INSTITUTO TECNOLÓGICO SUPERIOR DEL SUR DE GUANAJUATO

Diseño y desarrollo de planta virtual controlable por un PLC vía MODBUS/TCP

Opción II Titulación Integral – Tesis profesional

Elaborada por:

Alexis Roberto García Guzmán

Que presenta para obtener el título de:

INGENIERO ELECTRÓNICO

Asesor:

M.C. Julio Ortega Alejos

Uriangato, Gto. Octubre de 2025

“Diseño y desarrollo de planta virtual controlable por un PLC vía MODBUS/TCP”

Elaborada por:

Alexis Roberto García Guzmán

Aprobado por. . . . . .

M.C Julio Ortega Alejos
Docente de la carrera de Ingeniería en Sistemas Automotrices
Asesor de Tesis Profesional

Revisado por. . . . . .

Ing. Netzahualcóyotl Martínez Cázares
Docente de la carrera de Ingeniería Electrónica
Revisor de Tesis Profesional

Revisado por. . . . . .

M.C. Susana Violeta Martínez Hernández
Docente de la carrera de Ingeniería Electrónica
Revisor de Tesis Profesional

LIBERACIÓN DE PROYECTO PARA LA TITULACIÓN INTEGRAL

Uriangato,Gto.,31/octubre/2025

Asunto: Liberación de proyecto para la titulación integral

M.C. José Gabriel Aguilera González
Director Académico
ITSUR
PRESENTE

Por este medio informo que ha sido liberado el siguiente proyecto para la titulación integral:

Nombre de estudiante y/o egresado(a): Alexis Roberto García Guzmán	
Carrera: Ingeniería Electrónica	Núm. de control: E20120406
Nombre del proyecto: Diseño y desarrollo de planta virtual controlable con un PLC vía MODBUS/TCP	
Producto: Tesis Profesional	

Agradezco de antemano su valioso apoyo en esta importante actividad para la formación profesional de nuestras y nuestros egresados.

ATENTAMENTE

MTI. Eduardo Arroyo Ortega
Jefe de División de Ingeniería Electrónica
ITSUR

Instituto Tecnológico Superior del Sur de Guanajuato
ESTADOS UNIDOS MEXICANOS
COORDINACIÓN INGENIERÍA ELECTRÓNICA

La comisión revisora ha tenido a bien aprobar la reproducción de este trabajo.

M.C. Julio Ortega Alejos Asesor (a)	Ing. Netzahualcóyotl Martínez Cázares Revisor(a)* 1	M.C. Susana Violeta Martínez Hernández Revisor(a)* 2

c.c.p.- Expediente

Resumen

El presente proyecto tuvo como objetivo el diseño, desarrollo e implementación de una planta virtual controlable por un PLC Mitsubishi FX5U, utilizando el protocolo MODBUS/TCP.

El desarrollo de esta planta virtual surge de la necesidad de contar con herramientas didácticas que permitan la comprensión y el entrenamiento en temas de automatización industrial. Reduciendo el costo, y los tiempos de desarrollo de proyectos educativos en este campo de estudio.

La planta virtual fue diseñada en SolidWorks, optimizada en Blender e implementada en Unity 3D. En ella se implementaron sensores, actuadores, un sistema de visión artificial, un eje de movimiento, programados en C#, y comunicados con el PLC en tiempo real, utilizando la librería NModbus, para utilizar el protocolo MODBUS/TCP.

Durante las pruebas se comprobó el funcionamiento de los subsistemas, la estabilidad en la comunicación entre la planta virtual y el PLC, y el rendimiento de la aplicación de la planta virtual. La planta virtual tiene una frecuencia de sondeo de 1000 HZ, permitiendo que la interacción entre la planta virtual y el PLC se ejecute en tiempo real., replicando de forma precisa el comportamiento de una planta física.

El resultado final es una herramienta funcional y accesible para la enseñanza del control y automatización, que permite una simulación fiel de componentes reales.

Palabras claves: Planta virtual, PLC, MODBUS/TCP, Unity 3D.**,** FX5U, simulación.

Abstract

The present project aimed at the design, development, and implementation of a virtual plant controllable by a Mitsubishi FX5U PLC, using the MODBUS/TCP protocol.

The development of this virtual plant arises from the need for didactic tools that facilitate understanding and training in industrial automation topics, reducing costs and development time for educational projects in this field of study.

The virtual plant was designed in SolidWorks, optimized in Blender, and implemented in Unity 3D. It includes sensors, actuators, a vision system, and a motion axis, all programmed in C# and connected to the PLC in real time through the NModbus library to implement the MODBUS/TCP protocol.

During testing, the operation of the subsystems, the stability of communication between the virtual plant and the PLC, and the overall performance of the application were verified. The virtual plant maintains a polling frequency of 1000 Hz, allowing real-time interaction with the PLC and accurately replicating the behavior of a physical plant.

The final result is a functional and accessible tool for teaching control and automation, enabling a faithful simulation of real industrial components.

Key Words: Virtual plant, PLC, MODBUS/TCP, Unity 3D., FX5U, simulation.

Agradecimientos

Agradezco profundamente a mis padres, por brindarme la oportunidad y apoyo para desarrollarme a lo largo de toda mi trayectoria escolar. Por los sacrificios que realizaron para permitirme vivir de la forma que lo hago. Por confiar en mí, y por todo el amor que me han dado.

A mis hermanos, por su ejemplo, cariño, y cada juego que hemos tenido a lo largo de toda mi vida.

A Diana Lucía Juárez Salgado, mi novia, por inspirarme, por su apoyo, y por motivarme a perseguir mis sueños.

Dedicatoria

A mi padre, Roberto García Gutiérrez, por ser la chispa que encendió mi curiosidad. Por inculcarme el amor por el conocimiento. Y por dedicar su vida para que sus hijos tuvieran una vida mejor que la él tuvo.

A mi madre, Elena Guzmán Aguilera, por educarme par ser una persona de bien, por su apoyo, y por su amor incondicional.

A mi novia, Diana Lucía Juárez Salgado, por ser la luz de mi vida.

Tabla de contenido

TABLA DE ILUSTRACIONES

Capítulo 1

Introducción.

En la actualidad, los entornos de simulación 3D representan una herramienta capaz de replicar con precisión el comportamiento de sistemas industriales. Ya que no necesitan los equipos reales, representan, además, un ahorro de costos, riesgos, y tiempos de implementación. Resultando en una alternativa eficaz para la enseñanza del control y programación de este tipo de sistemas.

El presente proyecto tiene como propósito el desarrollo de una planta virtual basada en una línea de ensamble de chumaceras, diseñada en SolidWorks, optimizada en Blender,e implementada en Unity 3D. Dicha planta incluye la simulación de sensores, actuadores, sistemas de control de movimiento, y un sistema de visión artificial, todos ellos comunicados en tiempo real con un PLC Mitsubishi FX5U utilizando el protocolo industrial MODBUS/TCP.

La propuesta surge de la necesidad de contar con herramientas que permitan la enseñanza del control y procesos industriales, de forma accesible y sin depender de equipo físico. Permitiendo que alumnos sean entrenados en configuración, programación, y operación de sistemas industriales.

Durante el desarrollo de este proyecto, se realizaron procesos de modelado 3D, optimización de mallas, texturizado, animación, desarrollo de shaders personalizados, programación en C#, y la implementación de una comunicación industrial entre el entorno virtual y el PLC.

El resultado final es una planta virtual funcional, capaz de replicar el comportamiento de una línea de ensamble física de forma realista, a la que se le pueden aplicar rutinas de control, manejo de señales de sensores y actuadores, y analizar el desempeño general del sistema bajo condiciones simuladas en tiempo real.

Capítulo 2

Marco teórico (Antecedentes).

2.1 Automatización industrial

La automatización industrial consiste en la aplicación de tecnologías de control para operar maquinaria y procesos con una intervención humana mínima. Su objetivo principal es mejorar la productividad, la calidad del producto y la seguridad de las operaciones.

Los sistemas automatizados integran sensores, actuadores, controladores lógicos programables (PLC), redes industriales y software de supervisión. En ellos, los PLC son los encargados de recibir señales de los sensores, procesarlas mediante una lógica definida y activar los actuadores según las condiciones del proceso (Mitsubishi Electric, s. f.).

El desarrollo de entornos de simulación para la automatización permite analizar, probar y optimizar estas interacciones sin depender de hardware físico, reduciendo costos y riesgos en la implementación (Jiao, Li & Bing, 2018).

2.2 Controladores lógicos programables (PLC)

El PLC (Programmable Logic Controller) es un dispositivo electrónico diseñado para ejecutar rutinas de control de manera confiable en ambientes industriales. Su arquitectura se basa en entradas digitales y analógicas, una unidad central de procesamiento (CPU) y salidas que controlan actuadores.

Los PLC permiten programar y modificar fácilmente los procesos mediante lenguajes estandarizados (como Ladder o Structured Text) definidos en la norma IEC 61131-3.

En este proyecto se utilizó un PLC Mitsubishi FX5U, el cual pertenece a la serie iQ-F. Este modelo integra funciones avanzadas de comunicación, posicionamiento y manejo de altas velocidades, lo que facilita su integración con sistemas externos mediante protocolos industriales (Mitsubishi Electric, s. f.).

La programación del controlador se realizó utilizando GX Works 3, un entorno de desarrollo integrado que permite la creación, depuración y monitoreo de programas en los controladores Mitsubishi de las series FX, L y Q (Mitsubishi Electric, 2024a).

2.3 Comunicación industrial MODBUS/TCP

MODBUS/TCP es una versión del protocolo MODBUS adaptada para redes Ethernet. Su función es estandarizar la comunicación entre dispositivos industriales, permitiendo el intercambio de información entre controladores, sensores y sistemas de supervisión.

En este protocolo, el dispositivo maestro (en este caso, el PLC Mitsubishi FX5U) envía solicitudes a dispositivos esclavos (como el entorno virtual desarrollado en Unity 3D), los cuales responden con los valores de las variables solicitadas.

Gracias a su simplicidad, compatibilidad y amplia adopción, MODBUS/TCP se ha consolidado como una de las opciones más utilizadas en la comunicación entre PLCs y sistemas externos (Ibrahim, Andang & MSN, 2024).

Para la implementación del protocolo en C#, se utilizó la librería NModbus, un paquete .NET de código abierto que proporciona clases para la lectura y escritura de registros MODBUS, soportando las variantes RTU y TCP (NModbus, 2024).

2.4 Simulación de procesos industriales

La simulación industrial permite recrear el comportamiento de una planta o línea de producción mediante modelos digitales, posibilitando el análisis y optimización de los procesos antes de su implementación física.

Este enfoque reduce costos, riesgos y tiempos de desarrollo, y facilita la detección de errores en la programación de control (Villarroel, Toapanta, Naranjo & Ortiz, 2023).

Los modelos virtuales pueden incluir características físicas, sensores, actuadores y controladores simulados, logrando una representación funcional del sistema. Cuando la simulación se conecta a un PLC real, se genera un entorno híbrido donde el software reproduce el comportamiento del proceso físico y el controlador ejecuta las mismas rutinas que utilizaría en una planta real (Haces-García & Zhu, 2024).

2.5 Desarrollo de entornos virtuales en Unity 3D

Unity 3D es un motor de desarrollo multiplataforma ampliamente utilizado para simulaciones interactivas, realidad virtual y entornos industriales. Permite integrar modelos 3D, texturas, animaciones, lógica de comportamiento y comunicación con sistemas externos mediante programación en C# (Unity Technologies, 2024a).

Varios proyectos han demostrado su aplicabilidad para simulaciones industriales, control de procesos y validación de sistemas de automatización (Jiao et al., 2018; Villarroel et al., 2023).

En este proyecto, Unity se utilizó como base para la creación de una planta virtual que replica una línea de ensamble de chumaceras. Los modelos fueron diseñados

en SolidWorks y optimizados en Blender para garantizar un rendimiento fluido en tiempo real.

El motor permitió implementar la interacción entre sensores y actuadores virtuales mediante componentes como RigidBody, BoxCollider y los métodos OnTriggerEnter, OnTriggerStay y OnTriggerExit, además de manejar la comunicación con el PLC utilizando clases de red como TcpClient y NetworkStream (Unity Technologies, 2024b).

2.6 Shaders y representación visual en entornos virtuales

Los shaders son programas que se ejecutan en la unidad de procesamiento gráfico (GPU) y determinan la forma en que los objetos son renderizados en pantalla. Estos controlan propiedades visuales como la iluminación, el color, las sombras, las reflexiones y la transparencia de los materiales utilizados dentro de un entorno virtual. Su correcto diseño permite mejorar la calidad visual de una escena y optimizar el rendimiento general de la aplicación (Unity Technologies, 2024c).

En Unity, los shaders pueden desarrollarse mediante código o utilizando el entorno visual Shader Graph, el cual permite construir y editar shaders de forma modular mediante nodos que representan operaciones matemáticas y gráficas. Esta herramienta facilita la creación de materiales complejos sin necesidad de programación directa, permitiendo ajustar propiedades como metalicidad, suavidad, normal maps y efectos de iluminación en tiempo real (Unity Technologies, 2024d).

En este proyecto, los shaders personalizados fueron utilizados para representar de forma realista algunos componentes de la planta virtual,

2.7 Modelado y optimización en SolidWorks y Blender

El diseño mecánico y estructural de los modelos virtuales se desarrolló en SolidWorks, un software CAD que permite crear geometrías paramétricas, ensamblajes y análisis de movimiento aplicados a ingeniería (Dassault Systèmes, 2024).

Posteriormente, los modelos fueron exportados a Blender, una herramienta de modelado, animación y renderizado 3D de código abierto utilizada para la optimización de mallas, texturizado y UV mapping (Blender Foundation, 2024).

Este flujo de trabajo permitió reducir la cantidad de polígonos y materiales, logrando un mejor rendimiento en Unity. La correcta preparación de los modelos en Blender es fundamental para garantizar una simulación fluida, especialmente en aplicaciones donde intervienen sistemas de física y comunicación en tiempo real (Khan, Choi & Hong, 2022).

2.8 Simulación de sensores y actuadores

En los sistemas automatizados, los sensores proporcionan información sobre el estado del proceso (posición, presencia, velocidad, temperatura, etc.), mientras que los actuadores ejecutan las acciones físicas necesarias (movimiento lineal, rotación, transporte o sujeción).

En entornos virtuales, estos elementos pueden reproducirse mediante scripts que imitan su comportamiento. En el caso de este proyecto, los sensores inductivos y ópticos fueron simulados utilizando los métodos de detección de colisiones OnTriggerEnter, OnTriggerExit y OnTriggerStay de Unity (Unity Technologies, 2024b), mientras que los actuadores (bandas transportadoras y ejes de movimiento) se programaron mediante funciones que controlan desplazamiento, rotación y animación.

Este tipo de simulación se ha utilizado en investigaciones recientes para analizar comportamientos dinámicos en entornos virtuales de precisión (Khan et al., 2022).

2.9 Ventajas del uso de plantas virtuales

El uso de plantas virtuales en la enseñanza y validación de la automatización industrial ofrece beneficios significativos:

- Permite probar rutinas de control sin riesgo de dañar equipos reales.
- Facilita la comprensión visual del proceso y el comportamiento de los dispositivos.
- Reduce costos de infraestructura y mantenimiento.
- Posibilita la práctica simultánea de múltiples usuarios en un mismo entorno.
- Favorece la integración de tecnologías digitales con sistemas industriales reales.

Estas características convierten a las plantas virtuales en herramientas efectivas para la formación técnica y el desarrollo de soluciones industriales (Haces-García & Zhu, 2024; Blender Foundation, 2024; Dassault Systèmes, 2024).

2.10 Trabajos relacionados

Diversos proyectos han implementado la simulación de procesos controlados por PLC. En la literatura se reportan ejemplos de plantas virtuales desarrolladas en Unity 3D o Factory I/O, empleadas con fines educativos para el entrenamiento de operadores o la validación de rutinas de control (Villarroel et al., 2023; Jiao et al., 2018).

Sin embargo, la mayoría de estos sistemas presentan limitaciones en cuanto a personalización o compatibilidad con diferentes protocolos industriales.

El presente proyecto propone una alternativa abierta, configurable y adaptable, que integra comunicación directa con un PLC Mitsubishi FX5U mediante MODBUS/TCP y simula de manera fiel la interacción entre los distintos subsistemas de una línea de ensamble.

Capítulo 3

Planteamiento del problema

3.1. Identificación.

Dentro de los entornos educativos, la enseñanza del control y el entrenamiento en automatización requiere el uso de dispositivos físicos, que suelen tener precios elevados, requieren mantenimiento continuo y conllevan tiempo y esfuerzo de planificación e implementación.

Estas limitaciones dificultan la práctica en sistemas de automatización, reduciendo las oportunidades de aprendizaje y el acercamiento a este tipo de equipos para la capacitación en su uso. Además, los entornos de simulación disponibles comercialmente suelen tener precios elevados, compatibilidad con hardware limitado, requerir equipos con hardware moderno y un número limitado de usuarios con acceso a dichas plataformas.

Ante esta situación, se identifica la necesidad de una herramienta que permita simular y controlar un proceso industrial de forma virtual, emulando el comportamiento de los sistemas físicos, y con una comunicación directa con un PLC real, o soluciones virtuales compatibles. Esta herramienta debe permitir ejecutar y validar rutinas de control, simular señales de sensores, reaccionar a señales de control y reproducir un proceso lógico de una planta real con condiciones seguras y repetibles.

3.2. Justificación.

El desarrollo de una planta virtual controlada por un PLC representa una alternativa accesible para la enseñanza de los sistemas de control y automatización, al permitir que los estudiantes interactúen con dispositivos

virtuales con representaciones fieles para comprender su comportamiento sin depender de una estructura física.

Esta propuesta resulta útil en el contexto educativo, donde la disponibilidad de dispositivos físicos es limitada, o se requiere de una gran cantidad de estos para las prácticas o entrenamiento de los estudiantes. La utilización de este entorno virtual permite que varios grupos de trabajo interactúen de forma independiente dentro de una misma aula, optimizando los tiempos de estudio y los recursos disponibles.

La integración de una planta virtual con un PLC, mediante el protocolo MODBUS/TCP, ofrece un sistema estable y configurable, teniendo una interacción en tiempo real deseable para el análisis de componentes físicos.

3.3. Alcance.

El presente proyecto contempla el diseño, desarrollo e implementación de una planta virtual de automatización industria, basada en una línea de ensamble de chumaceras. La planta fue modelada y diseñada en SolidWorks, optimizada en Blender, e implementada en Unity 3D.

El sistema integra la simulación de sensores inductivos, ópticos, de efecto reed, así como actuadores neumáticos, un eje de movimiento, y un sistema de visión artificial. La comunicación entre la planta virtual y el PLC se realiza mediante el protocolo MODBUS/TCP, permitiendo la interacción de variables en tiempo real.

El proyecto abarca:

- El diseño CAD de la planta virtual.

- La optimización de mallas, materiales, y animaciones para lograr una representación, visual y funcional, fiel a los componentes reales propuestos.

- La programación de los comportamientos lógicos en C# dentro del motor Unity 3D.

- La integración y validación funcional del sistema con el PLC.

Quedan fuera del alcance del proyecto la simulación de condiciones físicas avanzadas (fuerzas, fricción, desgaste mecánico), y, debido a esto, la simulación del ensamble de la chumacera también es excluido.

Capítulo 4

Objetivos

4.1. Objetivos generales.

Diseñar y desarrollar una planta virtual en Unity 3D usando como base el modelo CAD de una línea de ensamble de chumaceras, incluyendo la simulación de sensores, actuadores, control de movimiento, y visión artificial; que se comunique con un controlador a través del protocolo MODBUS/TCP.

4.2. Objetivos específicos.

- Diseñar un modelo CAD de la planta.
- Optimizar y acondicionar la malla del modelo 3D en Blender.
- Implementar del modelo 3D de la malla en Unity 3D.
- Desarrollar materiales y animaciones para representar la apariencia y comportamiento de los materiales utilizados en la planta virtual.
- Simular sensores inductivos, ópticos, y de efecto Reed; así como actuadores neumáticos, motores, y un sistema de inspección mediante programación en C# e interacciones del sistema de físicas de Unity 3D
- Implementar el protocolo de comunicación MODBUS/TCP entre la planta virtual y el controlador.
- Validar el comportamiento de la planta virtual mediante la ejecución de rutinas de prueba en GX Woks 3.

Capítulo 5

Metodología

El presente capítulo detalla el proceso de diseño, desarrollo, e implementación que se utilizó para elaborar una planta virtual para entrenamiento en automatización industrial.

La metodología se estructuró en X cantidad de fases, priorizando la simulación de sistemas físicos en un entorno virtual.

5.1 Diseño de la planta

Sensores utilizados en la planta:

Se utilizaron 3 tipos de sensores para el funcionamiento de la planta, siendo estos los sensores inductivos PR18-5DP, utilizados para el control del movimiento del eje de la planta, es decir, como sensores de límites de movimiento, y como guías para la posición cero del eje; los sensores ópticos CNY70, utilizados para detectar la presencia de las piezas en distintas zonas del recorrido de estas; y los sensores de efecto Reed D-Z73, utilizados para conocer el estado de los cilindros neumáticos utilizados.

Actuadores utilizados en la planta:

Se utilizaron los cilindros neumáticos TDA10-10 y MHZ2-20D para que el eje tuviera dos tipos de movimiento; y los cilindros MHC2-20D, CDG1BN32-50, y CDO2L20-150DMZ para el proceso de ensamble.

Se utilizaron además un motor a pasos Nema 17 para el movimiento del eje, seleccionado por ofrecer un par adecuado para la masa del motor, y motorreductores rectos 48:1 de 5V para el control de las bandas transportadoras.

Se utilizó la cámara Banner Presence Plus P4 Area 1.3, por su capacidad de procesamiento de imágenes autónomo (programable) y su interfaz dedicada de

entradas y salidas para una comunicación directa con el sistema de control. Se utilizó un objetivo de 16mm F1.4 montado en la cámara.

Se utilizaron además los rodamientos ABEC-9 como materia prima para el ensamble, y los rodamientos lineales SC8UU para el movimiento del eje.

Materiales adicionales:

Se utilizó perfil de aluminio de 30x30mm para la estructura que sostiene al eje, y perfil de aluminio 20x20 para las estructuras adicionales. Además, se utilizó filamento PLA para la impresión de las piezas estructurales y funcionales de la planta en general, esto, utilizando una impresora FDM (Modelado por Deposición Fundida) Voxelab Aquila X2.

5.2 Diseño CAD de la planta

Se propuso el diseño de una planta ensambladora de chumaceras con una estructura IN-Line, de modo que fuera relativamente simple de comprender tanto en funcionamiento, como en flujo de trabajo.

El proceso en el cual se basó la planta se muestra en el diagrama de flujo de la figura 1.

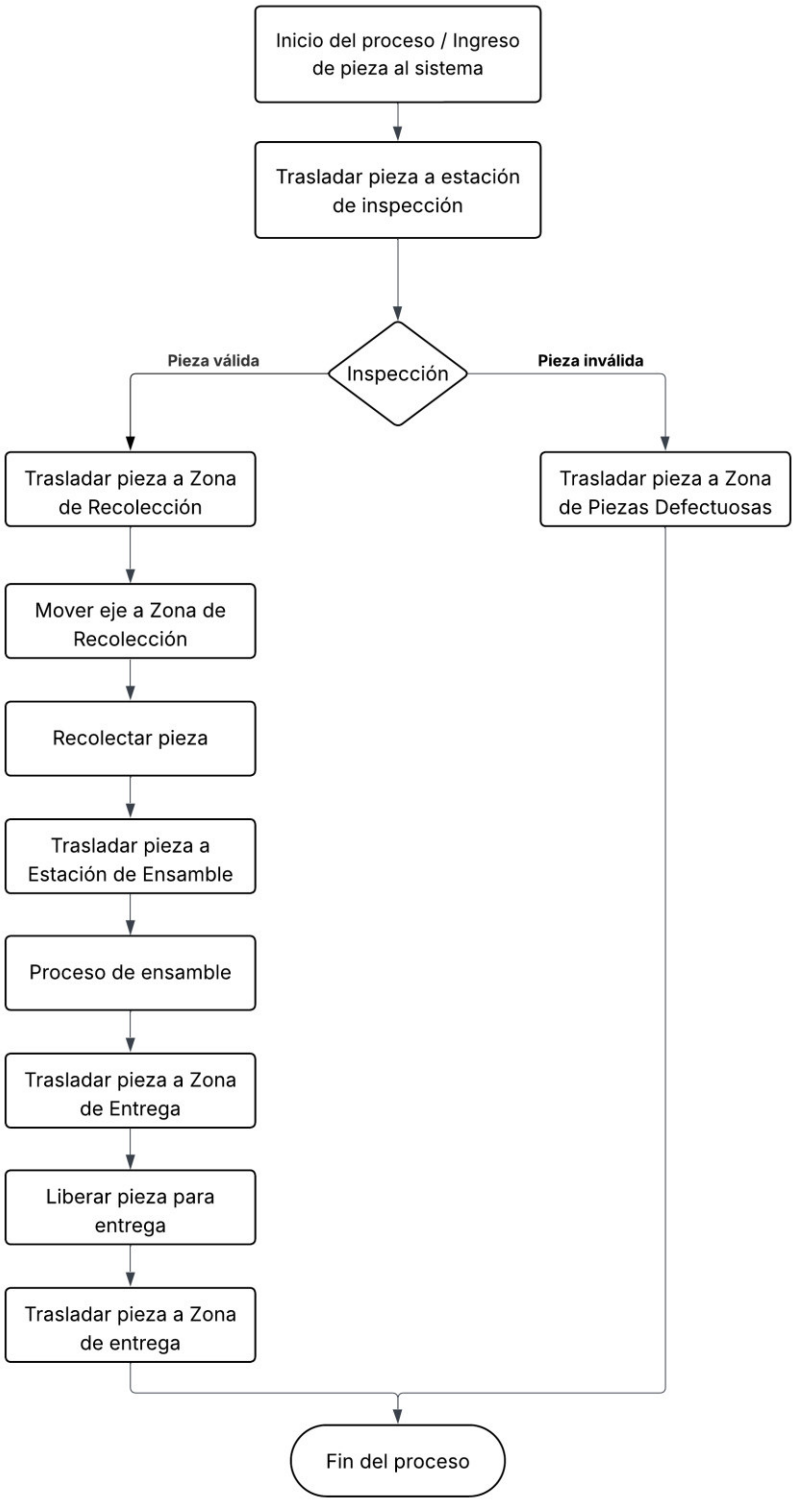

Figura 1: Diagrama de flujo del proceso propuesto para la planta diseñada.

El diseño CAD de la planta se realizó en SolidWorks de modo que cumpliera con el proceso propuesto, para lo cual, se realizó el diseño de las bandas transportadoras, basado en un sistema de piñones y cadena. El diseño de los eslabones se ejecutó para utilizar pernos de 3mm para unirlos, estos eslabones se muestran en la figura 2. El primer piñón se diseñó para ser colocado sobre la flecha de un motorreductor recto 48:1 de 5V, y el segundo para montarse sobre un rodamiento ABEC-9, estos diseños también son mostrados en la figura 2.

Una vez ensamblados, el sistema de la banda transportadora tiene el aspecto mostrado en la figura 3.

Posteriormente, se añadieron soportes para que la banda transportadora pudiera sostenerse del piso, utilizando perfil de aluminio de 20x20mm. El diseño final de la banda transportadora se muestra en la figura 4.

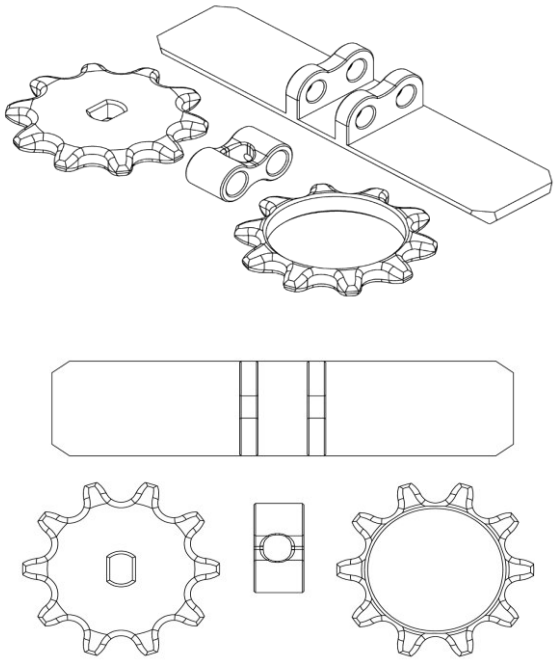

Figura 2: Diseño de eslabones y piñones para banda transportadoras.

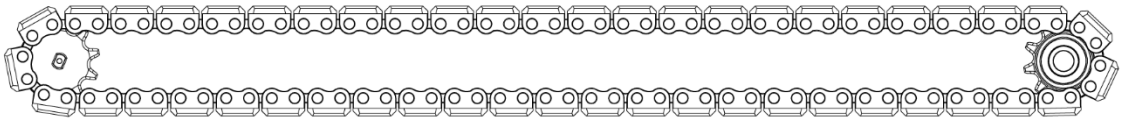

Figura 3: Sistema Banda transportadora.

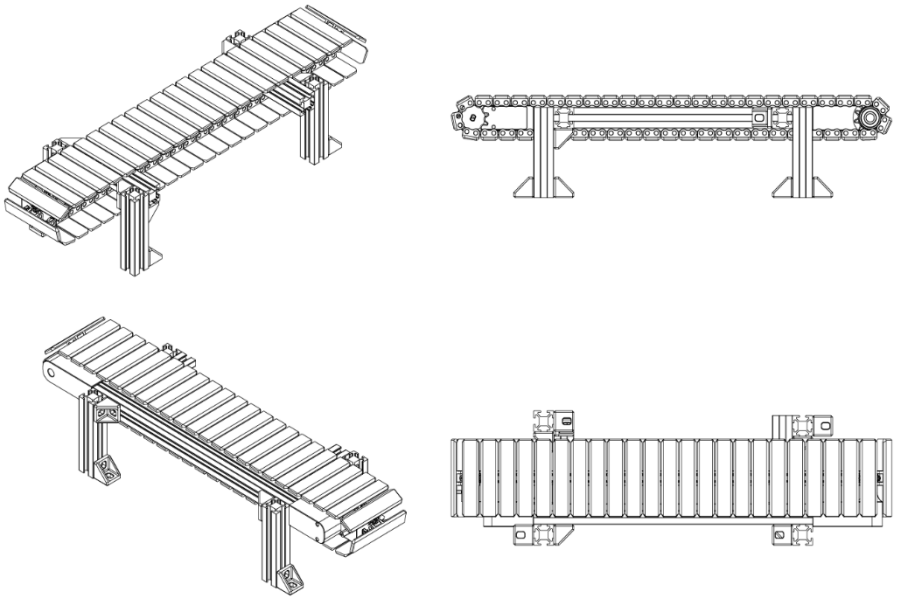

Figura 4: Diseñó final de banda transportadora.

Se diseñó adicionalmente un sistema de alimentación de rodamientos para ensamble. Este sistema cuenta con un depósito donde se almacenan los rodamientos en espera. El primero de estos es impulsado por el vástago del cilindro neumático BBB, haciendo que se deslice a lo largo de un carril. Este carril dirige al rodamiento hasta un orificio de posicionamiento, que lo alinea correctamente para su integración en el ensamble. Este sistema se muestra en la figura 5.

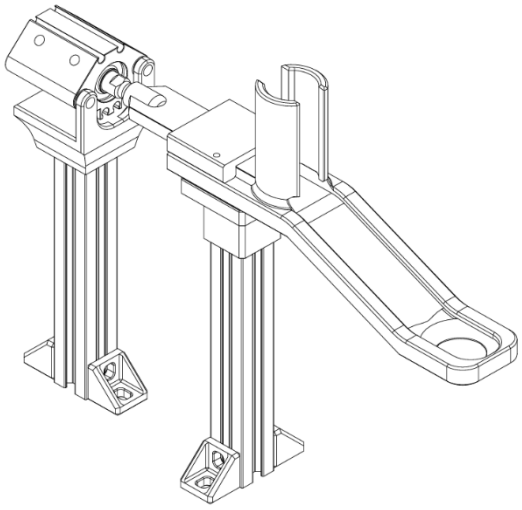

Figura 5: Sistema de alimentación de rodamientos para el ensamble.

El diseño CAD completo de la planta se muestra en la figura 6, donde se destacan los subsistemas:

1.- Banda transportadora 1.

2.- Estación de Inspección.

3.- Manipulador

4.- Contenedor de piezas defectuosas.

5.- Estación de ensamble.

6.- Banda transportadora 2.

La ubicación de estos subsistemas se muestra en la figura 7.

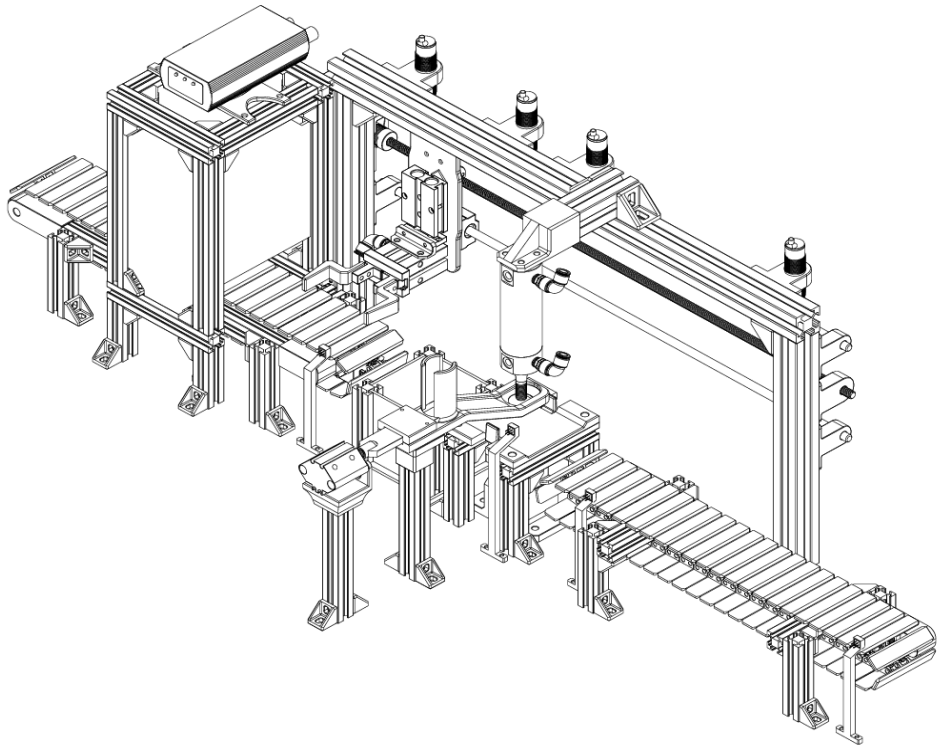

Figura 6: Diseño CAD de la planta.

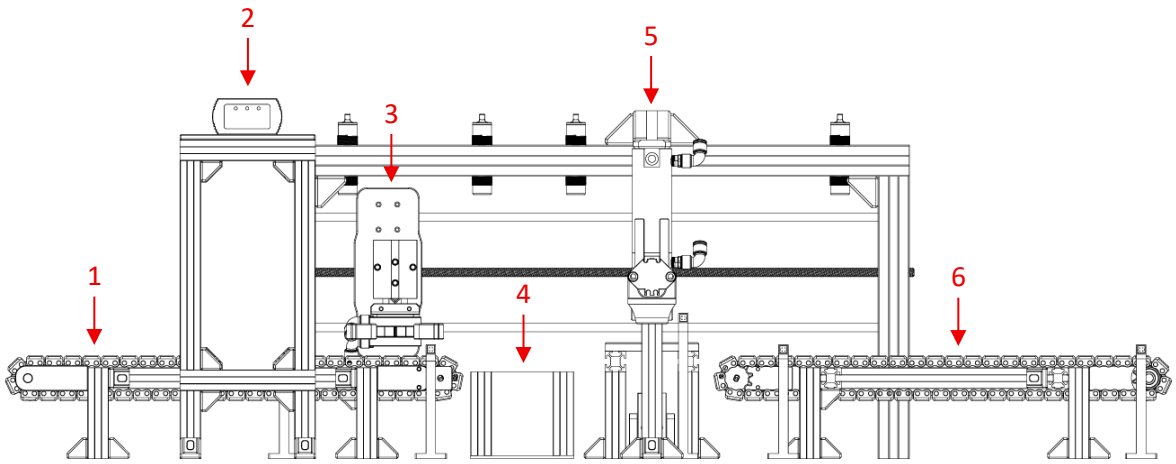

Figura 7: Ubicación de Subsistemas en la planta.

5.3 Ubicación de los sensores en la planta.

En la tabla 1 se estipulan la cantidad, tipo, y función de cada sensor, adicionalmente, en las figuras 8, 9, y 10, se muestra su ubicación en la planta.

Nombre	Tipo	Función
SI 1	Inductivo	Límite de movimiento del eje.
SI 2	Inductivo	posición cercana al origen.
SI 3	Inductivo	Posición cero del eje.
SI 4	Inductivo	Límite de movimiento del eje.
SO 1	Óptico	Pieza en posición de inspección.
SO 2	Óptico	Pieza en posición de recolección.
SO 3	Óptico	Pieza en posición de ensamble.
SO 4	Óptico	Pieza en inicio banda 2.
SO 5	Óptico	Pieza en posición de entrega (final).
SR 1	Reed	Vástago del cilindro TDA10-10 extendido.
SR 2	Reed	Vástago del cilindro TDA10-10 retraído.
SR 3	Reed	Vástago del cilindro MHZ2-20D extendido.
SR 4	Reed	Vástago del cilindro MHZ2-20D retraído.
SR 5	Reed	Vástago del cilindro MHC2-20D extendido.
SR 6	Reed	Vástago del cilindro MHC2-20D retraído.
SR 7	Reed	Vástago del cilindro CDG1BN32-50 extendido.
SR 8	Reed	Vástago del cilindro CDG1BN32-50 retraído.
SR 9	Reed	Vástago del cilindro CDO2L20-150DMZ extendido.
SR 10	Reed	Vástago del cilindro CDO2L20-150DMZ retraído.

Tabla 1: Nombre y descripción de los sensores utilizados en la planta.

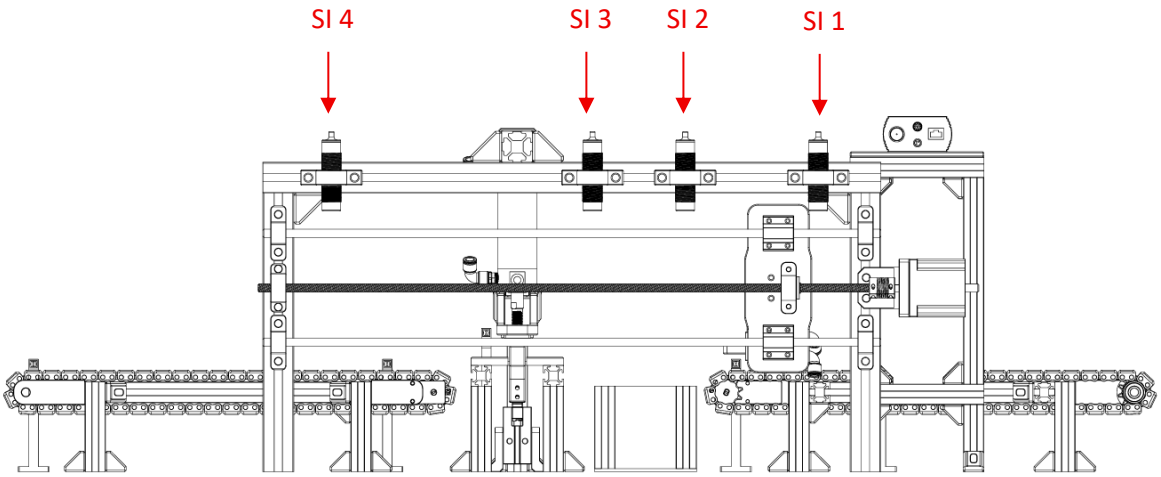

Figura 8: Ubicación de los sensores inductivos en la planta.

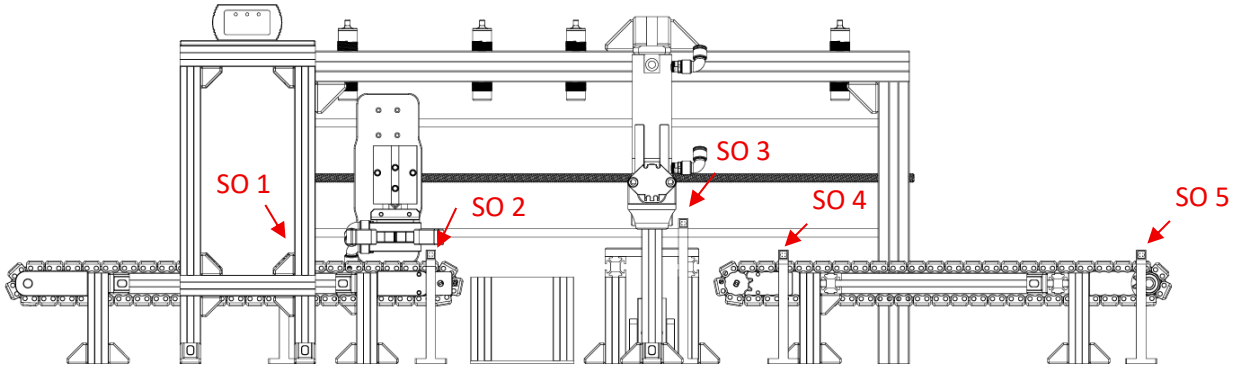

Figura 9: Ubicación de los sensores ópticos en la planta.

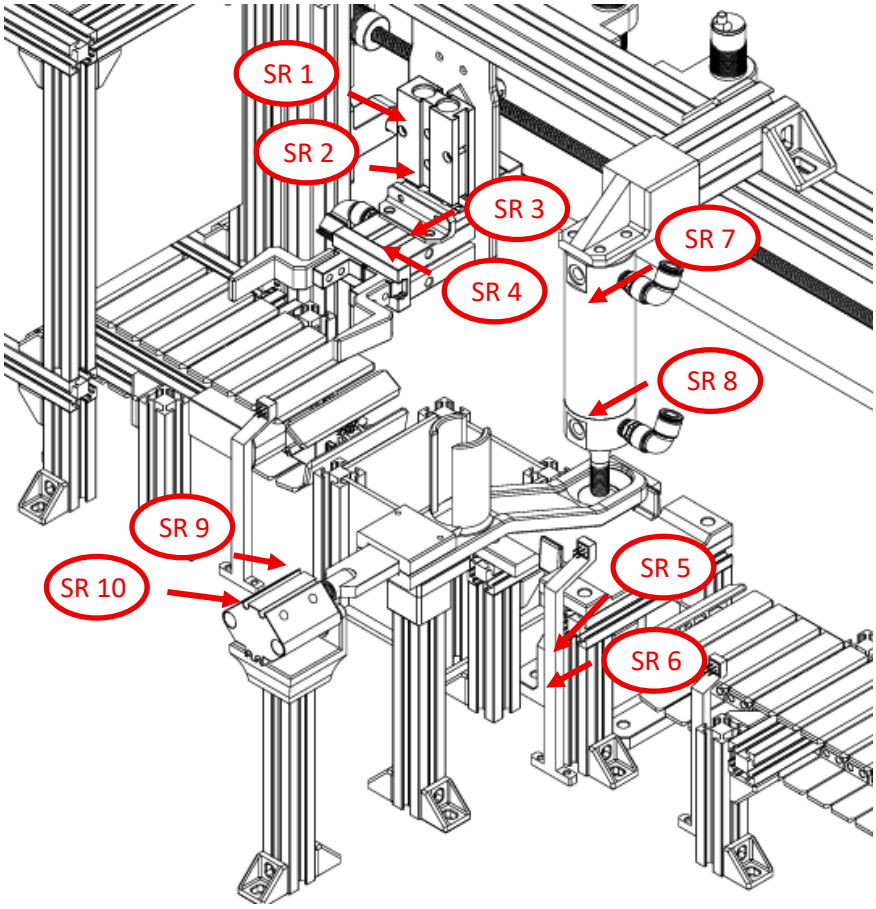

Figura 10: Ubicación de los sensores de efecto Reed en la planta.

El modelo CAD 3D de la planta virtual no puede exportarse en un formato compatible con Unity 3D directamente desde Solidworks, por lo que se utiliza el software Blender como intermediario, para, además, se realizará en él un proceso de optimización de la malla.

5.4 Diseño de la chumacera para ensamble

El diseño propuesto para las chumaceras utilizadas durante el proceso de ensamble se trata de una chumacera cuadrada, cuyas dimensiones son 70x70x13mm. Con un orificio de 22 milímetros para el ensamble de un rodamiento ABEC-9 en él.

Estas dimensiones se definieron en base a las capacidades de movimiento de los actuadores neumáticos.

El diseño CAD de la chumacera se muestra en la figura 11.

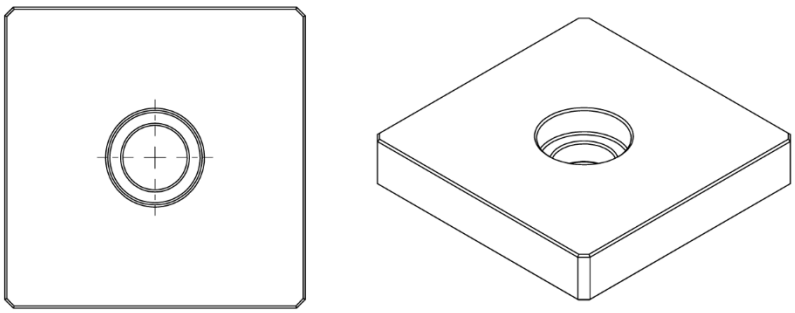

Figura 11: Diseño CAD de las chumaceras utilizadas por la planta.

Tras el ensamble del rodamiento, la apariencia de la chumacera adquiere el diseño mostrado en la figura 12.

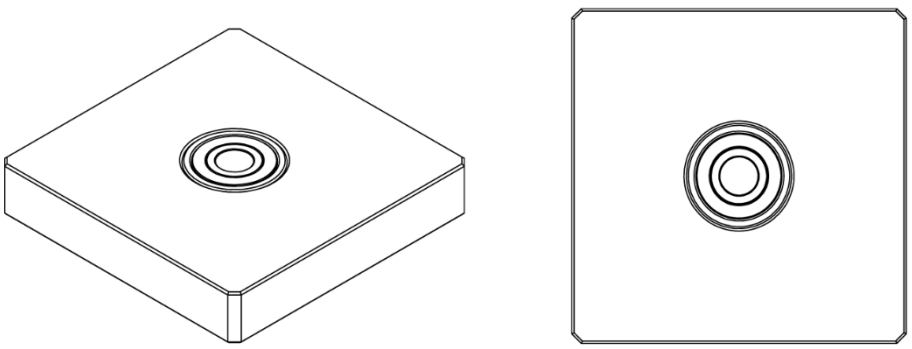

Figura 12: Apariencia de la chumacera después del proceso de ensamble.

5.5 Método de control

El control de la planta virtual, así como las pruebas de funcionamiento, se realizarán utilizando el PLC Mitsubishi FX5U, con contar con un módulo de comunicación Ethernet integrado, y por contar con salidas de alta velocidad compatibles con los módulos de control de movimiento.

5.6 Exportación y optimización de la malla

Reducción de detalles en la malla de la planta

Antes de realizar la exportación a Blender se realiza un proceso anterior, con la finalidad de tener una malla más ligera antes de llevar a cabo el proceso de reducción de vértices.

Esta optimización consiste en la realizar de una copia del ensamble con un nivel de detalle menor, para lo cual se eliminaron los redondeos y chaflanes de las esquinas de los modelos, se eliminan además las roscas de la mayoría de los modelos. La figura 13 muestra un ejemplo del resultado de este proceso de optimización.

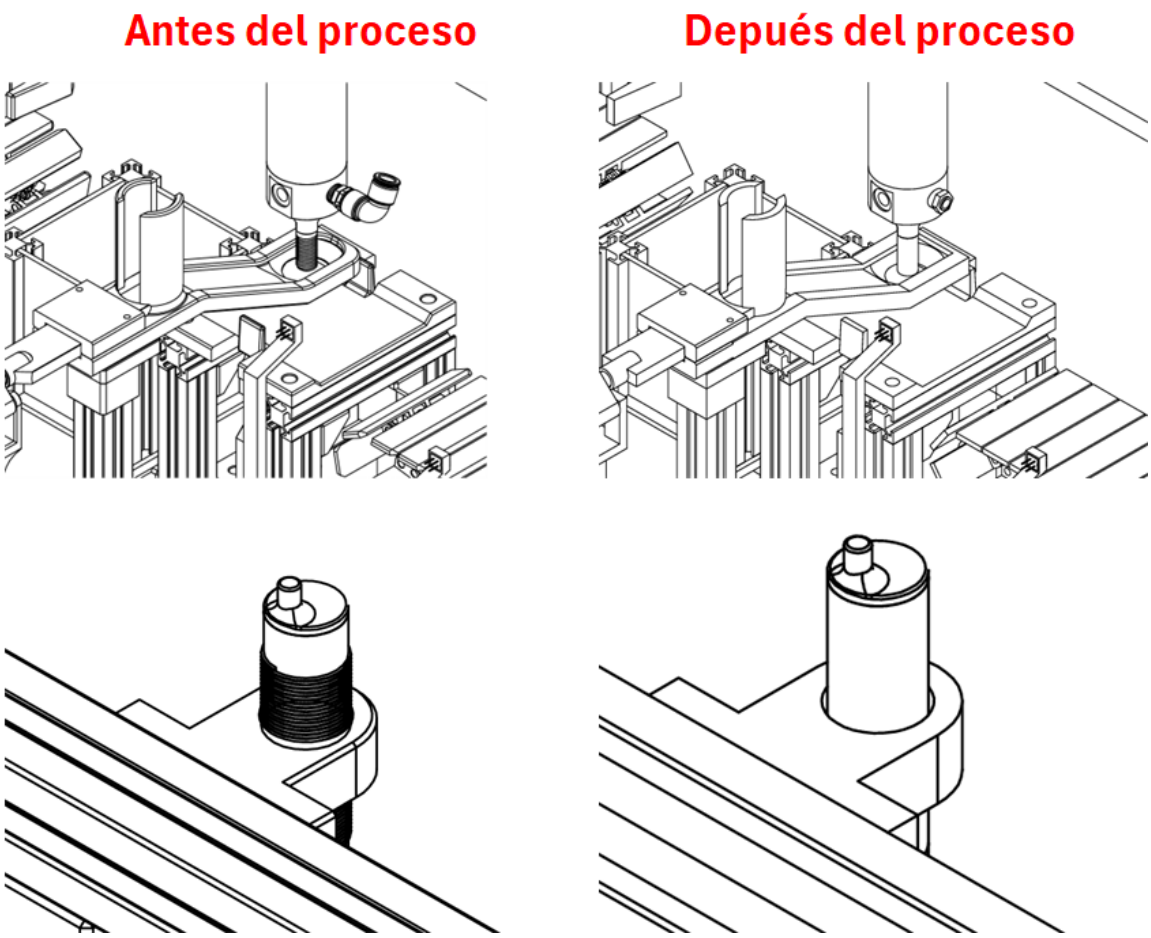

Figura 13: Ejemplos del resultado del primer proceso de optimización.

Esta malla será la que se utilizará para todos los procesos futuros.

Importación de la malla optimizada a Blender

Solidworks permite exportar el ensamble (y cualquier pieza diseñada) de la planta en el formato .stl, formato que puede ser importado a Blender de forma nativa.

Una vez importado el modelo CAD dentro de Blender, se observa la cantidad original de vértices que tiene la planta, mostrada dentro de la ventana gráfica en la figura 14.

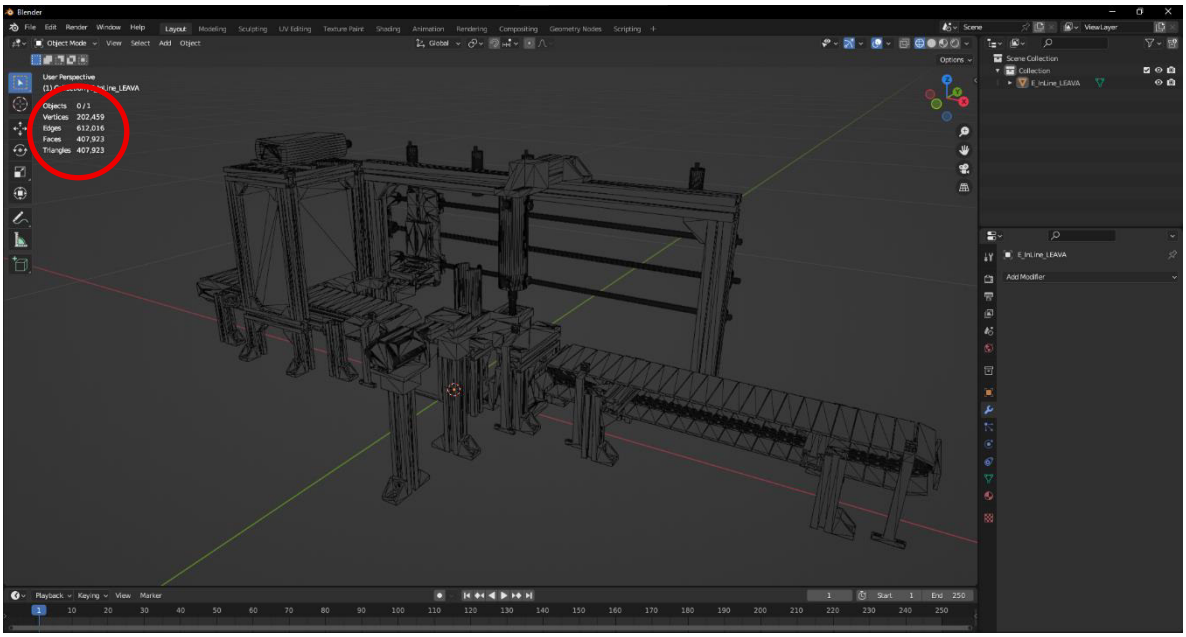

Figura 14: Cantidad de polígonos del modelo CAD de la planta, importado de SolidWorks a Blender.

La cantidad original de vértices es 202,459, alto para los dispositivos objetivo de nuestra aplicación, por lo que se aplica un segundo proceso de optimización de la malla.

Como segundo proceso de optimización, se realiza un proceso de diezmado, que, a grandes rasgos, busca reducir de forma significativa la cantidad de vértices de una malla 3D.

Para llevar a cabo este proceso se le aplica el modificador “Decimate”. La figura 15 muestra la diferencia de vértices una vez que este modificador es aplicado.

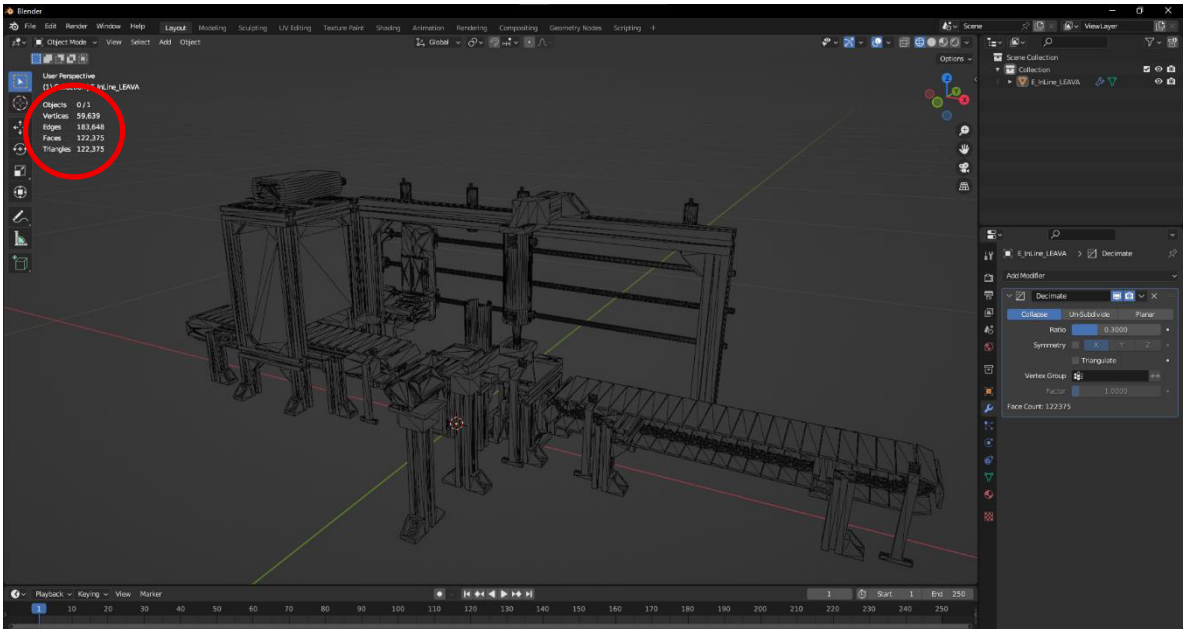

Figura 15: Modificador "Decimate" aplicado a la malla del modelo 3D de la planta, dentro de Blender.

Como resultado de este proceso, la cantidad actual de vértices es de 59,639, una cantidad que la mayoría de dispositivos pueden manejar.

Asignación de materiales en Blender.

Para tener una mejor representación de la planta virtual, se aplican materiales que permiten representar el color y la textura de los diferentes materiales de los diferentes componentes utilizados. Sin embargo, el uso de una gran cantidad de materiales en el motor Unity 3D resulta en un costo alto de recursos computacionales, y, por consecuencia, en el rendimiento de la aplicación, por lo que se utilizó una técnica llama “Atlas de texturas”, que consiste en la utilización de un solo material para todos (o un número alto) los objetos en una escena de Unity 3D. La planta virtual utiliza dos tipos de materiales en la escena: un material metálico, y un material plástico – rugoso.

Adicionalmente a esta técnica, se utilizó una sola textura PNG de 10x1 pixeles, con los colores que se utilizan en los componentes de la planta. Para signar la textura, se utiliza un nodo “Image texture” con la textura “textureAtlas” conectada a la entrada “Base color” del nodo “Principed BSDF”. Este nodo se conecta a “Material output” para poder mostrarse en la malla, y con esto, la textura estará asignada al material. Este proceso se muestra en la figura 16, y la textura “textureAtlas” se muestra en la figura 17.

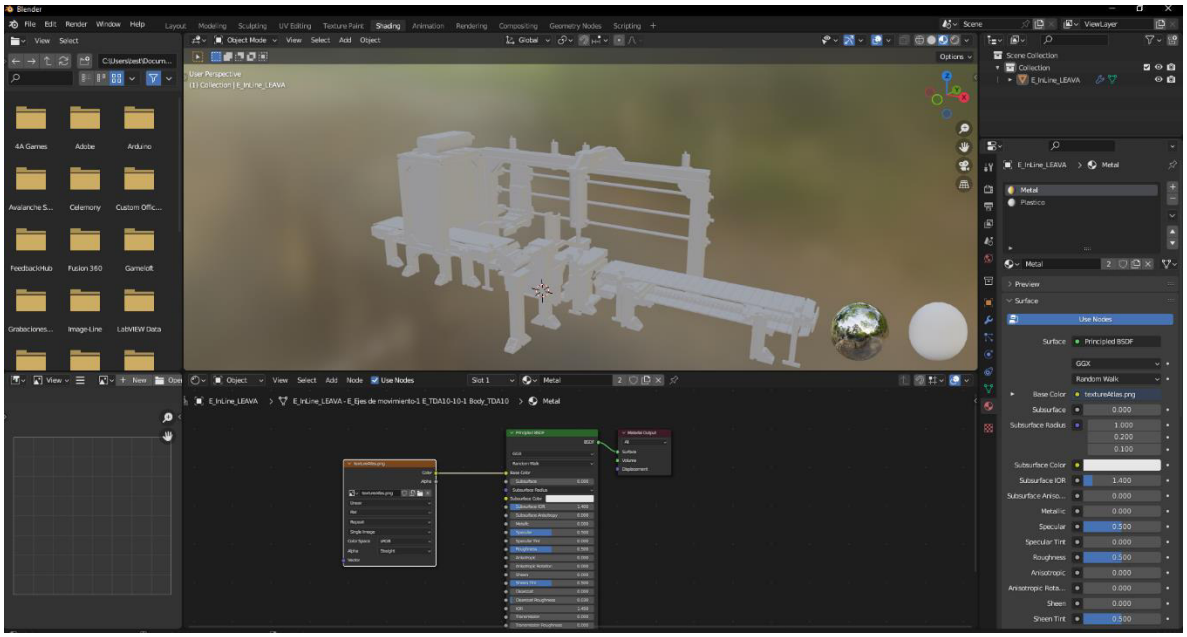

Figura 16: Asignación de la textura "textureAtlas" a un meterial en Unity 3D.

Figura 17: Textura "textureAtlas", escalada para su mejor visualización.

Este proceso se repite para el material “Plástico”. En el material “Metal” se modifica el valor “Metalic” del nodo “Principal BSDF” a 1, para que simular un material metálico. Esto se muestra en la figura 18, donde también se aprecia el cambio en la visualización del material debido a este cambio de parámetro.

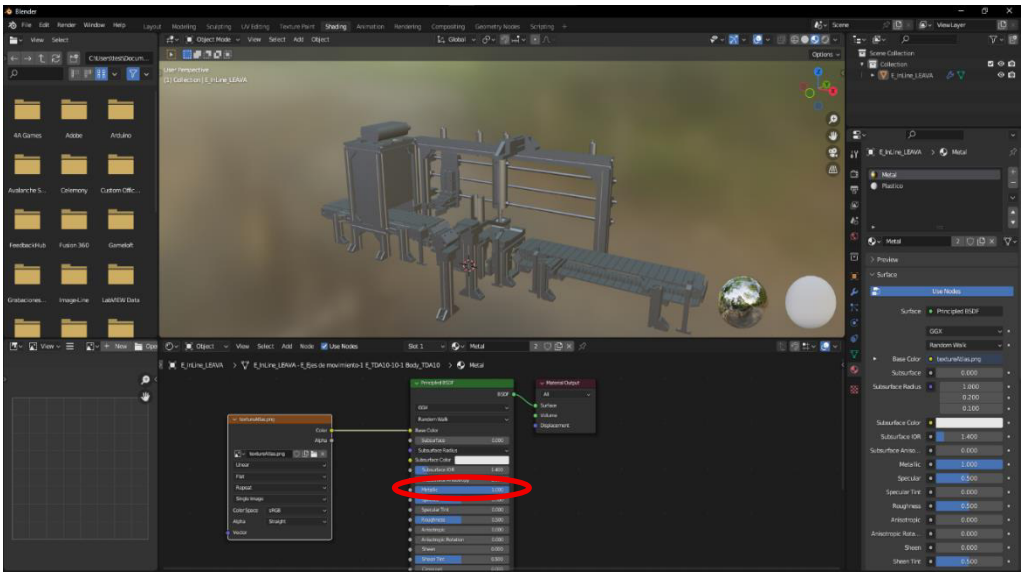

Figura 18: Cambio del parámetro "Metalic" del material metálico a 1.

Al tratarse de un solo material para múltiples objetos, el proceso para asignar un "color" de la textura a un objeto se realiza en la ventana "UV Editing".

Las UV's es una proyección de las caras de un modelo 3D en una textura 2D, esta proyección se escala y se coloca sobre el color de la textura en la ventana "UV editing". Este proceso se muestra en las figuras 19, y 20.

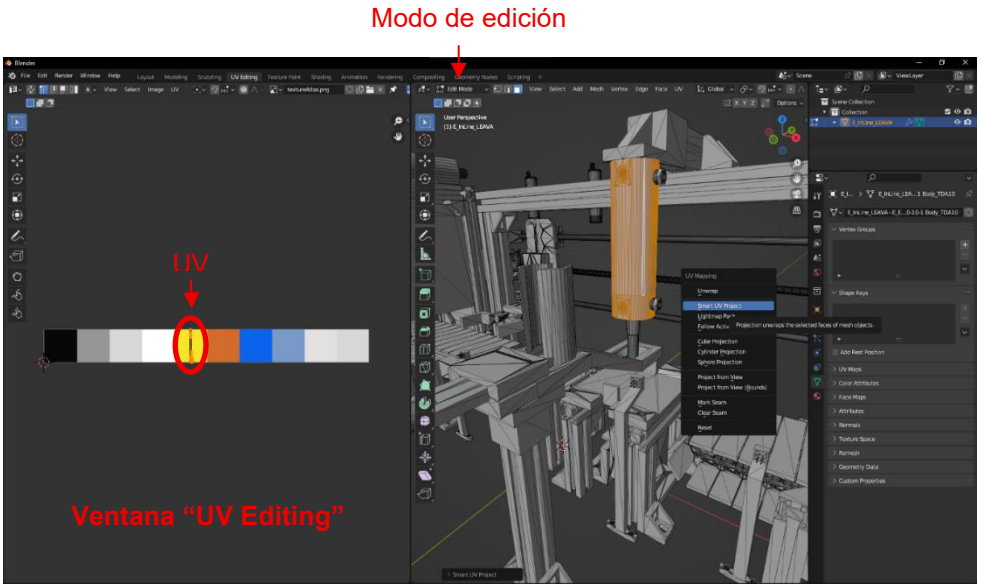

Figura 19: Despliegue de UV's de una malla en Blender.

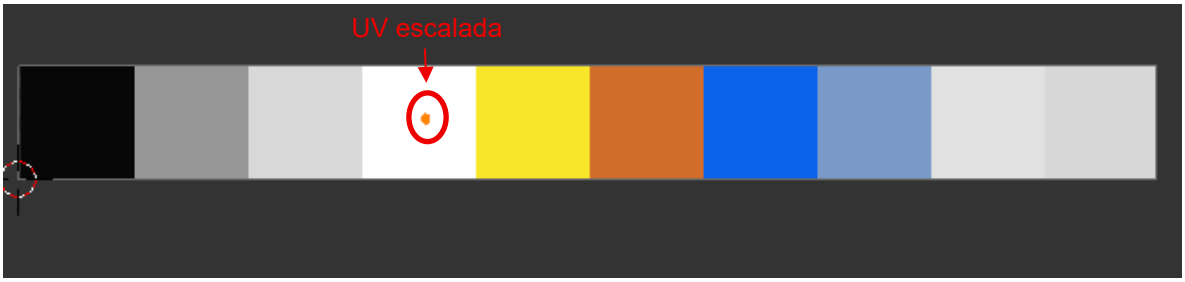

Figura 20: Escalado y signación del color de un UV en Blender.

El cambio de color en la planta, resultado de la asignación de color mostrado, se muestra en la figura 21.

Este proceso se repite para cada uno de las mallas restantes, tanto para el material metálico, como para el material plástico.

Se eligió el color amarillo para representar las piezas diseñadas para imprimirse en 3D.

Se añadieron 3 materiales adicionales para representar de manera fiel los indicadores de la cámara Banner Presence Plus P4 Area 1.3, siendo estos: LED Rojo, LED Verde, y LED Amarillo. Estos materiales se signan a la malla de los indicadores led de la cámara en el modelo 3D.

La visualización de la planta con todos los materiales aplicados se muestra en la figura 22.

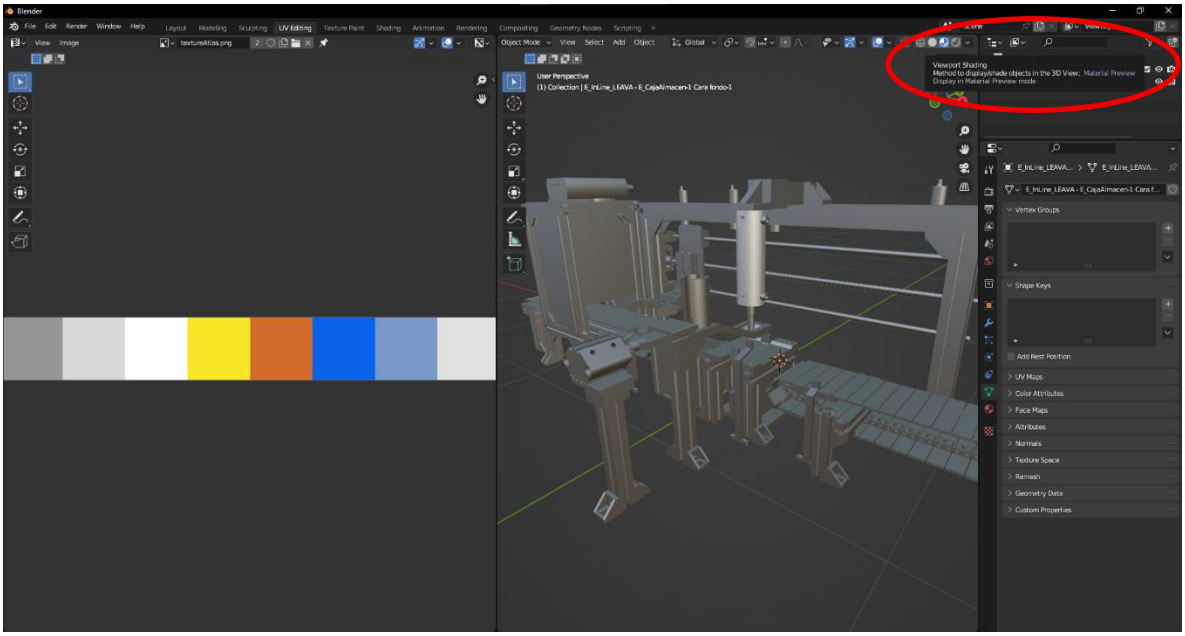

Figura 21: Visualización del color del material asignado a una malla en Blender.

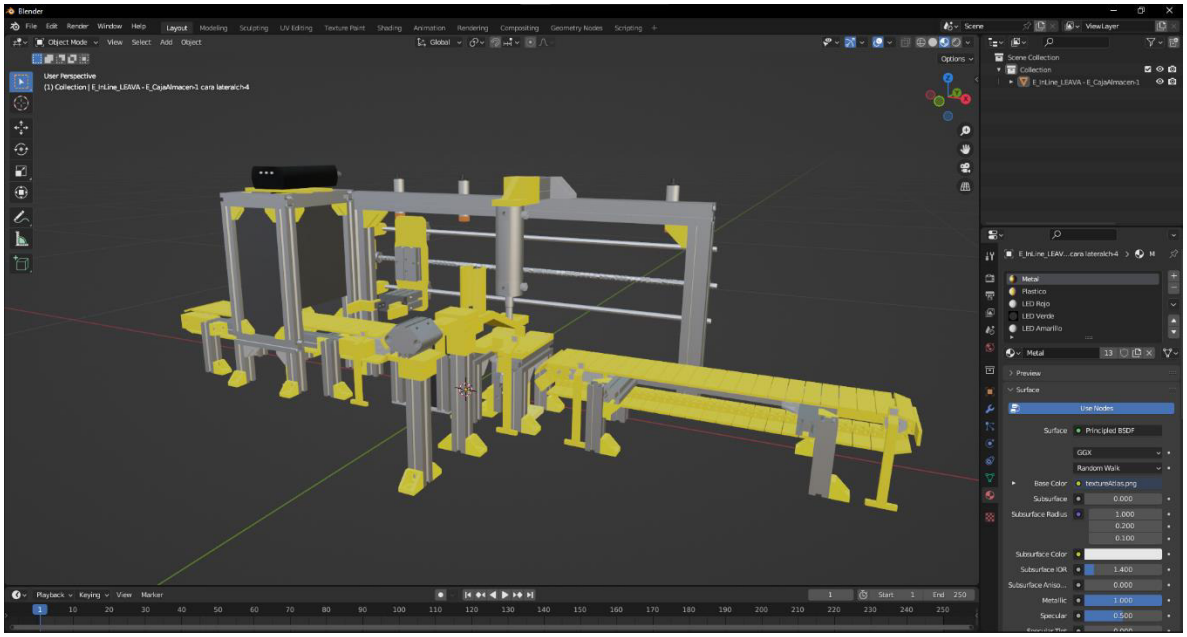

Figura 22: Visualización de la asignación de materiales en toda la malla de la planta en Blender.

5.7 Animaciones en Blender

Animación de los actuadores no neumáticos

La simulación del movimiento de los actuadores neumáticos se realiza mediante animaciones de la malla, utilizando Key Frames de un esqueleto de animación, siguiendo el rango de movimiento de los actuadores reales. Para esto, la malla de cada actuador se separa y guarda en archivos independientes para: Manipulador (actuador TDA10-10), el actuador MHZ2-20D, las pinzas del actuador MHC2-20D, el vástago del actuador CDG1BN32-50, y el vástago del actuador CDO2L20-150DMZ.

A demás, para simular el movimiento de la flecha del motor Nema 17, que transfiere su movimiento a una varilla dentada de 8mm, también se separa una malla en un archivo individual.

De este modo, se generan 7 archivos .blend que se trabajarán individualmente.

A continuación, se utiliza el archivo que contiene la malla del actuador MHZ2-20D para la demostración del proceso de animación.

Una vez que las mallas que componen al actuador están en un archivo independiente, se agrupan las mallas de forma que los tres componentes principales de este actuador son individuales: El cuerpo del actuador, su vástago, que, en este caso, corresponde a dos dedos de la pinza, cada uno, en su propia malla. Esta agrupación se muestra en la figura 23.

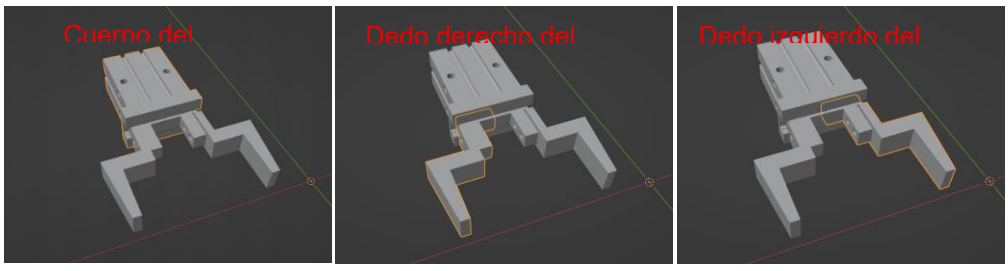

Figura 23: Agrupación por partes individuales del actuador.

Para añadir animaciones al actuador, se hace uso de los esqueletos de animación, para esto, se añade un nuevo hueso, que será el “hueso padre”, se posiciona de modo que coincida con el cuerpo del actuador, y en la ventana “Propiedades del hueso” se le asigna el nombre de “hueso padre” para tenerlo identificado, este proceso se muestra en la figura 24.

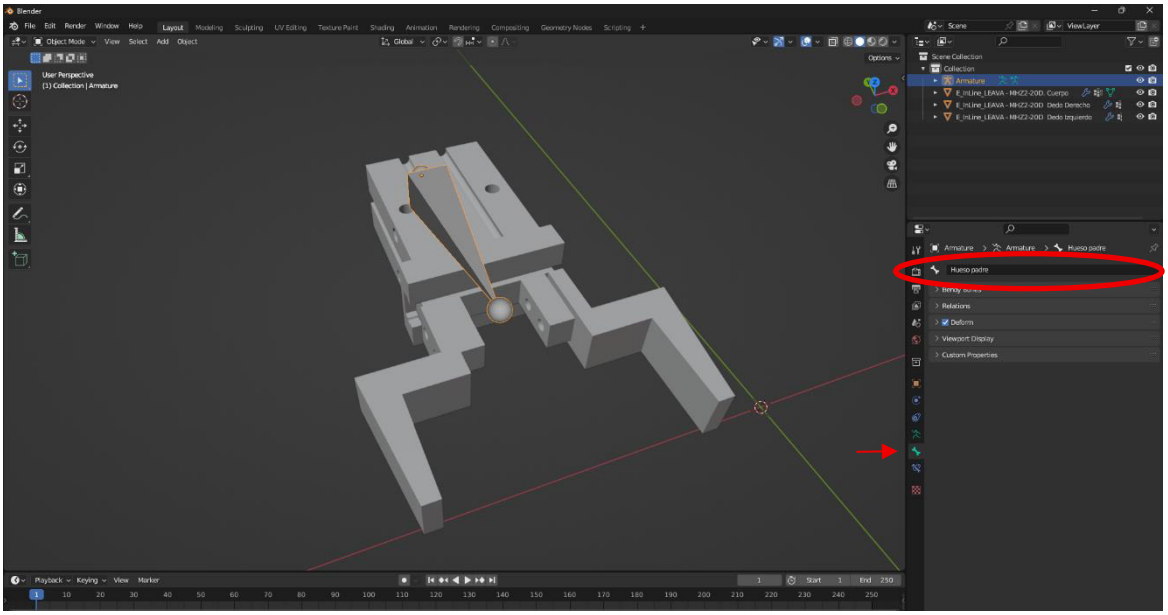

Figura 24: Alineación de hueso padre con el cuerpo del actuador neumático y su nombramiento para identificación.

Para el control de los dedos, se añade un nuevo hueso, alineado con el dedo derecho. Esta vez, el nombre se asigna como “Dedo.R”, como se aprecia en la figura 25. Posteriormente, utilizamos la opción “Symmetrize”. Para generar un hueso simétrico a “Dedo.R” llamado “Dedo.L.”. La creación del hueso “Dedo.L” se aprecia en la figura 26.

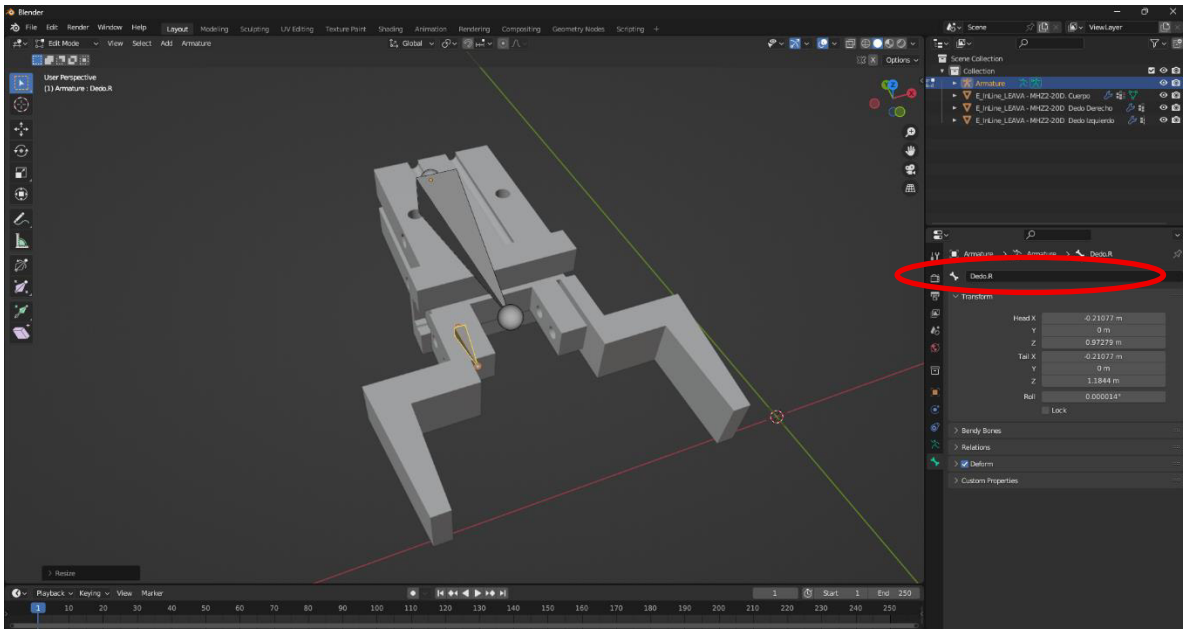

Figura 25: Creación y alineación del hueso "Dedo.R" en el esqueleto de animación.

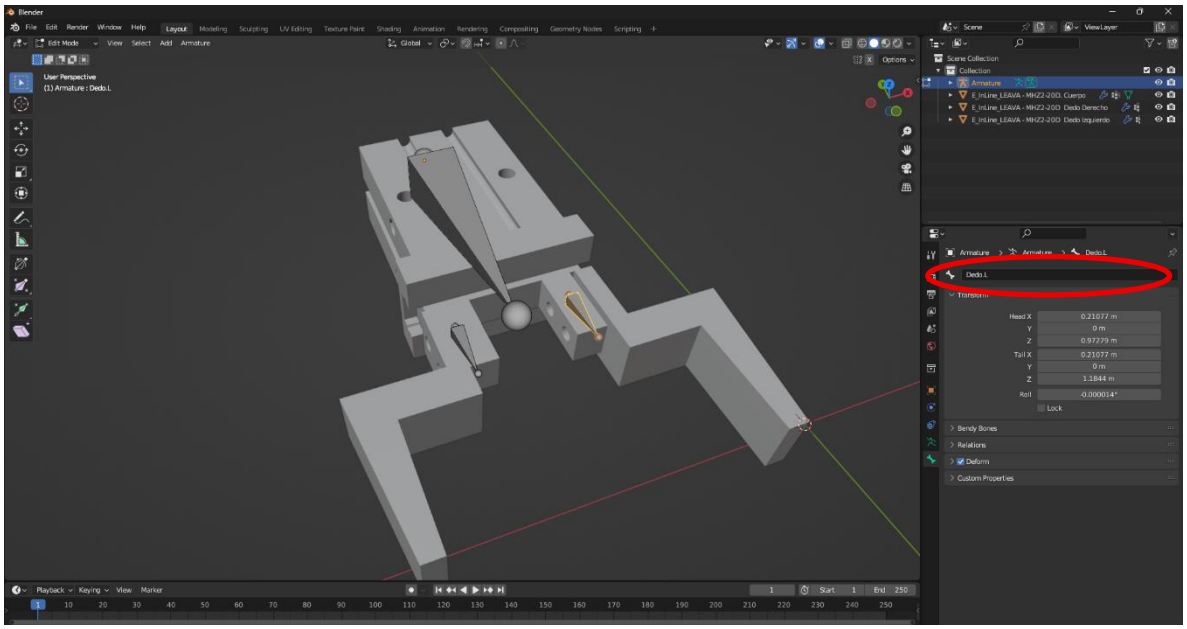

Figura 26: Creación del hueso "Dedo.L" a partir de la simetría del esqueleto de animación.

Con esto hecho, el esqueleto de animación está completo, sin embargo, este funciona como un objeto independiente, y si se desplaza (en el modo Pose), se aprecia que este no tiene efecto sobre la malla del actuador neumático, esto se muestra en la figura 27.

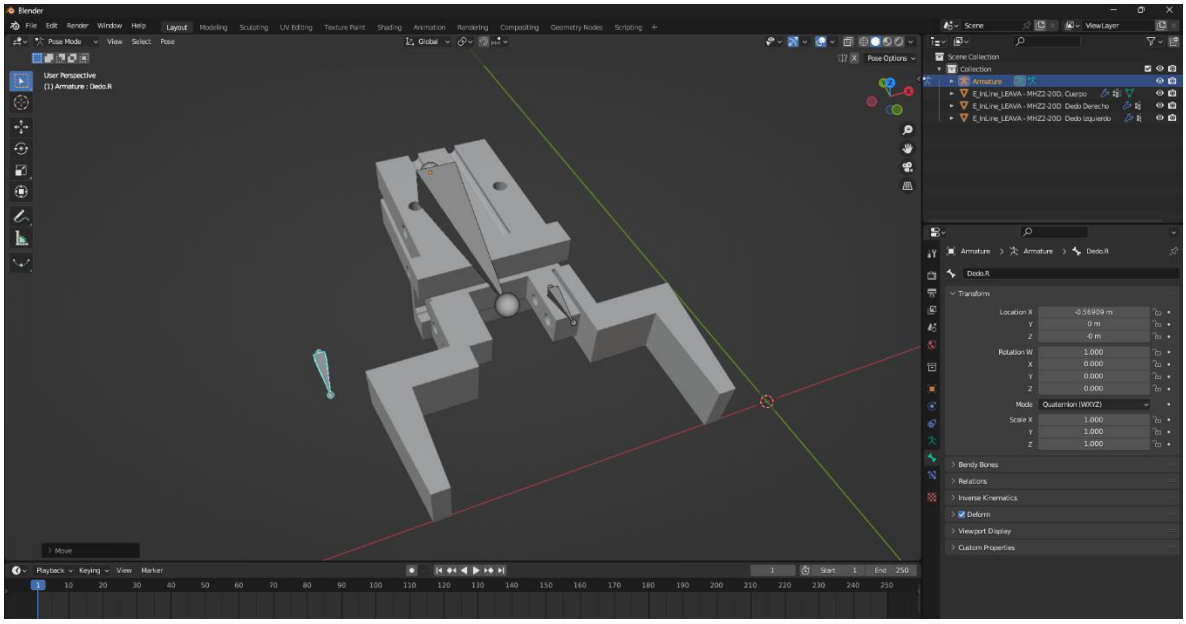

Figura 27: Independecia entre el esqueleto de animación y la malla del actuador neumático.

Para asociar el los huesos con la malla, se lleva un proceso llamado “Emparentar”, mediante la opción “With automatic weights”, que asignará de forma automática la influencia de cada hueso en la malla emparentada.

La influencia de cada hueso sobre la malla, se representa con una escala de 0 a 1, y esto se visualiza sobre la malla con una escala de color que va desde el Azul, que representa nula influencia (0), hasta Rojo, que representa influencia total (1).

Como ejemplo, se muestra este proceso en la malla “Dedo derecho”. En la figura 28 se muestra los pesos automáticos generados.

Figura 28: Influencia automática asignada a la malla emparentada.

Se aprecia que las influencias asignadas automáticamente no son perfectas, pues se tienen influencias de huesos que no deberían afectar a la malla seleccionada, pues, en este caso, sólo se desea la influencia del hueso “Dedo.R”.

Para corregir esto, se lleva a cabo un proceso de “limpiar influencias”, donde se utiliza el modo de interacción “Pintar Influencias” o “Weight Paint”, para seleccionar manualmente la influencia de un determinado hueso en cada vértice de la malla.

Las influencias corregidas de los huesos se muestran en la figura 29.

Figura 29: Influencia de los huesos corregida en la malla.

Con las influencias aplicadas, si ahora desplazamos el hueso “Dedo.R”, en el modo de interacción “Pose Mode” vemos cómo la malla sigue el movimiento de este, como resultado de un correcto emparejamiento con el esqueleto de animación. Esto, además de un ejemplo de influencias mal asignadas, se aprecia en la figura 30.

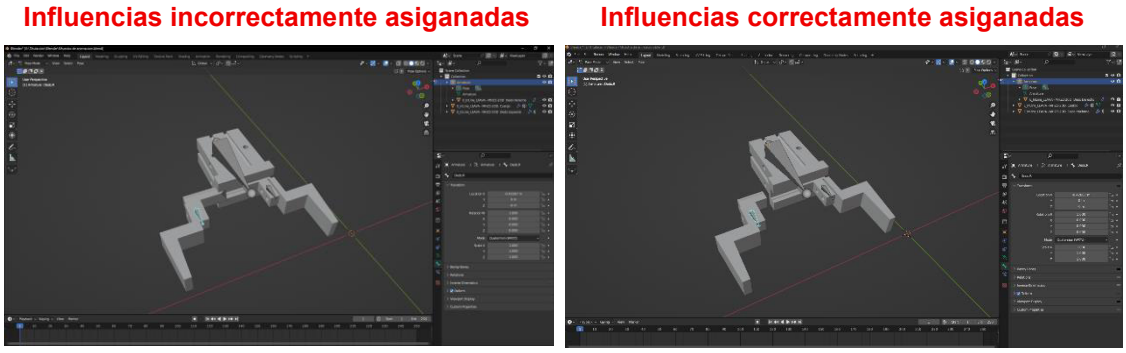

Figura 30: Comparación entre influencias incorrectamente asignadas, y correctamente asignadas.

Para este actuador, se repite este proceso para la malla correspondiente al dedo izquierdo. No es necesario asignar influencias de huesos para el cuerpo del actuador, puesto a que este permanecerá estático.

Una vez asignados los pesos correspondientes a cada malla se realiza el proceso de animación, esto se realiza utilizando el método de “Cinemática directa” (Forward kinematics), utilizada para animar mecanismos lineales y rígidos.

Este tipo de animación utiliza poses clave que almacenan las posiciones relevantes para el actuador

Una “Pose Clave” del esqueleto de animación se representa en la Línea de tiempo como un rombo de color amarillo, ejemplo de esto en la figura 31. Esta pose clave representa la posición inicial de todos los huesos.

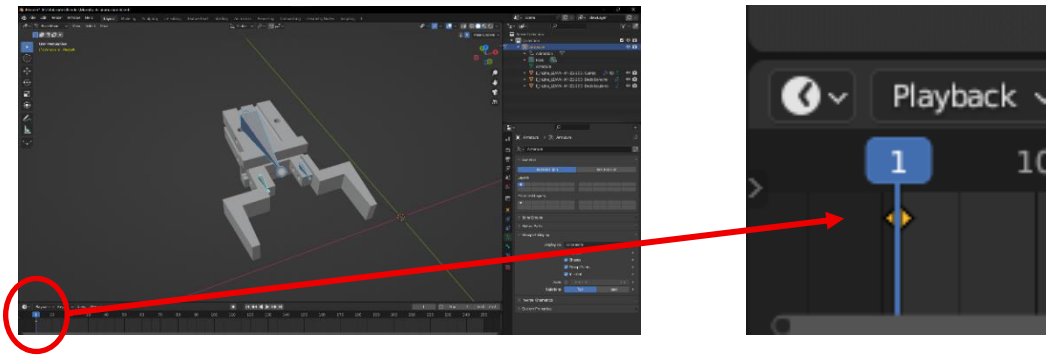

Figura 31: Representación de una "Pose Clave" en la línea de tiempo de Bleder.

La línea de tiempo representa fotogramas en su eje horizontal, y está configurada de forma nativa con una escala de 24 fotogramas por segundo, por lo que el tiempo de desplazamiento depende de la cantidad de fotogramas que haya entre las poses clave guardas para cada objeto/hueso. Debido a esto, para añadir un movimiento cuyo tiempo de movimiento sea 0.5s, debemos avanzar en la línea del tiempo 12 fotogramas proceso mostrado en la figura 32.

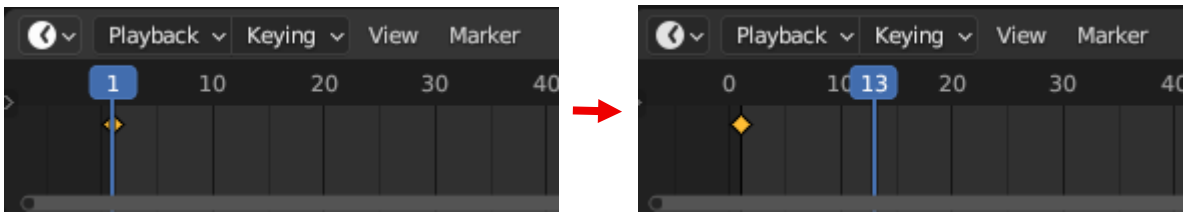

Figura 32: Desplazamiento de 12 fotogramas en la línea del tiempo.

Para que los cambios de posición, rotación, y escala del hueso “Dedo.R” sean aplicados de forma simétrica al hueso “Dedo.L”, se habilita la opción “Aplicar cambios al hueso correspondiente en el lado opuesto del eje x”, ubicada el en la esquina superior derecha de la interfaz gráfica. La ubicación de esta opción se muestra en la figura 33.

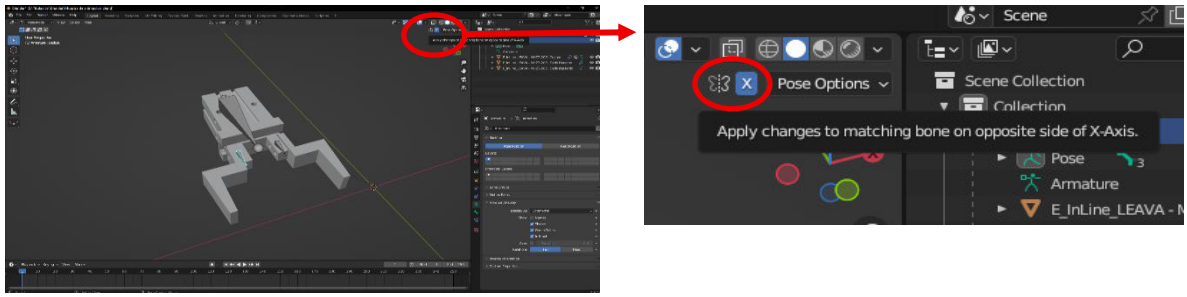

Figura 33: Habilitación de la opción “Aplicar cambios al hueso correspondiente en el lado opuesto del eje x” en Blender.

Ya que el modelo de planta tiene una escala 1:1 con respecto al modelo CAD, el hueso se desplaza una distancia equivalente a 1 CM, que es su rango de movimiento real. Este desplazamiento, al estar habilitada la opción “Aplicar

cambios al hueso correspondiente en el lado opuesto del eje x", también se verá reflejado en el hueso "Dedo.L", como se aprecia en la figura 34.

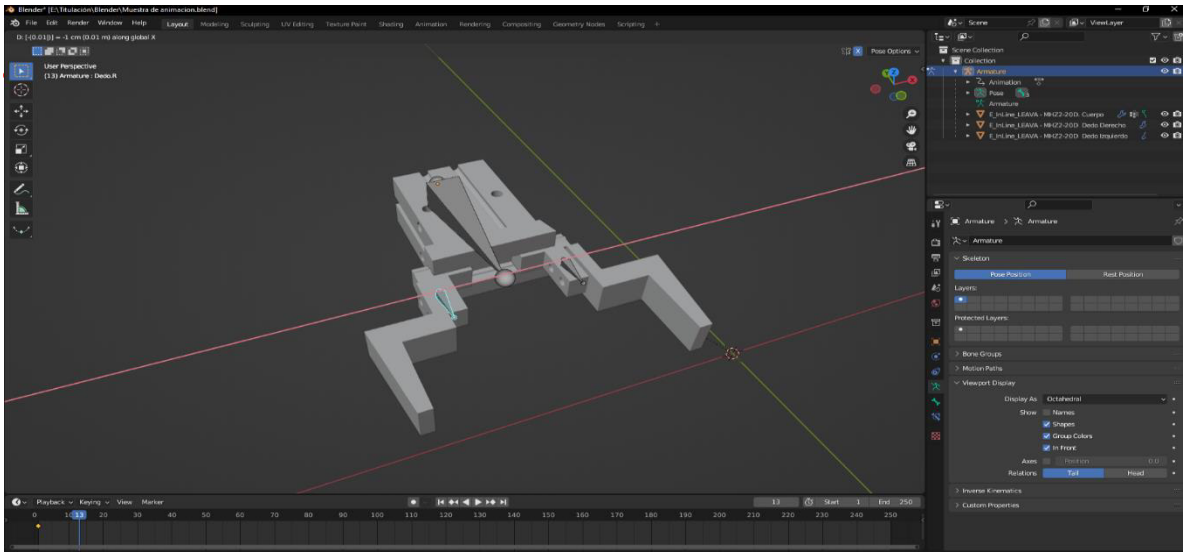

Figura 34: Desplazamiento simétrico de los huesos "Dedo.R", y "Dedo.L" en Blender.

Se repite el proceso de creación de la pose clave inicial.

Blender realiza una interpolación entre las poses clave de forma automática, de modo que el movimiento entre poses es fluido y no se perciben espacios vacíos. En la figura 35 se parecían distintos fotogramas a modo de demostración.

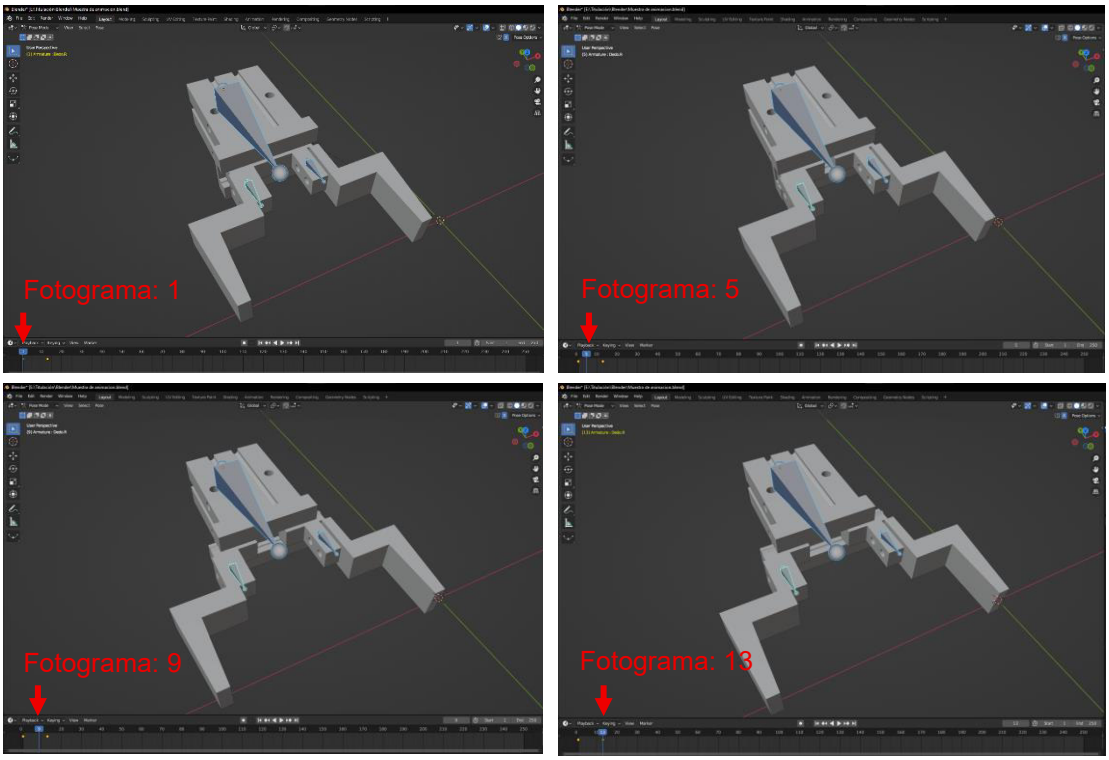

Figura 35: Interpolación automática de poses entre poses clave en Blender.

Con esto realizado se tiene completo el proceso de animación de “Extender” el vástago del actuador. Para realizar la animación de “Retraer” se copia la pose clave inicial, seleccionando todos los huesos, y se copia 12 fotogramas después de la pose clave anterior, es decir, en el fotograma 25. Este proceso se aprecia en la figura 36.

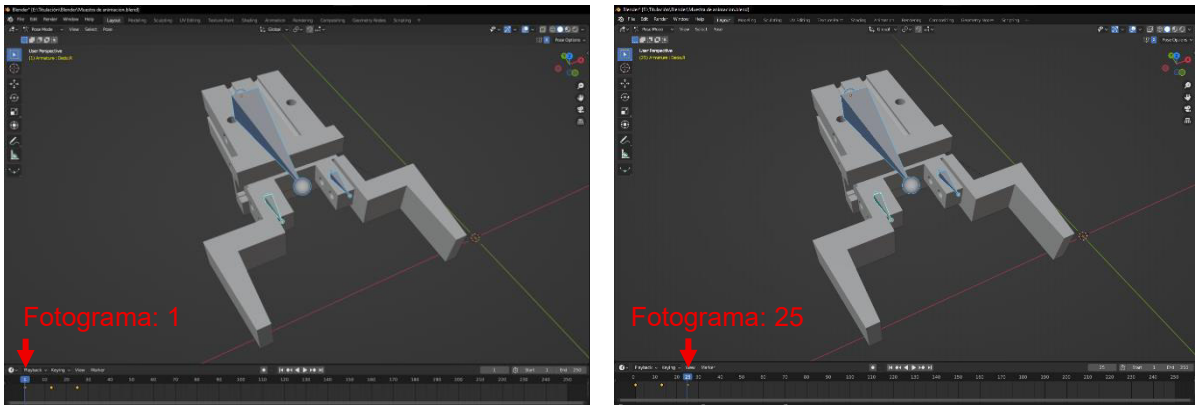

Figura 36: Pose clave del fotograma 25 idéntica a la pose clave del fotograma 1.

Así, el proceso de animación está completo para el actuador MHZ2-20D.

Como proceso de optimización para Unity 3D, las mallas del cuerpo del actuador, del dedo derecho, y dedo izquierdo, se combinan. Teniendo como resultado una sola malla con la misma asignación de pesos que cuando estaban separadas. El resultado de la combinación de las mallas se muestra en la figura 37.

‘

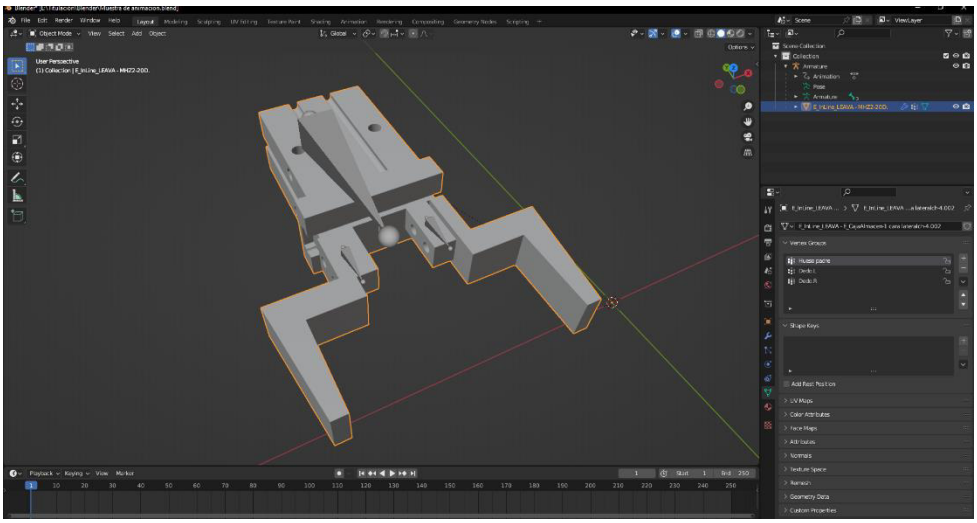

Figura 37: Combinación de mallas en el actuador MHZ2-20D. Último proceso de optimización en Blender.

Este proceso de animación y combinado de mallas se repite para el resto de actuadores neumáticos.

Animación de los actuadores no neumáticos

Los actuadores no neumáticos son las dos bandas transportadoras diseñadas, y el movimiento de la flecha del motor Nema 17.

Las animaciones de estos actuadores se realizaron de forma especial debido, en el caso de las bandas transportadoras a lo complejo que resultaría su animación por el método de “Cinemática directa”. Se decidió no animarlas, y en cambio

mostrar un indicador visual dentro del motor gráfico Unity 3D haciendo uso de un shader personalizado utilizando la herramienta “Shader Graph”.

Para hacer uso de este shader, se creó una malla que envuelve a los eslabones de la banda transportadora, siguiendo su trayectoria. Estas nuevas mallas se muestran en la figura 38. Es importante que estas mallas sean individuales, tanto entre ellas, como de la malla de la planta, además, se les asigna un material nuevo, llamado “Banda Shader”.

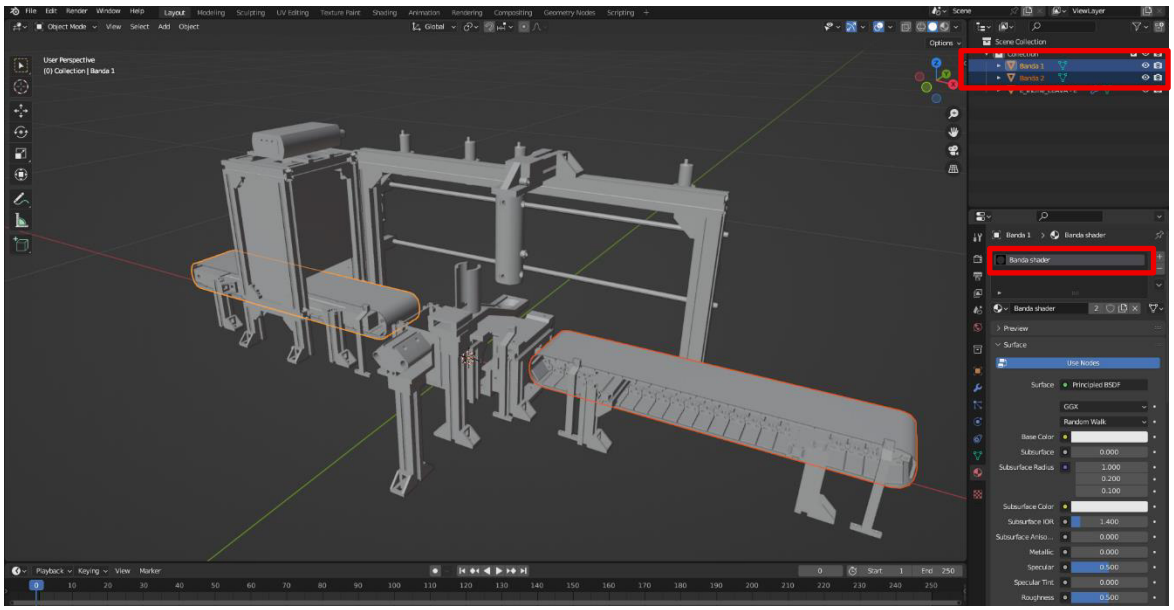

Figura 38: Creación de las mallas indicadoras de movimiento de las bandas transportadoras en Blender

Para la animación de la flecha del motor Nema 17, la varilla y la flecha se alinean al eje X en Blender, para su control dentro de Unity 3D. Esto se aprecia en la figura 39.

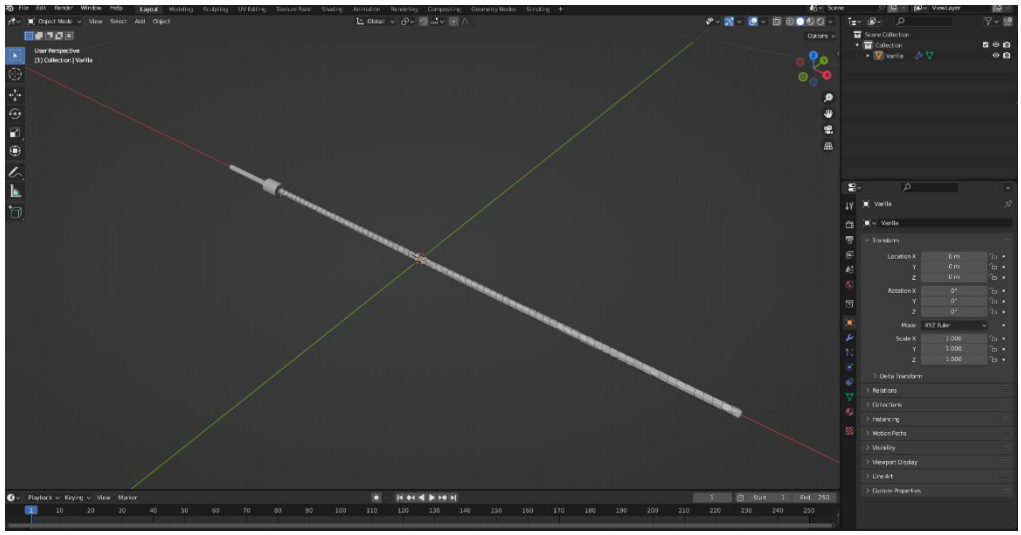

Figura 39: Flecha del motor Nema 17 alineada al eje x en Blender.

Con esto hecho, las animaciones de toda la planta están correctamente animadas, además, las mallas están optimizadas para su uso en Unity 3D.

5.8 Importación de archivos .blend al motor Unity 3D

Se usará la versión 6.1 LTS de Unity 3D, esta es compatible de forma nativa con los archivos .blend de Blender, por lo que, para importar las mallas que acondicionamos basta con copiar los archivos a una carpeta dentro del proyecto de Unity 3D.

Dentro de Unity 3D, en la carpeta “Assets” creamos una carpeta con el nombre “Recursos”, dentro de la cual, crearemos dos carpetas, llamadas “Planta”, y “Materiales”. Dentro la carpeta “Planta” copiamos los archivos .blend individuales que acondicionamos, y dentro de la carpeta “Materiales” copiamos la textura “textureAtlas” que utilizamos en Blender.

De este modo, el contenido de las carpetas creadas se muestra en figura 40 y 41.

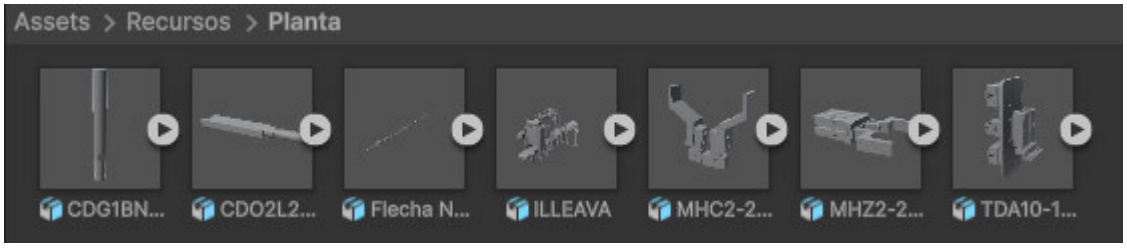

Figura 40: Contenido de la carpeta "Planta" en el proyecto de Unity 3D.

Figura 41:Contenido de la carpeta "Materiales" en el proyecto de Unity 3D.

5.9 Creación y asignación de materiales en Unity 3D

Para recrear la apariencia obtenida gracias a los materiales en Blender, se crean dos nuevos materiales en Unity 3D, en la carpeta omónima, de las mismas características: Un material con apariencia metálica, y uno con apariencia plástica-rugosa. A ambos materiales se les asigna la textura "textureAtlas" en el modificador "Base Map" del menú "Surface Input". Esto se muestra en la figura 42.

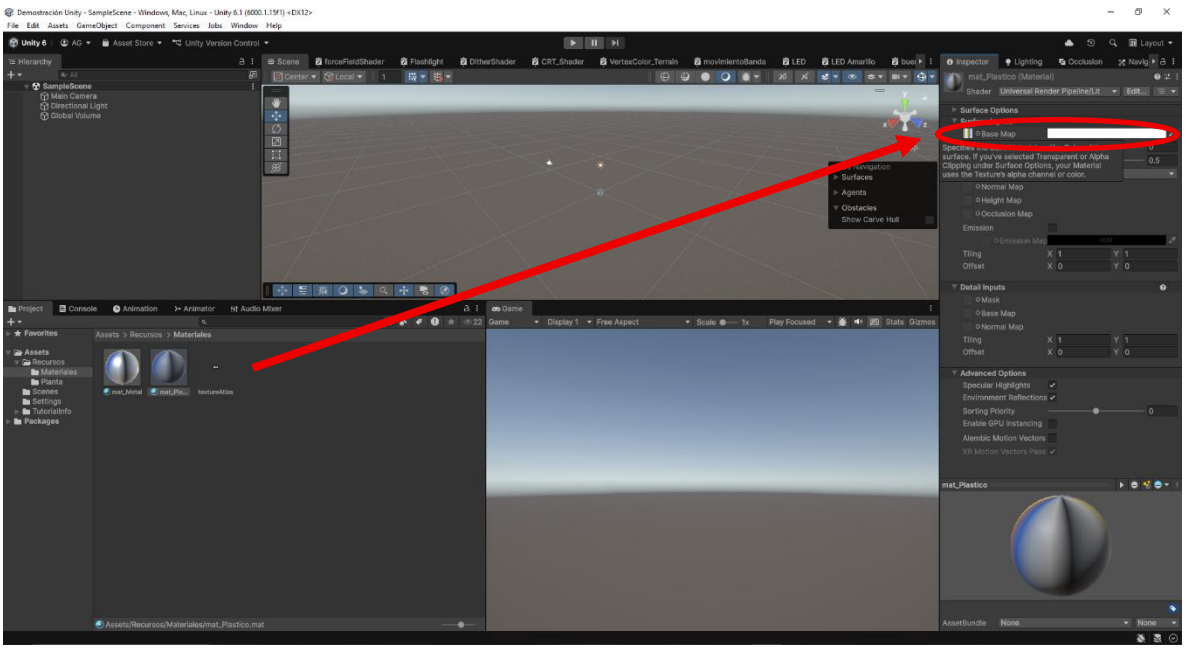

Figura 42: Asignación de la textura "textureAtlas" a los materiales en Unity 3D.

Al seleccionar los archivos .blend de la carpeta “Planta”, en la ventana “Inspector” veremos las opciones de importación de los modelos 3D. En la pestaña “Materials” de este menú, veremos dos materiales sin asignar, que son los materiales asignados en Blender. En estos espacios, seleccionamos los materiales creados anteriormente. Esto se muestra en la figura 43, donde además podemos ver la vista previa con los materiales creados en Unity 3D.

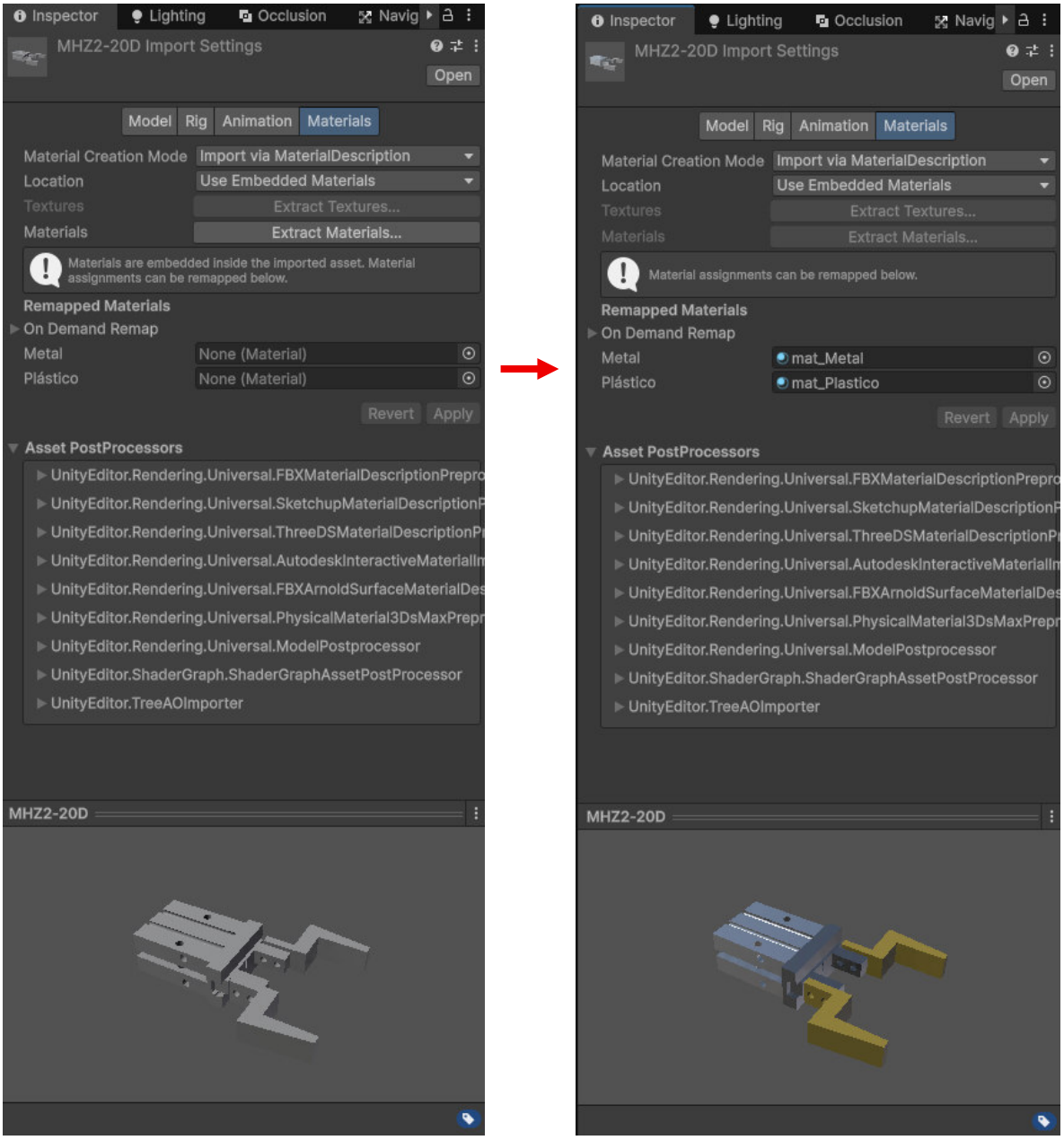

Figura 43: Asignación de los materiales creados en Unity 3D a los modelos de los archivos .blend importados.

5.10 Importar animaciones de Blender a Unity 3D.

Para importar las animaciones creadas en Blender, dentro de la ventana Inspector, en la pestaña “Animation” existe una sección llamada “Clips”. Existe un clip creado automáticamente llamado “Scene”, con la duración de fotogramas establecida en Blender, es decir, 25. Esto puede apreciarse en la figura 44.

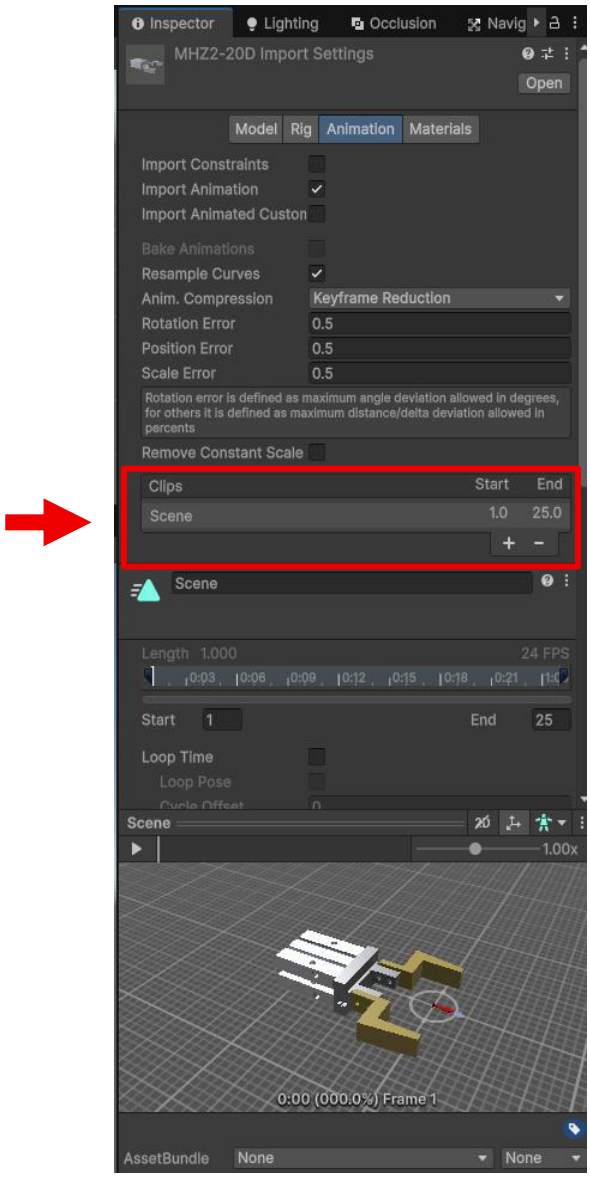

Figura 44: Clip genereado de forma automática al importar el modelo 3D a Unity 3D, con la cantidad de fotogramas asignada en Blender.

Ya que los modelos 3D tienen dos animaciones de 12 fotogramas, se crean dos clips de animación con los nombres: “nombre del actuador + Extender”, y “nombre del actuador + retraer”. Se les asigna el fotograma de inicio de cada animación. Para este caso, el clip “MHZ2-20D Extender” comienza en el fotograma 1, y termina en el fotograma 13; y el clip “MHZ2-20D Retraer” comienza en el fotograma 13, y termina en el fotograma 25. Los clips creados se muestran en la figura 45.

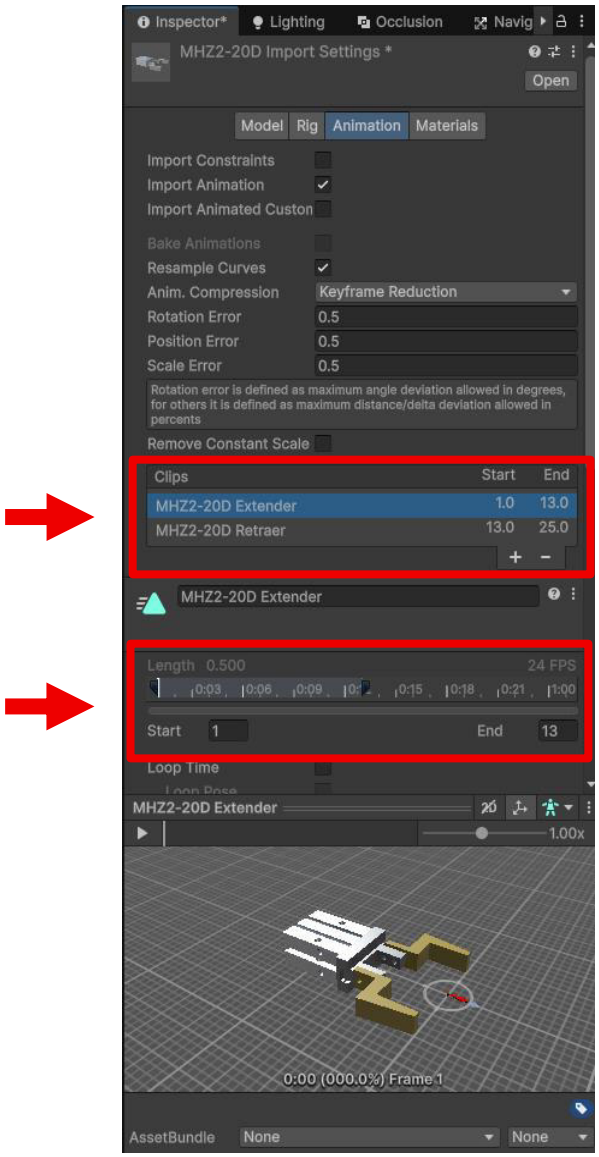

Figura 45: Creación de clips de animación correspondientes a cada actuador en Unity 3D.

5.11 Creación de shaders personalizados en Unity 3D

Para la malla de la planta genera se necesitan materiales no convencionales para generar la fidelidad visual que requiere, así como para generar un indicador visual del movimiento de las bandas transportadoras.

El motor Unity 3D cuenta con una herramienta para generar shaders personalizados de forma visual, llama Shader Graph.

A continuación, se explicará de forma breve el comportamiento de los Shaders generados.

Shaders para los indicadores LED

La cámara Banner Presence Plus P4 Area 1.3 incorpora tres indicadores led, cuya función y comportamiento es:

- LED amarillo: Indica que la cámara está en funcionamiento. Parpadea lentamente al estar encendida.

- LED verde: Indica que el resultado de inspección fue exitoso (pieza válida). Se enciende brevemente y luego se apaga.

- LED rojo: Indica que el resultado de inspección fue fallido (pieza inválida). Se enciende brevemente y luego se apaga.

Dado que los LED verde y rojo comparten el mismo comportamiento, diferenciándose únicamente por el color con el cuál encienden, se concluye que es necesario desarrollar dos shaders distintos: Uno para el Led Amarillo, y uno compartido por los LED verde y rojo.

LED Amarillo

Para simular el comportamiento del LED amarillo, se diseñó un shader personalizado. Este shader hace uso de la salida “Sine Time(1)” del nodo “Time”, que genera una función senoidal continua en función del tiempo.

Esta salida se conecta a un nodo “Clamp para limitar sus valores desde 0.05 a 1.0, evitando que el LED se vuelva completamente negro cuando la senoide alcanza su mínimo.

El resultado de esta operación se conecta a un nodo “Multiply”, junto con un color de tipo HDR (High Dynamic Range), para poder elegir la intensidad y color con que brilla el LED.

De este modo, la intensidad emisiva del LED se modula de forma oscilante ente los valores establecidos, generando el efecto visual de parpadeo suave en tiempo real.

El shader desarrollado se muestra en la figura 46.

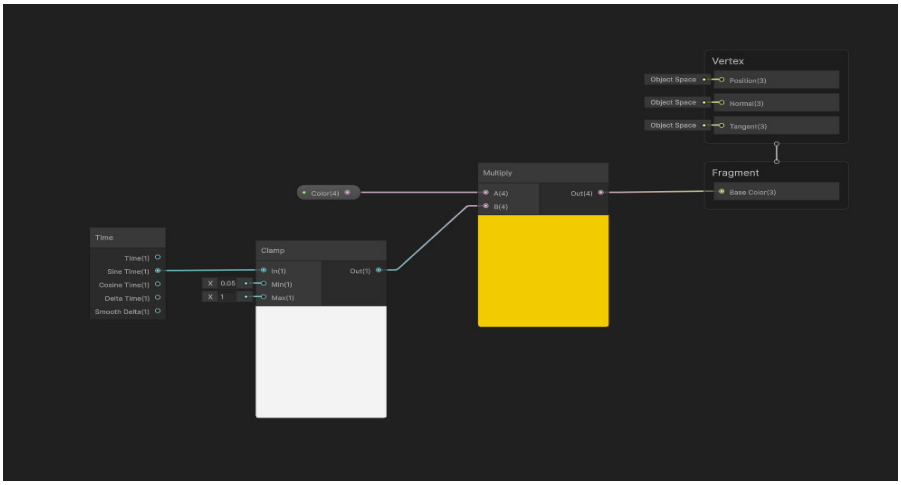

Figura 46: Shader personalizado desarrollado para el LED amarillo utilizando Shader Graph en Unity 3D.

LED verde y rojo

Para simular el comportamiento de los indicadores LED verde y rojo, se desarrolló un shader personalizado utilizando Shader Graph. Este shader hace uso de dos parámetros de entrada: un color HDR que define la tonalidad e intensidad máxima del brillo, y una variable de tipo “Float” (punto flotante) para modular dinámica la intensidad del color.

La estructura del nodo se basa en un nodo “Multiply”, el cuál combina ambos valores para generar un color final que permite controlar el encendido y apagado del LED mediante la modificación del valor de intensidad.

Este shader se muestra en la figura 47.

La gestión de esta variable se hace en una rutina escrita en C#, la cuál será detallada más adelante en este documento. Esta rutina permite activar o desactivar el brillo del LED en función del resultado de inspección obtenido por la cámara.

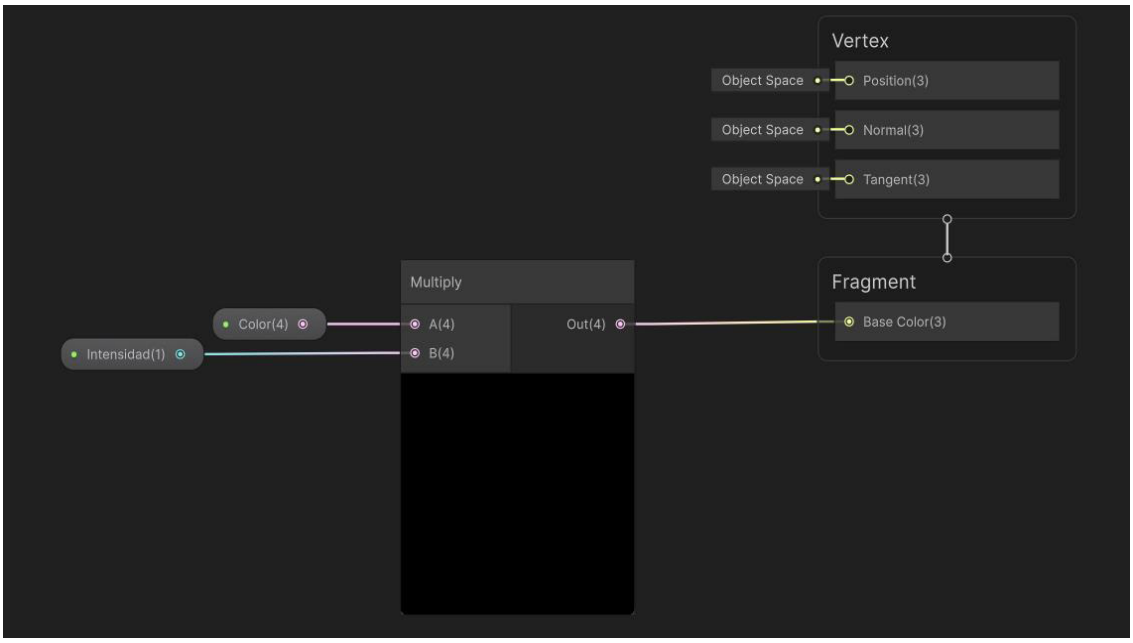

Figura 47: Shader personalizado desarrollado para los LED verde y rojo, utilizando Shader Graph, en Unity 3D.

Shader para las bandas transportadoras

Debido a que el movimiento de las bandas transportadoras es dinámico y constante. Se propuso un indicador visual de movimiento es una serie de flechas que simulen el desplazamiento de los eslabones, siguiendo la dirección de avance definida en diseñado mecánico. Para generar este indicador visual, se desarrolló un shader personalizado utilizando Shader Graph en Unity 3D.

La textura utilizada para representar las flechas es una textura PNG de 1024x1024 píxeles , que un triángulo blanco orientado hacia arriba, para representar la dirección del movimiento. Esta textura se muestra en la figura 48.

Figura 48: Textura de triángulo para la representación del movimiento del shader personalizado para las bandas transportadora en Unity 3D.

Para utilizar esta textura, se usa el nodo “Sample Texture 2D”, al cual se le asigna como entrada la imagen del triángulo.

Para generar el movimiento de esta textura, así como para generar un “mosaico” de flechas, se utiliza un nodo “Tilling and Offset”. La entrada “Tilling” especificamos un valor de tipo Vector 2 con los valores (1,15), de modo que la textura se repita una vez en el eje horizontal, y quince veces en el eje vertical, generando un matriz de triángulos que simulan el flujo de movimiento.

La entrada “Offset”, de este mismo nodo, controla el desplazamiento de la textura a lo largo del tiempo. Para generar este movimiento, se emplea el nodo “Time”,

por su salida “Time”, que representa el tiempo acumulado desde el inicio de la escena.

Esta salida se multiplica por una variable de tipo Float, denominada “Velocidad Flechas” empleando un nodo “Multiply”, lo que permite ajustar la velocidad del dezplazamiento.

Dado que el movimiento debe realizarse únicamente en el eje vertical, y en el sentido en que apunta la flecha, se emplea un nodo Vector 2, donde la salida del nodo “Multiply” se conecta a la entrada correspondiente al eje Y.

El shader desarrollado para el movimiento de las flechas genera un efecto visual de desplazamiento continuo de las flechas sobre la malla destinada para contener este shader, que nos permite representar el movimiento mecánido de las bandas transportadoras. La composición de este shader se muestra en la figura 49.

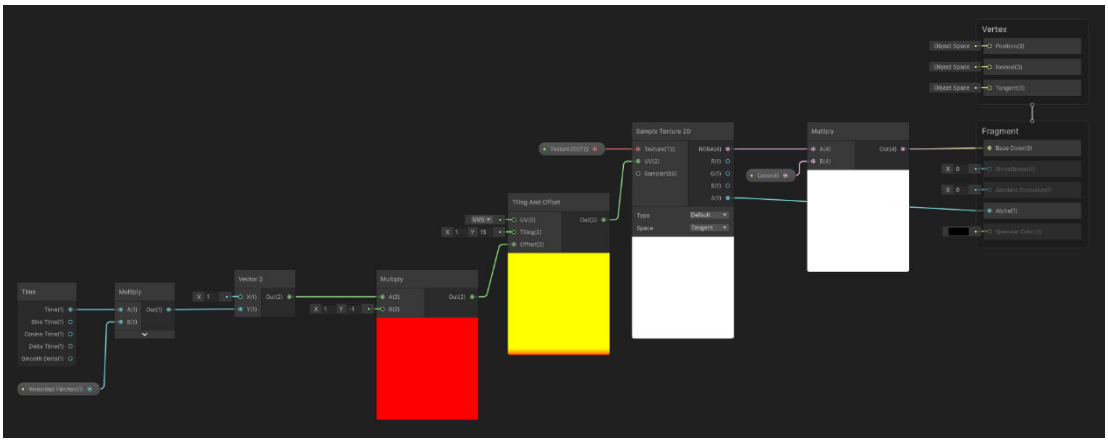

Figura 49: Shader desarrollado para simular el movimiento de las bandas transportedoras con Shader Graph, en Unity 3D.

La visualización del shader desarrollado se muestra en la figura 50.

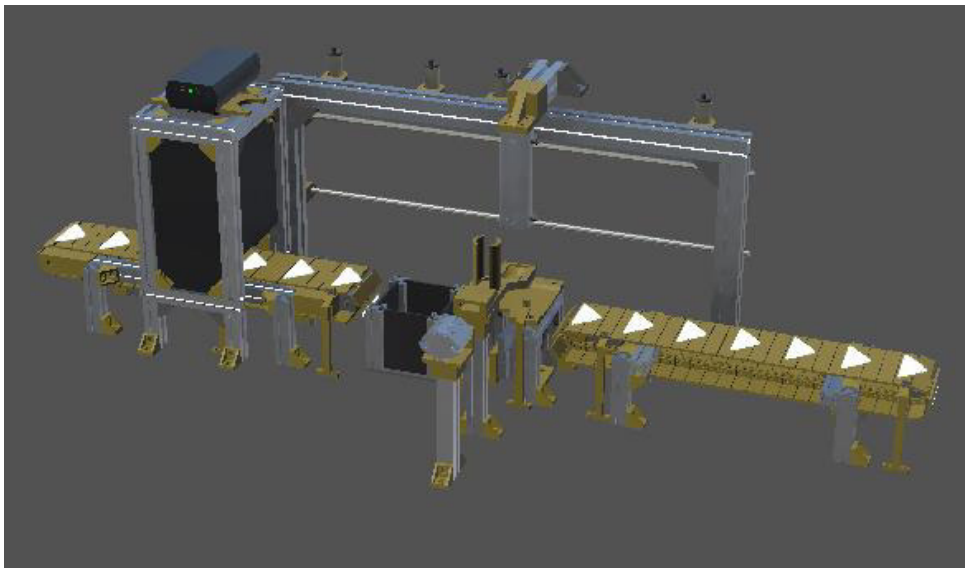

Figura 50: Visualización del shader desarrollado para representar el movimiento de las bandas transportadoras en Shader Graph, en Unity 3D.

Para utilizar los shaders personalizados desarrollados, se crean materiales individuales para cada shader, y se asignan a los modelos 3D donde son utilizados.

5.12 Importación de modelos 3D en escena de Unity 3D

Los objetos que se utilizan en el entorno 3D de Unity son conocidos como “GameObject” (objeto de juego). Para convertir los modelos 3D que fueron texturizados, y acondicionados, se arrastran los archivos a la ventana ”Hiererchy” (jerarquía) de Unity 3D. Para tener mejor orden, Se colocan dentro de un “Objeto vacío” “Empty Object”. Esto se muestra en la figura 51.

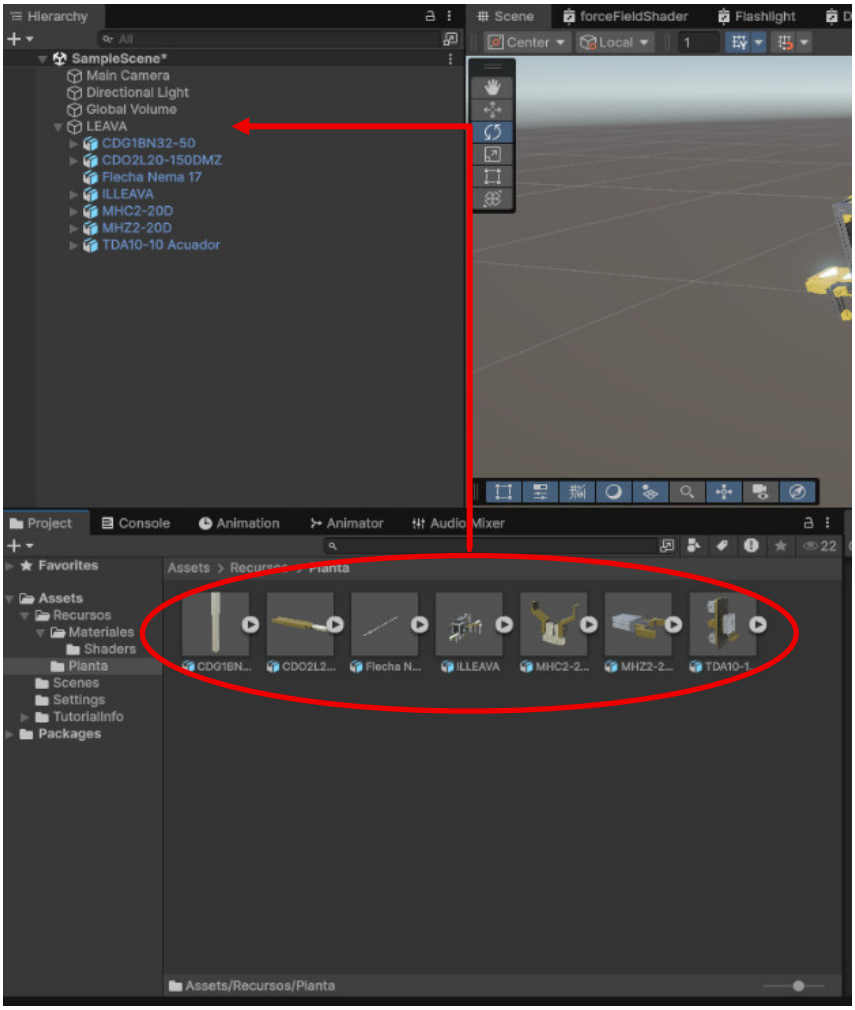

Figura 51: Importación de los modelos 3D texturizados a la jerarquía en Unity 3D.

Una vez arrastrados a la jerarquía, podemos apreciar la apariencia de la planta virtual con toros los materiales aplicados, en la figura 52.

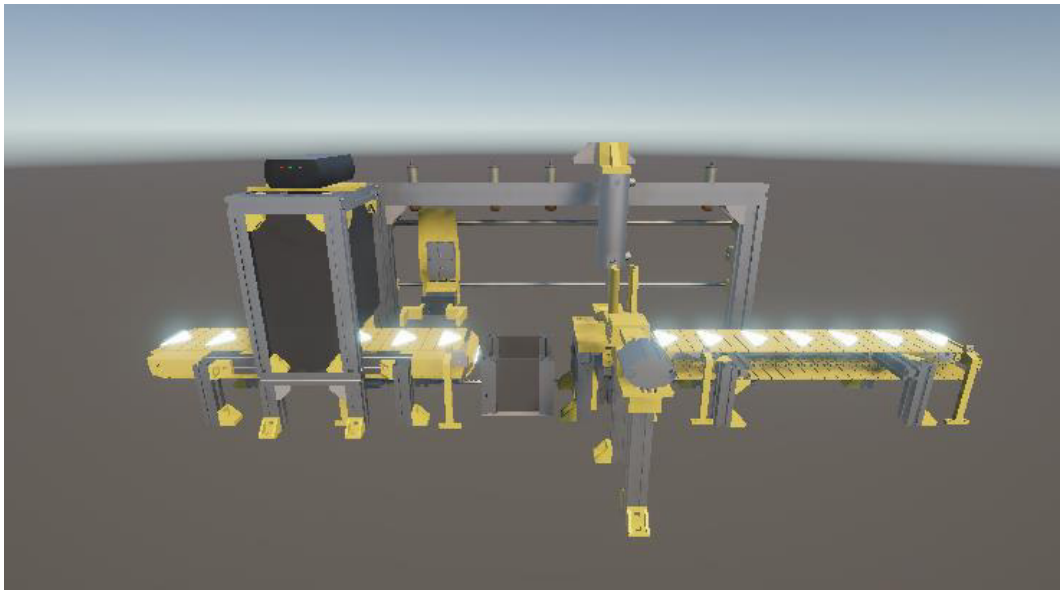

Figura 52: Visualización de la planta virtual con los materiales aplicados en el motor Unity 3D.

5.13 Creación de piezas representativas en Unity 3D

Para simular las piezas que se inspeccionaran para su posterior manejo, se crean dos GameObjects con las medidas establecidas durante el proceso de diseño de la planta (70x70x13mm).

A estas piezas se les asignan nuevos materiales: un material con un color base verde, para la pieza válida; y un material con color base rojo, para la pieza inválida.

Adicionalmente, se crean tags personalizados para cada una de estas piezas, siendo estos: “Good”, para la pieza válida; y “Bad” para la pieza inválida.

Se añaden además los componentes “Box collider” (colicionador), y “RigidBody”

El componente collider permite que los GameObjects participen en detecciones de colisiones con otros GameObjects dentro de la escena. Mientras que el rigidbody lo dota de propiedades físicas, como masa, el uso de gravedad, coliciones, etc.

Asignar estos componentes a los GameObjects de las piezas hará posible su interacción con múltiples procesos futuros, como su inspección por la cámara, y la capacidad de ser detectados por los sensores ópticos simulados.

Con este proceso aplicado, el GameObject de la pieza válida se muestra en la figura 53 a modo de ejemplo, y la visualización de la pieza puede apreciarse en la figura 54.

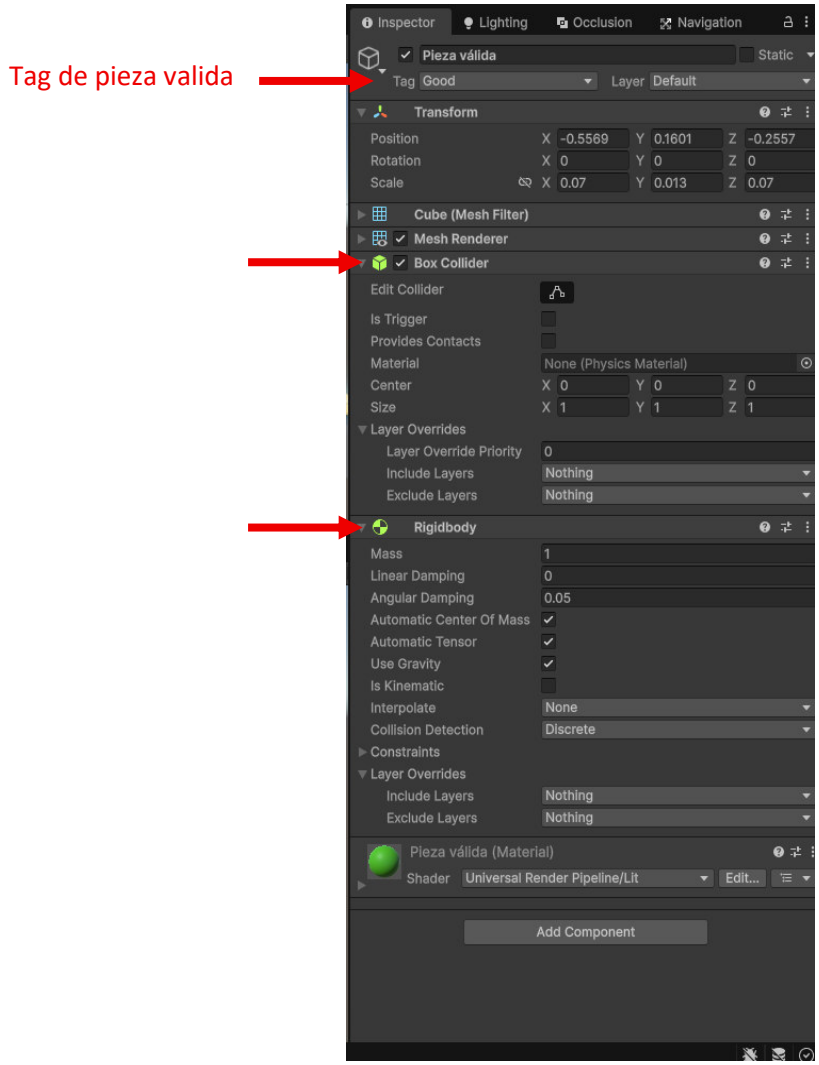

Figura 53: Asignación de compontes que permitirán manipular las piezas a inspeccionar.

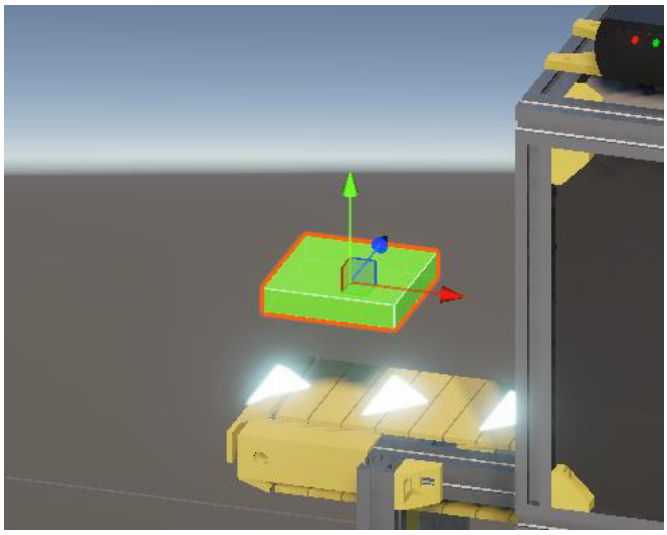

Figura 54: Visualización del GameObject de la pieza válda, en Blender 3D.

5.14 Creación de clips animación para los GameObjects de los actuadores neumáticos de la planta virtual de Unity 3D

Los clips de animación creados al importar los modelos 3D de Blender no pueden ser utilizados por los GameObjects de estos mismos modelos en Unity 3D.

Para implementar las animaciones a los GameObjects de la planta virtual, se deben crear clips de animación asociados a los GameObjects. Este proceso les añadirá un componente llamado “Animator”.

Dentro de la ventana “Animator” podremos ver los clips de animación creados para un GameObject, como los actuadores neumáticos tienen dos animaciones, los componentes Animator de estos, tendrán dos estados. Como ejemplo, en la figura 55 se muestran los estados de animación del componente Animator del actuador MHZ2-20D.

Para que los estados puedan alternarse, se crean transiciones que permitan pasar de un estado a otro, asociados a un parámetro de tipo booleano. Esto se muestra en la figura 56.

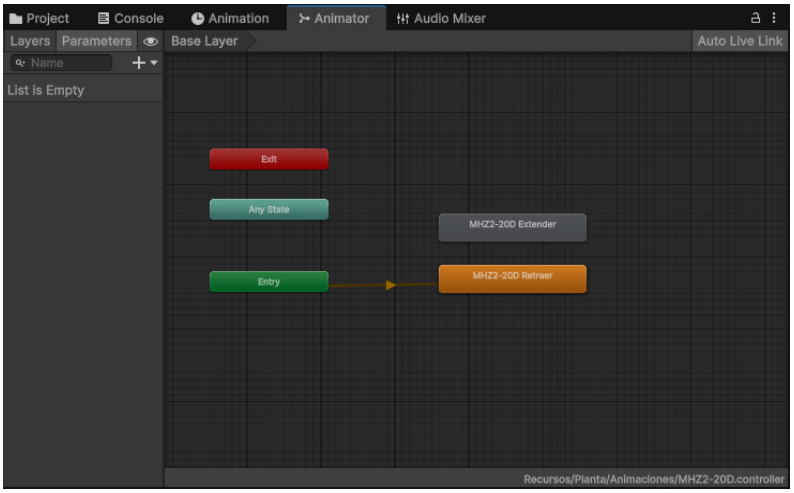

Figura 55: Estados de animación del actuador MHZ2-20D de la planta virtual en Unity 3D.

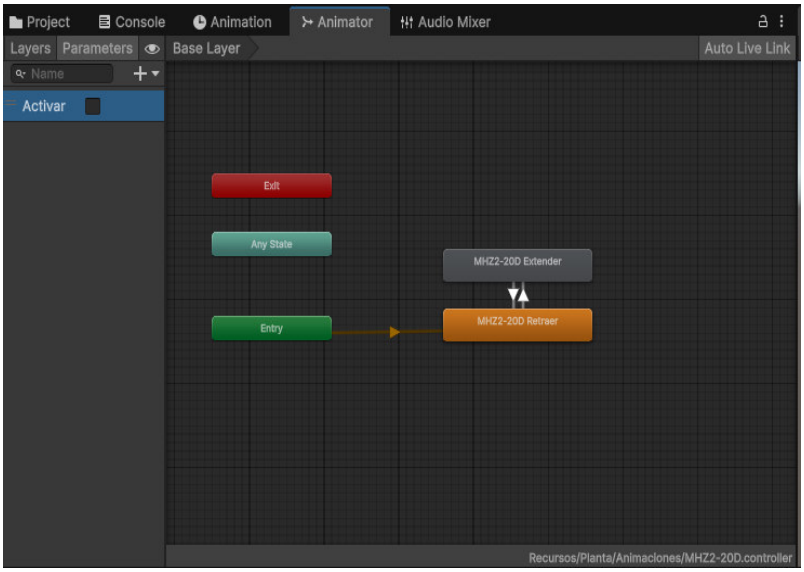

Figura 56: Creación de transiciones entre los estados de animación, y un parámetro booleano que permita realizar las transiciones de forma externa.

El parámetro booleano se asocia a las transiciones de modo que, si el booleano “Activar” es verdadero, el estado cambiará de “MHZ2-20D Retraer” a “MHZ2-20D Extender”, y se ejecutará su animación. Si el booleano cambia a falso, el flujo de transición será el contrario.

Este proceso se repite para el resto de los GameObjects de los actuadores neumáticos.

5.15 Programación

Programación de los actuadores neumáticos

Al tratarse de actuadores neumáticos de doble efecto, se desarrolló un código de C# que permite cambiar los estados de animación individualmente, simulando este comportamiento.

Para esto, se utilizaron las variables públicas mostradas en la figura 57:

```
[Header("Animación")]
public Animator animator;

[Header("Controles")]
public bool openCylinder;
public bool closeCylinder;
```

Figura 57: Variables del código C# de control para los actuadores neumáticos.

Se solicitan la variable “animator” de tipo “Animator”, que es una referencia al componente “Animator” presente en los actuares neumáticos, para sí poder acceder al estado de su parámetro de control.

Los booleanos públicos ”openCylinder”, y “closeCylidner” se utilizan como controles dentro del inspector de Unity 3D para cambiar el estado interno de el parámetro de control del Animator del actuador.

Para que este cambio de estados suceda, se comprueba dentro del ciclo Update() el estado de los controles, Si el control ”openCylinder” es Verdadero, se ejecuta la clase “open()”, que asigna el estado del parámetro “Activar” del Animator referenciado a Verdadero, mediante el uso de la instrucción “SetBool”.

Si “closeCylidner” es Verdadero, se ejecuta la clase “close()”, que asigna el estado del parámetro “Activar” del Animator referenciado a Falso

Esta comprobación de variables, así como la instrucción utilizada para escribir la variable del Animator del actuador, se muestran en la figura 58.

```
void Update()
{
    if(openCylinder)
    {
        open();
    }

    if(closeCylinder)
    {
        close();
    }
}

public void open()
{
    animator.SetBool("Activar", true);
}

public void close()
{
    animator.SetBool("Activar", false);
}
```

Figura 58: Comprobación de los controles, y escritura de la variable de control en el Animator, dentro del código C# de control de los actuadores neumáticos.

De este modo, se simula el comportamiento de doble efecto del actuador real en la planta virtual. Al hacer el estado del actuador dependiente del estado de dos variables boolenas.

Este código se asigna al GameObject del actuador, y se arrastra su componente Animator a la referencia en el código. Esto se muestra en la figura 59.

Este proceso se repite para cada actuador neumático.

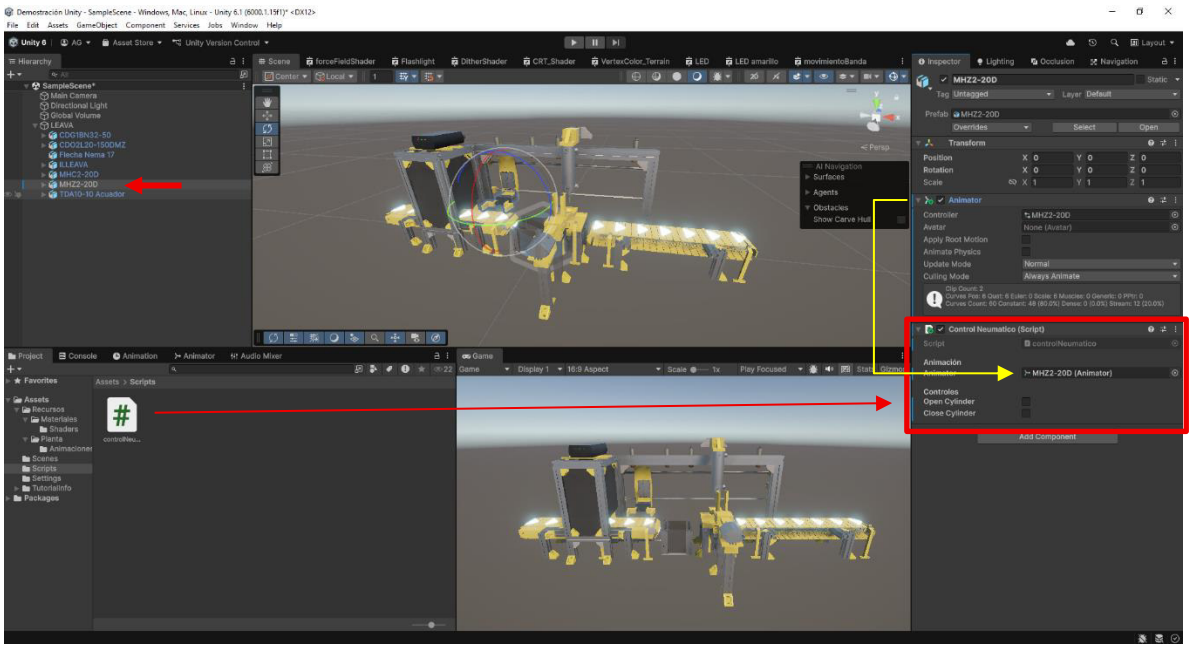

Figura 59: Asignación del código C# del control de animación de los actuadores, al GameObject del actuador. Además de la asignación del Animator de referencia.

Programación de actuadores no neumáticos

Programación de banda transportadora

La lógica de control de las bandas transportadoras se basa en la activación de un motor de corriente directa (DC), que permanece encendido mientras una señal de salida del PLC esté activa.

En el entorno virtual, la activación de una banda transportadora depende del estado de una varibale booleana, que determina la visibilidad del GameObject correspondiente. Para esto, se hace referencia directa al GameObject que representa la banda trasportadora, como se muestra en la figura 60.

```
[Header("Referencia de Bandas")]
public GameObject refBanda1;
public GameObject refBanda2;
```

Figura 60: Referencias a los GameObjects de las bandas transportadoras para su manejo.

El sistema de control hace uso de cuatro variables booleanas para gestionar el estado de las bandas transportadoras:

- Variables de control: “moverBanda1” y “moverBanda2”, definidas como variables públicas, que permiten modificar el estado de las bandas desde un sistema externo.

- Variables condicionales: ”banda1Activa“ y ”banda2Activa“, definidas como variables privadas, se encargan de gestionar la ejecución de las rutinas de activación y deasctivación.

El manejo del estado de la banda 1 se realiza mediante una clase que comprueba el estado de la variable “moverBanda1”, además de si la banda ya se encuentra activa. Si la banda 1 no se encuentra activa, el estado de la variable “banda1Activa” se establece Verdadera, y el GameObject correspondiente se activa.

La función de la variable condicionante es evitar que la activación de la banda se repita innecesariamente mientras la de control está activa.

Por otro lado, si el control cambia a Falso, y ”banda1Activa” es verdadero, el GameObject de la banda se desactiva, simulando la detención del motor.

La clase “manejoBanda1” se muestra en la figura 61. Esta misma lógica se aplica de forma análoga para la banda transportadora 2.

```
void manejoBanda1()
{
    if( moverBanda1)
    {
        if(!banda1Activa)
        {
            banda1Activa = true;
            refBanda1.SetActive(true);
        }

    }

    if(!moverBanda1)
    {
        if(banda1Activa)
        {
            banda1Activa = false;
            refBanda1.SetActive(false);
        }

    }
}
```

Figura 61: Clase encargada del manejo de la activación y desactivación del GameObject de la

Programación del arrastre de las bandas

Para simular el desplazamiento que generaría la activación de las bandas transportadoras sobre las piezas, se desarrolló una lógica utilizando el método “OnTriggerStay”.

Este método se invoca mientras un collider externo permanece solapando al collider del objeto sobre el que se aplica. Para esto, se asigna un collider a cada banda transportadora con la opción “Is Trigger” activa. Esta opción, permite que los collider puedan estar solapados, es decir, que la colisión sea continua.

El método comprueba el tag del GameObject del collider con el que la colisión está activa. Si el GameObject tiene asignado el tag “Good” correspondiente a la pieza válida, o “Bad”, para la pieza inválida; el GameObject es desplazado, con respecto

a su posición en el cuadro anterior, una distancia determinada por la variable “speedMultiplier”.

El método OnTriggerStay se muestra en la figura 62.

```
public float speedMultiplier;

void OnTriggerStay(Collider other)
{
    if(other.gameObject.CompareTag("Good"))
    {
        other.gameObject.transform.position = new Vector3( other.gameObject.transform.position.x + speedMultiplier,  other.gameObject.transform.position.y,
    }

    if(other.gameObject.CompareTag("Bad"))
    {
        other.gameObject.transform.position = new Vector3( other.gameObject.transform.position.x + speedMultiplier,  other.gameObject.transform.position.y,
    }
}
```

Figura 62: Método OnTriggerStay utilizado para simular el arrastre de la banda transportadora en las piezas.

Esta lógica se asocia al GameObject de cada banda, referenciados para la activación de las bandas. Así, mientras las bandas no estén activas, las piezas, aunque estén dentro del rango del collider, no se moverán.

Programación de la flecha del motor NEMA 17

Dado que la flecha del motor NEMA 17 tiene únicamente un movimiento de rotación, en dos sentidos: “Forward” (‘hacia adelante), y “Reverse” (hacia atrás); la programación lógica de programación se basó en la rotación de la varilla a través del eje X del GameObject correspondiente.

Para controlar el sentido de la rotación, se utilizan dos variables booleanas para elegir el sentido del giro de la flecha: “moveForward”, y “moveReverse”, que determinan si el sentido del giro es positivo o negativo.

La rotación se realiza utilizando el método “Rotate” de la clase “Transform”, dentro del ciclo Update(). De este modo, obtenemos una rotación continua relativa al estado actual del GameObject, en este caso, en el eje X.

Esta rotación es positiva cuando la variable booleana “moveForward” es Verdadera, y negativa cuando la variable boolaeana “moveReverse” es verdadera, simulando de este modo el comportamiento bidireccional del motor.

El contenido del ciclo Update() se muestra en la figura 63.

```
void Update()
{
    if(moveForward)
    {
        transform.Rotate(5f, 0f, 0f);
    }

    if(moveReverse)
    {
        transform.Rotate(-5f, 0f, 0f);
    }
}
```

Figura 63: Ciclo Update() del código C# que controla el sentido del giro del GameObject de la flecha del motor NEMA 17.

Programación del eje de movimiento

La lógica de programación de del eje de movimiento se desarrolló teniendo en mente la Instrucción DRVA de GX Works 3, software utilizado para programar el PLC Mitsubishi FX5U, que se trata del dispositivo objetivo de control de la planta virtual.

Esta instrucción requiere el valor de un registro de tipo Int (entero) donde se especifica una dirección “objetivo”, y una marca que indica que la instrucción está activa.

Para simular esto en el comportamiento de la planta virtual, se utilizan las siguientes variables:

- activarMovimiento: De tipo boolena pública, que permite el control del movimiento de forma externa.

- activarMovimientoAnterior: De tipo booleana privada. Almacena el valor anterior de la variable “activarMovimiento” para detectar cuánto existe un cambio de valor en esta.

- posObjetivo: De tipo entero, y pública. Esta almacena el valor del registro utilizado en GX Works 3.

- posObjetivoAnterior: De tipo entero, y privada. Almacena el valor objetivo al que llegó, o intentó llegar, para conocer la dirección en la que debe moverse el eje.

- velocidadMovimiento: De tipo entero, y pública. Cuya función es de determinar la velocidad con la que el eje se mueve a lo largo del eje horizontal.

- Se tiene además una referencia al código encargado de rotar el GameObject de la flecha del motor, llamada “scriptVarilla”.

Dentro del ciclo Update() se verifica el estado de la variable “activarMovimiento”, de modo que, mientras sea verdadera, la clase “ejecutarMovimiento” se ejecuta.

Dentro de esta clase, se comprueban el valor actual de la posición objetivo, con el valor anterior. En base a esto, si la posición objetivo actual es mayor que la posición objetivo anterior, el movimiento se considera positivo, y el eje se desplaza de forma positiva en el eje x. Para fidelidad visual de este movimiento, se acompaña con la rotación de la flecha del motor NEMA 17, escribiendo el valor de la variable moverFoward a “verdadero”.

Si la posición objetivo actual es menor que la posición objetivo anterior, se considera que el movimiento es negativo, y el eje se desplaza de forma negativa

en el eje x. Este desplazamiento escribe la variable moverReverse de la flecha del motor a “verdadero”.

El contenido de la clase ”ejecutarMovimienot()” se muestra en la figura 64.

```
void ejecutarMovimiento()
{
    if(posObjetivoAnterior < posObjetivo)
    {
        transform.position = new Vector3(transform.position.x + 0.001f * velocidadMovimiento, transform.position.y, transform.position.z);
        scriptVarilla.moveForward = true;
    }

    if(posObjetivoAnterior > posObjetivo)
    {
        transform.position = new Vector3(transform.position.x - 0.001f * velocidadMovimiento, transform.position.y, transform.position.z);
        scriptVarilla.moveReverse = true;
    }

}
```

Figura 64: Contenido de la clase "ejecutarMovimiento" del código en C# para la simulación del comportamiento del movimiento del eje.

Ya que la variable de activación “activarMovimiento” se desactiva una vez que se alcanza la posición objetivo, dentro del ciclo Update() se hacer también la gestión de este evento. Cuando sucede, se ejecuta la clase “guardarPos()”, donde la posición objetivo anterior se actualiza por la posición objetivo actual, es decir, a la posición donde se encuentra una vez que finaliza el movimiento.

Con esto, el movimiento de eje responde directamente a la instrucción que se utilizaría si se programara para utilizar un motor NEMA 17 físico con el PLC.

Programación de la cámara Banner Presence Plus P4 Area 1.3

Programación del comportamiento de los indicadores LED

Durante la creación del shader correspondiente a los LED verde y rojo, se mencionó el uso de una variable que regula la intensidad del brillo del material.

Se desarrolló una lógica que permita que esta variable tenga un comportamiento de encendido, y tras un tiempo de espera, vuelva a un estado de inactividad.

Para esto, se utilizaron como referencia los materiales que asignados a los modelos 3D, con la finalidad de acceder a sus variables intensidad.

Para esto, se utiliza una clase pública que comienza la ejecución de una rutina IEnumerator(), dentro de la cuál, la variable "_intensidad" del shader es asignada con un valor de 1. Para que, después de un tiempo de espera, se le asigne un valor de "inactividad" de 0.01.

El valor de la intensidad en inactividad no es cero, para evitar que el color del LED sea negro y tener una mejor representación visual.

En la figura 65 se muestra la clase y rutina utilizadas para el manejo del LED verde. El proceso se repite para el LED Rojo.

```
public void goodPiece()
{
    StartCoroutine("greenLED");
}

IEnumerator greenLED()
{
    materials[greenInt].SetFloat("_Intensidad", 1f);

    yield return new WaitForSeconds(ledDelay);

    materials[greenInt].SetFloat("_Intensidad", 0.01f);
}
```

Figura 65: Rutina de activación y desactivavión de los indicadores LED verde y rojo en la Cámara Banner.

Programación del proceso de inspección

Ya que el proceso de inspección con visión convencional resultaría negativo para el rendimiento de la aplicación, la inspección se simula utilizando el método “Raycast”.

El método Raycast lanza un rayo desde un punto de origen, en una dirección, y de una distancia máxima, contra todos los colliders en la escena.

Este método devuelve un booleano Vedadero cuando el rayo choca contra un collider, de otro modo, es Falso.

Los rayos, permiten acceder a información del collider contra el que chocaron, como la distancia a la que se encuentra, y el tag que esos tienen asignados.

De este modo, disparar un rayo en dirección a la pieza que quiere inspeccionarse nos brinda la información necesaria para saber si la pieza es válida o inválida, recreando fielmente un proceso de inspección de una forma que no afecte directamente al rendimiento de la aplicación.

La clase ”Inspeccion()” crea este rayo en la posición local del GameObject como origen y en la dirección “Forward” (eje Z +). Cuando el rayo impacta, la clase comprueba el tag del collider contra el que choca, si el tag corresponde a la pieza válida, se usa la clase “goolPiece()” del código encargado de animar la iluminación de los LED, para así tener un indicador visual del resultado de la inspección, y, del mismo modo, la variable “piezaBuena” es asignada como Verdadera. Esto sucede de forma análoga cuando el tag de la pieza inválida.

Este comportamiento simula el comportamiento de las salidas digitales físicas de la cámara.

Esta clase se llama cuando una variable de tipo booleana “triggerInspección” es Verdadera dentro del ciclo Update(), simulando el comportamiento de la entrada física “Trigger” de la cámara.

El contenido de la clase “Inspeccion()” se muestra en la figura 66.

```
public void Inspeccion()
{
    Ray ray = new Ray(transform.position, transform.forward);
    RaycastHit hit;

    if (Physics.Raycast(ray, out hit, distanciaInspeccion)) // Comprueba la colición del raycast en un distancia establecida
    {
        GameObject hitObject = hit.collider.gameObject;     // Se almacena el tag del GameObject con que colicionó el Raycast

        string objectTag = hitObject.tag;
        Debug.DrawRay(transform.position, transform.forward * distanciaInspeccion, Color.blue);  // Se dibuja el rayo disparado (EDITOR)

        if (objectTag == goodTag)      // Comprueba si el tag almacenado corresponde al de la pieza válida
        {
            Debug.Log("Pieza buena");
            ledManager.goodPiece();   // inicia la rutina de encendido de LED
            piezaBuena = true;        // Escribe el valor de salida "Pieza válida" a Verdadero.
        }

        else if (objectTag == badTag)   // Comprueba si el tag almacenado corresponde al de la pieza inválida
        {
            Debug.Log("Pieza mala");
            ledManager.badPiece();    // inicia la rutina de encendido de LED
            piezaMala = true;         // Escribe el valor de salida "Pieza inválida" a Verdadero.
        }

        else                          // Comprueba si el tag almacenado no corresponde a las piezas
        {
            Debug.Log("No hay pieza");
        }

    }

    StartCoroutine("turnOffSensors");   // Inicia rutina para reestablecer el etado de las salidas.
}
```

Figura 66: Clase "Inspeccion()" encargada de simular el proceso de inspección de la cámara Banner Presence Plus P4 Area 1.3.

Después de la inspección, se inicia la rutina “turnOffSensors”, que tas un periodo corto de tiempo, devuelve el estado de las variables “piezaBuena” y “piezaMala” a Falso, para así reiniciar el proceso.

La visualización del rayo se muestra en la figura 67. En ella, podemos apreciar el origen, y dirección en que se dispara el rayo, además de la distancia máxima en que se realiza la inspección. E l rayo está representado por una línea de color azul. Este rayo es únicamente visible en la ventana “Scene”, y no será visible durante la ejecución de la aplicación.

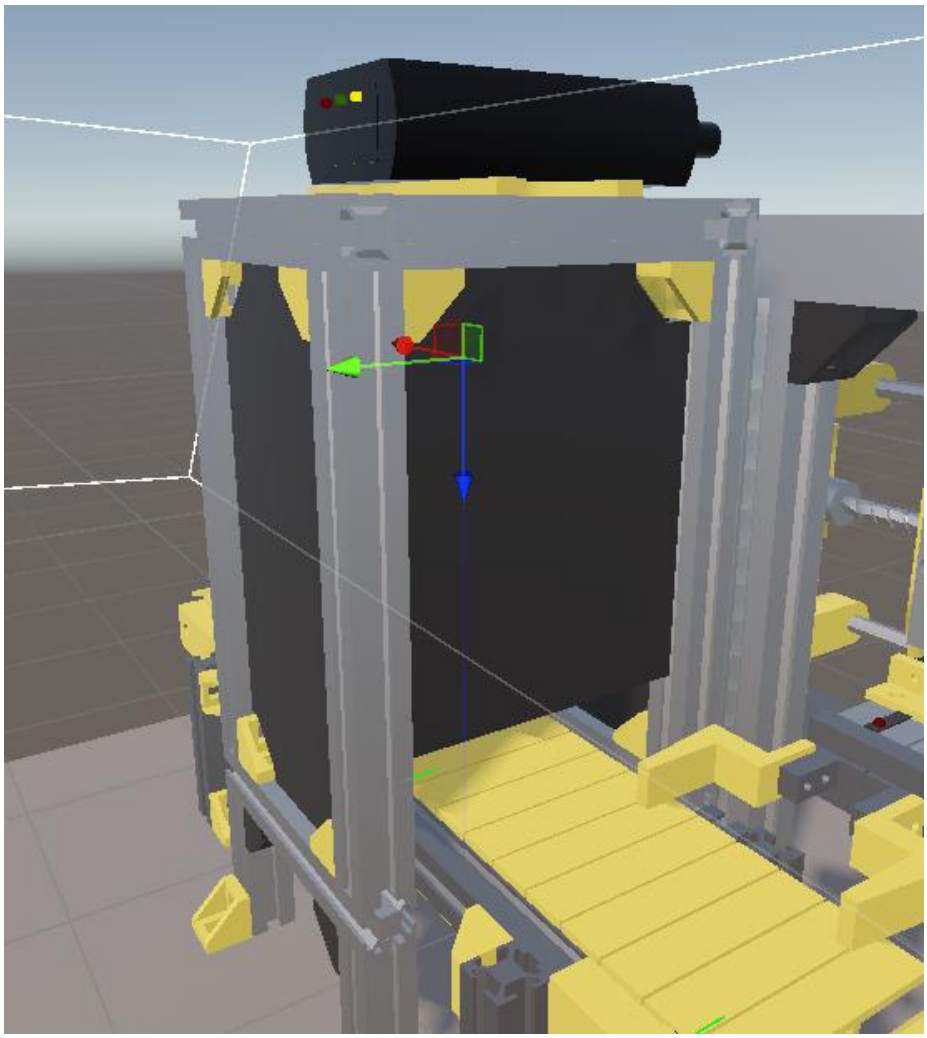

Figura 67: Origen, dirección, y distancia máxima a la que se dispara el rayo encargado de simulación la inspección de las piezas.

Programación de colocador de piezas

Al tratarse de un sistema In-Line, las piezas deberían llegar de forma automática a la banda transportadora 1, sin embargo, al tratarse de una estación individual, las piezas deben colocarse se forma manual.

Se desarrolló un programa que permita colocar piezas válidas e inválidas de forma que su instancia dependa de salidas del PLC individuales para el estado de la pieza que se quiere colocar.

La lógica se diseñó para tener como controladores dos variables booleanas, llamadas “spawnPiezaValida”, y “spawnPiezaInvalida”, y se utiliza una estructura similar a la utilizada en la lógica para el encendido de las bandas transportadoras, utilizando variables condicionales para que la lógica se ejecute una sola vez, aunque la señal de control permanezca encendida.

Dentro de la clase “manejoPiezaVálida”, se comprueba si la pieza ha sido colocada, si no lo ha sido, los GameObjects de las piezas (ambas) se desactivan, y la pieza válida se coloca en la posición inicial, establecida por el GameObject donde se asigna este código. Una vez que la pieza está en la posición correcta, el GameObject se activa de nuevo.

La estructura de la clase “manejoPiezavalida” se muestra en la figura 68.

```
void manejoPiezaValida()
{
    if(spawnPiezaValida)
    {
        if(!piezaValidaSpawneada)
        {
            piezaValidaSpawneada = true;
            piezaValida.SetActive(false);
            piezaInvalida.SetActive(false);
            piezaValida.transform.position = transform.position;
            piezaValida.SetActive(true);
        }
    }
    if(!spawnPiezaValida)
    {
        piezaValidaSpawneada = false;
    }
}
```

Figura 68: Clase "manejoPiezaValida()" encargada de posicionar la pieza válida en la posición inicial para la rutina de la planta virtual.

Una clase análoga se usa para la colocación de la pieza inválida.

Programación de sensores

La programación de los sensores de la planta virtual es un proceso importante para su fidelidad del comportamiento con respecto a la planta real para su correcta detección por el PLC.

Programación de sensores de ópticos CNY70

El sensor de proximidad utilizó una lógica similar a la lógica encargada de simular el proceso de inspección de las piezas para su diseñó.

Se utilizó la clase “lectura()”, que usa el método Raycast para comprobar si el sensor tiene un objeto delante de él, la distancia “distanciaInspección”, configurada en 5mm para simular la distancia a la que el sensor CNY70 opera.

Si hay algún GameObject con un collider asignado frente a él, la variable “detectando” se escribe como Verdadera, en caso contrario, como Falsa.

La clase “lectura()” se ejecuta en cada frame. Esta se muestra en la figura 69.

```
public void lectura()
{
    Ray ray = new Ray(transform.position, transform.forward);
    RaycastHit hit;

    int capaFinal = ~capaIgnorada.value;

    Debug.DrawRay(transform.position, transform.forward * distanciaInspeccion, Color.green);

    if (Physics.Raycast(ray, out hit, distanciaInspeccion, capaFinal))
    {
        detectando = true;
    }

    else
    {
        detectando = false;
    }
```

Figura 69: Clase "lectura()" encargada de simular el comportamiento de los sensores ópticos CNY70.

En la figura x70, se puede apreciar el rango de alcance del sensor. Este rango es sólo visible en el editor de Unity 3D, y no se muestra en la aplicación final.

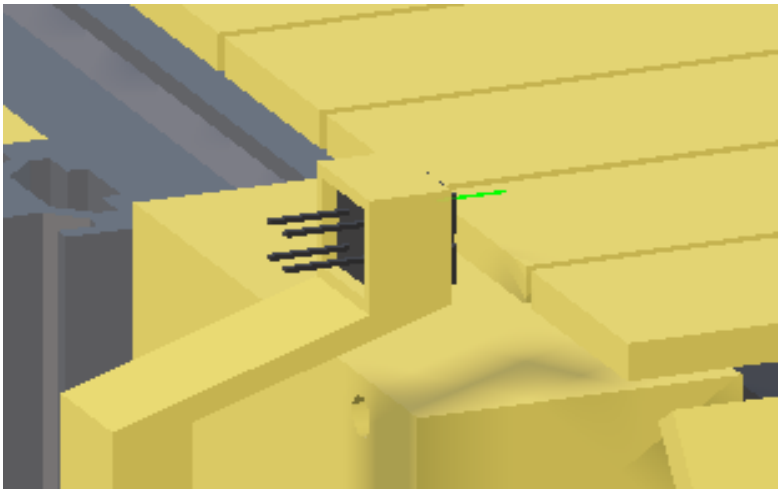

Figura 70: Representación del funcionamiento del sensor virtual CNY70.

Programación de sensores inductivos PR18-5DP

Debido a que la lógica de detección de los sensores PR18-5DP es, en términos prácticos, igual al de los sensores ópticos CNY70, es decir, emitir una señal, y medir la respuesta para detectar presencia, se utilizó la misma lógica para su simulación.

Ya que los sensores PR18-5DP tienen un LED a modo de indicador visual, este comportamiento se añadió en la simulación de dicho sensor. Para esto, se creo un GameObject con el mismo material creado para el LED rojo.

Se creo un código de C# auxiliar para controlar el “encendido” de este LED. Similar al código creado para los indicadores LED de la cámara Banner, sin embargo, esta no posé una rutina de “apagado” automático, de modo que el LED indicador del sensor PR18-5DP sólo se “apague” una vez que la lógica determine

que no está detectando la presencia de un objeto. Esta lógica se muestra en la figura 71.

```
public void activo()
{
    material.SetFloat("_Intensidad", 1f);
}

public void inactivo()
{
    material.SetFloat("_Intensidad", 0.01f);
}
```

Figura 71: Clases encargadas del "encendido" y "apagado" del indicador LED de los sensores PR18-5DP.

A la clase utilizada para el sensor CNY70, se añadió el uso de las clases “activo()” e “inactivo()” para encender y apagar el indicador LED del sensor PR18-5DP.

La clase modificada para ser utilizada en el sensor PR18-5DP se muestra en la figura 72, y en la figura 73 se muestra un ejemplo de su funcionamiento.

```
public void lectura()
{
    Ray ray = new Ray(transform.position, transform.forward);
    RaycastHit hit;

    int capaFinal = ~capaIgnorada.value;

    Debug.DrawRay(transform.position, transform.forward * distanciaInspeccion, Color.green);

    if (Physics.Raycast(ray, out hit, distanciaInspeccion, capaFinal))
    {
        detectando = true;
        ledIndicador.activo();
    }

    else
    {
        detectando = false;
        ledIndicador.inactivo();
    }

}
```

Figura 72: Clase utilizada para simular el comportamiento del sensor PR18-5DP.

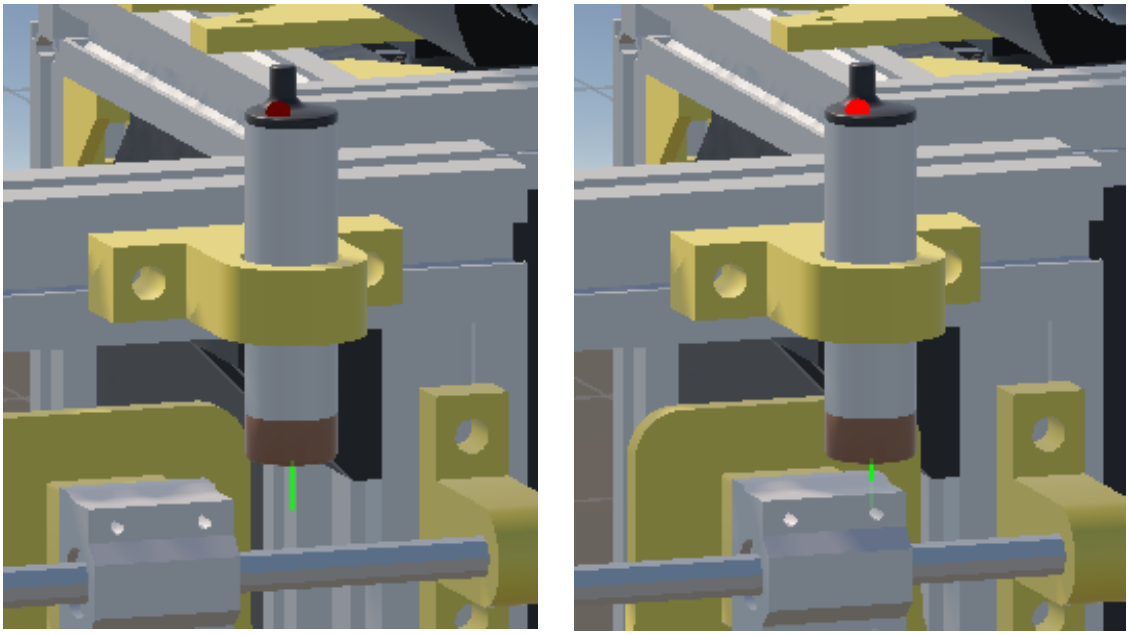

Figura 73: Demostración del funcionamiento de los indicadores visuales de los sensores PR18-5DP.

Programación de sensores de efecto Reed D-Z73

Los sensores de efecto Reed, utilizados en actuadores neumáticos reales, reaccionan a un imán ubicado en la punta del vástago para detectar su estado.

Este comportamiento se simuló utilizando colliders ubicados en los límites de movimiento del acutador (correspondientes a los dos sensores destinados a cada actuador), y un collider adicional ubicado en la punta del vástago móvil, representando fielmente el comportamiento de los sensores reales en la planta virtual.

La interacción entre los sensores y el vástago utiliza los métodos “OnTriggerStay”, y “OnTriggerExit”.

El método “OnTriggerExit” se ejecuta cuando los colliders dejan de interactuar entre ellos.

De este modo, en la lógica desarrollada, se utiliza el método OnTriggerStay para detectar cuándo el vástago se encuentra activando al sensor, y la variable "detectando" se escribe como Verdadera. Cuando se activa el sensor, y los colliders dejan de interactuar, se ejecuta el método OnTriggerExit, y la variable "detectando" se establece como Falsa. Estos métodos se muestran en la figura 74.

```
void OnTriggerStay(Collider other)
{
    if(other.gameObject.CompareTag("Vastago"))
    {
        detectando = true;
        material.SetFloat("_Intensidad", 1f);
    }
}

void OnTriggerExit(Collider other)
{
    if(other.gameObject.CompareTag("Vastago"))
    {
        detectando = false;
        material.SetFloat("_Intensidad", 0.01f);
    }
}
```

Figura 74: Métodos utilizados para simular la interacción entre los sensores D-Z73 y el vástago de los actuadores neumáticos virtuales.

Este tipo de sensor tienen un indicador LED como indicador visual cuando se encuentran activados. Para simular esto, se crearon GameObjects con el material creado para LED rojos. Su manejo se realiza dentro de los métodos utilizados en la lógica, cambiando el valor de la variable "Intensidad" para su encendido y apagado.

La distribución de los colliders involucrados, además de un ejemplo de funcionamiento para la simulación de los sensores D-Z73, se muestra en la figura 75.

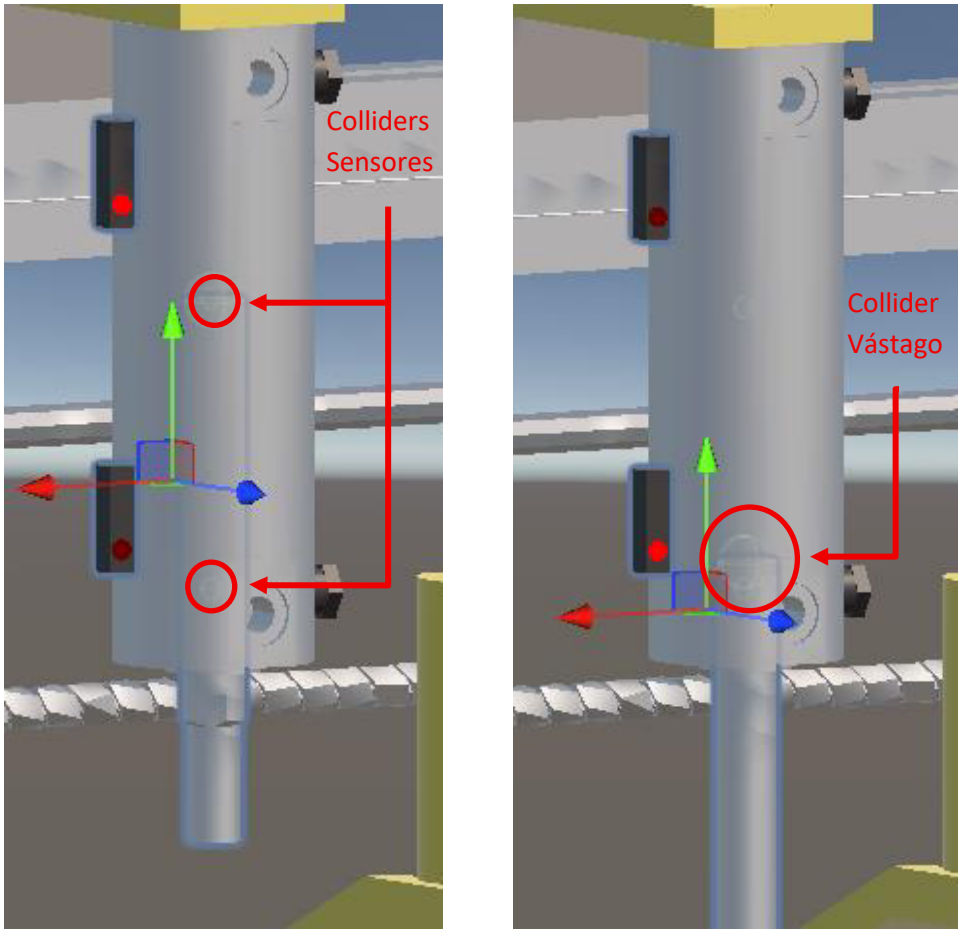

Figura 75: Prueba de funcionamiento de los sensores de efecto Reed D-Z73.

5.16 Programación de la comunicación MODBUS/TCP

Agrupación de señales

Se realizó un código llamado “managerPlanta”, utilizado como centro común para el almacenamiento y control de todas las variables involucradas para el control de la planta.

En este código, se definieron variables booleanas divididas en dos principales secciones :

- Sensores:

Las variables “sensores” son variables locales a este código, y su valor depende del estado de las variables en los códigos individuales, referenciados dentro del código.

Como ejemplo, si el del sensor SI 1 en la planta virtual está activo, la variable “detectando” de su propio código determinará el estado de la variable “sensorProximidad1” en “managerPlanta”. Esta asignación se muestra en la figura 76.

```
// Asignación de sensores Proximidad
sensorProximidad1 = refSensorProximidad1.detectando;
```

Figura 76: Asignación del valor de los sensores en las variables locales del código "managerPlanta", en base al estado de los sensores virtuales en la planta.

Esta estructura se repite para todas las variables asociadas a sensores en “managerPlanta”. Con esta asignación, la escritura de las variables locales queda bloqueada, y se vuelven de “sólo lectura”.

Dentro de esta categoría de sensores, se encuentran las variables asociadas al resultado de inspección de las piezas:

- Controles.

Las variables “controles”, escriben su valor a las variables en el código individual de cada actuador.

Como ejemplo, para extender el vástago del actuador TDA10-10, la variable “openCylinder” de su código debe ser Verdadera. Esta variable se asocia a la variable local “elevadorGripperIn” de “managerPlanta”. Así, cuando esta

variable es Verdadera en el código, se ejecuta la animación de extender el vástago del actuador virtual.

La estructura de esta asignación se muestra en la figura 77.

```
// Control de Actuadores
refControlElevadorGripper.openCylinder = elevadorGripperInControl;
```

Figura 77: Asignación del valor de una variable en "managerPlanta", al valor de una variable en el código individual de un actuador.

Esa estructura se repite para cada variable de control necesaria en la planta. Así, estás variables son de tipo “Lectura y escritura”. Al no estar determinadas por el estado de una variable externa.

Dentro de esta categoría de controles, entran el trigger de inspección de la cámara, y variables que controlan la colocación de la pieza válida, e inválida.

La asignación de ambos tipos de variables se realiza dentro del ciclo Update().

Conexión a TCP Cient

Para que Unity sea compatible con las funciones de comunicación MODBUS/TPC, se hace uso de la librería “NMOODBUS”. Instalada como Pluggin en el proyecto de Unity 3D.

5.17 Creación de interfaz de usuario para conexión a la red

Para que la comunicación pueda establecerse, se necesitan tres datos para conectase de forma exitosa con el PLC, siendo estos:

- Dirección IP (Internet Protocol) del PLC.

- Puerto usado para la conexión.

- ID del esclavo

Para el puerto, se utiliza el 502, debido a que este es el estándar de las comunicaciones industriales para adaptar el MODBUS a redes TCP/IP. Y el ID (Unit Identifie) del PLC es determinado 255 por el fabricante. Por lo tanto, estos datos permanecerán constantes.

Por otro lado, la Dirección IP del PLC dependerá del dispositivo al que quiere comunicarse, y de cómo éste esté configurado.

Para poder asignar la dirección IP del PLC para realizar la conexión, se creó una interfaz de usuario que contiene un bloque de texto editable, asociada a una variable de tipo String utilizada para establecer la conección.

La interfaz de usuario desarrollada se muestra en la figura 78.

Esta interfaz se abre cada vez que inicia la aplicación. En ella tenemos una entrada de texto, un indicador llamado “Estado”, un botón con la leyenda “Intentar Conexión”, y un botón que permite cerrar esta ventana.

Figura 78: Interfaz de usuario "Menú de conexión", para establecer la dirección IP del PLC al que se desea conectar, con un botón para establecer la conexión a dicha dirección IP.

El indicador “Estado” puede representar tres posibles casos:

- Conectado: Cuando se pudo establecer conexión con la dirección IP escrita en el campo de texto.

- Desconectado: Cuando la conexión no puedo establecerse.

- No hay IP: Cuando se intenta realizar la conexión, con el cuadro de texto vacío.

El botón “Intentar conexión” ejecuta la clase “conectarAPLC”, explicada a continuación.

El código C# desarrollado para establecer la conexión mediante MODBUS/TCP tiene el nombre “modbusEnlace”.

La clase “conectarAPLC” realiza un par de verificaciones antes de intentar establecer la conexión con la IP escrita en el campo de texto. Esta clase se muestra en la figura 79.

```
public void conectarAPLC()
{
// 1. Verificar si ya estamos conectados
    if (isConnected)
    {
        return; // Salir de la función si ya se está conectado
    }

    // Verificar que el cuadro de texno no esté vacío
    if (ipInputField == null || string.IsNullOrEmpty(ipInputField.text))
    {
    // Forzamos el estado de DESCONECTADO en el UI
        CleanUpConnection();

        if (statusIndicatorText != null)
        {
            statusIndicatorText.text = "No hay IP";
            statusIndicatorText.color = Color.yellow;
        }
        return; // Salir de la función si no hay IP
}

// 2. Si la validación pasa, asigna la IP y procede a intentar la conexión
    plcIP = ipInputField.text;
    AttemptConnection();
}
```

Figura 79: Clase "conectarAPLC" encargada de verificar que exista texto en la dirección IP a la que se quiere conectar (Inicio de conexión a PLC).

Una vez que la validación se realiza, se ejecuta la clase “AttempConnection”.

Esta clase crea el cliente TCP necesario para la comunicación MODBUS, que intenta comunicarse con la dirección IP establecida en la interfaz de usuario, a través del puerto 502.

Si la conexión es exitosa, se establece la IP el PLC como la IP del maestro, y se llama a la clase “UpdateConnectionIndicator”, que actualiza el estado del indicador del estado de la conexión como “CONECTADO”. Si la conexión no fue exitosa, se llama de igual forma a la clase “UpdateConnectionIndicator”, y en el indicador de estado de conexión se actualiza a “DESCONECTADO”.

SI el intento de conexión falla, se bloquea el cierre de la comunicación, y se reinician las variables utilizadas para establecer la conexión.

El contenido de la clase “AttempConnection” se muestra en la figura 80, y el de la clase “CleanUpConnection()” en la figura 81.

```
public void AttemptConnection()
{
    try
    {
        tcpClient = new TcpClient();      // Creación de Cliente TCP
        IAsyncResult result = tcpClient.BeginConnect(plcIP, port, null, null);
        bool success = result.AsyncWaitHandle.WaitOne(TimeSpan.FromSeconds(3));

        if (success && tcpClient.Connected)
        {
            // Bloquea solo la asignación final del maestro y el estado
            lock (connectionLock)
            {
                master = ModbusIpMaster.CreateIp(tcpClient);
                master.Transport.ReadTimeout = 1000;
                isConnected = true;
            }

            //Debug.Log($"CONECTADO a {plcIP}:{port}.");
            UpdateConnectionIndicator();

        }
        else
        {
            CleanUpConnection();
            //Debug.LogWarning("Conexión fallida (Timeout o error de red).");
            UpdateConnectionIndicator();
        }
    }
    catch (System.Exception ex)
    {
        //Debug.LogError($"❌ Error crítico al intentar conectar: {ex.Message}");
        CleanUpConnection();
    }
}
```

Figura 80: Clase "AttempConnection()" encarcada de establecer conexión con el PLC a través del protocolo MODBUS, utilizando la dirección IP establecida en la interfaz de usuario.

```
void CleanUpConnection()
{
    // Bloquea el cierre para que ningún otro hilo lo use mientras se cierra
    lock (connectionLock)
    {
        isConnected = false;
        master?.Dispose(); // Cierra el maestro Modbus
        master = null;     // Asegura que la referencia se anule
        tcpClient?.Close(); // Cierra el cliente TCP
        tcpClient = null;  // Asegura que la referencia se anule
    }

    UpdateConnectionIndicator();
}
```

Figura 81: Clase "CleanUpConnection" encargada de cerrar la comunicación, y reiniciar las variables de conexión.

Con la conexión entre Unity 3D y el PLC, el intercambio de estados de bobinas puede realizarse de forma satisfactoria.

5.18 Escritura de estados de sensores de la planta virtual, a marcas en PLC.

La librería NModbus utiliza la instrucción mostrada en la figura 82 para escribir el estado múltiples bobinas.

```
master.WriteMultipleCoils(slaveID, startingAddress, sensorStates);
```

Figura 82: Instrucción "WriteMultipleCoils" utilizada por la librería NMODBUS.

Donde la variable “slaveID” tiene el valor constante de 255; “startingAddress” es una variable de tipo ushort, donde se especifica la dirección a partir de cuál se escribirán las bobinas; y la variable “sensorStates” se trata de un arreglo de variables booleanas con lo estados que quieren escribirse, en este arreglo se insertan los valores de sensores de “managerPlanta”, como se muestra en la figura 83. La variable “planta” represente la referencia a “managerPlanta”.

```
bool[] sensorStates = new bool[]
{
    planta.sensorInductivo1, planta.sensorInductivo2, planta.sensorInductivo3, planta.sensorInductivo4,
    planta.sensorProximidad1, planta.sensorProximidad2, planta.sensorProximidad3, planta.sensorProximidad4, planta.sensorProximidad5,
    planta.elevadorGripperIn, planta.elevadorGripperOut,
    planta.gripperInSensor, planta.gripperOutSensor,
    planta.ensambleInSensor, planta.ensambleOutSensor,
    planta.gripperEnsambleInSensor, planta.gripperEnsambleOutSensor,
    planta.baleroInSensor, planta.baleroOutSensor,
    planta.piezaBuenaIndicador, planta.piezaMalaIndicador,
};
```

Figura 83: Creación de arreglo "sensorState" donde se almacena el estado de los sensores de la planta virtual.

Se asigna el valor 1 a la variable “startingAdress”, para que, así, la asignación de estados de las bobinas comience en M1 (Coil 1) del PLC.

Con este proceso, se escribe el valor de 21 variables correspondientes a estados de sensores, en la planta virtual, en marcas internas del PLC.

5.19 Lectura de registros de PLC a Unity 3D

La lectura de registros se realiza utilizando la instrucción mostrada en la figura 84.

```
ushort[] data = master.ReadHoldingRegisters(slaveID, D0Address, 1);
```

Figura 84: Instrucción utilizada por la librería NMODBUS para leer el valor de registros.

La instrucción “ReadHoldingRegisters” nos permite leer múltiples registros desde el PLC, para lo cual, se espcifican los valores “slaveID” (255), la dirección a partir de la cuál queremos leer registros, y la cantidad de registros que quieren leerse.

Debido a que la planta virtual necesita un solo registro para su funcionamiento, se asigna el valor 1 a este parámetro en la instrucción.

Para asignar el dato leído a la variable correspondiente en “managerPlanta”, siendo esta la variable “registroPos”, se utiliza la línea de código mostrada en la figura 85.

```
planta.registroPos = (short)data[0];
```

Figura 85: Asignación del valor de registro leído a la variable correspondiente en la referencia a "managerPlanta".

Donde se establece el valor del primer elemento del arreglo leído del PLC a la variable correspondiente para su uso en la planta virtual.

5. 20 Lectura de estados de controles del PLC a Unity 3D

De forma similar a la lectura de registros, la instrucción “ReadCoils” necesita los parámetros: “slaveID”, “startCoilAddress”, y “numCoilsToRead”. Que representan

el esclavo del que se quieren leer los datos (255 en este caso); la dirección a partir de la cual se quiere hacer la lectura, y la cantidad de bobinas que quieren leerse.

Esta instrucción retorna un arreglo de valores booleanos con los estados de los registros leídos.

Dado a que se utilizaron bobinas anteriormente para la escritura de bobinas, la lectura debe hacerse a partir de una bobina donde las instrucciones de lectura y escritura no se sobrepongan.

Además, existe un límite de 10 elementos (bobinas) que puede entregar el arreglo leído, porque esta instrucción se utiliza dos veces para leer la totalidad de las marcas necesarias para el control de la planta virtual.

La asignación del valor de los elementos dentro del arreglo, hacia las variables correspondientes se realiza de la forma mostrada en la figura 86. Donde se asigna el valor de una dirección en el arreglo a las variables de “managerPlanta”.

```
planta.elevadorGripperInControl = coilStatus[0];
planta.elevadorGripperOutControl = coilStatus[1];
planta.gripperInControl = coilStatus[2];
planta.gripperOutControl = coilStatus[3];
planta.ensambleInControl = coilStatus[4];
planta.ensambleOutControl = coilStatus[5];
planta.gripperEnsambleInControl = coilStatus[6];
planta.gripperEnsambleOutControl = coilStatus[7];
planta.baleroInControl = coilStatus[8];
planta.baleroOutControl = coilStatus[9];
```

Figura 86: Escritura de diez estados de bobinas (marcas) del PLC a controles para la planta virtual.

5.21 Velocidad de sondeo

Tanto la lectura como la escritura de variables se ejecutan con respecto a una rutina de sondeo establecida en la clase “ModbusPollingLoop”. Dentro de esta, mientras la planta se encuentre conectada al PLC, se ejecutará las rutinas de lectura y escritura en función de un tiempo de espera entre ejecuciones establecido por la variable “pollRate”.

La estructura de esta rutina de sondeo se muestra en la figura 87.

Para que el funcionamiento de la planta sea lo más similar al funcionamiento real, se necesita establecer una velocidad de sondeo tan alta como sea posible sin sacrificar el rendimiento de la aplicación. Gracias a los múltiples procesos de optimización realizados, fue posible establecer la velocidad de sondeo en 0.001s.

```
void ModbusPollingLoop(CancellationToken token)
{
    while (!token.IsCancellationRequested)
    {
        if (isConnected && master != null)
        {
            // A. Manejo de Lecturas de Polling (PLC -> Unity)
            PollDataContinuously();

            // B. Escritura de Sensores (Unity -> PLC)
            WriteSensorsToPLC();

        }

        // Pausa el hilo usando el pollRate
        Thread.Sleep(TimeSpan.FromSeconds(pollRate));
    }
}
```

Figura 87: Rutina que establece la velocidad den sondeo de los sensores y controles de la planta virtual, y el PLC.

Con esto hecho, la comunicación entre la planta virtual y el PLC mediante el protocolo MODBUS/TCP se realiza en tiempo real, sin pérdidas.

5.22 Desarrollos alternos

Sistema de detección de colisiones no deseadas

Ya que, en la planta virtual, los desplazamientos se realizan utilizando código y no un sistema de físicas, para añadir colisiones no deseadas se desarrolló un código que utiliza el método OnTriggerEnter para detectar colisiones entre piezas que resultarían en fallas mecánicas, con la finalidad de añadir realismo al manipular la planta virtual.

Cuando una colisión de este tipo es detectada, una ventana con la leyenda “¡Hubo una colisión!” se muestra en la pantalla. En esta ventana, hay un botón con la leyenda “Reiniciar Escena”, que al ejecutarse recarga la aplicación, devolviéndola a su estado original.

La ventana creada para la interfaz de usuario se muestra en la figura 88.

Figura 88: Interfaz gráfica para detección de una colisión entre objetos no deseada.

Dentro de la instrucción para escribir el estado de los sensores, se incluyó un valor dedicado a la detección de este tipo de eventos.

Botones en interfaz gráfica

Se añadieron cuatro botones a la interfaz gráfica con las siguientes funciones:

- Botón “Reiniciar Escena”: Si existe algún error en el comportamiento de la planta, incongruencias de movimientos, o reacciones no deseadas por la planta virtual, este botón permite reiniciar la aplicación en cualquier momento.

- Botones “M42”, “M43”, y “M44”: Estos botones permite cambiar el estado de las marcas correspondientes en el PLC, permitiendo comenzar rutinas, u acciones de forma directa en la aplicación.

La interfaz gráfica, con estos botones añadidos, tiene el aspecto mostrado en la figura 89.

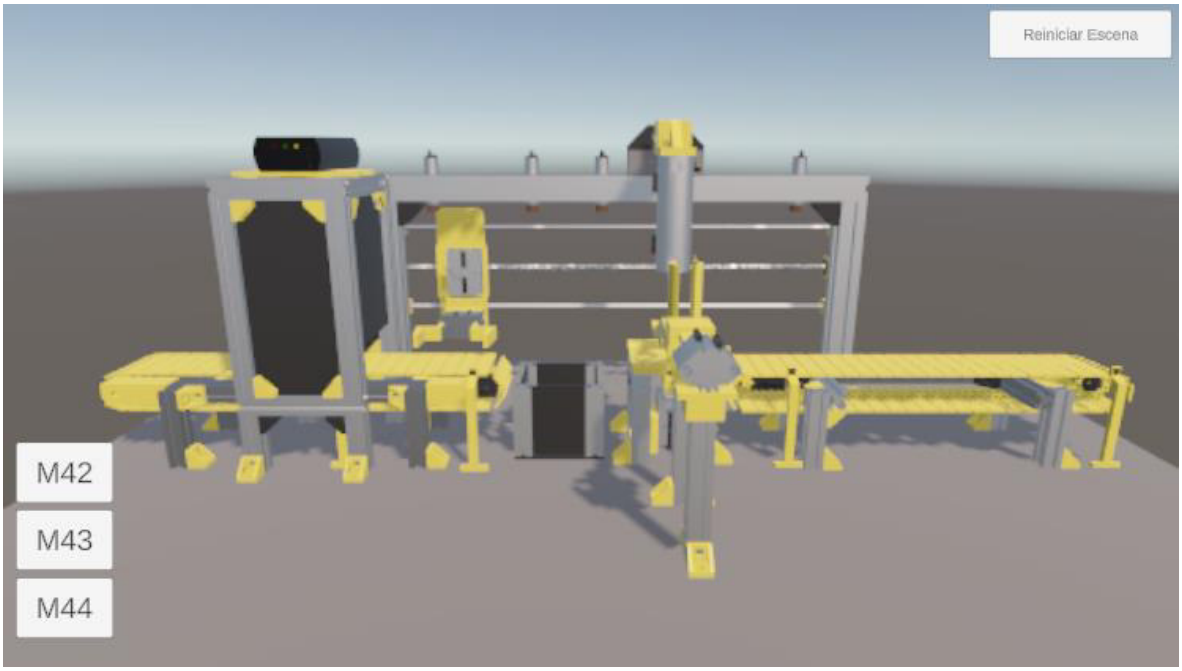

Figura 89: Interfaz gráfica final de la aplicación.

Marcas (bobinas) necesarias para el funcionamiento de la planta virtual

En las rutinas de comunicación entre el PLC y la planta virtual se utilizan un total de veintidós (22) bobinas que son escritas como sensores en el PLC, y diecisiete (17) bobinas leídas como controles por la planta virtual. Haciendo un total de treinta y nueve (39) bobinas involucradas en la comunicación.

La asignación detallada de las bobinas se es presentada en el Anexo A.

5.23 Configuración para el PLC Mitsubishi FX5U

La configuración de la comunicación del PLC Mitsubishi FX5U para su comunicación mediante el protocolo MODBUS/TCP se realiza a través del software GX Works 3.

Dentro del software, se configura el puerto Ethernet para habilitar el protocolo MODBUS, desde la opción “Ethernet Port” en la ventana “Navegation” (navegación). En la pestaña “Module Parameter Ethernet Port” (Parametros del módulo de puerto Ethernet) se asigna la dirección IP del PLC. Para que la comunicación con la planta virtual sea posible, el PLC debe estar en la misma RED de internet.

Por ejemplo, si el adaptador de red que utilizamos en la computadora donde se ejecutará el programa tiene asignada la IP 192.168.1.1, la IP del PLC debe corresponder con 192.168.1.X, donde el valor X es un número cualquiera.

Para este caso de ejemplo, la IP del PLC será asignada como 192.168.1.10. Esto se muestra en la figura 90.

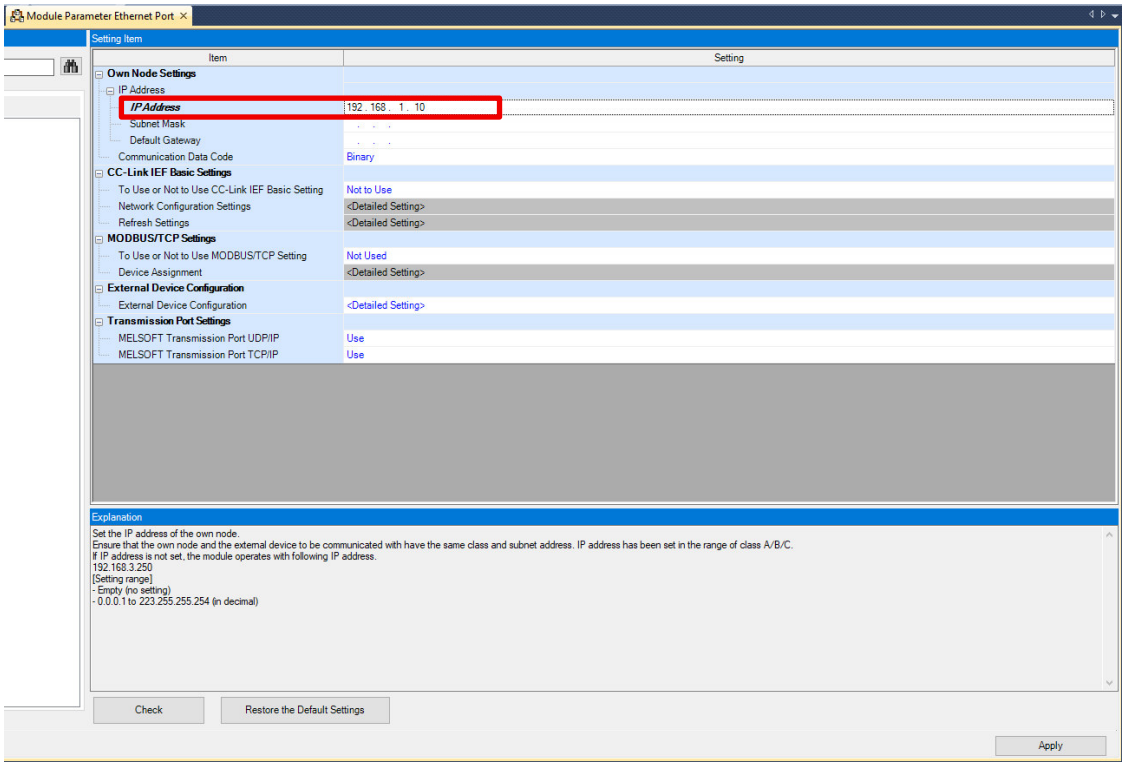

Figura 90: Asignación de dirección IP en GX Works 3 para el PLC Mitsubishi FX5U.

Dentro de esta misma pestaña, en las “opciones detalladas” de la “configuración de dispositivos externosa” (External Device configuration), especificamos el uso del modo de conexión MODBUS/TCP. En esta ventana, podremos observar la dirección IP asignada al PLC, así como el puerto utilizado para la comunicación mediante el módulo MODBUS/TCP, como se muestra en la figura 91. Se aprecia que el puerto utilizado es el puerto 502, mismo puerto establecido en la comunicación de la planta virtual.

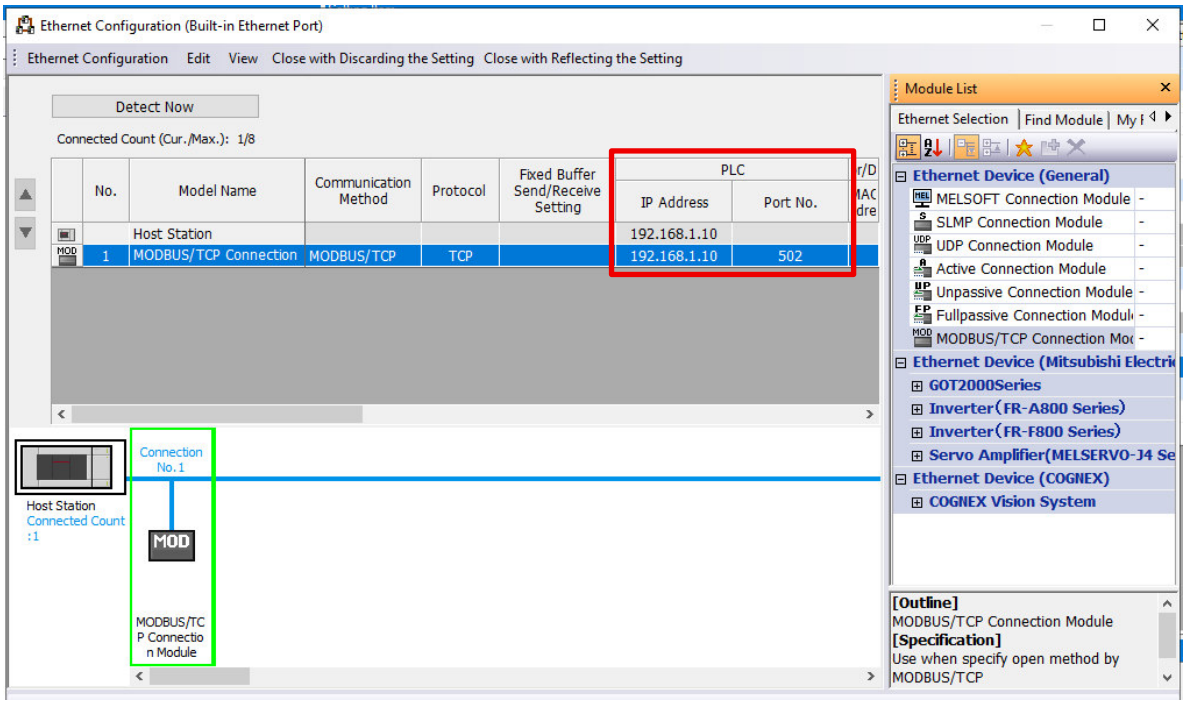

Figura 91: Establecimiento del uso del protocolo MODBUS/TCP para el puerto Ethernet del PLC, así como la visualización del puerto y dirección IP utilizadas para la comunicación.

Hecho esto, para asignar los dispositivos que erán accesibles, mediante la comunicación del protocolo MODBUS/TCP, accedemos a la "Configuración detallada" de "Device Assignment", de la opción "MODBUS/TCP settings", en la ventana Module Parameter Ethernet Port. La ubicación de esta configuración se muestra en la figura 92.

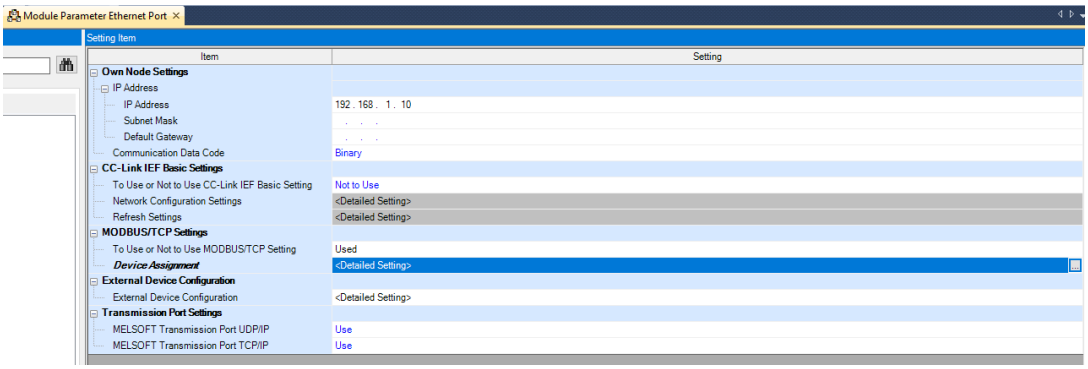

Figura 92: Ubicación de la configuración que nos permite asignar los dispositivos conectados mediante MODBUS/TCP en GXWorks 3.

En la ventana “MODBUS Device Allocation Parameter” se observa la asignación estándar de dispositivos y su configuración. Ya que para el control de la planta se utilizan únicamente dispositivos de tipo M0, y D0 (marcas y registros), en la pestañá “Allocation 1” eliminamos la asignación de dispositivos Y0, y X0, así como su número de marcas asignadas. El la columna Coil, se asigna el dispositivo M0, y “Allocation Points” se asignan 50. En la columna “Holding Registers”, la cantidad de dispositivos asignados D0 de asigna como “1”.

Con esta configuración, se establece el uso de 50 Marcas (M0), y 1 Registro (D0), para la comunicación MODBUS/TCP del PLC Mitsubishi FX5U.

El estado original de los dispositivos asignados “Allocation 1”, y su estado una vez configurado, se muestra en la figura 93.

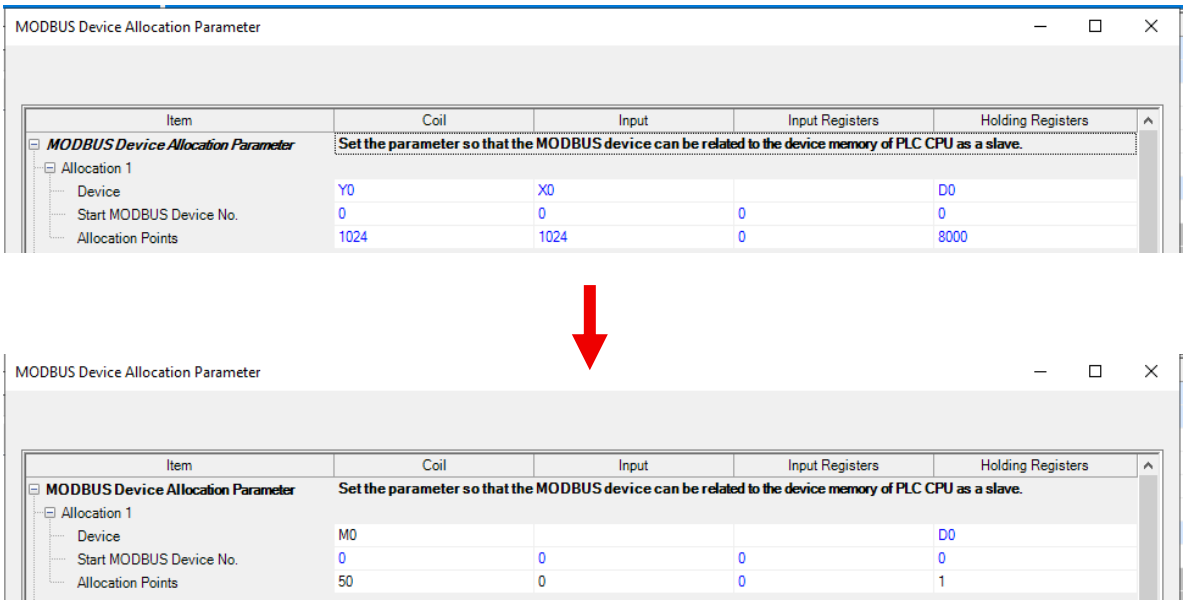

Figura 93: Asignación de dispositivos para la comunicación entre el PLC y la planta virtual mediante MODBUS/TCP.

Esta configuración de parámetros se descarga al PLC para realizar pruebas de comunicación con la planta virtual.

5.24 Pruebas de comunicación

Para realizar la comunicación entre el PLC y la planta virtual, se conecta el PLC a la computadora donde se ejecutará la aplicación a través de un puerto Ethernet.

Con la aplicación ejecutada, se escribe la dirección IP del PLC, y se realiza la conexión con el botón “Intentar conexión” de la ventana “Menú de conexión”.

Podremos observar que el estado se establece como “Conectado” una vez que la comunicación se establece.

Esto se muestra en la figura 94.

Figura 94: Verificación de comunicación entre PLC físico y la aplicación de la planta virtual conectados a través de un cable Ethernet.

5.25 Pruebas de funcionamiento

Para comprobar que el estado de las bobinas en el PLC se ve afectado por el estado de los sensores, hacemos uso de la ventana “Device/Buffer Memory Batch Monitor”, que nos permite monitorear el estado de las bobinas del PLC en tiempo real.

En ella, asignamos M0 en el dispositivo a monitorear, así, veremos que el estado de las bobinas M10, M12, M14, M16, y M18 están activas, respondiendo al estado actual de los actuadores virtuales. Esto se muestra en la figura 95.

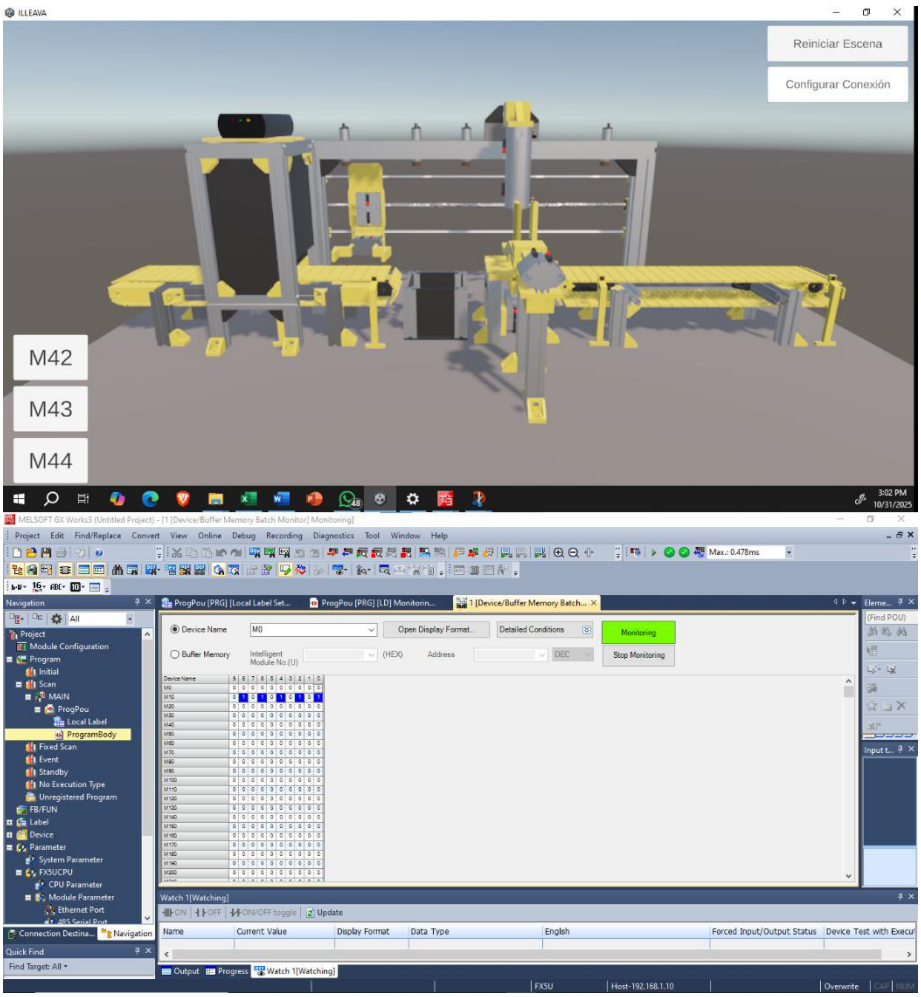

Figura 95: Comprobación de escritura del estado de los sensores en la planta virtual en las bobinas del PLC.

Para comprobar el control de los actuadores utilizando el estado de las bobinas en el PLC, se activa la bobina M25, encargada de extender el vástago del actuador TDA10-10.

En la figura 96 se muestra este cambio de estado en la marca, el resultado en la planta virtual, y el cambio en la detección de los sensores asociados a este actuador en la planta virtual.

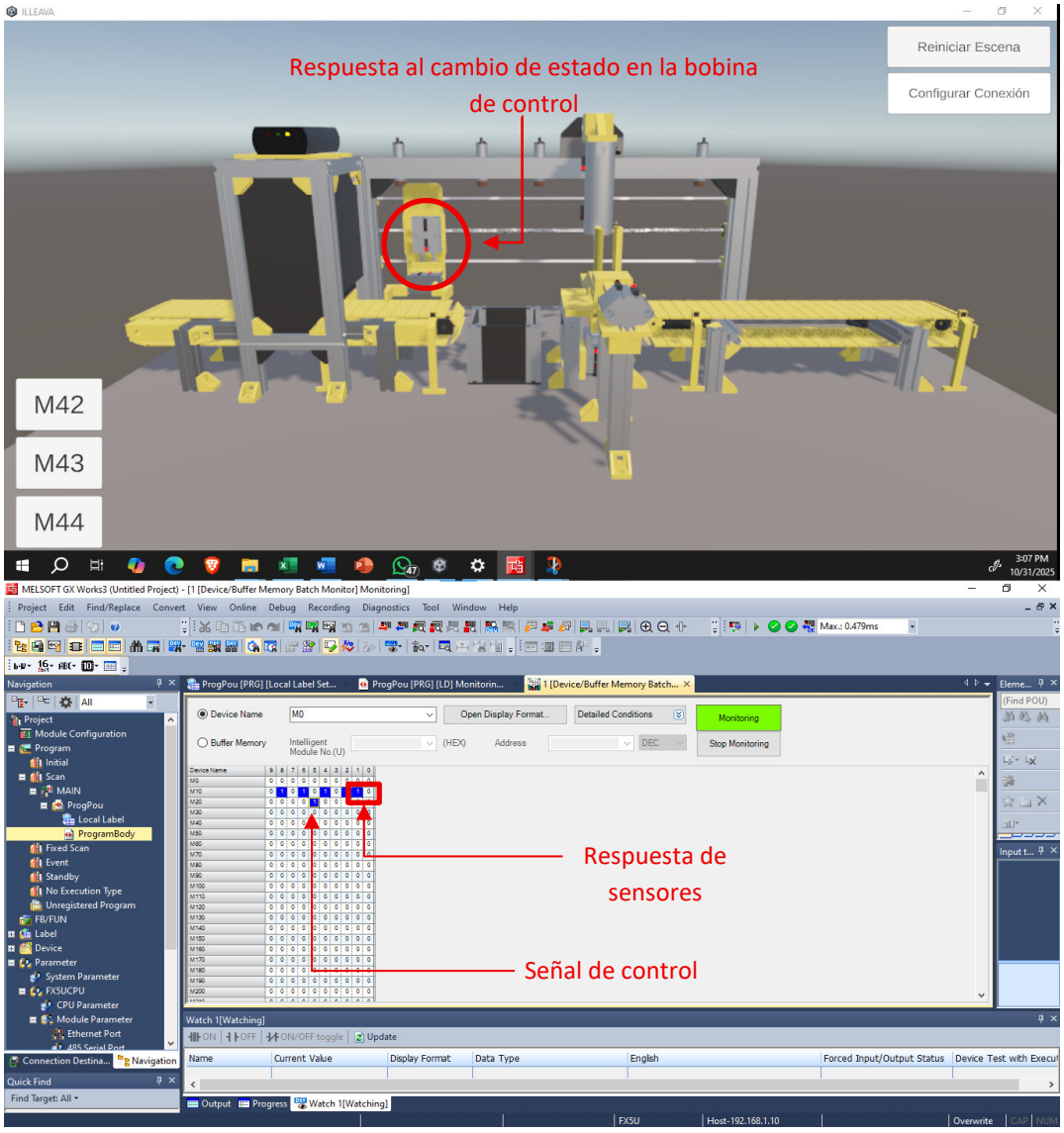

Figura 96: Comprobación de las respuestas de la planta virtual ante las señales de control del PLC.

Con estas pruebas, se determina que la comunicación entre la planta virtual y el PLC s exitosa.

Capítulo 6

Resultados

6.1 Resultados de implementación

La integración de la planta se logró de forma exitosa, implementando todos los subsistemas y optimizaciones requeridas para su funcionamiento.

Los componentes virtuales recrean de forma precisa su comportamiento físico, brindando un nivel de fidelidad alto al comportamiento de una planta física. Así mismo, el aspecto visual de la planta representa de forma correcta los múltiples materiales utilizados en su construcción.

No se presentaron problemas de interacción entre los distintos tipos de componentes utilizados.

El entorno desarrollado para la planta virtual, se muestra en la figura 97.

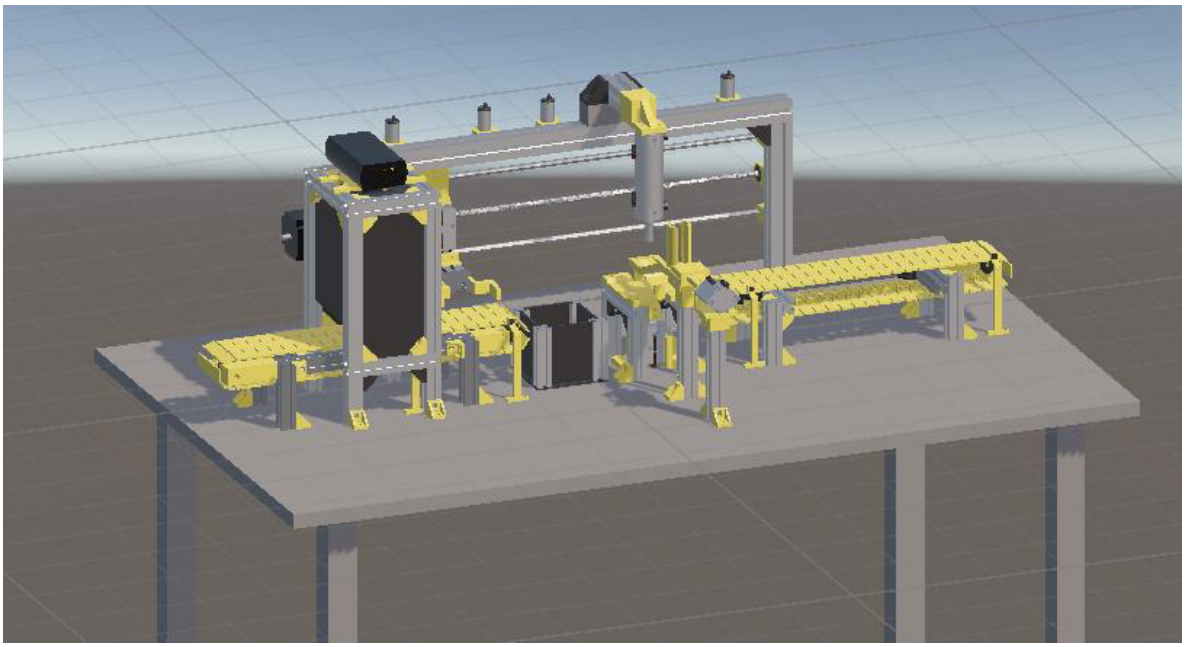

Figura 97: Entorno 3D utilizado para la planta virtual.

Las interfases gráficas utilizadas presentan una disposición clara y funcional, permitiendo al operador interactuar con la planta virtual de forma eficiente.

6.2 Comportamiento funcional de la planta

Los subsistemas desarrollados representan fielmente la función para la cual fueron diseñados.

Durante la simulación se observó el correcto funcionamiento del sistema de las bandas transportadas, al tener la capacidad de desplazar las piezas de un punto en la escena a otra. Esto se muestra en la figura 98.

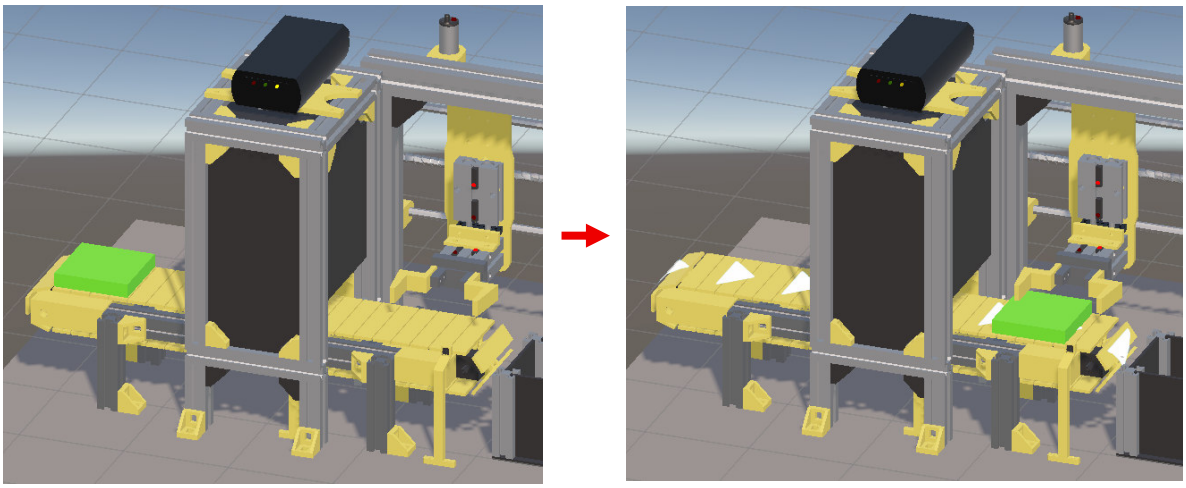

Figura 98: Resultado de desplazamiento de piezas gracias al funcionamiento de la banda transportadora.

También se comprobó el funcionamiento de la simulación del proceso de inspección y validación de piezas.

La inspección de una pieza válida y su impacto en las bobinas del PLC se muestra en la figura 99. En esta, se aprecia que el control del trigger de inspección está activa (marca M36), y el resultado de la inspección reflejado en la marca M20. Además, en la cámara se enciende el LED correspondiente a la inspección de una pieza válida.

La inspección de una pieza inválida se muestra en la figura 100. En esta se observa el trigger de inspección activado, sin embargo, al tratarse de una pieza inválida, la marca M21 está activa, del mismo modo, el indicador LED correspondiente a la inspección de una pieza inválida está activo. Siendo apreciable la diferencia entre la inspección de una pieza válida, y una pieza inválida.

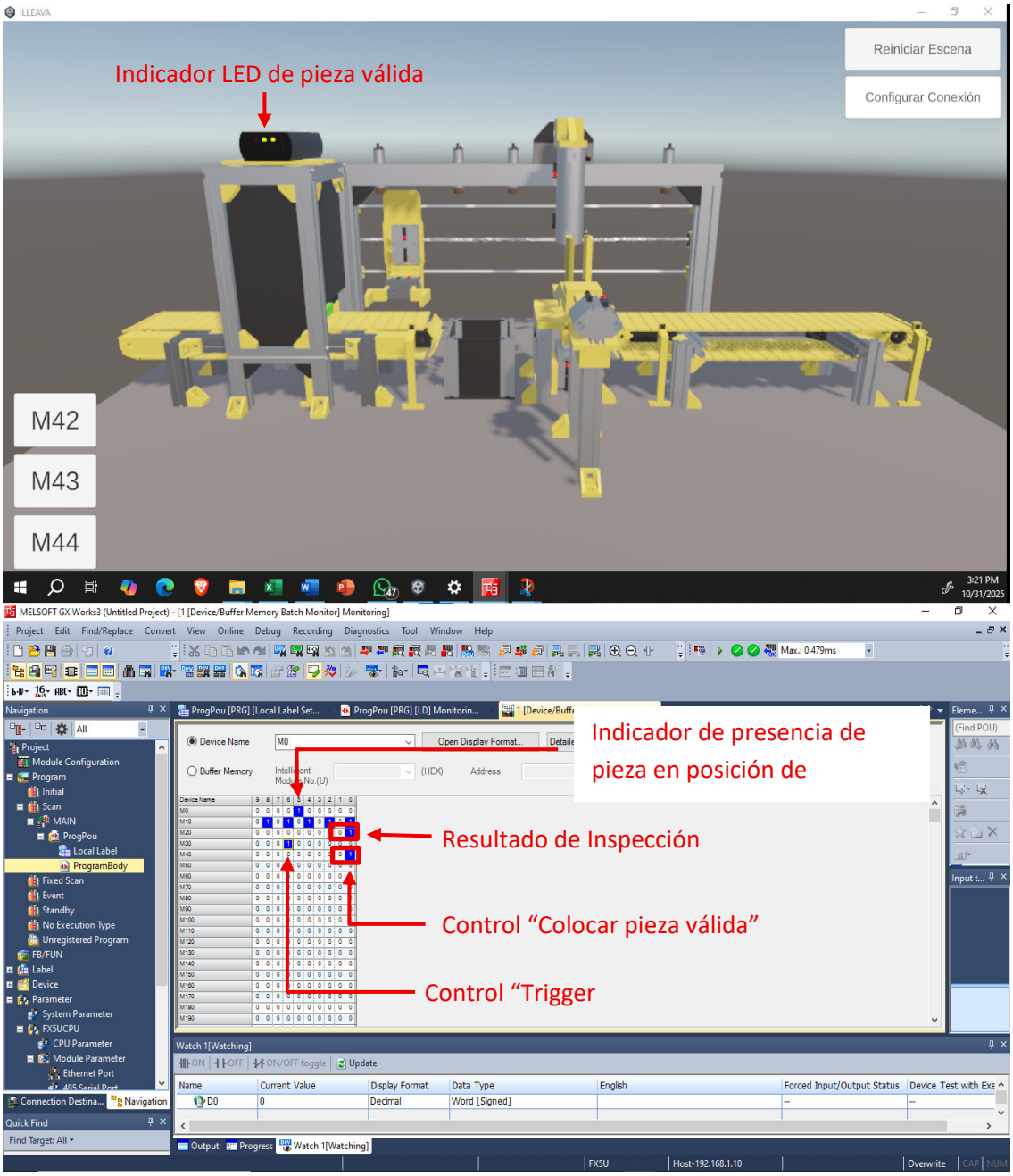

Figura 99: Proceso de inspección de una pieza válida en la planta virtual, y sus efectos en las bobias del PLC.

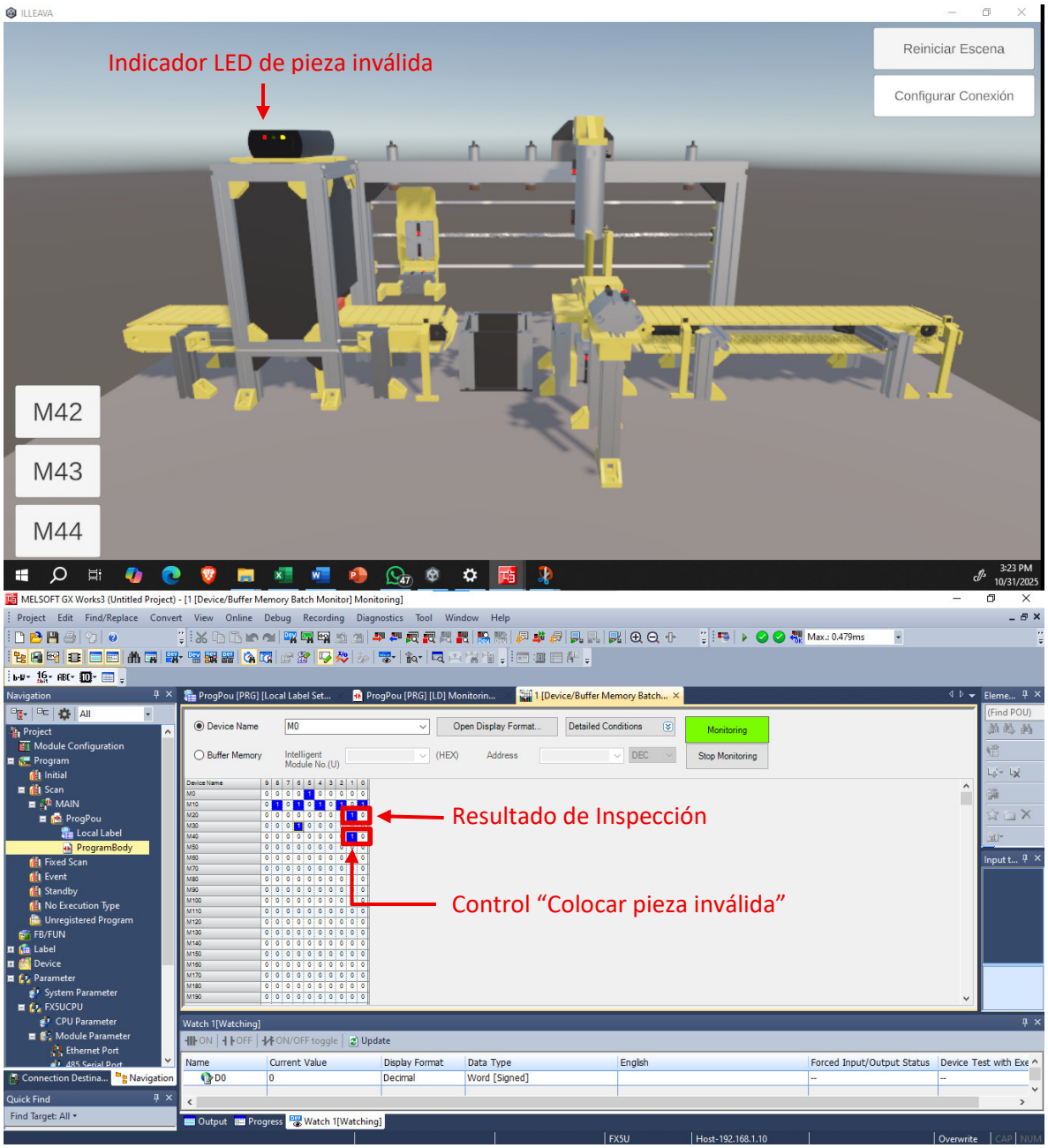

Figura 100: Proceso de inspección de una pieza inválida en la planta virtual, y sus efectos en las bobias del PLC.

Se comprobó el funcionamiento del eje de movimiento, y de este modo, la lectura del registro D0 del PLC. La figura 101 muestra el desplazamiento del eje en dirección “forward”, con este movimiento, se muestra también el correcto funcionamiento de los sensores PR18-5DP virtuales.

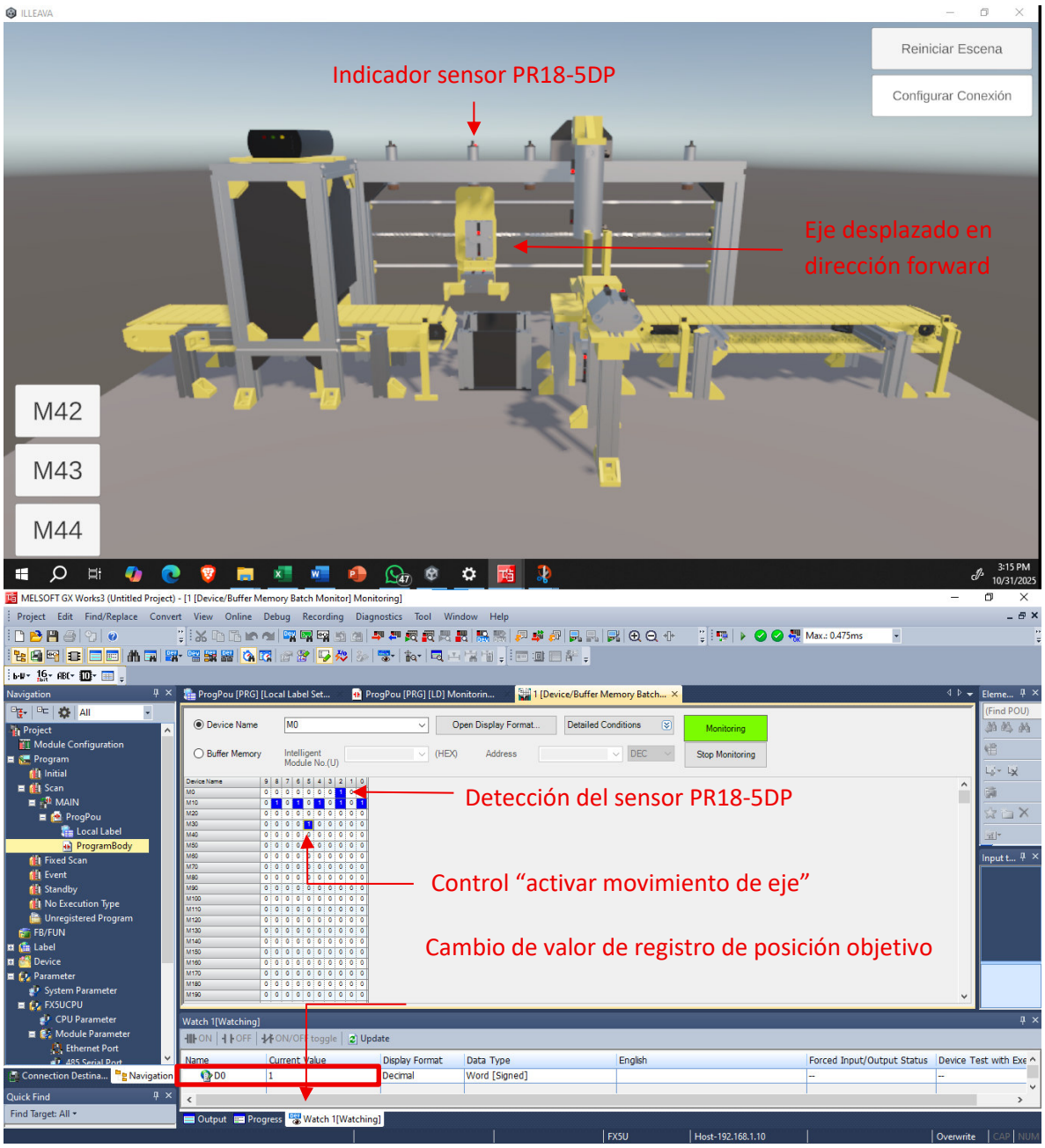

Figura 101: Comprobación de la ejecución el movimiento del eje.

Se validó la capacidad del eje para recoger y transportar las piezas, esta capacidad se muestra en la figura 102.

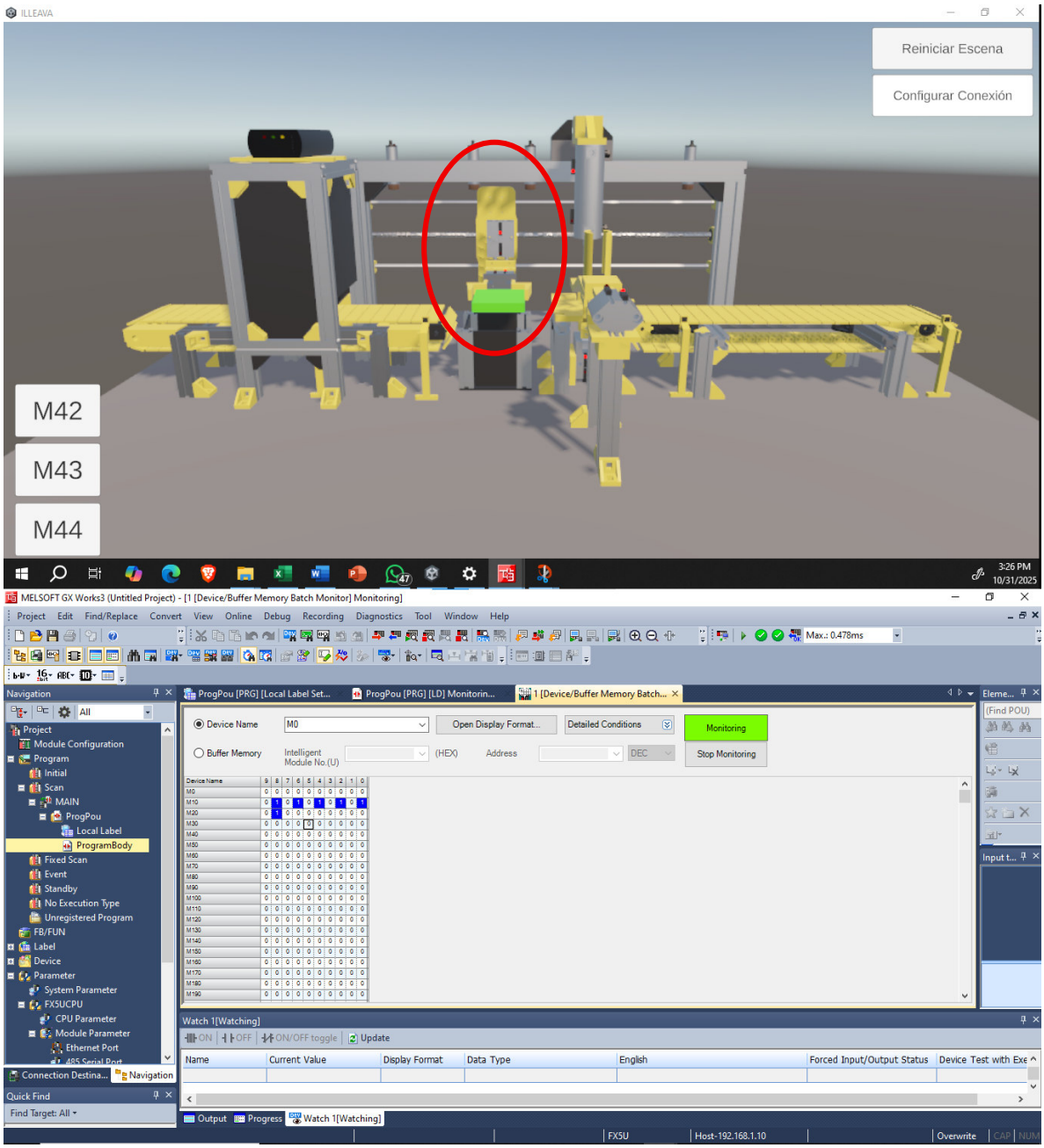

Figura 102: Capacidad del eje para ejecutar la tarea de recolección y transporte de piezas.

6.3 Pruebas de funcionamiento

Para validar le funcionamiento y correcta interacción entre todos los subsistemas desarrollados, se desarrolló la programación de múltiples rutinas en GX Works 3

para que pudieran ser ejecutados por el PLC Mitsubishi FX5U utilizando la planta virtual.

La estructura de los códigos desarrollados se muestra en el Anexo B.

La ejecución de la lógica de la rutina propuesta se realizó de forma exitosa por la planta virtual.

6.4 Resultados de rendimiento

El rendimiento de la aplicación, tras los procesos de optimización utilizados, resulta fluido todos los dispositivos donde se probó. Teniendo una tasa de visualización de 60 cuadros por segundo, y una tasa de sondeo de comunicación MODBUS/TCP de 1000 Hz.

Del mismo modo, la aplicación y el PLC se mantienen en comunicación constante, en tiempo real, sin pérdida de datos, y sin bajadas en el rendimiento de la aplicación.

Capítulo 7

Análisis de Resultados

El correcto funcionamiento de los distintos componentes utilizados demostró que la estructura de desarrollo, programación, y comunicación se realizó de forma correcta, brindando un entorno de simulación confiable para realización de entrenamientos de automatización industrial.

Cada subsistema utilizado cumplió la función para la cuál fueron diseñados, demostrando que los métodos de simulación que se eligieron para estos fueron correctos, permitiendo que la planta virtual sea tan responsiva como una planta física.

Durante las pruebas, la comunicación entre el PLC Mitsubishi FX5U y la planta virtual se mantuvo estable, y sin retrasos ni interrupciones notables. El intercambio de datos utilizando el protocolo MODBUS/TCP con una frecuencia de sondeo de 1000 Hz permite que la comunicación se realice en tiempo real.

Para esto, la optimización de mallas y materiales fue crucial para disminuir la carga gráfica, reduciendo de forma directa los retrasos en la ejecución de los procesos de simulación y comunicación.

Estos resultados confirman que la utilización de un entorno virtual es una alternativa viable para simular rutinas de control industrial, debido al grado de fidelidad que puede alcanzarse con este tipo de métodos. Del mismo modo, se confirma que la arquitectura de comunicación elegida es eficiente para este tipo de aplicaciones.

Capítulo 8

Conclusiones y trabajo a futuro

8.1. Conclusiones:

El desarrollo de la planta virtual compró la posibilidad de integrar un entorno de simulación 3D con un sistema de control industrial real, como un PLC.

EL correcto funcionamiento de los distintos dispositivos utilizados demostró que la estructura de desarrollo, programación, y comunicación se realizó de la forma correcta, obteniendo un entorno de simulación confiable para la enseñanza y el entrenamiento de la automatización industrial. Cada sistema integrado cumplió la función para el cuál fue diseñado, demostrando que los sistemas de simulación empleados fueron adecuados para permitir que la planta virtual tenga un comportamiento fiel a la realidad.

La optimización de las mallas, texturas y materiales permitió mantener un rendimiento estable durante la simulación, permitiendo una experiencia de operación fluida y confiable, permitiendo, a su vez, que no se presenten retrasos, o perdida de datos durante la comunicación mediante el protocolo MODBUS/TCP, teniendo una interacción entre la planta virtual y el controlador en tiempo real.

El proyecto demuestra que el uso de herramientas digitales como Unity 3D puede contribuir al desarrollo de plataformas de enseñanza y entrenamiento en automatización industrial, representando una alternativa viable a los sistemas físicos sin sacrificar tiempos de respuesta, o fidelidad visual y/o funcional.

8.2. Trabajo a futuro.

Como parte del trabajo a futuro, se propone ampliar la cantidad de aplicaciones de control dentro del mismo entorno 3D, para de este modo, tener más de una celda de entrenamiento dentro de la misma aplicación, don distintos objetivos, niveles de complejidad, y dispositivos utilizados, ofreciendo una experiencia más amplia y enriquecedora para la enseñanza de los sistemas de automatización industrial.

Se propone, además, la migración a un entorno de Realidad Virtual (VR), para aumentar el nivel de inmersión de la planta virtual.

Referencias bibliográficas

Blender Foundation. (2024). Blender Manual. Blender. https://docs.blender.org/manual/en/latest/

Dassault Systèmes. (2024). SolidWorks User Guide 2024. Dassault Systèmes. https://help.solidworks.com/2024/english/SolidWorks/sldworks/toc.htm

Haces-García, A., & Zhu, W. (2024). Building an accessible and flexible multi-user robotic simulation framework with Unity-MATLAB bridge. Computers, 13(11), 282. https://doi.org/10.3390/computers13110282

Ibrahim, H. F., Andang, A., & MSN, F. (2024). Data communication between PLC using internet-based Modbus protocol. JIS: Sistem Kendali-Tenaga-Elektronika-Telekomunikasi-Komputer. https://jurnal.untirta.ac.id/index.php/jis/article/view/29172

Jiao, H., Li, H., & Bing, Z. (2018). R&D of PLC teaching simulation training platform based on Unity 3D. En Proceedings of the 8th International Workshop on Computer Science and Engineering (pp. 441–446). https://doi.org/10.18178/wcse.2018.06.077

Khan, L., Choi, Y.-J., & Hong, M. (2022). Cutting simulation in Unity 3D using position based dynamics with various refinement levels. Electronics, 11(14), 2139. https://doi.org/10.3390/electronics11142139

Mitsubishi Electric. (2024a). GX Works 3 Operating Manual. https://www.mitsubishielectric.com/fa/download/software/gxworks3/

Mitsubishi Electric. (2024b). MELSEC iQ-F FX5S/FX5UJ/FX5U/FX5UC User's Manual (Hardware). Recuperado de https://www.mitsubishielectric.com/app/fa/download/search.do?kisyu=%2Fplcf&mode=manual

NModbus. (2024). NModbus Library Documentation. GitHub. https://github.com/NModbus/NModbus

Unity Technologies. (2024a). Unity Manual. https://docs.unity3d.com/Manual/index.html

Unity Technologies. (2024b). Unity Scripting API. https://docs.unity3d.com/ScriptReference/

Unity Technologies. (2024c). Shader Graph Manual. https://docs.unity3d.com/Packages/com.unity.shadergraph@latest

Unity Technologies. (2024d). Shader Reference. https://docs.unity3d.com/Manual/ShadersOverview.html

Villarroel, A., Toapanta, D., Naranjo, S., & Ortiz, J. S. (2023). Hardware in the loop simulation for bottle sealing process virtualized on Unity 3D. Electronics, 12(13), 2799. https://doi.org/10.3390/electronics12132799

Anexos

Anexo A – Asignación de señales de control y sensores a las marcas comunicadas entre la planta virtual mediante MODBUS/TCP, y el controlador.

ACTUADORES

Controles en Acutuadores Neumárticos

Elevador Gripper

Marca	Descripción
M25	Sacar Vástago
M26	Guardar Vástago

Gripper

Marca	Descripción
M27	Abrir Pinza
M28	Cerrar Pinza

Actuador Ensamble

Marca	Descripción
M29	Sacar Vástago
M30	Guardar Vástago

Gripper Ensamble

Marca	Descripción
M31	Abrir Pinza
M32	Cerrar Pinza

Actuador Empuja Balero

Marca	Descripción
M33	Sacar Vástago
M34	Guardar Vástago

Control Movimiento del Eje

Marca	Descripción
M35	Activa movimiento

Control de las bandas

Marca	Descripción
M38	Activar Banda 1
M39	Acivar Banda 2

Control del proceso de Inspección

Marca	Descripción
M36	Trigger de la cámara
M37	Activar Flash

Colocación de Piezas

Marca	Descripción
M41	Colocar Pieza Válida
M42	Colocar Pieza Inválida

SENSORES

Sensores Inductivos

Marca	Descripción
M1	Sensor Inductivo 1
M2	Sensor Inductivo 2
M3	Sensor Inductivo 3
M4	Sensor Inductivo 4

Resultados de Inspección

Marca	Descripción
M20	Pieza Válida
M21	Pieza Inválida

Sensores Colición

Marca	Descripción
M22	Ocurrió una Colición

Senores de Proximidad

Marca	Descripción
M5	Sensor en posición de Inspección
M6	Sensor en posición de Recolección
M7	Sensor en posición de Ensamble
M8	Sensor en posicion de Entrega
M9	Sensor en posición Final

Sensores en Acutuadores Neumárticos

Elevador Gripper

Marca	Descripción
M10	Vástago adentro
M11	Vástago afuera

Gripper

Marca	Descripción
M12	Pinza Cerrada
M13	Pinza Abierta

Gripper Ensamble

Marca	Descripción
M16	Pinza Cerrada
M17	Pinza Abierta

Actuador Ensamble

Marca	Descripción
M14	Vástago adentro
M15	Vástago afuera

Actuador Empuja Balero

Marca	Descripción
M18	Vástago adentro
M19	Vástago afuera

Figura A.1: Asignación de señales de control y sensores comunicadas por la planta virtual mediante MODBUS/TCP al PLC Mitsubishi FX5U.

Anexo B – Códigos desarrollados en GX Works 3 para probar de los dispositivos de la planta virtual, utilizando el PLC Mitsubishi FX5U mediante el protocolo MODBUS/TCP.

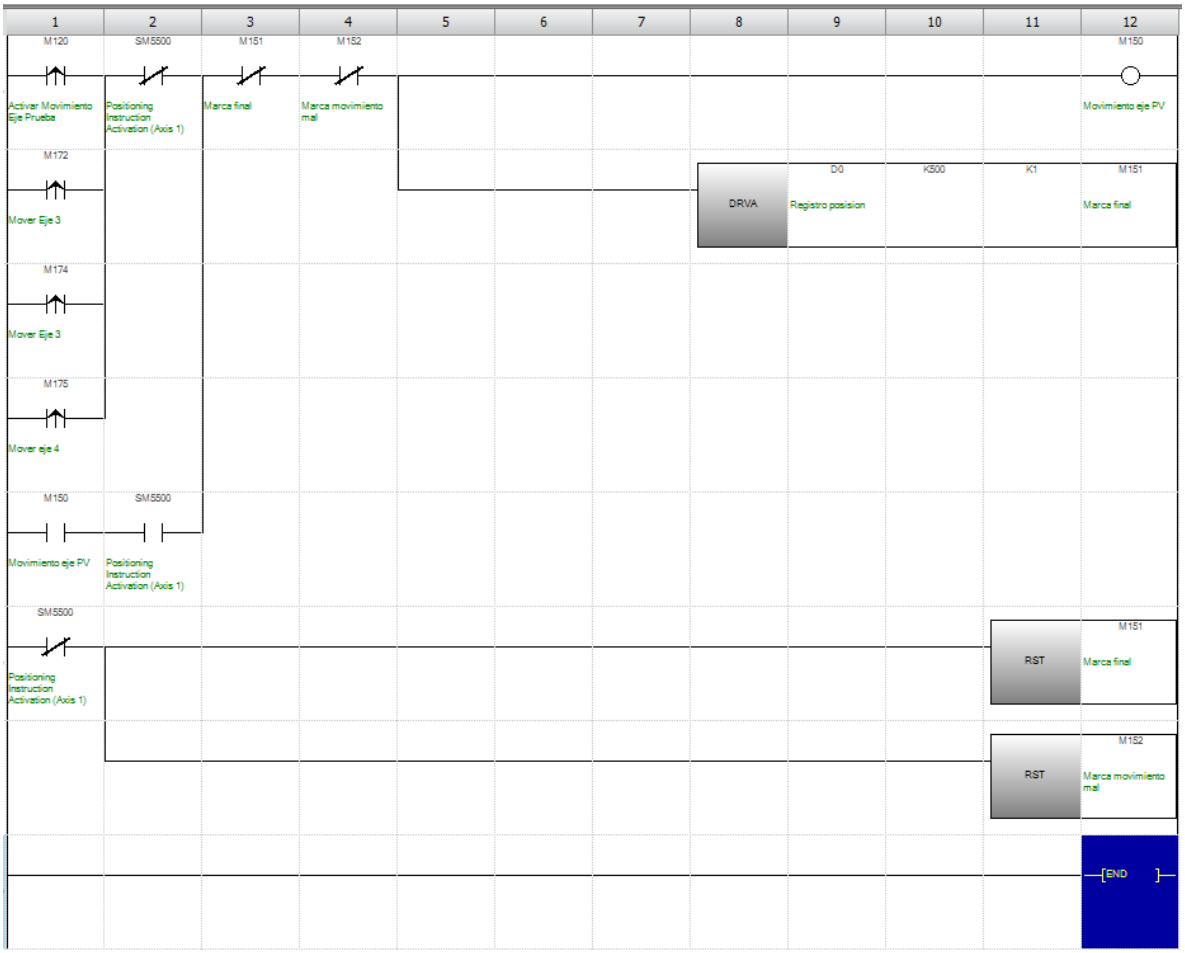

Figura A.2.1: Instrucción utilizada para el control manual del eje de movimiento.

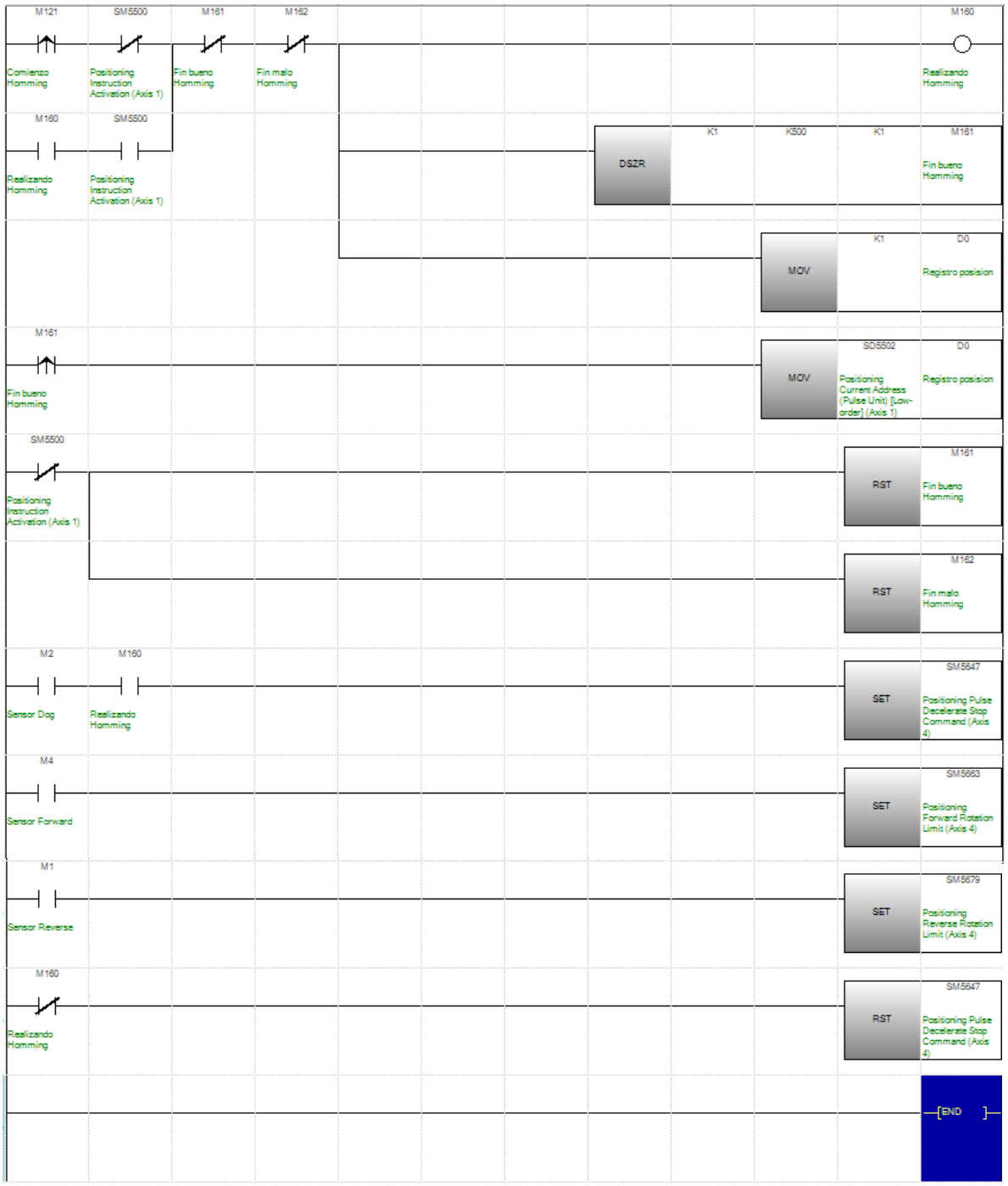

Figura A.2.2: Programa utilizado para la rutina "Volver al origen" del eje de movimiento.

Figura A.3.1: Parte 1 del código de asignación de "entradas" y "salidas" en las marcas utilizadas para la comunicación MODBUS/TCP.

Figura A.3.2: Parte 2 del código de asignación de "entradas" y "salidas" en las marcas utilizadas para la comunicación MODBUS/TCP.

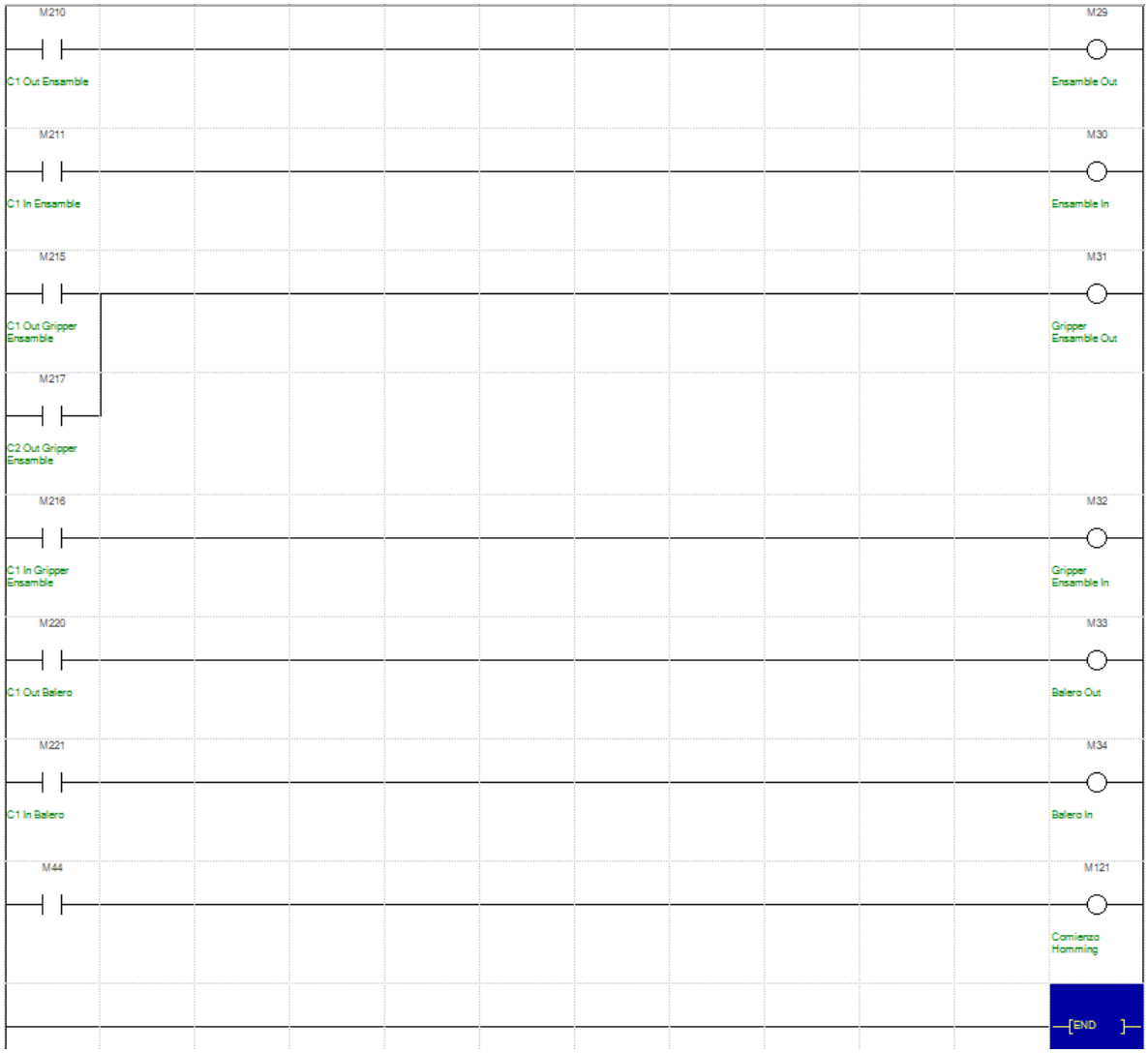

Figura A.2.3: Parte 3 del código de asignación de "entradas" y "salidas" en las marcas utilizadas para la comunicación MODBUS/TCP.