrayList();
    }
```

Figura 67. Clase Iluminación

```
public void setAmbiental(Ambiental ambiental) {
    this.ambiental = ambiental;
}

public void setDireccionales(ArrayList<LuzDireccional> direccionales) {
    this.direccionales = direccionales;
}

public void addLuzDireccional(LuzDireccional direccional){
    direccionales.add(direccional);
}

public void setEspecular(LuzEspecular especular) {
    this.especular = especular;
}


public void setFocos(ArrayList<Foco> focos) {
    this.focos = focos;
}

public void addFoco(Foco foco){
    focos.add(foco);
}

public void setLightFalloff(LightFalloff caidaluz) {
    this.caidaluz = caidaluz;
}
```

Figura 68. Métodos de la clase Iluminación

Mediante los métodos set de cada iluminación, se inicializan las iluminaciones implementadas. (Ver Figura 68).

Mediante los métodos Add (ver Figura 69) se añaden iluminaciones al ArrayList de iluminaciones creadas, este método ha sido creado para las luces direccionales y los focos, que pueden se añadidos cuantos se necesiten para la iluminación de la escena.

```
public void procesaIluminacion(PApplet root) {
    root.lightFalloff(caidaluz.getConstante(),caidaluz.getLineal(),caidaluz.getCuadratico());

    root.lightSpecular(especular.getRed(),especular.getGreen(),especular.getBlue());

    root.ambientLight(ambiental.getRed(), ambiental.getGreen(), ambiental.getBlue());
    for(LuzDireccional ld : direccionales){
    root.directionalLight(ld.getRed(),ld.getGreen(),ld.getBlue(),ld.getNx(),ld.getNy(),ld.getNz());
    }

    for(Foco fc : focos){
    root.spotLight(fc.getRed(),fc.getGreen(),fc.getBlue(),fc.getX(),fc.getY(),fc.getZ(),fc.getNx(),
    fc.getNy(),fc.getNz(),fc.getAngle(),fc.getConcentration());
    }
```

Figura 69. Procesamiento de la iluminación en la clase Escena

El método procesaIluminación configura todos los tipos de iluminación. El parámetro root es de la clase Papplet que sirve para procesar los métodos de Processing. Es importante que la caida de luz y la luz especular vayan en primer lugar para que afecten a las iluminaciones generadas después.

Una vez creada la iluminación general esta se añade al draw de la clase principal GemeloDigital, de este modo se genera la iluminación deseada en la escena.

Un ejemplo de iluminación completa de una escena se observa en la Figura 70 sus valores en el fichero XML y la iluminación que resulta en la escena en la Figura 71.

```
<iluminacion>
  <ambiental>
        <red>128</red>
        <green>128</green>
        <blue>128</blue>
    </ambiental>
    <luzDireccional>
        <red>128</red>
        <green>128</green>
        <blue>128</blue>
        <nx>0</nx>
        <ny>0</ny>
        <nz>-1</nz>
    </luzDireccional>
  <luzEspecular>
        <red>50</red>
        <green>50</green>
        <blue>50</blue>
    </luzEspecular>
     <lightFalloff>
        <constante>1</constante>
        <lineal>0</lineal>
        <cuadratico>0</cuadratico>
    </lightFalloff>
</iluminacion>
```

Figura 70. Ejemplo de iluminación completa, XML

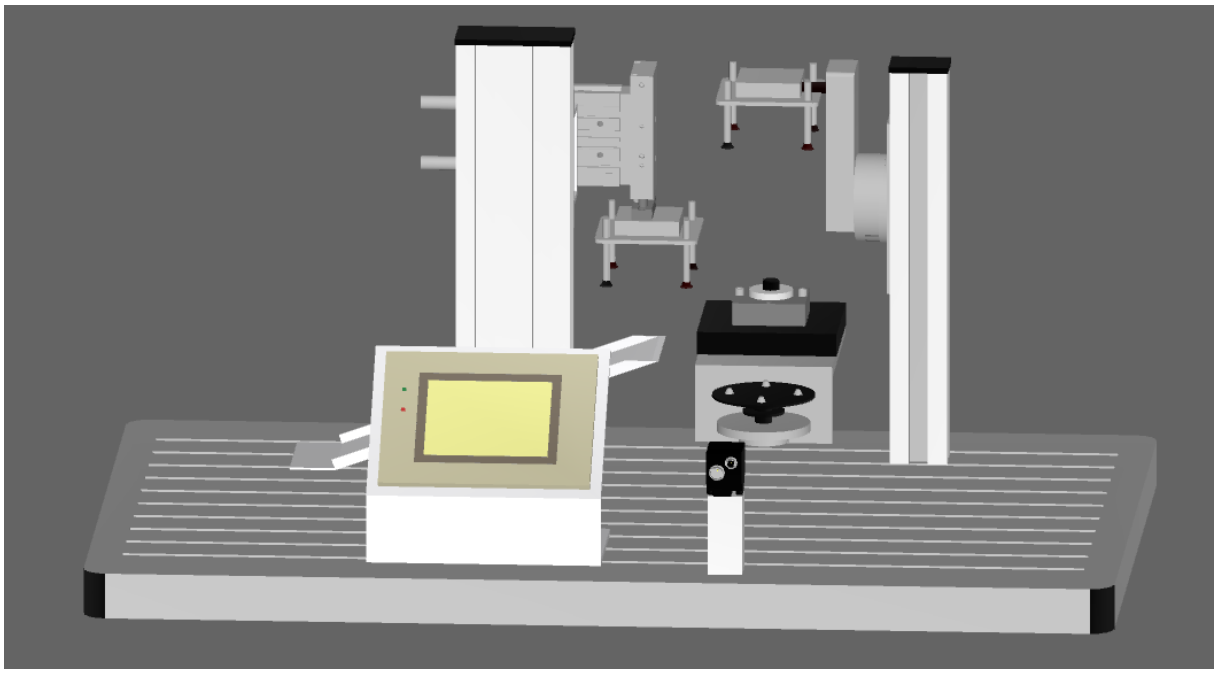


Figura 71. Ejemplo de iluminación completa en la escena

En este ejemplo se ha añadido:

Una luz ambiental con una tonalidad de luz blanca intermedia de valores RGB iguales de 128 cada uno

Una luz direccional de valores RGB 128 para proporcionar una luz blanca de media intensidad, esta luz proviene de la dirección (0,0,-1), lo que es una luz frontal.

La generación de sombras se ha optado por una generación de sombras constante con el parámetro 1

La luz especular acorde con las luces añadidas anteriormente tiene un valor de 128 en valore RGB que proporciona una reflexión de luz adecuada.

En este ejemplo no se ha estimado oportuno añadir focos.

5 Modelo Digital FMS-210

En este capítulo se detalla información sobre el modelado digital que ha sido necesario para realizar el gemelo digital de la estación FMS-210. Para empezar, se hace una breve introducción a la célula flexible de fabricación FMS-200, para después centrarse en la estación FMS-210, de la cual se detallan sus entradas y salidas, así como todos los modelos 3D que han sido necesarios realizar para componer su gemelo do digital. Por último, ya que la estación FMS-210 es la encargada del control de calidad de las piezas, se presentan los modelados 3D de todos los componentes de la pieza que va a ser examinada y los casos de piezas completas que son añadidos a la plataforma.

5.1 Descripción general de la estación

La célula de fabricación flexible FMS200 de Siemens (Figura 72) (SMC, s.f.) es un sistema de fabricación flexible, diseñado para automatizar y optimizar la producción en la industria. Se compone de varias estaciones de trabajo que trabajan de manera autónoma y están interconectadas entre ellas por un sistema de transporte.

Figura 72. Célula de fabricación flexible FMS-200

La célula flexible de fabricación FMS200 está compuesta por una serie de estaciones individuales, cada una de las cuales lleva a cabo un proceso de inserción de un componente específico. Como resultado, se obtiene un mecanismo de giro completamente ensamblado.

Las estaciones que componen FMS200, son utilizadas en la industria, de forma que se puede trabajar con elementos reales para proporcionar un aprendizaje de calidad.

El cuadro de control es totalmente modular y cada estación puede ser desmontada de manera fácil y rápida. Cada una de las estaciones llevan a cabo una tarea específica las cuales son:

- FMS-201: Alimentación de la base
- FMS-202: Inserción del rodamiento
- FMS-203: Prensado hidráulico del rodamiento:
- FMS-204: Inserción del eje
- FMS-205: Inserción de la tapa
- FMS-206: Inserción de tornillos
- FMS-207: Ensamblaje y atornillado robotizado
- FMS-208: Almacén automático
- FMS-209: Secado de pintura en horno
- **FMS-210: Control de calidad por visión artificial**
- Sistema de Transporte de la pieza.

La inclusión de esta estación FMS-210 (ver Figura 73) en la familia FMS-200 implica la integración de la tecnología de visión artificial, que se utiliza ampliamente en los procesos productivos automatizados, para llevar a cabo el control de calidad. El producto en proceso que llega desde la estación anterior se transporta a la posición de inspección, donde una cámara de visión artificial evalúa una serie de variables en dos posiciones. Los resultados de las variables evaluadas se utilizan para realizar el control de calidad del producto en curso.

Figura 73. Estación FMS-210 Control de calidad por visión artificial

Entradas y salidas de la estación

En este punto se muestran las tablas de entradas (Tabla 1. Entradas de la estación FMS-210, sensores y Tabla 2. Entradas de la estación FMS-210, sistema de visión) y la tabla de salidas, Tabla 3. Salidas de la estación FMS-210 las cuales contienen todas las entradas y salidas de las que dispone la estación.

Entrada (sensores)	
Dirección	Función
I2.0	Marcha (start)
I2.1	Paro (stop)
I2.2	Auto/man
I2.3	Rearme (reset)
I2.4	Actuador de giro atrás (a0)
I2.5	Actuador de giro en medio (a1)
I2.6	Actuador de giro adelante (a2)
I2.7	Vacío de ventosas, actuador de giro (v1)
I3.0	Cilindro enclavador plato adelante (b1)
I3.1	Man. Evacuación atrás (c0)
I3.2	Man. Evacuación adelante (c1)
I3.3	Man. Evacuación arriba (d0)
I3.4	Man. Evacuación abajo (d1)
I3.5	Vacío ventosas evacuación (v2)
I3.6	Cámara RUN (run)
I3.7	Cámara error

Tabla 1. Entradas de la estación FMS-210, sensores

Entradas (sistema de visión)	
I0.0	Cámara BUSY (busy)
I0.1	Cámara GATE
I0.2	Cámara salidas OR (or)
I0.3	Resultado medida salida 0 (d0)
I0.4	Resultado medida salida 1 (d2)
I0.5	Resultado medida salida 2 (d2)
I0.6	Resultado medida salida 3 (d3)
I0.7	Resultado medida salida 4 (d4)

I1.0	Resultado medida salida 5 (d5)
I1.1	Resultado medida salida 6 (d6)
I1.2	Resultado medida salida 7 (d7)
I1.3	Resultado medida salida 8 (d8)
I1.4	Resultado medida salida 9 (d9)
I1.5	Resultado medida salida 10 (d10)
I1.6	Resultado medida salida 11 (d11)
I1.7	Resultado medida salida 12 (d12)
I4.0	Resultado medida salida 13 (d13)
I4.1	Resultado medida salida 14 (d14)
I4.2	Resultado medida salida 15 (d15)

Tabla 2. Entradas de la estación FMS-210, sistema de visión

Salidas	
Q2.0	Defecto (ALARM)
Q2.1	Avanzar actuador de giro (A+)
Q2.2	Pulsos del motor (PULSES)
Q2.3	Retroceder actuador de giro (A-)
Q2.4	Comenzar vacío (V1+)
Q2.5	Parar vacío (V1-)
Q2.6	Avanzar cilindro enclav. plato (B+)
Q2.7	Avanzar man evacuación (C+)
Q3.0	Retroceder man evacuación (C-)
Q3.1	Bajar man. Evacuación (D+)
Q3.2	Comenzar vacío manipulador (V2+)
Q3.3	Parar vacío manipulador (V2-)
Q3.4	Cámara RESET (reset)
Q3.5	Cámara petición de datos (DSA)
Q3.6	Cámara step (STEP)
Q3.7	Entrada 0 de datos a cámara (DIO)

Tabla 3. Salidas de la estación FMS-210

5.2 Modelos 3D de la estación

En este punto se considera oportuno la descripción del modelado 3D de los elementos que componen la estación FMS-210. Más adelante se detallará como se deben de configurar para componer el gemelo digital en la aplicación.

La estación FMS-210 ha sido desarrollada gráficamente mediante el software Autodesk Inventor 2022, a continuación, se presentan varios conjuntos los cuales componen la misma.

Base fija

Inicialmente se ha desarrollado un modelado 3D de todos los elementos que permanecen fijos en la estación, omitiendo los elementos móviles (cilindros) y otros casos excepcionales (cámara, piezas y botones), como se observa en la Figura 64.

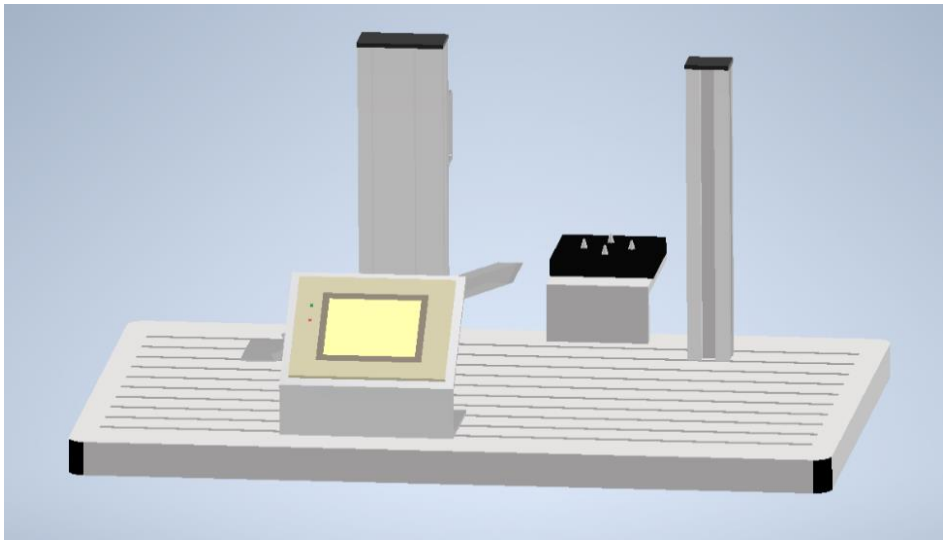


Figura 74. Base fija de la estación FMS-210

La Figura 64 muestra el ensamblaje fijo, compuesto de la mesa de trabajo, la pantalla de control de visión por computador, el soporte inicial de las piezas, la rampa para desechar piezas y las columnas que sirven como soporte para los cilindros. El motivo de este ensamblaje es debido a la manera en la que se estructura la programación del gemelo, en la cual resulta más sencillo si se añaden todos los objetos inmóviles como un solo sólido.

Conjunto A

El conjunto A (ver Figura 78) está compuesto por un cilindro MSQB50A (ver Figura 75), un brazo giratorio (ver Figura 76) y un sistema de agarre mediante un soporte de ventosas (ver Figura 77).

El cilindro MSQB50A (SMC MSQB50A, s.f.) es un actuador de tipo neumático que convierte la energía del aire comprimido en movimiento giratorio. Está diseñado con un diámetro de 50 mm y una carrera máxima de 1000 mm, lo que permite una amplia variedad de aplicaciones en la automatización de procesos industriales. Además, el cilindro MSQB50A cuenta con una carcasa de aleación de aluminio, que lo hace resistente a la corrosión y a las condiciones ambientales adversas.

El cilindro MSQB50A es ampliamente utilizado en la industria para realizar tareas como el movimiento de piezas y herramientas, la sujeción de objetos, el posicionamiento de elementos, entre otras. Se caracteriza por su alta eficiencia y durabilidad, lo que lo convierte en una solución fiable y rentable para la automatización de procesos industriales.

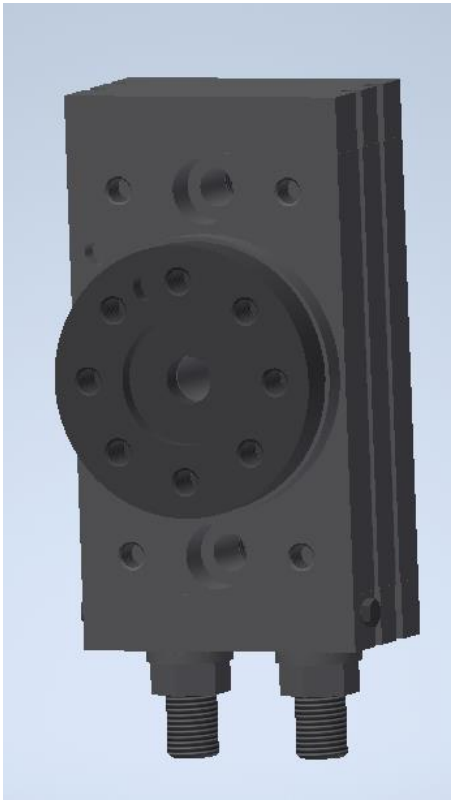


Figura 75. Cilindro MSQB50A

Asociado al cilindro se ensambla un brazo giratorio para que actúe como vástago del mismo y tenga alcance para desplazar las piezas de la posición inicial a la posición de control de calidad.

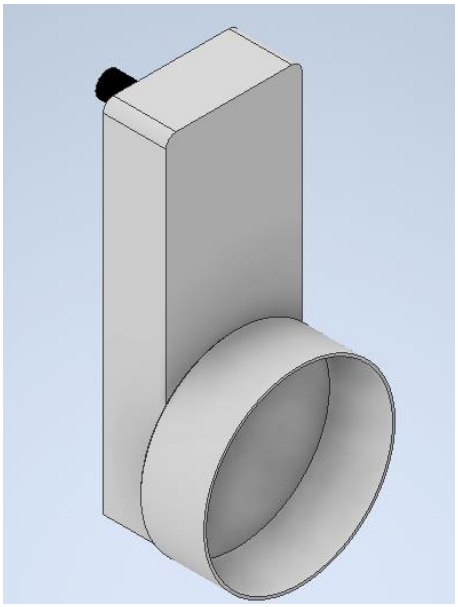


Figura 76. Brazo giratorio

Unido a la parte final del brazo giratorio se encuentra un sistema de ventosas que, a través de la acción del vacío neumático, tienen la capacidad de conseguir que se adhiera la pieza a las ventosas para ser desplazada.

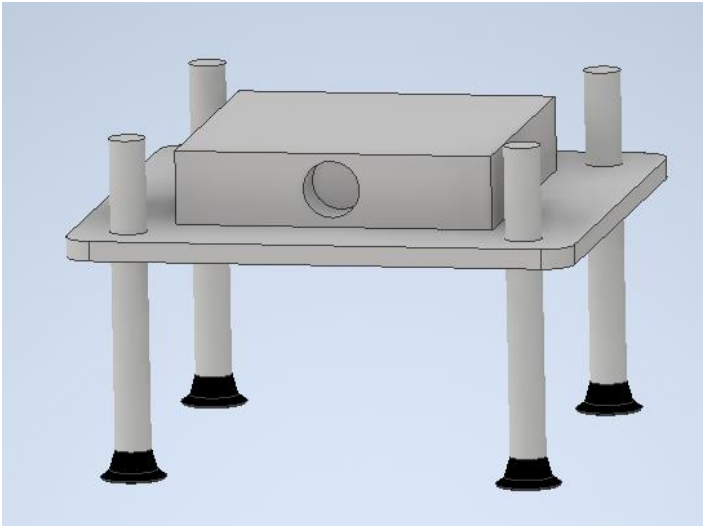


Figura 77. Sistema de vacío

La unión de las 3 piezas anteriores formaría en conjunto A o cilindro A, que quedaría de la siguiente manera, todo junto ensamblado.

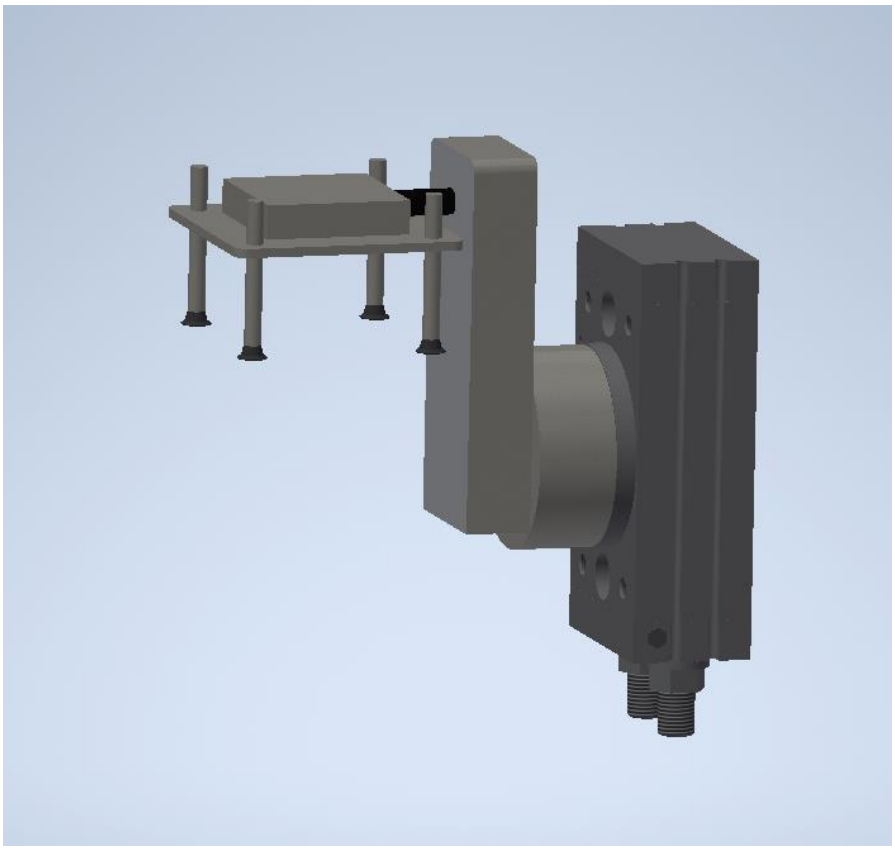


Figura 78. Conjunto A

Conjunto C y conjunto D

El conjunto C y el conjunto D (conjunto C y D ensamblados Figura 84) son los encargados de retirar la pieza tras la revisión de las misma, están compuestos de dos cilindros encargados uno del movimiento horizontal (C) y otro del movimiento vertical (D).

El conjunto C está compuesto del cilindro MGPM20-100 (ver Figura 79), (SMC MSQB50A, s.f.) fabricado por SMC Corporation. Este es un cilindro neumático compacto y altamente funcional. Con un diámetro de 20 mm y una carrera de 100 mm, ofrece un diseño compacto que permite su fácil integración en aplicaciones donde el espacio es limitado. Su acción simple lo hace eficiente y confiable, utilizando la presión neumática para extender el pistón en un sentido, mientras que una fuerza externa, como un resorte o carga, se encarga de retraerlo en el otro sentido.

Además de sus dimensiones y acción simple, el cilindro MGPM20-100 destaca por su versatilidad en cuanto a la configuración de montaje. Esto significa que se puede adaptar fácilmente a diferentes requisitos de instalación y es compatible con una variedad de sistemas y aplicaciones industriales.

En Autodesk Inventor se puede visualizar el cilindro en la Figura 79.

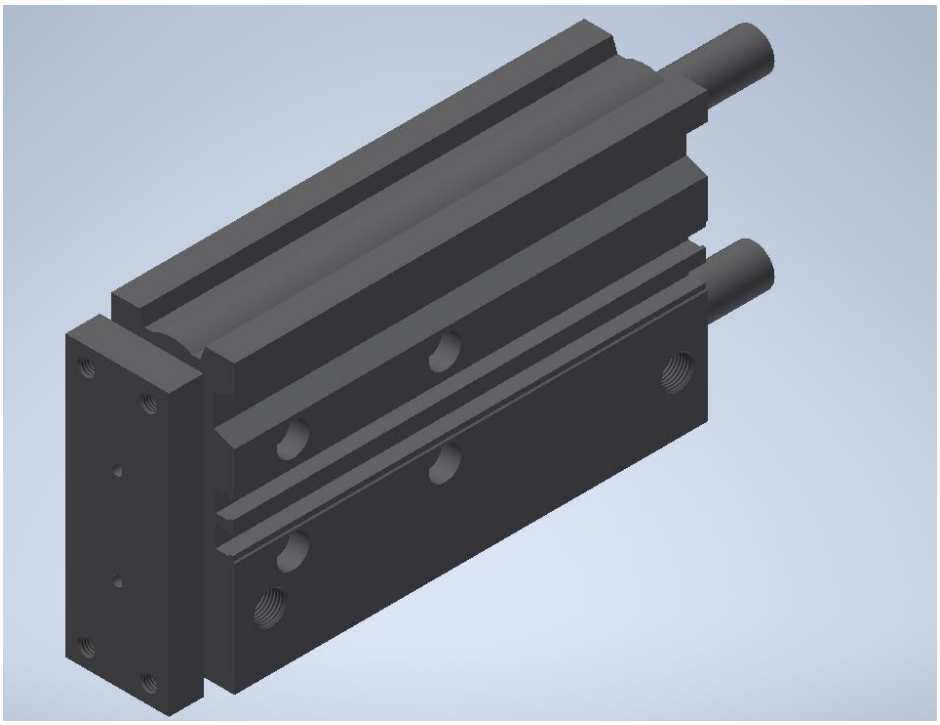


Figura 79. Cilindro MGPM20-100

Se pueden observar el cilindro y su pistón de manera diferenciada en la Figura 80.

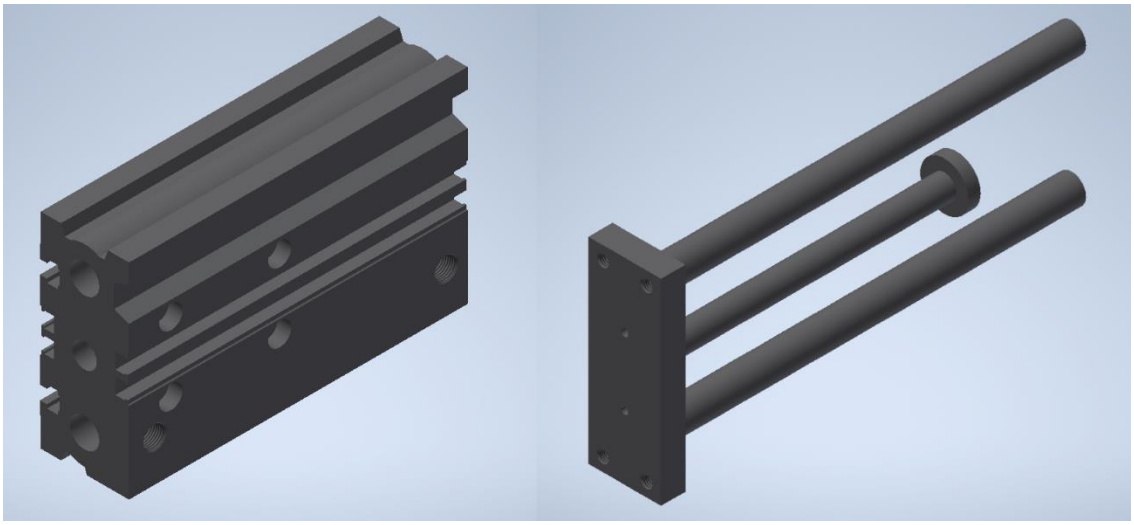


Figura 80. Cilindro y vástago MGPM20-100

El conjunto D se encuentra acoplado al conjunto C, este conjunto se encarga del movimiento vertical.

El conjunto D está compuesto, por el cilindro CXSM15-50 (ver Figura 81). Este cilindro (SMC CXSM15-50, s.f.) es un componente mecánico utilizado en una amplia variedad de aplicaciones industriales y comerciales. Se caracteriza por su diseño compacto y robusto, que le permite soportar altas cargas y resistir condiciones adversas. Su diámetro exterior es de 15 mm, mientras que su longitud alcanza los 50 mm.

El CXSM15-50 está fabricado con materiales de alta calidad, como acero inoxidable, lo que garantiza su durabilidad y resistencia a la corrosión. Este cilindro es accionado mediante un sistema neumático, lo que le proporciona un funcionamiento eficiente y preciso. Además, cuenta con una serie de características de diseño que lo hacen altamente versátil, como su capacidad para montarse en diferentes posiciones y su capacidad de ajuste fino.

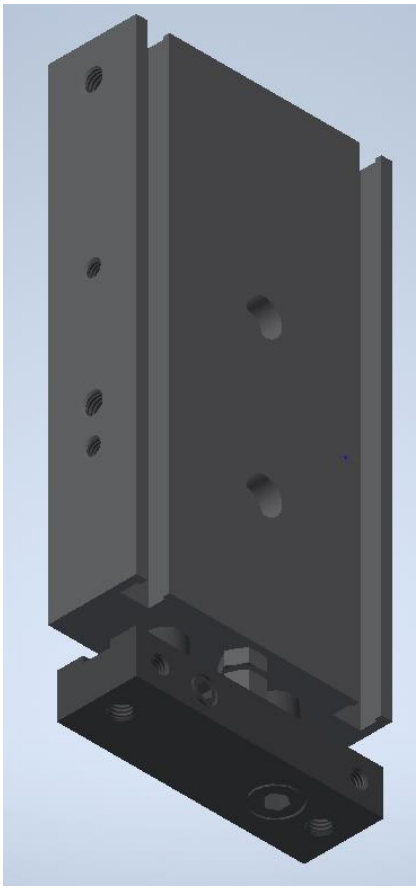


Figura 81. Cilindro CXSM15-50

Se pueden observar de manera separada el cilindro y el pistón en la Figura 82.

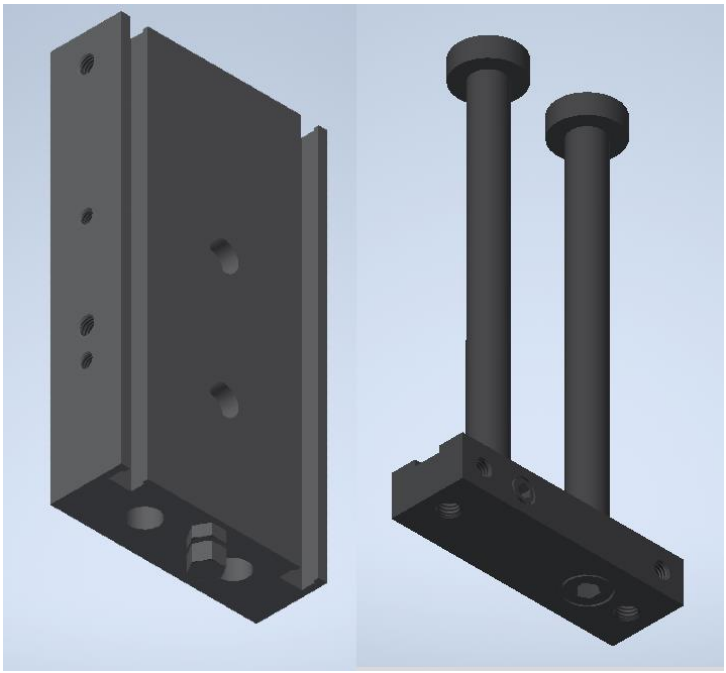


Figura 82. Cilindro y vástago CXSM15-50

Para completar la función de los conjuntos C y D, se les ensambla un sistema de ventosas que tiene la función de imprimir vacío sobre las piezas, para transportarlas de la fase de examen de calidad de la pieza, a la rampa de desecho de piezas. Este sistema de vacío se puede ver en la Figura 83.

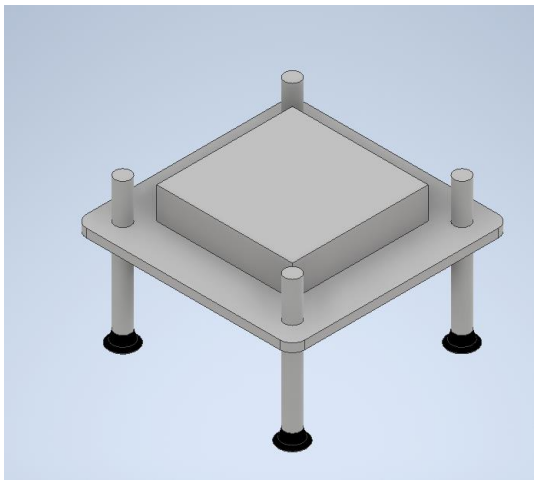


Figura 83. Sistema de vacío

El conjunto C y el conjunto D, ensamblados quedarían de la siguiente forma Figura 84, permitiendo un desplazamiento horizontal, mediante el cilindro de doble efecto MGPM20-100 y la aproximación vertical a la pieza mediante el cilindro de simple efecto CXSM15-50.

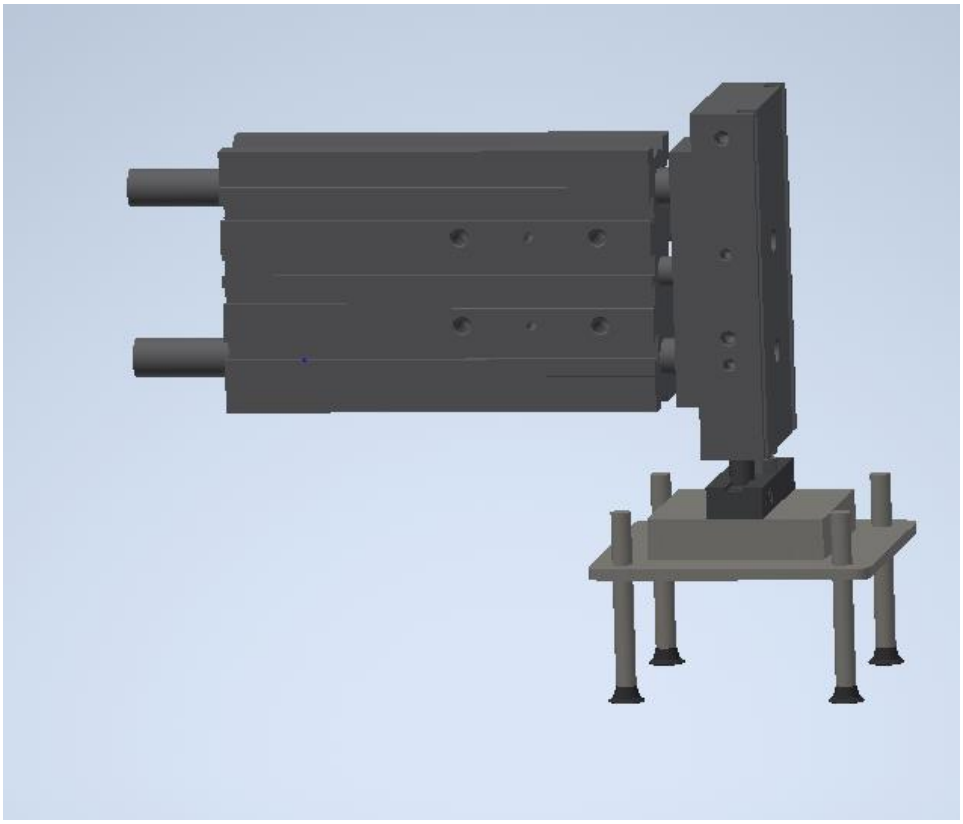


Figura 84. Conjunto C y D

Motor de pulsos (cilindro B)

La generación de motores de pulsos aún no está implementada en la plataforma de gemelos digitales, por lo que en sustitución se ha tratado al motor de pulsos como un cilindro giratorio. Se le ha implantado un movimiento de rotación de 360°. Dado que la rotación de la pieza no está disponible, no se visualizará la animación en este punto del proceso. Se ha añadido una representación 3D similar la cual es observable en la Figura 85.

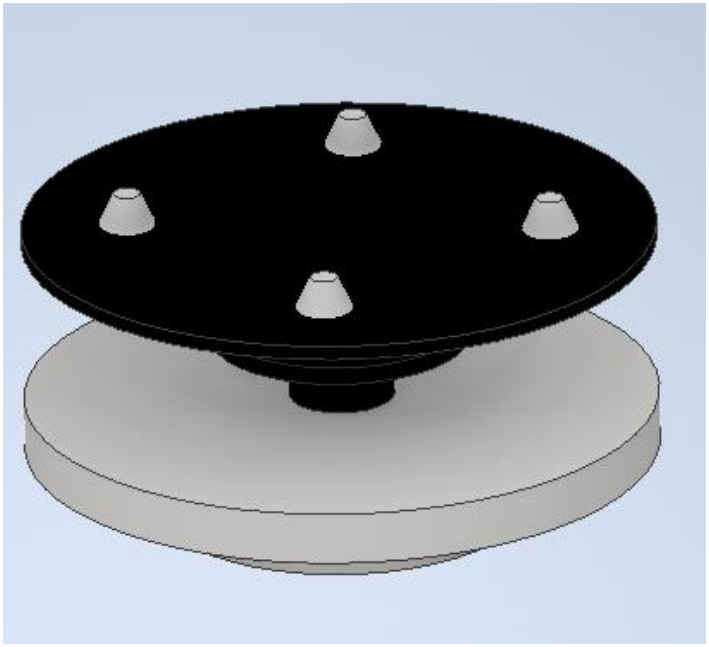


Figura 85. Motor de pulsos

Sensor

Como representación gráfica de los sensores de la estación se ha usado es el D-M9PL.

El sensor D-M9PL (ver Figura 86) (SMC CORPORATION, 2021) es un dispositivo de detección versátil y de alta precisión utilizado en aplicaciones industriales. Diseñado para detectar la presencia o ausencia de objetos, este sensor ofrece un rendimiento confiable y una respuesta rápida en entornos exigentes.

Equipado con tecnología avanzada, el D-M9PL utiliza un principio de detección fotoeléctrico para proporcionar resultados precisos y consistentes. Su diseño compacto y robusto lo hace adecuado para una amplia gama de aplicaciones, incluyendo sistemas de automatización, maquinaria industrial y control de procesos.

Con características como un haz de detección ajustable, detección estable en distancias variables y una alta inmunidad al ruido, el sensor D-M9PL ofrece una solución confiable y eficiente para la detección de objetos. Su facilidad de uso y capacidad de montaje flexible hacen que sea una opción popular en entornos industriales donde se requiere una detección precisa y confiable.

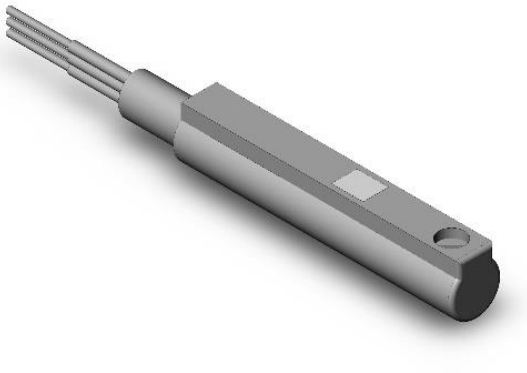


Figura 86. Sensor D-M9PL

Cámara

La cámara F150_S1A (ver Figura 87) (OMRO, 2005) es una cámara digital avanzada. Con un sensor de imagen de alta resolución y un amplio rango dinámico, esta cámara captura imágenes nítidas y detalladas con colores vibrantes y precisos. Su diseño robusto y resistente al agua la hace ideal para su uso en entornos exigentes.

Además de su calidad de imagen excepcional, la F150_S1A también ofrece una amplia gama de características y funciones avanzadas. Con una alta sensibilidad ISO y una velocidad de disparo rápida, es capaz de capturar imágenes en condiciones de poca luz y sujetos en movimiento sin comprometer la calidad. También cuenta con una variedad de modos de disparo, incluyendo disparo en ráfaga y lapso de tiempo, lo que brinda a los usuarios la flexibilidad de capturar diferentes tipos de escenas y momentos. Con su combinación de rendimiento y versatilidad, la F150_S1A es una cámara propicia para ambientes industriales que requieren precisión y calidad.

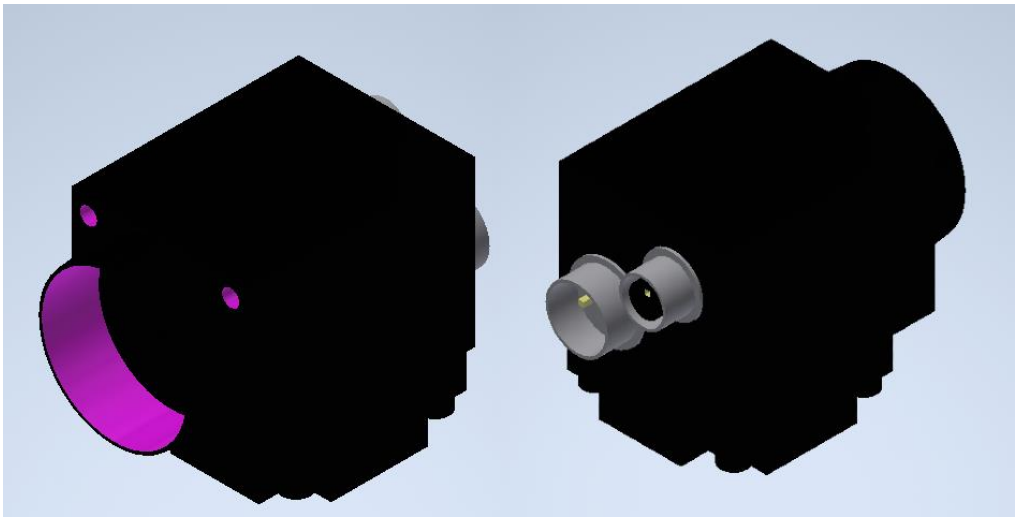


Figura 87. Cámara F150_S1A

Botonera

Se ha añadido una botonera la cual está compuesta de 5 botones con las funciones de Start, Stop, Reset, Alarma y cambio AUTO/MAN, estos botones se sitúan de manera individual en la parte frontal de la estación y se accede a ellos manualmente.

Botón de Start (Figura 88), es el encargado de iniciar el proceso cuando se detecta su activación.

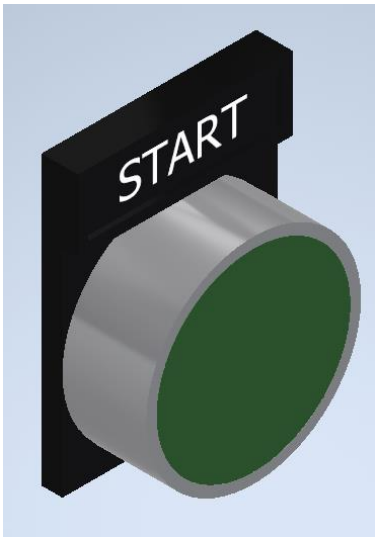


Figura 88. Botón de Start

Botón de Stop (Figura 89), tiene como función parar el funcionamiento del gemelo digital al finalizar la etapa actual en la que se encuentra.

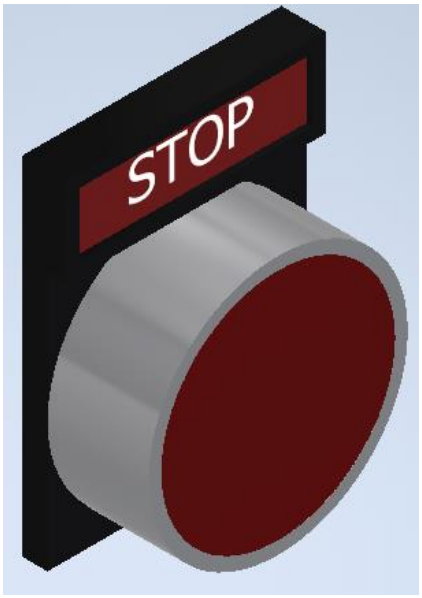


Figura 89. Botón de Stop

Botón de Reset (Figura 90), se utiliza para reiniciar el proceso del gemelo digital, debe de restablecer a sus valores iniciales todos los sensores y los actuadores a su posición inicial.

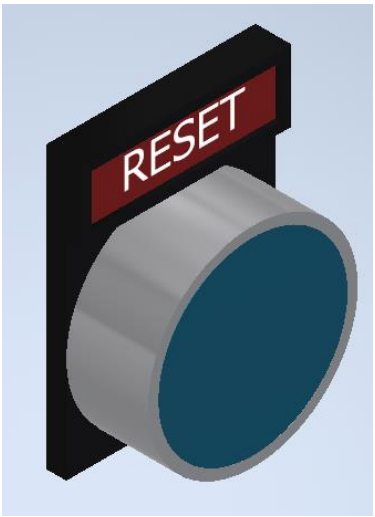


Figura 90. Botón de Reset

Botón de AUTO/MAN (Figura 90), este botón tiene la función de cambiar entre dos modos de trabajo. El modo automático deja que el gemelo digital trabaje de manera autónoma y siga todo el proceso cíclicamente por sí mismo. En el modo manual, se realiza el proceso paso a paso, previa confirmación del usuario para seguir con el mismo.

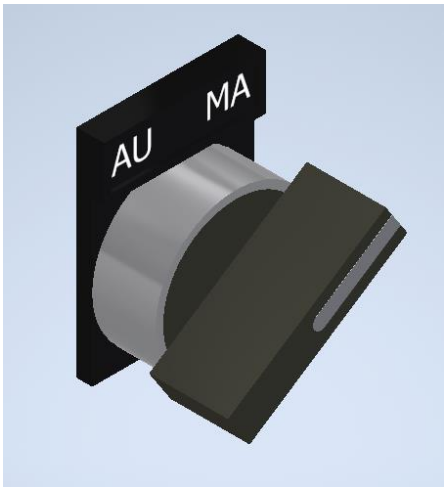


Figura 91. Botón AUTO/MAN

Seta de emergencia (Figura 92), como su nombre indica se utiliza en casos de emergencia, esta seta para de forma inmediata el proceso, desactivando cualquier proceso que haya en curso.

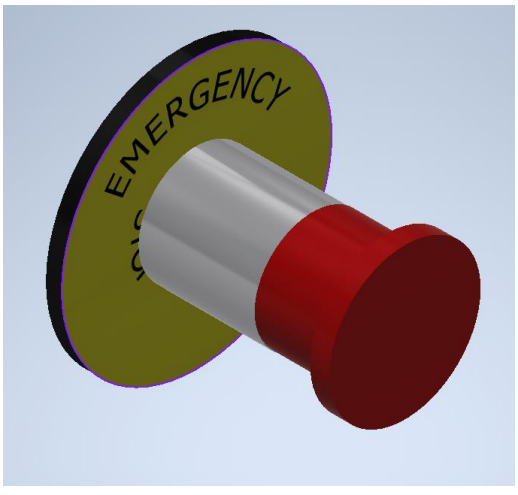


Figura 92. Seta de emergencia

5.3 Modelos 3D de los tipos de pieza

En este apartado se describen todos los elementos que componen la pieza que va a ser inspeccionada en la fase de control de calidad y los casos que han sido propuestos para tal tarea. Se debe aclarar que esta pieza no tiene otro propósito más que el de la enseñanza, no es una pieza que vaya a ser integrada después en un mecanismo.

Para empezar, se describen todos los elementos que forman la pieza:

La base de la pieza (Figura 93), es de forma cuadrada con perforaciones circulares para la inserción de tornillos y un agujero central en el que se sitúa gran parte del mecanismo.

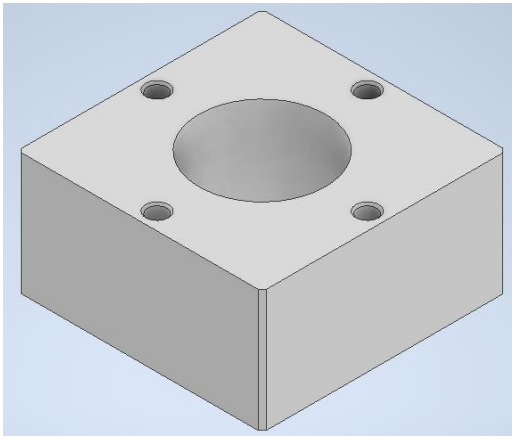


Figura 93. Base de la pieza

El rodamiento (Figura 94), se sitúa en el agujero central de la pieza, para ello la base cuenta con un soporte el cual hace que quede correctamente ajustado a ella.

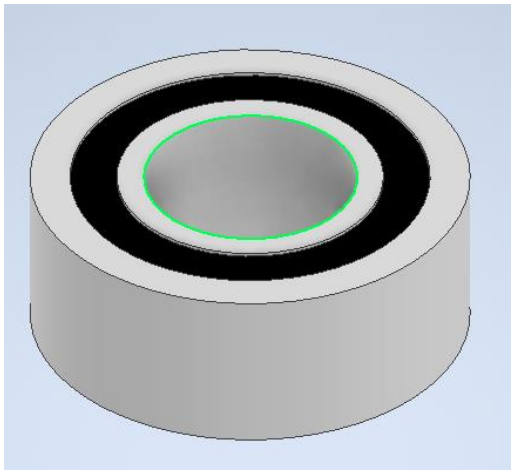


Figura 94. Rodamiento

Seguido al rodamiento se introduce un eje que puede ser de color negro (hecho de nylon) o de color gris (material metálico), este eje encaja en el rodamiento sin sobresalir por la parte inferior de la pieza, pero sí por la parte superior. Ambos tipos de ejes se pueden ver en la Figura 95.

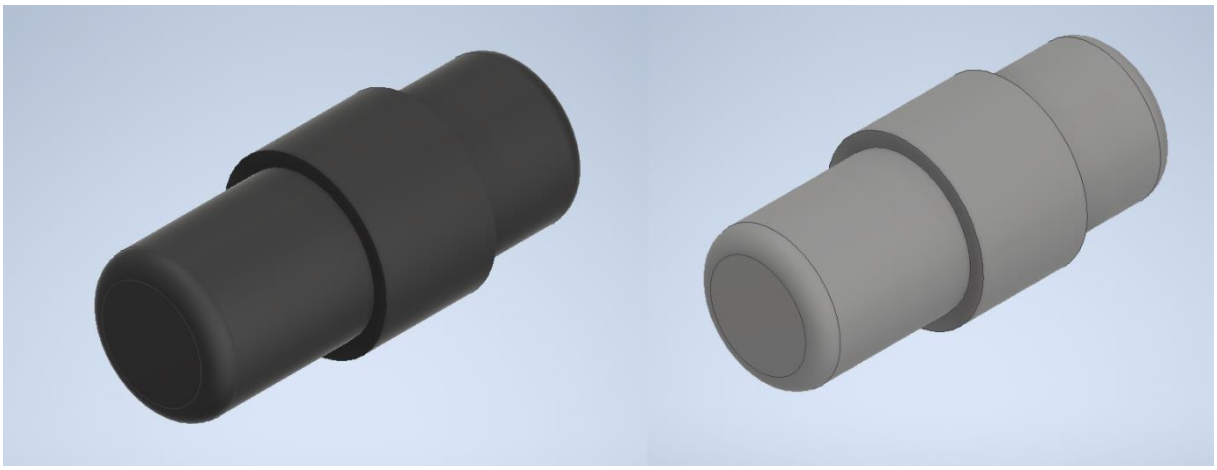


Figura 95. Ejes

A continuación, se introduce una tapa que puede ser gris (metálica), negra (nylon) o blanca, esta tapa se introduce para que quede bien encajado el eje al rodamiento. El modelado 3D de los 3 tipos de tapas se observa en la Figura 96.

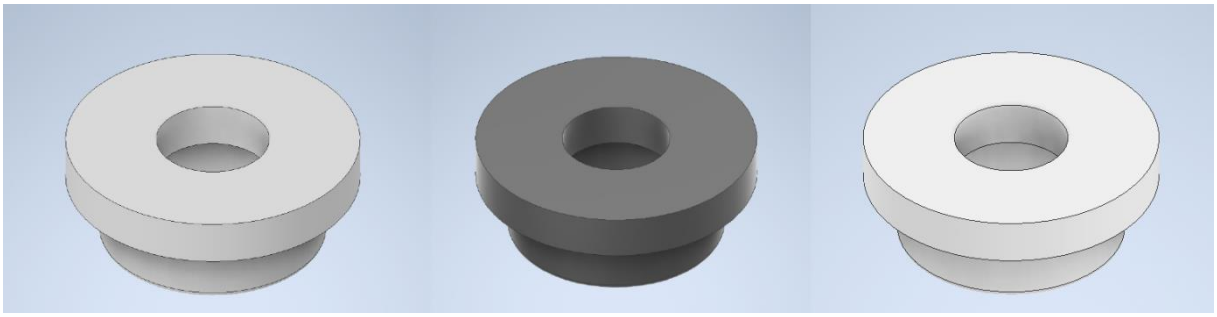


Figura 96. Tapas

Para finalizar la pieza se añaden tornillos a ambos lados de la pieza, en la parte delantera y trasera formando 90° entre sí, simulando un ajuste final de la pieza. El modelado 3D del tornillo se visualiza en la Figura 97.

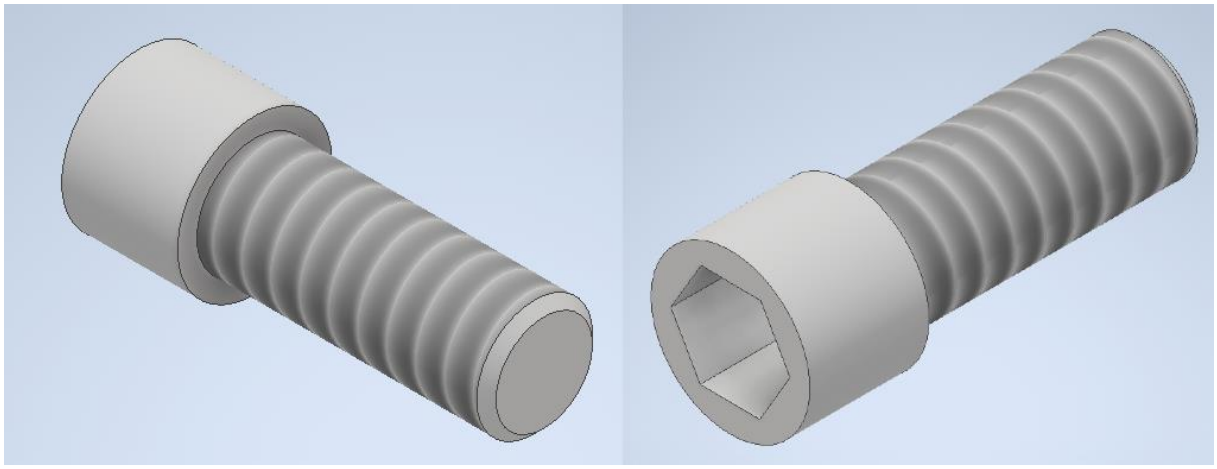


Figura 97. Tornillo

Se puede visualizar una pieza completa, con rodamiento, eje negro, tapa blanca y los cuatro tornillos. La pieza quedaría ensamblada y se observaría en la Figura 98.

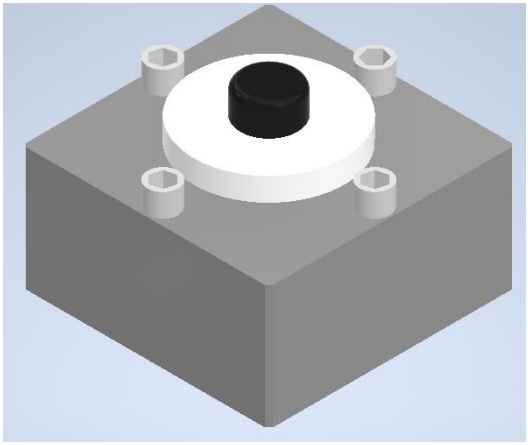


Figura 98. Pieza completa

Casos de piezas:

Para simular la alimentación de piezas de la estación se han creado 5 casos distintos de piezas que aparecerán en la plataforma de forma aleatoria y sucesiva. De este modo se consigue simular el análisis del sistema de control de calidad, el cual según la pieza que se muestre dará validez a ella o no, siguiendo el criterio que se le imponga. Inicialmente se la ha impuesto que la pieza será válida siempre y cuando hay presencia de rodamiento, eje, tapa, tornillo izquierdo y derecho. De este

modo no importa qué tipo de tapa o eje haya para que la pieza sea válida, basta con la presencia de ellos para que la pieza se valore como completa y válida. No se han tenido en cuenta los tornillos de la posición delantera y posterior ya que el motor de pulsos no ha sido implementado, y la visibilidad de la pieza solo se contempla en la posición inicial.

- Caso 1 (Figura 99)

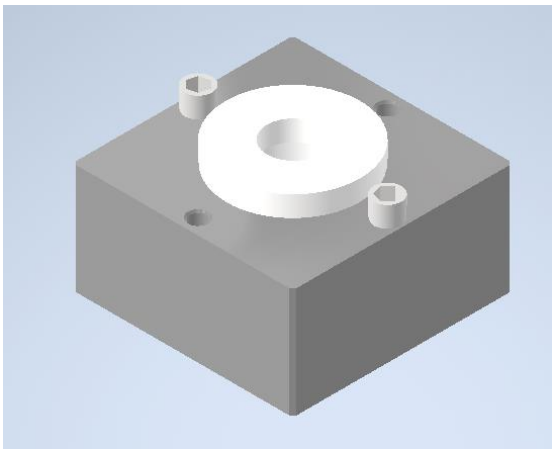


Figura 99. Caso 1 de control de calidad de la pieza

Como elementos en el caso 1, figuran una tapa blanca y los tornillos, esta pieza quedaría descartada por el sistema de revisión de la pieza, debido a la falta del eje.

- Caso 2 (Figura 100)

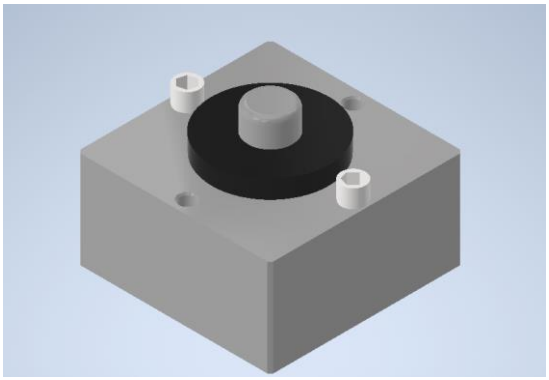


Figura 100. Caso 2 de control de calidad de la pieza

El caso 2 sería marcado como válido por el sistema de visión ya que tiene presentes todos los elementos requeridos para que la pieza sea válida.

- Caso3 (Figura 101)

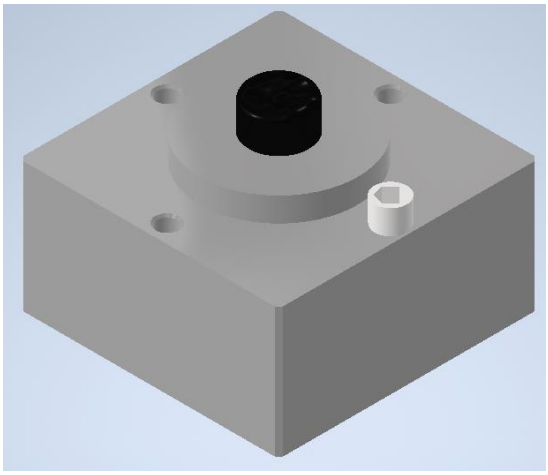


Figura 101. Caso 3 de control de calidad de la pieza

En este caso la pieza no sería válida ya que falta el tornillo izquierdo.

- Caso 4 (Figura 102)

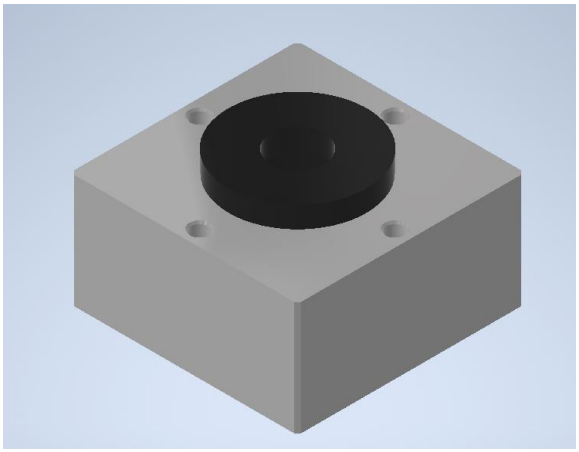


Figura 102. Caso 4 de control de calidad de la pieza

La falta de tornillos y eje hacen que esta pieza sea descartada por el sistema de revisión de calidad.

- Caso 5 (Figura 103)

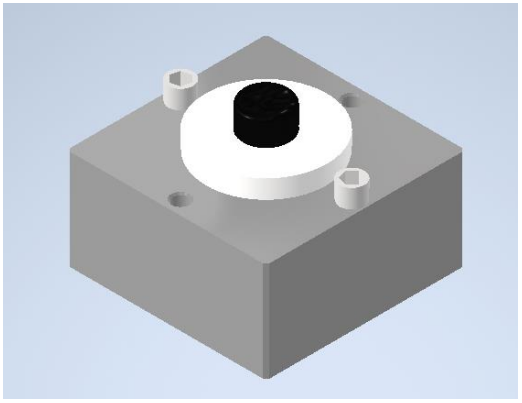


Figura 103. Caso 5 de control de calidad de la pieza

El caso 5 sería procesado como válido por el sistema de visión ya que tiene presentes todos los elementos requeridos para que la pieza sea válida.

6 Modelo de información de FMS-210

Para la creación de un determinado gemelo digital a través de la plataforma se debe de tener un contexto general de la misma, entendiendo todas sus clases para poder simular todas las funciones reales de la estación. Teniendo claro esto, resulta bastante intuitivo la manera de proceder a la creación del gemelo digital.

Para crear el gemelo digital deben de añadirse todas sus características a un fichero XML, ligado a la clase escena. A partir de este fichero se obtiene la programación gráfica y lógica de la estación.

Una vez vistos todos los diseños 3D que componen el gemelo digital, se puede proceder a explicar como añadirlos a la plataforma para así componer la estación FMS-210. En los anexos se incluye la configuración completa del fichero XML para componer el gemelo digital de la estación FMS-210, en este capítulo se hace una explicación genérica para comprender como añadir cada tipo de elemento.

En la Figura 104 se muestra un esquema general de los elementos del gemelo digital de la estación con su estructura ordenada según las clases que la forman.

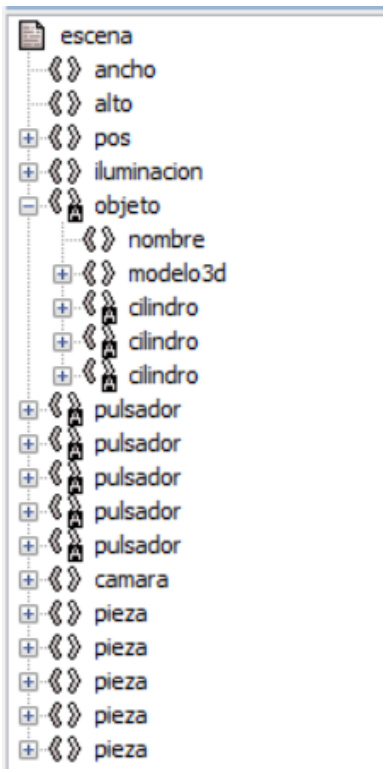


Figura 104. Esquema XML FMS-210

En primer lugar, se debe de definir el ancho y alto de la escena, seguido de la posición donde se va a situar la imagen, en el caso de la Figura 105 para el gemelo digital de la estación FMS-210 toma los siguientes valores.

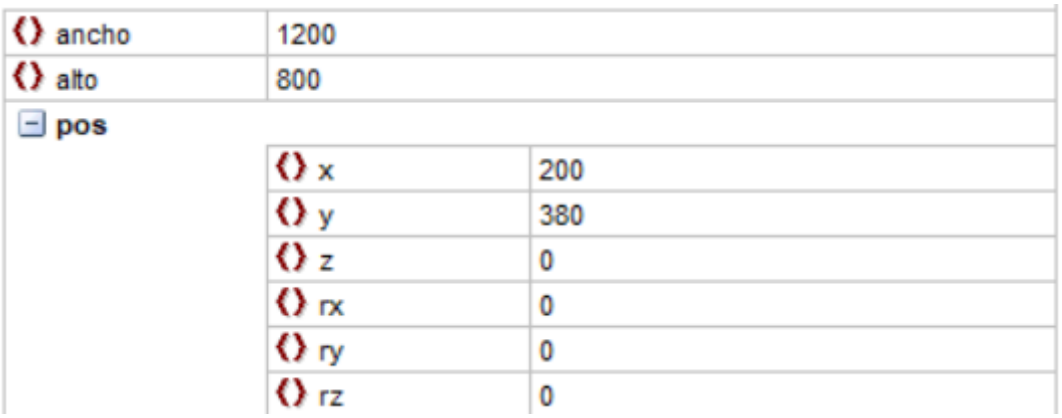


Figura 105. Ancho y alto XML

Objetos fijos: para añadir objetos, como la base fija de la estación se debe de hacer debe de seguir la estructura de la Figura 106.

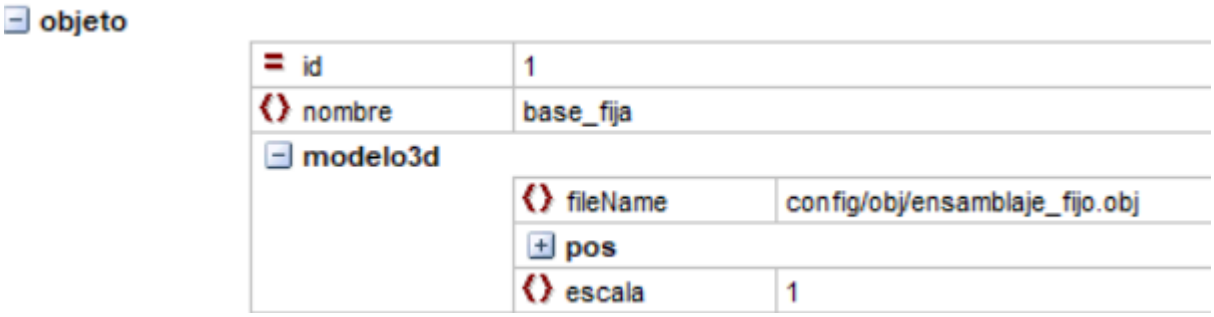


Figura 106. Objeto XML

De este modo se añade un objeto con el identificador número 1 y el nombre de base_fija. Para añadir el archivo .obj, se debe primero añadir ese mismo archivo .obj a la ruta del proyecto *digital-twin/config/obj*, una vez añadido ya se puede acceder a ella a través de la etiqueta filename y la ruta correspondiente.

Por último, se configuran los parámetros de posición/orientación los cuales se componen de 6 variables, tres para la posición y tres más para la orientación del objeto. Se debe de definir la escala. Una vez realizado quedaría definida la base de la estación.

Cilindros:

Dentro de la base fija se han añadido los cilindros, estos deben ir colocados dentro de la base_fija para que ocupen su posición respecto a ella de igual modo a la manera en que realizan sus movimientos. (Ver Figura 107 y Figura 108)

cilindro (3)

	id	nombre	modelo3d	modelo3dMovi	tipo	posMax	posMin
1	2	cilindroA	modelo3d	modelo3dMovi	2	140	-140
2	3	cilindroB	modelo3d	modelo3dMovi	2	360	0
3	4	cilindroC	modelo3d	modelo3dMovi	1	100	0

Figura 107. Cilindro XML I

vel	valvula	sensor	cilindro	sensor
2	valvula	sensor (2)		sensor (3)
2	valvula			
2	valvula	sensor (2)	cilindro id=5	sensor (2)

Figura 108. Cilindro XML II

La etiqueta <nombre>, define el nombre del cilindro.

La etiqueta <modelo3d>, define el modelo 3D del cilindro a través de su ruta, para obtener el archivo .obj, también debe de añadirse la posición en la que se quiere situar el cilindro dentro de la escena.

La etiqueta <modeo3dMovil> sigue el mismo criterio que la etiqueta <model3D>, pero se usa para el vástago del cilindro, esta etiqueta además de la ruta al archivo .obj y la posición en la escena, se le añade una <posMax> y <posMin>, que son la posición máxima y mínima del vástago en la escena, así se puede establecer su rango de movimiento, por último, se le añade la etiqueta <vel>, que asigna la velocidad con la que se mueve el cilindro en la escena.

La etiqueta <valvula> asigna las direcciones de las salidas en el PLC, ya sea para extender o retraer el cilindro.

El cilindro C y D se sitúan consecutivamente ya que para realizar el movimiento el Cilindro C, desplaza al Cilindro D.

Sensor:

sensor (2)

	id	nombre	modelo3d	posMax	posMin	input
1	100	a0	modelo3d	-139	-143	input
2	101	a1	modelo3d	2	-2	input

Figura 109. Sensor XML

Los sensores son añadidos al final de los cilindros, y cuentan con un identificador, un modelo 3D, una posición máxima y otra mínima que sirven para indicar la región en la que el sensor está activado o desactivado. También cuentan con una entrada asociada al PLC que se identifica con la etiqueta input en la que se le asigna el bit y el byte a la entrada. (Ver Figura 109).

Pulsador:

Los pulsadores cuentan, con un identificador, un nombre que suele indicar su función, el tipo que indica si es interruptor o pulsador, un modelado 3D y una dirección asociada a una entrada al PLC a través de la etiqueta input. (Ver Figura 110)

pulsador (5)

	id	nombre	tipo	modelo3d	input
1	20	START	0	modelo3d	input
2	21	STOP	0	modelo3d	input
3	22	AUTO/MAN	0	modelo3d	input
4	23	REARME	0	modelo3d	input
5	23	ALARMA	0	modelo3d	input

Figura 110. Pulsador XML

Cámara:

La clase Camara tiene asignado un nombre y un modelo 3D. También cuenta con dos entradas y una salida. Las entradas son: input, que indica cuando ha finalizado el análisis, y resultado, que indica la validez de la pieza. La salida output es la que se encarga de simular que el análisis de pieza se está realizando. (Ver Figura 111)

camara

nombre	camara
modelo3d	
input	
output	
resultado	

Figura 111. Cámara XML

Pieza:

Para crear un objeto de tipo Pieza, es necesario añadir un nombre, su correspondiente modelado 3D, asignado le también una posición respectiva y un parámetro booleano que definirá si la pieza es válida o no.

Para la creación de piezas se añade un nombre y un modelado 3D, al que se le asigna la posición donde se quiere que sea visible la pieza. Por último, tiene un parámetro que indica si la pieza es válida o no, este parámetro es leído por la cámara para simular el análisis de calidad de la pieza. (Ver Figura 112)

pieza (5)

	nombre	modelo3d	validez
1	caso1	modelo3d	true
2	caso2	modelo3d	true
3	caso3	modelo3d	false
4	caso4	modelo3d	false
5	caso5	modelo3d	true

Figura 112. Pieza XML

7 Simulación de control de calidad

En este capítulo se detalla cómo se ha llevado a cabo la generación de piezas, así como su posterior aparición de manera aleatoria en el gemelo digital. Este capítulo también incluye la explicación del proceso de control de calidad de la pieza en la clase Camara, la cual indica si la pieza es válida o no.

7.1 Generación de piezas

Para la generación de piezas se ha creado una clase llamada Pieza (Figura 113), la cual es la encargada de gestionar y almacenar todos los casos de piezas que se añadan al proceso.

```
public class Pieza {

    private String nombre;
    private Model3d modelo3d;
    private boolean pieza_valida;
    ArrayList<Pieza> casos;
```

Figura 113. Clase Pieza

Cada pieza tiene un nombre, un modelo 3D y un valor booleano que indica si la pieza es válida o no. Cada uno de estos atributos tienen sus métodos get y set para ser accesibles.

Esta clase también crea un ArrayList de piezas que incluye todos los casos, que se añadan, este ArrayList es leído por la clase Escena, para poder ser procesado cada uno de los casos.

La clase también cuenta con las funciones de setup(), que establece las características de las piezas, y de Display() que es la encargada de su representación en la escena.

Para que la generación de piezas sea aleatoria se ha añadido un parámetro llamado nAleatorio en la clase escena y una pieza llamada PiezaVisible. El parámetro nAleatorio número corresponde a la posición en el ArrayList de casos de piezas y piezaVisible es la representada en la escena.

```
piezaVisible = casos.get(nAleatorio);
piezaVisible.Display();
```

Figura 114. Pieza visible en la escena

De este modo la función Display() de escena representa únicamente el caso de pieza que indica el nAleatorio (Figura 114). Para modificar este número, en la clase GemeloDigital, que es la principal que se ejecuta, se ha añadido la función a la tecla “m” para que cada vez que sea pulsada se genere un nAleaotrio nuevo, como se muestra en el código de la Figura 115.

```
case 'm':
   escena.nAleatorio = (int)(Math.random()*4);

    for (Camara c : escena.camaras){
        c.setPieza_Visible(escena.casos.get(escena.nAleatorio));

    }

    break;
```

Figura 115. Cambio de pieza en la escena

Este caso se añade en el método keyPressed(), que se ejecuta cuando se detecta una tecla pulsada. Math.random()*4, genera un número aleatorio entre 0 y 5 que corresponde a los casos que hay añadidos.

7.2 Clase cámara

La clase Camara (Figura 116) es hija de la clase objeto y es la encargada de gestionar 2 entradas y una salida que simulan el comportamiento del sistema de visión. La salida procesa, actúa como salida de procesado de imagen, mientras que la entrada procesado_fin, indica el fin del procesado y resultado_pieza indica la validez de la pieza.

```
public class Camara extends Objeto {
    private PlcInput procesado_fin;
    private PlcOutput procesa;
    private PlcInput resultado_pieza;
    private Pieza pieza_Visible;

    private long actualTiempo;
    private long tiempoProcesado;

    private boolean  procesarOLD;
```

Figura 116. Clase Camara

Las dos entradas y la salida del sistema de visión tienen implementado los métodos get y set para hacerlos accesibles desde otras clases. Para actualizar dichas variables se ha creado un método Update() (ver Figura 117), este método se encarga de actualizar la entrada procesado, cuando se detecta que se ha activado la salida procesa, guarda el tiempo actual, este tiempo lo guarda y lo compara con la variable tiempo procesado hasta que transcurren 5000ms (5 segundos), lo que indica que el tiempo de procesado ha transcurrido, en este momento cambia la entrada procesado a true. Una vez se desactiva la salida procesa, se desactiva también la entrada procesado.

El valor de validez de la pieza (resultado_pieza), se obtiene de la propia pieza en sí, aunque sería más coherente que este valor se mostrara al final el procesado, este valor está disponible desde el mismo momento en que la pieza aparece en la zona inicial del proceso.

```
public void Update(){

    if(procesa== null |procesado_fin == null){return;}
    boolean procesarActual = procesa.getEstado();

    if(procesarActual != procesarOLD & !procesarOLD ){
        actualTiempo = System.currentTimeMillis();
        System.out.println("Estado Procesado " +procesarActual);
    }

    if((System.currentTimeMillis() > tiempoProcesado +actualTiempo)&procesarActual ){
        procesado_fin.setEstado(true);
    }else{
        procesado_fin.setEstado(false);
    }

    procesarOLD = procesarActual;

    if (pieza_Visible != null){
    resultado_pieza.setEstado(pieza_Visible.isPieza_valida());
```

Figura 117. Actualización de la cámara

8 Manual de usuario

Este manual de usuario pretende ser una guía de uso donde se detallarán los pasos a seguir para hacer uso del gemelo digital y se explicarán las funcionalidades de que dispone.

En primer lugar, se necesita que el equipo en el que se vaya a trabajar tenga instalado NetBeans, TIA Portal V14 con PLCSIM y NetToPLCsim.

Con el software NetToPLCSim (Figura 118) se establece la conexión entre el PLCsim y el Gemelo Digital, creando un servidor que actúa de medio de comunicación. Es necesario ejecutar el programa como administrador para que funcione correctamente. Al abrir el programa aparece la siguiente ventana, en la cual se debe de pulsar Add para añadir el servidor.

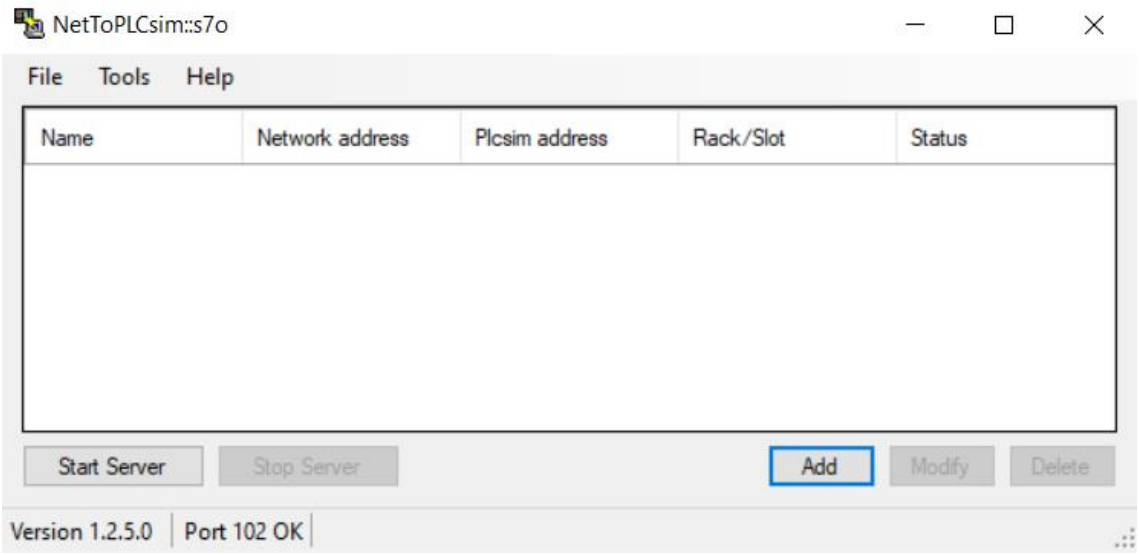


Figura 118. NetToPLCsim, inicio

Una vez pulsado Add, se entra en la ventana de configuración del servidor en la cual hay que configurar 4 parámetros, como se muestra en la Figura 119.

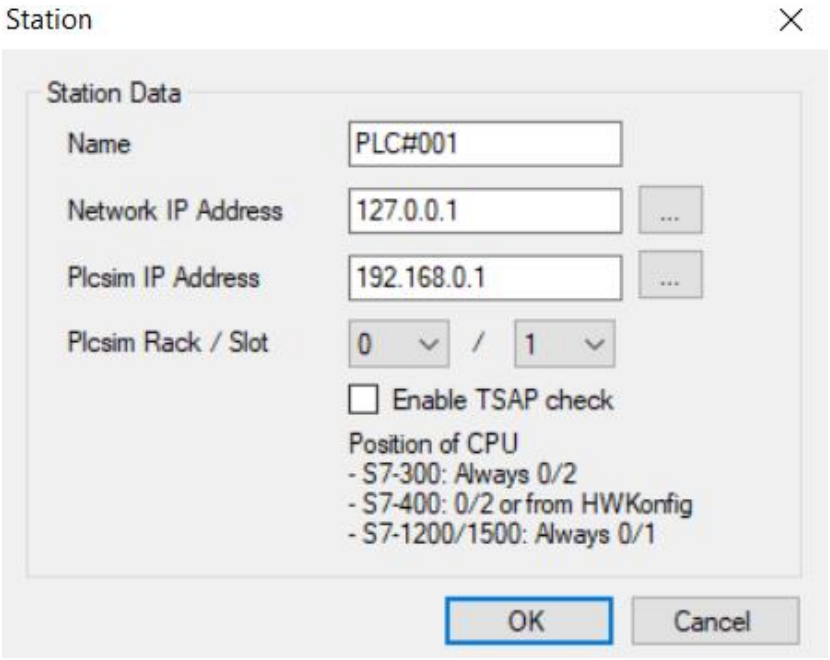


Figura 119. NetToPLCsim, Station Data

Network IP Address es el parámetro que se refiere a la dirección IP de la red en la que se encuentra el PLC simulado. Es la dirección IP asignada a la red en la que están conectados tanto el PLC simulado como los dispositivos externos que se comunican con él. Esta dirección IP de red es necesaria para establecer la comunicación entre el PLC simulado y los dispositivos externos. La dirección IP de loopback (127.0.0.1) se utiliza para hacer referencia al propio dispositivo en el que se encuentra. Cuando se establece la dirección IP de loopback, el PLC simulado se está comunicando consigo mismo, sin involucrar a ninguna red externa.

PLCSim IPAddress es el parámetro se refiere a la dirección IP asignada específicamente al PLC simulado. Es la dirección IP que se utiliza para identificar y comunicarse con el propio PLC simulado. Esta dirección IP del PLC simulado debe estar en el mismo rango de direcciones IP que la red en la que se encuentra para que los dispositivos externos puedan establecer la comunicación con el PLC simulado. En este caso la dirección IP es la 192.168.0.1.

El parámetro "PLCSim Rack/Slot" se refiere a la ubicación virtual del módulo de E/S en el rack del controlador lógico programable (PLC) emulado por el software PLCSim. Un rack de PLCSim puede contener varios módulos de E/S, cada uno con una ubicación única en el rack que se identifica mediante un número de rack y un número de slot.

El número de rack se refiere a la ubicación física del rack en el sistema de automatización industrial real, mientras que el número de slot se refiere a la ubicación física del módulo de E/S en el rack. En NetToPLCSim, estos números se utilizan para especificar la ubicación virtual del módulo de E/S en el rack de PLCSim.

Es importante configurar correctamente el parámetro "PLCSim Rack/Slot" para que coincida con la configuración del hardware del PLC real que se está emulando. De esta manera, se asegura que el programa de automatización que se está desarrollando funcione correctamente en el entorno virtual antes de implementarlo en el entorno real.

Una vez introducidos todos los parámetros, se pulsa OK, para iniciar el servidor, la Figura 120 muestra el servidor funcionando correctamente.

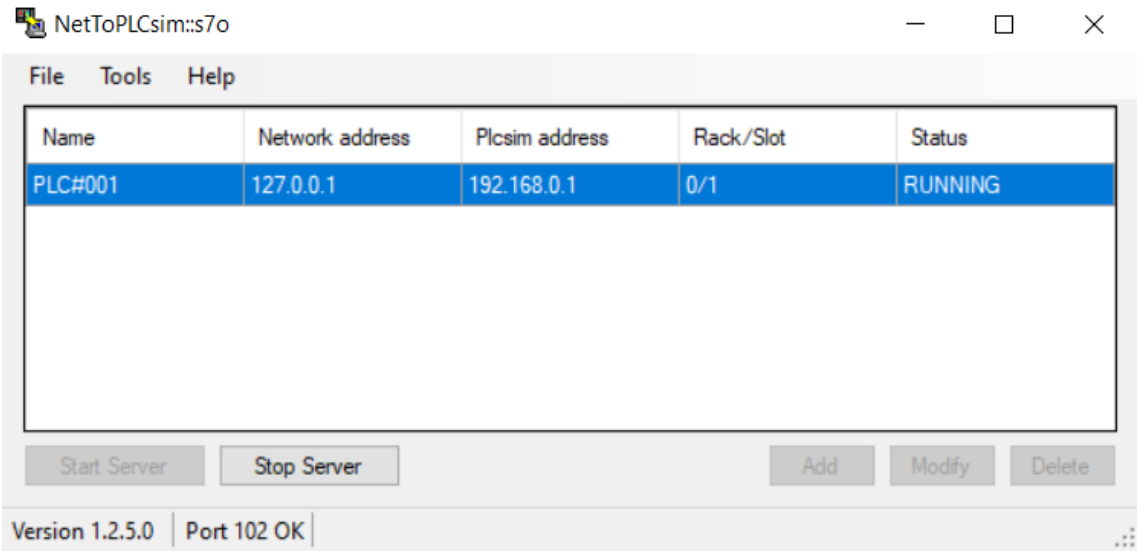


Figura 120. Servidor funcionando

Seguidamente se debe de ejecutar NetBeans y abrir el proyecto digital-twin, como se muestra en la Figura 121. Una vez abierto, para ejecutar la aplicación se debe de pulsar el botón de Run Project.

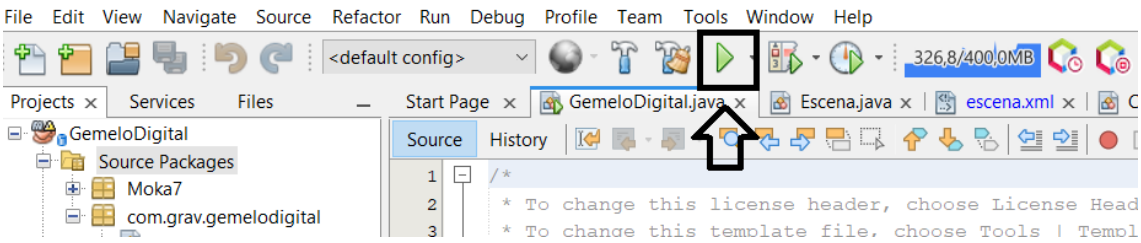


Figura 121. Ejecutar gemelo digital

Lo cual abrirá la aplicación, en la cual, se puede testear manualmente las salidas mientras no se establezca conexión con el PLC. Nótese que la conexión al iniciar la aplicación siempre estará en el estado false y la dirección IP será la 127.0.0.1 (ver Figura 122).

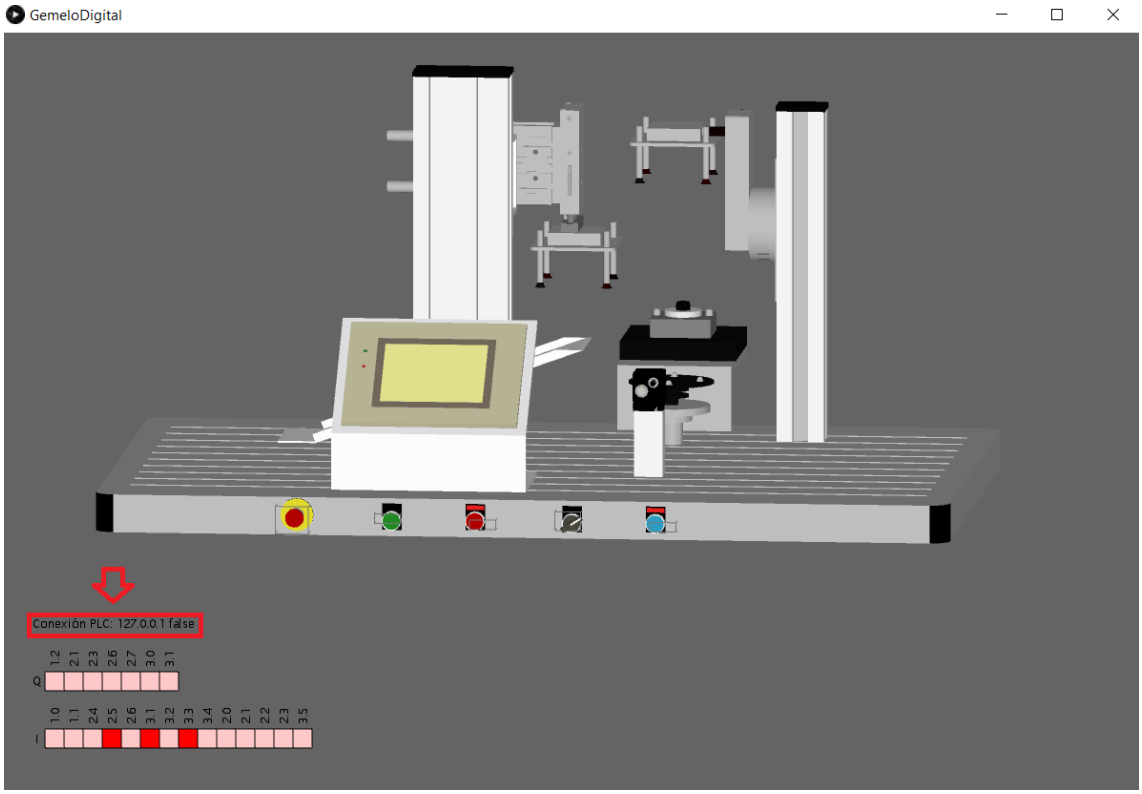


Figura 122. Inicio de la aplicación

El paso siguiente será abrir el proyecto de TIA Portal, en el cual se encuentra la programación que se haya realizado de la estación. En este caso ha sido creado un proyecto llamado dt_vision, este proyecto incluye los movimientos esenciales de la estación y el análisis de calidad de la pieza.

Si no se tiene ningún proyecto creado, se debe de crear un nuevo proyecto, para ello se pulsa en la barra de tareas en proyecto y se selecciona Nuevo en el desplegable que aparece.

Lo primero que se debe hacer al crear un nuevo proyecto es añadir un dispositivo, para ello, en la pestaña del proyecto, se pulsa agregar dispositivo, y se selecciona el PLC que se quiera añadir, en este caso se debe añadir el PLC de la Figura 123.

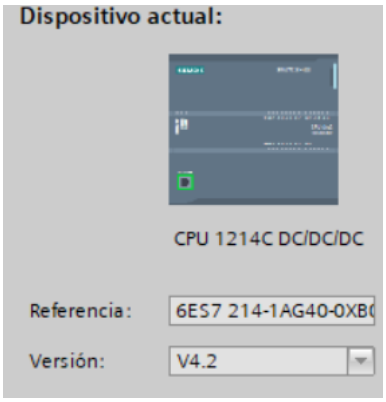


Figura 123. Elección de PLC

Aparecerá añadido al proyecto del siguiente modo, con el rack 0 y el módulo 1 como se le indica en NetToPLCSim como se observa en la Figura 124.

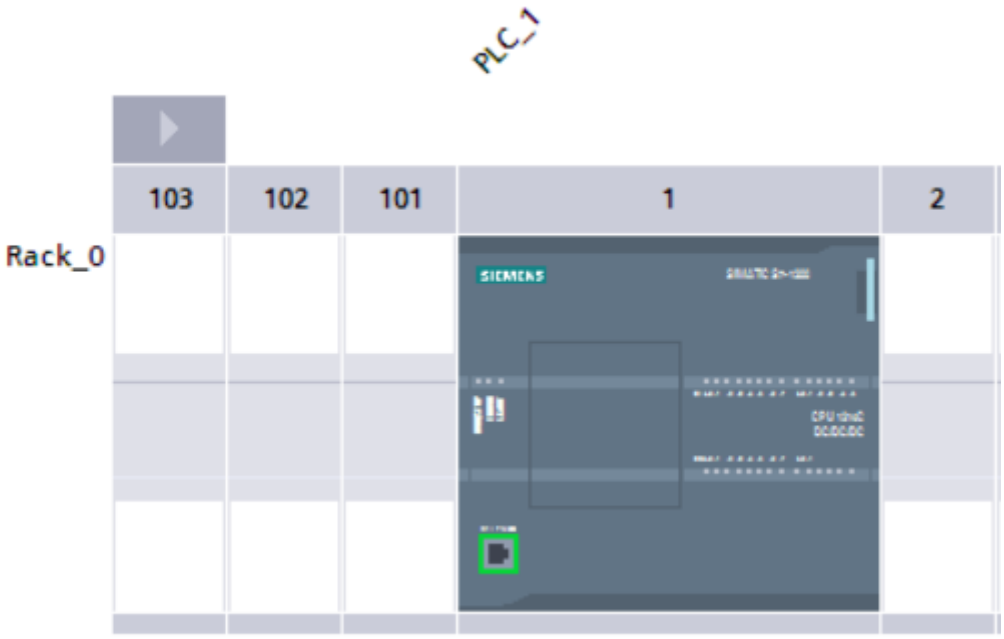


Figura 124. Rack y Slot del PLC

Una vez añadido el PLC se debe realizar la programación de la estación, configurando todas las variables que se vayan a utilizar y los bloques de programa, tanto Startup (OB100), como el main (OB1).

Una vez finalizado el programa que se quiera que lleve a cabo la estación, para iniciar la simulación se deben seguir los pasos que muestran las figuras: Figura 125, Figura 126 y Figura 127.

Figura 125. Inicio de simulación

Se pulsa el botón para iniciar simulación:

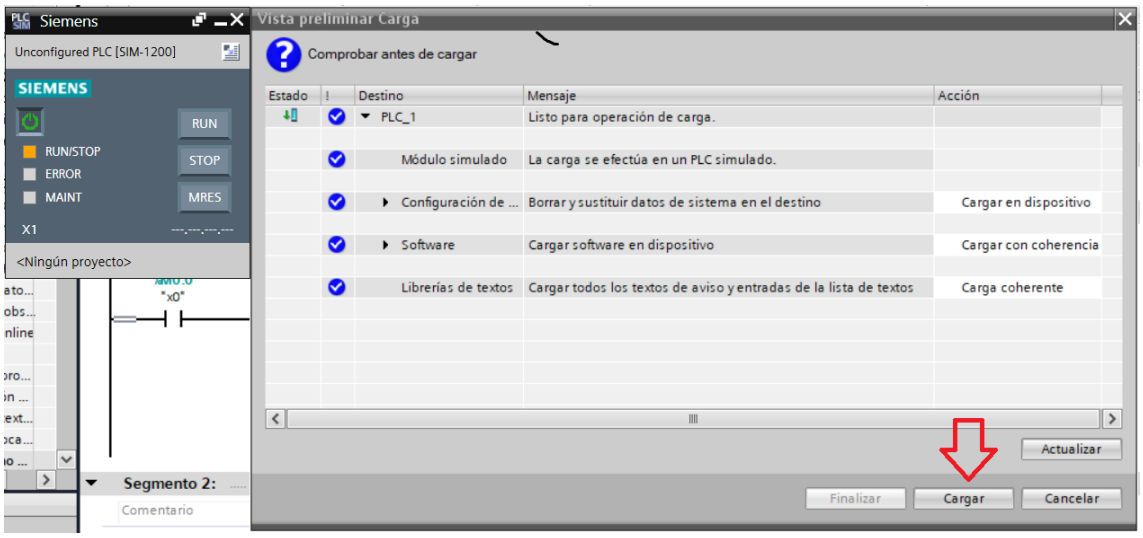


Figura 126. Carga del PLC

Se abrirá una ventana que indica el estado del PLC, la configuración del módulo y del software, si todo esta correcto se pulsa cargar.

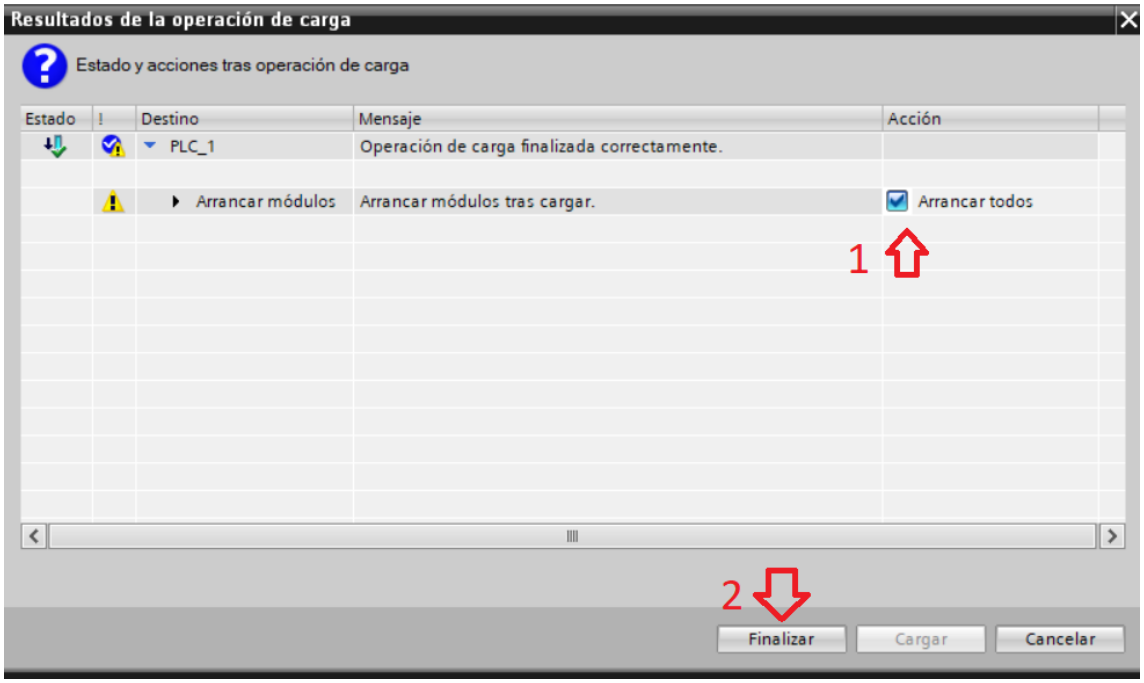


Figura 127. Arranque del PLC

Una vez cargado, aparecerá una ventana en la cual se debe indicar, arrancar todos los módulos, para por último finalizar la configuración.

Si todo se ha realizado correctamente el PLC simulado debe de estar funcionando.

Para ver el estado del proceso, en la barra del proyecto, se pulsa en el main OB1.

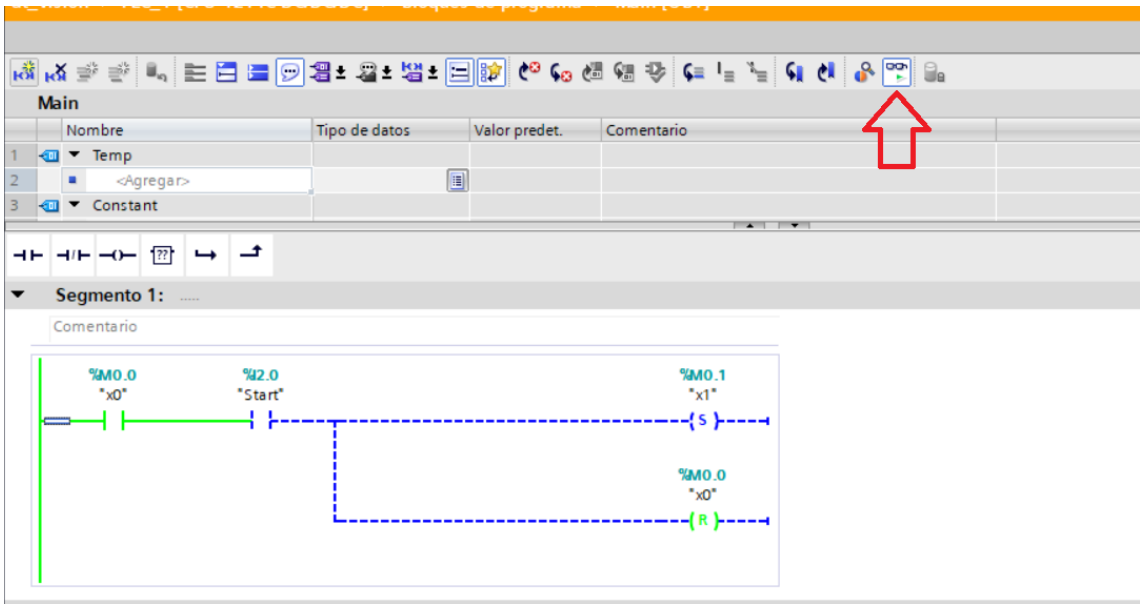


Figura 128. Ventana de simulación

Para ver en tiempo real el proceso de la estación se pulsa el botón de las gafas, en la barra de tareas de la simulación (ver Figura 128).

Por último, hay que volver a la aplicación y pulsar el botón “p” para establecer la conexión, si se han realizado todos los pasos anteriores correctamente el estado de la conexión cambiara a true.

Con la conexión realizada se pulsa el botón de start en la aplicación para iniciar el proceso. (ver Figura 129)

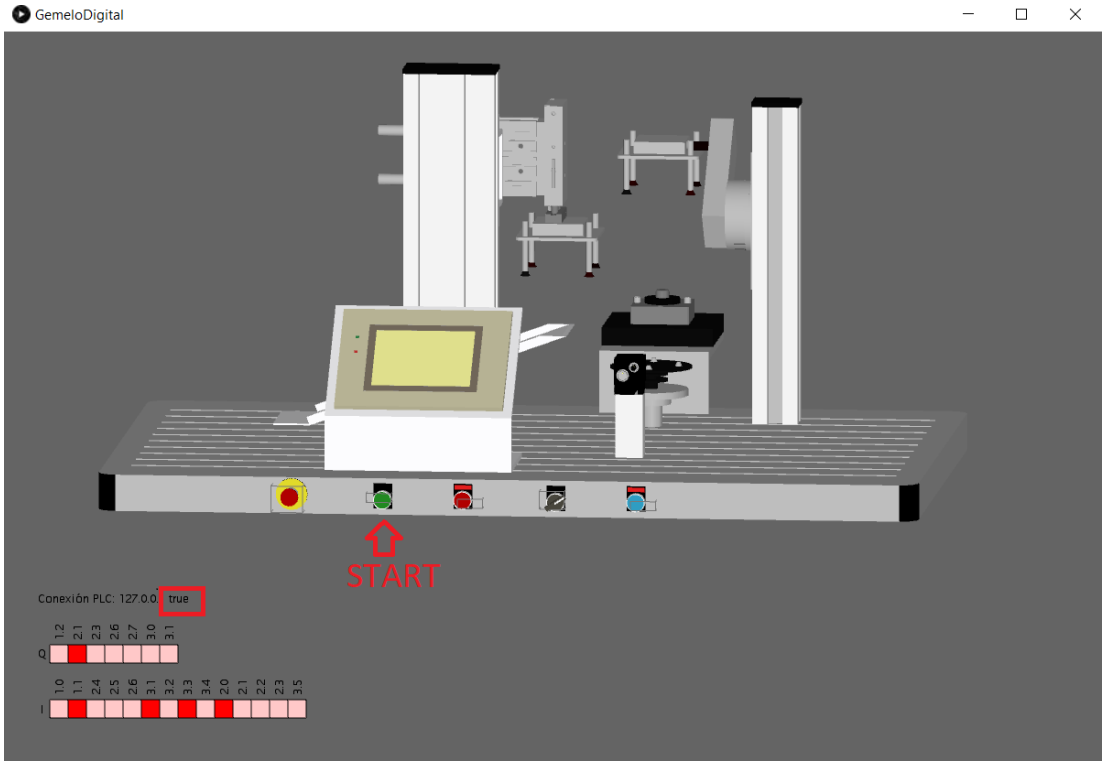


Figura 129. Conexión del gemelo digital correcta e inicio del proceso

El proceso se inicia con la rotación del conjunto A, el cual simula desplazarse para recoger la pieza y moverla desde el punto inicial al punto de examen de la pieza, en la Figura 129, se observa el cilindro rotando para coger la pieza.

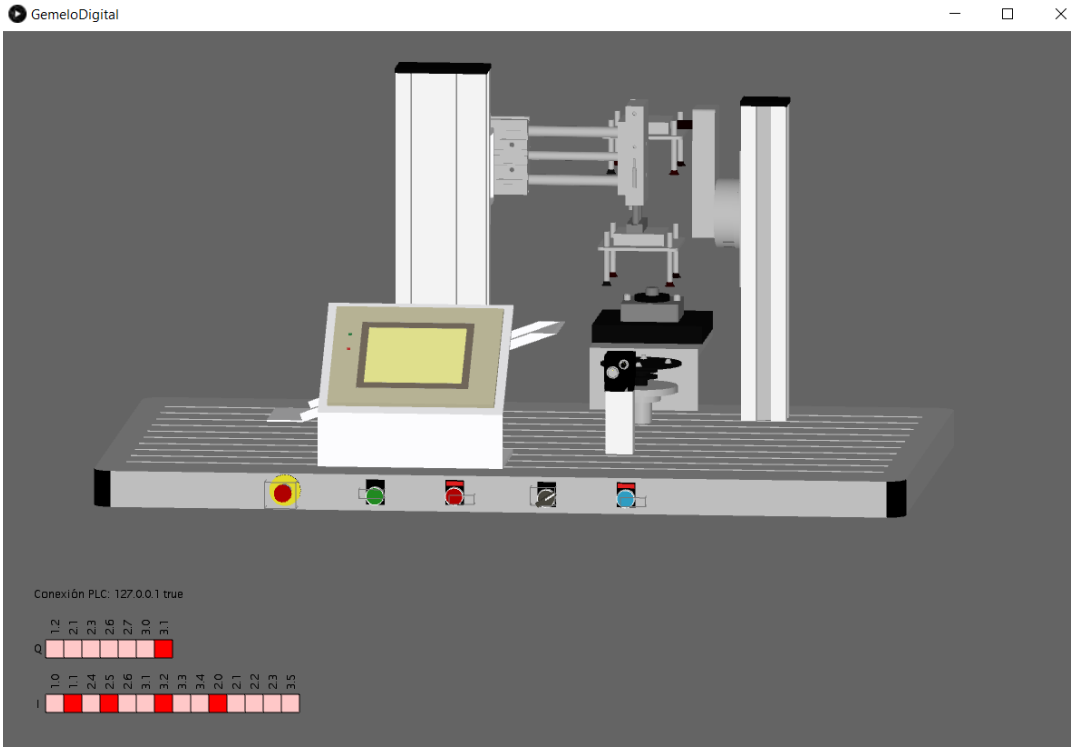


Figura 130. Proceso

Una vez terminado el examen de la pieza el conjunto C y D la transportan de la zona de control de calidad a la posición final, como se puede observar en la Figura 130.

En lo referente al control de calidad de la pieza, será indicado de forma inmediata por la entrada 1.1. En el caso de la Figura 130 se indica la pieza como válida, para ver un ejemplo de pieza no válida se presenta la Figura 131.

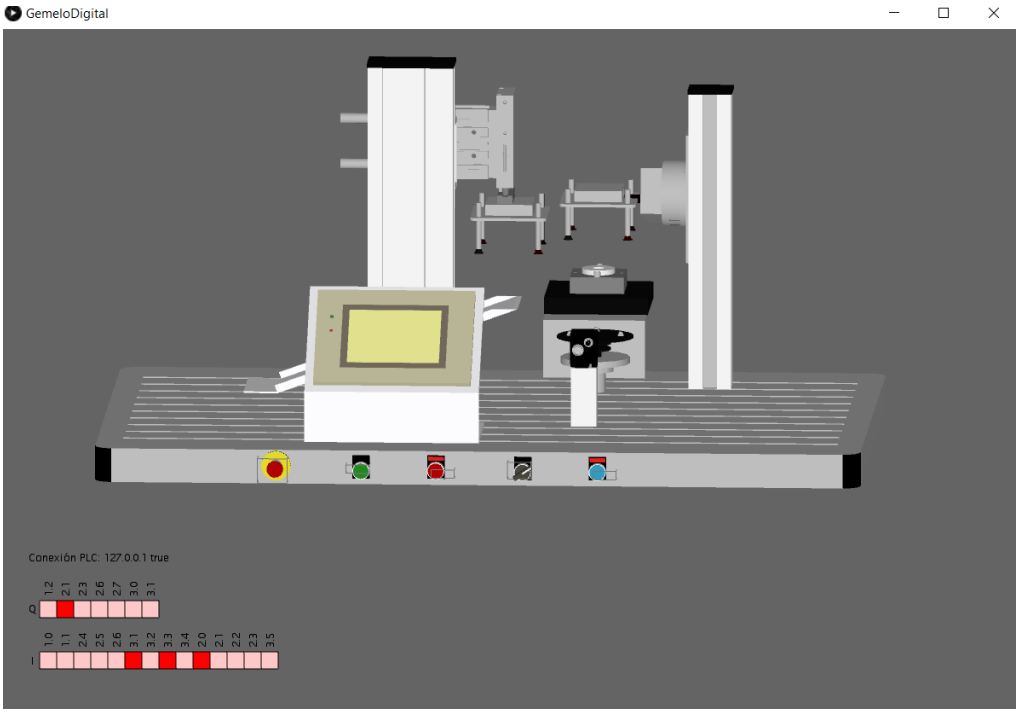


Figura 131. Caso de pieza no valida.

De este modo queda simulado el proceso revisión de calidad de la estación FMS-210 mediante su gemelo digital.

Se incluye un código QR con una demostración en video del proceso completo.

9 Conclusiones

Una vez finalizado el proyecto, hay que pensar en los objetivos propuestos al inicio y la realización de los mismos. En lo referente al diseño de la iluminación de la escena, se ha conseguido un avance muy considerable, el cual resultará útil a los usuarios de la aplicación. Se ha desarrollado de forma que resulte intuitivo y fácil de poner a prueba para el usuario, de modo que se consiga una iluminación de la escena adecuada sin problemas.

Por otra parte, observando el gemelo digital de la estación FMS-210, aunque este gemelo se amolda bastante al real, aún quedan mejoras que deben de ser implementadas para el futuro. Sería necesario añadir motores paso a paso a la aplicación, para poder simular mejor el funcionamiento del gemelo digital. Siguiendo en la misma línea, también debería de trabajarse en la implementación de la visibilidad de la pieza a lo largo del proceso.

En la parte referente a el control de calidad de la pieza, hay que destacar que, aunque es fiable, también es básico, han quedado sin implementar muchas de las entradas del sistema de visión, se debería de poder mejorar para simular un comportamiento más completo del sistema.

Para terminar, se espera haber contribuido al desarrollo de la aplicación de manera positiva con todo lo aportado. Se prevé que este trabajo resulte útil para los fines de desarrollo de gemelos digitales y se haga buen uso de la aplicación por los alumnos de la universidad.

Referencias

AUTODESK. (2022). *AUTODESK*. Obtenido de https://www.autodesk.es/

Class PApplet. (2022). *github*. Obtenido de https://processing.github.io/processing-javadocs/core/processing/core/PApplet.html

infoPLC. (10 de Enero de 2021). Obtenido de https://www.infoplc.net/descargas/107-siemens/software-step7-tiaportal/tia-portal/3195-nettoplcsim-extension-red-simulador-plcsim-siemens

ITI. (30 de Junio de 2020). *INVESTIGATE TO INNOVATE*. Obtenido de https://www.iti.es/proyectosidi/proyecto-gemelos-digitales-industria-4-0/

Mateo, M. (30 de Septiembre de 2020). *integratecnologia.es*. Obtenido de https://www.integratecnologia.es/la-innovacion-necesaria/digital-twins-la-quintaesencia-de-la-industria-4-0/

NETBEANS. (Junio de 2023). *NETBEANS.APACHE*. Obtenido de https://netbeans.apache.org/

NetToPLCsim. (2022). *nettoplcsim*. Obtenido de https://nettoplcsim.sourceforge.net/

OMRO. (Marzo de 2005). Obtenido de https://assets.omron.eu/downloads/manual/en/z141_f150-v3_setup_manual_en.pdf

Pozas, J. L. (30 de Junio de 2022). *CIO Mexico*. Obtenido de https://cio.com.mx/que-es-un-gemelo-digital-una-representacion-virtual-en-tiempo-real/

Processing. (s.f.). Obtenido de https://processing.org/reference/directionalLight_.html

Processing. (s.f.). Obtenido de https://processing.org/reference/lightSpecular_.html

Processing. (s.f.). Obtenido de https://processing.org/reference/lightFalloff_.html

Processing. (s.f.). Obtenido de https://processing.org/reference/spotLight_.html

Processing. (s.f.). Obtenido de https://processing.org/reference/ambientLight_.html

Rojko, M. (25 de Octubre de 2017). Obtenido de https://github.com/xtrinch/moka7-live

SIEMENS. (s.f.). Obtenido de https://resources.sw.siemens.com/es-ES/e-book-leverage-closed-loop-digital-twin

SIEMENS. (JUNIO de 2023). *TIA Portal (Totally Integrated Automation Portal)*. Obtenido de https://www.siemens.com/ar/es/productos/automatizacion/software-industrial/tia-portal.html#:~:text=El%20Portal%20de%20automatizaci%C3%B3n%20totalmente,integrada%20hasta%20la%20operaci%C3%B3n%20transparente.

SMC CORPORATION. (2021). Obtenido de https://docs.rs-online.com/205d/A700000006746896.pdf

SMC CXSM15-50. (s.f.). Obtenido de https://www.smcpneumatics.com/CXSM15-50.html

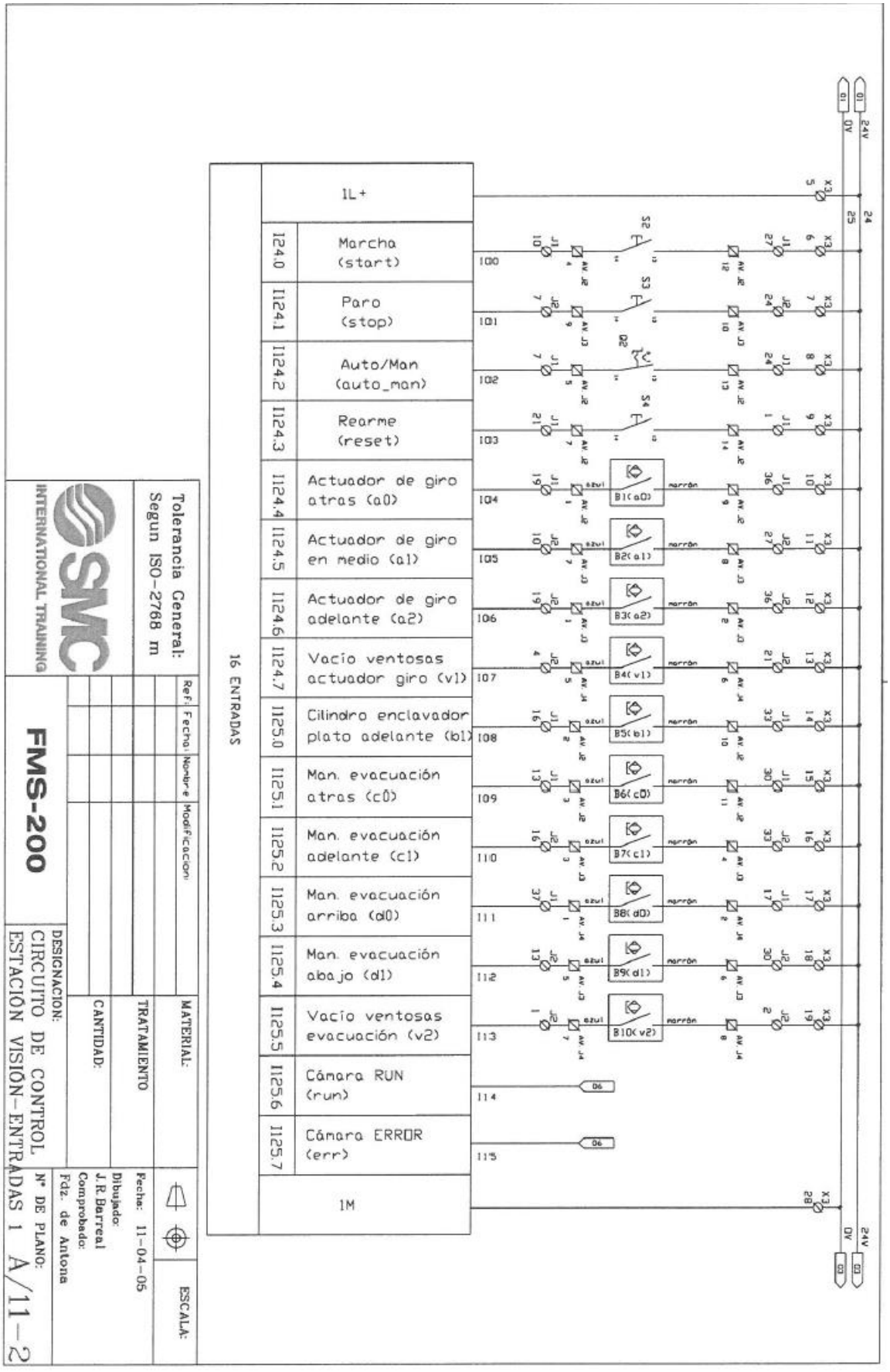
SMC. (s.f.). *https://www.smctraining.com/es*. Obtenido de https://www.smctraining.com/es/webpage/indexpage/287

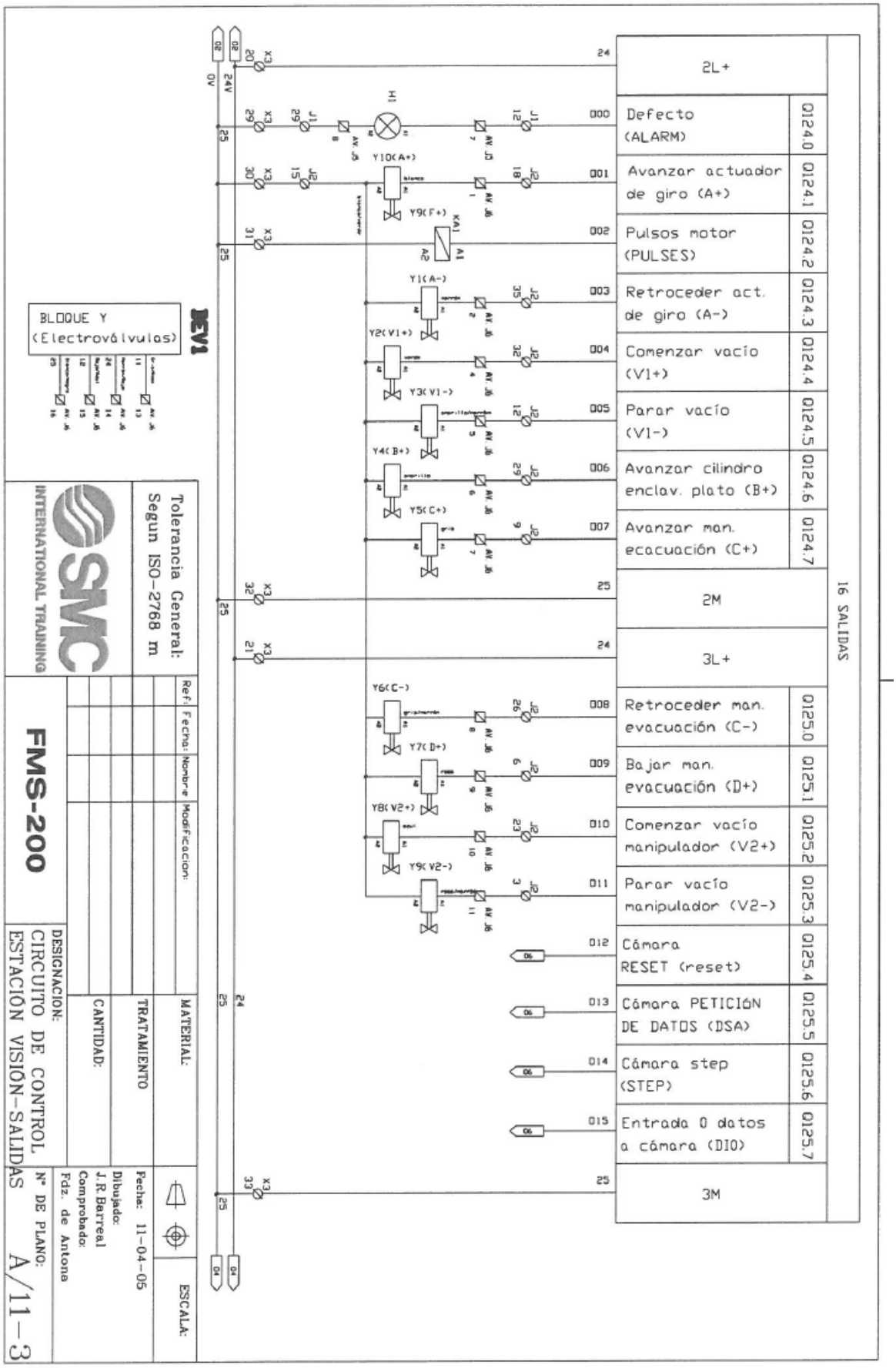
SMC MGPM20-100. (s.f.). Obtenido de https://smcpneumatics.com/MGPM20-100.html

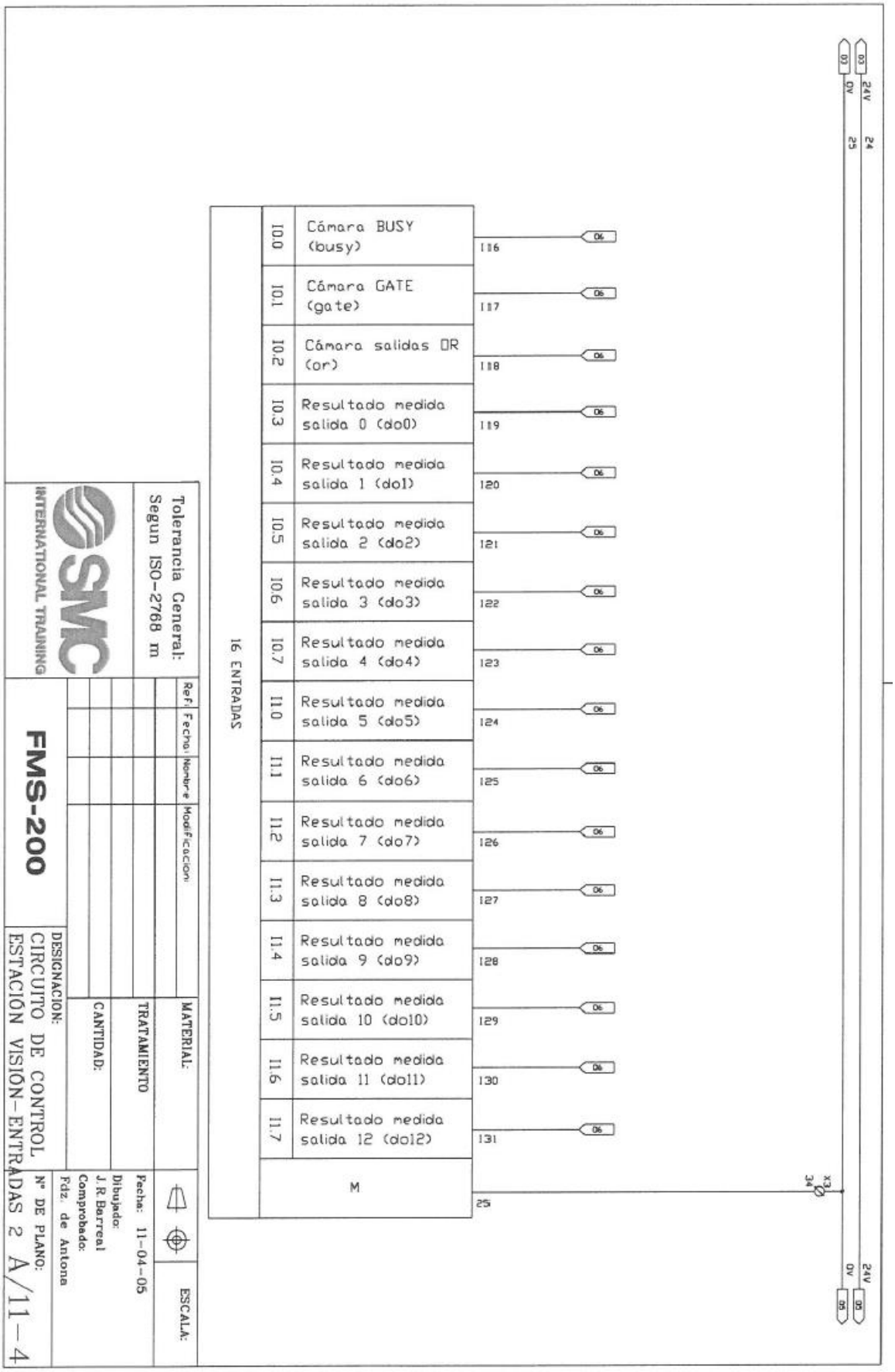
SMC MSQB50A. (s.f.). *https://www.smcpneumatics.com/*. Obtenido de https://www.smcpneumatics.com/MSQB50A.html

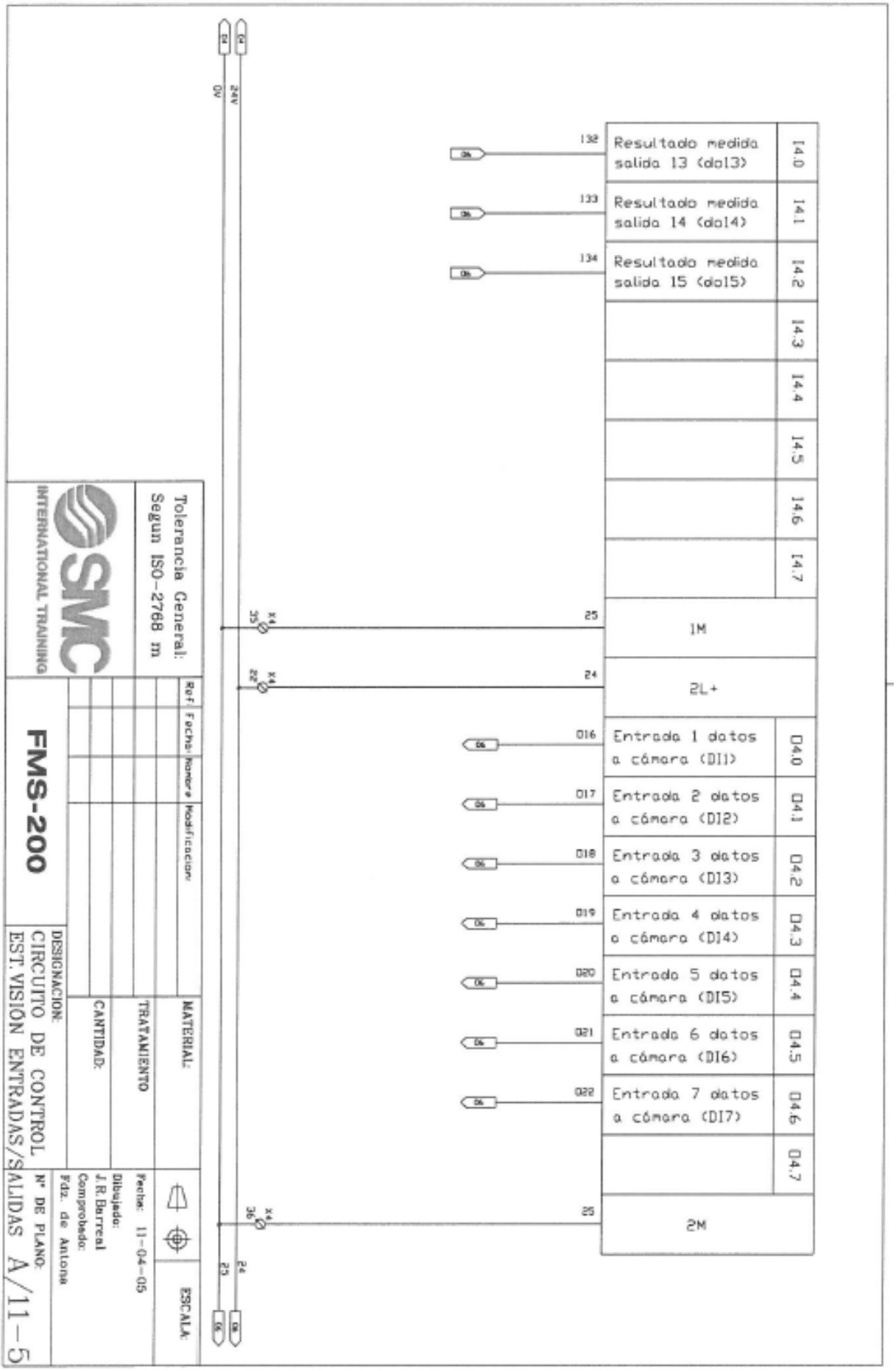
XML. (2022). *aprenderaprogramar*. Obtenido de https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=102:ique-es-y-para-que-sirve-el-lenguaje-de-etiquetas-xml-extensible-markup-language&catid=46&Itemid=163

ANEXO A: PLANOS ELÉCTRICOS DE LA ESTACIÓN




16 SALIDAS
2L+
Defecto (ALARM) Q124.0
Avanzar actuador de giro (A+) Q124.1
Pulsos motor (PULSES) Q124.2
Retroceder act. de giro (A-) Q124.3
Comenzar vacío (V1+) Q124.4
Parar vacío (V1-) Q124.5
Avanzar cilindro enclav. plato (B+) Q124.6
Avanzar man. ecacuación (C+) Q124.7
2M
3L+
Retroceder man. evacuación (C-) Q125.0
Bajar man. evacuación (D+) Q125.1
Comenzar vacío manipulador (V2+) Q125.2
Parar vacío manipulador (V2-) Q125.3
Cámara RESET (reset) Q125.4
Cámara PETICIÓN DE DATOS (DSA) Q125.5
Cámara step (STEP) Q125.6
Entrada 0 datos a cámara (DIO) Q125.7
3M
BLOQUE Y (Electroválvulas)
REV1
Tolerancia General: Según ISO-2768 m
INTERNATIONAL TRAINING
FMS-200
DESIGNACION: CIRCUITO DE CONTROL ESTACIÓN VISIÓN-SALIDAS
MATERIAL:
TRATAMIENTO
CANTIDAD:
ESCALA:
Fecha: 11-04-05
Dibujado: J.R.Berreal
Comprobado: Fdz. de Antona
Nº DE PLANO: A/11-3


16 ENTRADAS
I0.0 Cámara BUSY (busy)
I0.1 Cámara GATE (gate)
I0.2 Cámara salidas OR (or)
I0.3 Resultado medida salida 0 (do0)
I0.4 Resultado medida salida 1 (do1)
I0.5 Resultado medida salida 2 (do2)
I0.6 Resultado medida salida 3 (do3)
I0.7 Resultado medida salida 4 (do4)
I1.0 Resultado medida salida 5 (do5)
I1.1 Resultado medida salida 6 (do6)
I1.2 Resultado medida salida 7 (do7)
I1.3 Resultado medida salida 8 (do8)
I1.4 Resultado medida salida 9 (do9)
I1.5 Resultado medida salida 10 (do10)
I1.6 Resultado medida salida 11 (do11)
I1.7 Resultado medida salida 12 (do12)
M
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
25
24V
0V
24
25
X3
34
Tolerancia General:
Segun ISO-2768 m
INTERNATIONAL TRAINING
FMS-200
DESIGNACIÓN:
CIRCUITO DE CONTROL
ESTACIÓN VISIÓN-ENTRADAS 2
Nº DE PLANO:
A/11-4
MATERIAL:
TRATAMIENTO
CANTIDAD:
Fecha: 11-04-05
Dibujado:
J.R Barreal
Comprobado:
Fdz. de Antona
ESCALA:


Resultado medida salida 13 (dol3)
Resultado medida salida 14 (dol4)
Resultado medida salida 15 (dol5)
1M
2L+
Entrada 1 datos a cámara (DI1)
Entrada 2 datos a cámara (DI2)
Entrada 3 datos a cámara (DI3)
Entrada 4 datos a cámara (DI4)
Entrada 5 datos a cámara (DI5)
Entrada 6 datos a cámara (DI6)
Entrada 7 datos a cámara (DI7)
2M
0V
24V
Tolerancia General: Segun ISO-2768 m
INTERNATIONAL TRAINING
FMS-200
DESIGNACION:
CIRCUITO DE CONTROL EST.VISIÓN ENTRADAS/SALIDAS
CANTIDAD:
TRATAMIENTO
MATERIAL:
ESCALA:
Dibujado: J.R.Barreal
Fecha: 11-04-05
Comprobado:
Fdz. de Antona
Nº DE PLANO: A/11-5