UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Comunicación entre un Controlador de Lógica Programable (PLC) y un entorno virtual

MATERIAL DIDÁCTICO

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Pedro Luis Galindo Roblero

ASESOR DE MATERIAL DIDÁCTICO

M.F. Gabriel Hurtado Chong

Ciudad Universitaria, Cd. Mx., 2025

PROTESTA UNIVERSITARIA DE INTEGRIDAD Y HONESTIDAD ACADÉMICA Y PROFESIONAL
(Titulación con trabajo escrito)

De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado COMUNICACION ENTRE UN CONTROLADOR DE LOGICA PROGRAMABLE (PLC) Y UN ENTORNO VIRTUAL que presenté para obtener el titulo de INGENIERO MECATRÓNICO es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Entidad Académica, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de titulación.

PEDRO LUIS GALINDO ROBLERO
Número de cuenta: 316282264

AGRADECIMIENTOS

A mis padres, Elizabeth Roblero González y José Luis Galindo Galindo, que gracias a su apoyo incondicional y al afecto inmenso que siempre me brindaron, han sido la fuente de energía para mis aspiraciones académicas y personales desde el momento en que nací.

A mi hermana Elizabeth Alicia Galindo Roblero quien siempre apoya mis decisiones y me anima a no rendirme en la búsqueda de mis objetivos; ella es una inspiración de éxito para mí.

A mis asesores M.F. Gabriel Hurtado Chong y M.A. L. Yair Bautista Blanco quienes me han compartido parte de su conocimiento y que me han guiaron durante la planeación y desarrollo del presente trabajo. Cabe mencionar al Dr. Hoover Mujica, quien guio parte de mi camino en la facultad y me condujo con mis asesores.

También a todos mis familiares y amigos que me han apoyado a lo largo de la vida, Antonio, Ariel, Samanta, por creer en mí y confiar en mi trabajo.

Finalmente, pero no menos importante, al Programa de Apoyo a Proyectos para Innovar y Mejorar la Educación, PAPIME con claves PE113324 y PE109325, por los apoyos brindados para la realización de este trabajo.

Contenido

1. Introducción

En el presente trabajo se aborda el proceso de integración de un gemelo digital, así como la importancia de implementarlo en un entorno educativo, teniendo como antecedentes los entornos de realidad virtual que actualmente están disponibles en la página web del Laboratorio de Automatización Industrial. El objetivo principal es demostrar cómo esta tecnología se está convirtiendo en un punto de partida para el desarrollo de soluciones más avanzadas tanto en entornos industriales, como en procesos complejos de automatización que requieran la operación de Controladores de Lógica Programable o PLC, por sus siglas en inglés (*Programmable Logic Controllers*) y la integración de un monitoreo en tiempo real, tal y como la industria 4.0 actualmente lo ha integrado en procesos de manufactura y de la industria aeroespacial, por citar algunos ejemplos.

Para ello, se exploran conceptos clave como los gemelos digitales, sus fundamentos teóricos y su relación con protocolos de comunicación industriales como Modbus, Profinet y Ethernet I.P. Además, se emplean herramientas como Unity, para la visualización 3D interactiva; software CAD (Autodesk Inventor), que permite la digitalización del equipo de trabajo; TIA Portal (en su versión 15.1), para la programación del PLC; y NetToPLCsim como puente de comunicación TCP/IP, entre el entorno virtual y el sistema de control simulado. El uso de NetToPLCsim permite ejecutar en un mismo equipo de cómputo todo el gemelo digital, a pesar de que el uso de un simulador genera controversia en la definición de gemelo digital, este procedimiento responde a la problemática de no poder acceder al equipo físico (PLC) en todo momento.

Inicialmente, la demostración práctica del funcionamiento se centra en dos pruebas: el encendido y apagado de un indicador luminoso, y la extensión y retracción del vástago de un cilindro neumático. Así se evidencia el potencial didáctico de los gemelos digitales, posteriormente, se integran dos pruebas más complejas. La primera, un encendido secuencial de luces (replicando el funcionamiento de un semáforo); la segunda, una secuencia de movimientos a través de la extensión y retroceso de tres vástagos de pistones de doble efecto (rutina de control). Con ello, se incrementa y mejora las herramientas de enseñanza del Laboratorio de Automatización Industrial para impartir las asignaturas de Automatización Industrial y Automatización Avanzada.

2. Antecedentes

La ingeniería es una profesión muy noble que utiliza la ciencia para transformar las ideas en elementos físicos permitiendo a la humanidad tener una mejor calidad de vida y, sobre todo, cómoda. Hoy en día, el desarrollo científico es totalmente abrumador, descubriendo, generando o innovando diversos productos a un ritmo muy acelerado. Por ejemplo, los teléfonos, que prácticamente mes con mes, generan actualizaciones tanto de software como de hardware por parte de los fabricantes. Por ende, la ingeniería también avanza a pasos enormes. Se vive en un mundo de actualizaciones y en el área académica, no ha sido la excepción, por lo que las nuevas generaciones deben integrarse a este mundo de una forma cómoda y sin titubeos, por ello, en la Facultad de Ingeniería (en especial en las carreras como Mecatrónica y Mecánica), constantemente busca implementar en la formación de los ingenieros gran parte de las nuevas tecnologías, teniendo como resultado un perfil ingenieril más atractivo para las empresas, en el momento de solicitar un primer empleo.

Actualmente la industria está migrando al fenómeno 4.0. La tecnología 4.0 construye sus cimientos en el uso de las redes industriales que, a grandes rasgos, se centran en el intercambio de información. Por tanto, el medio por donde se transmite la comunicación de un sistema o el puente por donde se envía datos se convierte en un tema fundamental que se aplica a los diversos dispositivos que: solo envían información (como los sensores); dispositivos que reciben y ejecutan acciones (actuadores como bombas o electroválvulas), y a los dispositivos con capacidad de envío y recepción de datos, como las computadoras, los microcontroladores o los controladores industriales. [1,2]

Para que un sistema tenga comunicación, debe contar con ciertas características que permitan la transmisión de información y asegurar la correcta recepción en un punto remoto. De acuerdo a la naturaleza analógica o digital de la información, la comunicación será del mismo tipo. Los elementos básicos de un sistema de comunicación son el transmisor, el canal y el receptor. El transmisor procesa la señal generada por la fuente de información adecuándola al medio o canal de transmisión (se puede utilizar etapas de codificación y modulación). El canal es el medio físico que cubre la distancia entre el transmisor y el receptor (puede ser cable, fibra óptica o el espacio radioeléctrico). El receptor es el punto de destino o la señal resultante a la salida del canal de comunicación, elimina los posibles efectos producidos por el canal y realiza operaciones inversas al transmisor (decodificación y demodulación) para interpretar la señal de la manera más fiel posible a cómo se transmitió. [3,4]

En la búsqueda por actualizar e implementar mejoras en la calidad de la enseñanza, el Laboratorio de Automatización Industrial de la Facultad de Ingeniería cuenta con un entorno virtual que replica el funcionamiento de una mesa neumática, el cual se conoce como laboratorio de “Neumática Educativa en Realidad Virtual (NERV)” [Imagen 1a, 1b].

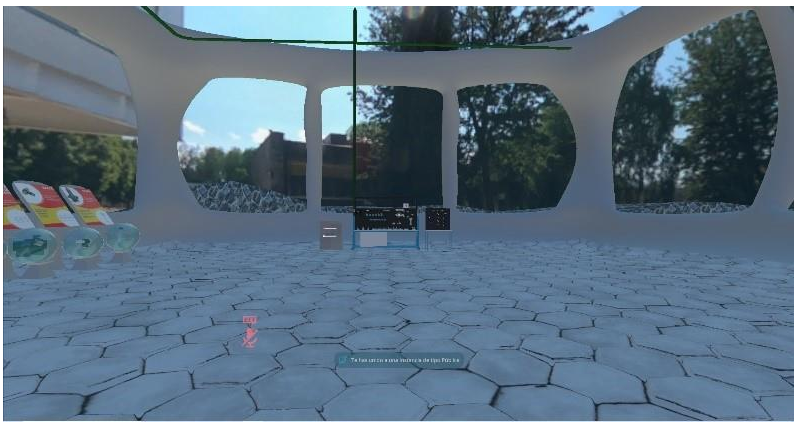


Imagen 1a Laboratorio de Realidad Virtual; VRChat

Imagen 1b Laboratorio de Automatización Industrial

El entorno NERV genera una experiencia interactiva adecuada y funcional con los alumnos, permitiendo desarrollar las prácticas de laboratorio, ejercicios propuestos por sus docentes o practicar sin necesidad de acudir al laboratorio de forma presencial [Imagen 2a,2b].

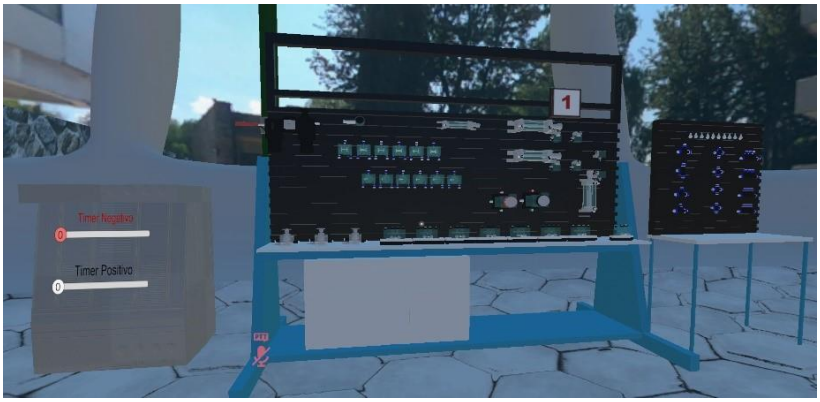


Imagen 2a Mesa neumática virtual, Estación 1; VRChat

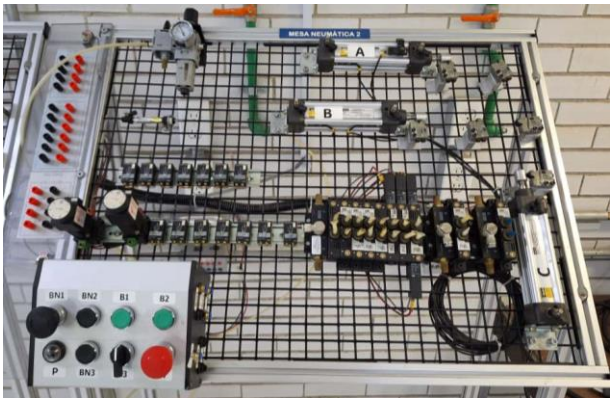


Imagen 2b Mesa neumática; Laboratorio de Automatización Industrial

Los alumnos le consideran una muy buena experiencia, sin embargo, se pierde un poco la habilidad y emoción de utilizar el equipo físico [Imagen 3a, 3b]. Por lo que llega la pregunta, ¿qué pasaría si además de utilizar este entorno enteramente digital, se añade también un controlador físico?, un entorno que no solo opera de forma digital, sino que también se acompañe de un accionamiento en el mundo real. Esto es posible y se encuentra operando en sistemas industriales; es conocido como gemelo digital. Su papel es operar de forma síncrona el desempeño de ambos sistemas (digital y físico) permitiendo el ingreso de información por ambos ambientes, además, el operador puede operar el proceso de forma remota garantizando su seguridad. Actualmente es muy común visualizar estos sistemas en funcionamiento en empresas manufactureras, farmacéutica y aeroespaciales, siendo un medio de trabajo aún reservado y poco diversificado, por lo que en el entorno educativo aún no es tan utilizado. Debido a ello, es esencial que las futuras generaciones de ingenieros conozcan y diseñen estas tecnologías, agregando valor al perfil de egreso de un ingeniero, ampliando sus conocimientos sobre redes industriales.

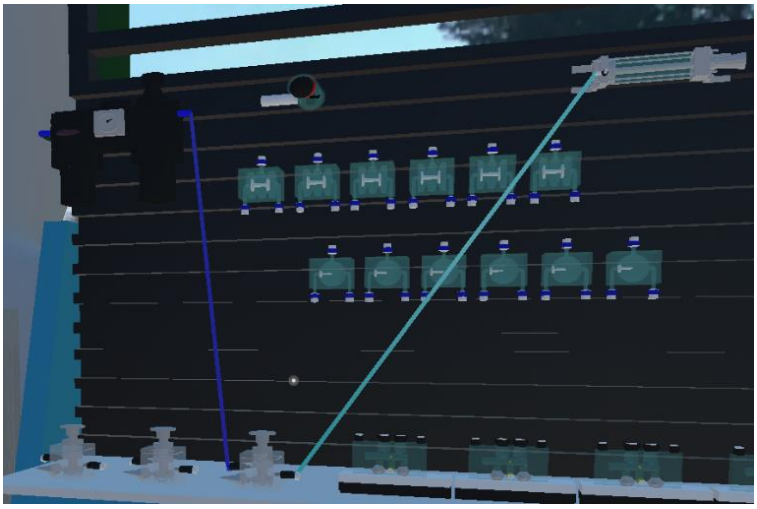


Imagen 3a Desarrollo de práctica 5, Cilindro de simple efecto; VRChat

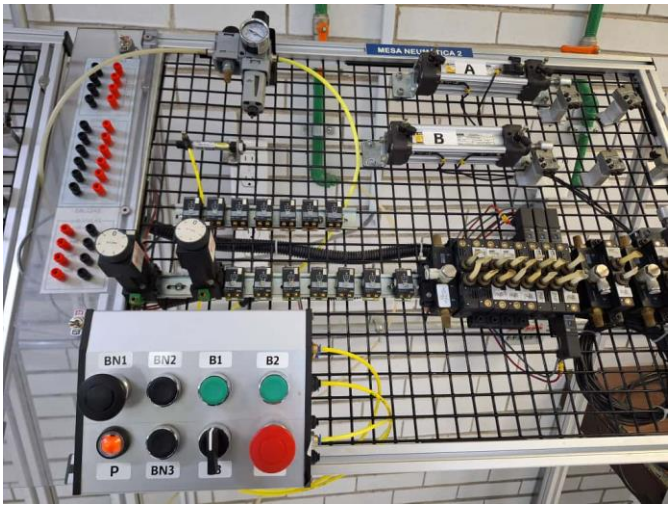


Imagen 3b Desarrollo de práctica 5, Cilindro de simple efecto; Laboratorio de Automatización Industrial

2.1 Objetivos

- Establecer una comunicación funcional entre los entornos de Unity y TIA Portal (Siemens) para enviar y recibir datos de control.
- Utilizar la comunicación entre los entornos de Unity y TIA Portal (Siemens) para el envío y recepción de datos de control en una rutina de proceso.
- Generar un manual de usuario que explique los pasos a seguir para establecer la comunicación entre los entornos de Unity y TIA Portal.
- Vincular un proceso físico con una réplica virtual, de tal forma que pueda observarse un funcionamiento simultáneo.

2.2 Trabajo previo

Los alumnos al cursar la asignatura de Automatización Industrial deben adquirir la capacidad para diseñar un proceso industrial automatizado mediante el uso de sensores, actuadores, controladores de lógica programable y/o neumática; para lograr el objetivo de la asignatura es necesario el uso de los equipos con los que cuenta el laboratorio los cuales son: las estaciones neumáticas (que integran sensores y actuadores) y los gabinetes con controladores (PLC). [5]

Ya que la matrícula de alumnos del Laboratorio de Automatización es grande, ocasiona saturación y poca disponibilidad de horarios para acceder de forma libre para practicar y complementar lo visto en clase. Además, la mayoría de los estudiantes manifiestan el no poder acceder a las horas libres debido a que los horarios disponibles coinciden con las clases de otras de sus asignaturas.

Para resolver lo mencionado, los alumnos pueden practicar la lógica de programación de un PLC, a través de simuladores gratuitos en línea o accediendo de forma remota a máquinas virtuales que cuentan con todo el software que necesitan. Para la parte de neumática, colaboradores del Laboratorio de Automatización Industrial desarrollaron un laboratorio virtual de neumática (NERV [apartado de *antecedentes*]), replicando el funcionamiento físico de las mesas neumáticas [Imagen 1-3]. Sin embargo, ambos sistemas trabajan de forma independiente, por lo que para el tema de electroneumática (parte final del temario de Automatización Industrial), se requiere que ambos sistemas operen de forma conjunta y favorezcan el aprendizaje de los alumnos sin necesidad de limitarlos a interactuar solo con el equipo físico. También hace falta la integración de algunos dispositivos que están en desarrollo. Como en todo proyecto, se deben hacer pruebas y asegurar que los elementos no cuenten con errores, por ejemplo, algunos elementos faltantes son los botones eléctricos, las electroválvulas y los indicadores visuales de los sensores Reed. [6,7]

Imagen 4 Elementos disponibles en el entorno de trabajo (Parte 1).

Imagen 5 Elementos disponibles en el entorno de trabajo (Parte 2).

Imagen 6 Elementos disponibles en el entorno de trabajo (Parte 3).

El proyecto actual cuenta con dispositivos como las válvulas 5/2, los pistones de simple y doble efecto, válvulas "AND" y "OR", botón neumático, unidad de mantenimiento y un indicador piloto [Imagen 4-6]. Los elementos se diseñaron para que se vean translúcidos, así que, al accionarlos, se observa su funcionamiento mecánico interno. Por ejemplo, al accionar un pistón de simple efecto, puede observarse el desplazamiento del émbolo en su interior y la contracción del muelle. Cuando se deja de inyectar aire, el muelle regresa a la posición inicial, el proceso se distingue con claridad, algo que sería imposible de forma física [Imagen 7,8].

Gracias al esfuerzo y dedicación de diversos colaboradores del Laboratorio de Automatización Industrial, ha sido posible el desarrollo de todo el entorno, mejorando detalles e implementando más funciones para lograr un proyecto más innovador y completo.

Imagen 7 Visualización de funcionamiento de un pistón de simple efecto en el laboratorio virtual.

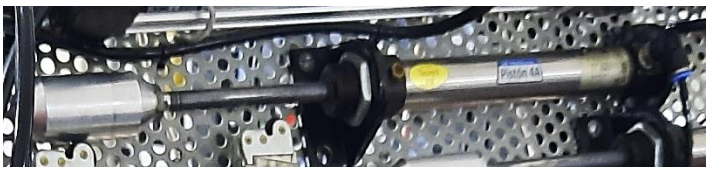


Imagen 8 Visualización de funcionamiento de un pistón de simple efecto físico.

A la fecha, se cuenta con los controladores SIMATIC S7-1200 de Siemens que destacan por tener un mayor espacio de almacenamiento [200 kB] y un protocolo de comunicación mejorado [Profinet], con diferencia de sus antecesores, los S7-200 [4 kB en memoria y protocolo Profibus]. Contar con un mayor almacenamiento en el PLC permite tener múltiples procesos en ejecución, representando una ventaja al generar una conexión entre un entorno digital y un entorno físico. [8,9]

El objetivo es utilizar el protocolo de comunicación nativo del controlador, Profinet, para generar simultáneamente una respuesta tanto en un modelo virtual como en el mundo real. Para lograrlo, se integraron botones, sensores y actuadores. Se espera que el desarrollo de las simulaciones permita a los estudiantes mejorar su formación.

Ahora que se cuenta con los recursos y la infraestructura suficiente, converge la investigación y empeño de integrar a los sistemas previamente descritos. Anteriormente se mencionaba que existe la tecnología en el entorno industrial que replica el funcionamiento de un sistema físico y convive de manera síncrona con un símil digital, que se denomina gemelo digital, del cual se parte como referencia para seguir generando más conocimiento y experiencia, no solo para el Laboratorio de Automatización Industrial, sino para la universidad.

2.3 Estado del arte

En el apartado previo se mencionó que la industria trabaja con gemelos digitales; Siemens, por ejemplo, utiliza un gemelo digital para monitorear el comportamiento de generadores de vapor de recuperación de calor; recopilando parámetros reales, se generan simulaciones que permitan al sistema reducir su actual tiempo de inactividad (calculan una reducción del 70%). [Imagen 9] [10]

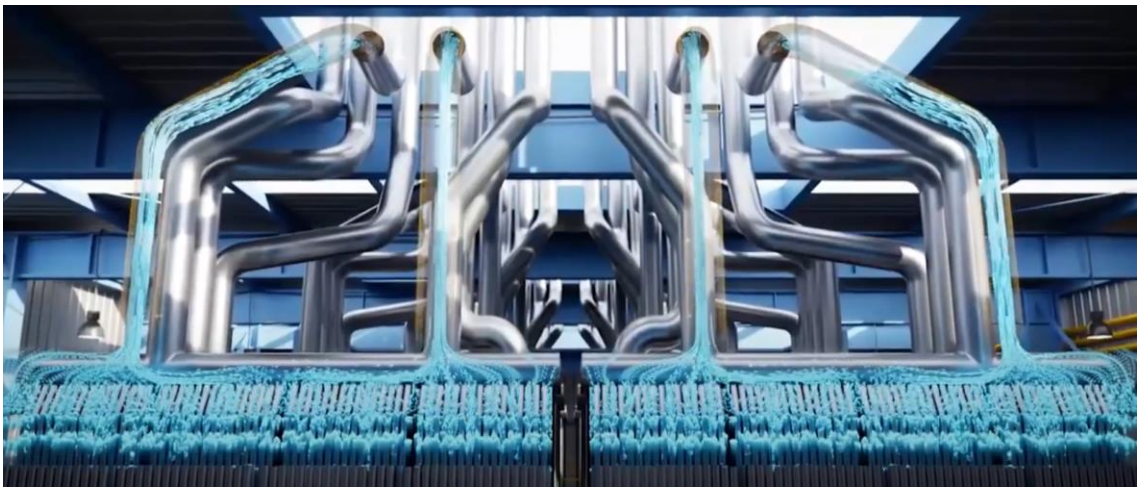


Imagen 9 Simulación con Gemelo Digital del HRSG de Siemens Energy con NVIDIA Modulus y Omniverse

La empresa Siemens, está en una etapa de mejora en sus procesos de comunicación, el objetivo de la empresa es lograr que sus clientes consigan una conectividad en sus productos en todo momento, iniciando con los bocetos e ideas hasta que el producto llegue a las manos del consumidor. Para llevar a cabo estas metas, Siemens introduce la plataforma "Simcenter"; en esta plataforma se integran aplicaciones de simulación predictiva, permitiendo un análisis detallado de los productos que se generan sin la necesidad de producirlos. Esto se logra al evaluar pruebas en entornos completamente digitales, por tanto, se reducen costos a la hora de construir nuevos equipos, al innovar o si se requieren hacer modificaciones a diseños ya elaborados. [11]

Como tal, la apuesta de Siemens es la generación de hilos digitales que interconecten los sistemas que operan en la planta, generando grandes puentes de comunicación y procesamiento de datos. Pero, ¿qué son los hilos digitales de Siemens?, los hilos digitales se plantean como la red de datos, una red que en todo momento se está alimentando y genera bancos de información. La base de datos, disponible en la nube, genera actualizaciones del estado del producto. Los hilos digitales tienen la intención de que en cualquier momento se pueda acceder a la etapa en que se encuentre el producto, para conocer e interactuar con el estado del producto, sin olvidar pasar por el área de producción, de manufactura y de embalaje. Y es aquí en donde entra la tecnología de interés para el proyecto, Siemens desarrolla una virtualización de los equipos que operan en las plantas donde se integran los hilos digitales, generando modelos totalmente digitales, los cuales reciben información del proceso físico, con lo que hacen un procesamiento digital que los hilos digitales reciben. [12]

Como en el ejemplo de los generadores de vapor (Imagen 9), la virtualización de los equipos es un proceso importante para el uso de esta aplicación. Siemens presenta su plataforma "DT Ops" en donde ofrecen a sus clientes *"ejecutables de los sistemas de simulación autónomos, fácilmente adaptables y reutilizables que mantienen fuera de la vista su complicada vida matemática interna"*. [13]

Siemens proporciona a sus clientes gemelos digitales que cumplen con los requisitos de velocidad y precisión, además de permitir que los modelos cuenten con un entorno de simulación integrado. Los gemelos digitales se pueden utilizar directamente, además se pueden adaptar sin necesidad de conocimientos matemáticos expertos. También ofrecen componentes modulares básicos, por ejemplo, componentes que permiten calibrar y actualizar

continuamente los modelos mientras están en funcionamiento, sin duda una gran puesta de valor por parte de esta empresa líder en el desarrollo tecnológico. [Imagen 10] [11,13]

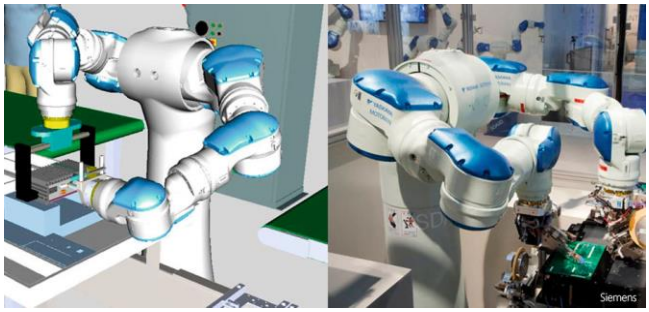


Imagen 10 Gemelo digital (Digital Twin) por Siemens

Rockwell Automation, a través de su software Emulate3D, desarrolla simulaciones que permiten obtener trayectorias óptimas para un robot manipulador que trabaja en sincronía con una línea de embalaje. Rockwell menciona que un gemelo digital permite al usuario ver más información antes en un ciclo de proyecto, permitiéndoles mejorar el diseño, reducir riesgos, aumentar la rentabilidad, entre otras cualidades. [Imagen 11] [14]

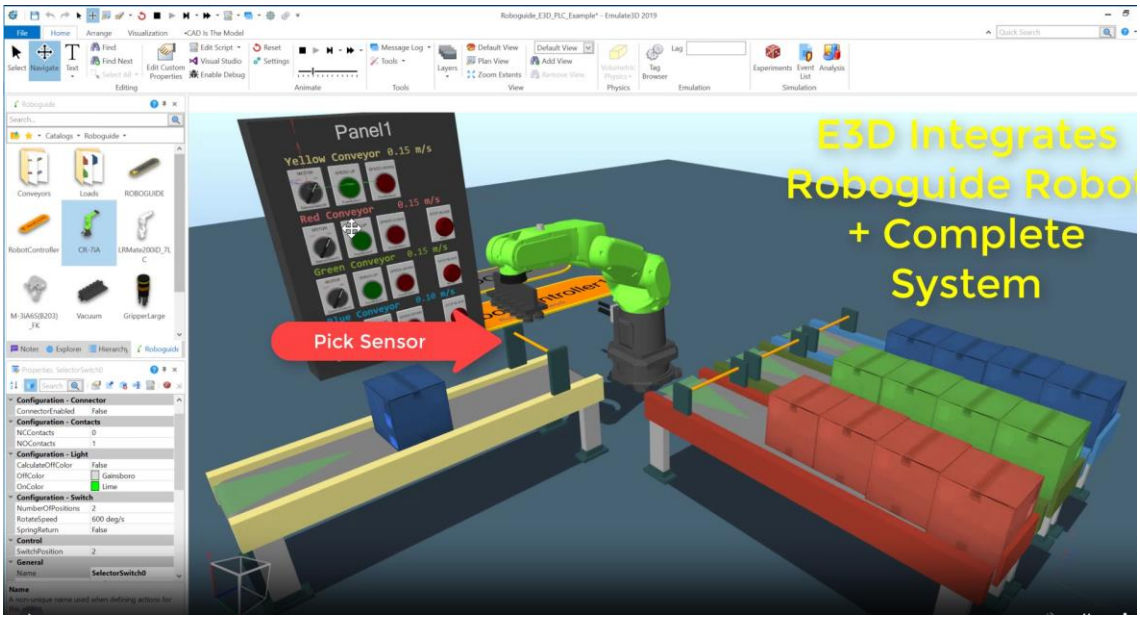


Imagen 11 Simulación en Emulate3D

Rockwell, muestra al consumidor algo no tan similar a lo que plantea Siemens; Rockwell apuesta por los sistemas basados en realidad aumentada. Un sistema de realidad aumentada no es lo mismo que un gemelo digital, dado que la realidad aumentada superpone elementos digitales a elementos del mundo real mientras que los gemelos son entornos completamente virtuales. No obstante, el software Vuforia Engine permite llevar la realidad aumentada a otro nivel, en el cual permite el desarrollo de procedimientos técnicos, como el reemplazo de componentes en un sistema no tan complejo o el mantenimiento de algún motor o máquina paso a paso (Imagen 12). Además, Vuforia permite el monitoreo en tiempo real de los sistemas digitalizados, por ejemplo, los valores de temperatura y de carácter eléctrico en un proceso térmico (Imagen 13). La apuesta de valor de Vuforia es crear interacciones potentes y detalladas de los dispositivos que se desean virtualizar, con ello, ofrece a sus clientes la solución a capacitaciones tan laboriosas y largas, ya que, se orienta a los trabajadores de primera línea a realizar sus actividades de forma rápida y precisa. Para información más detallada, el lector puede profundizar sobre este tema en diversas publicaciones de Rockwell Automation. [14,15,16]

Imagen 12 Realidad aumentada de Vuforia

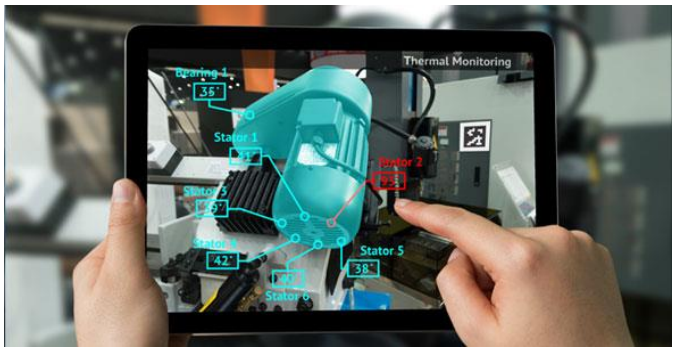


Imagen 13 Visualización en tiempo real de un motor

Por otro lado, la ingeniera Lorena Rodríguez [17] en el Posgrado Interinstitucional De Ciencia Y Tecnología (PICYT) presenta una tesis para recibir el título de maestría en ciencia y tecnología en la especialidad de mecatrónica, donde el problema central es la digitalización de un robot colaborativo con el objetivo de tener un dimensionamiento adecuado (con ayuda de la realidad aumentada) para que los usuarios puedan determinar el espacio requerido además, esto ayuda a que el robot no se vea limitado en funcionamiento y se aprovechen al máximo sus capacidades aunado a esto, se plantea el uso de gemelos digitales para la parte de comunicación y operación remota del equipo, añadiendo que la interface de uso permite solucionar la complejidad de uso para los usuarios y operadores. La propuesta de valor en el proyecto es que el sistema funciona en tiempo real, haciendo que la información se tenga de forma síncrona, además, la interfaz gráfica es fácil de visualizar, proporcionando al operador la posibilidad de programar el robot sin tener los conocimientos tan desarrollados o la habilidad para programar, por lo que se reduce el personal y por ende los costos de operación de la empresa.

Como resultado, se obtiene una mezcla de realidad aumenta y gemelo digital. [5]

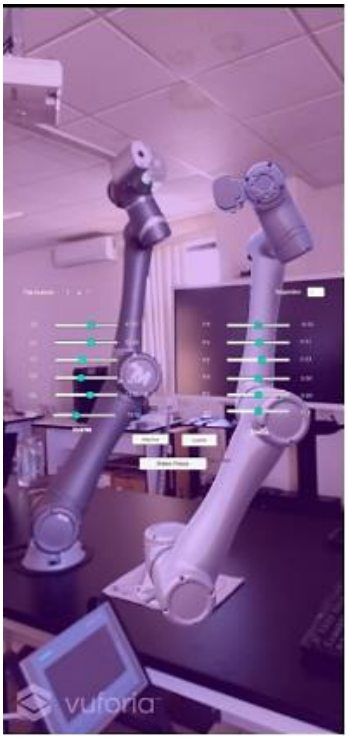


Imagen 14 Menú para seleccionar la posición del robot y establecer rutina.

2.4 Comunicación

En la base de la sociedad humana, la comunicación es algo imprescindible, ya que permite crear un entendimiento entre dos o más individuos para buscar el bienestar y la satisfacción individual y colectiva. Industrialmente, la comunicación recolecta la información de los elementos periféricos para tomar decisiones según una base de datos preprogramada, y así cumplir las metas que el programador le asigna a la máquina o al grupo de estas, por ejemplo, en una embotelladora existen sensores que indican la presencia de una botella, si está vacía o llena; o si algo no funciona bien, los sensores ayudan a detener la línea para evitar cualquier percance. Hasta este punto solo se ha hablado de la importancia que tiene la comunicación, así como un ejemplo, de manera formal y según la RAE, comunicación es:

"Acto, gesto o actitud que permite trasladar mensajes entre los miembros de un grupo social o entre diversos grupos sociales" (Rae, s. f.) [18]

Aplicando este concepto al área de interés, el mensaje del transmisor permite que el receptor pueda procesar la instrucción, ya sea para ejecutar una rutina o solo para el almacenamiento en los históricos. De forma contraria, si no existe este entendimiento, probablemente la acción que se espera sucederá mal o simplemente no se hará. Dependiendo de la operación que se realiza, se puede tener sincronía o asincronía en la comunicación, en ambos casos se debe tener un medio eficiente e informativo, sin importar su simpleza. En sistemas complejos, por ejemplo, en una cervecera, existen sensores (de nivel, temperatura, etc.), motores, bandas, entre otros elementos que alimentan de información a un sistema central, por lo que el circuito debe tener comunicación clara, y la capacidad de atender a las señales con mayor prioridad, como una falta de líquido en los inyectores o botellas atascadas, evitando la pérdida de datos.

Aclarada la necesidad de comunicarse con los elementos de trabajo en una cadena de producción o en general de cualquier proceso, existen diversos protocolos de comunicación industrial. Dentro de estos protocolos se pueden mencionar: "PROFINET", "Ethernet IP" y "Modbus". Las tecnologías mencionadas anteriormente se diseñaron específicamente para cumplir con los estrictos requisitos de los exigentes campos de aplicación; los entornos a los que se enfrenta la comunicación se caracterizan por fuertes interferencias electromagnéticas, tensiones mecánicas, temperaturas críticas y humedad. Por ello, la transmisión de datos debe tener cualidades como ser en tiempo real, tener determinismo y fiabilidad. [19,20]

2.5 Gemelos digitales

Hoy en día este término no es tan desconocido en la rama industrial que apunta a la tecnología 4.0; se encuentra en la industria manufacturera, de salud e incluso en aquellos ambientes de ciudades inteligentes, pero realmente, ¿qué es?, pues bien, el término comenzó a utilizarse en los años 2000, sin embargo, no fue hasta 2012 que la NASA formalizó el término. [21]

"Un gemelo digital es una simulación probabilística multifísica integrada y multiescala de un vehículo o sistema tal como está construido que utiliza los mejores modelos físicos disponibles, actualizaciones de sensores, historial de la flota, etc., para reflejar la vida útil de su gemelo correspondiente.". [22]

La definición anterior es en extremo técnica y poco entendible para el desarrollo de este escrito, por lo que la definición planteada por Madni es más pertinente:

"Un gemelo digital es una instancia virtual de un sistema físico (gemelo) que se actualiza continuamente con los datos de rendimiento, mantenimiento y estado del sistema, a lo largo del ciclo de vida del sistema" [23].

Con las definiciones anteriores se tiene que, los gemelos digitales comprenden dos interpretaciones de un mismo sistema, uno de características virtuales y el otro de elementos físicos, ambos funcionan a la par (por ello el término gemelo); este conjunto de sistemas comparte una actualización de datos en tiempo real permitiendo una retroalimentación. Si existe una señal digital, el sistema físico la interpretará y ejecutará la acción pertinente, sucede el mismo impacto en el entorno digital si la señal es física. Esto funciona de manera continua a lo largo del periodo de trabajo del equipo. [24]

El aspecto crítico que se destaca es la existencia de una transferencia de datos entre el sistema físico y su contraparte digital, estableciendo una comunicación síncrona. A lo largo del presente escrito, se detalla cómo se emplea el protocolo nativo del hardware para habilitar dicho sistema síncrono. Esto se debe a que un gemelo digital no se define únicamente por el modelado visual de los sistemas o la representación de su funcionamiento, sino por la capacidad de obtener y procesar datos en tiempo real. Contar con un gemelo digital funcional permite aumentar la seguridad en la operación de equipos industriales. Por ejemplo, al usar una prensa, el operador ya no tendría que activarla directamente desde los elementos integrados en el equipo, evitando exponerse a posibles fallas que pudieran derivar en un accidente. Gracias al gemelo digital, se controlaría la máquina de forma remota desde un cuarto de control, contando en todo momento con información en tiempo real sobre el estado del proceso. Esto no solo incrementa la seguridad operativa, sino que también mejora significativamente las condiciones laborales del operador.

2.6 PROFINET (PROcess FIeld NETwork)

Desarrollado por la organización PROFIBUS & PROFINET International (PI), esta tecnología pretende conectar sensores, motores u otros dispositivos electrónicos; el protocolo permite la conexión entre los dispositivos sin importar el fabricante desde el nivel de campo hasta el nivel de gestión, generando comunicación homogénea basada en la IEC 61158. Actualmente el protocolo ha sido optimizado con herramientas de diagnóstico, detección de equipos y en caso de fallas, una recuperación rápida, por tanto, se encuentra ganando mercado en la industria moderna. [25]

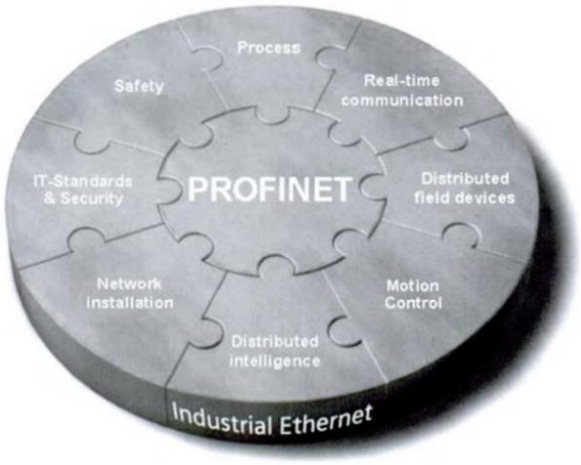


Imagen 15 PROFINET como un concepto modular.

PROFINET es una tecnología de red de campo (buses), por lo que permite una integración perfecta de los sistemas Fieldbus para la comunicación entre controladores y equipos industriales; tiene sus bases en Ethernet y conecta a los dispositivos en sistemas automatizados con alta velocidad de comunicación (de hasta 100 Mbps con conexión Full Dúplex). [Imagen 15] [25,26]

El protocolo PROFINET es de los más versátiles, intercambia datos de forma precisa y rápida (la velocidad varía dependiendo la aplicación, desde segundos para instrumentos de procesos hasta milisegundos para dispositivos

de control y sincronizaciones de movimiento), funciona mediante un cable Ethernet o fibra óptica, admitiendo protocolos flexibles, por ejemplo, TCP-UDP/IP, SNMP, LLDP y DHCP, se añade que permite la comunicación inalámbrica Bluetooth y WLAN. [25,26]

La arquitectura es escalable en tiempo real y con el mismo medio de comunicación (el cable Ethernet), gracias al uso de múltiples protocolos y topologías que permiten comunicaciones bidireccionales entre los dispositivos enlazados en la red, computadoras, servidores y equipos industriales. La escalabilidad se logra mediante la arquitectura modular, a su vez, esto permite que los usuarios puedan acceder a la visualización de los otros dispositivos sin intervenir en las funciones de los equipos. La arquitectura modular permite a los usuarios agregar más dispositivos o módulos (si se requieren), también, permite actualizar dinámicamente la red para mejorar el rendimiento del sistema. Adicionalmente, al ser un sistema Ethernet, se tiene acceso remoto sin interrupción. [Imagen 16] [26, 27]

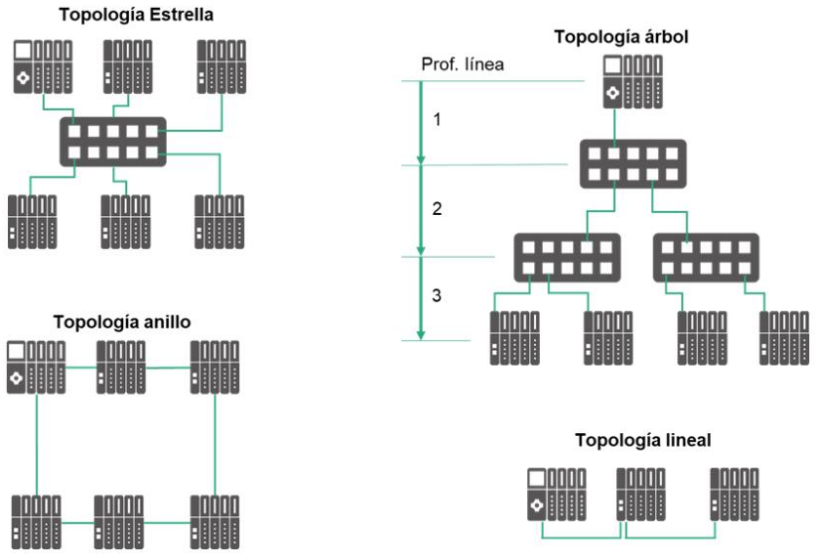


Imagen 16 Arquitecturas PROFINET.

La red PROFINET puede estar integrada por múltiples estaciones de trabajo, se integran desde dispositivos de entradas y salidas (E/S) digitales, actuadores neumáticos, sistemas de visión, escáneres, sensores, electroválvulas, entre otros. Estos elementos se clasifican en tres grupos, según su función: [Imagen 17]

+Controladores: Encabezan al sistema, ejecutan un programa de automatización e intercambian datos con otros elementos. Los controladores asignan los datos a los dispositivos PROFINET, con ello los dispositivos ejecutan su funcionamiento, regresan señales al controlador y este vuelve a procesarlas, los datos de respuesta son usados en el programa de control. El equipo debe soportar el intercambio de datos cíclicos y acíclicos, el primero hace referencia a los datos controlador–dispositivos y dispositivos–controlador; el segundo, intercambia datos de diagnóstico y de configuración. También debe poder procesar la conexión.

+Dispositivos: Sensores o actuadores conectados al controlador mediante PROFINET.

+Supervisores: Elementos de monitoreo, puesta en marcha o análisis de diagnóstico, como una HMI o una PC. Recopilan datos de diagnóstico (lectura y escritura) ya sean internos o proporcionados por un dispositivo.

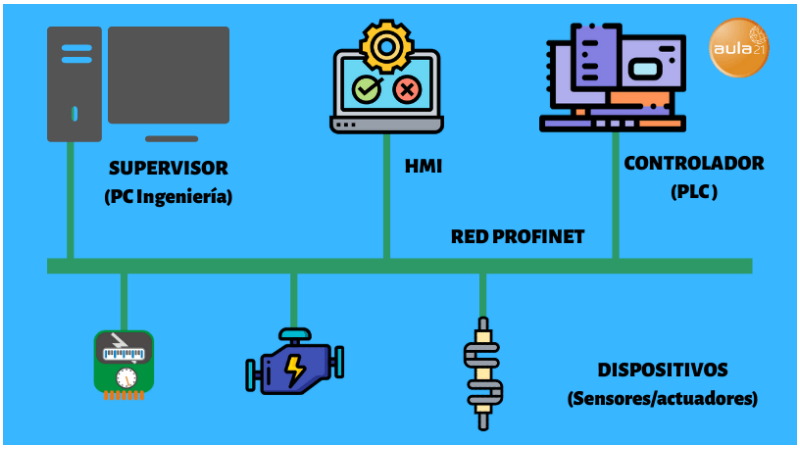


Imagen 17 Diagrama básico de PROFINET

Es importante mencionar que PROFINET puede utilizar tres canales de comunicación, los cuales son:

- TCP/IP: Para funciones no deterministas, como transferencia de datos a un sistema de Tecnología de la Información (TI), parametrización o transmisiones de video y audio.
- PROFINET en tiempo real (PROFINET RT): Se omiten las capas TCP/IP, con ello se logra un rendimiento determinista de las aplicaciones de automatización. Catalogada como una solución basada en software, idónea para aplicaciones de I/O típicas y control de movimiento.
- PROFINET en tiempo real isócrono (PROFINET IRT): Permite la priorización de señales y la conmutación programada dando como resultado una sincronización de alta precisión para aplicaciones como el control de movimiento. Se pueden lograr ciclos en el rango de fracciones de milisegundos, sin embargo, se requiere compatibilidad de hardware. [28,29]

2.7 Ethernet IP (Ethernet Industrial Protocol)

Se introdujo en el año 2001 a la industria, de forma similar a PROFINET, Ethernet IP es un bus de campo, que ofrece una alta velocidad de transferencia de datos, control, configuración y recolección de datos simultáneos en una red, además, es compatible con el protocolo Ethernet. En la industria es mayormente encontrado en aplicaciones de control dada su velocidad de 10 Mbps o 100 Mbps, y hasta los 1500 bytes por paquete. Es importante mencionar que este protocolo se apega completamente a los estándares de Ethernet, utiliza capas físicas, de enlaces de datos, de red y de transporte estándar de Ethernet, con ello, puede soportar un número ilimitado de nodos, sin embargo, se debe ser cuidadoso con este beneficio dado que se debe evitar la latencia para que la comunicación sea en tiempo real. [30]

Ethernet IP realiza una clasificación de los nodos de acuerdo con los dispositivos preestablecidos. El protocolo tiene cimientos en el Protocolo de Control e Información (Control and Information Protocol – CIP), gracias a esto el sistema es robusto y seguro, desde la planta hasta la red central de la empresa. CIP organiza los nodos, define los accesos, atribuciones y extensiones. Además, se utiliza una sola herramienta para configurar los dispositivos CIP en diversas redes desde un único punto de acceso sin la necesidad del software propietario. Para información más detallada, el lector puede profundizar sobre este tema en diversos artículos acerca de Control and Information Protocol. [30,31]

Ethernet funciona a través de la administración de sus nodos respetando la capa de enlace de datos en la cual se define el método de acceso a los medios. Los enlaces semidúplex (topologías de bus o estrella), utilizan el sentido de portadora y acceso múltiple con detección de colisiones (CSMA/CD). Por tanto, se permite que múltiples nodos tengan el mismo nivel de acceso a la red, todos los nodos de la red Ethernet monitorean continuamente las transmisiones en los medios. [Imagen 18]

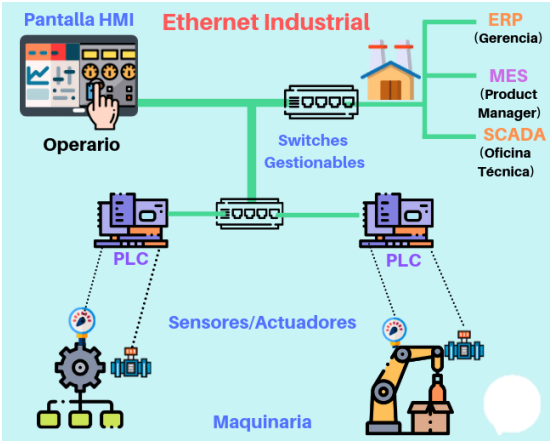


Imagen 18 Diagrama básico de Ethernet Industrial

La cuestión es, ¿cómo se hace la transmisión de datos? Pues bien, si un nodo necesita transmitir, espera hasta que la red esté inactiva, entonces comienza a transmitir. Mientras el nodo transmite la información, cada nodo monitorea su propia transmisión y compara lo que está 'oyendo' con lo que está tratando de enviar.

Ahora, si dos nodos comienzan a transmitir al mismo tiempo, las señales se superponen, haciendo que las señales originales se arruinen, por lo que ambos nodos notarán una señal diferente a la que están tratando de enviar. Esto es conocido como 'colisión'. Si hay una colisión, cada nodo deja de transmitir y solo intenta retransmitir después de un retardo preestablecido, para cada nodo es diferente. Por ello es fácil la adición o eliminación de nodos de una red. Al conectar un nodo nuevo, este 'escuchará' y transmitirá cuando la red esté disponible. [32]

Ethernet puede hacer uso de hardware comercial, esto es, enrutadores, conmutadores, cableado, entre otros, por lo que ofrece una instalación a menor precio, también elimina la necesidad de puertas de enlace adicional, lo que deriva en un menor tiempo de la puesta en servicio, finalmente, ofrece un servicio de diagnóstico a través de un servidor web integrado. [33]

2.8 Modbus

Modbus es un protocolo que generalmente se usa para comunicación en red tipo SCADA entre los dispositivos, si se implementa como un servidor de gran magnitud, puede manejar un Controlador de Lógica Programable (PLC) o un Controlador de Automatización Programable (PAC), los cuales operan toda la línea de automatización. Fue desarrollado en 1979 por Modicon (hoy Schneider Electric) como protocolo de transferencia de datos en una capa serial, sin embargo, se ha expandido para incluir implementaciones a través del protocolo serial, TCP/IP y UDP (User Data Protocol) en redes modernas para sistemas de monitorización y telecontrol. [34]

La estructura de este protocolo de comunicación reside en la arquitectura maestro – seguidor. El maestro transmite una solicitud al seguidor y espera por la respuesta, por su parte, el seguidor envía los datos solicitados. El maestro tiene un control completo sobre el flujo de la información. Es importante destacar que la transmisión de la información entre varios dispositivos conectados es a un mismo bus, sobre el bus de datos solo existe un maestro (Master) y varios seguidores (Slaves), ver Imagen 19. [34, 35]

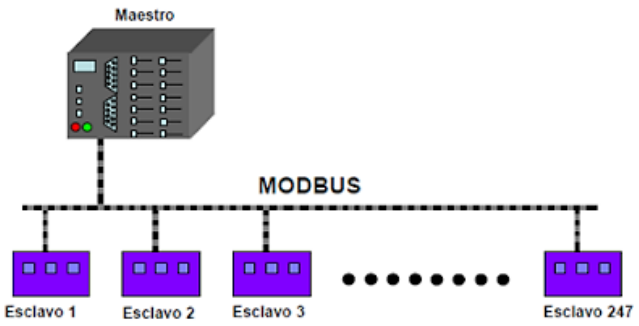


Imagen 19 Relación Solicitud-Respuesta de Maestro-Seguidor en los dispositivos Modbus

Modbus trabaja mediante la solicitud de datos a sus seguidores, sin embargo, esta solicitud se distribuye en capas. Una de las capas, considerada como la primera, es el ADU (Unidad de Datos de la Aplicación), se dice que el ADU se puede interpretar como el tipo de Modbus que se utiliza. Existen tres variantes, ASCII, RTU (Unidad de Terminal Remota) y TCP/IP. [Imagen 20]

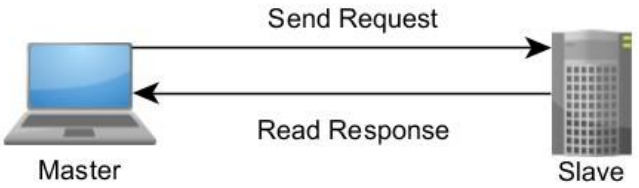


Imagen 20 Protocolo Modbus

RTU y ASCII son formatos seriales antiguos y la diferencia entre ambos radica en que RTU utiliza una representación binaria compacta, por otro lado, ASCII envía datos como cadenas de caracteres ASCII. TCP es un formato moderno, permite un manejo eficiente de solicitudes y respuestas Modbus en software, contiene un sistema de conexiones e identificadores dedicados para cada solicitud.

La selección del tipo de ADU, depende completamente de la red física deseada, el número de dispositivos que estarán en la red y los ADUs que puedan soportar los dispositivos maestros y seguidores en la red. Dentro del ADU existe una unidad de datos de protocolo (PDU), cada PDU contiene un código de función y datos asociados. Cada uno de estos códigos tiene una respuesta bien definida. Como en cualquier situación, pueden existir errores, por lo que Modbus define PDU específicos para estos casos, con esto, el maestro puede saber qué es lo que ocurrió. Dependiendo del controlador que se utilice, estos traducen este código a un formato que tenga sentido para el lenguaje o la aplicación de uso. Por ello el PDU se considera el núcleo de Modbus. [35,36]

Modbus administra los datos de una manera flexible y simple. Originalmente, el protocolo soportaba únicamente dos tipos de datos; valores booleanos y enteros sin signo de 16 bits. Sin embargo, en sistemas de mayores dimensiones, como los SCADA, los valores de los datos Modbus se tienen que dividir en cuatro rangos. Cada seguidor puede definirse como 65,536 elementos en cada rango. [36]

Tabla 1 Bloques de modelo de datos de Modbus [36]

Bloque de memoria	*Tipo de datos*	*Acceso de Maestro*	*Acceso de Seguidor*
Bobinas	Booleano	Lectura/Escritura	Lectura/Escritura
Entradas discretas	Booleano	Lectura	Lectura/Escritura
Registros de retención	Palabra sin signo	Lectura/Escritura	Lectura/Escritura
Registros de entrada	Palabra sin signo	Lectura	Lectura/Escritura

2.9 UNITY:

Unity es un software que permite el desarrollo de entornos virtuales, ya sea en formato 2D como 3D. Contiene una extensa documentación que permite la integración de "bibliotecas"; gracias a ello se puede diseñar, crear y hacer funcionar un entorno interactivo digital, que generalmente, es un videojuego.

Unity tiene un motor gráfico para renderizar los gráficos, un motor físico que simula las leyes de la física, permite la programación del usuario, integra sonidos y animaciones, convirtiéndolo en una herramienta versátil y efectivo para diversos propósitos, en especial el académico. Unity opera bajo dos versiones, una gratuita y otra de paga; la versión gratuita es bastante extensa y documentada, por lo que para fines académicos es perfecta. [37]

Como se mencionó anteriormente, Unity permite el desarrollo de videojuegos no solo para PC, sino también para dispositivos móviles y consolas. Esto significa que el alcance de los proyectos no se ve limitado a una sola plataforma, lo que contribuye a una mayor diversificación de los resultados obtenidos. Algunos de estos desarrollos son "Monument Valley", "Gris" o "Cuphead". [37,38]

Imagen 21 Cuphead, elaborado con el motor gráfico Unity.

- ***Scripts en Unity***

Unity solo es un entorno gráfico por lo que la interacción y desarrollo de eventos no se realiza dentro de la misma plataforma. El procesamiento y ejecución de eventos o acciones se logra gracias a la intervención de Scripts; una vez desarrollados, se añaden como una propiedad del objeto que se haya creado dentro del entorno gráfico. Los Scripts son bloques o fragmentos de código que permiten activar secuencias, animaciones o efectos que representan la cinemática de los juegos. Los Scripts pueden ejecutarse a partir de la interacción entre objetos o en respuesta a eventos específicos. El desarrollo de estos bloques de programación es posible mediante el uso de Visual Studio, quien se encarga del procesamiento de código. [39,40]

3. Trabajo desarrollado

El desarrollo de este proyecto se dividió en tres etapas secuenciales que permitieron lograr los objetivos planteados para este informe. El siguiente diagrama de flujo condensa parte de la información:

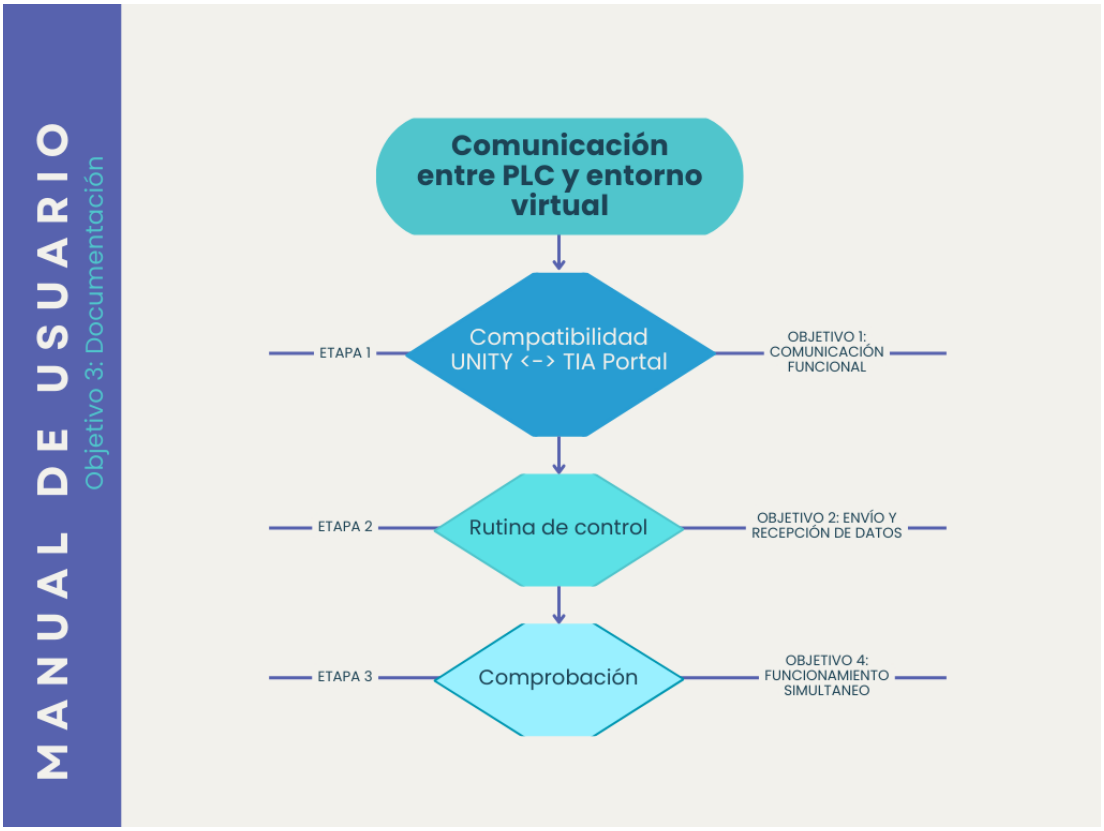


Imagen 22 Diagrama de flujo de las etapas de trabajo en el proyecto

La primera etapa consistió en generar una compatibilidad con el software de animación Unity y el entorno de programación y control del PLC, TIA Portal. Como se mencionó en el apartado *antecedentes*, se cuenta con el entorno virtual NERV, desarrollado mediante Unity. Para complementar el entorno, se planteó añadir la capacidad de comunicación para contar con un sistema electroneumático. Resuelta la comunicación, se mejoraría el trabajo previo y se cumpliría el primer objetivo del proyecto. No obstante, la mejora del entorno NERV se llevará a cabo en una etapa futura por parte de los colaboradores del Laboratorio de Automatización Industrial, ya que, para este informe, la solución de comunicación se implementó en un entorno independiente con el propósito de investigar y experimentar las capacidades del sistema, como la calidad gráfica y procesamiento de datos, sin las limitaciones de flexibilidad propias de un sistema previamente establecido (lo cual se detalla en capítulos posteriores).

La segunda etapa se centró en el desarrollo y la puesta en marcha de una rutina de control para el PLC S7-1200, que respondiera no solo de forma física, como es habitual, sino también a través de eventos virtuales. La solución implementada en la etapa uno, junto con la infraestructura del Laboratorio de Automatización Industrial, dio como resultado una respuesta de software y hardware. El software TIA Portal, que de forma nativa emplea el protocolo Profinet en sus controladores, posee la capacidad de enviar y recibir señales que pueden utilizarse para ejecutar la rutina de control en un PLC. En el laboratorio se cuenta con gabinetes eléctricos equipados con PLC S7-1200, los cuales permiten el control de actuadores neumáticos; gracias al diseño de los gabinetes también es posible conectar elementos externos, como es el caso de una botonera iluminada que funciona como torreta industrial de tres colores. Con todo lo anterior, se logró cumplir el segundo objetivo del proyecto.

Finalmente, la última etapa fue establecer una comunicación efectiva y bidireccional entre los entornos ya mencionados, realizando pruebas que evalúen el procedimiento. Para ello, se utiliza la biblioteca “S7.Net” de GitHub, la cual proporciona herramientas de sintaxis en los scripts para intercambiar datos (envío y recepción) mediante una red similar a Profinet, en donde los dispositivos Profinet pueden interpretar estas señales (más adelante se detallará cómo funciona). Las pruebas se dividieron en dos: la primera, encender y apagar un indicador; la segunda, ejecutar una rutina de control que consistió en un ciclo de accionamiento de pistones de doble efecto y la lectura del estado del vástago (reposo o extensión) mediante sensores de tipo Reed. Con las pruebas mencionadas, se evaluó la claridad y velocidad de la comunicación, cumpliendo así el cuarto objetivo. De forma paralela, se elaboró la documentación tipo manual, cumpliendo así el tercer objetivo del proyecto y facilitando la operación para los usuarios.

3.1 Entorno UNITY

Como se mencionó anteriormente, el motor gráfico empleado fue Unity. Este entorno cuenta con una extensa comunidad que ha documentado y generado bibliotecas, que permiten aprender e implementar códigos para el desarrollo de entornos más completos, eliminando posibles fallas, ya que, la misma comunidad refleja los errores que ha tenido, así como el proceso para su resolución. La relevancia de Unity es fundamental porque permite trabajar la parte visual de forma asombrosa, personalizable y, sobre todo, gratuita.

En el apartado 2.9 se mencionan las cualidades de Unity, entre ellas el uso de “Scripts”, los cuales incorporan programación que permite la interacción entre sus elementos a través de eventos que reflejan acciones o animaciones. En este proyecto, además de los scripts que generan animaciones de movimiento y efectos visuales, se emplea un script que apertura la comunicación con el PLC mediante el protocolo Ethernet - Profinet. Al ejecutar el script de comunicación, los scripts también responden ante eventos físicos. Un ejemplo de ello es el indicador visual verde de una torreta industrial, cuyo funcionamiento en la escena de Unity consiste en encenderse o apagarse a través de dos botones virtuales: un pulsador y un enclavado, respectivamente (Imagen 23, 24). De esta forma, al ejecutar el script de comunicación, el indicador visual responde tanto a las señales generadas en el entorno virtual de Unity como a las provenientes del entorno físico controlado desde TIA Portal.

En TIA Portal, la rutina de control para encender o apagar una bobina se programa de forma que pueda activarse mediante una señal física (botón físico) o una señal de memoria (señal virtual). La activación de la bobina enciende físicamente un indicador luminoso, de modo que, con ambos sistemas en ejecución, tanto un pulsador físico como uno virtual pueden generar la misma respuesta sin depender de un único tipo de señal.

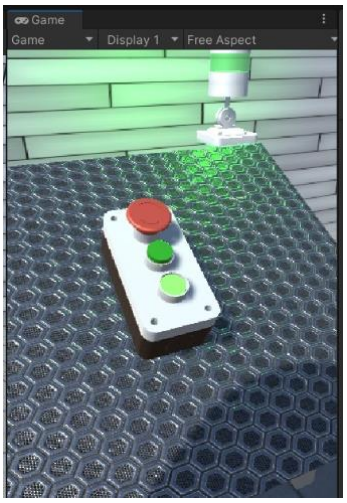


Imagen 23 Encendido de indicador verde en Unity

Imagen 24 Apagado del indicador visual en Unity

El script de comunicación genera un puente de comunicación bidireccional en donde se transmite la información pertinente acerca del evento (accionamiento), posteriormente se recaba la información y se envía al controlador (PLC); dentro de la rutina de control del equipo, se ejecuta la respuesta que este tenga programada, y como resultado, se activa una salida que puede reflejarse como un indicador (luz) o como un elemento de movimiento (un pistón), cuya acción física, también se reflejará en el entorno virtual.

3.1.1 Modelos 3D

Para integrar un gemelo digital es necesario contar con la digitalización del equipo de trabajo. Para virtualizar los objetos de trabajo, se realizó un modelado de acuerdo a las especificaciones disponibles en los diagramas del fabricante; por lo que, mediante el software *Inventor*, se recrearon los elementos. Cabe destacar que para aquellas piezas que contienen una animación de desplazamiento, su modelo tridimensional debió ser considerado como un objeto por partes, similar a un ensamble, tal es el caso de: la botonera y el pistón (Imagen 25). Ya que, si se desarrolla todo el elemento, al añadirlo a Unity, este entenderá que es una pieza sólida y es imposible realizar algún efecto de desplazamiento.

Al virtualizar un objeto, en ocasiones resulta conveniente generar ensambles ya que, al momento de agregar la pieza a Unity, es posible asignar diferentes materiales al objeto o incluso darle apariencias transparentes para darle un efecto visual más agradable, tal es el caso de la torreta industrial (Imagen 26).

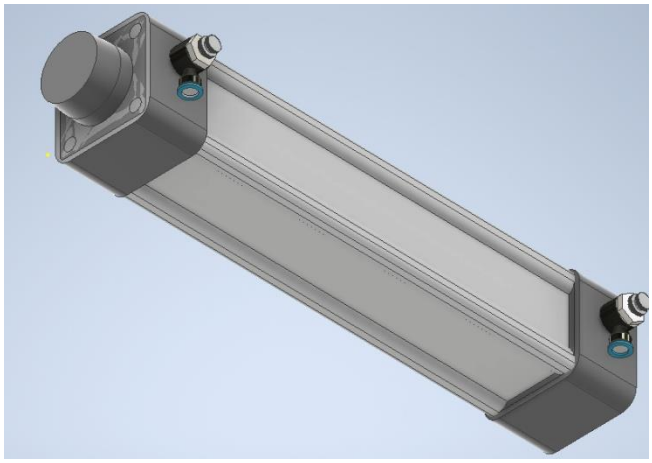


Imagen 25 Pistón virtual

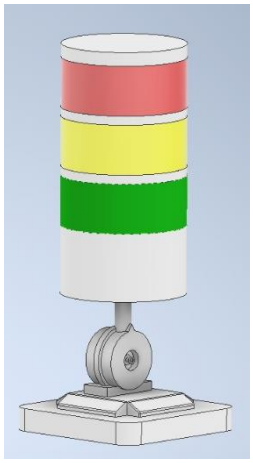


Imagen 26 Torreta virtual

Dentro de Unity, los elementos (importados en ".obj"), se encontrarán sin textura, solo como objetos tridimensionales genéricos (Imagen 27), por lo que parte del trabajo, también se refleja en el diseño y mejora de las cualidades físicas de los elementos. Se generaron diversas texturas que permitieron que la apariencia del entorno, fuera mejor y apegado a los elementos reales.

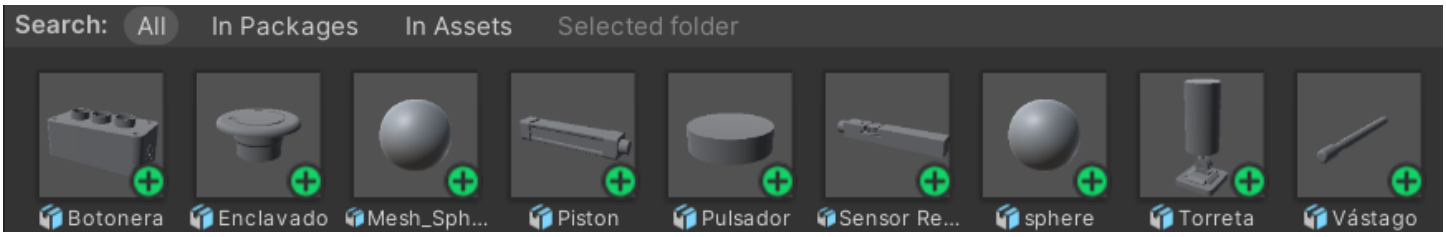


Imagen 27 Objetos tridimensionales requeridos en el proyecto

Ya integrados los objetos tridimensionales, se generaron dos mesas de trabajo virtuales, teniendo como referencia las mesas neumáticas del Laboratorio de Automatización Industrial (LAI). (Imagen 28)

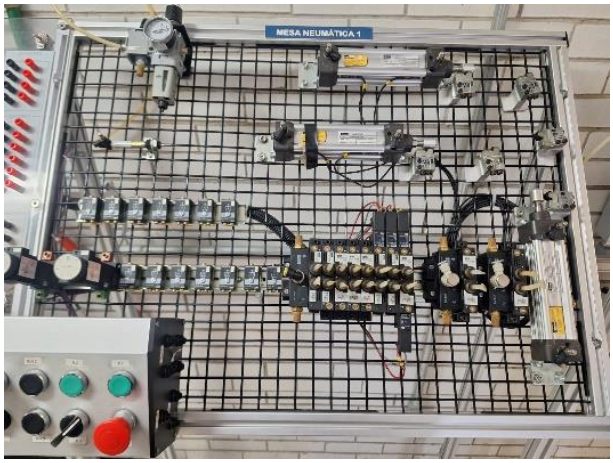


Imagen 28 Mesa neumática del LAI

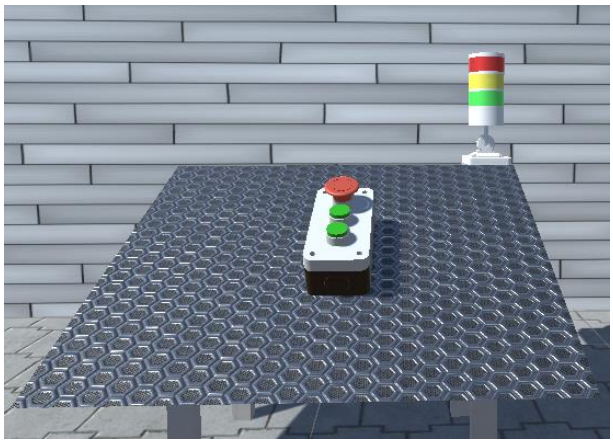


Imagen 29 Mesa 1 virtual; botonera y torreta industrial

En la mesa virtual de trabajo 1 (Imagen 29) se muestran dos elementos, una torreta industrial y una botonera. La mesa virtual cuenta con la operación de encendido y apagado de una luz. En una primera prueba, solo se activará el indicador verde. En la segunda prueba, se utilizarán los tres indicadores luminosos de la torreta (rojo, ámbar, verde) para visualizar un control de luces, similar al funcionamiento de un semáforo.

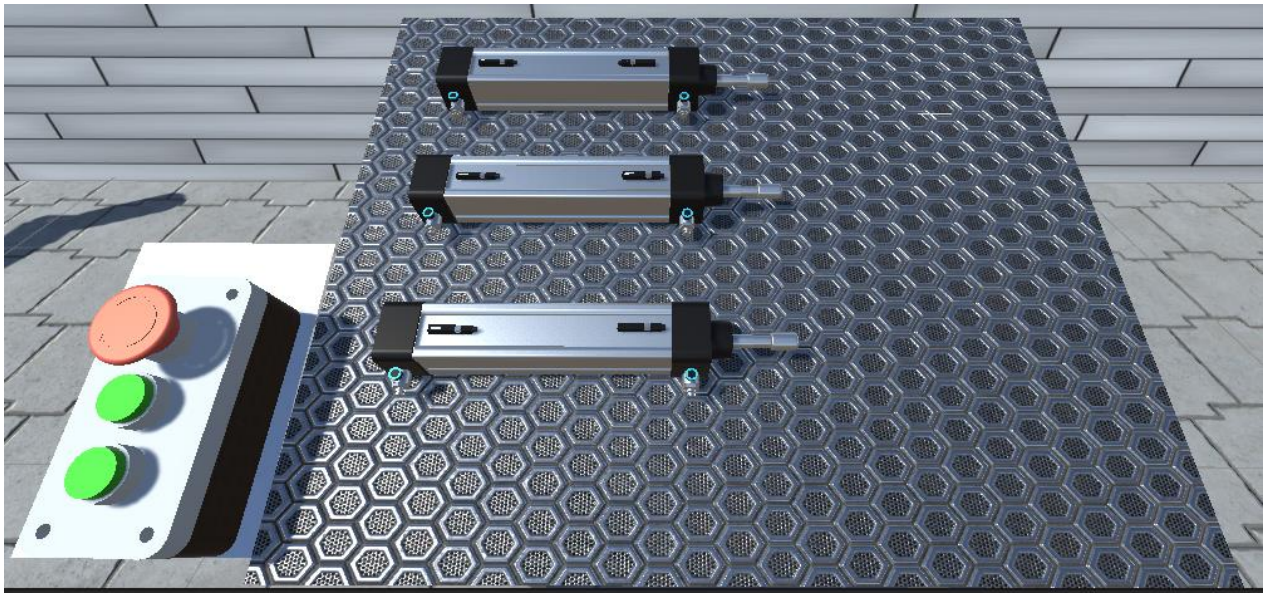


Imagen 30 Mesa 2 virtual; pistones, sensores Reed y botonera

En la mesa virtual de trabajo 2 (Imagen 30) se muestran elementos como una botonera, pistones de doble efecto y sensores de efecto Reed. La primera prueba para esta mesa consiste en la extensión y retracción del vástago de un cilindro de doble efecto. En la prueba dos, un proceso secuencial que involucre a los 3 pistones.

3.1.2 Programación en Visual Studio

Ya contando con el objeto virtual y mejorada su apariencia visual, los scripts de Unity responden a la lógica de programación de C#. No obstante, Unity no es un editor de código C#, por lo que esta función está a cargo de Visual Studio. Aquí se genera toda la parte de programación que Unity requiere para ejecutar eventos muy específicos en su entorno, en especial cuando hay interacciones físicas; por ejemplo, en el videojuego *Cuphead (Imagen 21)*, los disparos y ataques que el avatar realiza son ejecutados mediante códigos en C# que condicionan la cadencia, tamaño y daño de cada disparo que el personaje realiza.

La creación de los scripts que corresponden a las animaciones se explica con mayor detenimiento en el *Manual de conexión Unity - TIA Portal para un Gemelo Digital* (consultar el apartado *Anexo*).

Para operar el programa es necesaria la instalación de una biblioteca (S7.Net) que contiene los complementos y paquetes de datos que se ocupan para la conexión vía Profinet, en este caso, todo el procedimiento se lleva a cabo desde el entorno de UNITY, como a continuación se describe:

Una vez creado un proyecto en UNITY y haber hecho un acomodo visual adecuado, se debe programar el script correspondiente, en el cual, se inicie la comunicación con el PLC (habilitación de puertos de comunicación) y se preparen ambos sistemas para el envío y recepción de datos. Esto se logra gracias a la biblioteca S7.Net, proceso que a continuación se describe con mayor precisión.

Es importante aclarar que la biblioteca S7.Net contiene una sintaxis precargada que permite la pasarela de datos (biblioteca ".dll"), en este caso, como UNITY abre archivos específicos para su ejecución (archivos ".cs"), no se pueden indexar o agregar complementos, por lo que se requiere añadir un archivo ".dll" o Biblioteca de Vínculos Dinámicos, la cual contiene toda la sintaxis que se requiere para las conexiones. De forma general, una DLL contiene datos y códigos para que se puedan usar en más de un programa al mismo tiempo [41].

Cabe aclarar que, para hacer uso pleno de la biblioteca, se debe cargar un archivo con extensión ".dll" en UNITY, permitiendo activar o desactivar variables en todos los elementos que tengan interacción o manden señales al controlador. La añadidura de este paquete de datos se puede hacer desde Visual, creando un archivo de tipo ".dll" y pegando el código pertinente de la página de "GitHub" buscando el apartado "S7Net". Para este proyecto, se descargó el archivo directamente de la web y se colocó directamente en la carpeta destinada para los archivos de esta índole, a continuación, se describe este proceso. [42,43]

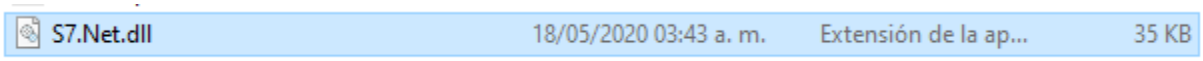


Imagen 31 Archivo ".dll" obtenido de la página web "PLCFOCUS" [28]

- ***Integración de una biblioteca de vínculos dinámicos:***

Para integrar una biblioteca de vínculos dinámicos es vital conocer la versión en la cual se está desarrollando el proyecto. Unity recibe anualmente actualizaciones y mejoras en su software, por lo que la integración de nuevas carpetas y archivos llega a variar dependiendo de la versión, en este caso, la integración de la biblioteca necesaria para la versión de Unity 2022.3.10f1; requiere ser colocada en la ruta de ejecución de las bibliotecas nativas del software donde se encuentran almacenados los datos del proyecto, ya que, al momento de ejecutar el programa, Unity la identificará y cargará en automático.[44]

La ruta de acceso en donde se ejecuta el programa (dependerá completamente de la ubicación designada para el almacenamiento de los archivos del proyecto) es, por ejemplo:

C:\Users\Alien1VR\Proyecto_Comunicacion

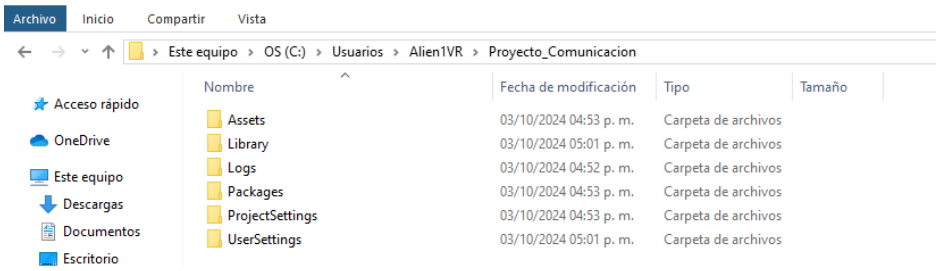


Imagen 32 Ruta a la ubicación del proyecto de UNITY

Para agregar una biblioteca, se debe colocar el archivo sobre la carpeta “Assets” y dentro de esta, se debe seleccionar la subcarpeta Plugins o Editor. La selección de carpeta dependerá del propósito de uso.

- Plugins contiene bibliotecas nativas o DLLs que se desean utilizar en el proyecto, pueden ser bibliotecas de C#, C++ o cualquier otro código nativo. Unity automáticamente cargará las DLLs que se encuentran en la carpeta al iniciar el proyecto. [44, 45]
- Editor es para el uso de herramientas o extensiones del editor, por lo que las clases dentro de esta carpeta se ejecutarán solo mientras se esté utilizando el editor de Unity y no estarán incluidas en la compilación final del juego. [44, 45]

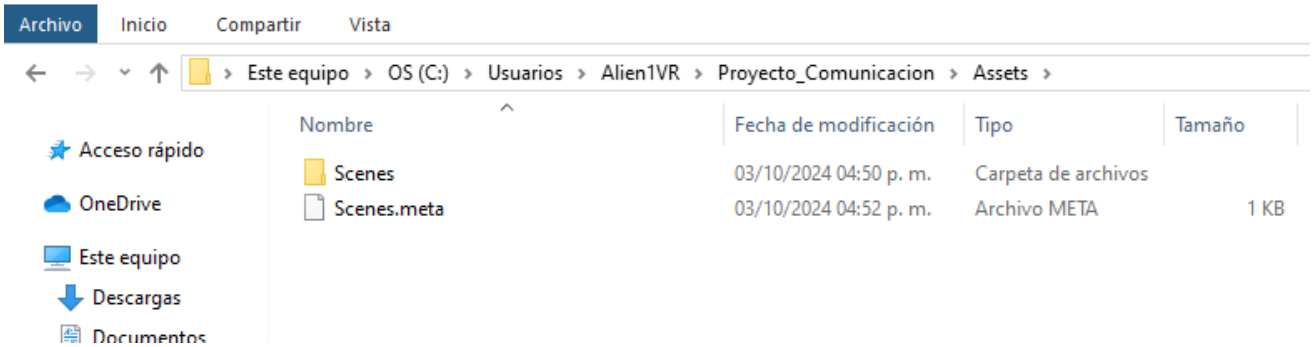


Imagen 33 Carpeta Assets

Una vez accedido a la carpeta “Assets” (Imagen 33), se observa que no existe ninguna de las dos subcarpetas que se describieron anteriormente, por lo que se deben crear manualmente para que Unity las reconozca. En este caso, se generará la subcarpeta *Plugins*, ya que, Unity entenderá que se trata de una carpeta con referencias de bibliotecas externas y debe ser utilizada en el proyecto, añadiendo que la carpeta *Plugins* contiene los archivos DLL de un proyecto y que estos archivos sí seguirán presentes en la ejecución final del proyecto con diferencia de los archivos que se colocan en la carpeta *Editor*. Esta acción se observa en la Imagen 34.

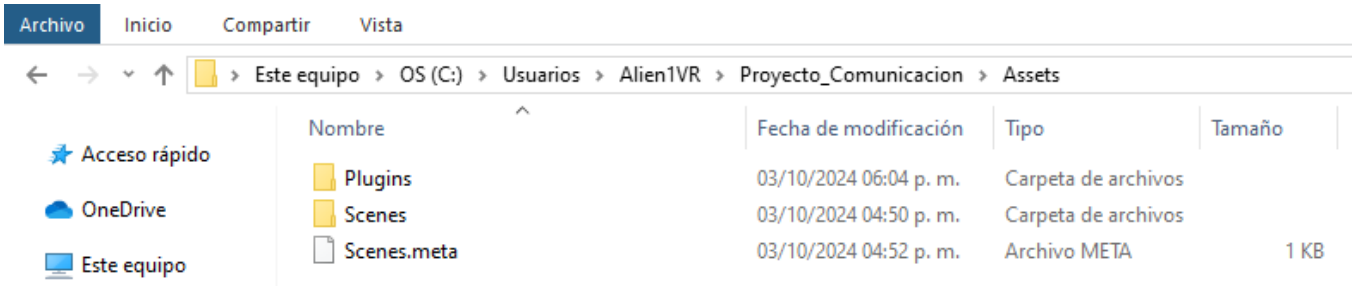


Imagen 34 Creación de carpeta "Plugins"

A continuación, se debe colocar el archivo DLL que contiene las referencias necesarias para generar la comunicación Unity – TIA Portal – PLC (Imagen 35). Con ello, solo es cuestión de escribir la sintaxis correspondiente dentro del script principal.

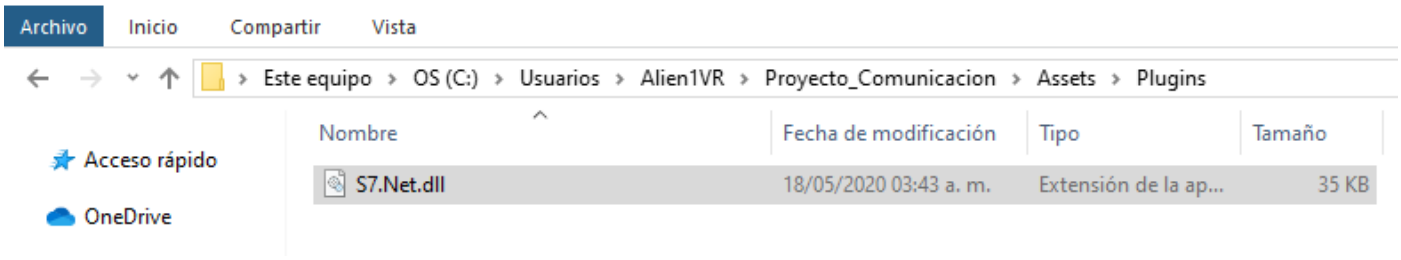


Imagen 35 Archivo .dll

- *Eliminación de error: "Unable to open Assets/Plugins/S7.Net.dll: Check external application preferences"*

Al cargar la biblioteca y compilar el programa, se genera un error referente a las referencias externas, el cual es una advertencia de Unity, ya que, el sistema no reconoce el origen de este archivo como propio.

Ir la cinta de opciones de Unity, en la pestaña de "*Edit"* se selecciona la opción de "*Project Settings"* y la opción de "*Player"*.

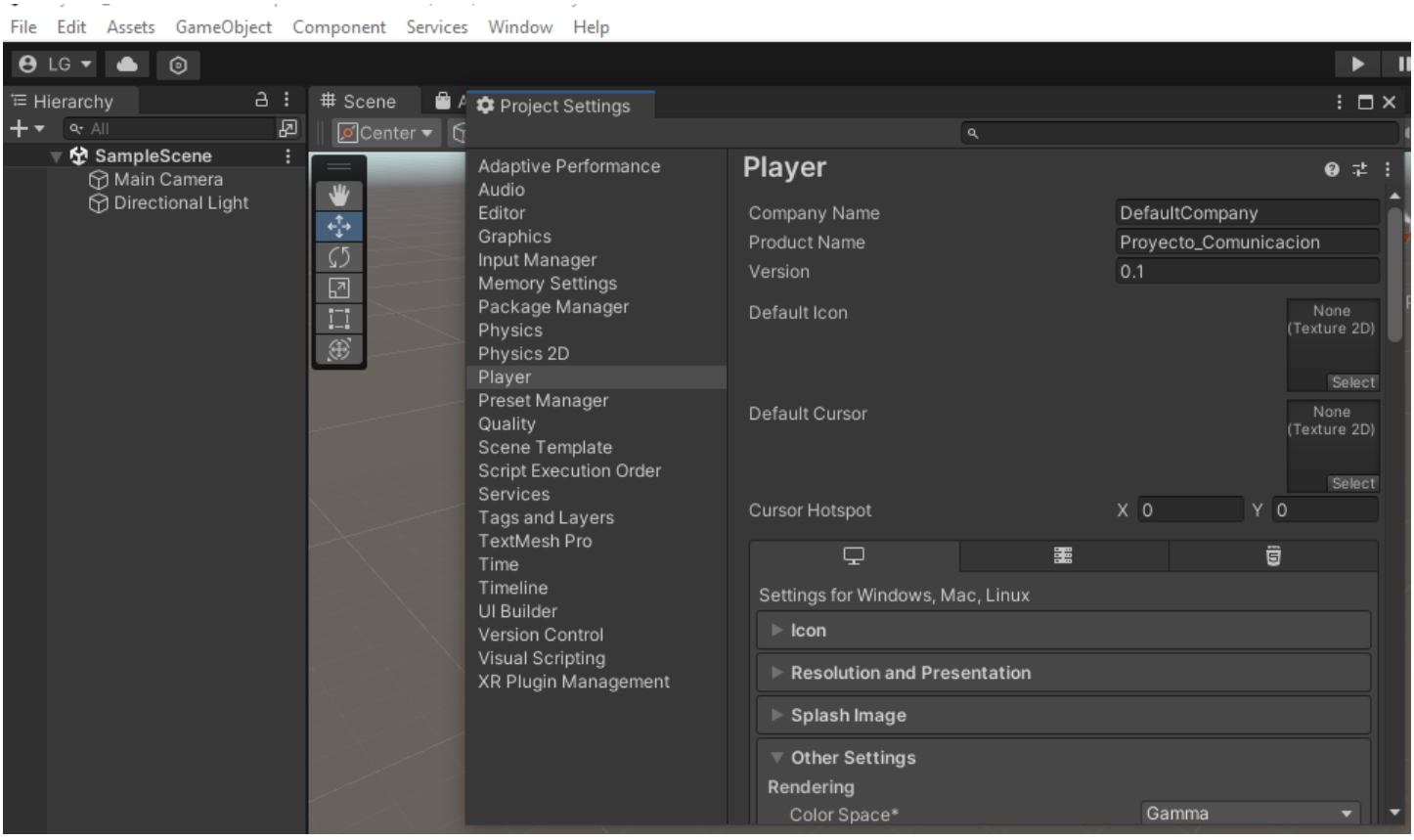


Imagen 36 Project Settings

Una vez accedido a este menú, se debe desplazar hasta el apartado "*Other Settings*", al desplegar la pestaña, se debe dirigir a la opción de "*configuration*" y posteriormente, la opción que dice "*Api Compatibility Level*" se debe cambiar a la opción "*.NET Framework*", una vez seleccionada esta opción Unity de inmediato reconocerá el cambio y se compilará, eliminando el error.

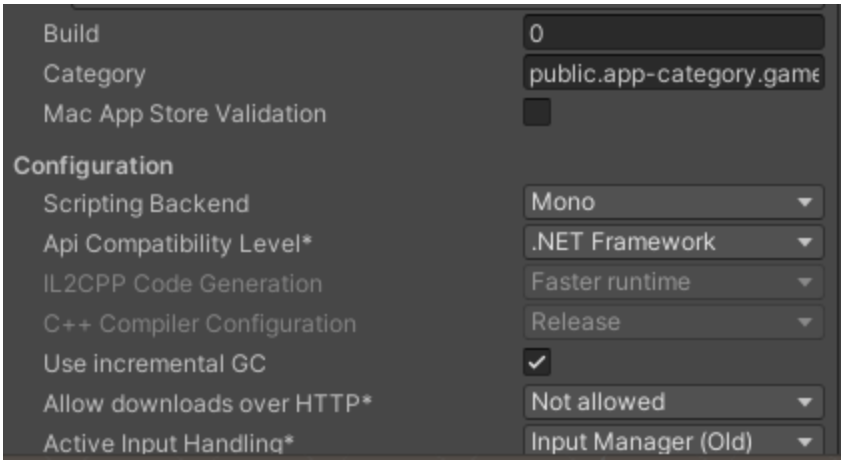


Imagen 37 Configuration -> Api Compatibility Level -> .NET Framework

Finalmente, lo que corresponde es añadir a los scripts las variables que se obtendrán desde TIA Portal, haciendo referencia a la función principal del script de comunicación.

3.2 Rutina de control en PLC

Teniendo en mente las circunstancias por las cuales se eligió el protocolo PROFINET y por ende el entorno TIA Portal, no está de más mencionar que el protocolo Modbus es más sencillo de implementar en una aplicación de esta índole (comunicación por red) sin embargo, en la búsqueda de documentación y referencias para poder aplicarlo a este proyecto, no se encontró una gran cantidad de datos y los procedimientos de programación se volvían más complejos. Tanto para Modbus como para Profinet, se debe ser cuidadoso en la ubicación de los registros en donde se colocarán los datos, en este caso, delimitar los sectores de memoria del controlador para que se pueda almacenar los datos de interés, por lo que se habla de un orden estricto al momento de programar y elegir los registros, sin embargo, los controladores en ocasiones no cuentan con el espacio o memoria suficiente y la administración de sus registros, no siempre es accesible. De manera general, el error común es una sobre escritura de registros, en donde totalmente se corrompen los datos de interés e incluso se pueden afectar los datos de la rutina de control, por tanto, en controladores con memoria restringida o no administrable, se podría llegar a tener fallas en la generación de un gemelo digital.

El protocolo PROFINET también brinda un reto en la programación del gemelo digital. A pesar de ello, TIA Portal resuelve el dilema acerca del uso de la memoria en los registros, a través del uso de registros “DB (Data Blocks)”, donde el usuario, propiamente administra y limita el acceso a ellos. Sin la sintaxis adecuada, no se accede a estos datos, dando una excelente ventaja para el uso de memoria del controlador. Limitar la memoria y proceder mediante el nombre del dispositivo (nombre PROFINET) y su I.P. brinda una mayor facilidad para la pasarela de datos, además de que resuelve la situación del orden en el uso de registros. La biblioteca que se utiliza (S7.NET), bastante documentada, indica que al hacer algunas modificaciones menores (se modifica la longitud de datos a leer y escribir) permite delimitar el uso de los bloques de memoria, generando bloques de acuerdo a la cantidad y tipo de variables a usar.

La rutina de control para la primera prueba se desarrolla como un simple enclavamiento con dos posibles activaciones, una de ellas corresponde a una señal física, la cual es un botón pulsador. La otra señal corresponde a una memoria en donde se registrará el cambio virtual. Es importante destacar y hacer entender a los usuarios que, aunque las señales virtuales son un símil a un elemento físico, estas jamás podrán tener una interpretación equivalente a una señal física en el PLC, por tanto, siempre que se tenga una señal que no sea física, debe poseer

el identificador de memoria (%M) y a partir de la lógica desarrollada, puede operar a lo largo de una rutina de control.

El método de declaración en el script correspondiente requiere identificar el tipo de dato, por tanto, en la taba de tags de cada DB, cuidar declarar los tipos de datos de trabajo, ya que, de no existir coincidencias, el programa falla.

3.2.1 Control de encendido y apagado de un indicador luminoso

Dentro de TIA Portal, en el *Main*, se genera una lógica muy sencilla para encender una bobina, con un enclavamiento con prioridad a la desconexión.

Para que el sistema opere sin inconvenientes:

- Los DB deben encontrarse sin optimizar
- El PLC debe tener activado el acceso completo (sin restricciones de usuario)
- El PLC debe tener activa la casilla de PUT/GET communication

En la rutina de la Imagen 38 se observa que al momento de recibir un estado verdadero de una variable de entrada física (*BSecundario2*) o de la variable virtual (*DB_Recibir_Datos.Start_Virtual_M1*), se activan dos salidas, una física llamada *Luz_Verde*, y la otra, *DB.Enviar_Datos.Edo_Luz_Verde*.

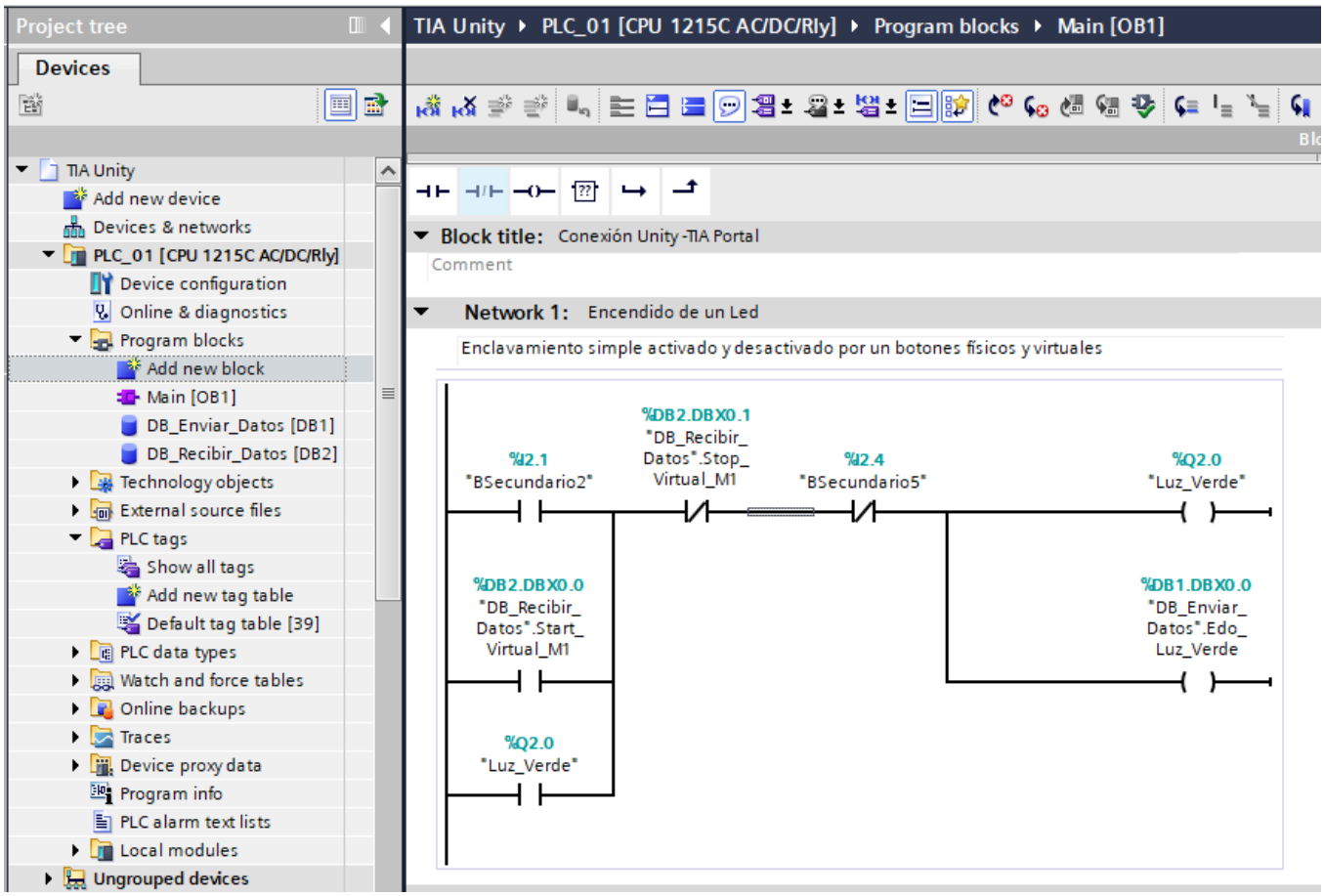


Imagen 38 Network 1: Encendido de un indicador luminoso en lenguaje escalera

3.2.2 Control de un actuador neumático, pistón de doble efecto

Reutilizando el programa de la Imagen 38 y con los elementos con los que cuentan las mesas neumáticas en el LAI [Imagen 28]; en un nuevo peldaño se coloca la configuración correspondiente para activar y desactivar los pilotajes 14 y 12 de la electroválvula biestable que controla al pistón A de forma física, como se muestra en la Imagen 39.

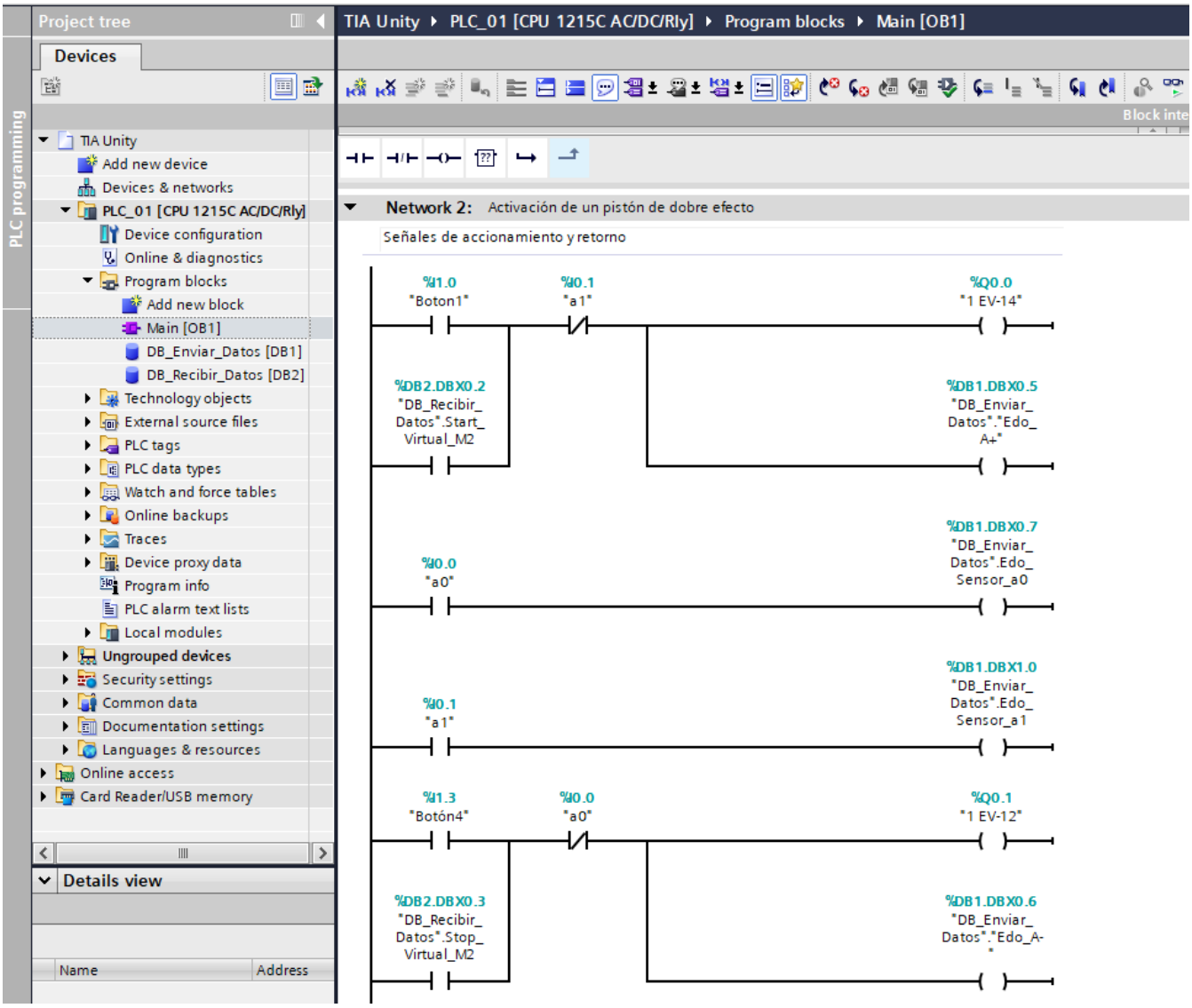


Imagen 39 Network 2: Expulsión y retracción del vástago de un cilindro de doble efecto

En la Imagen 39 se observa que las señales virtuales provienen del *DB* "*Recibir_datos*", operando de forma paralela a las señales físicas. Para tener una retroalimentación del sistema, los sensores de tipo Reed, envían directamente su respectiva señal al entorno virtual; en la escena de Unity se recibe la señal y permite leer el estado de los indicadores para que visualmente estos reflejen el dato (encendido o apagado).

3.3 Testing de comunicación

Para corroborar que el sistema está funcionando, se generó un script de prueba en donde no solo visualmente se aprecie la pasarela de datos, sino que, mediante el uso de la consola de Unity, se lea la información obtenida en tiempo real. Este script solo se utilizó en una ocasión ya que, para versión final, los datos y operaciones se encuentran en el script "PLC_Comunicación" dentro de la carpeta *Conexión*. Cabe mencionar que el sistema puede interactuar de dos maneras a través de TIA Portal; la primera desde un entorno totalmente simulado (haciendo uso del PLCSIM) o de forma física, directamente operando el controlador físico.

La prueba de comunicación se realizó con el entorno simulado (PLCSIM).

El primer paso es arrancar con la simulación en TIA Portal con ayuda de PLCSIM (Imagen 40); es más sencillo si se usa una "Tabla SIM" para este proceso (Imagen 41).

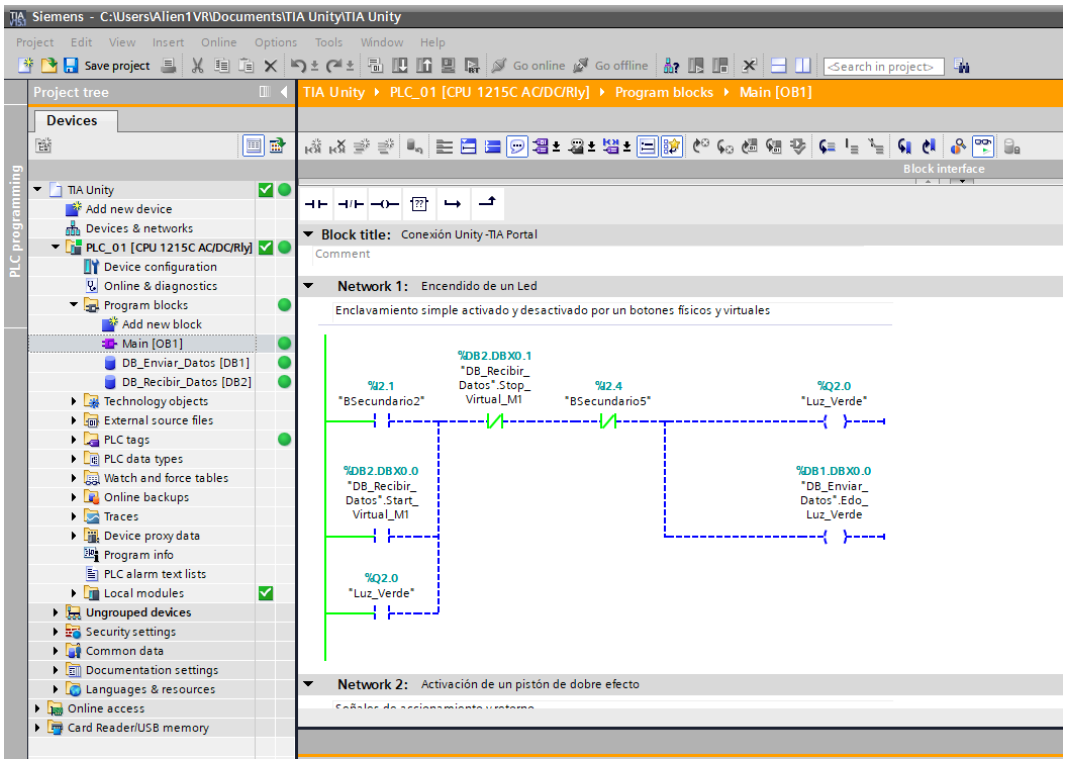


Imagen 40 Simulación de Network 1 (encendido de un led)

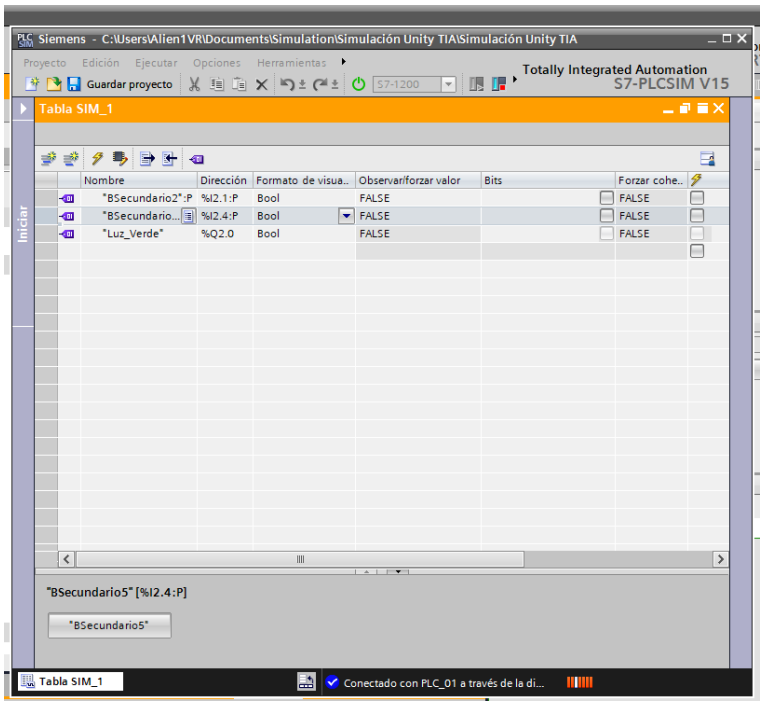


Imagen 41 Tabla SIM; Simulación encendido de un led

Ahora, en el proyecto de Unity, asociar un script (*Test*) a un objeto vació (*Empy Object*) que se encuentre en la escena, no interesa la ubicación del objeto, solo que esté dentro de la escena.

El script *Test* contiene la referencia a la DLL *S7.Net* como biblioteca de uso, previamente cargada en la carpeta *Plugins*; un objeto público del tipo GameObject (referencia al indicador verde), y una variable privada del tipo PLC. La primera función en el código apertura la comunicación de Unity con TIA Portal, tomando como referencia que es un PLC S7-1200 que se encuentra en la red 127.0.0.1; esta configuración es tal y como se indica en el manual de uso de *S7.Net* (Imagen 42). Dentro del código, se hace un pedimento de lectura de la dirección "*DB1.DBX0.0*" y se almacena en la variable *señalLuz*, quien se convierte en verdadera o falsa de acuerdo a lo que detecte. A través de esta lógica, se puede activar o desactivar el *GameObject.* También, se manda a imprimir el valor en la consola (Imagen 43). [46]

```
Test.cs*
Archivos varios
LuzDesdePLC
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using S7.Net;
5
6  public class LuzDesdePLC : MonoBehaviour
7  {
8      public GameObject lVer;            // Arrastra aquí el Animator de tu luz
9      public bool luzV;                  // Referencia al objeto
10
11     private Plc plc;
12
13     void Start()
14     {
15         plc = new Plc(CpuType.S71200, "127.0.0.1", 0, 1);
16
17         try
18         {
19             plc.Open();
20             Debug.Log(" Conectado al PLC.");
21         }
22         catch
23         {
24             Debug.LogError(" No se pudo conectar al PLC.");
25         }
26     }
```

Imagen 42 Inicio de comunicación con TIA Portal con S7.Net (script)

```
try
{
    // Leer señal desde el PLC (ej: DB1.DBX0.0)
    bool señalLuz = (bool)plc.Read("DB1.DBX0.0");

    if (señalLuz)
    {
        lVer.SetActive(true);
    }
    else
    {
        lVer.SetActive(false);
    }

    Debug.Log("Señal desde PLC: " + señalLuz);
}
catch
{
    Debug.LogWarning(" Fallo al leer del PLC.");
}
```

Imagen 43 Activación y desactivación de GameObject (script)

Compilado y cargado el programa, ejecutar la aplicación “NetToPLCsim” como administrador, debido a que esta aplicación al estarse ejecutando, debe mantener un domino sobre el puerto 102 de comunicación para generar una conexión TCP/ IP. De forma somera, esta herramienta permite acceder a la aplicación PLCSim, haciendo uso de una comunicación TCP/IP desde una red local generada directamente en la computadora en la que se ejecuta la simulación utilizando la interfaz de red. [46]

La configuración de la aplicación es sencilla, se requiere de un nombre de red, una IP de trabajo (127.0.0.1) y la IP de operación del PLCSIM (192.168.105.121). Finalmente, configurar el Rack y Slot de trabajo del CPU, en este caso será 0 y 1, respectivamente. Iniciar con el servidor y poner en ejecución (Imagen 44).

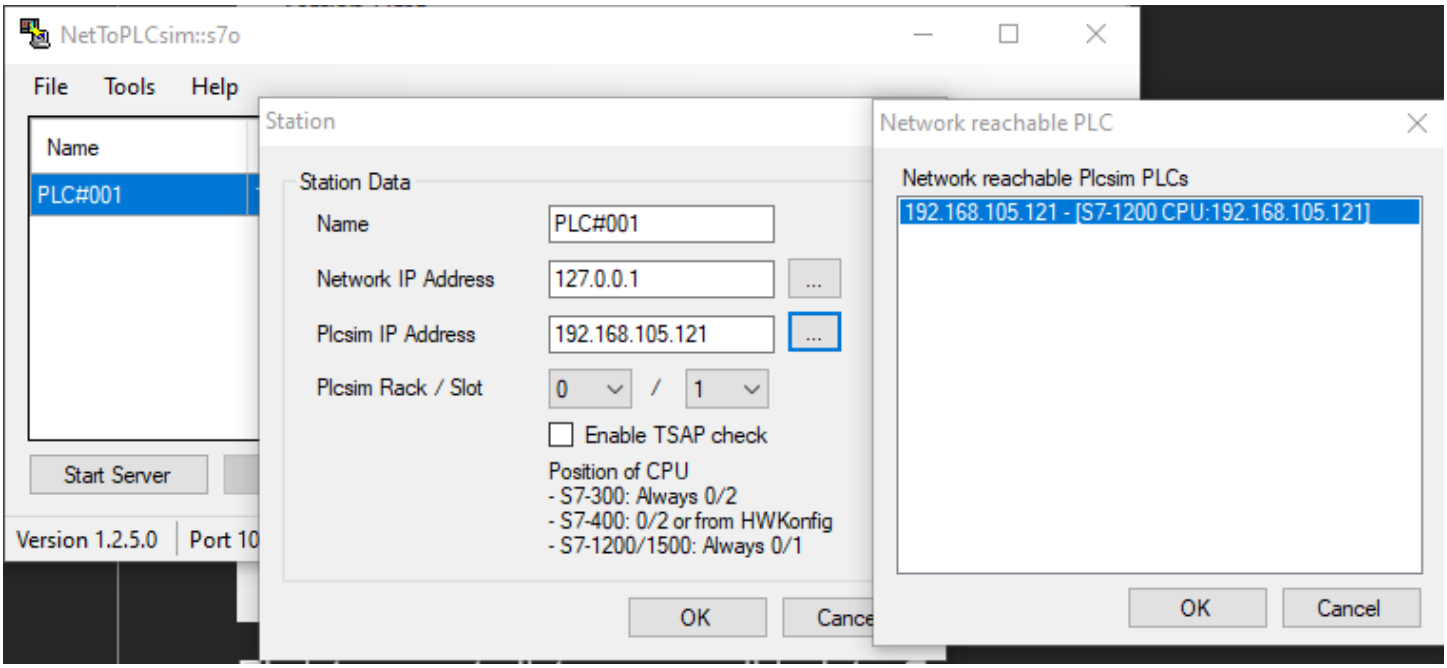


Imagen 44 Configuración de la aplicación NetToPLCsim

Con la red generada, ejecutar el proyecto en Unity. En cuanto compile los archivos, de inmediato aparecerá en la consola que el sistema se ha conectado al PLC y comenzará a mandar el estado de la señal (Imagen 39).

Para obtener un cambio, activar la señal de entrada *BSecundario2* de la Tabla Sim. La salida *Luz_Verde* se vuelve verdadera al igual que la salida *EDO_Luz_Verde* del bloque de datos de envío. En el entorno virtual de Unity, el indicador verde de la torreta se activa.

Ahora, TIA Portal y Unity están comunicándose y tienen un reflejo de sus acciones (Imagen 45). Hasta el momento, en una sola dirección por lo que, en las siguientes pruebas, se hará bidireccional. En general, no es necesario estar mandando a llamar el valor de las señales en la consola (Imagen 46) sin embargo, es recomendable que el estado del PLC sea reflejado en la consola.

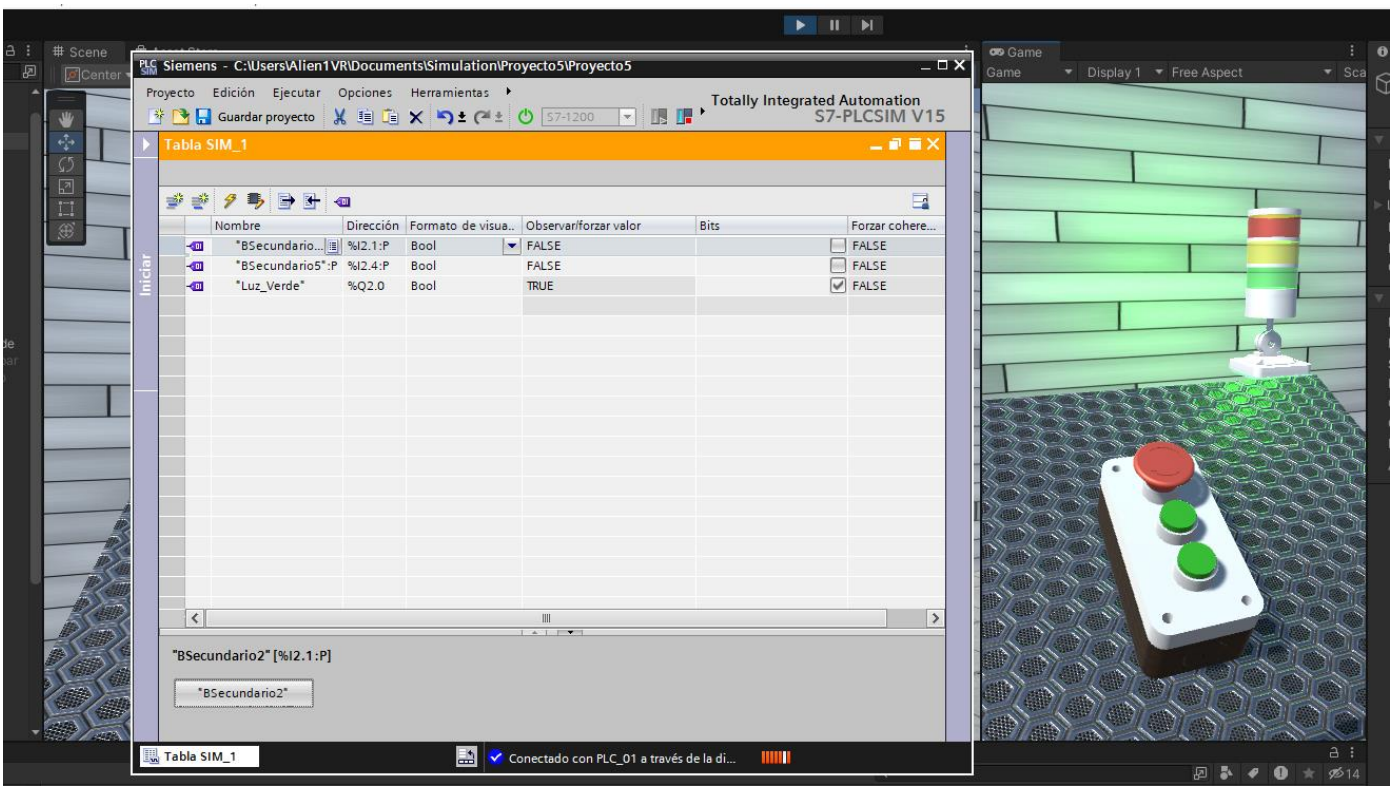


Imagen 45 Reflejo en la activación de una salida física en Unity desde TIA Portal

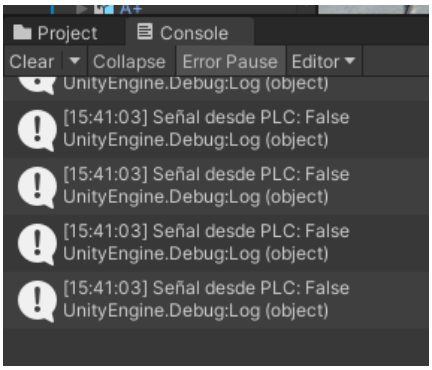


Imagen 46 Información de la consola

3.3.1 Gemelo digital mediante PLCSIM

A partir del script *Test*, se genera un nuevo script (PLC_Comunicación) que contiene la sintaxis para el pedimento de los valores de las direcciones requeridas, así como los bloques de funciones en donde se colocan las direcciones a escribir de acuerdo con los datos generados en el entorno digital. Es primordial seguir el orden de activación de cada software, ya que, de no ser así, el sistema falla y no se puede generar la bidireccionalidad de datos.

Los pasos son:

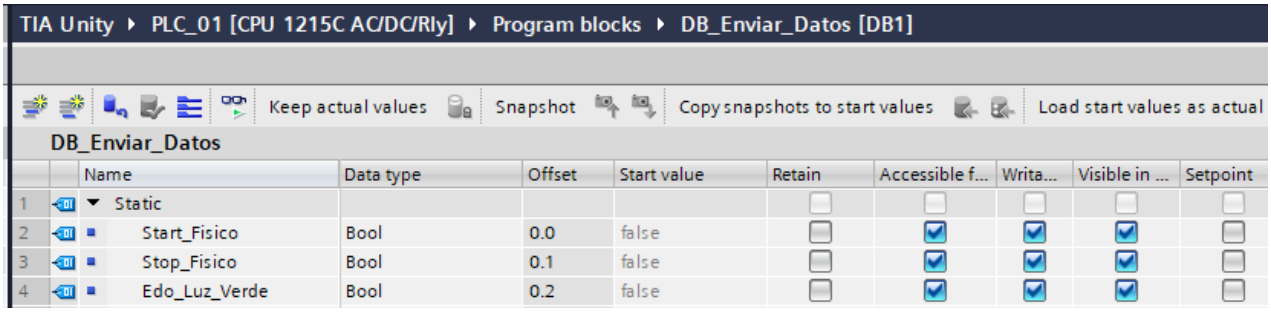
- Iniciar el PLCSIM con su respectiva Tabla Sim para agilizar la activación y desactivación de señales.
- Arrancar el software NetToPLCsim [Network extension for PLCSIM]. La dirección de trabajo es 127.0.0.1 y la IP del PLCSIM estará en 192.168.105.121 (se puede variar de acuerdo a las necesidades del usuario).
- Ejecutar la escena de Unity y observar los cambios visuales al mandar señales desde TIA Portal y activar las señales desde la escena de Unity y ver el reflejo en la Tabla Sim.

- ***Prueba de encendido de indicador luminoso.***

De acuerdo con la lógica presentada en la sección 3.2.1 (Imagen 38), se modifica el código de TIA Portal, añadiendo dos nuevos peldaños que envían directamente la señal de los botones físicos al DB de envío de datos.

Imagen 47 Peldaños para envío de datos

Las señales físicas también se envían al entorno virtual (Imagen 48), dado que ahora activan los eventos de encendido y apagado del indicador luminoso verde (Imagen 49). Para comprobar el sistema virtual, se presiona el botón de encendido. Aunque en TIA Portal se utilice una tabla SIM, los cambios se pueden observar, como se muestra en la Imagen 50.



		Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint
1		Static								
2		Start_Fisico	Bool	0.0	false					
3		Stop_Fisico	Bool	0.1	false					
4		Edo_Luz_Verde	Bool	0.2	false					

Imagen 48 Envío de señales a través de DB "Enviar_Datos"

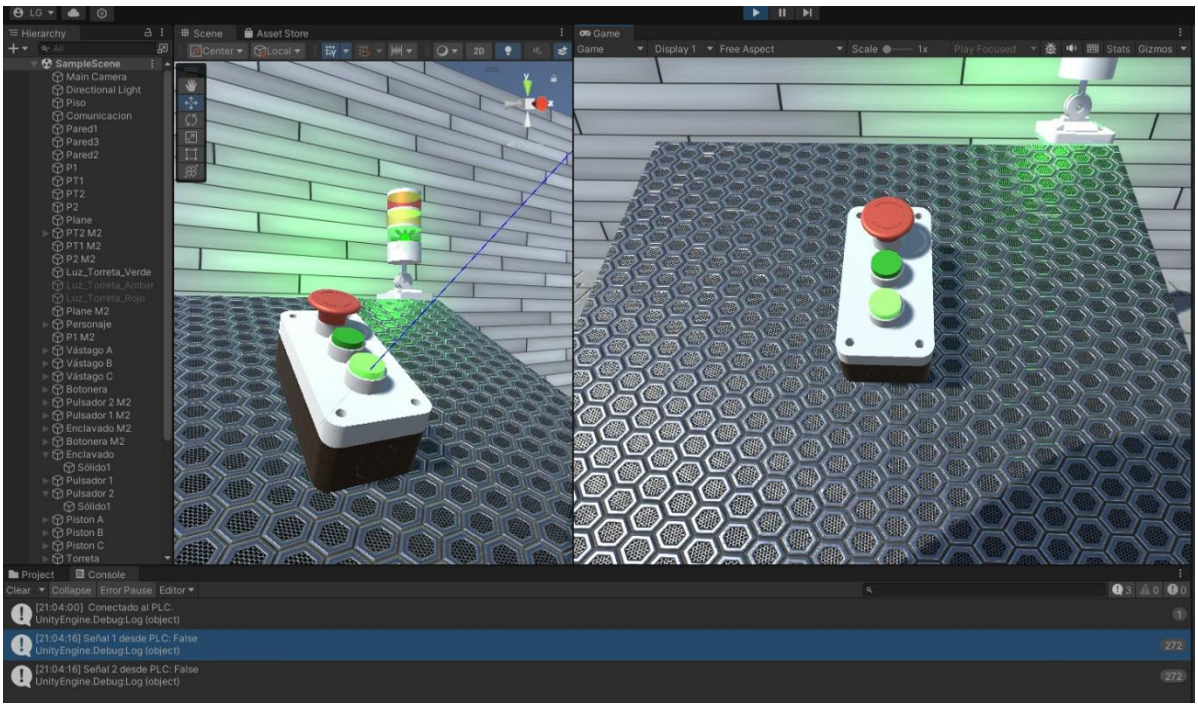


Imagen 49 Encendido de indicador luminoso verde desde Unity.

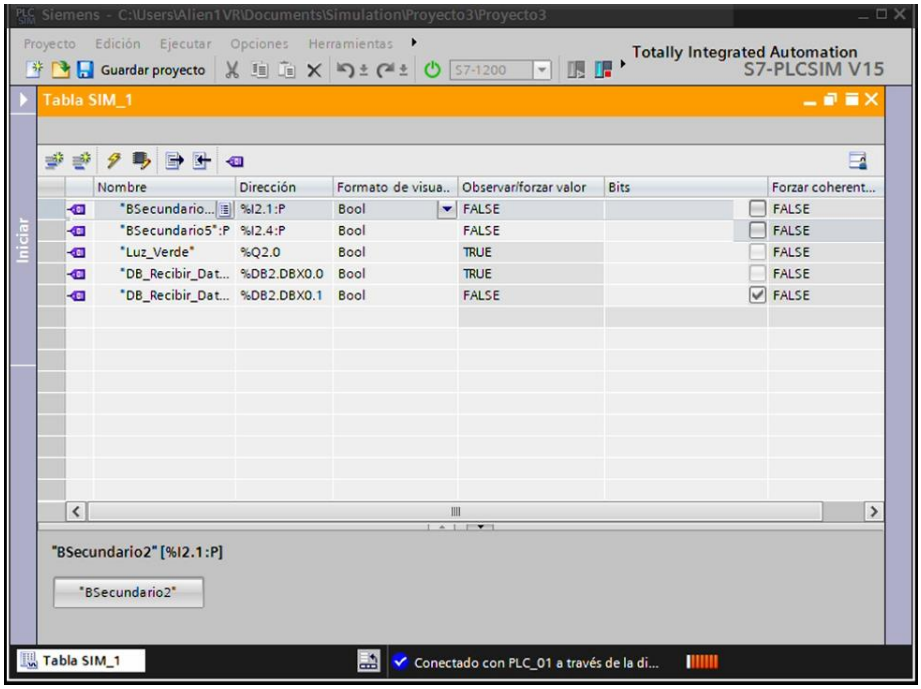


Imagen 50 Monitoreo en tabla SIM; señal recibida desde Unity

Al activar el botón enclavado para apagar el indicador luminoso verde (Imagen 51), la señal también es reflejada en TIA Portal con la ayuda de una tabla SIM (Imagen 52).

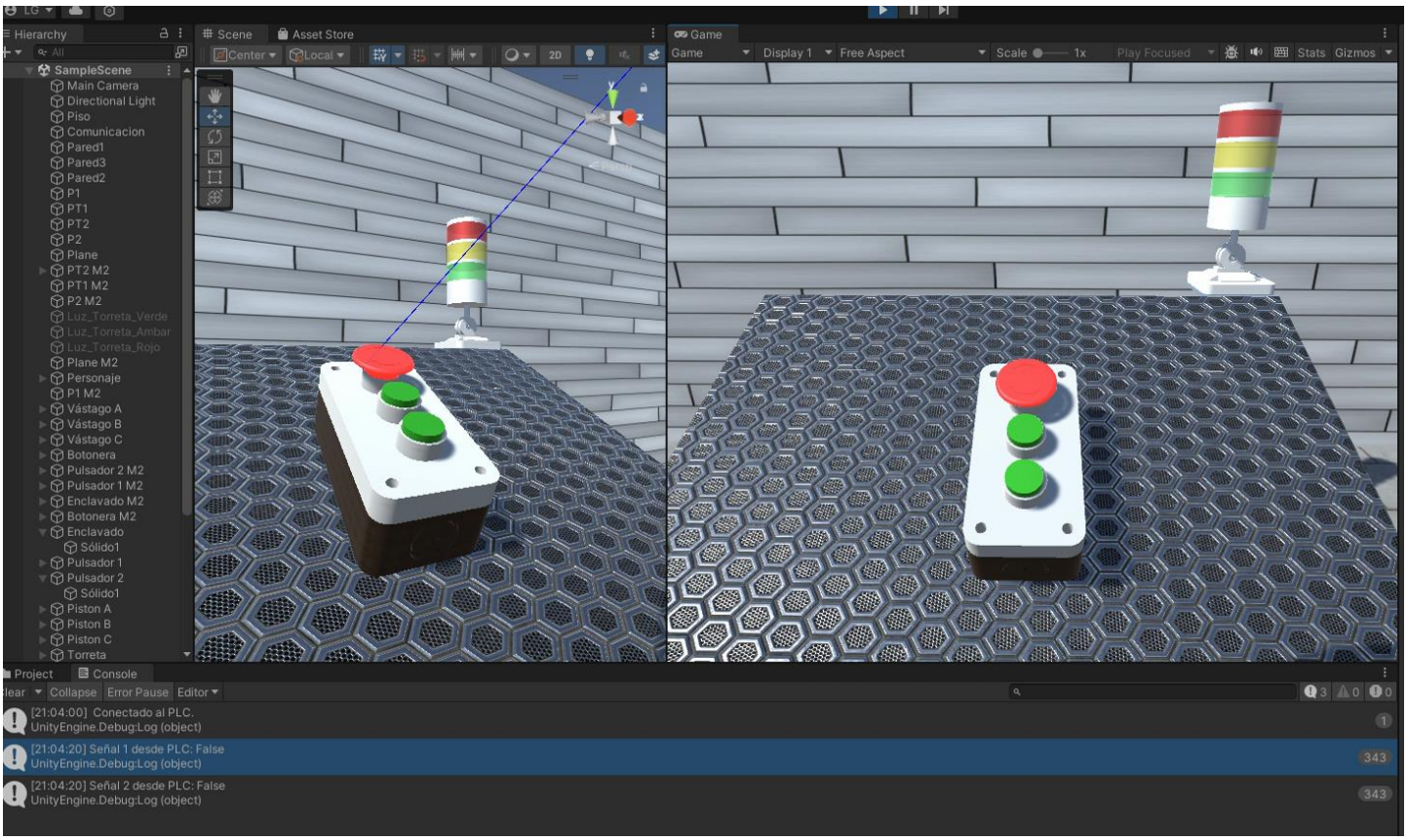


Imagen 51 Apagado del indicador luminoso verde por medio de Unity

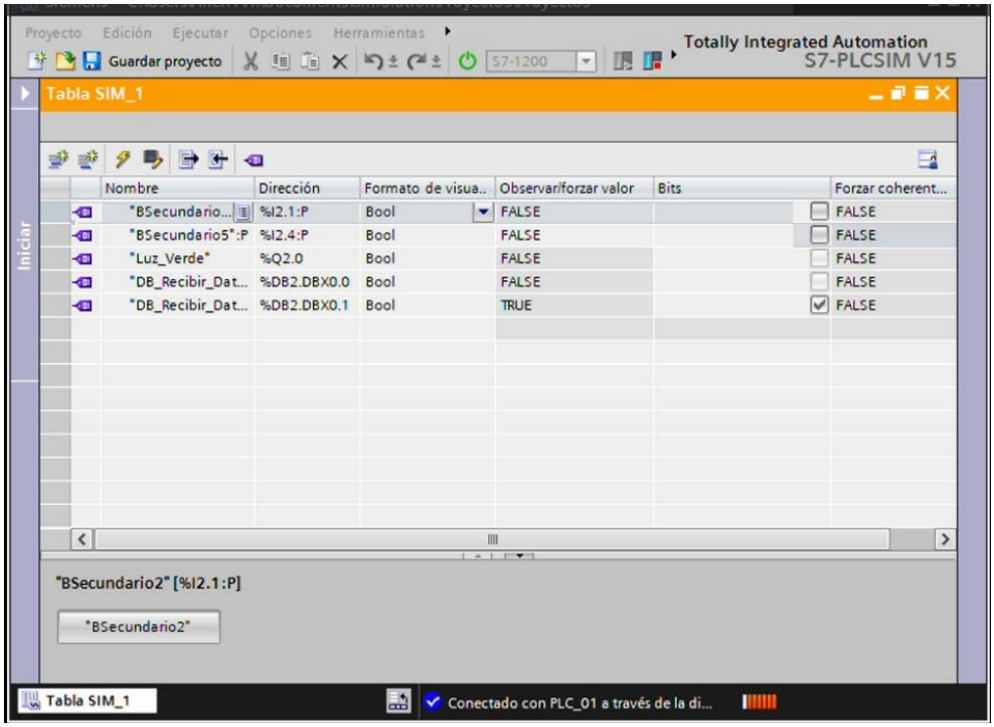


Imagen 52 Monitoreo desde la tabla SIM; apagado desde Unity

Ahora, se comprueba el envío de señales. Al activar la señal %I2.1 mediante una tabla SIM (Imagen 53), el inspector de Unity registra la señal; visualmente enciende el indicador luminoso verde (Imagen 54).

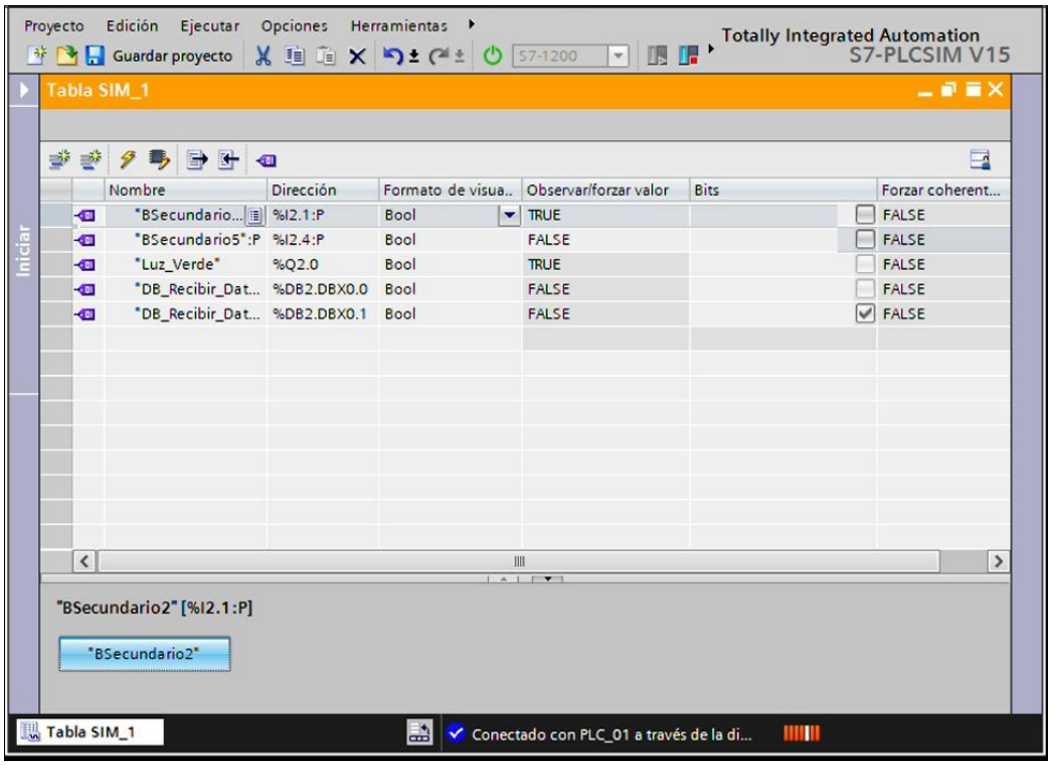


Imagen 53 Activación de la dirección %I2.1 (PLCSIM)

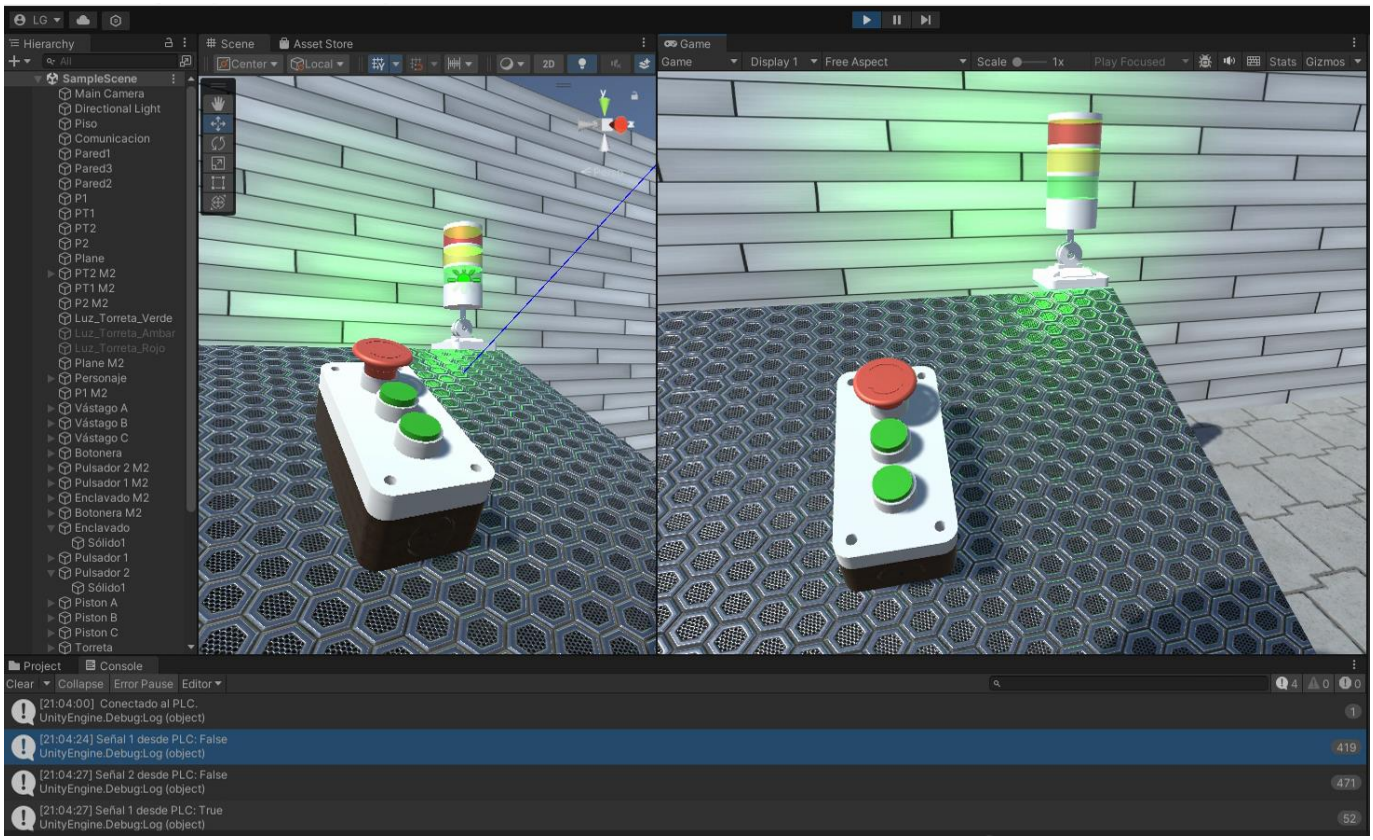


Imagen 54 Activación del indicador luminoso verde en Unity con la señal %I2.1

Finalmente, se activa la señal %I2.4 en TIA Portal por medio del PLCSIM (Imagen 55), la señal tiene como función apagar el indicador luminoso verde, tal y como se muestra en la Imagen 56. Esta señal también es monitoreada en el inspector de Unity.

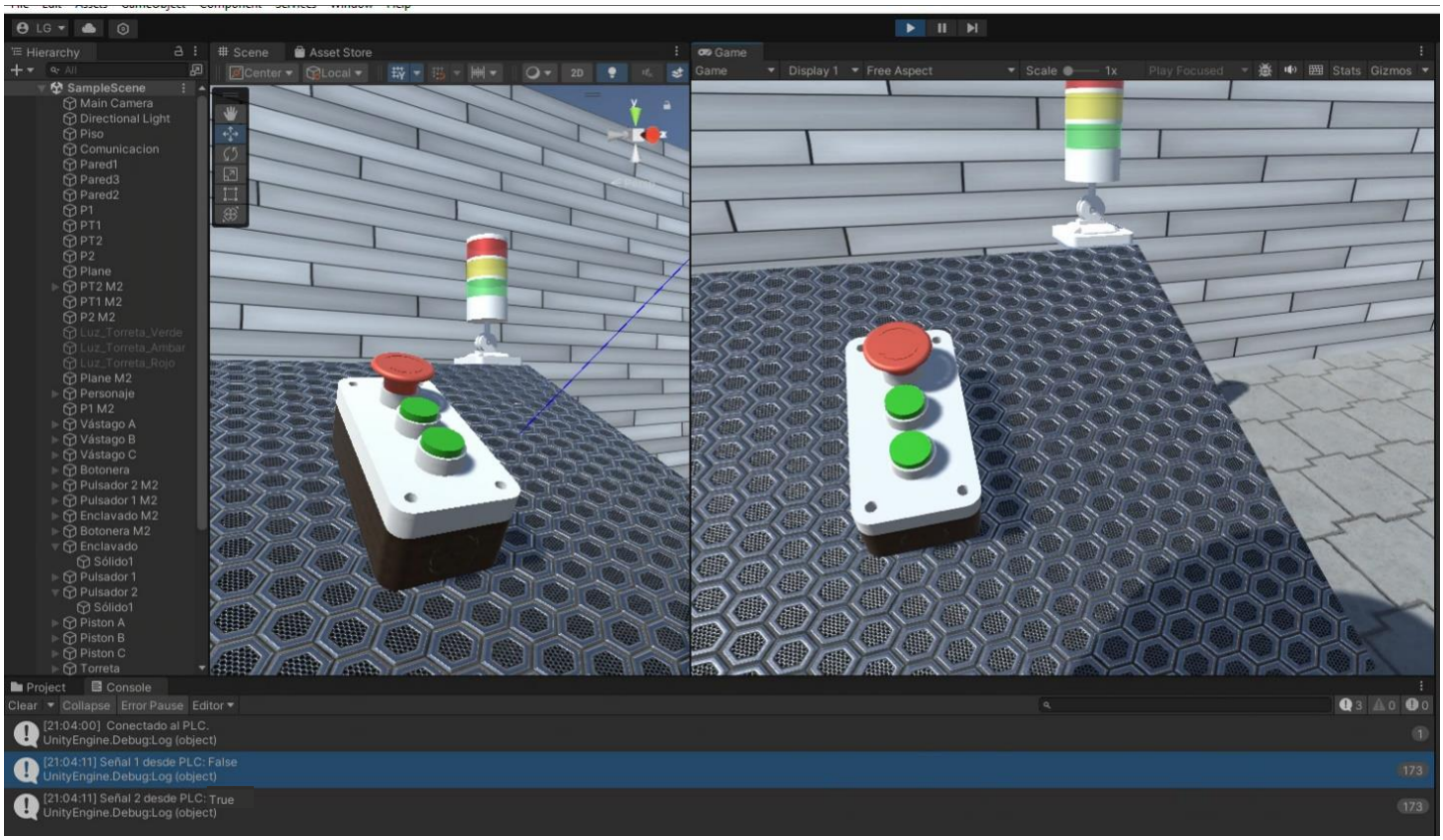


Imagen 55 Apagado del indicador luminoso verde mediante la señal %I2.4

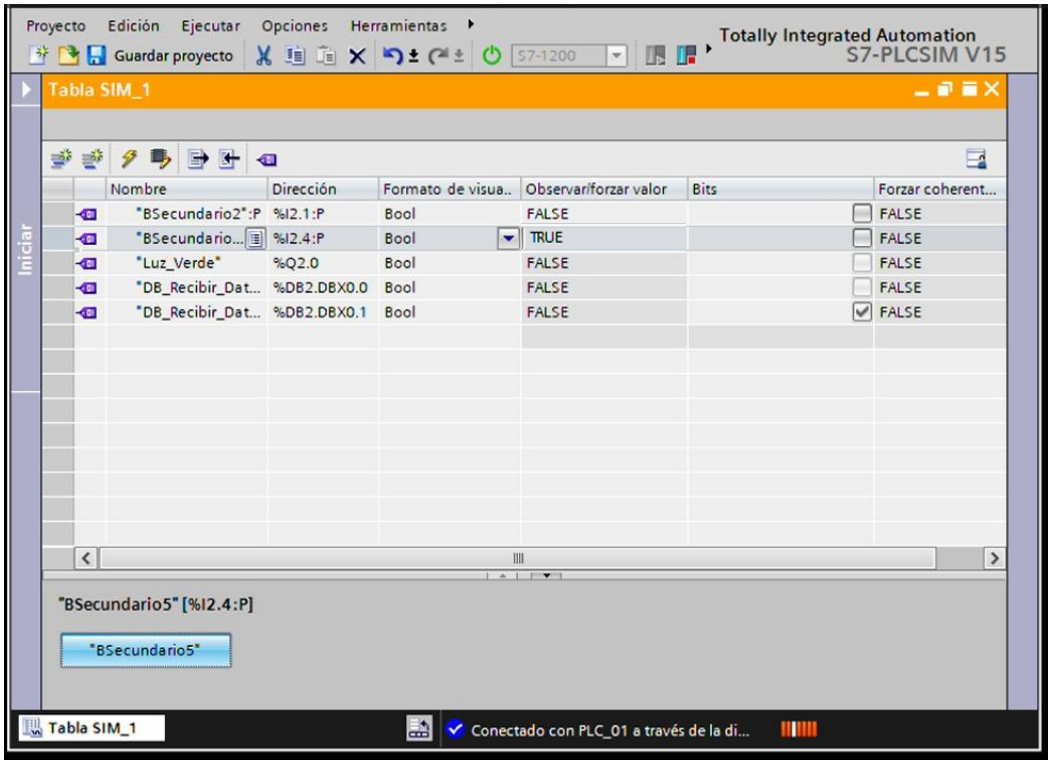


Imagen 56 Activación de la señal %I2.4 desde PLCSIM

- ***Prueba de expulsión y retroceso de un vástago de un cilindro de doble efecto***

De acuerdo a la lógica presentada en la sección 3.2.2 (Imagen 39), y de manera similar al ejercicio anterior, colocar dos nuevos peldaños que envían directamente la señal de los botones físicos al DB de envió de datos (Imagen 57); se actualiza el DB para tener ambos modelos funcionando (Imagen 58).

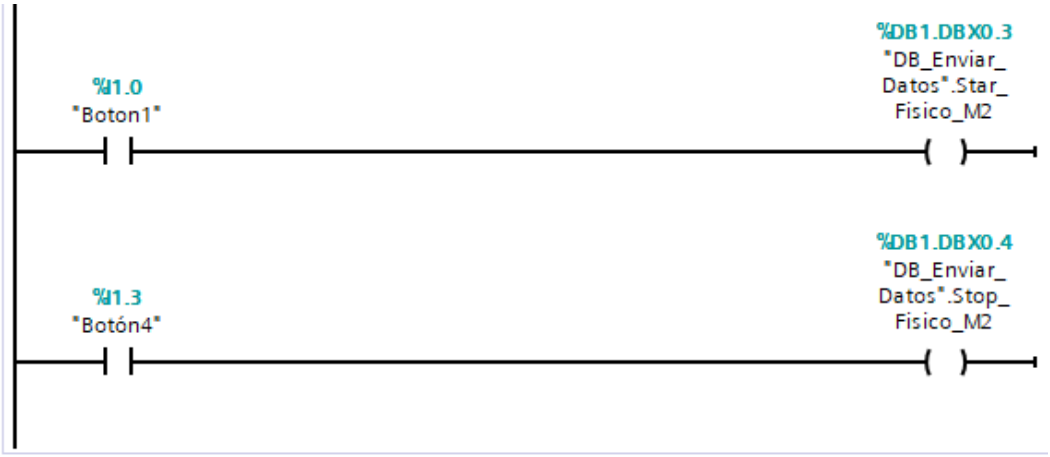
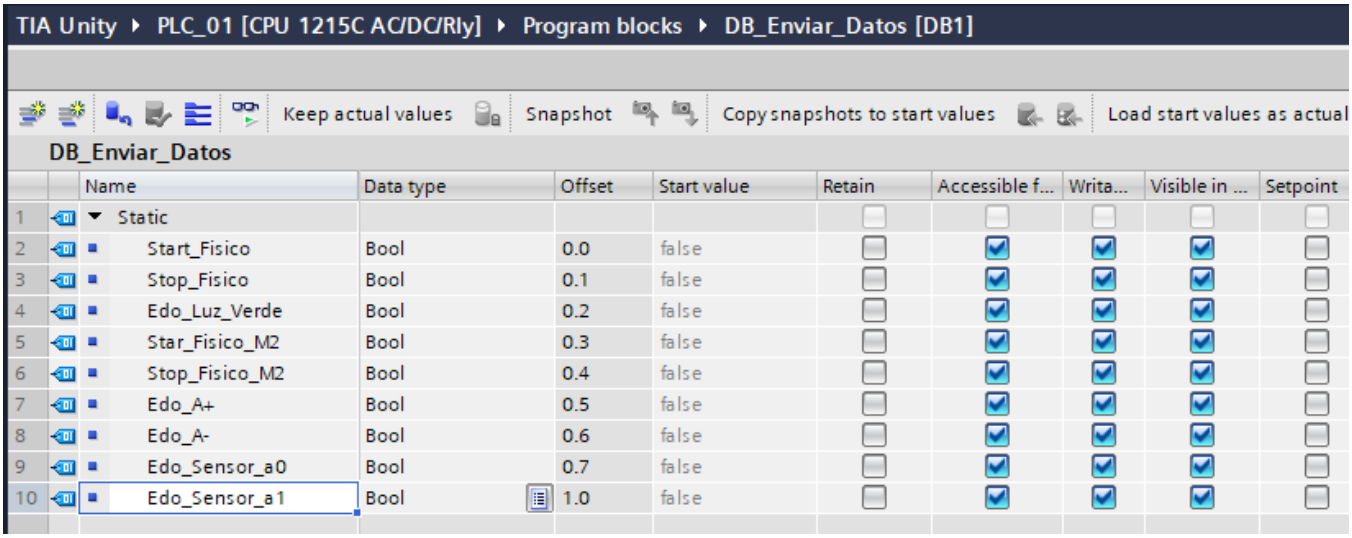


Imagen 57 Envío de señales físicas al ambiente virtual

Para este ejercicio, el botón de paro será compartido con la rutina de encendido del indicador verde sin embargo, para la activación de esta prueba, se utilizará otra señal de activación (%I1.0).



	Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint
1	Static								
2	Start_Fisico	Bool	0.0	false					
3	Stop_Fisico	Bool	0.1	false					
4	Edo_Luz_Verde	Bool	0.2	false					
5	Star_Fisico_M2	Bool	0.3	false					
6	Stop_Fisico_M2	Bool	0.4	false					
7	Edo_A+	Bool	0.5	false					
8	Edo_A-	Bool	0.6	false					
9	Edo_Sensor_a0	Bool	0.7	false					
10	Edo_Sensor_a1	Bool	1.0	false					

Imagen 58 DB "Enviar_Datos" (actualizado)

Al simular el entorno de TIA Portal, es importante considerar que las señales por default, son falsas. En un sistema físico, algunas de las señales podrían ser verdaderas inicialmente. Las señales activas representan una condición inicial del sistema, por ejemplo, un sensor Reed mandaría una señal verdadera si el vástago del pistón se encuentra retraído, indicando una posición de reposo. Por tanto, las condiciones iniciales en la tabla SIM para este ejercicio se muestran en la Imagen 59.

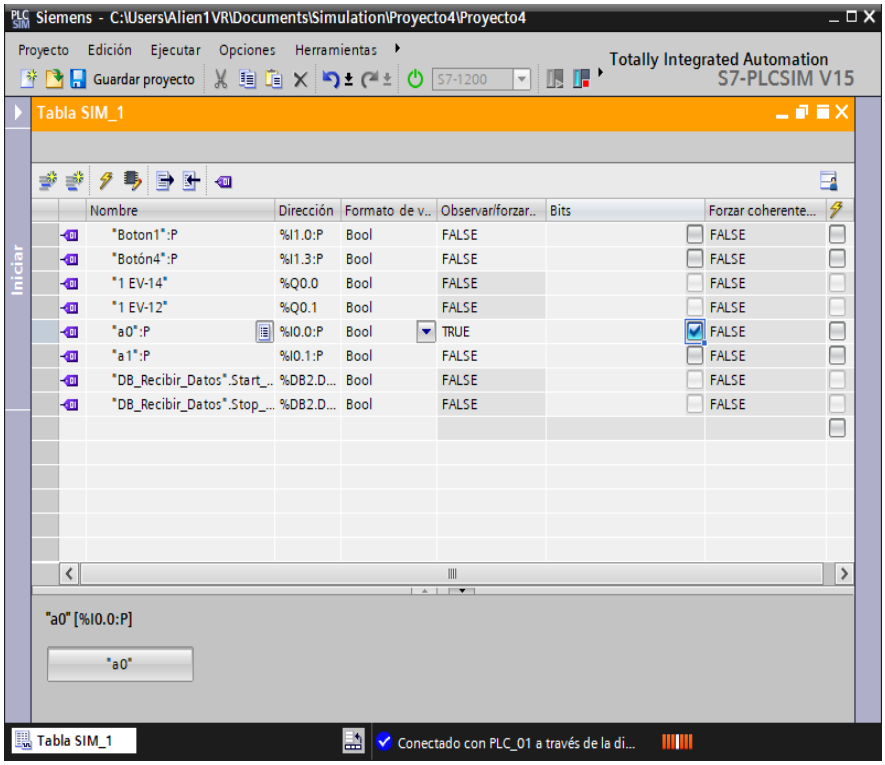


Imagen 59 Condiciones iniciales para prueba de expulsión de un vástago

En la mesa 2 del entorno virtual, el sensor Reed correspondiente a la señal “a0”, tiene que estar encendido (Imagen 60).

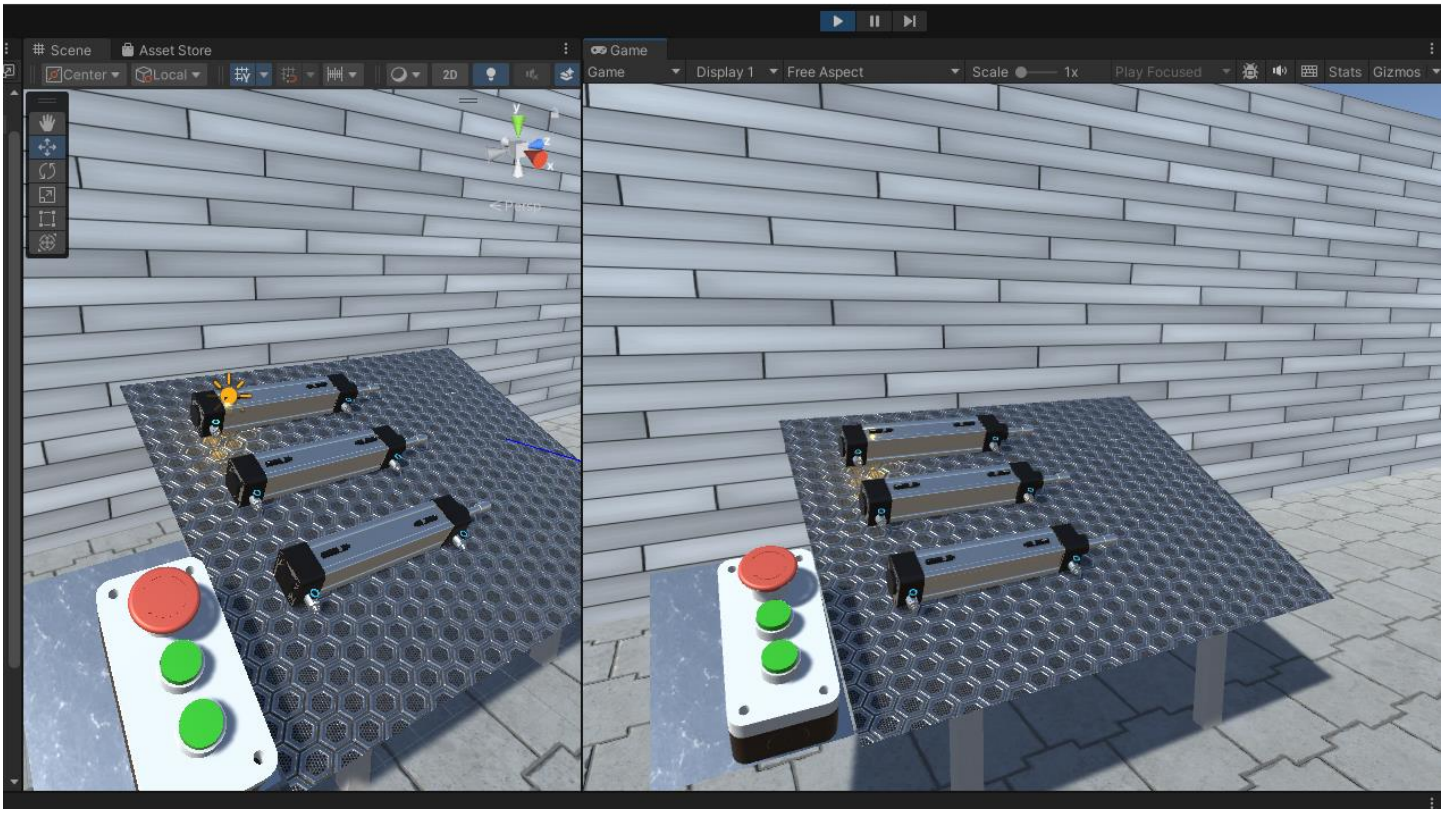


Imagen 60 Estado inicial del sistema virtual, Mesa 2

Desde Unity, se pulsa el botón inferior verde de la botonera (Imagen 61); en el entorno virtual se observa el desplazamiento del vástago del pistón A. En la tabla SIM se observa que la salida %Q0.0 es verdadera, lo cual indicaría la activación de la electroválvula del pistón A en su pilotaje 14 o posición de trabajo (Imagen 62).

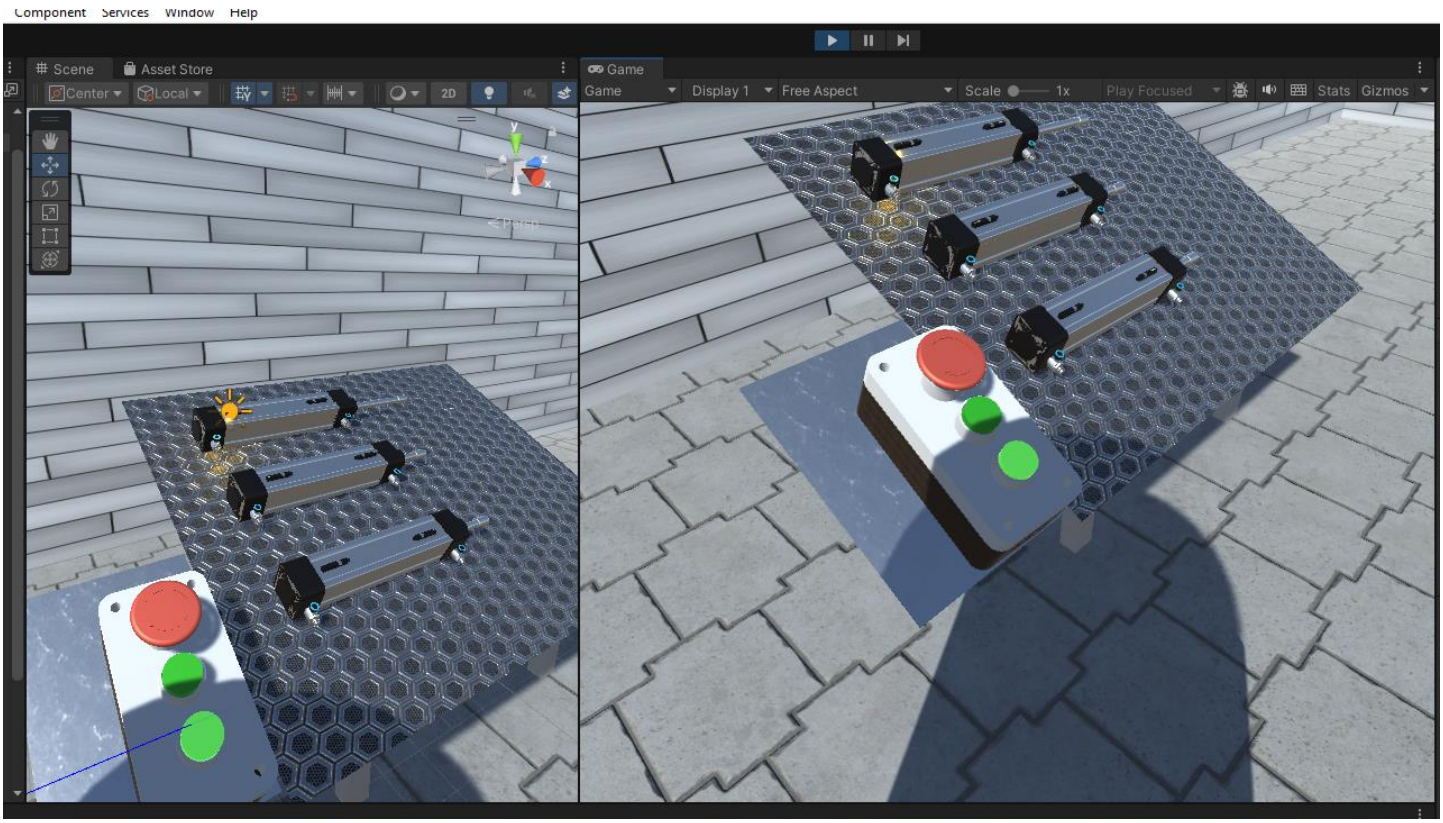


Imagen 61 Activación virtual del sistema; cambio a posición de extensión

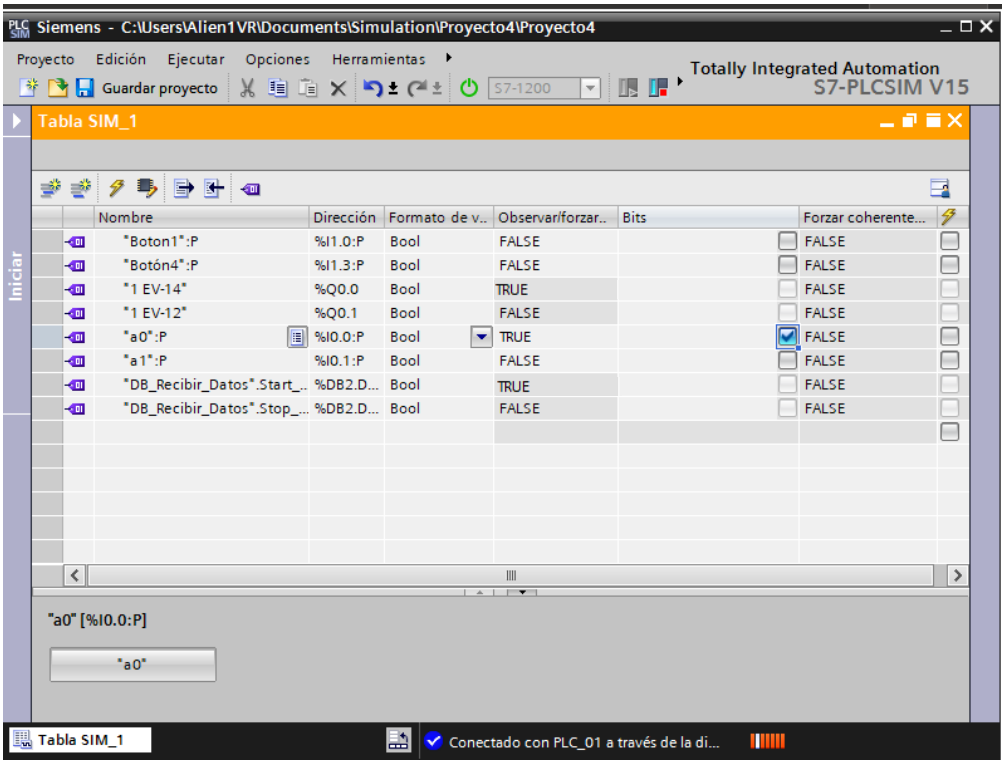


Imagen 62 Monitoreo desde tabla SIM; posición de extensión

En la Imagen 61 se aprecia que el vástago del pistón A se encuentra extendido pero la señal del sensor Reed “a0” permanece activa. Al tratarse de una simulación, se debe de activar y desactivar las señales manualmente, por lo que en la tabla SIM de la Imagen 62 debe desactivarse la casilla de la señal “%I0.0” (correspondiente a la señal a0), y activarse la casilla de la señal “%I0.1” (señal a1), tal y como se muestra en la Imagen 63. Una vez realizado este cambio, en el entorno virtual el sensor Reed para la señal a1, se enciende (Imagen 64).

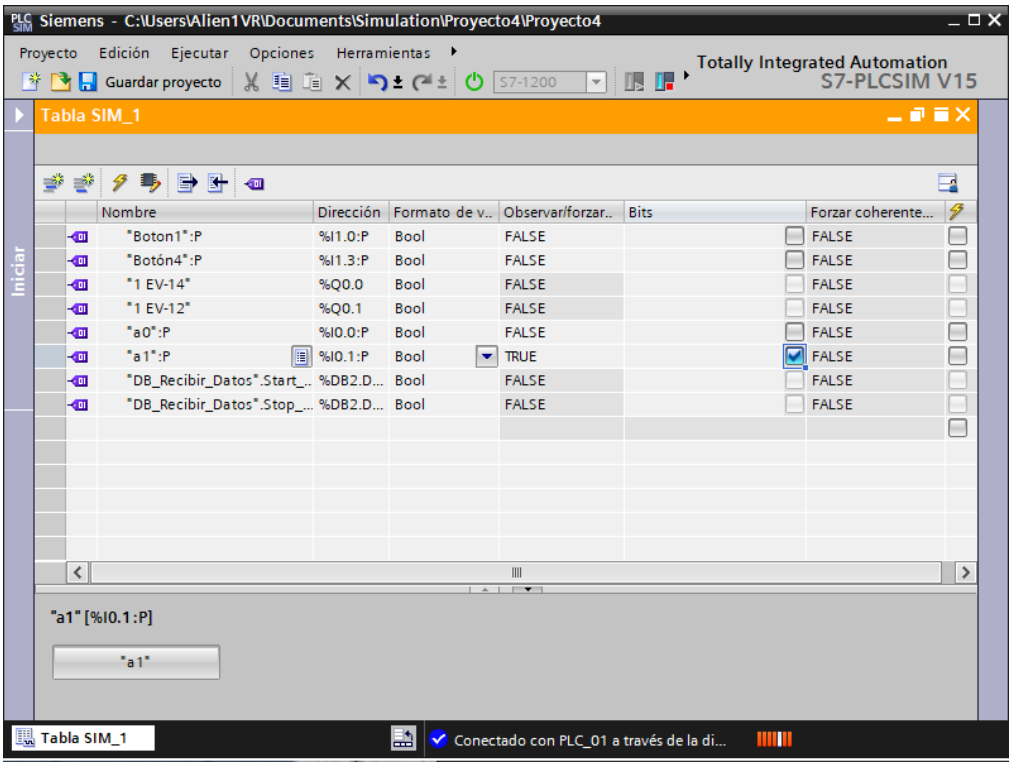


Imagen 63 Señal "a1"; activación manual

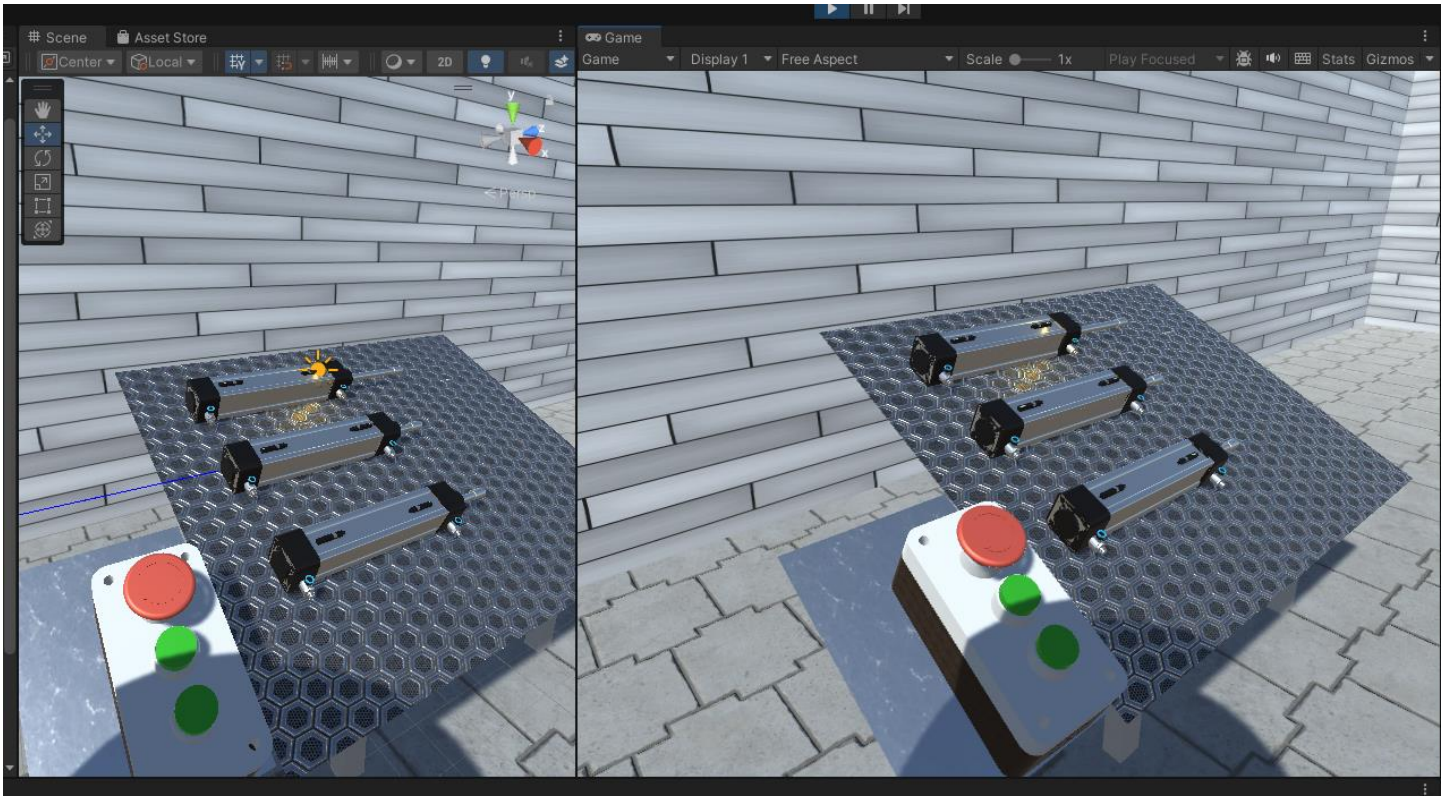


Imagen 64 Posición de extensión del vástago del pistón A

Para colocar el sistema en condiciones iniciales, pulsar el botón enclavado del sistema virtual o activar desde la tabla SIM la señal %I1.3, a continuación, desde la tabla SIM, desactivar la casilla de la señal a1 y activar la casilla de la señal a0. Con este el sistema está listo para una nueva interacción, ya sea física o virtual.

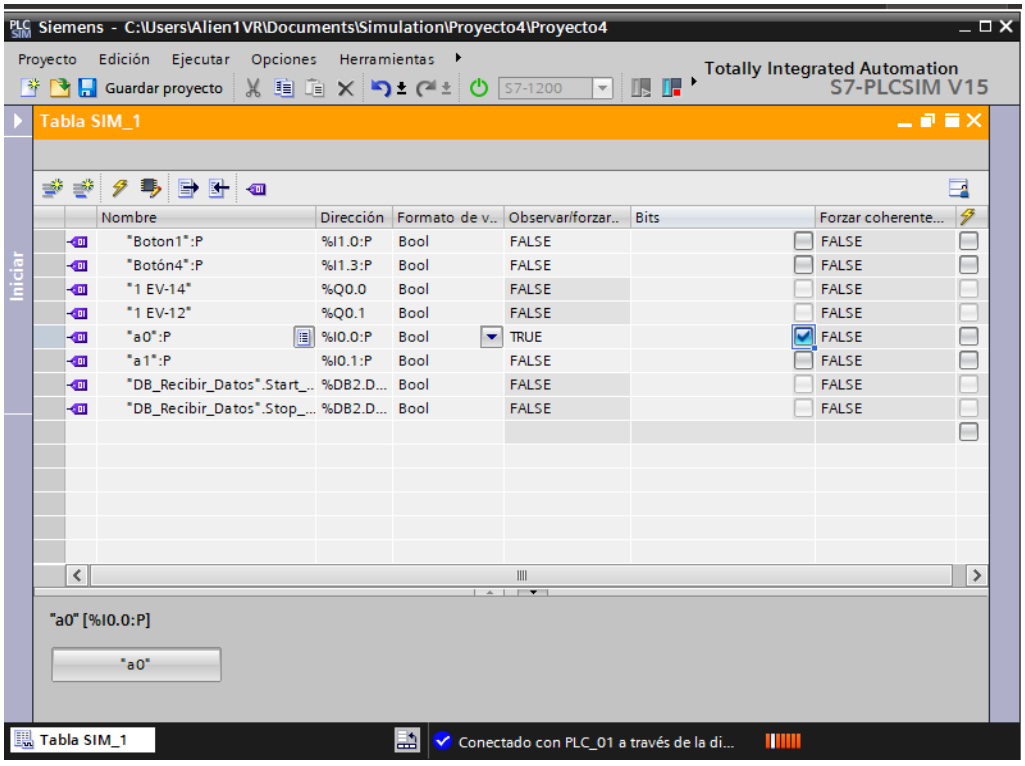


Imagen 65 Sistema en condiciones iniciales

3.3.2 Gemelo digital físico

El procedimiento para trabajar con el equipo que se encuentra en el LAI es más sencillo que de forma simulada. Para comprender el entorno de trabajo, observar la Imagen 66.

En el LAI se cuentan con 10 estaciones de trabajo que son casi idénticas. Cada estación cuenta con un PLC y una mesa neumática – electroneumática.

En la Imagen 66 se observan tres divisiones enmarcadas y numeradas en diferentes colores: rojo (1), azul (2) y amarillo (3).

→(1) Gabinete de control: Gabinete eléctrico que resguarda los PLC de las estaciones 1 y 2 del LAI. En la Imagen 66 se distingue una HMI; cabe aclarar que solo el *Gabinete 1* cuenta con esta interfaz. También observar múltiples conectores eléctricos a los costados del gabinete y en la parte superior una antena de comunicación.

→(2) Conectores externos: En el recuadro 1 se menciona que hay diversos conectores que salen del gabinete, dentro de estos conectores, se hace la distinción de unas mangueras naranjas las cuales contienen el cableado correspondiente al módulo de expansión con el que cuenta el PLC

→(3) Mesa de trabajo: Estación de trabajo que permite el desarrollo de prácticas neumáticas y electroneumáticas. Se encuentran elementos como temporizadores neumáticos, válvulas 5/2 con pilotaje neumático y pilotaje por solenoide (electroválvulas), pistón de simple efecto y doble efecto, sensores de efecto Reed, botones eléctricos y neumáticos entre otros dispositivos.

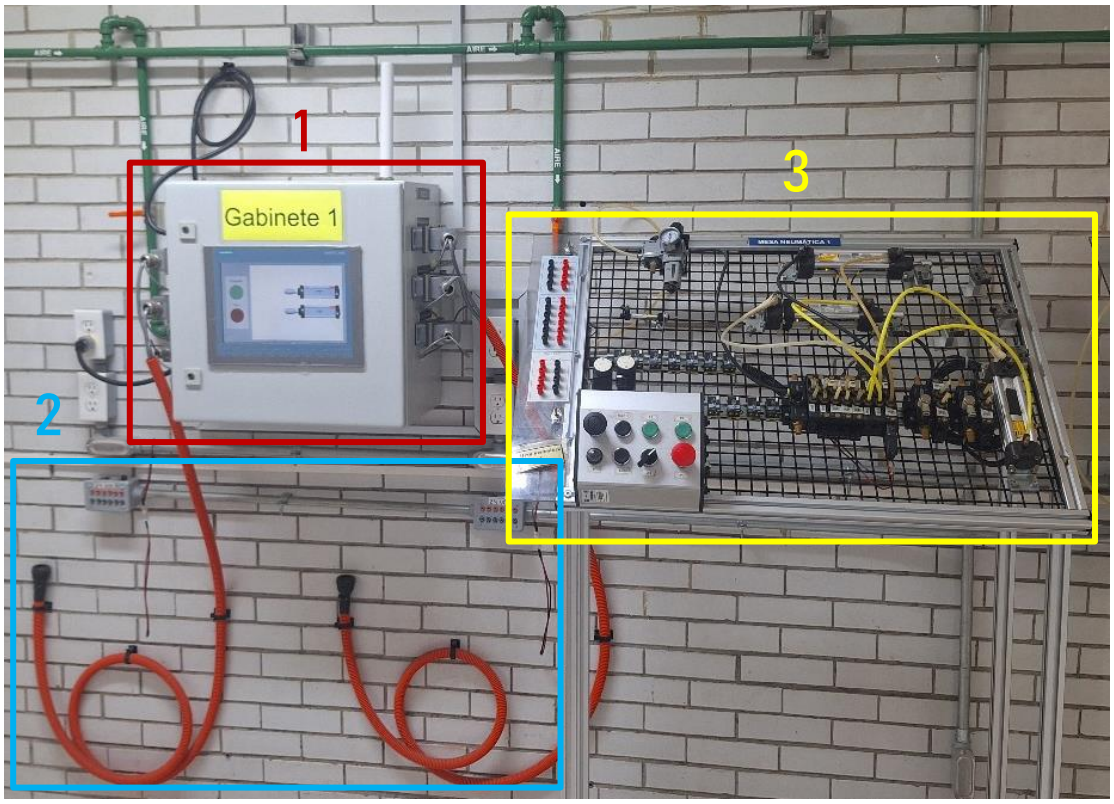


Imagen 66 Infraestructura LAI; 1: Gabinete de control; 2: Conectores externos; 3: Mesa de trabajo

Identificado el espacio de trabajo, se requiere de las siguientes etapas para establecer una comunicación Unity – TIA Portal:

- Conectar un cable Ethernet (CAT6 [Imagen 67]) del puerto RJ45 de la computadora al puerto de conexión del PLC ubicado en el *Gabinete 1* (Imagen 71).
- Cargar la rutina de control al PLC y ponerlo en modo ejecución (RUN); verificar que los datos sean enviados por el puerto de comunicación recién colocado.
- Debido a que el PLC que se trabaja no cuenta con una torreta, conectar los botones iluminados (Imagen 68) y una botonera externa en el módulo de expansión del PLC (Imagen 70); para este procedimiento se requiere conectar la caja de conexiones (Imagen 69) en la terminal naranja del gabinete; observar la Imagen 66 en el recuadro número 2.
- Verificar que la computadora de trabajo esté en el segmento de red del PLC (Imagen 72).
- Ejecutar la escena de Unity; ahora la lectura del estado de los sensores Reed se modificará en automático. Los botones eléctricos que se ubican en la mesa neumática permiten la activación y desactivación del sistema; utilizar la Imagen 66 (recuadro 3) para mayor referencia.

Imagen 67 Cable Ethernet CAT6A

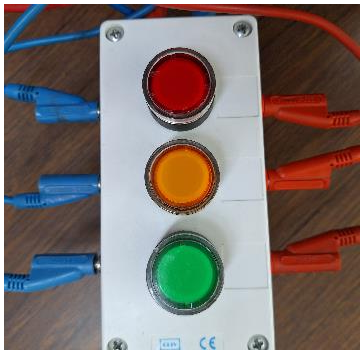


Imagen 68 Botones iluminados

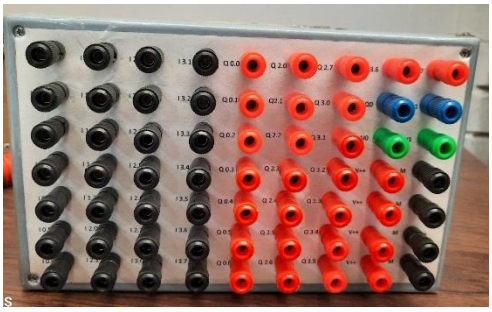


Imagen 69 Caja de conexiones para PLC S7-1200

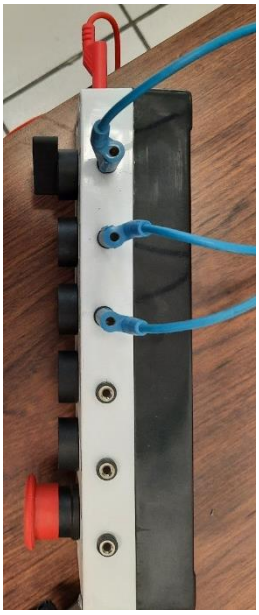


Imagen 70 Botonera secundaria (vista lateral)

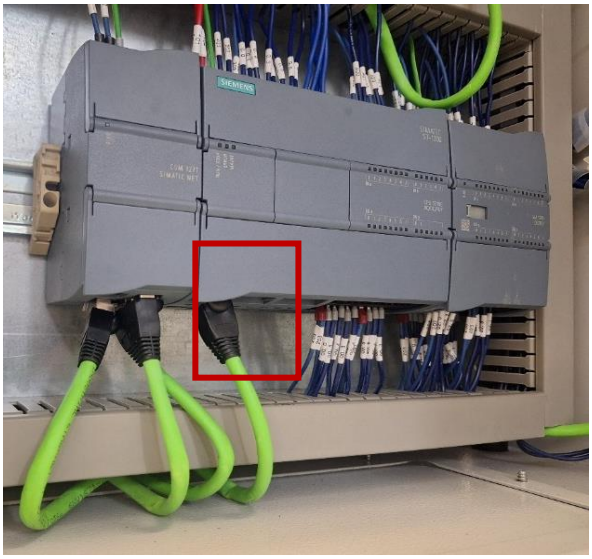


Imagen 71 PLC S7-1200; terminal disponible del CPU

Para verificar si el equipo de cómputo está conectado al nodo de red en una conexión por Ethernet, es necesario entrar a las configuraciones de red del panel de control (Imagen 72).

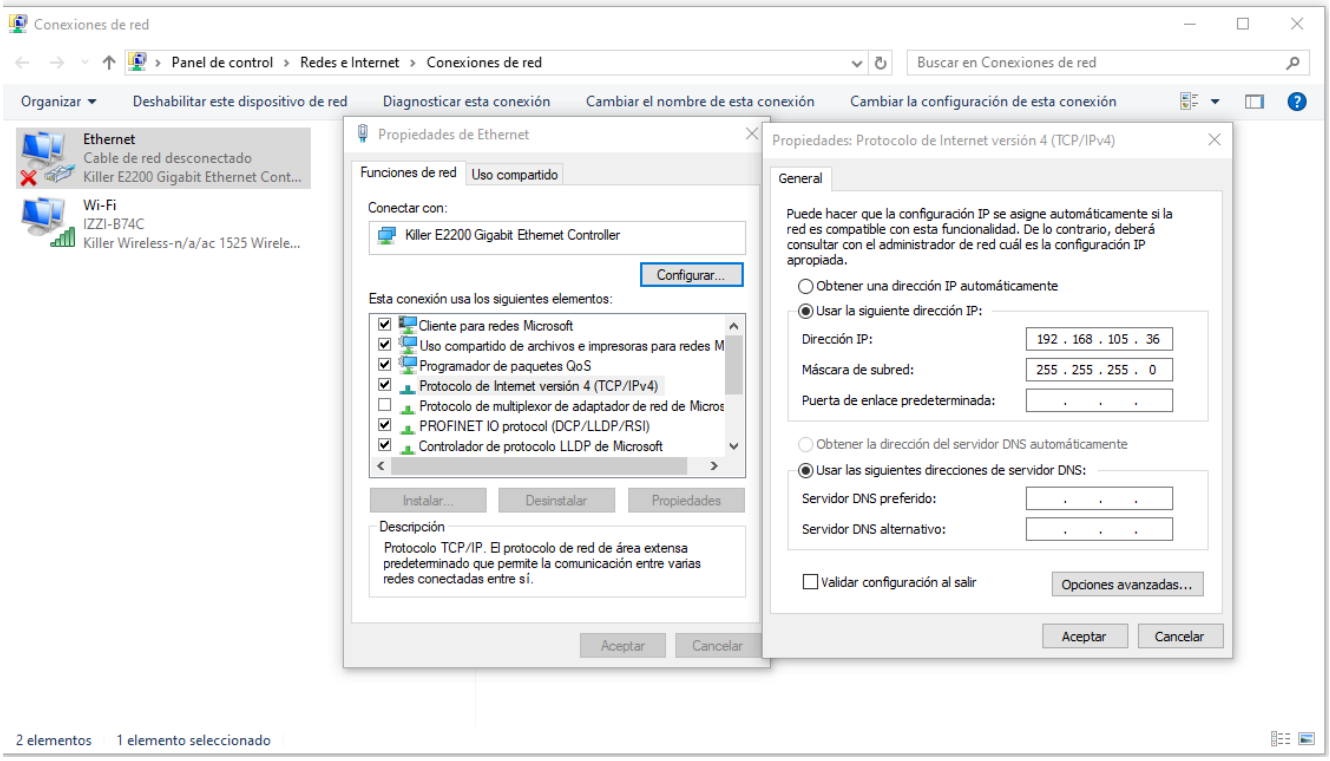


Imagen 72 Verificación de acceso al segmento de red (192.168.105.###)

- ***Prueba de encendido de indicador luminoso.***

Ahora solo es necesario tener la rutina de control de TIA Portal en modo monitoreo y ejecutar la escena de trabajo de Unity.

Cuando se monitorea una rutina de control en el PLC, es más sencillo entender qué está ocurriendo en el momento. TIA Portal ofrece una interfaz sencilla de analizar; al acceder a la sección *Main*, la ventana principal permite observar los peldaños de programación.

En la Imagen 73 se aprecia que los peldaños se encuentran iluminados en dos tonalidades. Del lado izquierdo (cerca de la pestaña “Project tree”), en color verde; del lado derecho, en azul. La tonalidad verde indica que el sistema contiene energía eléctrica y que a partir de las señales que se reciban en los contactos, la energía puede continuar sobre el peldaño o detenerse su paso. En la imagen 74 se aprecia el tránsito de energía al presionar un botón físico que asocia su señal al contacto con la dirección %I2.1.

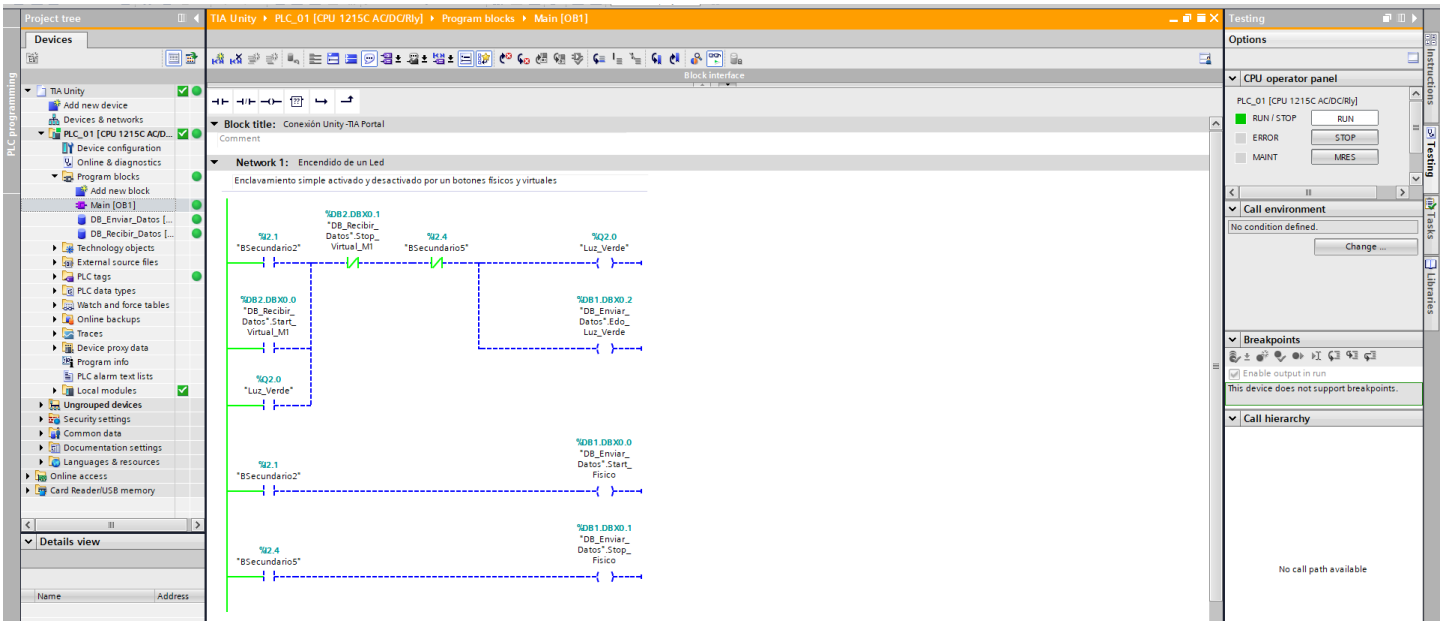


Imagen 73 Estado inicial del sistema en TIA Portal

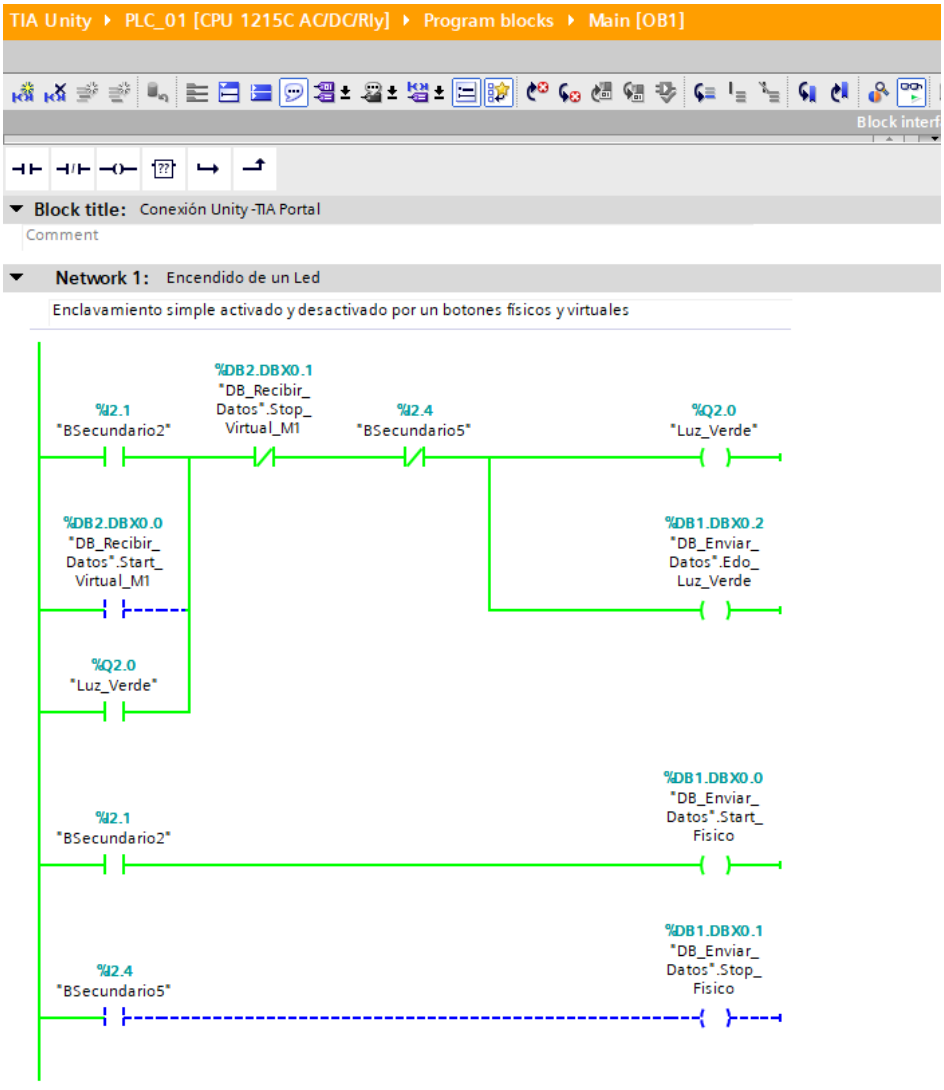


Imagen 74 Activación desde botón pulsador físico

Previamente a arrancar con la ejecución del código, es necesario modificar la IP en el script de comunicación, ya que, en las pruebas anteriores (sistema simulado) la red a la que estaba conectado el sistema virtual ya no existe: La información debe ser direccionada a la IP del controlador físico (192.168.105.121), tal y como se muestra en la Imagen 75.

```
void Start()
{
    plc = new Plc(CpuType.S71200, "192.168.105.121", 0, 1);   //Simulado: "127.0.0.1" Físico "192.168.105.121"

    try
    {
        plc.Open();
        Debug.Log(" Conectado al PLC.");
    }
    catch
    {
        Debug.LogError(" No se pudo conectar al PLC.");
    }
}
```

Imagen 75 Cambio de IP para comunicación con el sistema físico

Para un mayor entendimiento de los cambios requeridos en los códigos de los scripts, consultar el anexo de este documento.

Realizados los cambios pertinentes, se ejecuta la escena de Unity. Con ambos sistemas en línea y sin ningún error en la consola de Unity, se presiona el botón físico ubicado en la botonera externa (%I2.1). En la escena de Unity se aprecia el efecto en el indicador luminoso verde de la torreta (Imagen 76).

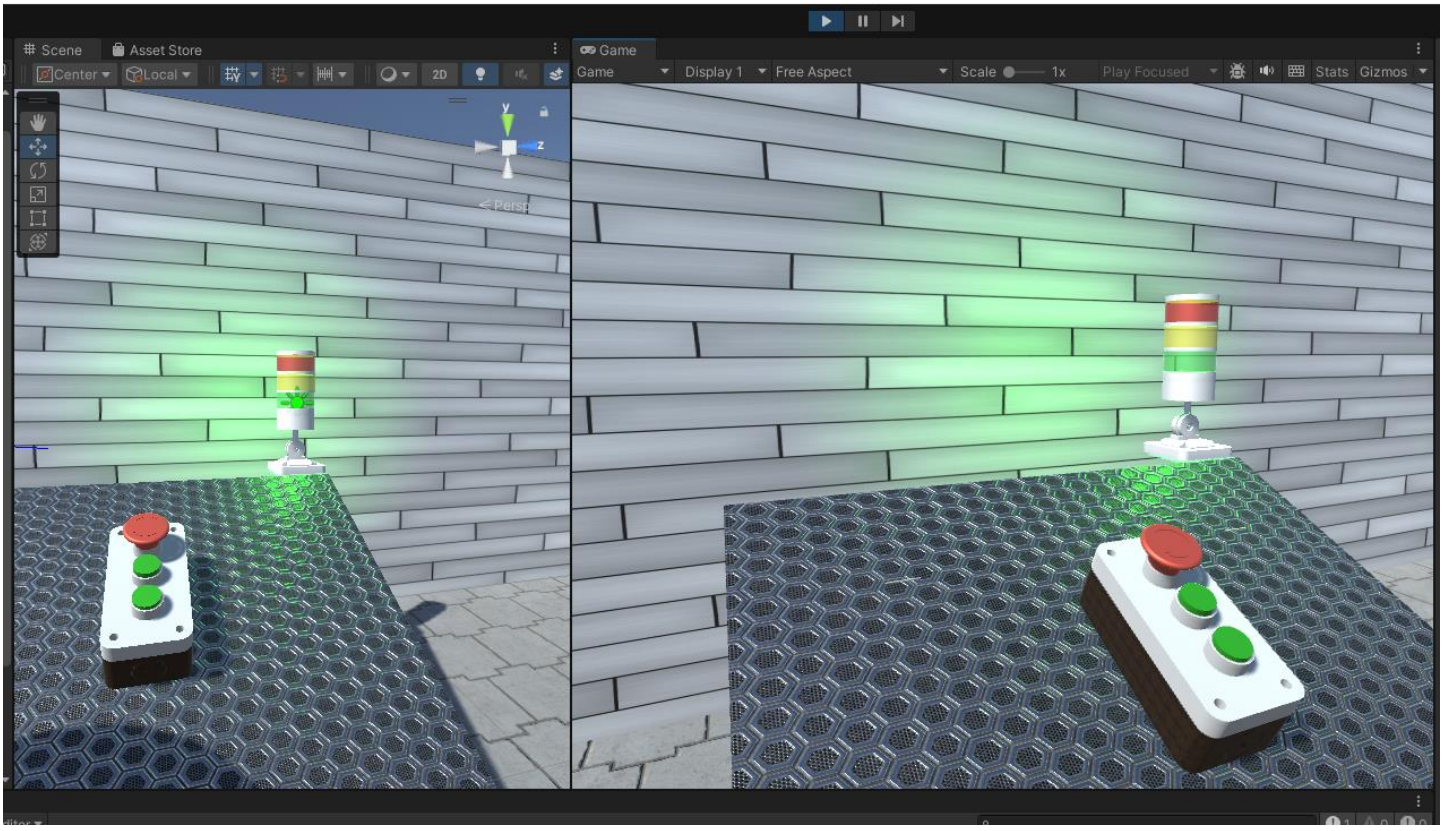


Imagen 76 Respuesta a la activación física de botón pulsador en el entorno de Unity

En TIA Portal el comportamiento es como se muestra en la Imagen 74, el sistema se mantendrá retroalimentado por la lógica de autoenclavamiento. A pesar de que se pulse múltiples ocasiones el botón, el sistema no fallará. Si se presiona el botón enclavado físico (%I2.4), el indicador luminoso verde se apaga y permanece así hasta que se

desenclave la señal %I2.4 y nuevamente el botón %I2.1 active el sistema. En la ilustración 77 se observa que se envía la señal del botón %I2.4 (botón enclavado) a Unity.

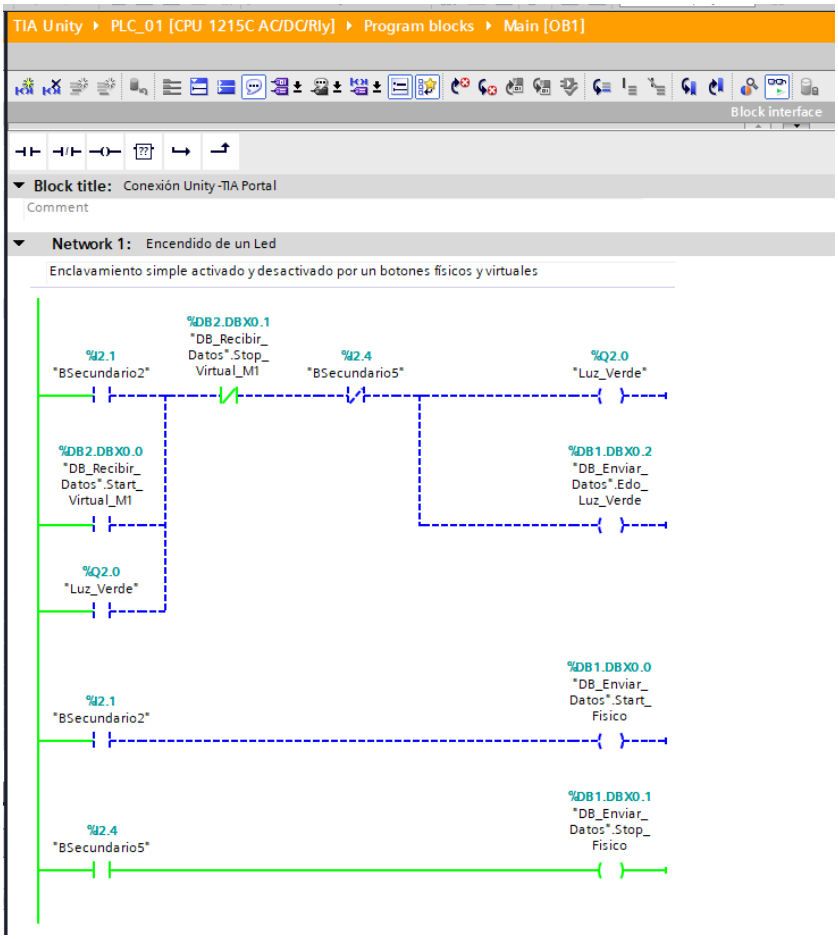


Imagen 77 Activación de botón enclavado físico (apagado de luz)

Si se presiona el botón virtual de la Mesa 1 (botón inferior verde), el sistema también responderá y encenderá el indicador luminoso verde, como se muestra en la Imagen 78. Este evento también es posible visualizarlo en la interfaz de TIA Portal (Imagen 79) y de forma física en el led verde (Imagen 80).

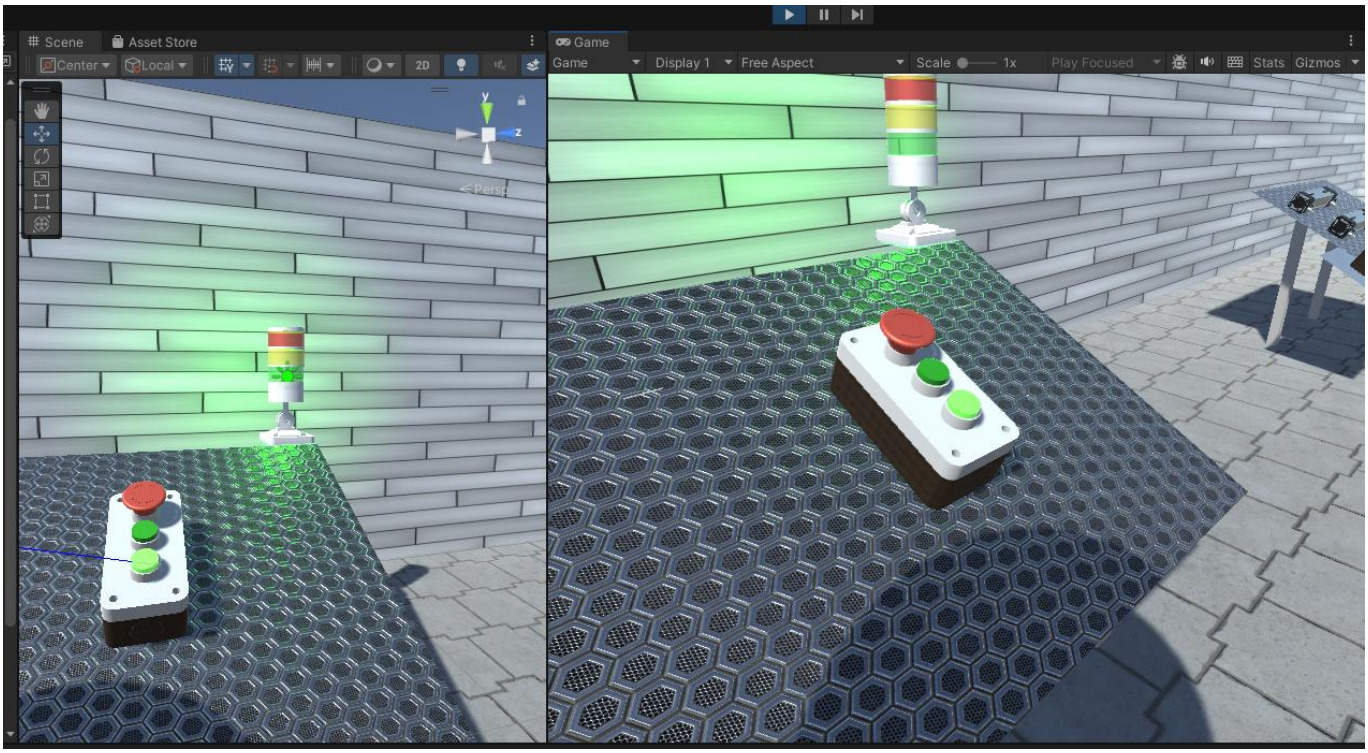


Imagen 78 Activación de sistema mediante botón pulsador virtual

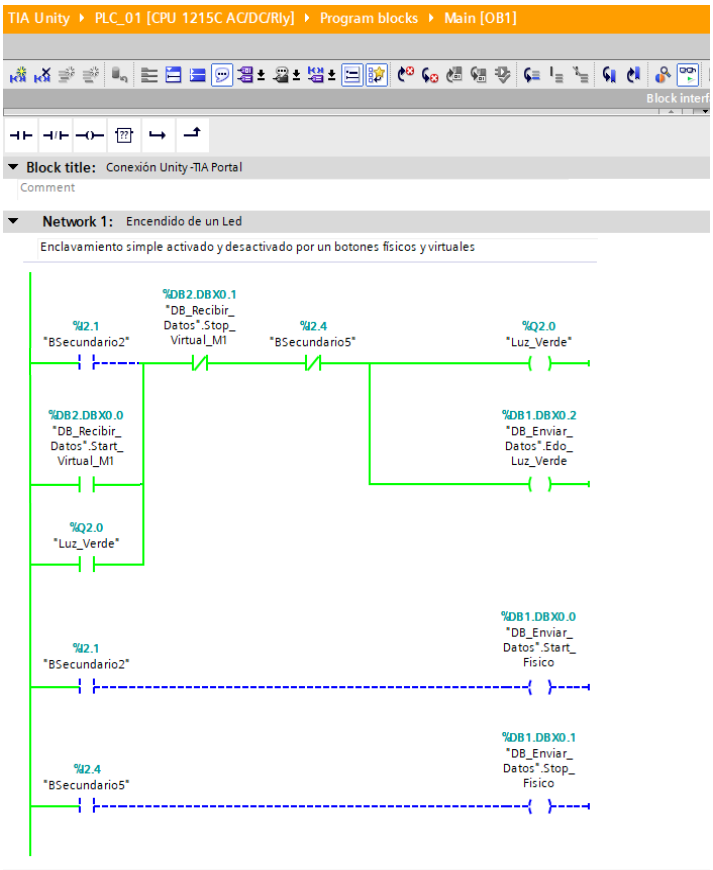


Imagen 79 Desactivación del sistema mediante un botón enclavado físico

Imagen 80 Estado de led verde

De manera similar, si se presiona el botón enclavado en la Mesa virtual 1, el indicador luminoso se apagará, como lo muestra la ilustración 81.

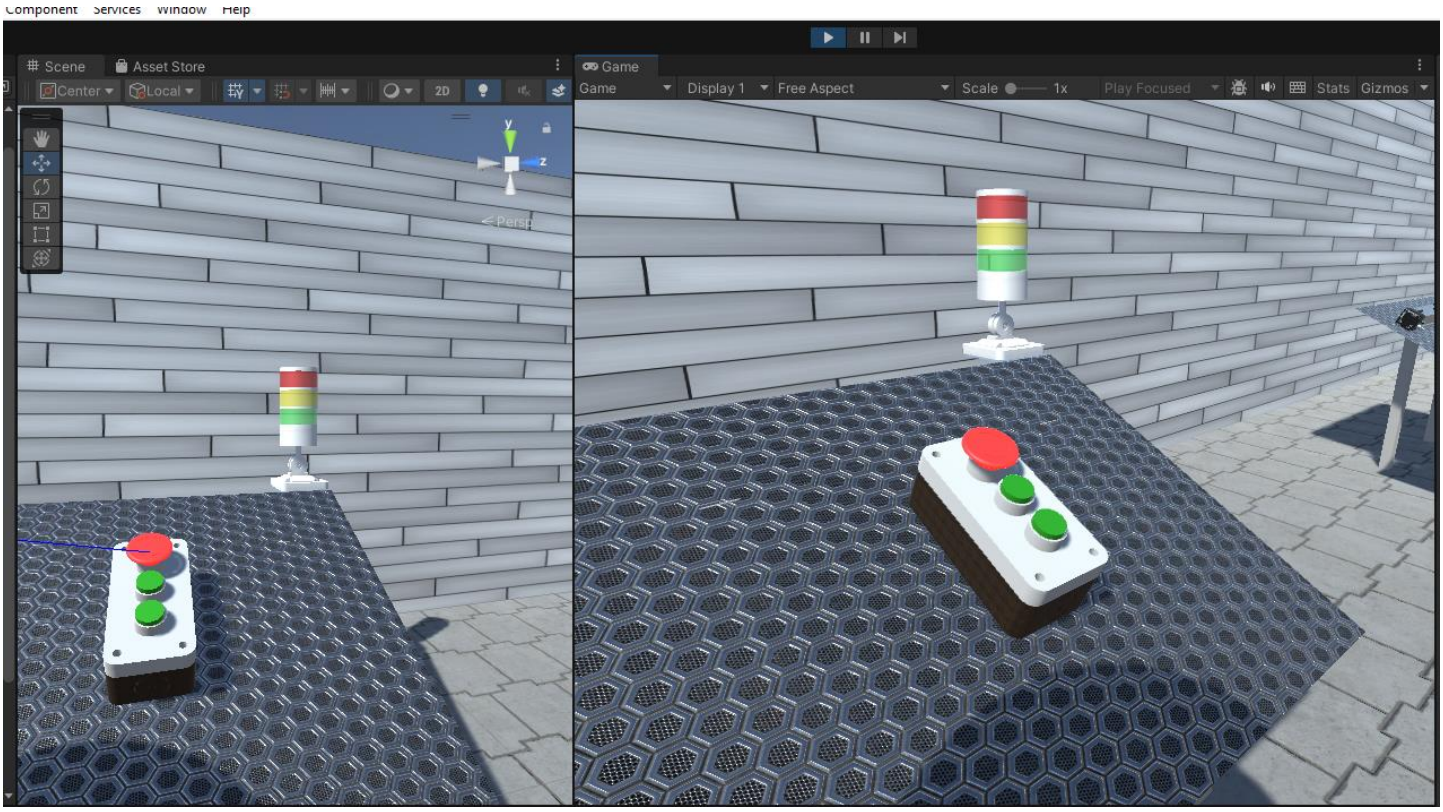


Imagen 81 Apagado de sistema mediante un botón enclavado virtual

- ***Prueba de expulsión y retroceso de un vástago de un cilindro de doble efecto***

Teniendo en cuenta que la prueba de encendido del indicador luminoso ocurre en la Mesa virtual 1, no existe restricción alguna para cambiar al ejercicio de extensión y retroceso en la misma ejecución del programa, tanto de TIA Portal (en modo monitoreo) cómo de Unity (escena en ejecución). Si el programa es detenido o se arranca por primera ocasión, los pasos a seguir son:

1. Verificar que las conexiones sean las adecuadas:
   - Botonera → módulo de expansión
   - Botones iluminados → módulo de expansión
   - Mesa neumática (física) este energizada → switches de cola de rata en posición "encendido"
   - Conexión con cable Ethernet PLC – computadora.
2. Ejecutar TIA Portal → activar el modo monitoreo como se muestra en la imagen 82
3. Ejecutar la escena de Unity → desplazarse a la Mesa virtual 2

En la imagen 82 se observa que la rutina de control está a la espera de la señal de inicio; también se percibe que la señal "a0" se envía al bloque de datos "DB_Enviar_Datos", indicando que el vástago del pistón se encuentra retraído.

Al activar la señal %I1.0 (botón pulsador físico), el sistema activa la electroválvula del pistón A en su pilotaje 14, ocasionando el movimiento de extensión y al mismo tiempo, enviando una señal a Unity para activar la animación correspondiente (Imagen 83). Cuando el vástago llega a su posición de extensión, la señal "a1" se vuelve verdadera y se envía al entorno virtual, tal y como se muestra en la Imagen 84.

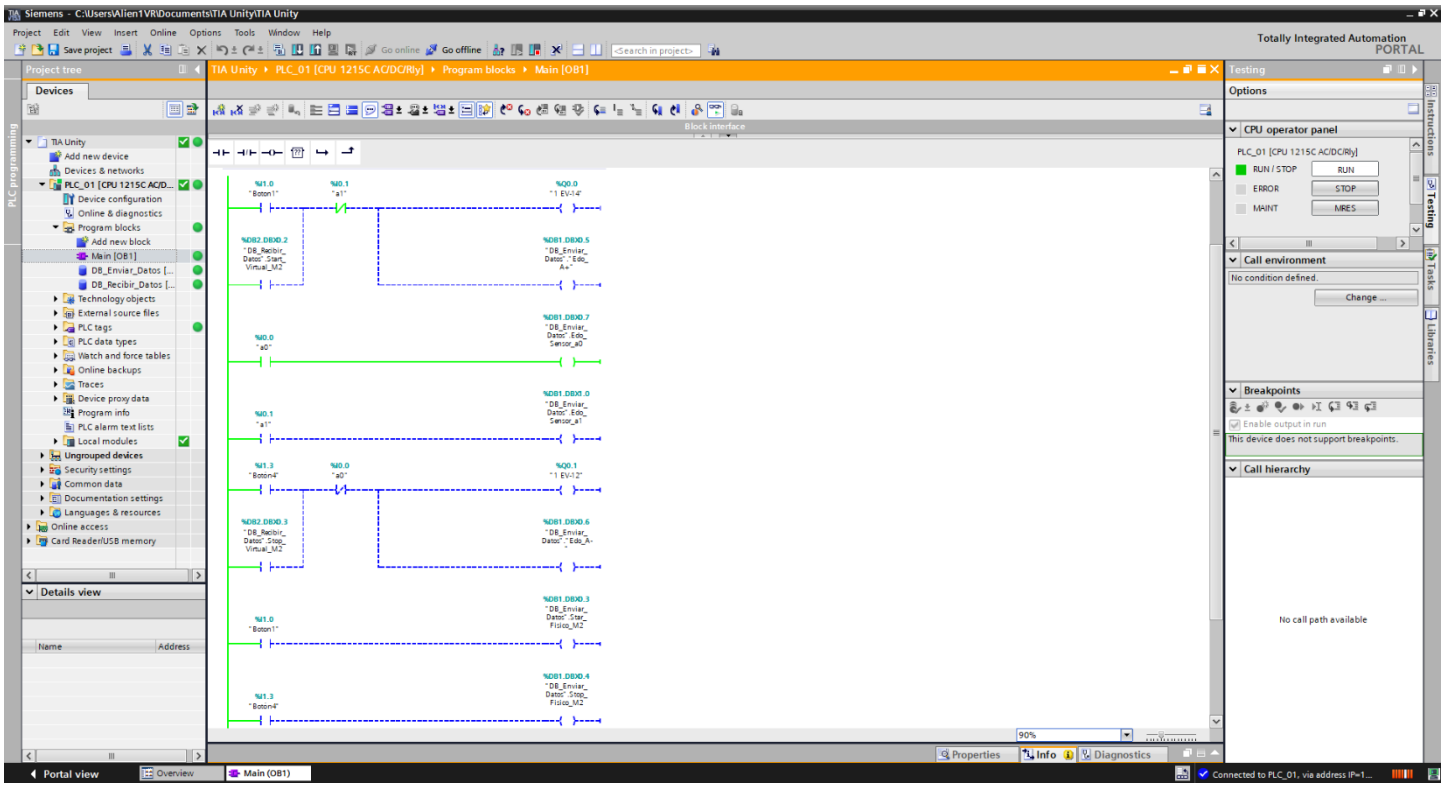


Imagen 82 Monitoreo en TIA Portal del sistema en estado inicial

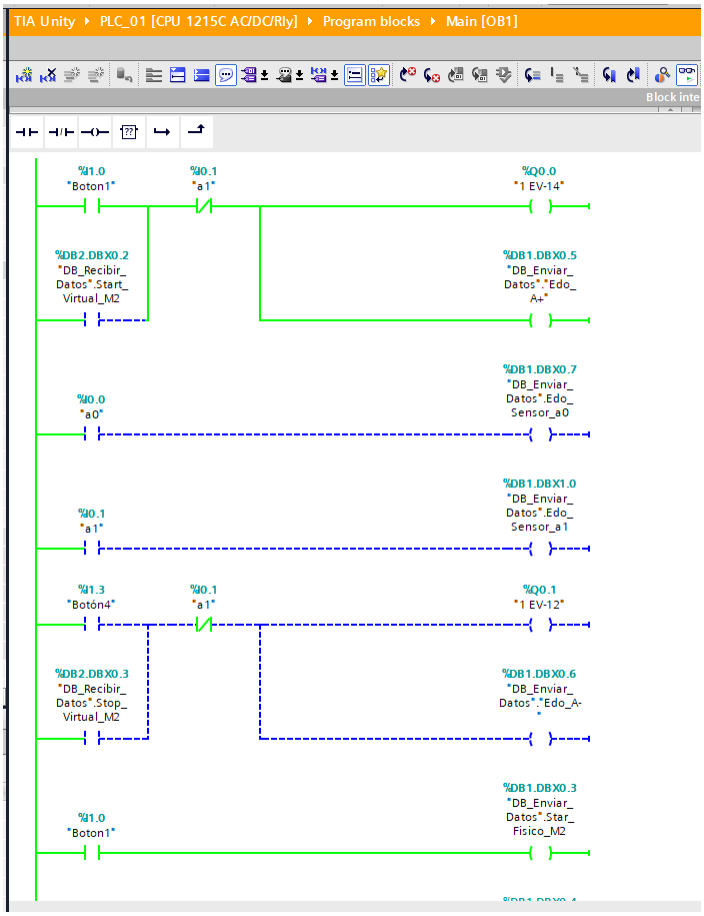


Imagen 83 Activación de %I1.0 (botón pulsador físico); "a0" deja de detectarse

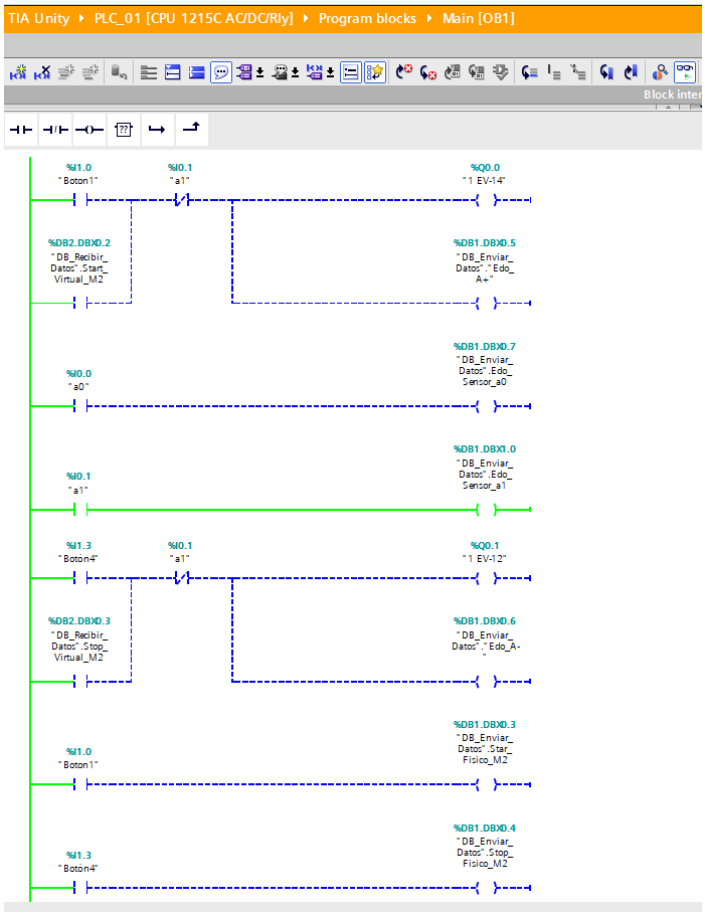


Imagen 84 Posición extendida del vástago; "a1"es detectado

Regresando a la escena de Unity, es posible notar la animación del desplazamiento del vástago del pistón A, así como la activación del sensor Reed correspondiente, “a1” (Imagen 85), semejante a como sucede en los elementos físicos (Imagen 86).

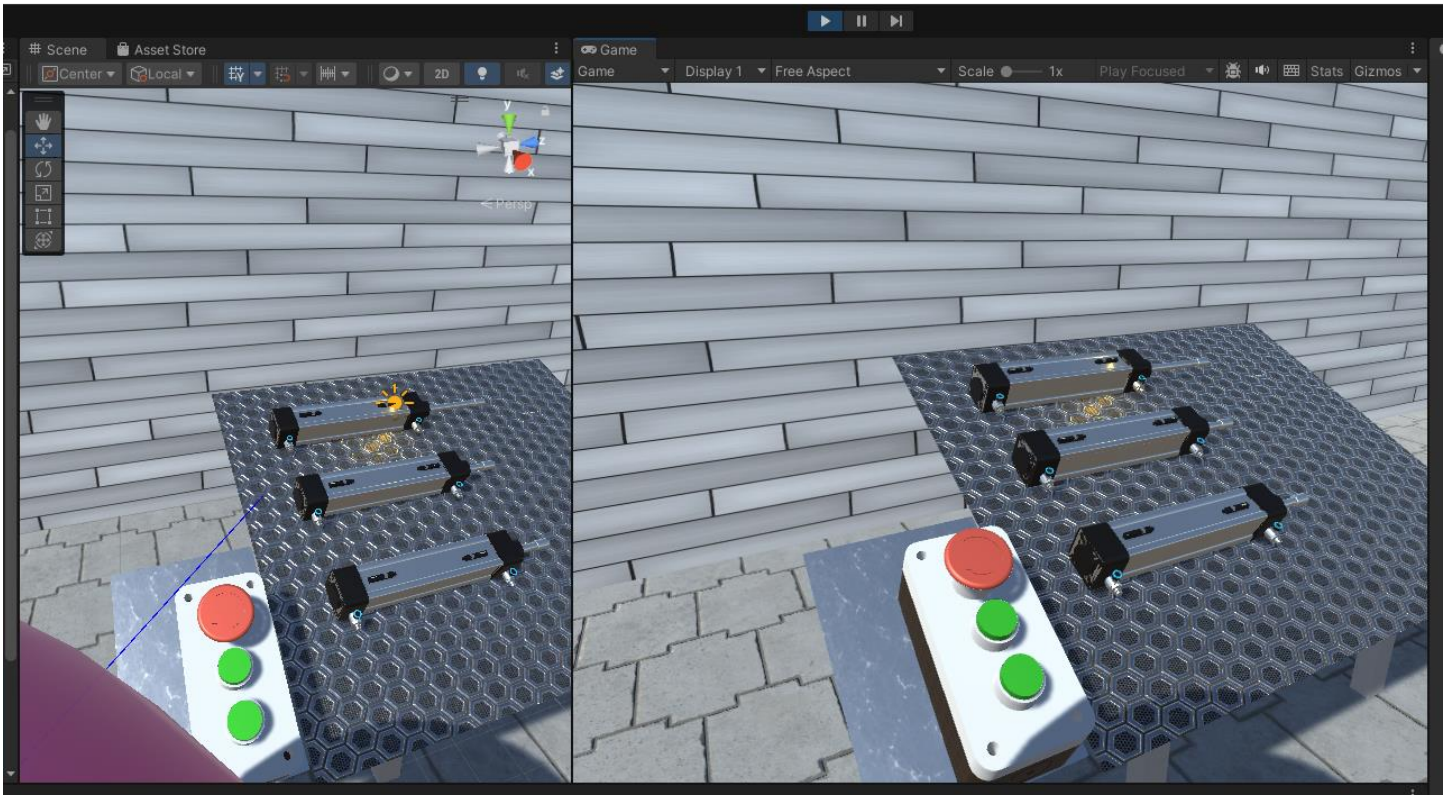


Imagen 85 Posición extendida del vástago desde Unity

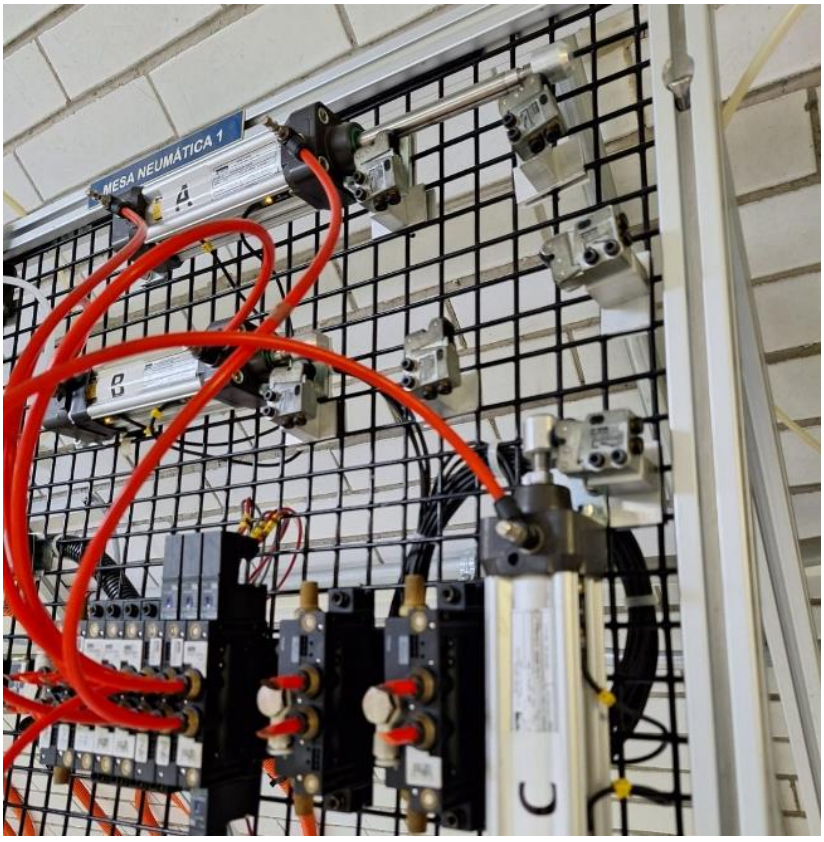


Imagen 86 Extensión de vástago; mesa neumática LAI

Para activar el retroceso del sistema, activar la señal %I1.3 (botón enclavado), que activa la electroválvula del pistón en el pilotaje 12. En la Imagen 87 se aprecia la activación del botón enclavado, en respuesta, también se envía una señal al entorno virtual para que realice la animación de retroceso del vástago. En la Imagen 88 se observa que, aunque el botón enclavado siga activo, la señal del sensor "a0" evita que la energía continúe en el peldaño, además esta misma señal es enviada a Unity.

En la parte inferior de la Imagen 88 se aprecia que la señal del botón enclavado es enviada al entorno virtual; esto tiene como objetivo deshabilitar el funcionamiento del botón pulsador virtual, ya que, de accionarlo, en el sistema virtual, se activaría la animación de expulsión. En el equipo físico, se puede implementar un contacto (normalmente cerrado) en el peldaño de activación del botón pulsador. No obstante, para este ejercicio no es necesario.

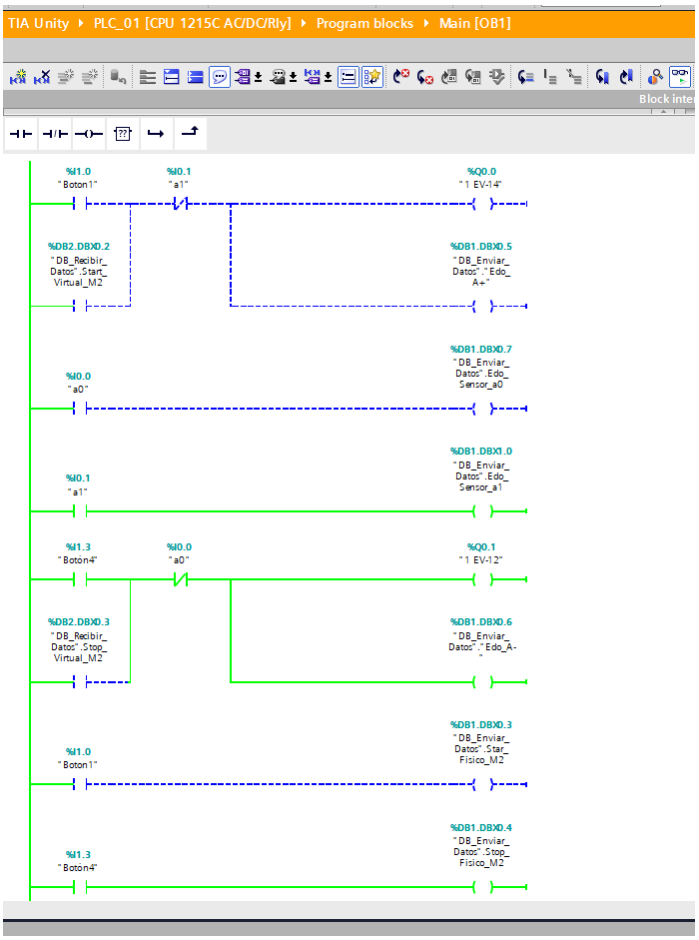


Imagen 87 Señal %I1.3 activa el pilotaje 12 de la electroválvula (EV-12)

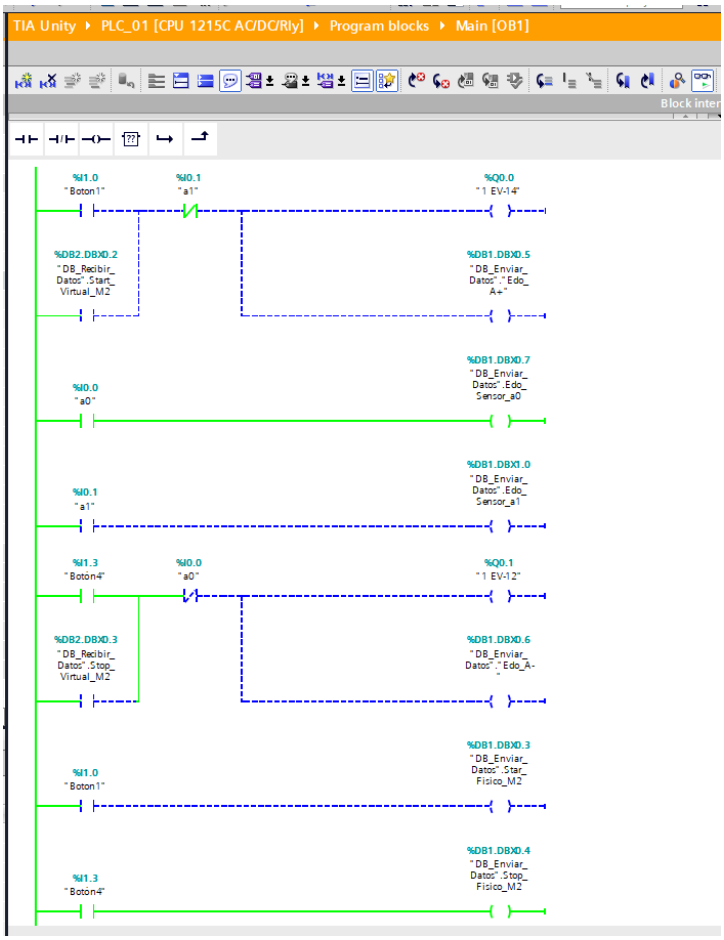


Imagen 88 Señal %I1.3 inhibida por la señal del sensor "a0"

Ya comprobada la funcionalidad de forma física, al regresar el sistema a condiciones iniciales (vástago retraído y ningún botón activo), se prueba que el sistema responda a la activación desde la escena de Unity. En la Imagen 89 se aprecia como TIA Portal recibe las señales provenientes del entorno virtual. Primero, se activa el botón pulsador virtual para la expulsión del vástago. En la Imagen 90, se activa el botón enclavado y se visualiza su efecto en el monitoreo de Unity. En la Imagen 91 se muestra la animación de expulsión y en la Imagen 92, el efecto físico del evento.

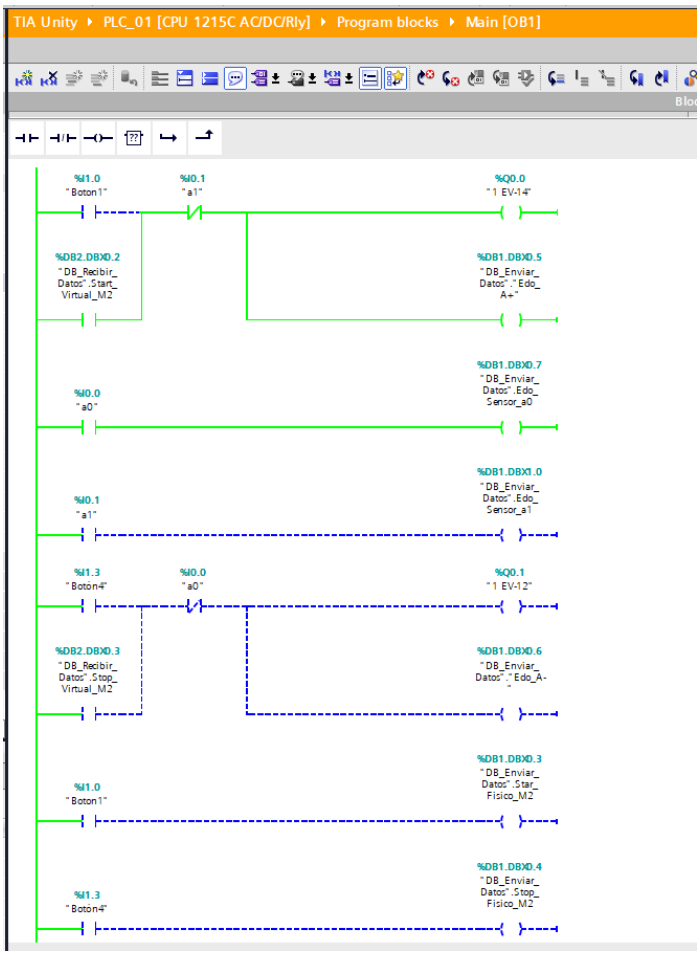


Imagen 89 Activación del sistema desde botón pulsador virtual, monitoreo en TIA Portal

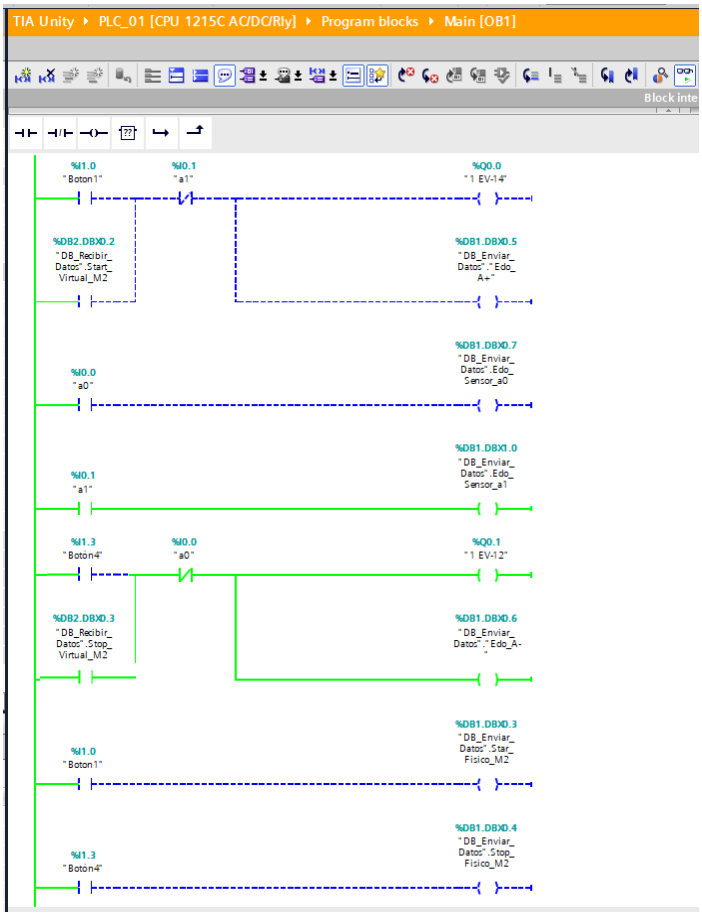


Imagen 90 Retroceso del vástago con botón enclavado virtual monitoreo en TIA Portal

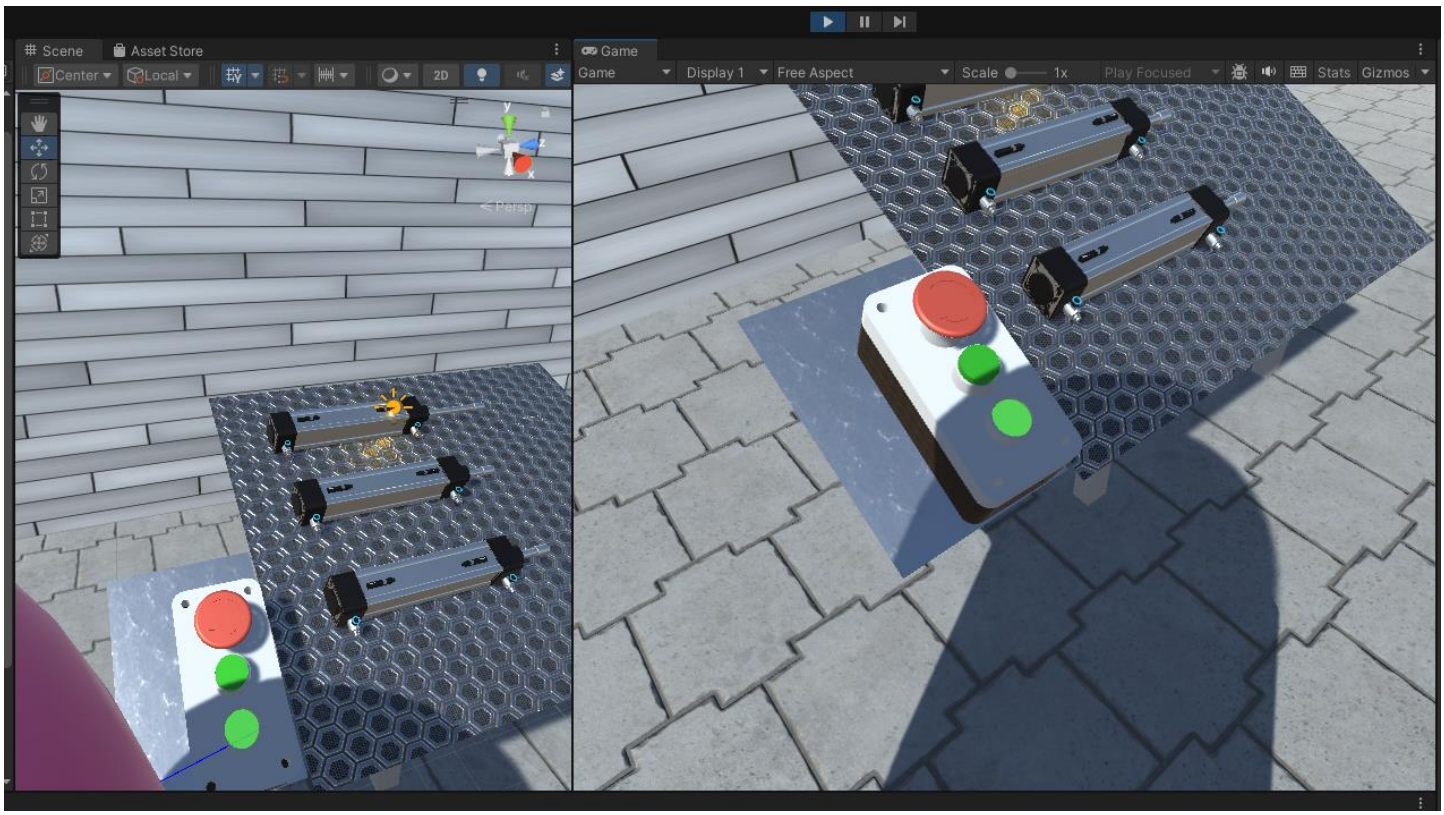


Imagen 91 Activación del sistema desde el entorno virtual, mediante un botón pulsador virtual

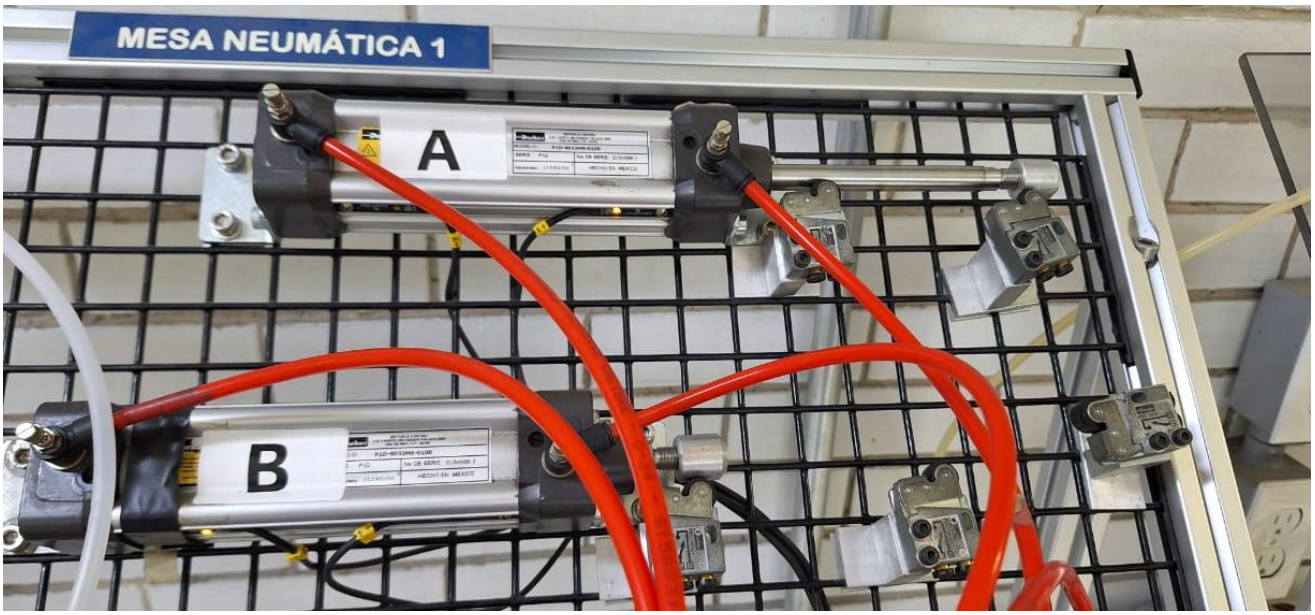


Imagen 92 Activación del vástago A desde el entorno virtual

Como se mencionó en la Imagen 90, al activar el botón enclavado virtual, el vástago retorna a su posición inicial o, dicho de otra manera, vuelve a su condición inicial; en la Imagen 93 se visualiza esta animación, así como en la Imagen 94, el resultado físico.

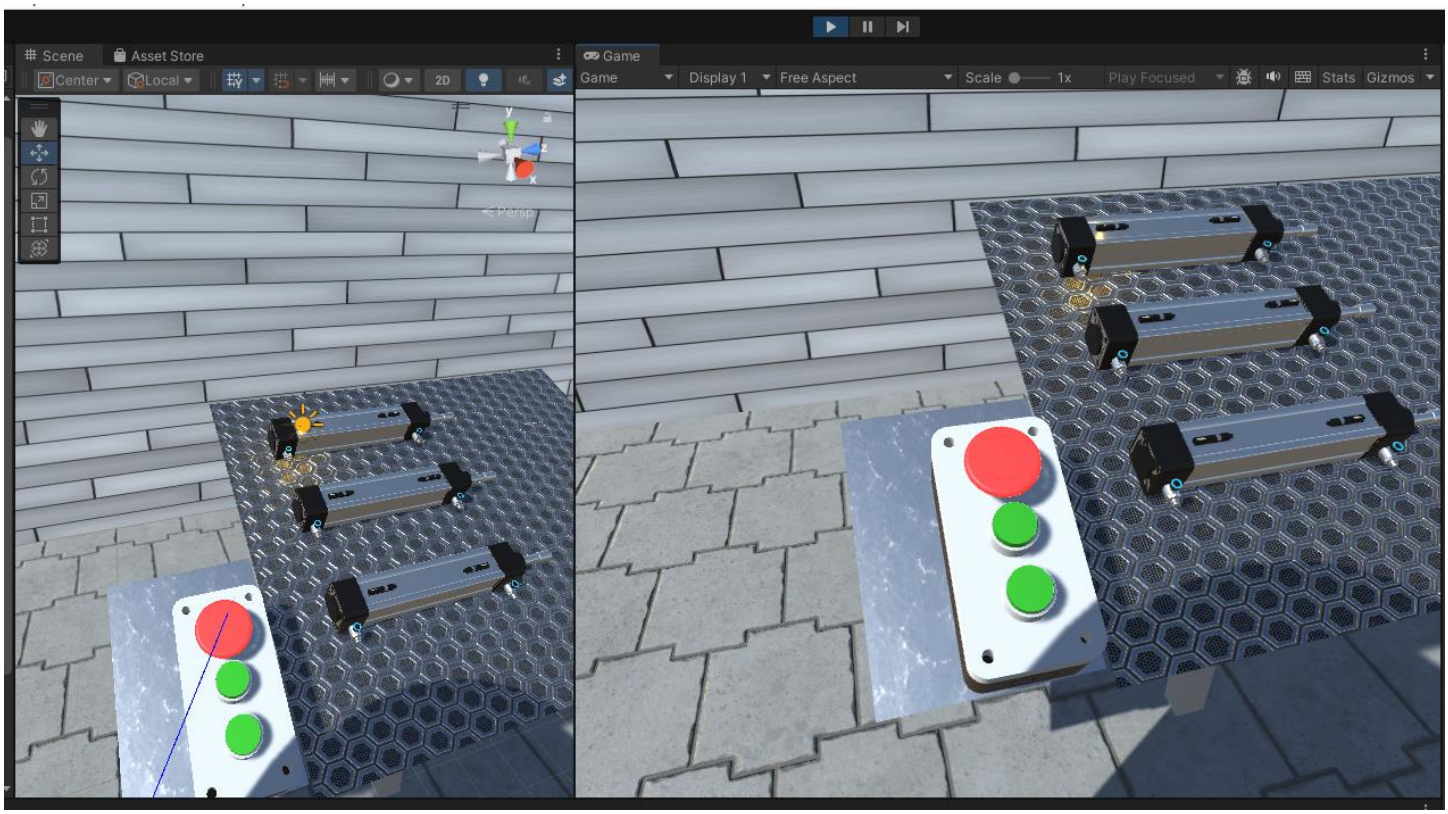


Imagen 93 Retorno de vástago a través de botón enclavado virtual

Imagen 94 Retroceso de vástago mediante la activación de botón enclavado virtual

4. Resultados

Con la evidencia del apartado anterior (*Testing de comunicación*), se puede afirmar lo siguiente:

Tanto Unity como TIA Portal responden correctamente ante la interacción de señales booleanas físicas como virtuales, incluso aunque el sistema físico se simule a través del software PLCSIM (de TIA Portal).

Según la NEMA, un PLC tiene la capacidad de ejecutar funciones lógicas, secuenciales, de registro y control de tiempos, conteo y operaciones aritméticas. En cierto modo, se podría comprobar si la conexión entre ambos sistemas permite explotar las características previamente mencionadas del PLC, pero para este trabajo, no es el objetivo. [47]

Para una exploración más detallada acerca del uso de señales booleanas, se pueden generar rutinas de control más elaboradas, por ejemplo, generar una secuencia de activación de indicadores luminosos o secuencias de trabajo con los pistones neumáticos. Ya que la escena de Unity para este proyecto está diseñada convenientemente para hacer uso de hasta tres indicadores luminosos y tres pistones de doble efecto, es posible crear dos rutinas de control secuencial a través del intercambio de datos de lógica booleana. Integrando nuevos scripts y añadiendo señales de lectura y escritura en el script de comunicación de Unity, se realizaron las pruebas "*Control de un semáforo*" y "*Secuencia Electroneumática*". En los apartados contiguos se describe más a detalle el funcionamiento de las rutinas, así como una descripción del comportamiento de la animación y la velocidad en la transmisión de los datos.

4.1 Control de un semáforo

En el apartado de *Testing de comunicación* se demuestra que la operación de encendido y apagado de un indicador luminoso es funcional, tanto de forma simulada, como de forma física. Para aumentar los eventos de encendido y apagado de varios indicadores luminosos y que sigan la secuencia de activación que se muestra en la *Tabla 2*, primero se debe generar una rutina de control que permita lograr el objetivo deseado. La *Tabla 3* muestra las direcciones que se ocupan en el programa del PLC.

Tabla 2 Diagrama de tiempos para control de un semáforo

Intervalo (s) / Indicador		3s	1s	1s	1s	1s	1s	1s	3s	10s
Verde	1									
	0									
Ámbar	1									
	0									
Rojo	1									
	0									

Tabla 3 Direccionamiento para el control de un semáforo

Etiqueta	*Identificador*	*Dirección*	*Descripción*
BSecundario2	I	%I2.1	Botón pulsador NA
BSecundario5	I	%I2.4	Botón enclavado NA
Luz_Verde	Q	%Q2.0	Indicador
Luz_Ambar	Q	%Q2.1	Indicador
Luz_Roja	Q	%Q2.2	Indicador
Start_Virtual_M1	%DB2.DBX	%DB2.DBX0.0	Start virtual
Stop_Virtual_M1	%DB2.DBX	%DB2.DBX0.1	Stop virtual
EDO_BStart_Fisico	%DB1.DBX	%DB1.DBX0.0	Estado del botón (para Unity)
EDO_BStop_Fisico	%DB1.DBX	%DB1.DBX0.1	Estado del botón (para Unity)
EDO_Luz_Verde	%DB1.DBX	%DB1.DBX0.2	Estado del indicador (para Unity)
EDO_Luz_Ambar	%DB1.DBX	%DB1.DBX0.3	Estado del indicador (para Unity)
EDO_Luz_Roja	%DB1.DBX	%DB1.DBX0.4	Estado del indicador (para Unity)

En la Imagen 95 se encuentra parte del código de control previo a operar. El sistema enciende a través de una señal física (%I2.1) o con una señal virtual (%DB2.DBX0.0”). Mientras no esté activada la señal de paro (virtual o física), la señal de memoria “On” (“%M0.0”) se enclava. Este arreglo permite que el sistema sea cíclico. A continuación, si el sistema se encuentra energizado, se activará el indicador verde hasta que un tiempo definido en la instrucción de temporización (“T_Verde.Q”), corte la energía al indicador luminoso (“%Q2.0”). Ya que los semáforos normalmente encienden y apagan el indicador verde un par de ocasiones antes de cambiar al siguiente indicador (ámbar), se coloca un peldaño en paralelo que permite el envío de la señal de encendido y apagado (“T_OFF”) a partir de la lógica de temporizadores en cascada o dependientes. La lógica que se observa en la Imagen 95 permite entender que los eventos que sucedan a la salida “Luz_verde” también afectarán a la señal EDO_Luz_Verde”. Con este arreglo se garantiza la recepción de señal exacta en el entorno virtual.

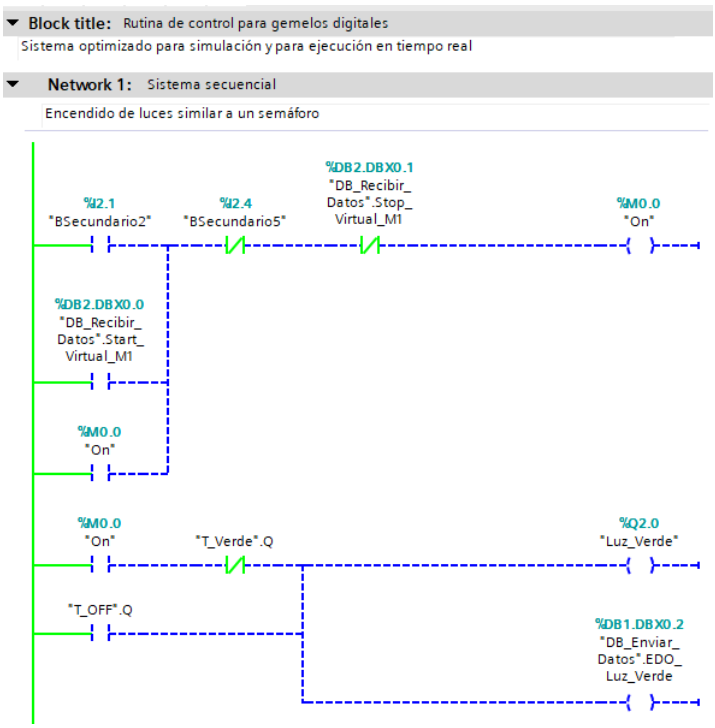


Imagen 95 Monitoreo del sistema previo a su arranque

Pasando la etapa de oscilaciones o encendido y apagado del indicador luminoso verde (señal "Oscilaciones.QU") y considerando que el sistema sigue activo (señal "On"), se activa la señal "Luz_ambar (%Q2.1)". La señal permanece hasta que un temporizador, que registra el tiempo encendido, interrumpa la corriente del indicador luminoso ámbar. ("%T_Ambar"). En la Imagen 96 se aprecia el mismo arreglo utilizado en el indicador verde para el envío de su estado al ambiente virtual; una conexión en paralelo de la bobina "EDO_Luz_Ambar".

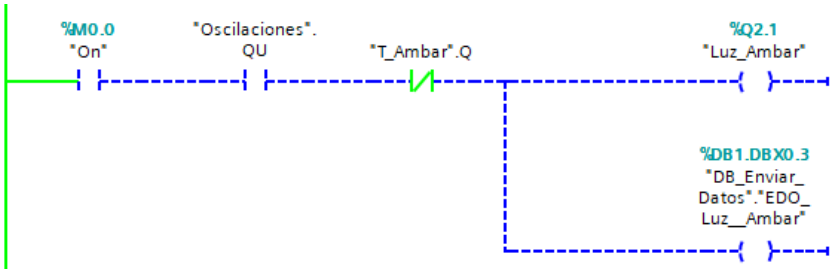


Imagen 96 Lógica de encendido para la luz ámbar

La parte final del código es la activación y desactivación del indicador luminoso rojo ("%Q2.2"). De forma similar al indicador ámbar, en la Imagen 97, se indica que la activación de la señal %Q2.2 depende de que el sistema esté energizado y que la cuenta del temporizador ámbar llegue a fin ("T_Ambar.Q"). Esta instrucción será verdadera hasta que el tiempo programado llegue a su valor objetivo ("T_Rojo.Q"). Tambien se coloca un peldaño en paralelo para que las señales lleguen a la señal "EDO_Luz_Roja".

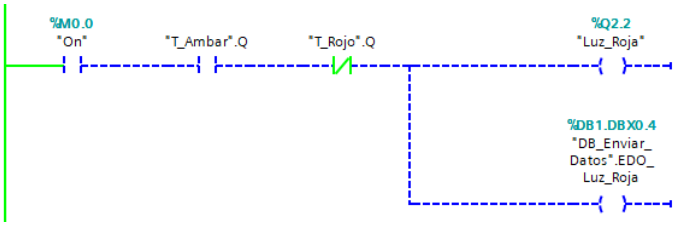


Imagen 97 Lógica de encendido para la luz roja

Ahora es necesario modificar los scripts correspondientes a la comunicación y para evitar mover las animaciones originales, se crea un script llamado “Semáforo” para las animaciones del sistema secuencial de indicadores luminosos. Para el script de comunicación; se incrementa el número de señales de lectura. El aumento es entendible ya que, ahora más señales comienzan a interactuar en el sistema. En la Imagen 98 se ve el cambio aplicado al programa.

```
// Leer señal desde el PLC (ej: DB1.DBX0.0)
bool StartVirtual = (bool)plc.Read("DB1.DBX0.0"); //Arranque semáforo
bool StopVirtual = (bool)plc.Read("DB1.DBX0.1");    //Paro semáforo

bool StartVirtualM2 = (bool)plc.Read("DB1.DBX0.3"); // Activación de sistema neumático; trabajo
bool StopVirtualM2 = (bool)plc.Read("DB1.DBX0.4");  //Paro de sistema neumático; reposo
bool EDO_L_Verde = (bool)plc.Read("DB1.DBX0.5");    // Estado de Indicador verde
bool EDO_L_Ambar = (bool)plc.Read("DB1.DBX0.6");    // Estado de Indicador verde
bool EDO_L_Roja = (bool)plc.Read("DB1.DBX0.7");    // Estado de Indicador verde
```

Imagen 98 Señales que lee Unity para la ejecución de las animaciones

Cada una de las señales leídas, se almacena en una variable que posteriormente es utilizada para ejecutar bloques de programación que realizan una animación. En la ilustración 99, se mandan a llamar a las variables “StartVirtual” y “StopVirtual”, en donde ejecutan el encendido o apagado del sistema del semáforo en un script llamado “Semáforo” (Imagen 100).

```
if (StartVirtual)
{
    EncenderIV();
}

if (StopVirtual)
{
    ApagarIV();
}
```

Imagen 99 Llamado a subrutinas

```
public void VerdeOn()
{
    Verde.SetActive(true);
}
public void AmbarOn()
{
    Ambar.SetActive(true);
}

public void RojaOn()
{
    Roja.SetActive(true);
}

public void ApagarSVerde()
{
    Verde.SetActive(false);
}

public void ApagarSAmbar()
{
    Ambar.SetActive(false);
}

public void ApagarSRojo()
{
    Roja.SetActive(false);
}
```

Imagen 100 Script semáforo, comparación del estado de los indicadores

Realizados los cambios pertinentes, TIA Portal se coloca en modo ejecución, con el monitoreo activado, y Unity también debe estar en modo ejecución. Cabe aclarar que este sistema se probó tanto de la forma física (Unity – PLC - Botonera), como de la forma simulada (Unity - PLCSIM). A continuación, se muestra el sistema Unity-PLC – Botonera (sistema físico) y posteriormente una intervención del sistema Unity – PLC – PLCSIM.

Para una mayor claridad de qué sucede en todos los entornos, se analizará la activación del código por partes. En la Imagen 101, se visualiza que el sistema se energiza en TIA Portal, de inmediato la bobina relacionada al indicador luminoso verde, enciende (físico y virtual) y el temporizador se activa. En las Imágenes 102 y 103 se observa el resultado de activar las señales *%Q2.0* (Luz_Verde) y *%DB.DBX0.2* (EDO_Luz_Verde), representando al actuador físico y al actuador virtual, respectivamente.

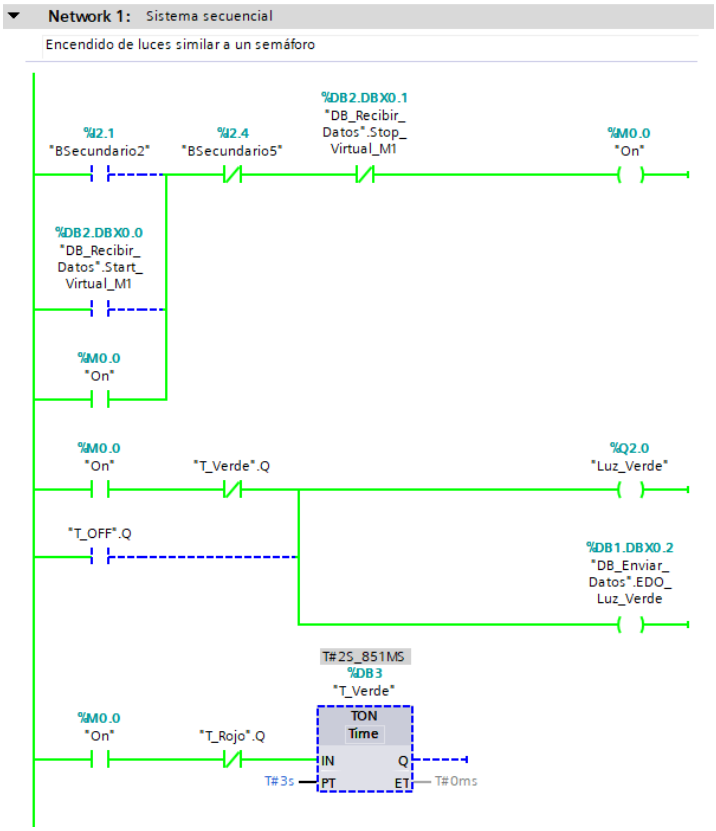


Imagen 101 Indicador luminoso verde encendido – TIA Portal

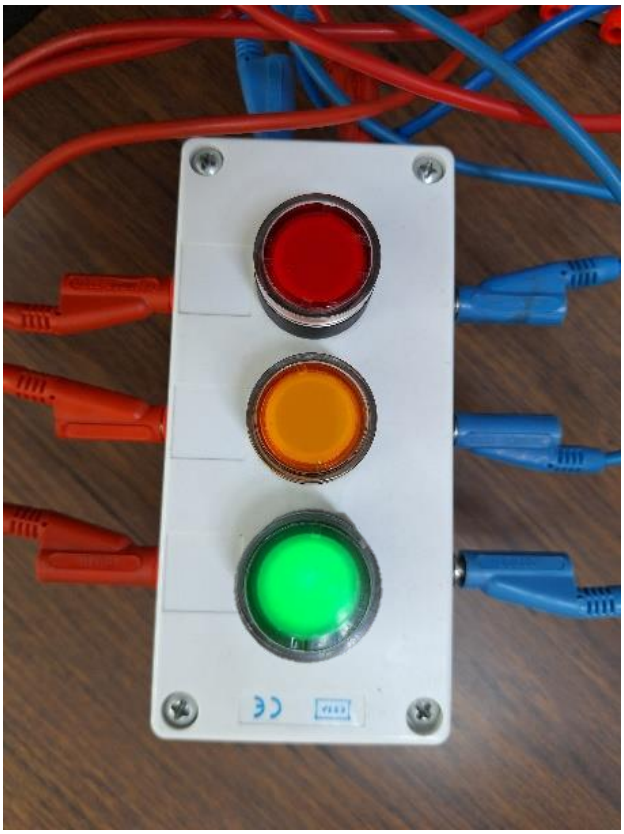


Imagen 102 Botón iluminado verde activo – físico

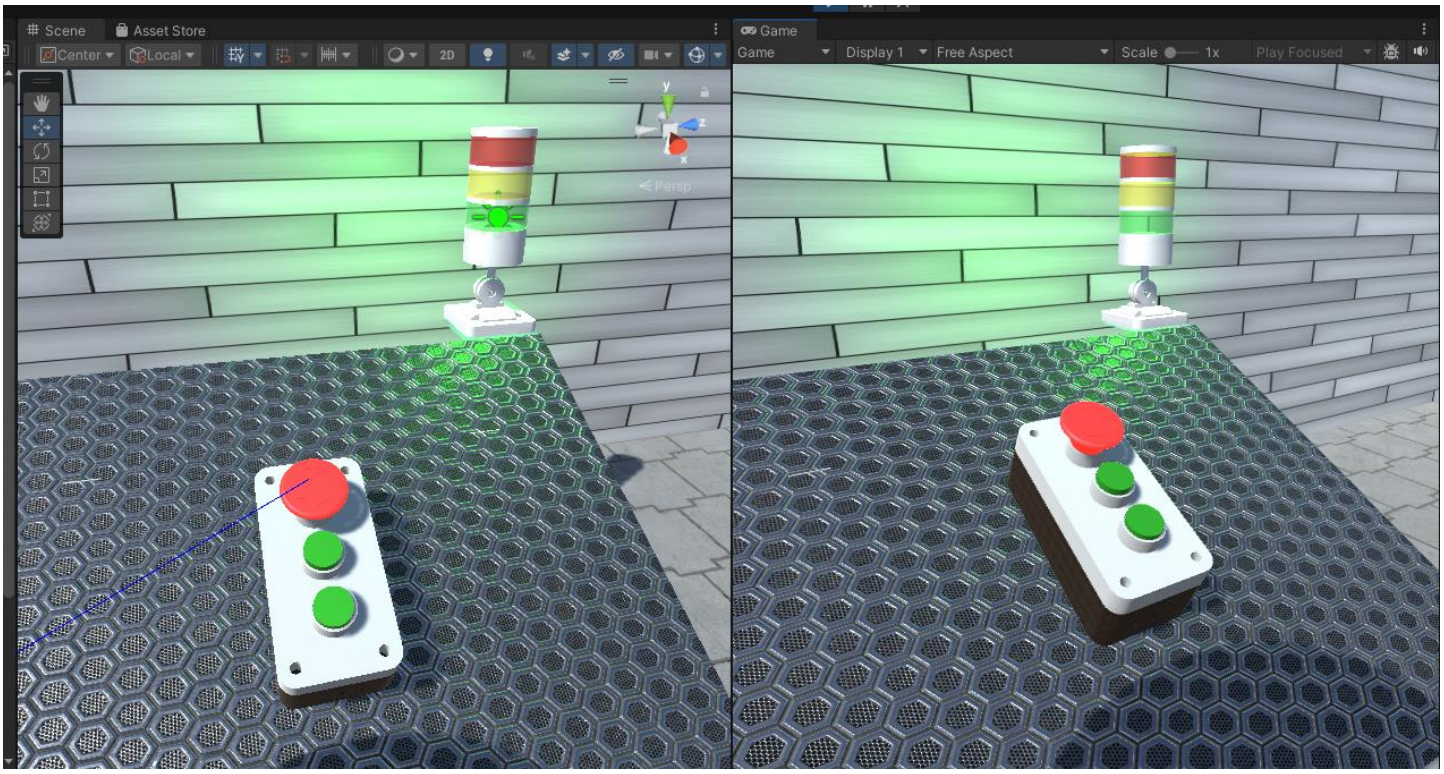


Imagen 103 Luz verde de torreta activa – virtual

Transcurrido el ciclo de ejecución (descrito en la Imagen 101), el sistema enciende el indicador luminoso ámbar (Imagen 104); el actuador permanecerá así hasta que la cuenta de su temporizador llegue a su fin. En las Imágenes 105 y 106 se observa el resultado de activar las señales *%Q2.1* (Luz_Ambar) y *%DB.DBX0.3* (EDO_Luz_Ambar), representando al actuador físico y al actuador virtual, respectivamente.

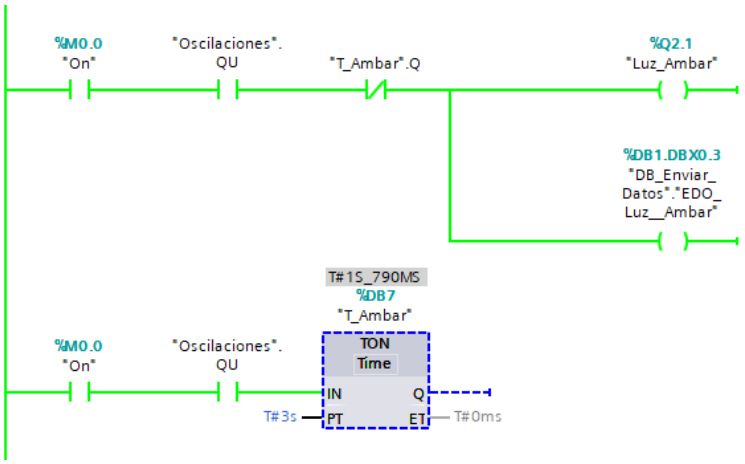


Imagen 104 Indicador luminoso ámbar encendido – TIA Portal

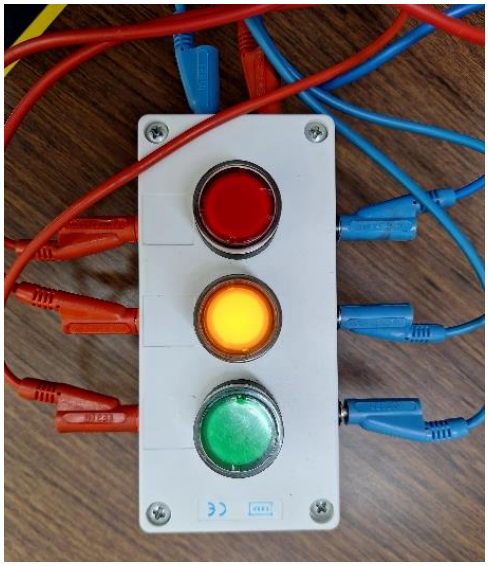


Imagen 105 Botón iluminado ámbar activo – físico

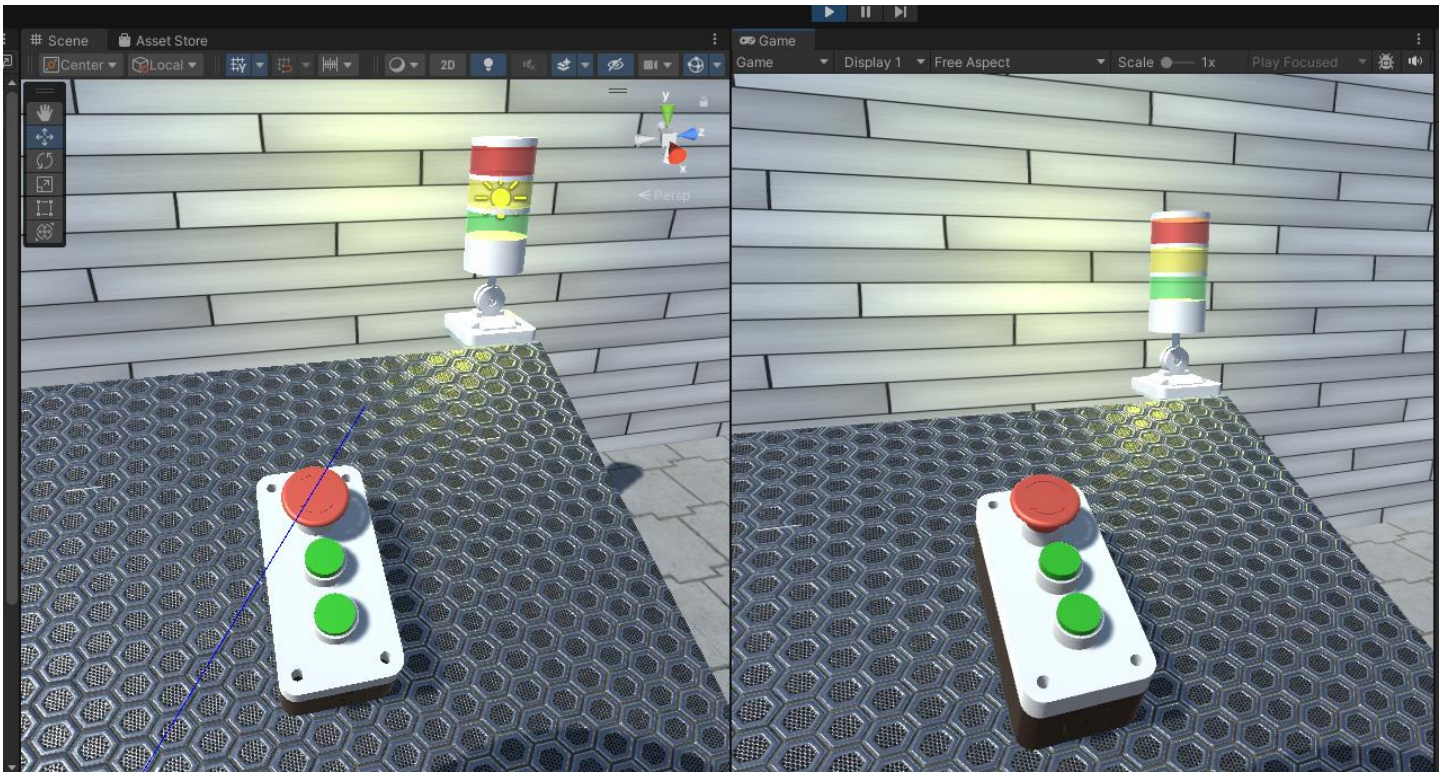


Imagen 106 Luz ámbar de torreta activa – virtual

Al término del tiempo de encendido del indicador luminoso ámbar (Imagen 104), el indicador rojo comienza con su tiempo de encendido, como se observa en el Imagen 107. En las Imágenes 108 y 109 se observa el resultado de activar las señales *%Q2.2* (Luz_Roja) y *%DB.DBX0.4* (EDO_Luz_Roja), representando al actuador físico y al actuador virtual, respectivamente.

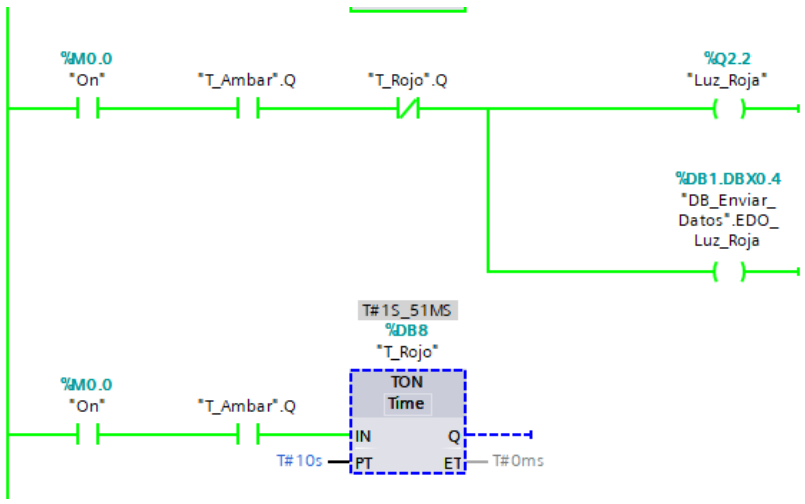


Imagen 107 Indicador luminoso rojo encendido – TIA Portal

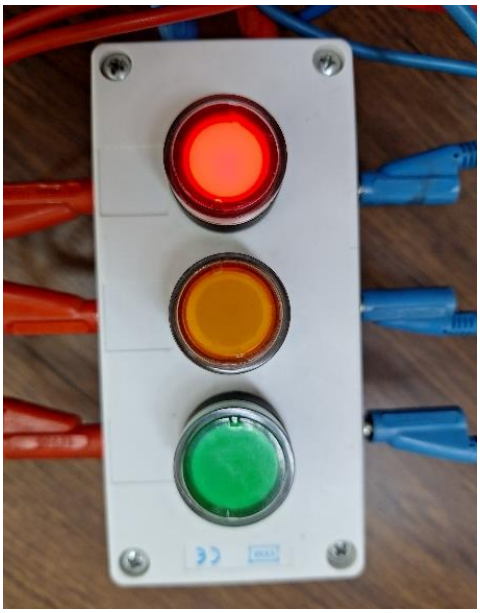


Imagen 108 Botón iluminado rojo activo – físico

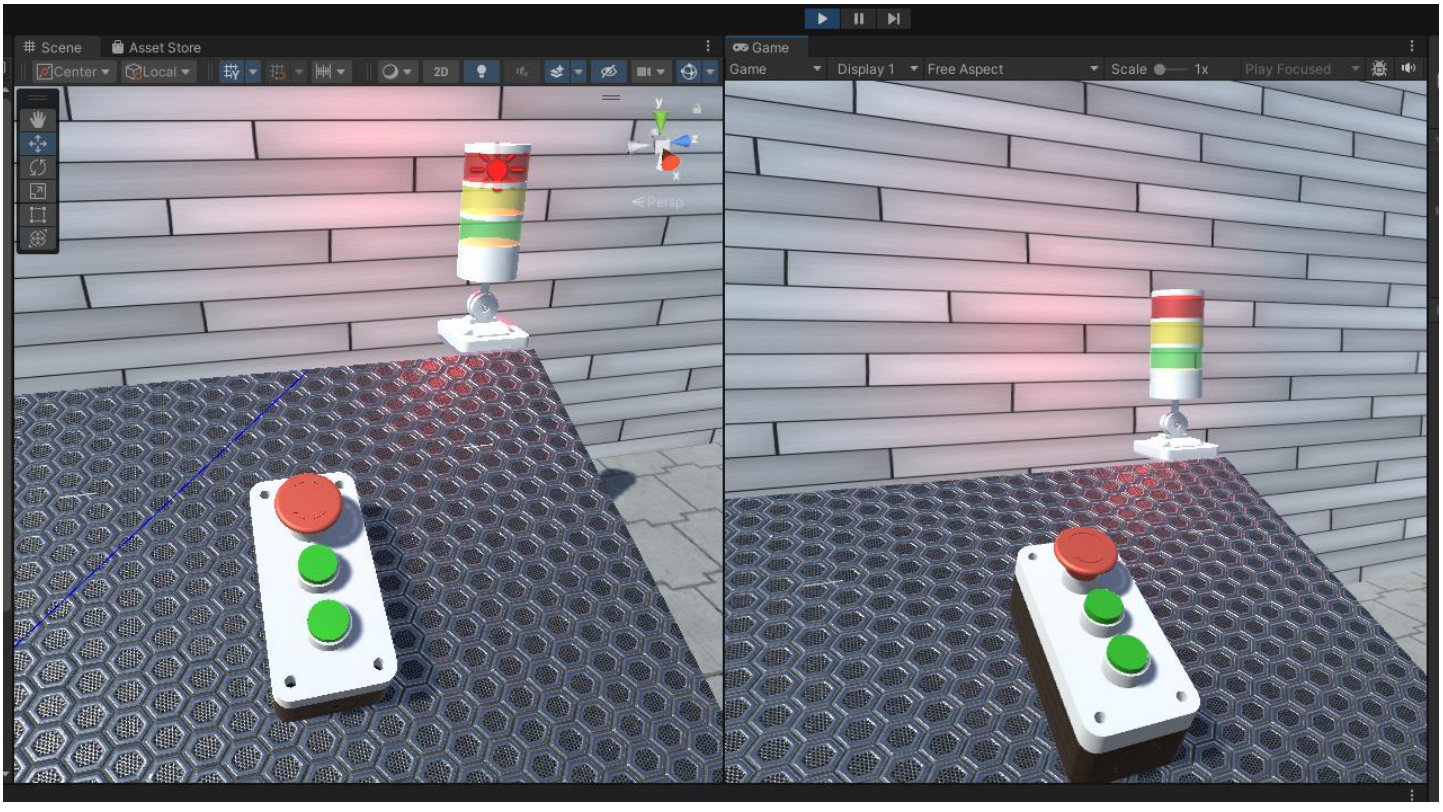


Imagen 109 Luz roja de torreta activa – virtual

El sistema continuará este proceso hasta que una señal de paro (física o virtual) interrumpa la secuencia.

Si el botón enclavado es presionado, ya sea el físico o el virtual, el sistema interrumpe su ejecución. Si el botón no se desenclava, no importará si se vuelve a presionar el botón de arranque, el sistema no trabajará; en la rutina de control se colocó un enclavamiento con prioridad a la desconexión, mejorando la seguridad del programa. En la Imagen 110 y 111, se monitorea cómo se detiene el ciclo de trabajo del sistema al accionar el botón enclavado desde PLCSIM.

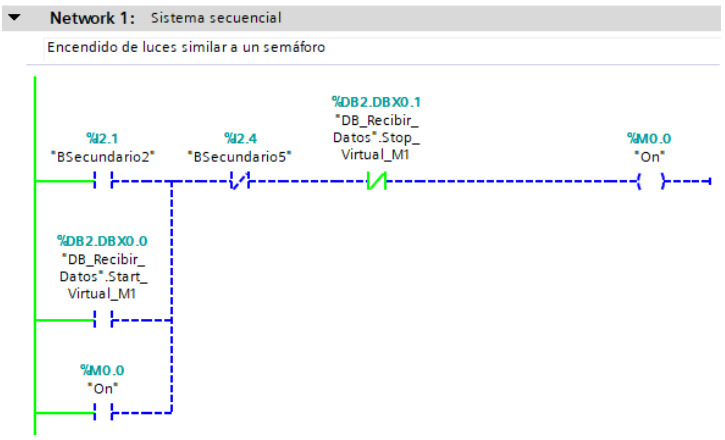


Imagen 110 Señal %I2.4 interrumpiendo ciclo de trabajo

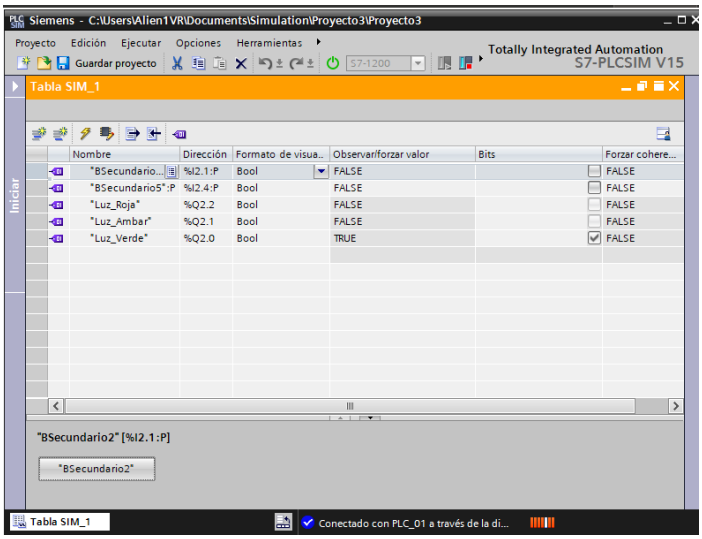


Imagen 111 Sistema monitoreado desde PLCSIM

Es importante mencionar que en Unity la tasa de adquisición de datos no es una unidad fija, sino que se puede configurar a través de los mismos scripts. La frecuencia con la que el motor gráfico procesa información de entrada y realiza la actualización o animación en la escena, se mide a través de fotogramas. La tasa de fotogramas o "framerate" se determina por la capacidad de procesamiento del hardware (CPU y GPU) y la cantidad de gráficos colocados en la escena [50]. El lector puede profundizar sobre este tema en diversas publicaciones de IBM y Unity. [48,49]

Para configurar la tasa de actualización de datos, dentro de los scripts se puede hacer uso de la función "Update"; la función *Update* permite incrementar o reducir la velocidad de adquisición de información por segundo. En la rutina *Control de un Semáforo* ocurría un error en la animación, ocasionado por la pérdida de información. Fue necesario considerar la velocidad en la trasmisión de los datos; en la rutina de control los eventos sucedían a intervalos de segundos, en Unity se muestreaban los datos, también, a una frecuencia de segundos. Como resultado, la animación era errónea, solo se activaba el indicador verde (en intervalo de tiempo fijo) y el indicador rojo; el indicador ámbar, solo en ocasiones prendía.

En la Imagen 112 se comparte el arreglo que fue más conveniente para solucionar la pérdida de datos ocasionada por una falta de velocidad en la actualización de información.

```
float timer = 0f;
float intervalo = 0.3f;
```

Imagen 112 Intervalo de actualización del script comunicación

De la Imagen 112, la variable "timer" inicializa en cero ya que, con la variable de Unity Time.deltaTime, se contabiliza el tiempo real transcurrido. En la Imagen 113, *timer* e *intervalo* se comparan; *intervalo*, al ser el tiempo que debe transcurrir antes de ejecutar la siguiente instrucción, permanece estático en un valor de 0.3 fotogramas (300 milisegundos). Ya que cualquier número es mayor a cero, se ejecutan las instrucciones que están anidadas, para este código, la adquisición de datos desde TIA Portal. Con ello, *intervalo* permite controlar la velocidad de actualización de los datos.

No olvidar lo mencionado en párrafos anteriores, la capacidad de procesamiento dependerá de la capacidad de la CPU y GPU, por lo que el utilizar los parámetros de la Imagen 112, pueden ocasionar retrasos o demoras en el procesamiento de la escena de Unity; también conocido como "lag". [51]

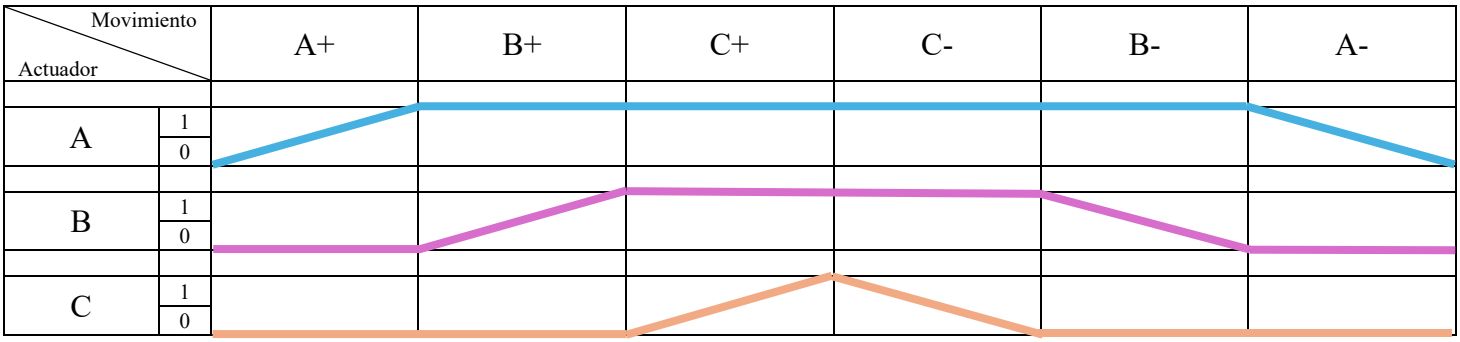
```
if (plc != null && plc.IsConnected)
{
    timer += Time.deltaTime;

    if (timer >= intervalo)
    {
        timer = 0f;

        try
        {
```

Imagen 113 Función de pedimento de datos en script comunicación

4.2 Secuencia electroneumática

Una vez dominado el movimiento de un actuador neumático, es más sencillo replicar el proceso a más elementos similares. Ahora se integra una rutina de tres actuadores neumáticos, con movimientos secuenciales y, además, cíclica; la secuencia de trabajo es: A+B+C+C-B-A-, como se muestra en la *Tabla 4*.

Tabla 4 Diagrama Espacio – Fase para secuencia A+B+C+C-B-A-



El gemelo digital permite monitorear el estado de los elementos, además, la activación o desactivación desde cualquier entorno (físico o virtual). Por tanto, la *Tabla 5* muestra las variables a considerar en la programación de TIA Portal.

Tabla 5 Variables de interés para la secuencia A+B+C+C-B-A-

Etiqueta	*Identificador*	*Dirección*	*Descripción*
a0	I	%I0.0	Sensor Reed a0
a1	I	%I0.1	Sensor Reed a1
b0	I	%I0.2	Sensor Reed b0
b1	I	%I0.3	Sensor Reed b1
c0	I	%I0.4	Sensor Reed c0
c1	I	%I0.5	Sensor Reed c1
Botón1	I	%I1.0	Botón pulsador NA
Botón2	I	%I1.1	Botón pulsador NA
Botón3	I	%I1.2	Botón selector NA
Botón4	I	%I1.3	Botón enclavado NA
1 EV-14	Q	%Q0.0	A+

Etiqueta	Identificador	Dirección	Descripción
1 EV-12	Q	%Q0.1	A-
2 EV-14	Q	%Q0.2	B+
3 EV-14	Q	%Q0.3	C+
Start_Virtual_M2	%DB2.DBX	%DB2.DBX0.2	Start virtual
Stop_Virtual_M2	%DB2.DBX	%DB2.DBX0.3	Stop virtual
EDO_A+	%DB1.DBX	%DB1.DBX1.0	Estado del actuador (para Unity)
EDO_A-	%DB1.DBX	%DB1.DBX1.1	Estado del actuador (para Unity)
EDO_B+	%DB1.DBX	%DB1.DBX1.2	Estado del actuador (para Unity)
EDO_B-	%DB1.DBX	%DB1.DBX1.3	Estado del actuador (para Unity)
EDO_C+	%DB1.DBX	%DB1.DBX1.4	Estado del actuador (para Unity)
EDO_C-	%DB1.DBX	%DB1.DBX1.5	Estado del actuador (para Unity)
a0	%DB1.DBX	%DB1.DBX1.6	Estado del sensor (para Unity)
a1	%DB1.DBX	%DB1.DBX1.7	Estado del sensor (para Unity)
b0	%DB1.DBX	%DB1.DBX2.0	Estado del sensor (para Unity)
b1	%DB1.DBX	%DB1.DBX2.1	Estado del sensor (para Unity)
c0	%DB1.DBX	%DB1.DBX2.2	Estado del sensor (para Unity)
c1	%DB1.DBX	%DB1.DBX2.3	Estado del sensor (para Unity)

Con el diagrama espacio – fase (*Tabla 4*) y las señales de trabajo (*Tabla 5*), la lógica de programación para activar los pilotajes de las electroválvulas se obtiene a través de la combinación del método cascada y las reglas del lenguaje escalera. El método cascada permite generar grupos de trabajo; cada grupo de trabajo es definido como un conjunto de movimientos que integran a la secuencia de trabajo sin repetir ningún elemento (actuador). Las reglas del lenguaje escalera establecen condiciones para la correcta ejecución del código programado en el PLC; esencialmente son tres reglas: la salida es lo último que se coloca en el peldaño, no se repiten las salidas y considerar el funcionamiento del contacto requerido, ya que no solo importa si la señal proviene de un dispositivo NA o NC. [52,53,54]

Al hacer uso de una combinación de ambos métodos, es sencillo integrar las referencias a los movimientos de retroceso para los pistones B y C ya que, la configuración de las electroválvulas con que cuentan ambos elementos, es del tipo monoestable; a diferencia del pistón A, que cuenta con una electroválvula biestable. Por ende, la activación de los pilotajes 12 para B y C se usan como memorias que se envían a Unity para hacer el reflejo del movimiento.

Para comprender el funcionamiento del sistema, primero se analiza la programación de TIA Portal. En la Imagen 114 se observa que el sistema está a la espera del botón de arranque, ya sea de forma física o de forma virtual. A continuación, una variable de memoria se autoenclava haciendo que el sistema sea cíclico. Una vez que el sistema está energizado, se activa el pilotaje 14 de la electroválvula del pistón A y también se envía la señal a Unity para ejecutar la animación de expulsión del vástago. Cuando el vástago A es detectado por el sensor *a1*, se expulsa el vástago del pistón B y de manera similar, se envía la señal a Unity. El proceso es idéntico para el vástago del pistón C, Imagen 115.

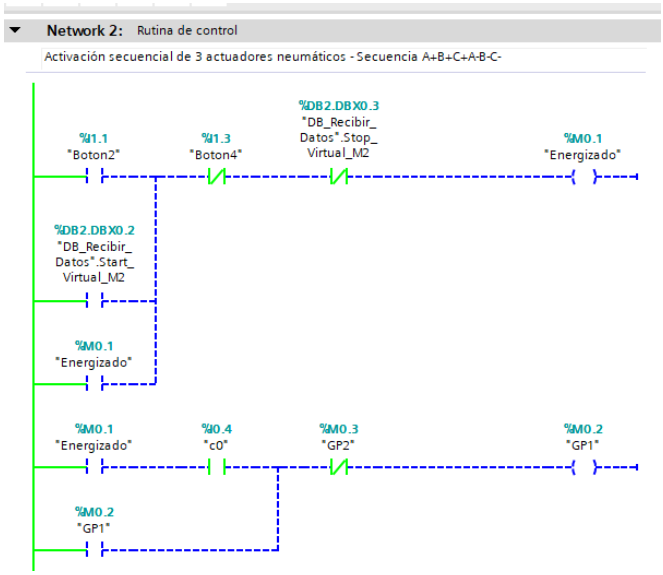


Imagen 114 Arranque de sistema electroneumático

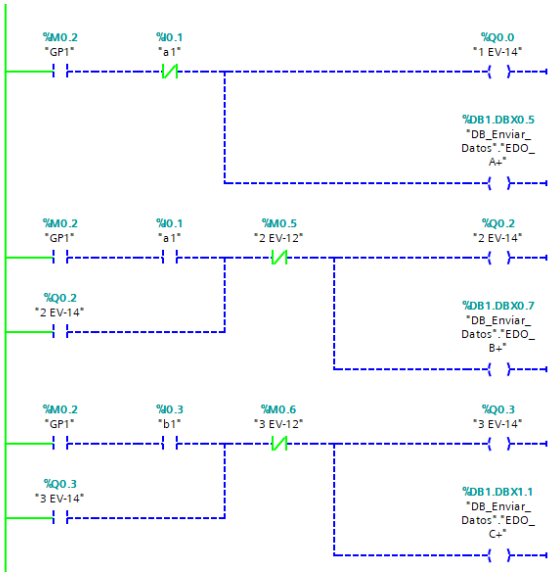


Imagen 115 Secuencia A+, B+ y C+.

Cuando los vástagos de los pistones han sido explulsados y con el análisis del diagrama de movimientos de la Tabla 4, la siguiente parte del código ejecuta el retorno de los vástagos de forma secuencial. Por lo tanto, mientras la señal *c0* no esté activa, se activa el pilotaje 12 de la electroválvula para el pistón C sin embargo, dada la configuración física de la electroválvula (monoestable), la señal se almacena en una memoria y se envía a desactivar el enclavamiento del pilotaje 14 del pistón C, como se muestra en la Imagen 115. Posteriormente, el sistema espera recibir la señal del sensor *c0*, y activa el pilotaje 12 del pistón B. Al igual que el pistón C, el retorno de B se efectua mediante una memoria. Cuando *b0* es activado, se activa el pilotaje 12 de A, *Q0.1* (electroválvula biestable), Imagen 116.

Como se mencionó con anterioridad, se envían las señales de activación y desactivación de las salidas del PLC, que corresponden a la estimulación de los pilotajes 12 y 14 de las electroválvulas. También es indispensable envíar el estado de los sensores de tipo Reed como se muestra en la Imagen 117.

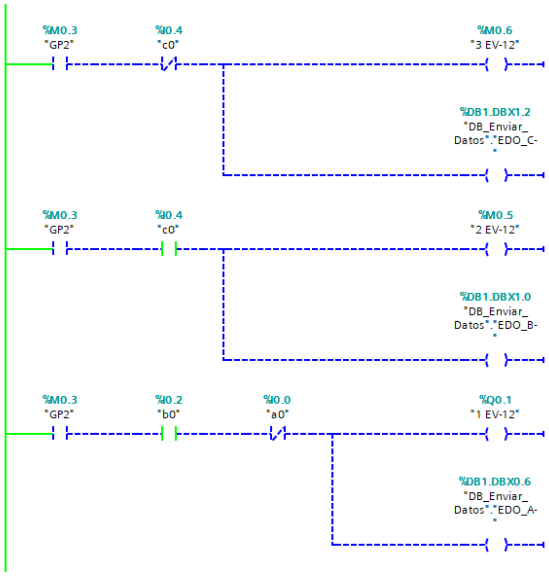


Imagen 116 Secuencia C-, B- y A-.

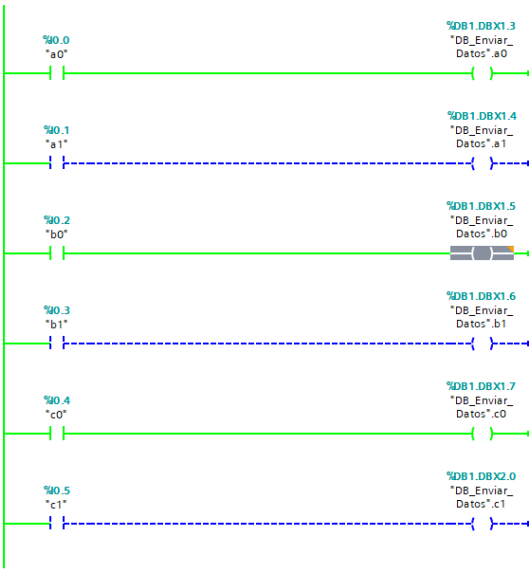


Imagen 117 Envío de señales de sensores Reed

Para la programación de los scripts de Unity se considera el formato previo (rutina de *Semáforo)* para el pedimento de datos y la activación de las animaciones. Dentro del script de *Comunicación*, se hace una solicitud de datos al controlador mediante el nodo de comunicación (NetToPLCsim o un cable Ethernet); si el valor es verdadero, se procede a la ejecución del evento correspondiente. En la Imagen 118 se observa la lectura de las variables que se envían mediante el *DB_Envío_Datos* desde TIA Portal.

```
// Leer señal desde el PLC (ej: DB1.DBX0.0)
bool StartVirtual = (bool)plc.Read("DB1.DBX0.0"); //Arranque semáforo
bool StopVirtual = (bool)plc.Read("DB1.DBX0.1");    //Paro semáforo

bool StartVirtualM2 = (bool)plc.Read("DB1.DBX0.3"); // Activación de sistema neumático; trabajo
bool StopVirtualM2 = (bool)plc.Read("DB1.DBX0.4");  //Paro de sistema neumático; reposo
bool EDO_L_Verde = (bool)plc.Read("DB1.DBX0.5");    // Estado de Indicador verde
bool EDO_L_Ambar = (bool)plc.Read("DB1.DBX0.6");    // Estado de Indicador verde
bool EDO_L_Roja = (bool)plc.Read("DB1.DBX0.7");    // Estado de Indicador verde
bool EDO_L_Apos = (bool)plc.Read("DB1.DBX1.0");    // Estado electroválvula A+
bool EDO_L_Aneg = (bool)plc.Read("DB1.DBX1.1");    // Estado electroválcula A-
bool EDO_L_Bpos = (bool)plc.Read("DB1.DBX1.2");    // Estado electroválvula B+
bool EDO_L_Bneg = (bool)plc.Read("DB1.DBX1.3");    // Estado electroválcula B-
bool EDO_L_Cpos = (bool)plc.Read("DB1.DBX1.4");    // Estado electroválvula C+
bool EDO_L_Cneg = (bool)plc.Read("DB1.DBX1.5");    // Estado electroválcula C-
bool Sensora0 = (bool)plc.Read("DB1.DBX1.6");       //Sensor de reposo
bool Sensora1 = (bool)plc.Read("DB1.DBX1.7");       //Sensor de trabajo
bool Sensorb0 = (bool)plc.Read("DB1.DBX2.0");       //Sensor de reposo
bool Sensorb1 = (bool)plc.Read("DB1.DBX2.1");       //Sensor de trabajo
bool Sensorc0 = (bool)plc.Read("DB1.DBX2.2");       //Sensor de reposo
bool Sensorc1 = (bool)plc.Read("DB1.DBX2.3");       //Sensor de trabajo
```

Imagen 118 Pedimento de datos desde Unity

El estado del dato es almacenado en una variable cuyo nombre hace referencia al elemento físico del cual se obtiene; por ejemplo, la señal *sensorb0* (Imagen 118) tiene origen en el sensor de efecto Reed ubicado en la parte trasera del pistón B físico, como se muestra en la Imagen 119. Dentro del script de comunicación, se evalúa los dos posibles casos del valor de la variable; si es verdadera o falsa. En la Imagen 120 se aprecia la sección correspondiente al análisis para las variables *Sensorb0* y *Sensorb1*.

Imagen 119 Pistón B; sensor de efecto Reed "b0" encendido

```
if (Sensorb0 == true)
{
    Sb0.SetActive(true);
}

if (Sensorb0 == false)
{
    Sb0.SetActive(false);
}

if (Sensorb1 == true)
{
    Sb1.SetActive(true);
}

if (Sensorb1 == false)
{
    Sb1.SetActive(false);
}
```

Imagen 120 Estados de las variables Sensorb0 y Sensorb1

Una vez considerado el estado de la variable, se ejecuta el código que permita la animación del evento; activar o desactivar una luz (sensor Reed) o el expulsar el vástago de un pistón. En la Imagen 121 se aprecia el código correspondiente a la animación de expulsión y retroceso del vástago del pistón B; el código se puede dividir en tres grupos.

1. Inicialización de variables y punto de referencia: se colocan las variables correspondientes a la posición del objeto a mover, con una posición inicial (A) y una final (B); identificación del avance (positivo o negativo); punto de inicio al momento de arrancar la ejecución del programa.

2. Cambio de posición: si el movimiento es positivo, se desplaza al siguiente punto con una velocidad establecida previamente; si el movimiento es negativo, regresa a la posición inicial. El movimiento se realiza en un solo eje (Z).

3. Llamada a método: activación de la secuencia de expulsión o retroceso al activar el evento desde un elemento dentro del entorno gráfico o desde una señal de TIA Portal.

```
public class PistonB : MonoBehaviour
{
    public Vector3 positionA;       // Posición inicial (A)
    public Vector3 positionB;       // Posición final (B)
    public float speed = 5f;        // Velocidad del movimiento

    public bool MovBMas = false;   // Control desde UI o input
    public bool MovBMenos = false;   // Control desde UI o input

    private Vector3 targetPosition;

    void Start()
    {
        transform.position = positionA;     // Iniciar en A
        targetPosition = positionA;
    }

    void Update()
    {
        // Lógica de movimiento
        if (MovBMas)
        {
            targetPosition = positionB;
        }
        else if (MovBMenos)
        {
            targetPosition = positionA;
        }

        // Movimiento suave hacia la posición objetivo
        transform.position = Vector3.MoveTowards(transform.position, targetPosition, speed * Time.deltaTime);
    }

    // Métodos para conectar con los botones de UI o eventos
    public void MoviPosB()
    {
        MovBMas = true;
        MovBMenos = false;
    }

    public void MovNegB()
    {
        MovBMas = false;
        MovBMenos = true;
    }
}
```

Imagen 121 Animación de extensión y retroceso de un vástago B

Al activar el sistema, se energiza el grupo 1 y comienza la ejecución de acuerdo al diagrama espacio – fase de la Tabla 4; como se muestra en la Imagen 122, se acciona el pilotaje 14 para el pistón A (A+) y tanto el equipo físico como el equipo virtual se activa, Imagen 123 e Imagen 124.

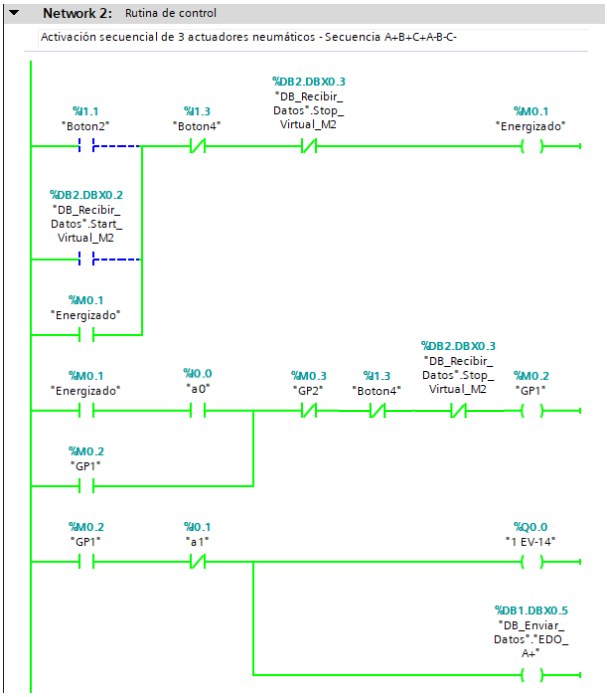


Imagen 122 Energización y movimiento A+

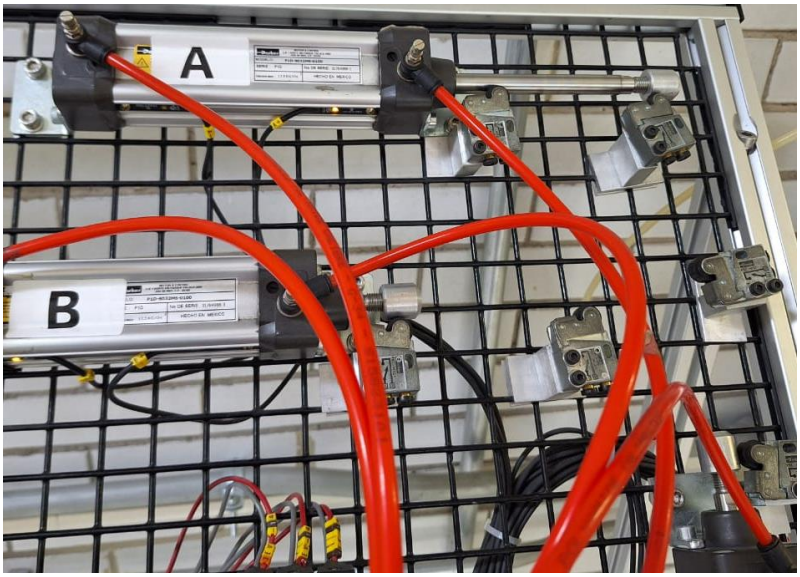


Imagen 123 Movimiento A+, mesa neumática

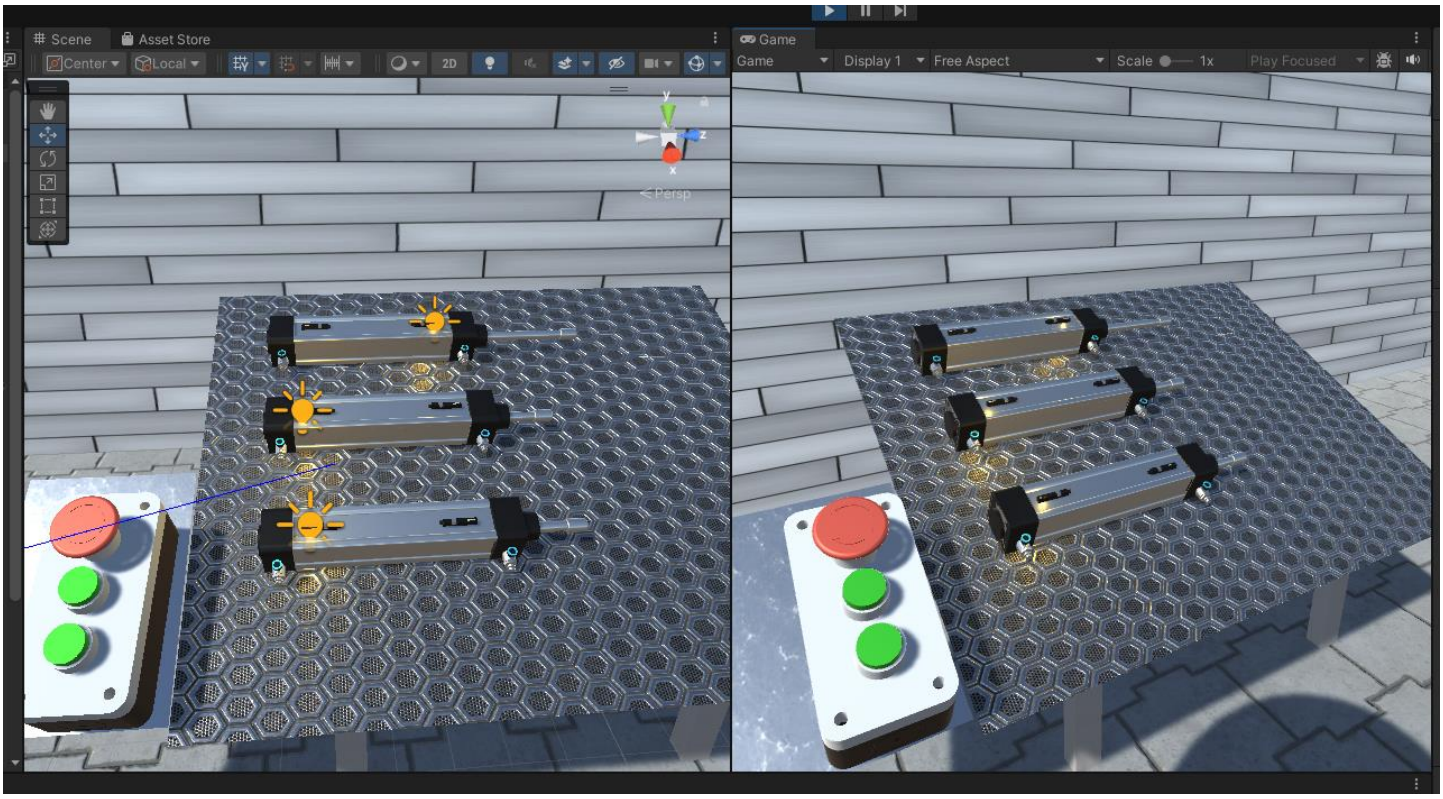


Imagen 124 Movimiento A+, mesa virtual

La rutina continúa con la etapa de activación del vástago del pistón B una vez detectada la señal *Sensora1*. Cuando la señal Sensorb1 es verdadera, el vástago del pistón C se expulsa. En la Imagen 125 se aprecia la activación de ambas electroválvulas (B+ y C+) en TIA Portal; en la mesa física, los tres vástagos se observan en la posición

extendida (Imagen 126) además de los sensores de efecto Reed encendidos. En el entorno virtual, también se aprecia que el sistema replica la ejecución junto a los sensores Reed. En la Imagen 127 la señal *Sensorc1* no se encuentra encendida debido a la velocidad en la que el sistema opera, la captura de pantalla se realizó asíncrona para el informe escrito.

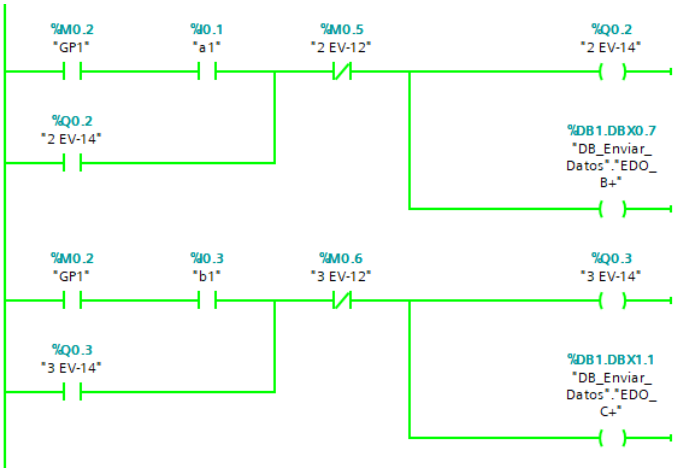


Imagen 125 Movimientos B+ y C+

Imagen 126 Movimientos A+, B+ y C+; mesa neumática

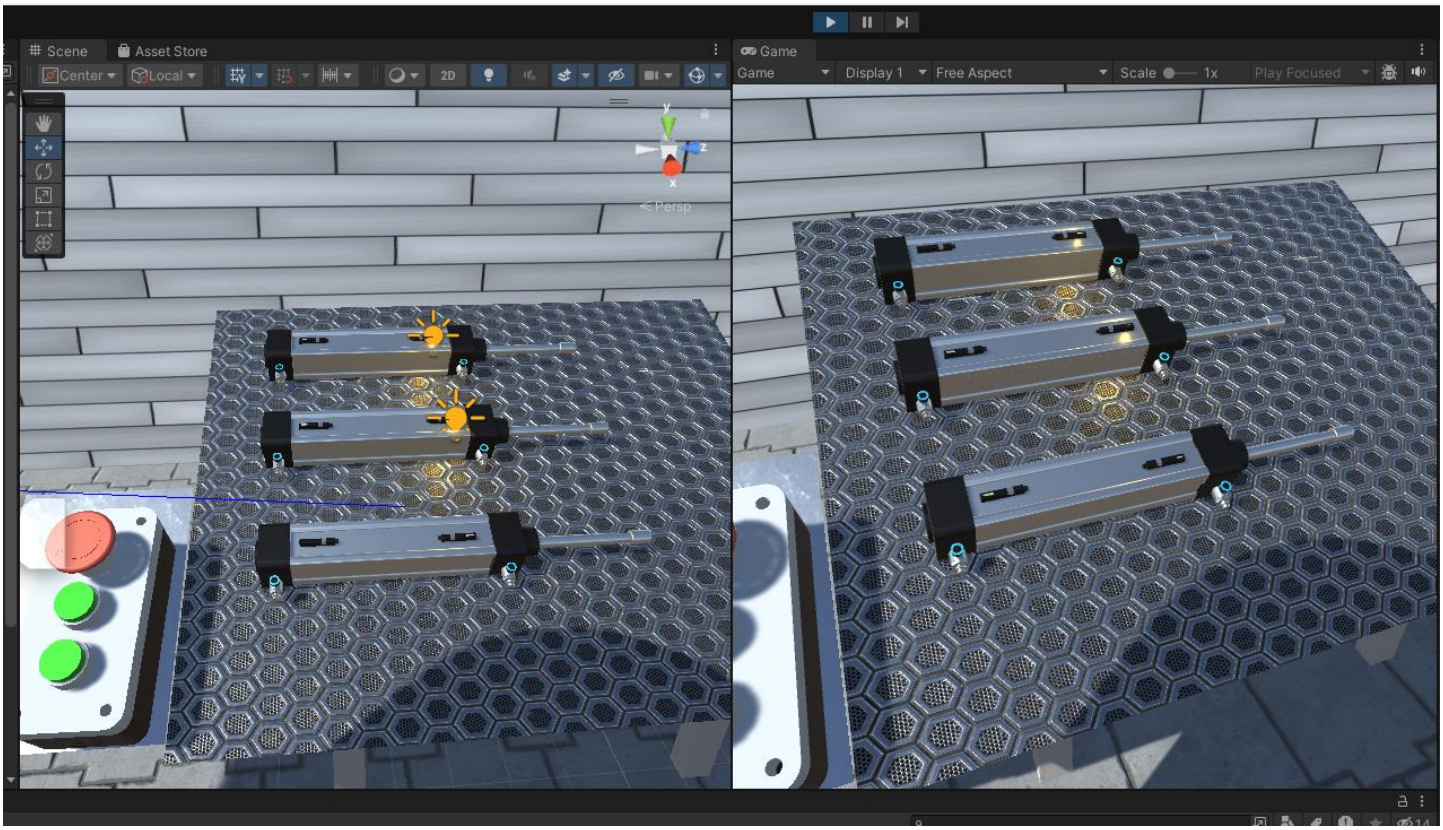


Imagen 127 Movimientos A+, B+ y C+; mesa virtual

Al ser activada la señal *Sensorc1*, el grupo 2 efectúa el movimiento C-, posteriormente B- y, al final, A-. Cuando el sistema tiene la señal Sensora0, se ha completado el ciclo y está listo para una nueva ejecución.

5. Conclusiones

5.1 Trabajo realizado

Para establecer una comunicación funcional (primer objetivo) a través del apartado *Testing*, se obtiene una respuesta en tiempo real del estado de los actuadores y, además, si se utiliza un botón en el entorno virtual, el entorno físico también responde de forma satisfactoria, por lo que cumple cualitativamente con lo esperado. Para comprender la utilidad de la comunicación entre Unity y TIA Portal (segundo objetivo), el capítulo *Resultados* permite explicar textual y visualmente cómo se generó el control para los elementos disponibles de forma virtual, siendo procesos secuenciales en donde se activa o desactiva el elemento y da paso a que el siguiente realice su funcionamiento; ya sea encenderse (para el caso de los indicadores luminosos en la torreta) o para posicionarse en su estado de trabajo (funcionamiento de los pistones).

En cuanto a la explicación detallada para que el usuario pueda hacer uso de los entornos (tercer objetivo), se ha desarrollado un manual paso a paso del procedimiento que se debe utilizar para una comunicación bidireccional entre los entornos de Unity y TIA Portal. Dentro del manual no solo se explica el procedimiento de conexión, sino que se brinda una explicación detallada del funcionamiento de los scripts desarrollados para el trabajo y los pasos a seguir para utilizar esta herramienta en operaciones totalmente virtuales como en operaciones con el equipo de hardware, sabiendo que no siempre está disponible el equipo físico.

Finalmente, para demostrar la vinculación de un proceso físico con su réplica digital (cuarto objetivo) se cumplió en los capítulos 3 y 4 (*Trabajo desarrollado; Resultados*), teniendo como consecuencia una gran cantidad de imágenes que validan la funcionalidad del sistema y su puesta en marcha con bastante evidencia documentada. Y en específico, en el apartado 4.2 (*Prueba de rutina de control*) se demuestra el potencial que puede tener este sistema.

Además de lo mencionado con anterioridad, a lo largo de este trabajo, se desarrollaron distintas pruebas e investigaciones para obtener el sistema más adecuado y, sobre todo, sencillo de utilizar, ya que esto representa un gran avance para el Laboratorio de Automatización Industrial. En muchas de las ocasiones la comunidad de alumnos presenta dificultades al momento de convivir por primera ocasión con la programación y simulación de un PLC, siendo que se vuelve un proceso demasiado abstracto. La cuestión resulta bastante recurrente y, por ende, provoca que el alumnado falle en sus evaluaciones. Se considera que esta herramienta, no solo es para el ámbito estudiantil, sino que es de gran importancia debido a que permite integrar una simulación en tiempo real de la programación que se está ejecutando, y con elementos tan sencillos como los actuadores discretos (indicadores luminosos y pistones). Por otra parte, si el programa del usuario contiene errores, el funcionamiento de las animaciones se verá afectado. En consecuencia, podría ocurrir que una luz no se encienda o que un pistón permanezca en la posición de trabajo sin corresponder al estado esperado. Esto hace que la abstracción de observar una bobina iluminada en TIA Portal se sustituya por la percepción de que “mi luz no está encendida” o “mi pistón no trabaja”.

Adicionalmente, el uso de estos gemelos digitales permite “automatizar” el proceso de pruebas mediante modelos virtuales animados que responden directamente a la programación del PLC y son capaces de interactuar con él, evitando así la necesidad de tener que realizar un proceso de pruebas completamente manual en PLCSIM, por ejemplo: teniendo que activar y desactivar, una por una, las señales correspondientes a los sensores magnéticos de final de carrera, para simular el movimiento de cada cilindro neumático.

5.2 Trabajo a futuro

Retomando el texto final del apartado anterior (trabajo realizado), el Laboratorio de Automatización Industrial actualmente presenta una demanda extrema de sus instalaciones, siendo que en los horarios hábiles casi siempre existen académicos impartiendo su cátedra, como tal, no está mal aprovechar al máximo los recursos que la universidad provee siendo que este espacio cuenta con material tan valioso de grado industrial, que muy pocas universidades se pueden dar el lujo de operar. Sin embargo, también existe la limitante de que los estudiantes solo dispongan de sus horas clase para entrar a interactuar con el equipo, y con agendas tan apretadas, se cierra la posibilidad del desarrollo de proyectos. En el caso de la asignatura de Automatización Avanzada, las escasas dos horas a la semana de laboratorio abierto, espacio designado para que los alumnos usen las instalaciones para lo que necesiten, no es tiempo suficiente para elaborar y desarrollar sus pruebas necesarias para sus proyectos. Por tanto, la implementación de un sistema de gemelos digitales totalmente virtuales permite que los alumnos puedan realizar sus trabajos desde una máquina virtual, teniendo una mayor disponibilidad de tiempo y accesibilidad.

Además, es una realidad que los equipos industriales son extremadamente caros, tanto su adquisición como su mantenimiento, por lo que el migrar a sistemas digitales logrará que los alumnos no disminuyan de manera drástica la vida útil de los equipos, siendo que, desde un entorno virtual, no pueden dañar a el equipo por errores de conexión o funcionamiento, como en diversas ocasiones sucede que dañan los elementos por mal uso (daño a en las conexiones eléctricas).

Finalmente, un aspecto que también se pone sobre la mesa es utilizar ambientes de esta índole para aumentar las capacidades de aprendizaje dentro del laboratorio, permitiendo una escalabilidad que va desde virtualizar equipos a partir de las especificaciones del fabricante sin la necesidad de adquirirlo propiamente o en su defecto, adquirir una unidad del equipo y replicar su funcionamiento de manera virtual en todas las estaciones de trabajo disponibles en el LAI.

Como dato adicional, los PLC S7-1200 también cuentan con el protocolo Modbus y Ethernet IP, todo recae en cambiar el modo de recepción de datos desde el entorno de TIA Portal, con ello, no solo se limita el trabajar un gemelo digital con equipo de esta marca, sino que se puede incrementar la capacidad de equipo industrial considerando la mejor relación de precios.

Bibliografía

[1] Lluva, M. (2021, 01 abril). Redes industriales y su importancia en la industria 4.0. Recuperado de https://www.mytra.es/blog-post/redes-industriales-su-importancia-en-la-industria-4-0-para-la-eficiencia-y-seguridad-de-la-produccion?, [Accedido en: 04-07-2025].

[2] Albarrán, C. (2024, 26 agosto). Qué es la Industria 4.0. Redes&Telecom. Recuperado de https://www.redestelecom.es/especiales/que-es-la-industria-4-0-ventajas-y-como-funciona/?utm, [Accedido en: 04-07-2025].

[3] Cabrera, M., & Tarrés, F. (s. f.). Introducción a los sistemas de comunicaciones (1.a ed.) [PDF]. Universitat Oberta de Catalunya. Recuperado de https://openaccess.uoc.edu/server/api/core/bitstreams/3fd9d375-e0b9-4d3a-b4b5-2637e3a2857d/content, [Accedido en: 04-07-2025].

[4] Zárate, J. (s. f.). Fundamentos de comunicaciones (1.a ed., Vol. 1) [PDF]. Universidat Oberta de Catalunya. Recuperado de https://openaccess.uoc.edu/server/api/core/bitstreams/c1af8b76-73be-462e-aed4-e0d926794bdd/content, [Accedido en: 04-07-2025].

[5] CONSEJO ACADÉMICO DEL ÁREA DE LAS CIENCIAS FÍSICO MATEMÁTICAS Y DE LAS INGENIERÍAS. (2015). PROYECTO DE MODIFICACIÓN DEL PLAN DE ESTUDIOS DE LA LICENCIATURA EN INGENIERÍA MECATRÓNICA. En Ingeniería Mecatrónica. UNAM. Recuperado de https://www.ingenieria.unam.mx/programas_academicos/licenciatura/Mecatronica/2016/asignaturas_mecatronica_2016.pdf, [Accedido en: 05-07-2025].

[6] CodingPLC. (2023, 14 junio). PLC Simulator Online Documentation - Learn & Master PLC Programming. Recuperado de https://plcsimulator.online/, [Accedido en: 06-07-2025].

[7] Pérez, M. (s. f.) MacroPLC Trainer: Free Online PLC Ladder Simulator, Automata Virtual gratuito. Recuperado de https://www.macroplc.com/simulador/, [Accedido en: 06-07-2025].

[8] Siemens (2025). Detalles de producto - Industry Mall - Siemens Mexico. Recuperado de https://mall.industry.siemens.com/mall/es/mx/Catalog/Product?mlfb=6ES7215-1BG40-0XB0&SiepCountryCode=MX, [Accedido en: 08-07-2025].

[9] Siemens (2025). 6ES7211-0BA23-0XB0 | Controlador lógico Siemens S7-200 tipo Digital | RS. Recuperado de https://es.rs-online.com/web/p/controladores-plcs-y-automatas/4886735, [Accedido en: 08-07-2025].

[10] NVIDIA Latinoamérica. (2023, 1 marzo). [español] Simulación con Gemelo Digital del HRSG de Siemens Energy con NVIDIA Modulus y Omniverse [Vídeo]. YouTube. Recuperado de https://www.youtube.com/watch?v=pyZaN8UWlpY, [Accedido en: 08-07-2025].

[11] Siemens (2025). Software de simulación de Simcenter. Siemens Digital Industries Software. Recuperado de https://plm.sw.siemens.com/es-ES/simcenter/, [Accedido en: 10-07-2025].

[12] Siemens (2025). El hilo digital: Reducir el riesgo de los programas de sistemas. Siemens Digital Industries Software. Recuperado de https://resources.sw.siemens.com/es-ES/white-paper-the-digital-thread/, [Accedido en: 10-07-2025].

[13] Siemens (2025). Gemelos digitales: Más rápido, más fácil y reutilizable. Siemens México. Recuperado de https://www.siemens.com/mx/es/compania/historias/gemelos-digitales-mas-rapido-mas-facil-y-reutilizable.html, [Accedido en: 27-08-2024].

[14] Sumelec. (2023, 28 febrero). El configurador de gemelos digitales dinámicos de Rockwell Automation es el Emulate3D. Sumelec. Recuperado de https://www.sumelec.es/emulate3d-el-configurador-de-gemelos-digitales-dinamicos-de-rockwell-automation/, [Accedido en: 08-07-2025].

[15] Dayra. (2024, 21 mayo). AR y VR: nueva frontera de la experiencia digital y gemelos virtuales. Foundtech. Recuperado de https://foundtech.me/integracion-ar-vr-gemelos-digitales/#:~:text=La%20AR%20permite%20superponer%20elementos,su%20comportamiento%20en%20tiempo%20real, [Accedido en: 28-08-2024]

[16] Rockwell Automation. (2024, 4 agosto). Vuforia: RA empresarial líder en el mercado. Recuperado de https://www.ptc.com/es/products/vuforia, [Accedido en: 28-08-2024]

[17] Rodríguez Islas, L. R. I. (2022). GEMELO VIRTUAL DE UN BRAZO ROBÓTICO COLABORATIVO DE 6 GRADOS DE LIBERTAD BAJO LA TÉCNICA DE REALIDAD AUMENTADA [Tesis de maestria]. POSGRADO INTERINSTITUCIONAL DE CIENCIA Y TECNOLOGÍA, [Accedido en: 29-08-2025]

[18] Rae, R. A. E.-. (s. f.). Comunicación. Diccionario panhispánico del español jurídico - Real Academia Española. Recuperado de https://dpej.rae.es/lema/comunicaci%C3%B3n, [Accedido en: 26-10-2023].

[19] Insercomp. (2022, 28 diciembre). Redes de comunicación industrial, ¿Cuáles son sus características? INSERCOMP. Recuperado de https://insercomp.cl/redes-de-comunicacion-industrial-cuales-son-sus-caracteristicas/, [Accedido en: 10-07-2025].

[20] Universal Robots A/S (2025). Redes Industriales: que son, principales tipos y su utilidad. Recuperado de https://www.universal-robots.com/mx/blog/redes-industriales-que-son-principales-tipos-y-su-utilidad/, [Accedido en: 10-07-2025].

[21] M. Grieves, Digital twin: Manufacturing excellence through virtual factory replication, Washington, DC, USA, 2014, [Accedido en 14-11-2023].

[22] E. Glaessgen and D. Stargel, "The digital twin paradigm for future NASA and U.S. Air force vehicles", Proc. 53rd AIAA/ASME/ASCE/AHS/ASC Struct. Struct. Dyn. Mater. Conf. 20th AIAA/ASME/AHS Adapt. Struct. Conf. 14th AIAA, pp. 1818, Apr. 2012, [Accedido en 14-11-2023]. Traducción propia.

[23] A. Madni, C. Madni and S. Lucero, "Leveraging digital twin technology in model-based systems engineering", Systems, vol. 7, no. 1, Jan. 2019, [Accedido en 16-11-2023]. Traducción propia.

[24] Digital Twin: Enabling Technologies, Challenges and Open Research. (2020). IEEE Journals & Magazine | IEEE Xplore. Recuperado de https://ieeexplore.ieee.org/document/9103025, [Accedido en 16-11-2023].

[25] Comunicación. (2023, 17 febrero). PROFINET: ¿Qué es y cómo funciona? Cursos Centro de Entrenamiento Internacional de PROFIBUS & PROFINET. Recuperado de https://profibus.com.ar/profinet-que-es-y-como-funciona/, [Accedido en 21-11-2023].

[26] PlantWeb University. (2002). Redes de nivel de campo. En Emerson Process Managment (Buses 101). Recuperado de https://www.emerson.com, [Accedido en: 22-11-2023].

[27] Pigan, R., Metter, M., & Siemens. (2008). Automating with PROFINET: Industrial Communication based on Industrial Ethernet: Extended Edition (Second Edition) [Google Books]. Siemens, [Accedido en 23-11-2023].

[28] Aula. (2023, 6 octubre). PROFINET: qué es y cómo funciona | Comunicaciones industriales. aula21 | Formación para la Industria. Recuperado de https://www.cursosaula21.com/profinet-que-es-y-como-funciona/#:~:text=PROFINET%20es%20una%20red%20%C2%ABconmutada,hay%20una%20conexi%C3%B3n%20full%20duplex, [Accedido en 23-11-2023].

[29] PI Norte América. (2025). Profinet: página de tecnología. Recuperado de https://us.profinet.com/tecnologia/profinet-es/#:~:text=PROFINET%20es%20un%20est%C3%A1ndar%20de,menos%20costos%20y%20mayor%20calidad, [Accedido en 05-12-2023].

[30] Siemon. (2025). Ethernet/IP - protocolo de red en niveles para aplicaciones. Default. Recuperado de https://www.siemon.com/es/home/Company/media-center/white-papers/03-10-13-ethernet-ip, [Accedido en 14-12-2023].

[31] INCIBE. (2019, 28 febrero). Protocolo EtherNet/IP: analizando sus comunicaciones y medidas de seguridad. INCIBE-CERT. Recuperado de https://www.incibe.es/incibe-cert/blog/protocolo-ethernetip-analizando-sus-comunicaciones-y-medidas-seguridad, [Accedido en 12-07-2025].

[32] Aula. (2023b, octubre 6). Qué es el protocolo Ethernet Industrial, cómo funciona y para qué sirve. aula21 | Formación para la Industria. Recuperado de https://www.cursosaula21.com/que-es-ethernet-industrial/, [Accedido en 14-12-2023].

[33] Rheonics. (2022, 27 abril). EtherNet/IP: The standard protocol for communication in industrial networks. Rheonics: Viscometer And Density Meter. Recuperado de https://es.rheonics.com/ethernet-ip-the-standard-protocol-for-communication-in-industrial-networks/, [Accedido en 15-12-2023].

[34] Martínez, J. (2020). Sistema de control y monitoreo bajo los protocolos Ethernet y Modbus RTU en el control de sistemas de cintas transportadoras para línea embotelladora de bebidas. Revista de Investigación Académica, 5(2), 45-58. Recuperado de https://www.researchgate.net/publication/339956581_Sistema_de_control_y_monitoreo_bajo_los_protocolos_ethernet_y_modbus_rtu_en_el_control_de_sistemas_de_cintas_transportadoras_para_linea_embotelladora_de_bebidas, [Accedido en 11-01-2024].

[35] Danfoss. (2016). Guia Rapida VLT Micro Drive FC51. Recuperado de http://www.ramonrusso.com.ar/documentos/Guia_Rapida_VLT_Micro_Drive_FC51.pdf, [Accedido en 16-01-2024].

[36] Emerson. (2014, 15 octubre). Introducción a Modbus. NI. Recuperado de https://www.ni.com/es/shop/labview/introduction-to-modbus-using-labview.html, [Accedido en 16-01-2024].

[37] MasterD. (2019, 8 noviembre). Qué es Unity y para qué sirve. MasterD. Recuperado de https://www.masterd.es/blog/que-es-unity-3d-tutorial, [Accedido en 08-02-2024].

[38] Unity Technologies. (2025). Get Started with Unity. Recuperado de https://unity.com/learn/get-started, [Accedido en: 21-08-2023].

[39] Abad, J. (2024, 28 octubre). Qué es un script - Definición, significado y para qué sirve. Arimetrics. Recuperado de https://www.arimetrics.com/glosario-digital/script [Accedido en 24-10-2024].

[40] Microsoft. (2025, 21 enero). Visual Studio: IDE y Editor de código para desarrolladores de software y Teams. Visual Studio Microsoft. Recuperado de Visual Studio. Recuperado de https://visualstudio.microsoft.com/es/ [Accedido en 24-10-2024].

[41] Deland-Han. (2024, 19 febrero). Biblioteca de vínculos dinámicos (DLL) - Windows Client. Microsoft Learn. Recuperado de https://learn.microsoft.com/es-es/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library **.** [Accedido en 01-11-2024].

[42] S7NetPlus. (2023). GitHub - S7NetPlus/s7netplus: S7.NET+ -- A .NET library to connect to Siemens Step7 devices. GitHub. Recuperado de https://github.com/S7NetPlus/s7netplus**.** [Accedido en 01-11-2024].

[43] PLC Focus – Automation solution. (s. f.). Recuperado de https://plcfocus.net/**.** [Accedido en 01-11-2024].

[44] OpenAI. (2024). Bibliotecas de vínculos dinámicos y su interacción en Unity. ChatGPT. Recuperado de. https://www.openai.com/chatgpt. [Accedido en 28-10-2024].

[45] Technologies, U. (2025). Unity - Manual: Managed plug-ins. Unity Documentation. Recuperado de https://docs.unity3d.com/6000.1/Documentation/Manual/plug-ins-managed.html. [Accedido en 28-11-2024].

[46] SourceForge. (2025). NetToPLCsim - Network extension for Plcsim. NetToPLCsim. Recuperado de https://nettoplcsim.sourceforge.net/. [Accedido en 05-05-2025].

[47] Farnell An Avnet Company. (2025). PLC design in Automation and Industrial Control Applications.. Farnell. Recuperado de https://es.farnell.com/industrial-application-plc-design#:~:text=%C2%BFQu%C3%A9%20es%20un%20controlador%20l%C3%B3gico,control%20de%20procesos%20y%20m%C3%A1quinas. [Accedido en 19-05-2025].

[48] Flinders, M., Susnjara, S., & Smalley, I. (2025, junio 26). ¿Qué es una GPU? Ibm.com. Recuperado de https://www.ibm.com/mx-es/think/topics/gpu. [Accedido en 22-07-2025].

[49] Technologies, U. (2025). Unity - Manual: Fixed updates. Unity Documentation. Recuperado de https://docs.unity3d.com/6000.1/Documentation/Manual/fixed-updates.html#:~:text=See%20in%20Glossary%2C%20a%20fixed,approximately%20128%20frames%20per%20second. [Accedido en 22-07-2025].

[50] Technologies, U. (2025). Unity - Manual: administrador del tiempo y framerate. Unity Documentation. Recuperado de https://docs.unity3d.com/es/530/Manual/TimeFrameManagement.html#:~:text=Unity%20tiene%20una%20propiedad%20Time,timeScale. [Accedido en 22-07-2025].

[51] BenQ. (2023, 15 septiembre). ¿Qué es el lag y cómo se puede evitar?. BenQ. Recuperado de https://www.benq.com/es-mx/centro-de-conocimiento/conocimiento/que-es-el-lag-y-como-se-puede-evitar.html. [Accedido en 22-07-2025].

[52] Alberto, S. P. (s. f.). Diseño de circuitos neumáticos: Método de cascada. SlideShare. Recuperado de https://es.slideshare.net/slideshow/004-neumtica-mtodo-de-cascada/240753875. [Accedido en 25-07-2025].

[53] Sl, K. A. (2025, 24 abril). ¿Qué es y cómo funciona el LENGUAJE LADDER? | Programación PLC. Konetia Automatizacion Industrial. Recuperado de https://www.konetia-automatizacion.com/que-es-el-lenguaje-ladder/. [Accedido en 25-07-2025].

[54] Cruz, R. (2025). Práctica 7: Diagrama de escaleras para la inversión de giro. – Control electromecánico. (s. f.). Recuperado de https://masam.cuautitlan.unam.mx/dycme/ce/practica-7/. [Accedido en 01-08-2025].

Imágenes de referencia

- Imagen 1a: Mesa neumática virtual, Estación 1; VRChat. [Imagen] (2024). De autoría propia. [Accedido en: 23-04-2024].
- Imagen 1b: Mesa neumática; Laboratorio de Automatización Industrial. [Imagen] (2025). De autoría propia. [Accedido en: 10-11-2025].
- Imagen 2a: Mesa neumática virtual, Estación 1; VRChat. [Imagen] (2024). De autoría propia. [Accedido en: 23-04-2024].
- Imagen 2b: Mesa neumática; Laboratorio de Automatización Industrial. [Imagen] (2024). De autoría propia. [Accedido en: 10-11-2025].
- Imagen 3a: Desarrollo de práctica 5, Cilindro de simple efecto; VRChat. [Imagen] (2024). De autoría propia. [Accedido en: 23-04-2024].
- Imagen 3b: Desarrollo de práctica 5, Cilindro de simple efecto; Laboratorio de Automatización Industrial. [Imagen] (2024). De autoría propia. [Accedido en: 10-11-2025].
- Imagen 4: Elementos disponibles en el entorno de trabajo (Parte 1). [Imagen] (2024). De autoría propia. [Accedido en: 23-04-2024].
- Imagen 5: Elementos disponibles en el entorno de trabajo (Parte 2). [Imagen] (2024). De autoría propia. [Accedido en: 23-04-2024].
- Imagen 6: Elementos disponibles en el entorno de trabajo (Parte 3). [Imagen] (2024). De autoría propia. [Accedido en: 23-04-2024].
- Imagen 7: Visualización de funcionamiento de un pistón de simple efecto en el laboratorio virtual. [Imagen] (2024). De autoría propia. [Accedido en: 23-04-2024].
- Imagen 8: Visualización de funcionamiento de un pistón de simple efecto físico. [Imagen] (2025). De autoría propia. [Accedido en: 08-07-2025].
- Imagen 9: Simulación con Gemelo Digital del HRSG de Siemens Energy con NVIDIA Modulus y Omniverse. [Imagen] (2025). Recuperada de https://www.youtube.com/watch?v=pyZaN8UWlpY, [Accedido en: 08-07-2025].
- Imagen 10: Gemelo digital (Digital Twin) por Siemens. [Imagen] (2024). Recuperado de https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.technologyrecord.com%2Farticle%2Fsiemens-and-bridg-partner-to-develop-digital-twin-technologies&psig=AOvVaw1oTgocETKAP4rZyiHmy8BK&ust=1753418497963000&source=images&cd=vfe&opi=89978449&ved=0CBUQjRxqFwoTCMDTh5jX1I4DFQAAAAAdAAAAABAL, [Accedido en: 27-08-2024].
- Imagen 11: 10 Simulación en Emulate3D. [Imagen] (2025). Recuperada de https://www.sumelec.es/emulate3d-el-configurador-de-gemelos-digitales-dinamicos-de-rockwell-automation/, [Accedido en: 08-07-2025].
- Imagen 12: Realidad aumentada de Vuforia. [Imagen] (2024). Recuperado de https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.iac.com.co%2Fvuforia-chalk-y-para-que-sirve%2F&psig=AOvVaw1PwP2_PM00zfLIcH6p7rHE&ust=1725333512789000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCOC18bOmo4gDFQAAAAAdAAAAABAh, [Accedido en: 28-08-2024].
- Imagen 13: Visualización en tiempo real de un motor. [Imagen] (2024). Recuperado de https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.mtwmag.com%2Fptc-vuforia-augmented-reality-ar-designtech-system%2F&psig=AOvVaw1PwP2_PM00zfLIcH6p7rHE&ust=1725333512789000&source=images&c

d=vfe&opi=89978449&ved=0CBQQjRxqFwoTCOC18bOmo4gDFQAAAAAdAAAAABAp, [Accedido en: 28-08-2024].
- Imagen 14: Menú para seleccionar la posición del robot y establecer rutina. [Imagen] (2024). Recuperado de Rodríguez Islas, L. R. I. (2022). GEMELO VIRTUAL DE UN BRAZO ROBÓTICO COLABORATIVO DE 6 GRADOS DE LIBERTAD BAJO LA TÉCNICA DE REALIDAD AUMENTADA [Tesis de maestría]. POSGRADO INTERINSTITUCIONAL DE CIENCIA Y TECNOLOGÍA, [Accedido en: 29-08-24]
- Imagen 15: Profinet as modular conception. Automating with PROFINET. [Imagen] (2023). Recuperado de Google Books. https://books.google.com.ec/books?id=iOXHMpMyMBgC&printsec=frontcover#v=onepage&q&f=false, [Accedido en: 22-11-2023].
- Imagen 16: Arquitecturas PROFINET. [Imagen] (2023). Recuperado de https://profibus.com.ar/wp-content/uploads/2023/02/image-7.png, [Accedido en: 23-11-2023].
- Imagen 17: Diagrama básico de PROFINET. [Imagen] (2023). Recuperado de https://www.cursosaula21.com/wp-content/uploads/2020/02/Diagrama-b%C3%A1sico-de-PROFINET.png, [Accedido en 05-12-2023].
- Imagen 18: Diagrama básico de Ethernet Industrial. [Imagen] (2023). Recuperado de https://www.cursosaula21.com/wp-content/uploads/2019/07/Diagrama-Ethernet-industrial.png, [Accedido en 15-12-2023].
- Imagen 19: Relación Solicitud-Respuesta de Maestro-Seguidor en los dispositivos Modbus. [Imagen] (2023). Recuperado de https://ni.scene7.com/is/image/ni/hjsgemis720378571073416452 7?scl=1, [Accedido en 11-01-2024].
- Imagen 20: Protocolo Modbus. [Imagen] (2024). Recuperado de https://2.bp.blogspot.com/-pHxBDDcHz_I/WyfhiphH0NI/AAAAAAAAHuQ/MYdTUVCddfA4DsNvWcd_KoN_ObsTa6PcQCEwYBhgL/s400/modbus.png, [Accedido en 15-01-2024].
- Imagen 21: Cuphead, elaborado con el motor gráfico Unity. [Imagen] (2024). Recuperado de https://drunkers.com.mx/wp-content/uploads/2020/06/Cuphead-coop.jpg, [Accedido en 13-02-2024].
- Imagen 22: Diagrama de flujo de las etapas de trabajo en el proyecto. [Imagen] (2025). De autoría propia. [Accedido en: 12-07-2025].
- Imagen 23: Encendido de indicador verde en Unity. [Imagen] (2025). De autoría propia. [Accedido en: 10-03-2025].
- Imagen 24: Apagado del indicador visual en Unity. [Imagen] (2025). De autoría propia. [Accedido en: 10-03-2025].
- Imagen 25.: Pistón virtual. [Imagen] (2025). De autoría propia. [Accedido en: 14-03-2025].
- Imagen 26.: Torreta virtual. [Imagen] (2025). De autoría propia. [Accedido en: 14-03-2025].
- Imagen 27.: Objetos tridimensionales requeridos en el proyecto. [Imagen] (2025). De autoría propia. [Accedido en: 14-03-2025].
- Imagen 28: Mesa neumática del LAI. [Imagen] (2024). De autoría propia. [Accedido en: 10-11-2024].
- Imagen 29: Mesa 1 virtual; botonera y torreta industrial. [Imagen] (2025). De autoría propia. [Accedido en: 12-03-2025].
- Imagen 30: Mesa 2 virtual: pistones, sensores Reed y botonera. [Imagen] (2025). De autoría propia. [Accedido en: 12-03-2025].
- Imagen 31: Archivo ".dll" obtenido de la página web "PLCFOCUS" [28]. [Imagen] (2024). De autoría propia. [Accedido en: 21-10-2024].
- Imagen 32: Ruta a la ubicación del proyecto de UNITY. [Imagen] (2024). De autoría propia. [Accedido en: 21-10-2024].
- Imagen 33: Carpeta Assets. [Imagen] (2024). De autoría propia. [Accedido en: 23-10-2024].

Anexo

Manual de conexión Unity – TIA Portal para un Gemelo Digital

Manual de conexión Unity - TIA Portal para un Gemelo Digital

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME Proyecto PE109325

Elaborado por:
Pedro Luis Galindo Roblero

Índice

Para el uso de este manual, se recomienda que el usuario cuente con nociones que abarquen el manejo de los paquetes de software Unity y TIA Portal.

I. Objetivos de aprendizaje

Objetivo general: Generar una comunicación entre Unity y TIA Portal para el intercambio de datos en tiempo real entre un sistema físico y uno virtual.

Objetivos específicos:

- Desarrollar un proyecto nuevo en Unity que permita agregar y configurar objetos tridimensionales representativos del sistema físico.
- Implementar eventos y animaciones en los objetos tridimensionales mediante el uso de scripts que simulen el comportamiento dinámico del sistema.
- Integrar una biblioteca de vínculos dinámicos (DLL) en el proyecto de Unity para habilitar la comunicación con el entorno de control.
- Crear un proyecto nuevo en TIA Portal V15 utilizando un controlador S7-1200 como herramienta de procesamiento y control.
- Configurar bloques de memoria en TIA Portal para el envío y recepción de datos desde y hacia Unity.
- Establecer una red local para intercambiar datos entre Unity y TIA Portal a través de la herramienta “NetToPLCsim”.
- Diseñar y programar una rutina de control en TIA Portal que permita operar un actuador físico y vincularlo con su modelo virtual.

II. Nuevo proyecto en UNITY

Es sumamente importante verificar la versión de Unity en la que se trabaja, ya que este manual garantiza el funcionamiento completo del proyecto si se siguen todos los pasos descritos, en especial los relacionados con la integración de la biblioteca de vínculos dinámicos (DLL). Dicha integración puede variar según la versión de Unity utilizada; por ello, si se emplea una versión distinta (ya sea anterior o posterior), no se asegura que el procedimiento sea el mismo, siendo necesario investigar el procedimiento correspondiente para dicha versión.

Figura 1 Unity Hub -> New Project.

Se ingresa al Unity Hub (Figura 1) a continuación, una ventana emergente aparecerá; si se ha trabajado anteriormente con Unity, aquí aparecerán todos los proyectos previos.

Elegir la opción “New project”.

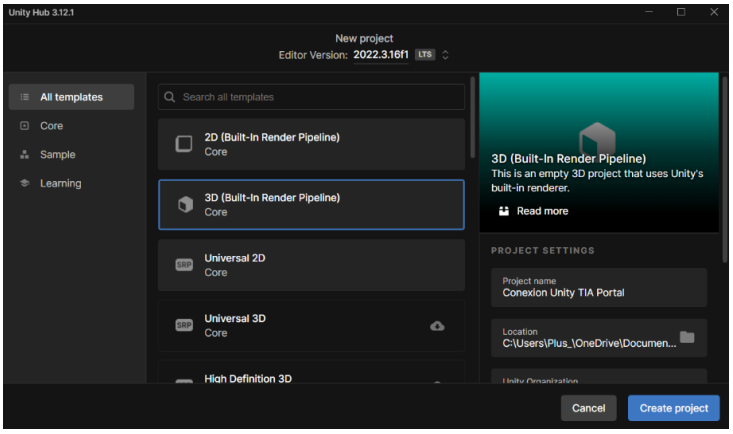


Figura 2 Versión de trabajo: 2022.3.16f1 -> 3D (Built-In Render Pipeline) -> Create Project.

En la ventana de “New project” (Figura 2), se puede seleccionar qué tipo de proyecto se creará en la aplicación. También se pueden modificar los parámetros de nombre y ubicación del proyecto. Se recomienda crear una carpeta en donde se contendrán todos los archivos a generar, desde los archivos tridimensionales hasta el proyecto de control (más adelante se llegará a ese punto).

El nuevo proyecto se nombrará “*Conexión TIA Portal*”.

Una vez realizados los cambios pertinentes, dar click en “Create project”. Ahora, Unity se autoconfigurará para integrar todos los archivos necesarios (herramientas y bibliotecas) para este nuevo proyecto, Figura 3.

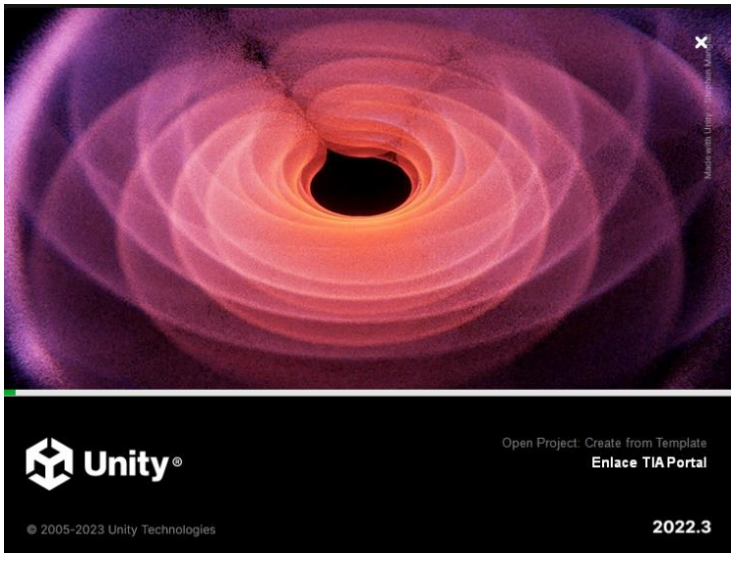


Figura 3 Pantalla de carga de Unity.

Una vez que Unity terminó de generar el nuevo programa, una nueva ventana aparecerá y en ella se puede comenzar a diseñar el entorno de trabajo.

El software presenta dos versiones de trabajo: la primera versión es local (Figura 4), todas las modificaciones que se hagan sobre el proyecto se conservarán en el equipo de trabajo (en el espacio designado); la segunda versión (Figura 5), carga los datos contenidos en la nube. Es conveniente la versión de la nube para proyectos colaborativos (no más de tres integrantes en la versión gratuita) y para el trabajo en dos dispositivos.

Ambas versiones contienen las pestañas de *herramientas*, *jerarquía*, *inspector*, *escenario* y *proyecto*, que resultan más que suficientes para el desarrollo del proyecto. La elección del espacio de trabajo dependerá de cada usuario; en este manual se muestra únicamente una manera de organizarlo según los requerimientos del proyecto.

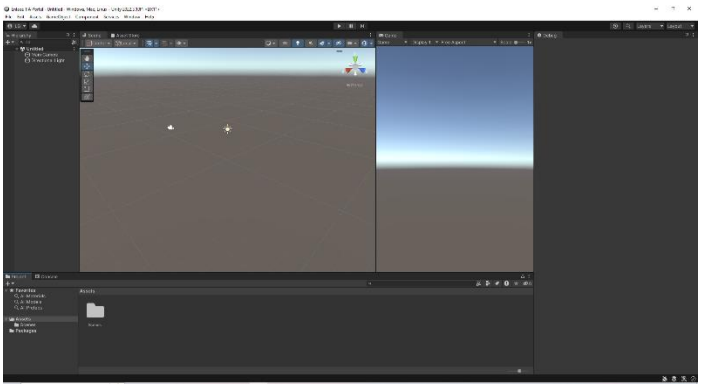


Figura 4 Vista principal de proyecto, versión local

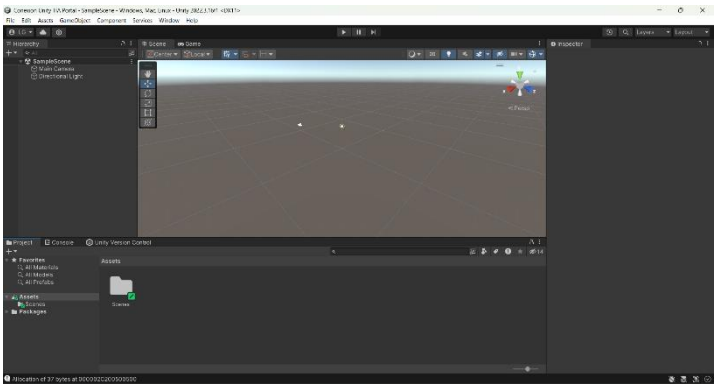


Figura 5 Vista principal de proyecto, versión para la nube (la pestaña Unity Version Control esta activada)

Creación de un escenario

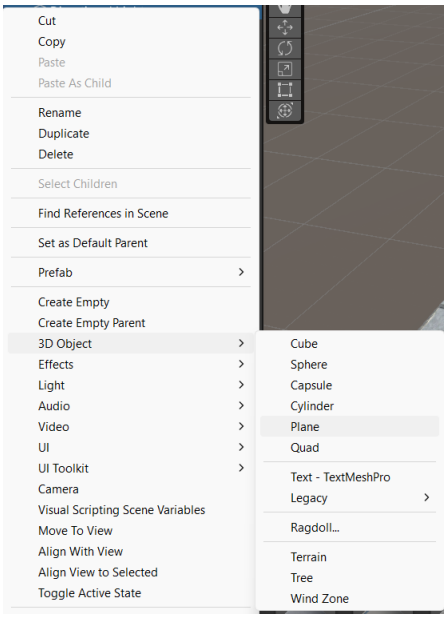


Figura 4 Integrar un plano en el scene

Para Unity la pantalla principal es “Scene” (escenario); aquí se concentrarán los objetos tridimensionales y aquellos elementos que sean necesarios. Al ser colocados los objetos gráficos, se les puede añadir el evento o eventos que se requiera para simular su comportamiento cinemático. No hay que dejar de lado que parte de los objetos no contendrán programación, ya que su función es la ambientación del entorno.

En este manual se iniciará colocando un plano, con ello se tendrá una superficie que funcione como piso y referencia para los demás elementos. En la pestaña de “Hierarchy” (jerarquía), dar click derecho y seleccionar la opción “3D Object”; seleccionar “Plane” (Figura 6).

En la escena se visualiza un rectángulo enmarcado por un cuadro naranja. En el inspector (parte derecha), se aprecia información acerca de sus propiedades: tipo de malla, material, entre otros. Es recomendable cambiar el nombre para que el objeto sea identificable: por ejemplo, “piso” (click derecho sobre el objeto en la jerarquía, *Rename*). Finalmente, escalar el objeto a conveniencia y posicionarlo en las coordenadas de origen (0,0,0) (Figura 7).

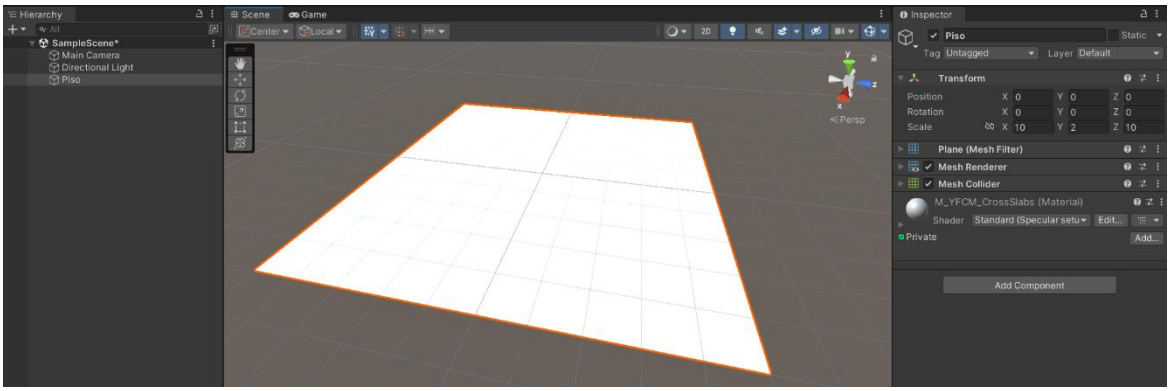


Figura 5 Renombrar a "Piso"; Escalar, X:10; Y:2; Z:10

Texturas

Para agregar una apariencia de objeto tridimensional y esta no requiere ser sofisticada o realista, se puede utilizar las herramientas de Unity destinadas a este ámbito (*Materials*). Sin embargo, si es indispensable colocar patrones o mallas muy específicas es más conveniente importarlas desde internet. Existen diversos sitios web para estos fines, incluso, Unity provee algunas de estas texturas de forma gratuita al utilizar el "*Unity AssetStore*" (Figura 8). En el manual, también se utilizaron texturas de la página web "*POLIIGON*". [1]

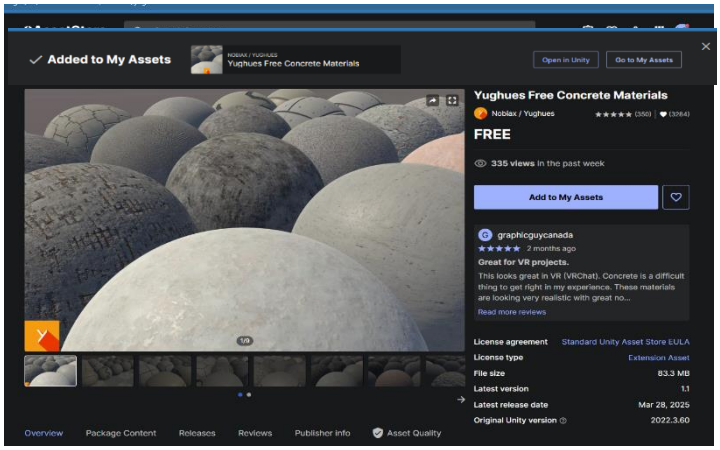


Figura 6 Unity AssetStore

Para descargar una textura de Unity AssetStore, (ya seleccionado el material o paquete de materiales), dar click en "Add to My Assets"; aceptar términos y condiciones. Una ventana emergente aparece, notificando que el archivo se agregó a la carpeta de "assets" personales en la cuenta de Unity en la nube.

El siguiente paso es agregar en el proyecto a este nuevo material desde la cuenta en la nube. Desde el apartado "My Assets" (Unity AssetStore), dar click en el apartado "Open in Unity"; una ventana emergente pedirá permiso para abrir el archivo desde la aplicación: dar click en "Abrir Unity Editor" (Figura 9).

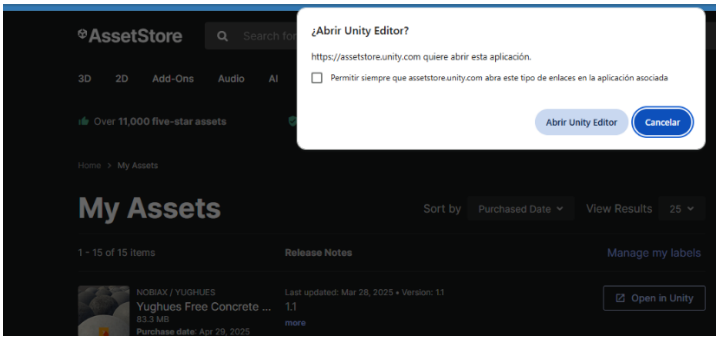


Figura 7 Integración de un paquete de texturas al proyecto de Unity

En el editor, se abrirá el "Package Manager" y aparecerá el archivo que acabamos de cargar a la cuenta en la nube (sino está seleccionado, buscar el nombre del archivo); dar click en descargar, así como se muestra en la Figura 10.

En ocasiones, no se selecciona por default la casilla de “Packages: My assets”, por lo que es necesario colocarse en ese apartado. Si previamente existen más paquetes en la nube, estarán disponibles para descargarse, por lo que es importante identificar correctamente el paquete a descargar. Si se selecciona un paquete por error, aumentará el tamaño del proyecto de forma innecesaria, resultando en una mayor demanda de recursos computacionales para compilar el proyecto.

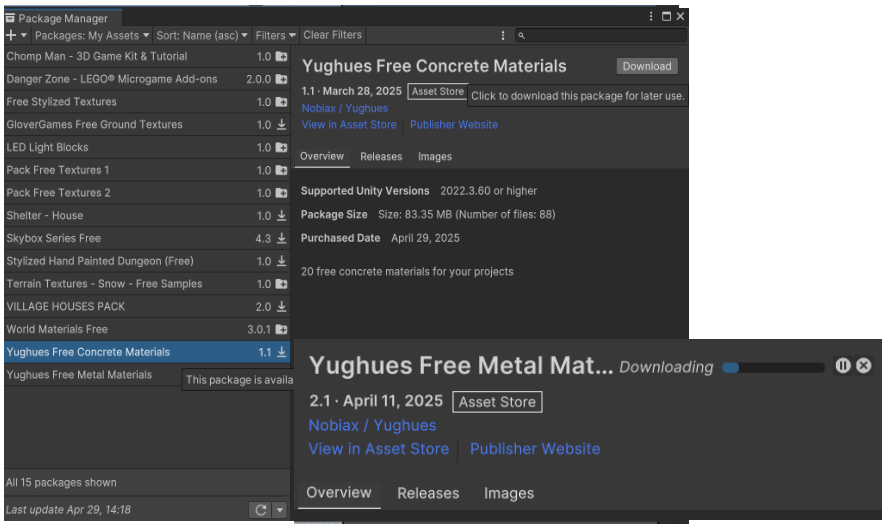


Figura 8 Package Manage -> Packages: My Assets -> Yughues Free Concrete/ Metal Materials

Ya descargado el paquete, dar click en “Import” (Figura 11); tardará unos minutos en agregar los archivos al proyecto. Concluido el anterior proceso, aparecerá una ventana emergente que permite seleccionar los archivos que se pueden añadir a la escena. En ocasiones solo es necesario descargar algunos archivos, sin embargo, en esta ocasión se descargarán todos (Figura 14).

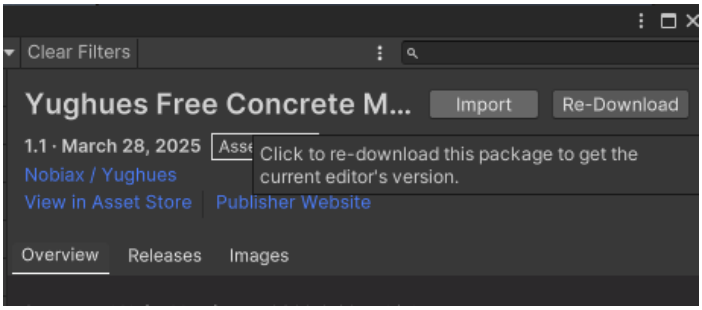


Figura 9 Importar archivos a proyecto

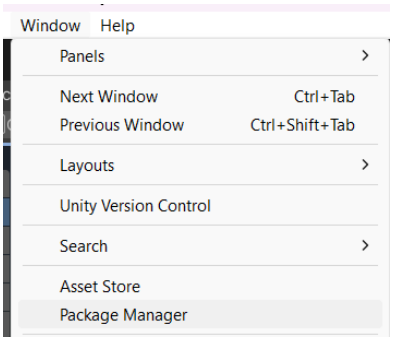


Figura 10 En la barra de herramientas -> Window -> Package Manager

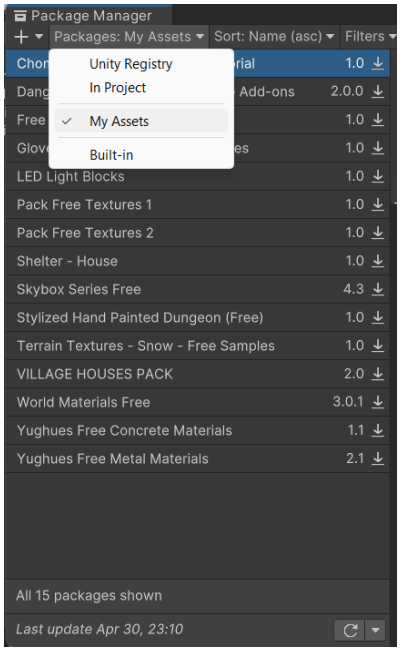


Figura 13 Descarga de un paquete de texturas Package Manager -> Packages: -> My Assets

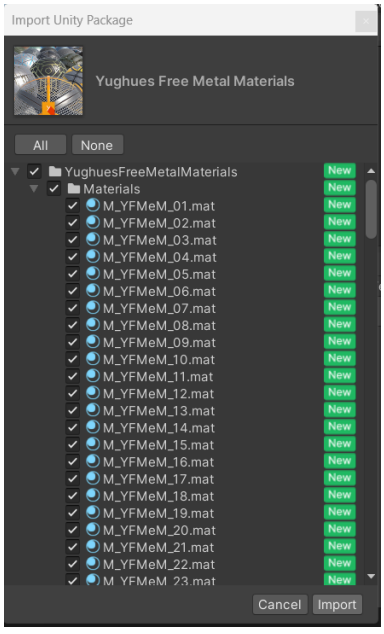


Figura 14 Importar materiales de la Unity AssetStore al proyecto

En ocasiones, se suele descargar múltiples paquetes de texturas o simplemente, no se tiene abierto el proyecto, por lo que el procedimiento descrito con anterioridad no es conveniente, sin embargo, existe otra manera de integrar los archivos, directamente abriendo el Package Manager desde el proyecto. Para ello, dar click en la pestaña de “Window”, seleccionar Package Manager, Figura 12. Por default, el Package Manager (primera pestaña), tiene seleccionada la opción “In Project”, cambiar el filtro a “My Assets”, como se observa en la Figura 13. Ahora, se puede seleccionar los materiales que se desean descargar, como se muestra en la Figura 14.

Para descargar una textura de “Poliigon” (o de cualquier página), se hace un procedimiento similar al de Unity. Ya seleccionado el elemento, dar click en descargar. En este caso, la página web, comprimirá el archivo y pedirá una ubicación para guardarlo; para este proyecto es conveniente guardar el archivo en una carpeta que está dentro de los objetos de trabajo del proyecto (Figura 15).

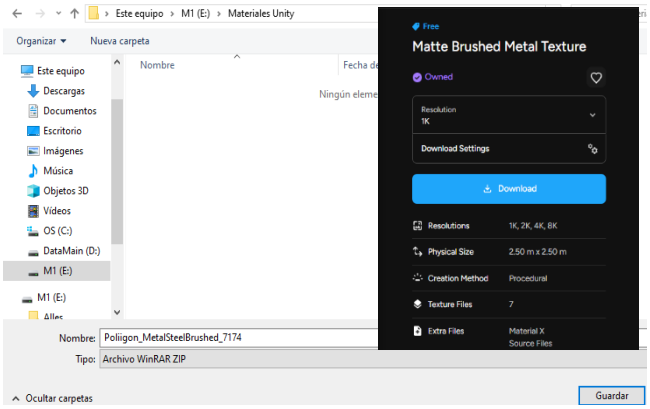


Figura 15 Descarga de un paquete de texturas de la página Poliigon

Posterior a la descarga, descomprimir los archivos. Desde Unity, generar una carpeta, junto a las carpetas que se generan de las descargadas desde la AssetStore, y nombrarla de forma conveniente para identificar que son los archivos de la web, como en la Figura 16. Dentro de la nueva carpeta, dar click derecho y crear un nuevo material, Figura 17. En la carpeta donde se descomprimió el archivo, copiar y pegar los archivos en el espacio designado al nuevo material que se ubica en el inspector; de tal forma que las propiedades “Albedo”, “Metallic”, “Normal Map”, “Height Map”, “Oclussion”, “Detail Mask” se agreguen de acuerdo a como están nombrados los archivos de origen (Figura 19). Generalmente, al descargar estos archivos, se identifican apropiadamente para configurar el material tal cual se ve en su página.

Al final, la textura debe tener una apariencia igual al que se presenta en la página web de donde se obtiene, Figura 18.

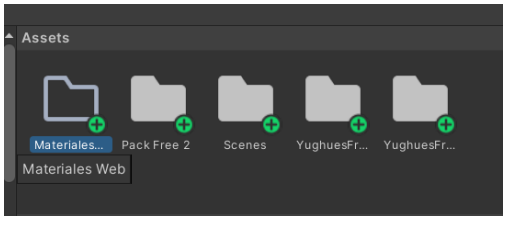


Figura 16 Creación de carpetas para generar un nuevo material

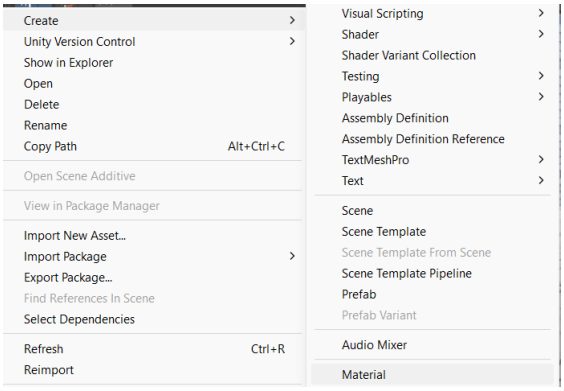


Figura 17 Materiales web -> Create -> Material

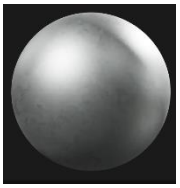


Figura 18 Textura mostrada en la página Poliigon

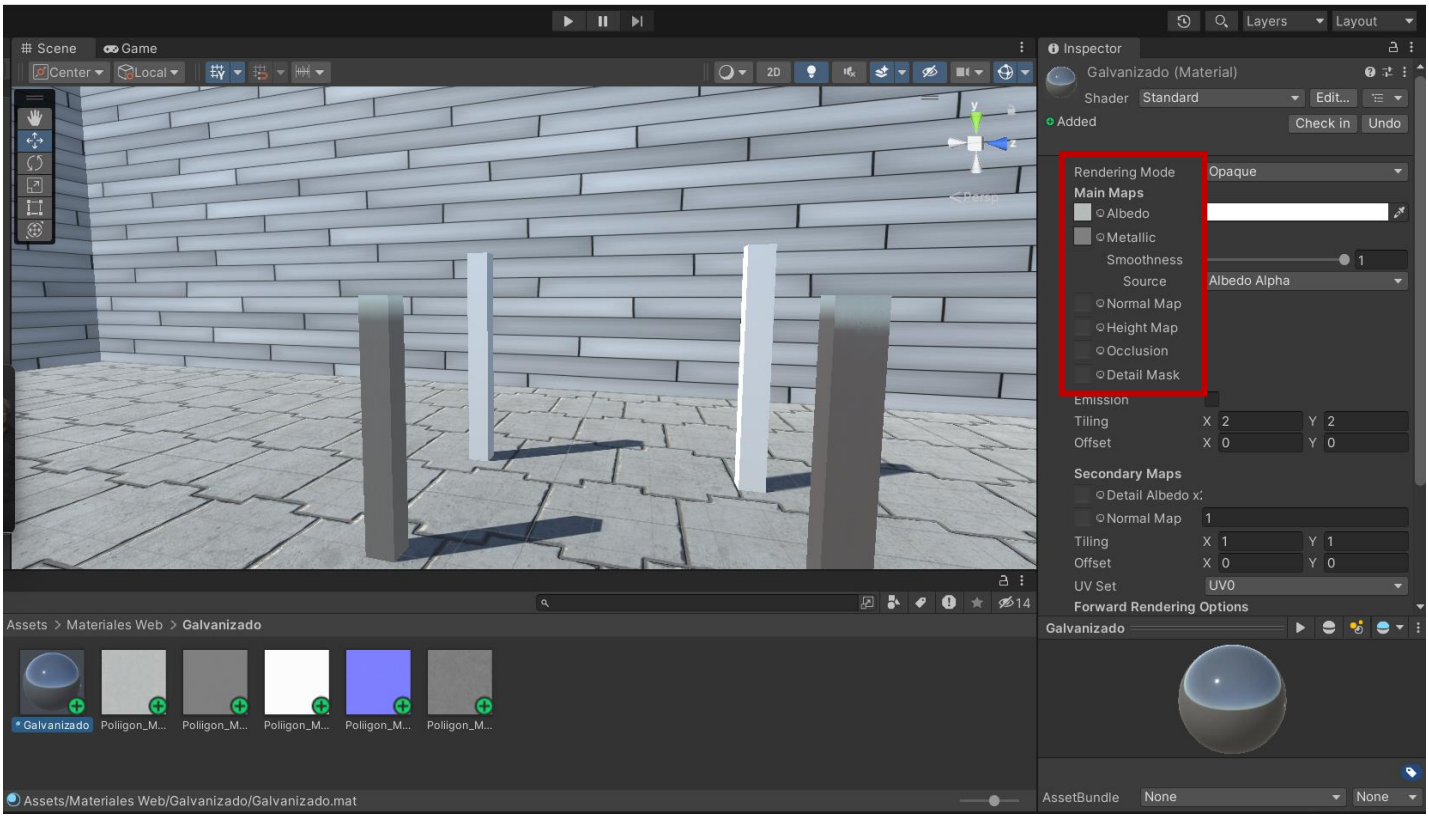


Figura 19 Integración de un nuevo material desde una página web

Como se comentó al inicio de este apartado, existe otra manera más de generar texturas para los objetos en Unity. Esta manera es directamente creando un material y sobre este material, configurar manualmente los valores del “Main Maps”, Figura 17. Del lado derecho de la opción “Albedo”, se encuentra un ícono de pintura, el cual, al presionarlo, despliega una paleta de colores en donde se puede seleccionar el que sea más adecuado. En la slider de “Smoothees”, al desplazarlo, se puede lograr una textura metalizada o plastificada, de acuerdo a la necesidad del creador. En la pestaña “Rendering Mode”, se puede seleccionar si el material es opaco o translúcido, una herramienta que es bastante conveniente cuando se necesita que un objeto sea transparente para observar algo de su interior, Figura 19.

Apariencia de objetos tridimensionales

Para agregar la textura a un objeto tridimensional, solo basta con dar click al material que se ha elegido y llevarlo hasta el objeto (Figura 20). Otra forma es dar click sobre el objeto, ubicar el material en las carpetas del proyecto y llevarlo hacia el inspector, con ello, Unity entenderá que se está agregando una propiedad de material al objeto.

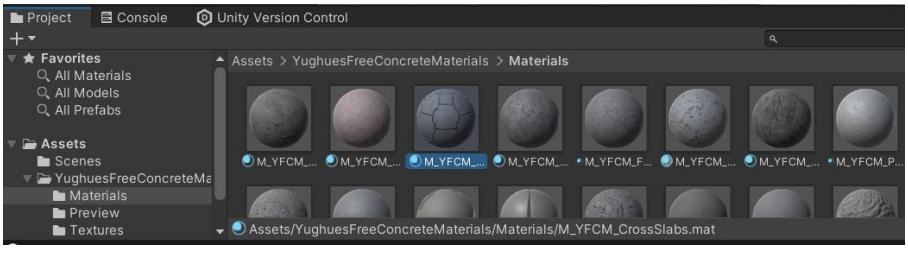


Figura 20 Carpeta Assets del proyecto, en donde se ubica las carpetas de descarga

Para el piso de este proyecto, se descargó una textura de malla, descargada de la Unity Store (se nombra como el creador la haya colocado) y se agrega el material al plano, Figura 21.

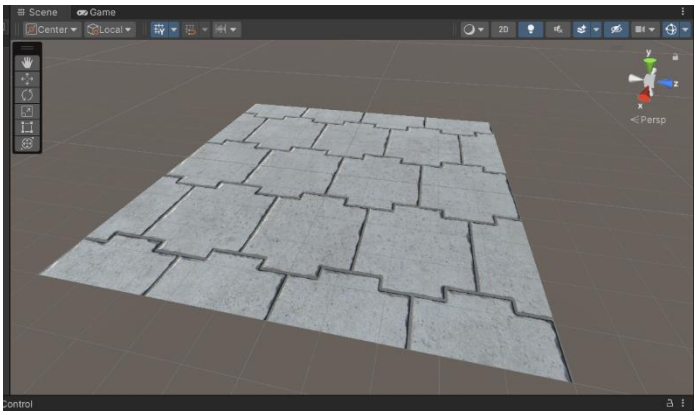


Figura 21 de piso en el proyecto8

En ocasiones, las texturas descargadas vienen en un tamaño por defecto y no se acoplan adecuadamente a los objetos tridimensionales, por tanto, se debe ajustar para un efecto más natural o visualmente agradable. En el proyecto, dar click sobre el piso; en el inspector, buscar las propiedades del material (Material), como en la Figura 22. En la opción “Tilling”, colocar los valores en “X” y “Y” más adecuados para que la textura adquiera el ajuste que se necesita.

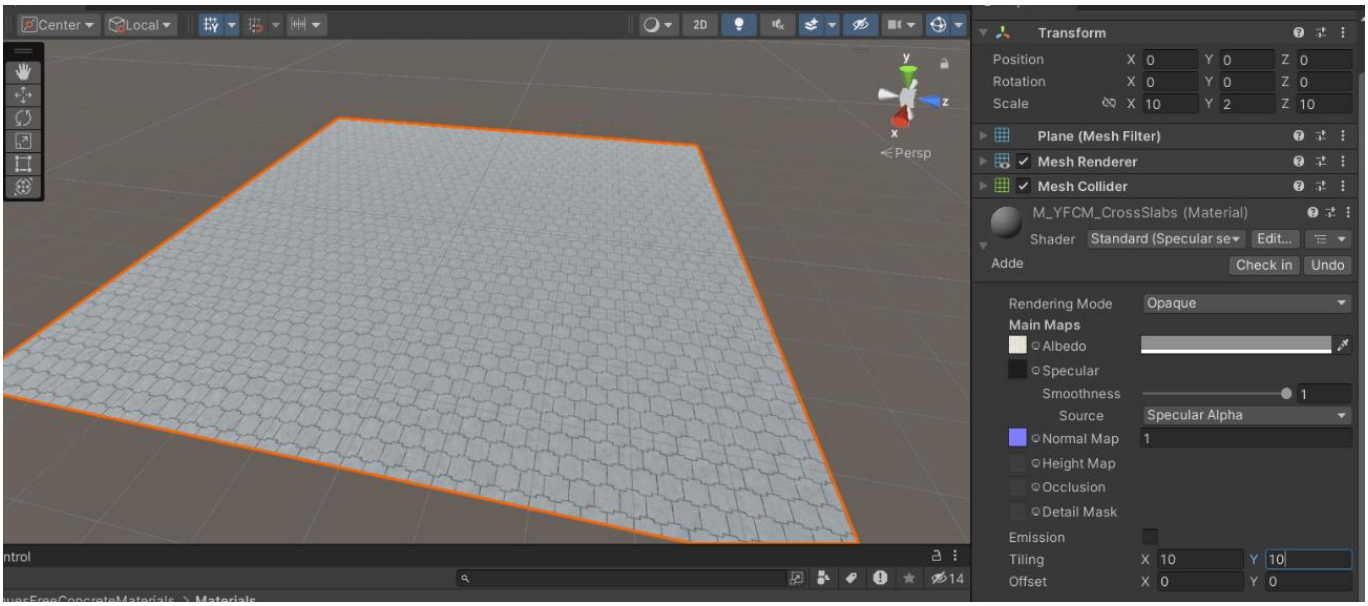


Figura 22 Ajuste visual de la textura de baldosas para el piso

Con las herramientas de Unity, crear un entorno que permita recrear una estación de trabajo neumática (Mesa neumática). En el siguiente apartado, se mencionará cómo se integrarán objetos específicos, que pueden ser diseñados con herramientas de software CAD: pistones, torreta y botonera.

III. Equipo de trabajo

Generación de dos mesas de trabajo

Gracias a las herramientas utilizadas en el apartado anterior de este manual (*Nuevo proyecto en UNITY*), se modeló un objeto tridimensional a partir de planos y rectángulos, con el fin de representar una mesa similar a las que se encuentran físicamente en el laboratorio de Automatización Industrial (Figura 23). Además, se aplicaron texturas metálicas y se duplicó el modelo para obtener dos mesas dentro del escenario (Figura 24).

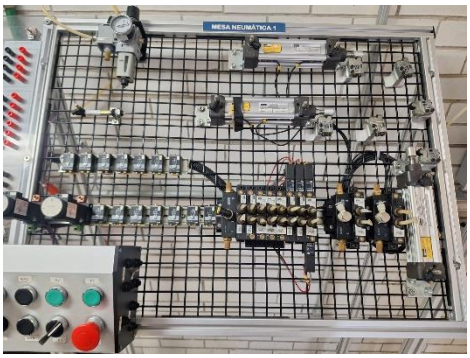


Figura 23 Equipo neumático y electroneumático en el Laboratorio de Automatización Industrial

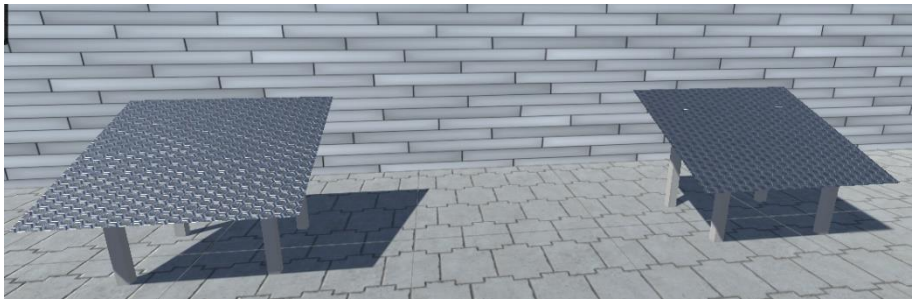


Figura 24 Mesas virtuales para pruebas de comunicación

Para complementar el espacio de trabajo, en la Figura 24 se observa que las mesas virtuales se encuentran en un entorno donde se ha añadido planos que simulan paredes, con texturas que logran hacerlo bastante identificable.

Modelos tridimensionales mediante software CAD

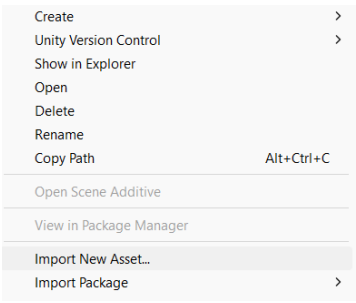


Figura 25 Importar objetos de extension 3D Object

Para integrar un modelo tridimensional complejo o que requiere mayor cantidad de herramientas para ser diseñado, como es el caso de los elementos que a continuación se presentan, se hace uso de software especializado en el diseño. Por tanto, al integrar estos nuevos elementos, se genera una carpeta que contenga todos los archivos. Una vez renombrada la carpeta, dentro de ella, dar click derecho y seleccionar “Import New Asset…”, Figura 25.

Al integrar un objeto diseñado en un software externo a Unity, es importante verificar que el archivo utilice una de las extensiones compatibles: “.fbx”, “.dae”, “.3ds”, “.dxf”, “.obj” o “.skp”. Si el objeto no se encuentra con alguno de estos formatos, Unity no podrá reconocerlo ni importarlo correctamente. [2][3]

Los elementos que a continuación se muestran, se diseñaron en el software *Inventor,* exportados a la extensión “.obj”. Se recomienda también usar el software *Blender,* ya que, además de ser de licencia libre, permite obtener información sobre la cantidad de polígonos que se han generado en las piezas

diseñadas; incluso permite reducir el número de polígonos generados mediante sus múltiples herramientas. El número de polígonos es un factor importante, ya que, al reducirlos en un modelo, la carga y procesamiento de datos permite un mejor rendimiento, tanto del programa como del equipo de cómputo que ejecuta el programa.

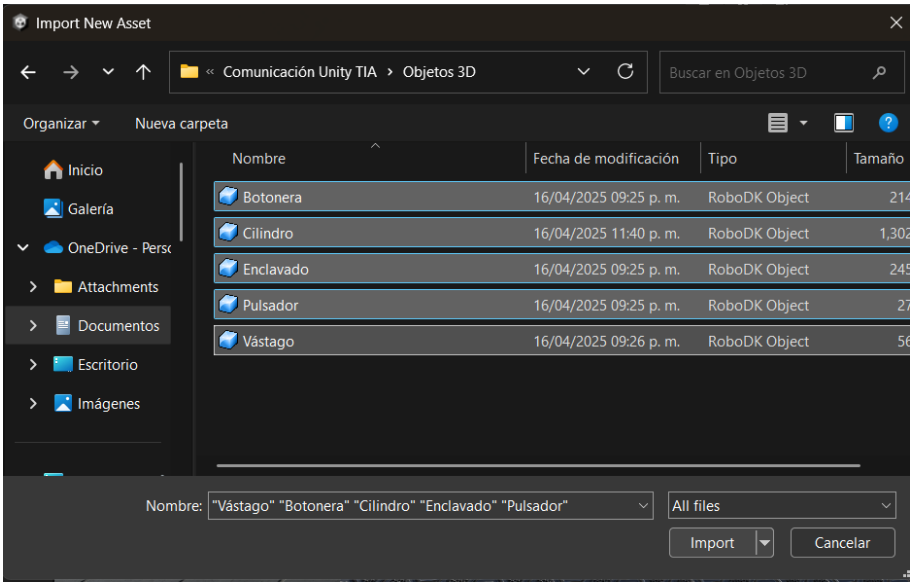


Figura 26 Importación de elementos tipo ensamble en extensión 3D Object

Para el desarrollo de los componentes que se utilizaron en el proyecto, fue necesario consultar la documentación del fabricante, debido a que ellos proporcionan las características de sus equipos, como las dimensiones. Tener las dimensiones del equipo conlleva una correcta digitalización, dado que permite mantener las proporciones en caso de ser escaladas.

Regresando a la importación de los objetos (Figura 25), seleccionar la ruta a los archivos y elegir el o los elementos para agregar, en este caso, todos los elementos que se observan en la Figura 26; dar click en el botón “Import”. Unity tardará algunos segundos en integrar por completo a los elementos. Es importante entender que el software no conservará los materiales o características visuales que se le hayan hecho a los objetos desde el software de diseño como se muestra en la Figura 27. Unity solo integrará sólidos, por ello es importante generar los sólidos necesarios para cada elemento. En la Figura 27, los sólidos serían: cuerpo de la torreta (color blanco), luz verde, luz ámbar y luz roja. En Unity será significativo contar con las texturas adecuadas o crearlas, para que el elemento sea visualmente entendible.

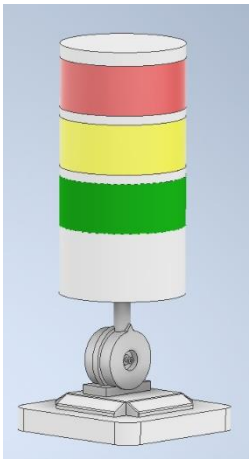


Figura 27 Torreta de tres colores

Cuando se crea la pieza, hay que considerar la funcionalidad que va a tener. Al generar diversos sólidos dentro de la misma pieza, permite que Unity no integre una sola textura al cuerpo. Sin embargo, el generar diversos sólidos, no es lo mismo que tener varias piezas del objeto. Por ejemplo, si se diseñar un pistón con vástago mediante varios sólidos y se integra a Unity, Unity entenderá que el pistón y el vástago son una única pieza (un sólido) y hará imposible la animación de desplazamiento ya que todo el objeto se moverá. Por tanto, se tendrán que integrar de forma individual estos elementos. La generación de sólidos en una pieza permite detallar la pieza. Por ejemplo, en la torreta (Figura 27) al agregarle una textura, todo sería de un solo color.

Retomando el concepto de gemelo digital; “*comprenden dos interpretaciones de un mismo sistema, uno de características virtuales y el otro de elementos físicos, ambos funcionan a la par de tal forma que permita el monitoreo de datos de interés o integrarlo a un sistema más complejo, además de permitir observar su comportamiento en tiempo real permitiendo una retroalimentación por ambas*

partes". Con ese concepto en mente y considerando los equipos físicos que existen en el laboratorio, se modelaron o *digitalizaron* equipos que envían y reciben señales discretas.

Para la parte de envío de señales se digitalizó una botonera y sensores de efecto Reed. Para los elementos que reciben señales, indicadores luminosos (torreta industrial) y un cilindro de doble efecto; ambos elementos también se pueden definir como actuadores.

Como se comentó previamente, se elaboró un armazón de la botonera y por aparte, se diseñaron los botones (Figura 28); cuenta con tres botones, dos pulsadores y un enclavado; en este proyecto se programarán para:

- Botón enclavado superior: desactivar cualquiera de las acciones de los eventos anteriores (Figura 29).
- Botón pulsador intermedio: activación de proceso secuencial (Figura 30).
- Botón pulsador inferior: activación de secuencia de prueba (Igual a la Figura 30).

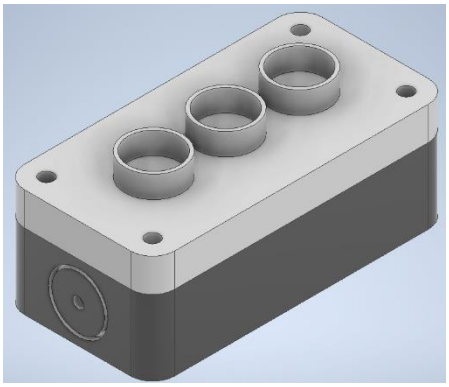


Figura 28 Botonera de control

Figura 29 Botón enclavado

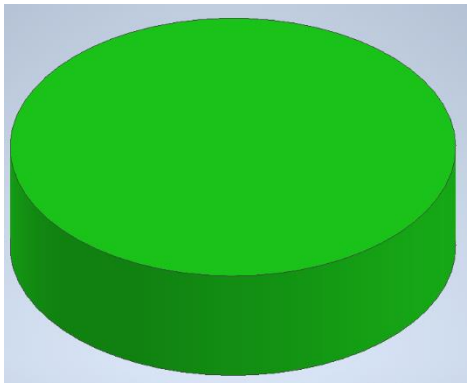


Figura 30 Botón pulsador

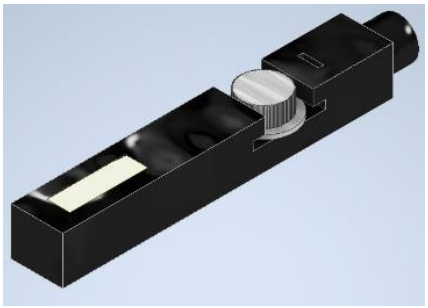


Figura 31 Sensor Reed

Los sensores de efecto Reed (Figura 31) permiten identificar en que posición se encuentra el vástago del pistón, por consiguiente, para cada pistón se requieren dos elementos como estos. Indicarán si el vástago está en retraído o extendido, enviando un valor alto (true), como señal de interés.

En cuanto a los elementos que responden a las señales de salida, se virtualizó una torreta industrial que cuenta con tres indicadores luminosos. Estos indicadores pueden mostrar distintos estados del sistema (encendido, falla, mantenimiento, etc.). En este manual, la torreta al inicio solo encenderá y apagará uno de sus indicadores y posteriormente realizará una rutina de control más compleja, como el funcionamiento de un semáforo; Figura 32.

Para el pistón, el modelo requirió más detalles y la consideración del desplazamiento de su vástago. Sin embargo, a pesar de ser una única pieza, se dividieron partes estratégicas para lograr que las texturas lo hicieran más vistoso. Algunos ejemplos de sólidos son: racores, tapas y tuercas; Figura 33.

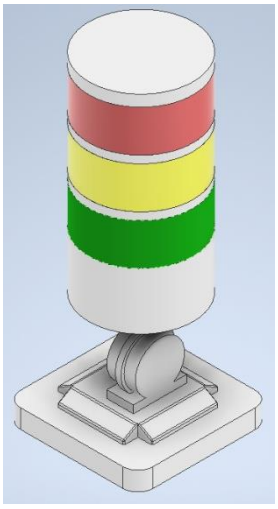


Figura 32 Torreta industrial

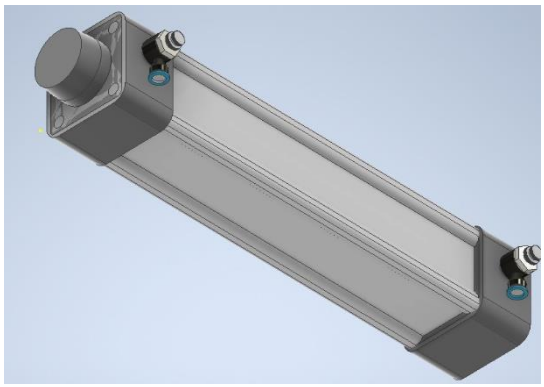


Figura 33 Cilindro de doble efecto

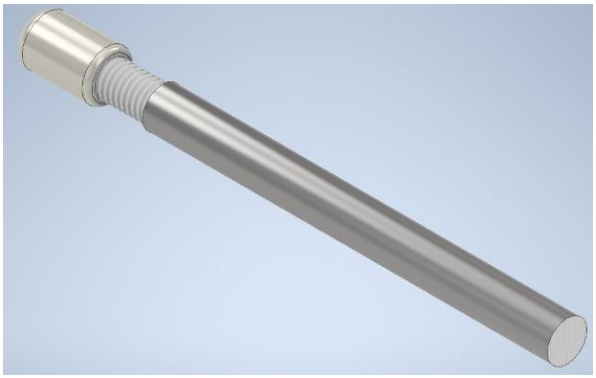


Figura 34 Vástago del cilindro

Y al final, se elaboró el vástago. Al igual que en la botonera, este elemento será animado, por lo que requiere ser independiente al cuerpo del cilindro. Contará con un desplazamiento positivo (posición de trabajo) y negativo a partir de la posición de extensión (posición de reposo).

Una vez integrados los elementos y definidos sus materiales en el proyecto de Unity, estos se distribuirán en las dos mesas virtuales, para simular estaciones de trabajo independientes.

- En la mesa 1 se colocará una botonera y la torreta.
- En la mesa 2, se ubicará una segunda botonera y tres pistones.

En la mesa 1 se busca activar el encendido y apagado de un indicador luminoso y posteriormente el encendido de todos los indicadores en una secuencia similar a los semáforos. El primer ejercicio se logra al presionar el botón pulsador inferior de la botonera; el segundo ejercicio se activará con el botón pulsador intermedio de la botonera. Ambos sistemas se detienen con el botón de paro (botón superior). En la Figura 35 se muestra el sistema.

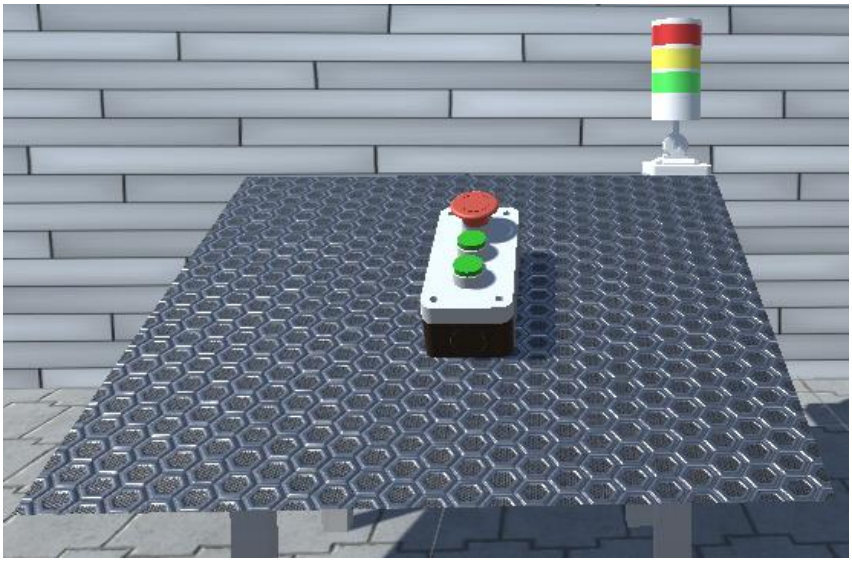


Figura 35 Mesa 1 en Unity

En la mesa dos, tras dominar la activación y desactivación de un solo pistón, se desarrollará una rutina de control más compleja, en la que tres pistones ejecuten una secuencia automática. En la botonera, el botón pulsador inferior activará a un pistón, el botón pulsador intermedio activará la secuencia y el botón enclavado (botón superior), detendrá todas las ejecuciones y regresará a la posición inicial los vástagos de todos los pistones. En la Figura 36 se aprecia la mesa 2 con todos los elementos previamente descritos, cabe resaltar que los pistones cuentan con sus dos sensores de efecto Reed.

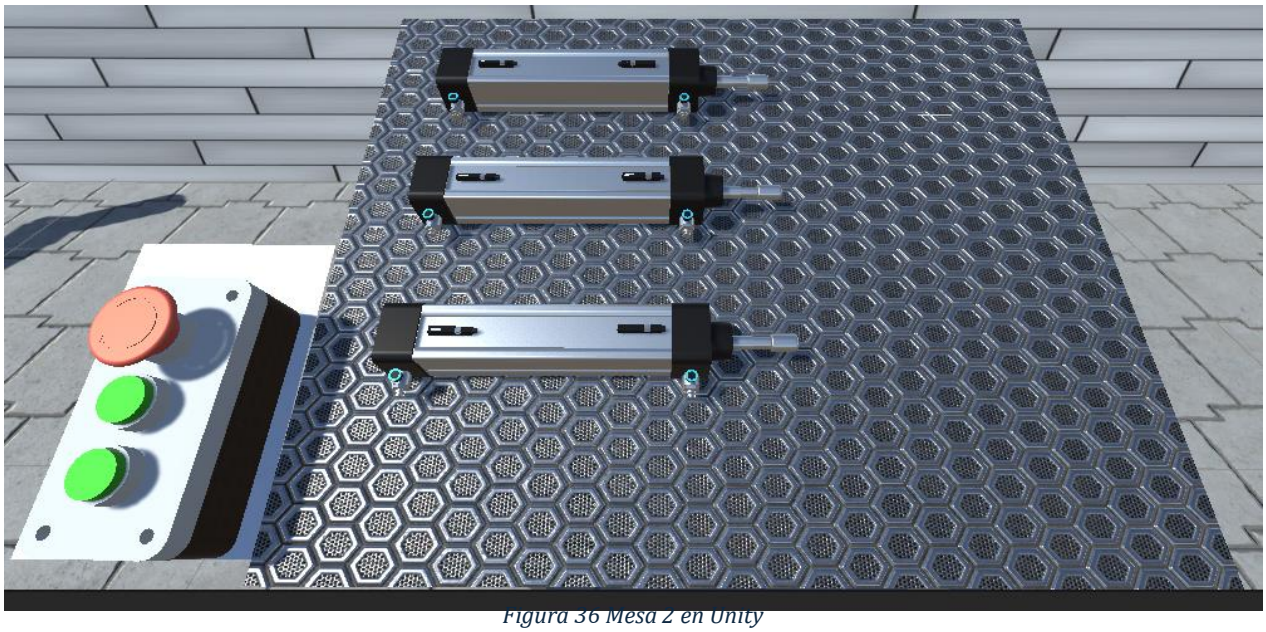


Figura 36 Mesa 2 en Unity

Interacción de la escena de trabajo

Con el escenario completamente representado, se recomienda integrar una cámara en tercera persona que pueda desplazarse a lo largo del entorno. La intención es lograr una interacción más fluida y realista durante la simulación. Para ello, se sugiere seguir el tutorial disponible en YouTube titulado "*Como MOVER una CAMARA en UNITY 3D con el MOUSE"*; el enlace se encuentra en el apartado de *Referencias* al final de este manual. [4,5]

El procedimiento en el video es:

1. Crear un objeto vacío en el apartado de jerarquía.
2. Dentro de este objeto, agregar un cuerpo tridimensional (por ejemplo, una cápsula).
3. Añadir una cámara y vincularla al mismo objeto vacío.
4. Crear un script *Mirada_Camara,* que permitirá controlar la orientación de la vista utilizando el movimiento del mouse.
5. Finalmente, crear el script *Movimiento_Personaje*, que habilitará el desplazamiento del personaje dentro del escenario mediante las teclas direccionales o las teclas "a,w,s,d".

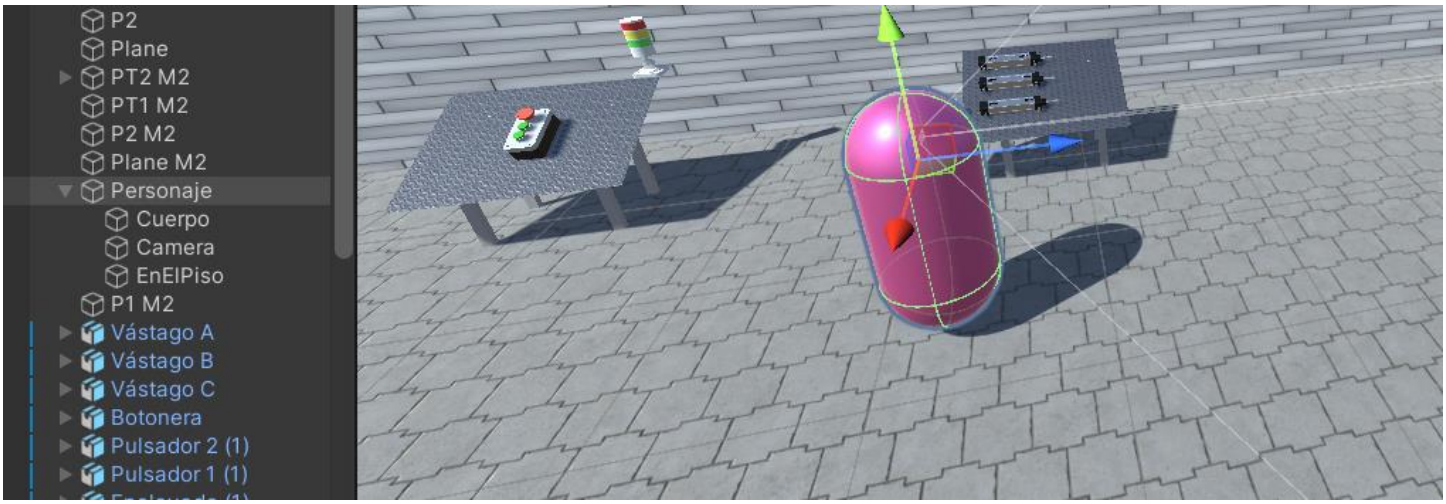


Figura 37 Cápsula que funciona como un elemento interactivo

En el anexo de este manual se puede encontrar el código pertinente a los scripts que permiten desarrollar lo anterior. En la Figura 37 se observa una captura relacionada con los objetos creados en la jerarquía y una vista general del personaje que permite la movilidad dentro de la escena. Con un personaje móvil en la escena, las activaciones y desactivaciones de los eventos se producirán únicamente cuando el usuario interactúe con ellas, algo similar a acercarse a la mesa física y presionar botones.

Es recomendable generar una carpeta en los *Assets* que se llame "*Scripts*", permitiendo tener mayor organización de la información. Adicionalmente, como buena práctica, crear subcarpetas que definan hacia dónde está orientados los scripts; por ejemplo, la carpeta *Cámara*, en donde se encuentran los scripts que interactúan con la vista del personaje, como se muestra en la Figura 38.

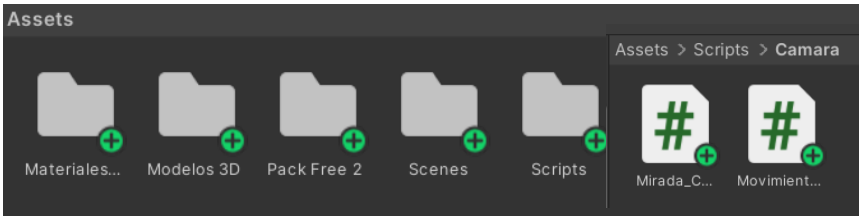


Figura 38 Assets -> Scripts -> Camara

IV. Programación de eventos

Una vez construida la escena visual y creado el personaje móvil capaz de interactuar con su entorno, es necesario programar los eventos y acciones que realizarán algunos objetos tridimensionales, conocidos también como animaciones. Para ello, se desarrollan programas utilizando el lenguaje C#, con ayuda del entorno de programación *Visual Studio*, el cual Unity preconfigura automáticamente al momento de crear y editar los scripts. Con este procedimiento, se establece la base para controlar las interacciones dinámicas dentro del escenario, permitiendo que los objetos respondan a acciones o estímulos definidos por el usuario.

En el capítulo anterior se trabajaron algunos scripts básicos apoyados en un tutorial de YouTube. Es importante considerar que este manual presenta un entorno de trabajo personalizado, por lo que en algunas ocasiones será poco probable encontrar tutoriales que coincidan exactamente con los requerimientos planteados. En los apartados siguientes se desarrollarán scripts específicos para realizar distintas funciones dentro del entorno de Unity, tales como:

- Encendido y apagado de la luz verde en la torrera
- Animación de los botones pulsadores y enclavados
- Desplazamiento de los vástagos de los pistones

Cada script será explicado de forma detallada, y al final del manual (en el *Anexo*) se incluirán los códigos completos utilizados a lo largo de este proyecto. Al realizar esto, los usuarios podrán consultar o adaptar fragmentos de código específicos según sus propias necesidades.

Por último, previo a conectar el entorno virtual con el físico, se recomienda tener construida toda la lógica de animación del escenario; no existen restricciones al programar estos entornos, sin embargo, para este manual, se desarrolla de esta manera. De ahí que inicialmente solo se verá el código para las animaciones virtuales y al final se integran las variables que recibirán las señales físicas para hacer funcionar todo el escenario.

Activación de eventos mediante un *RayCast*

La interacción entre los objetos tridimensionales y un personaje que se desplaza dentro de la escena, es algo común en la programación de videojuegos que utilizan a Unity como su motor gráfico y para ello existe el método *RayCast*. El RayCast activa eventos al hacer coincidir un rayo con los objetos en la escena, Figura 39. El RayCast se liga a la cámara del jugador, por ende, se añade un script que genera un rayo direccional; al entrar en contacto con la física de algún elemento (por ejemplo, un botón) y con referencia de una acción (un click o presionar una tecla), se puede activar o desactivar una función (encender o apagar una luz, por ejemplo). En

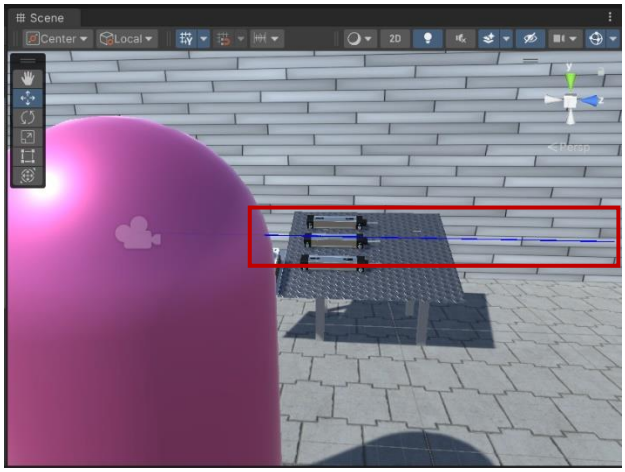


Figura 39 Rayo generado a partir de la cámara

general, el RayCast permite que las acciones de la escena solo sucedan cuando se interactúa con los elementos, algo similar a la realidad; físicamente, no podemos estar alejados de un botón y accionarlo.

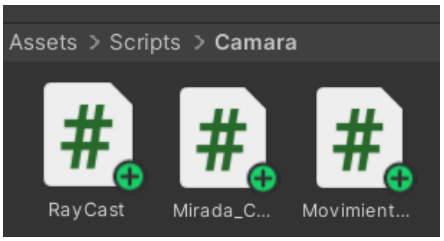


Figura 40 Carpeta "Cámara" -> Scripts contenidos

Cómo se mencionó, el método se liga directamente a la cámara del personaje, que hasta el momento ya se puede mover a través de todo el escenario. Por consiguiente, dentro de la carpeta *Cámara*, se creará un script que se llamará "*RayCast*", Figura 40. Con ayuda del tutorial "*Como hacer un interruptor de luz en Unity*", a partir del minuto 5:55, se genera el código que se encuentra disponible en el anexo de este manual (*Código de programación para el script que activa el RayCast*). [6]

Hasta el momento, los scripts que se han creado, con ayuda de los tutoriales de YouTube, y que se encuentran ligados a la cámara del jugador son: "Mirada_Camara", "Movimiento_Personaje" y "RayCast" (cómo se muestran en la Figura 40).

Con la programación desarrollada hasta este punto, la escena comienza a presentar una interacción más dinámica, entre el usuario y los objetos del entorno. Para mejorar esta cualidad, y con apoyo en la documentación disponible de Unity, se implementó un tutorial que permite modificar la apariencia visual de un objeto tridimensional al hacer coincidir el RayCast con su física.

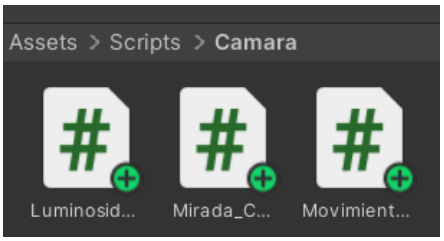


Figura 41 Script Luminosidad en carpeta "Cámara"

El procedimiento se base en el video tutorial *"Como INTERACTUAR con objetos ¡RAPIDO Y SENCILLO! Unity 2021",* cuyo método fue adaptado al proyecto. De manera general, se debe generar un nuevo script denominado *Luminosidad* (disponible en el Anexo), en el cual se incorpora un efecto visual tipo "Hover". Este efecto permite indicar al usuario que un objeto es interactivo o accionable, ya que se activa con la mirada (RayCast). Por consiguiente, el script deberá almacenarse dentro de la carpeta *Cámara (*Figura 41). [7]

Al ejecutar el código, el resultado es que los objetos interactivos (botones) presentan una luminosidad o un resplandor, ocasionando un cambio de apariencia llamativo. La Figura 42 muestra cómo es el entorno sin interacción; por su parte, en la Figura 43 del lado derecho (ventana Game), se aprecia un cambio notorio en el color, ahora es más brillante. Del lado izquierdo de la Figura 43 (ventana Scene), se aprecia el momento exacto en donde el rayo coincide con el botón. No olvidar que para que este efecto sea posible, el material que integra al botón debe tener marcada la casilla de "Emission", como se muestra en la ventana principal de la Figura 43.

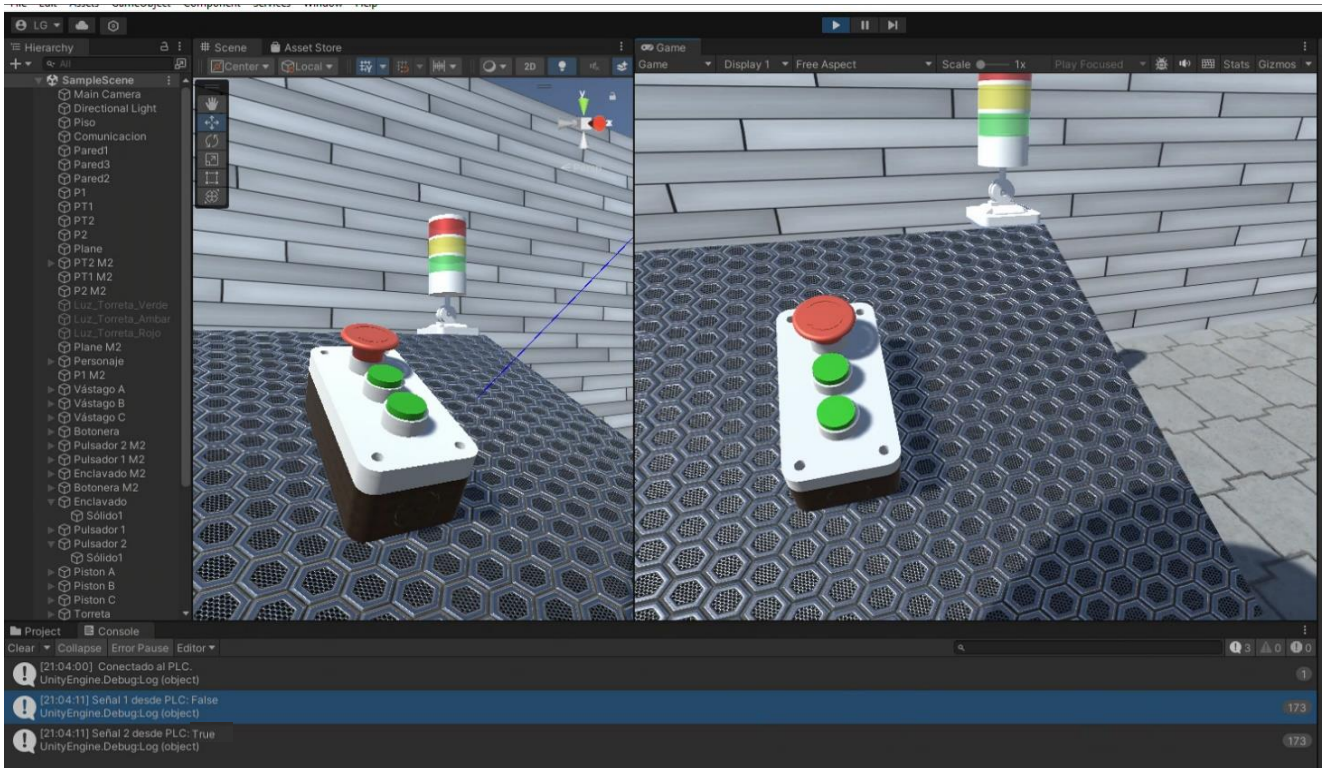


Figura 42 El RayCast no incide sobre el botón

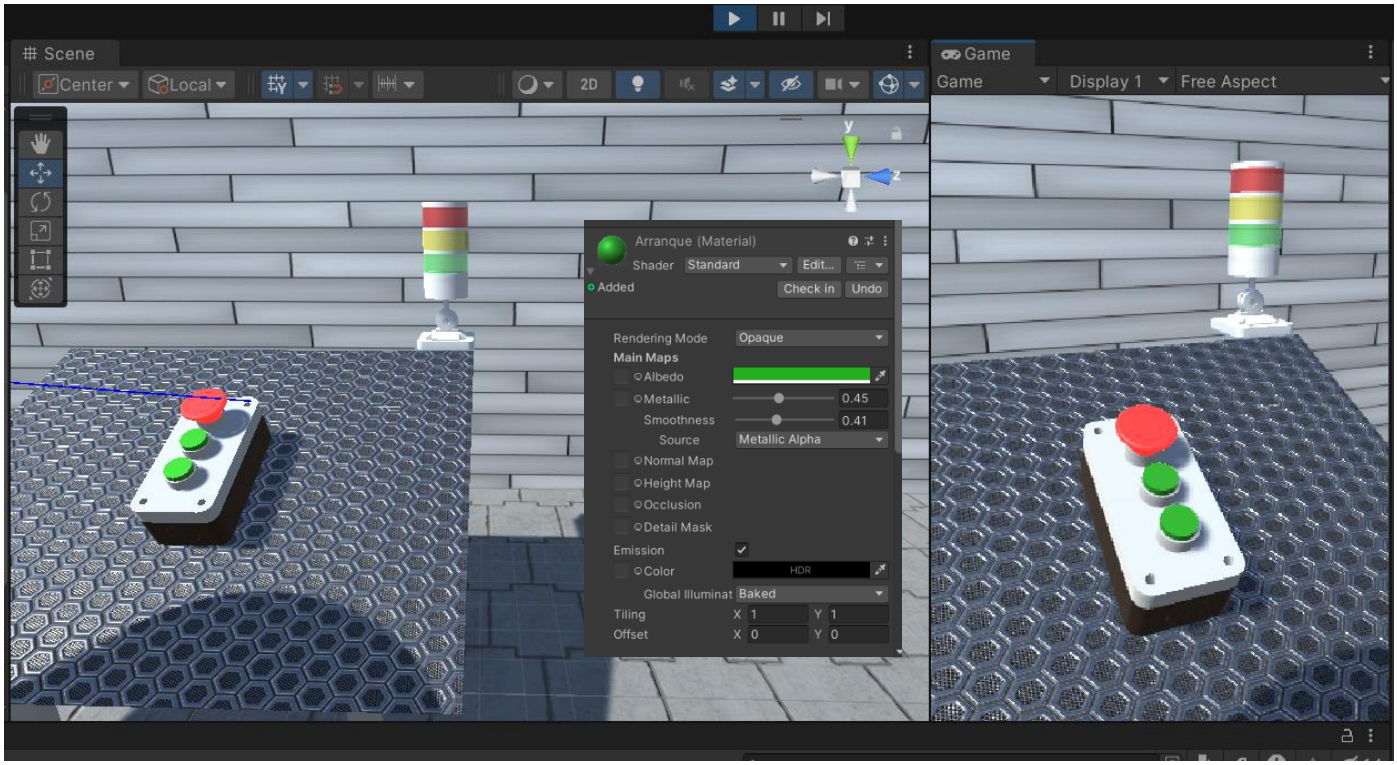


Figura 43 Efecto "Hover"

Activación y desactivación de un indicador luminoso

En este apartado, se abordarán tres animaciones diferentes; pulsar un botón para encender la luz; enclavar un botón para apagar la luz; encender y apagar un objeto luminoso.

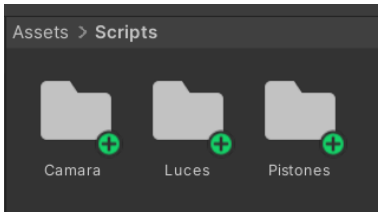


Figura 44 Carpeta de Scripts

- Botón pulsador

Para este evento, se generó una carpeta en los scripts de nombre “Luces” (Figura 44). Una vez dentro de esta carpeta, generar el script “BotonPulsadorB1”. Dentro del script (en Visual), declarar tres variables públicas del tipo flotante, como en la Figura 45; *distancia* y *distancia2*, configurarán el desplazamiento del botón en los ejes X, Y. Se debe realizar de esta manera por la posición inclinada de los botones en la botonera; *velocidad*, por su parte, configurará la velocidad a la cual se desplaza el botón.

```
BotonPulsadorB1.cs | Luminosidad.cs
Archivos varios | BotonPulsadorB1
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class BotonPulsadorB1 : MonoBehaviour
6   {
7       public float distancia = 1f;         // Cuánto se mueve el botón
8       public float distancia2 = 1f;
9       public float velocidad = 40f;           // Qué tan rápido se mueve
10
11      private Vector3 posicionInicial;
12      private Vector3 posicionPresionada;
13
14      private bool presionado = false;
```

Figura 45 Declaración de variables

Vector3 (*posicionInicial* y *posicionPresionado* [variables privadas]), es un tipo de variable que indicará a Unity que se está declarando un vector de posición; con coordenadas X, Y, Z.

Finalmente, *presionado* (variable privada) del tipo booleana, guarda el estado de activación o desactivación del botón.

```
17      void Start()
18      {
19          posicionInicial = transform.position;
20          posicionPresionada = posicionInicial - new Vector3(distancia2, distancia, 0);
21      }
```

Figura 46 Void Start

Para establecer una referencia de posición al iniciar el sistema, es necesario almacenar las coordenadas iniciales del objeto (*posicionInicial*), así como las coordenadas de desplazamiento (*distancia2* y *distancia*) que se registrarán cuando se ejecute el bloque “void Start” (Figura 46). La distancia objetivo a desplazarse (*posicionPresionada*), se obtendrá de la diferencia de la posición inicial y el nuevo vector de posición que tiene los parámetros de posición que manualmente se configura. El eje Z permanecerá fijo.

Para concluir con el código, se genera una función de carácter público, “Pulso”; *Pulso* se ejecuta al interactuar con el RayCast. En la Figura 47 se encuentra el código de la función *Pulso,* que se divide en cuatro casos.

- Primer caso: si el botón recibe el evento, click del mouse, *presionado* tomará el valor de verdadero.
- Segundo caso: si el botón dejó de recibir el evento, click del mouse, *presionado* es falso.
- Tercer caso: si *presionado* es verdadero, el objeto se desplaza a una segunda posición.
- Cuarto caso: si *presionado* es falso, el objeto no se desplaza o regresa a su posición de origen.

No olvidar agregar el script *BotonPulsadorB1* al objeto tridimensional.

```
24      public void Pulso()
25      {
26          if (Input.GetButtonDown("Fire1"))
27          {
28              presionado = true;
29          }
30
31          if (Input.GetButtonUp("Fire1"))
32          {
33              presionado = false;
34          }
35
36          // Movimiento suave
37          if (presionado)
38          {
39              transform.position = Vector3.Lerp(transform.position, posicionPresionada, Time.deltaTime * velocidad);
40          }
41          else
42          {
43              transform.position = Vector3.Lerp(transform.position, posicionInicial, Time.deltaTime * velocidad);
44          }
45      }
46  }
47
```

Figura 47 Estado de variable "pulso" y desplazamiento del objeto

- Botón de paro

De manera similar al código anterior, se generan tres variables públicas del tipo flotante para configurar desplazamiento (*distancia* y *distancia2*) y velocidad (*velocidad*) para la cinemática del objeto. Agregar dos variables privadas más del tipo Vector3 (*posicionInicial* y *posicionPresionada*). Además, agregar una variable de estado del tipo privada y booleana inicializada en *falso*; ver Figura 48.

```
7       public float distancia = 1f;      // Movimiento en Y
8       public float distancia2 = 1f;     // Movimiento en X
9       public float velocidad = 10f;     // Velocidad del movimiento
10
11      private Vector3 posicionInicial;
12      private Vector3 posicionPresionada;
13      private bool enclavado = false;
```

Figura 48 Declaración de variables

Para la referencia de posición inicial, *void Start* guarda la referencia de la posición inicial (*posicionInicial*) y la posición objetivo (*posicionPresionada*), como se muestra en la Figura 49.

```
15      void Start()
16      {
17          posicionInicial = transform.position;
18          posicionPresionada = posicionInicial - new Vector3(distancia2, distancia, 0);
19      }
```

Figura 49 void Start para botón enclavado

En el último segmento de programación se desarrolla la función de enclavamiento; en la Figura 50 se aprecian tres casos.

- Primer caso: si se presiona el botón izquierdo del mouse en una ocasión, *enclavado* se negará.
- Segundo caso: si *enclavado* es verdadero, el objeto tridimensional se desplazará a *posicionPresionada* (enclavado).
- Tercer caso: si *enclavado* es falso, el objeto tridimensional se desplazará a su posición inicial.

```
21      public void Enclavar()
22      {
23
24          // Detectar clic con Fire1 y raycast sobre este objeto
25          if (Input.GetButtonDown("Fire1"))
26          {
27              enclavado = !enclavado;
28          }
29
30          // Movimiento según estado enclavado
31          if (enclavado)
32          {
33              transform.position = Vector3.Lerp(transform.position, posicionPresionada, Time.deltaTime * velocidad);
34          }
35          else
36          {
37              transform.position = Vector3.Lerp(transform.position, posicionInicial, Time.deltaTime * velocidad);
38          }
39      }
```

Figura 50 Función "Enclavar"

- <u>Encender y apagar un *Point light*</u>

Para el evento de encendido y apagado de una luz, añadir un *Point Light* (Figura 51). Renombrarlo como Luz_Torreta_Verde y ubicarlo en la posición más conveniente de la escena. Como esta luz estará contenida en un objeto, se utiliza el tutorial "*Objetos transparentes en Unity*", Con ello la luminosidad tendrá un efecto de mejor calidad. [8]

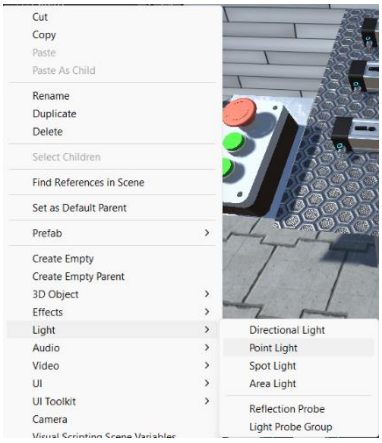


Figura 51 Hierarchy -> Click derecho -> Light -> Point Light

Con la luz en su posición ideal, desactivar la paloma que se observa del lado izquierdo del nombre del elemento en el inspector, ver Figura 52.

Dentro de la carpeta Luces, generar un nuevo script llamado "Luz_Verde". Abrir el editor de Visual y editar con la lógica que a continuación se recomienda.

Integrar una variable pública del tipo *GameObject*, que permite referenciar a *Luz_Torreta_Verde*. Agregar una variable pública y booleana (luz) (Figura 53). A continuación, crear la función *EncenderLuz*, que permite activar el *GameObject* (Figura 54). Finalmente, agregar la función *ApagarLuz*, que desactiva al GameObject (Figura 55).

La llamada a estos eventos se hace mediante el *RayCast,* (ver Anexo de este manual).

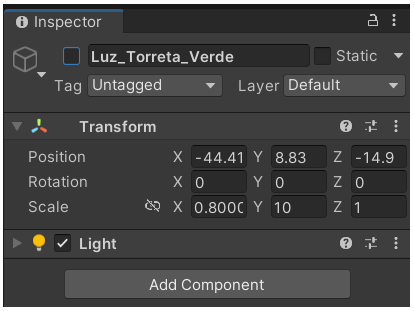


Figura 52 Seleccionar el light point -> Inspector -> Desactivar casilla

```
8       public GameObject lVerde;
9
10      public bool luz;
```

Figura 53 Declaración de variables

```
12      public void EncenderLuz()
13      {
14          lVerde.SetActive(true);
15      }
```

Figura 54 Función "EncenderLuz"

```
17      public void ApagarLuz()
18      {
19          lVerde.SetActive(false);
20      }
```

Figura 55 Función "ApagarLuz"

Con la integración de los códigos que se han presentado hasta el momento, la escena de Unity contiene una animación de un botón pulsador y un botón enclavado que activan y desactivan un indicador luminoso verde, respectivamente, ubicados en la mesa virtual 1. Ahora, los usuarios identifican correctamente el elemento con el que puden interactuar y una animación entendible y funcional. En la Figura 56 se encuentran las ventanas de *Scene* (izquierda) y *Game* (derecha); al ejecutarse el juego, los botones inferiores de ambas ventanas cambian su apariencia a un color claro y brilloso al hacer coincidir el RayCast. Al dar click con el botón izquierdo del mouse, el botón anima su desplazamiento y enciende la luz verde de la torreta.

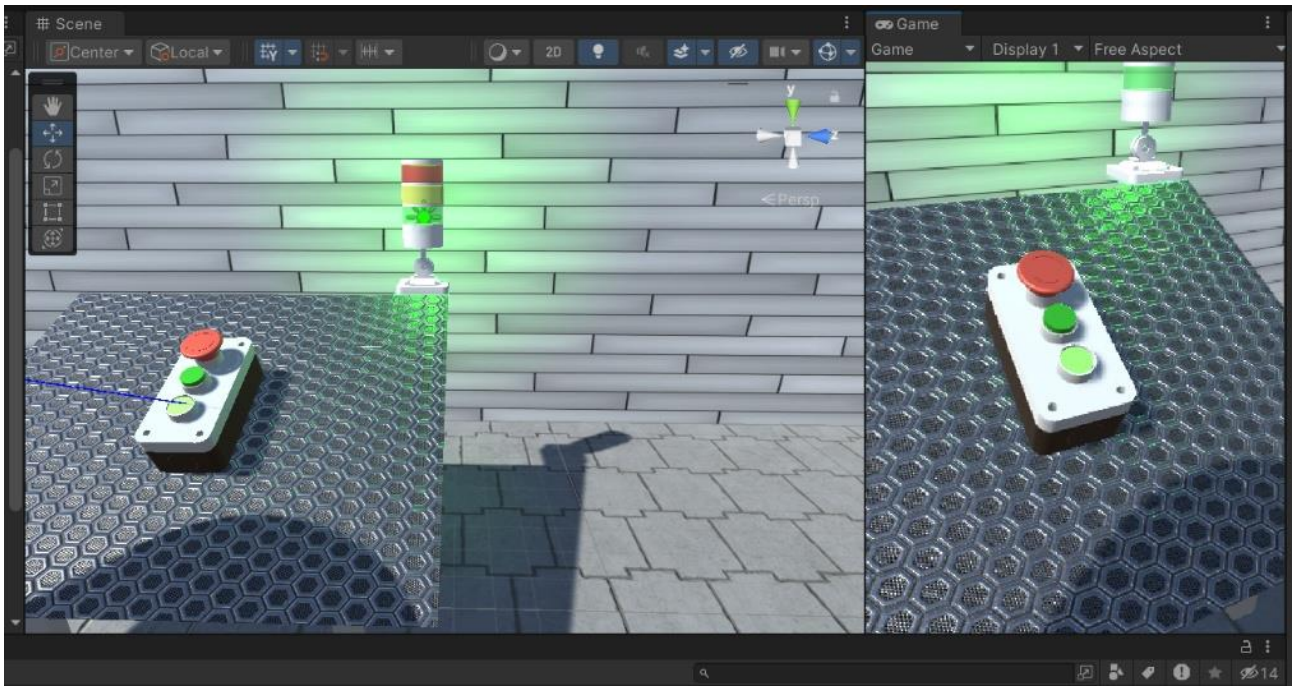


Figura 56 Botón pulsador activado

Por su parte, en la Figura 57 al hacer coincidir el RayCast con el botón enclavado, este último cambia a un color rojo más claro y brilloso. Si el botón es presionado una sola vez, se ejecutan la animación de movimiento del botón. Si no se vuelve a presionar el botón, este permanecerá en esta posición, similar a como sucede con los botones físicos, solo que, manualmente se desenclavan.

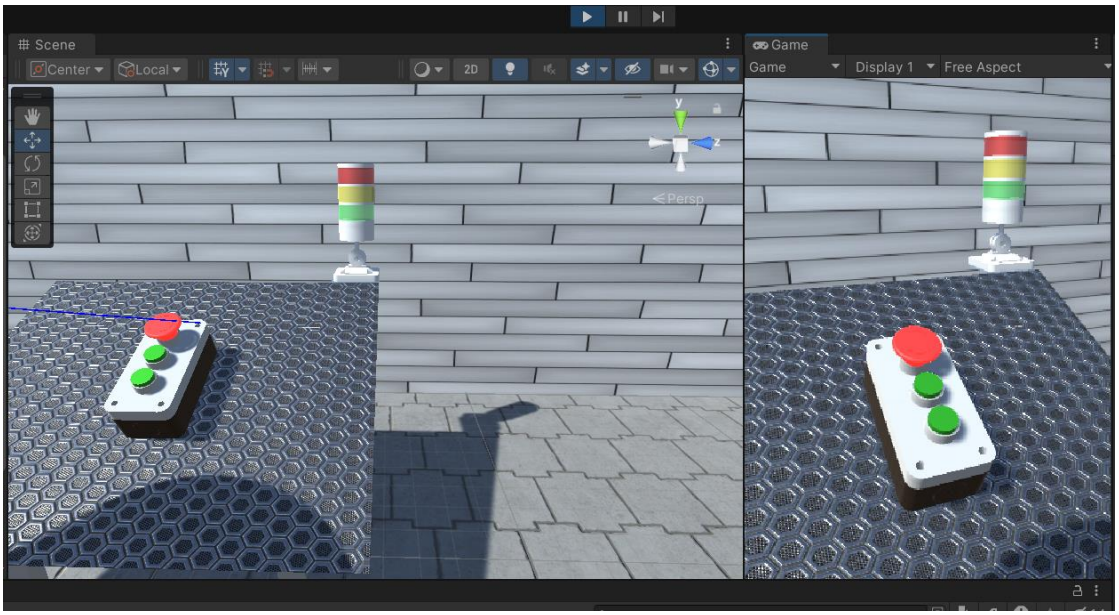


Figura 57 Botón enclavado activado

Activación y desactivación de un actuador neumático

El objetivo de este apartado es realizar la animación de la extensión y retroceso del vástago del pistón utilizando una botonera que se encuentra en la segunda mesa virtual; la extensión se activa con el botón pulsador inferior. El retroceso, a través del botón enclavado. Por tanto, es necesario generar tres nuevos scripts; el primero, se encargará de la animación del botón pulsador y al mismo tiempo activará la señal de extensión del vástago. El segundo, animará al botón enclavado y enviará la señal de retorno al vástago. El tercero, realizará la animación de extensión o retroceso del vástago.

- Movimiento positivo

En una nueva carpeta de Scripts (*Pistones*); generar un script llamado "*Activar_MOVPOS*". Para una mayor agilidad, copiar y pegar el código que se tiene de *BotonPulsadorB1*; a continuación, agregar la variable pública del tipo *MovimientoPositivo movimiento;* que referencia al script que se encargará de la animación del vástago. Crear una variable privada del tipo booleana llamada *presionado* e inicializarla en falso, Figura 58.

```
7       public float distancia = 1f;          // Cuánto se mueve el botón
8       public float distancia2 = 1f;
9       public float velocidad = 40f;           // Qué tan rápido se mueve
10
11      private Vector3 posicionInicial;
12      private Vector3 posicionPresionada;
13
14      public MovimientoPositivo movimiento;
15
16      private bool presionado = false;
```

Figura 58 Declaración de variables; MovimientoPositivo permite ligarse al vástago

Dado que se ha reutilizado el código para la animación del botón pulsador, no se cambiará la lógica para la animación del botón pulsador, solo se agregará un script que da la señal de referencia para realizar el movimiento de expulsión del vástago. Por tanto, la función *void Start* y *presionado,*

permanecen sin cambios. La función *PulsoPOS* almacena la referencia del evento, si el botón es "presionado" (click del mouse), se llama a los métodos "OnPulsadorPressed" y "OnPulsadorReleased" que se encuentran en el script *MovimientoPositivo*, Figura 59.

```
19      void Start()
20      {
21          posicionInicial = transform.position;
22          posicionPresionada = posicionInicial - new Vector3(distancia2, distancia, 0);
23      }
24
25      // Update is called once per frame
26      public void PulsoPos()
27      {
28          if (Input.GetButtonDown("Fire1"))
29          {
30              presionado = true;
31              movimiento.OnPulsadorPressed();
32          }
33
34          if (Input.GetButtonUp("Fire1"))
35          {
36              presionado = false;
37              movimiento.OnPulsadorReleased();
38          }
39
40          // Movimiento suave
41          if (presionado)
42          {
43              transform.position = Vector3.Lerp(transform.position, posicionPresionada, Time.deltaTime * velocidad);
44          }
45          else
46          {
47              transform.position = Vector3.Lerp(transform.position, posicionInicial, Time.deltaTime * velocidad);
48          }
49      }
```

Figura 59 Funciones void Start y PulsoPos para animar el botón pulsador de la segunda mesa virtual.

- Movimiento negativo

Reutilizando el código del botón enclavado (utilizado en el indicador luminoso) en un nuevo script llamado "Activar_MOVNEG". El funcionamiento del botón será el mismo, por ende, su programación no contendrá cambios. Añadir la variable pública del método *MovimientoPositivo* para referenciar al script de movimiento del vástago, Figura 60.

```
public float distancia = 1f;      // Movimiento en Y
public float distancia2 = 1f;     // Movimiento en X
public float velocidad = 10f;     // Velocidad del movimiento

private Vector3 posicionInicial;
private Vector3 posicionPresionada;
private bool enclavado = false;

public MovimientoPositivo movimiento;
```

Figura 60 Declaración de variables para activar el botón enclavado de la segunda mesa virtual.

En la Figura 61, la función EnclavarNeg, activa o desactiva el enclavamiento del botón; la modificación que distingue a este código es que en cada ocasión que se activa el evento (click izquierdo del mouse), el método "ToggleEnclavado" (ubicado en el script *MovimientoPositivo*) también es activado.

```
23      public void EnclavarNeg()
24      {
25
26          // Detectar clic con Fire1 y raycast sobre este objeto
27          if (Input.GetButtonDown("Fire1"))
28          {
29              enclavado = !enclavado;
30              movimiento.ToggleEnclavado();
31          }
32
33          // Movimiento según estado enclavado
34          if (enclavado)
35          {
36              transform.position = Vector3.Lerp(transform.position, posicionPresionada, Time.deltaTime * velocidad);
37          }
38          else
39          {
40              transform.position = Vector3.Lerp(transform.position, posicionInicial, Time.deltaTime * velocidad);
41          }
42      }
```

Figura 61 Función EnclavarNeg; para la animación del botón enclavado de la segunda mesa virtual

- Animación de desplazamiento del vástago

Teniendo las señales de activación (botón pulsador) y desactivación (botón enclavado), el script que se desarrollará a continuación contendrá la cinemática del actuador. Con anterioridad se mencionó un script llamado *MovimientoPositivo*; este script se ubicará también en la carpeta *Pistones*. En el editor, se colocarán seis variables, que continuación se mencionan y en la Figura 62 se muestran en el script:

- *positionA:* pública de tipo *Vector3;* almacena la posición actual del vástago.
- *positionB:* pública de tipo *Vector3;* almacena la posición objetivo (configurable en el inspector).
- *speed*: pública de tipo flotante; velocidad de desplazamiento del objeto tridimensional (vástago). Es configurable desde el inspector, pero se dejó como base 5f.
- *pulsadorPressed:* pública de tipo booleana; recibe el evento de activación o expulsión.
- *enclavadoActive:* pública de tipo booleana; recibe el evento de desactivación.
- *targetPosition: privada* de tipo *Vector3*; complemento para desplazar al vástago

```
7       public Vector3 positionA;       // Posición inicial (A)
8       public Vector3 positionB;       // Posición final (B)
9       public float speed = 5f;        // Velocidad del movimiento
10
11      public bool pulsadorPressed = false;   // Control desde UI o input
12      public bool enclavadoActive = false;   // Control desde UI o input
13
14      private Vector3 targetPosition;
```

Figura 62 Variables de trabajo en el script MovimientoPositivo

Como en los botones, la función *void Start,* almacena los parámetros iniciales de la posición del vástago, Figura 63.

```
17      void Start()
18      {
19          transform.position = positionA;     // Iniciar en A
20          targetPosition = positionA;
21      }
```

Figura 63 Parámetros de inicio / Función void Start

La función *Update* (ejecutada continuamente), realiza una comparación entre las interacciones de los métodos *pulsadorPressed* y *enclavadoActive*; en cualquiera de las ejecuciones, el vástago hará un desplazamiento suave de la posición A→B (extensión) o de B→A (retroceso), respectivamente (Figura 64).

```
23      void Update()
24      {
25          // Lógica de movimiento
26          if (pulsadorPressed)
27          {
28              targetPosition = positionB;
29          }
30          else if (enclavadoActive)
31          {
32              targetPosition = positionA;
33          }
34
35          // Movimiento suave hacia la posición objetivo
36          transform.position = Vector3.MoveTowards(transform.position, targetPosition, speed * Time.deltaTime);
37      }
```

Figura 64 Función Update

La parte final del script contiene los métodos *OnPulsadorPressed*, *OnPulsadorReleased* y *ToggleEnclavado*. Cuando el botón pulsador es presionado, la función *OnPulsadorPressed* establece la variable *pulsadorPressed* en verdadero, haciendo que el vástago se desplace de la posición base (A) a la posición objetivo (B). Para que la variable *pulsadorPressed* se restablezca, se utiliza el retorno del botón pulsador para volverla falsa que espere a recibir de nuevo el estado (función *OnPulsadorReleased*). Finalmente, si se desea regresar al vástago a la posición inicial, la función *ToggleEnclavado,* activa el estado de la variable *enclavadoActive* en verdadero. Al tratarse de un botón enclavado, al volver a interactuar con él para desactivarlo, se utiliza una negación para invertir el estado de la variable y reestablecerla. Todos los métodos anteriormente descritos se ubican en la Figura 65.

```
39      // Métodos para conectar con los botones de UI o eventos
40      public void OnPulsadorPressed()
41      {
42          pulsadorPressed = true;
43      }
44
45      public void OnPulsadorReleased()
46      {
47          pulsadorPressed = false;
48      }
49
50      public void ToggleEnclavado()
51      {
52          enclavadoActive = !enclavadoActive;
53      }
54  }
55
```

Figura 65 Estados para las variables pulsadorPressed y encalvadoActive

Al final, la carpeta *Pistones* debe contener los scripts:

- *Activar_MOVPOS*
- *Activar_MOVNEG*
- *MovimientoPositivo*

Como en la Figura 66.

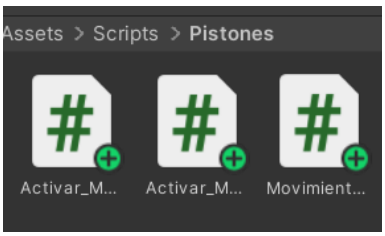


Figura 66 Vista a los scripts de la carpeta "Pistones"

V. Integración de una biblioteca de vínculos dinámicos

Unity ejecuta los códigos empleando las bibliotecas y archivos predeterminados del entorno. Sin embargo, cuando el usuario requiere establecer configuraciones adicionales o funciones específicas, como es el caso, es necesario incorporar bibliotecas externas. En esta práctica será necesario agregar una biblioteca de vínculos dinámicos (DLL) denominada *S7.Net, la cual* proporciona complementos y paquetes de datos necesarios para establecer una comunicación mediante el protocolo PROFINET. A continuación, se describen los pasos que se deben seguir para integrar esta biblioteca al proyecto.

Para esta práctica, el archivo DLL se obtiene desde el repositorio oficial de "GitHub", buscando el proyecto denominado "S7Net". Una vez localizado, se debe descargar el archivo DLL y colocarlo en la carpeta correspondiente del proyecto de Unity, destinada específicamente para las bibliotecas externas.

A continuación, se describen los pasos para realizar este procedimiento de manera adecuada. En la Figura 67 se muestra el archivo DLL descargado.

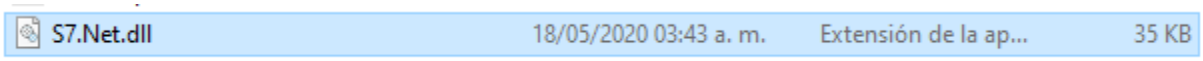


Figura 67 Archivo ".dll"

- Integración de una DLL en un proyecto de Unity:

Para esta práctica se trabaja la versión de Unity 2022.3.10f1. No olvidar que Unity recibe actualizaciones y mejoras de forma constante por lo que dichas actualizaciones pueden modificar la estructura de carpetas y el proceso de integración de archivos.

1. Cerrar el proyecto de Unity (en caso de que se esté editando). Localizar la carpeta donde se encuentra almacenado el proyecto de Unity. En la Figura 68 se muestra un ejemplo de la ruta de acceso al proyecto, la cual puede variar según la ubicación que el usuario haya definido al momento de su creación.

C:\Users\Alien1VR\Proyecto_Comunicacion

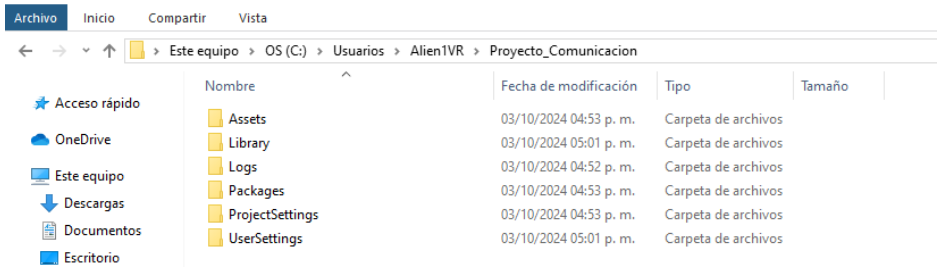


Figura 68 Ruta a la ubicación del proyecto de UNITY

2. Ubicar la carpeta "Assets" y dentro de esta, seleccionar la subcarpeta *Plugins*. En caso de no existir la subcarpeta *Plugins* (como en la Figura 69), crear la carpeta.

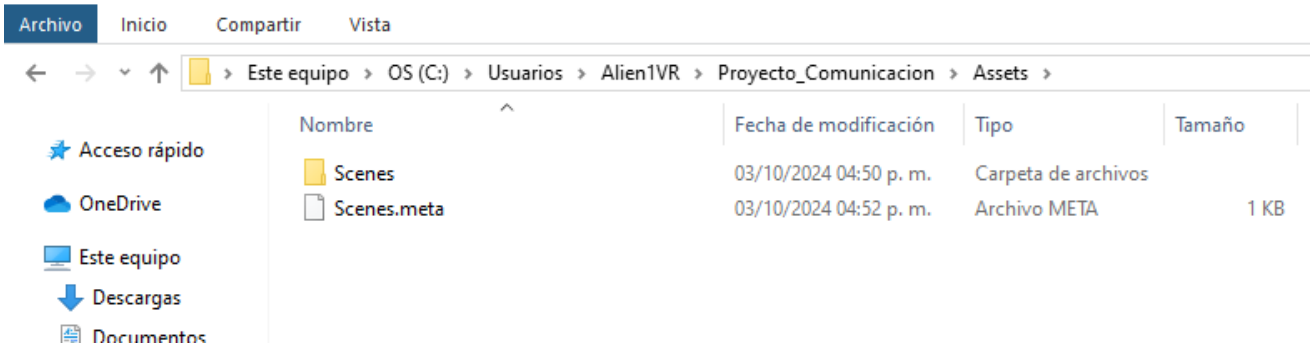


Figura 69 Interior de la carpeta Assets

Al generar la subcarpeta *Plugins*, Unity entenderá que se trata de referencias a bibliotecas externas que deben ser utilizadas en el proyecto. En la Figura 70 se muestra que se ha integrado la subcarpeta.

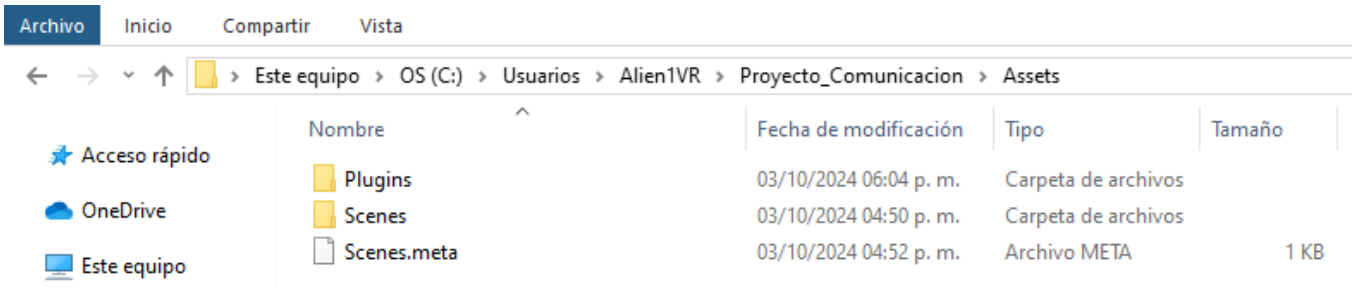


Figura 70 Creación de carpeta "Plugins"

3. Colocar el archivo DLL *S7.NET* dentro de la subcarpeta *Plugins* del proyecto. Esta acción permite que Unity genere las referencias necesarias para establecer la comunicación entre Unity – TIA Portal – PLC (Figura 71).

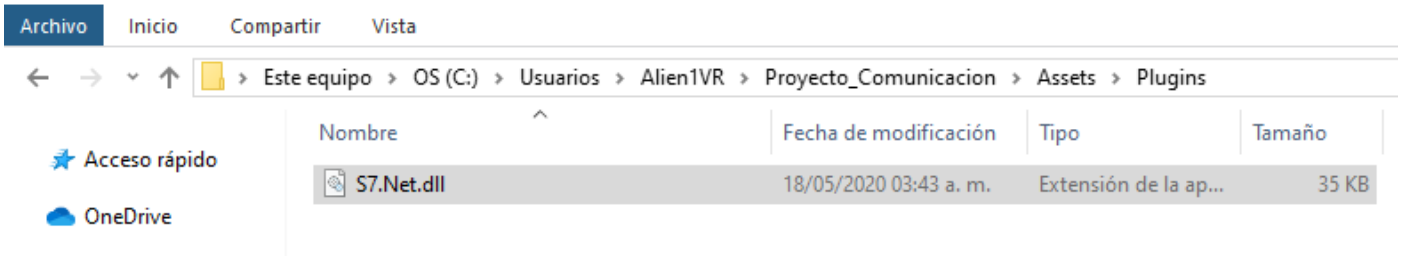


Imagen 71 Archivo .dll

Una vez realizada esta integración, el entorno de Unity podrá reconocer las clases y métodos de la biblioteca, por lo que sólo será necesario escribir la sintaxis correspondiente dentro de un nuevo script principal de comunicación que en el siguiente capítulo se detalla.

VI. Acondicionamiento de Unity

Cuando la biblioteca S7.NET esté integrada y el proyecto de Unity se encuentre abierto, el motor gráfico reconocerá automáticamente los archivos añadidos. A continuación, se debe crear un nuevo script con el nombre “PLC_Comunicacion”, en el cual se programará la estructura necesaria para solicitar los valores alojados en las direcciones del PLC. De forma simultánea, en este mismo script permitirá enviar datos hacia los bloques de datos (Data Blocks) configurados en TIA Portal, en caso de que alguno de los eventos de la escena de trabajo lo requiera. En el apartado siguiente se detallará el procedimiento y para el uso de los bloques de datos.

En el apartado de scripts, crear una nueva carpeta que se llame “Conexión”, dentro de ella generar el script “PLC_Comunicacion”, como se muestra en la Figura 72. Al abrir el editor de *PLC_Comunicacion,* agregar la referencia a la biblioteca recién agregada. En la Figura 73 son visibles las bibliotecas que por defecto tienen todos los scripts y además se añade la referencia “using S7.Net;”.

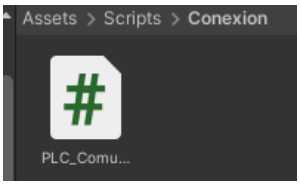


Figura 72 Carpeta Conexion

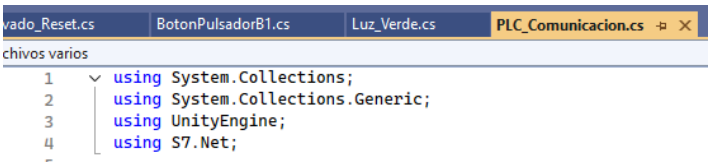


Figura 73 Referencia a S7.Net

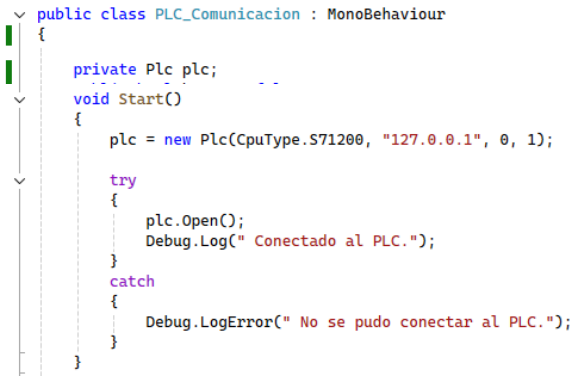


Figura 74 Declaración de PLC de trabajo

Declara una variable privada del tipo *PLC* denominada *plc*; previo a la función *void Start()*. Dentro del método público y dentro de la función *void Start*(), colocar la siguiente sintaxis:

→*plc = new Plc(CpuType.S71200, “127.0.0.1”, 0, 1)*

Con esta línea se declara una nueva variable del tipo “Plc”, indicando que la comunicación será con un CPU S7-1200 ubicado en la dirección 127.0.0.1. Los valores “0” y “1” corresponden al rack y slot donde se encuentra operando el PLC dentro del hardware asignado. A continuación, dentro del bloque *try,* escribir:

→plc.Open();
→Debug.Log(“ Conectado al PLC”);

Con estas líneas se establece la comunicación con el PLC y, al mismo tiempo, se envía un mensaje a la consola de Unity indicando que la conexión fue exitosa. En caso contrario, se utiliza el bloque *catch* para capturar el error y el siguiente mensaje:

→Debug.LogError(“No se pudo conectar al PLC. ”);

Con esta estructura se garantiza que el sistema identifique si hubo fallas al intentar establecer la comunicación y muestre una notificación clara al usuario. En la figura 74 se observan las líneas de código que se describieron.

La configuración mostrada está basada directamente en la documentación oficial de la biblioteca S7.Net, disponible en el mismo repositorio de donde se obtuvo el archivo DLL.

Si los pasos anteriores se han realizado correctamente, la comunicación entre Unity y TIA Portal estará establecida. En los siguientes capítulos se configurarán las variables necesarias para la lectura y escritura de datos, con el fin de completar la integración del sistema.

VII. Acondicionamiento del PLC

En la mayoría de los procesos industriales automatizados, es sustancial contar con un elemento capaz de dirigir y controlar la maquinaria responsable de ejecutar la producción. Dicho elemento es el Controlador de Lógica Programable (PLC, por sus siglas en inglés), el cual realiza esta función mediante el procesamiento de datos que recibe desde sensores y ejecuta las órdenes en los actuadores que estén en el sistema. Los PLC se pueden programar utilizando uno o varios lenguajes estandarizados, a través de los cuales interpreta las señales de entrada, ejecuta comparaciones y operaciones lógicas, y determina la activación o desactivación de los actuadores correspondientes.

Para este manual, se utiliza el software TIA Portal, en su versión 15.1 como entorno de programación. TIA Portal constituye la interfaz principal para los PLC disponibles en el Laboratorio de Automatización Industrial, y permite crear, simular y poner en marcha los programas de control que se desarrollan en los siguientes apartados.

Al arrancar TIA Portal V15.1 aparece una interfaz como en la Figura 75; crear un nuevo proyecto sin olvidar cambiar la ubicación del archivo a la misma carpeta donde se contiene el proyecto de UNITY.

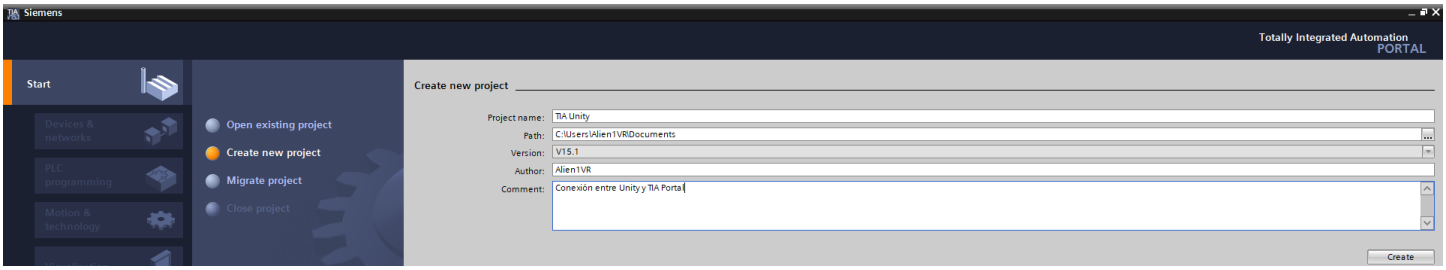


Figura 75 TIA Portal -> Create new project -> Colocar un nombre y elegir ubicación -> Create

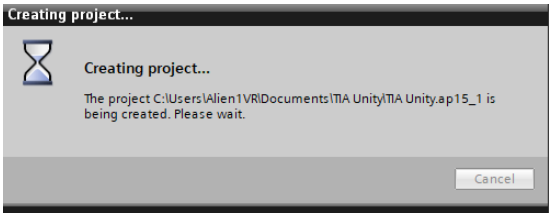


Figura 76 Ventana de carga de un nuevo proyecto

TIA Portal tardará unos segundos en generar el archivo del proyecto (Figura 76); al término del proceso, la interfaz cambiará. Aquí es posible colocar el modelo exacto del CPU a trabajar y para ello, seleccionar la pestaña “Devices & networks” y elegir la opción “Add new device” (Figura 77).

Casi de forma inmediata, una ventana emergente aparece. La ventana contiene todos los dispositivos disponibles para trabajar. Seleccionar la pestaña de "Controllers" y buscar el CPU: S7-1200, con CPU 1215C AC/ DC/ RLY; este dato se obtiene directamente del modelo del PLC de trabajo. Este proceso se aprecia en la Figura 78.

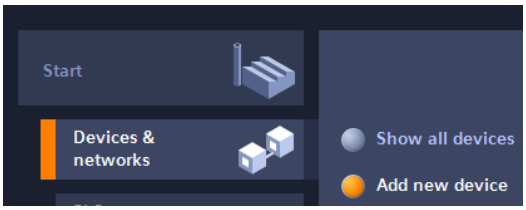


Figura 77 Devices & Networks -> Add new device

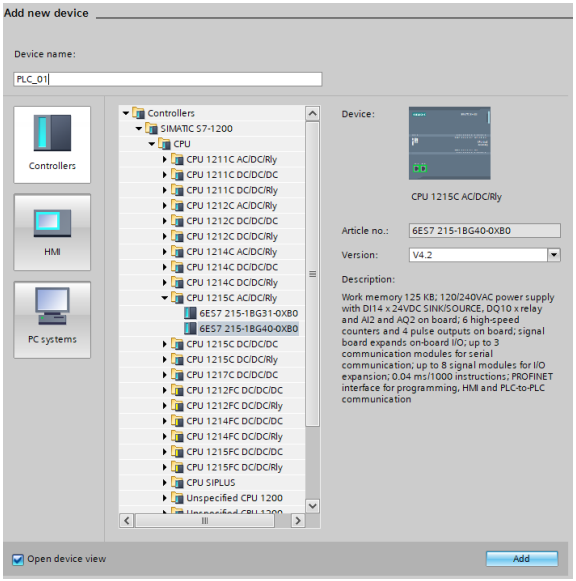


Figura 78 Device: PLC_01 -> Controllers -> SIMATIC S7-1200 -> CPU -> CPU 1215C AC/DC/RLY -> 6ES7 215-18G40-0XB0

Ya seleccionado correctamente el modelo, renombrar como PLC_01; dar click al botón "Add", ver Figura 78. Ahora, TIA Portal comenzará a cargar los datos de configuración de hardware, este proceso tardará unos segundos.

La interfaz está lista si se ve como en la Figura 79.

El siguiente paso es añadir el módulo de expansión que el equipo físico integra, de no integrar el módulo, el PLC físico entrará en modo falla por incompatibilidad de hardware. Del lado derecho de la Figura 79, en la pestaña de *Hardware catalog* (Figura 80), ubicar la carpeta "DI/DQ"; buscar la subcarpeta *DI 16x24VDC/DQ 16xRelay,* elegir el modelo "6ES7 223-1PL32-0XB0".

Un último paso de las configuraciones del CPU, es colocar la dirección IP de trabajo del equipo, ya que, más adelante será de suma importancia.

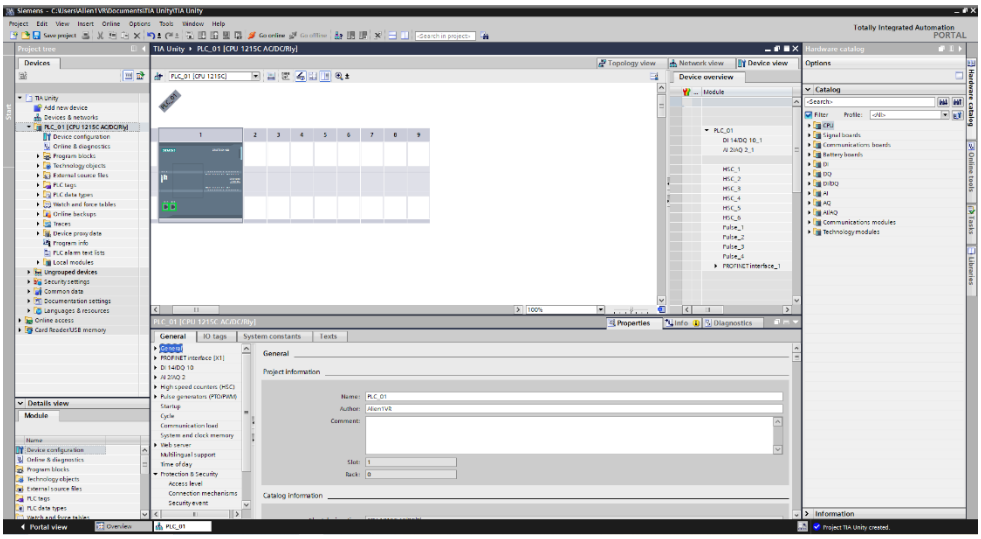


Figura 79 Interfaz inicial

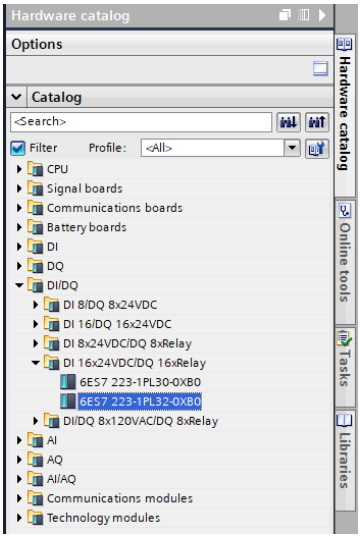


Figura 80 Hardware -> DI/DQ ->DI 16x24VDC/DQ 16XRELAY -> 3ES7 223-1PL32-0XB0

Dar click en el CPU y seleccionar las pestañas *Properties* y *General,* ubicadas en el inspector (ventana inferior a la principal), al igual que en la Figura 81. A continuación, buscar la pestaña *PROFINET interface [x1];* cambiar la dirección IP que tiene por defecto a la dirección: 192.168.105.121

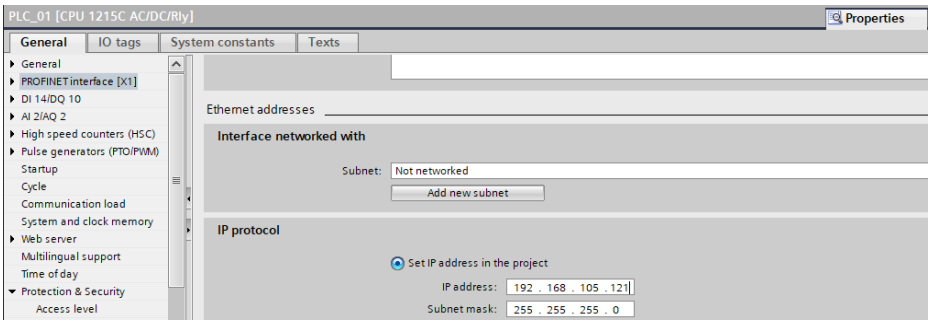


Figura 81 CPU -> PROFINET interface [x1] -> IP protocol -> IP address: 192.168.105.121

Si todos los pasos anteriores se realizaron con éxito, el proyecto debe ser igual a la Figura 82.

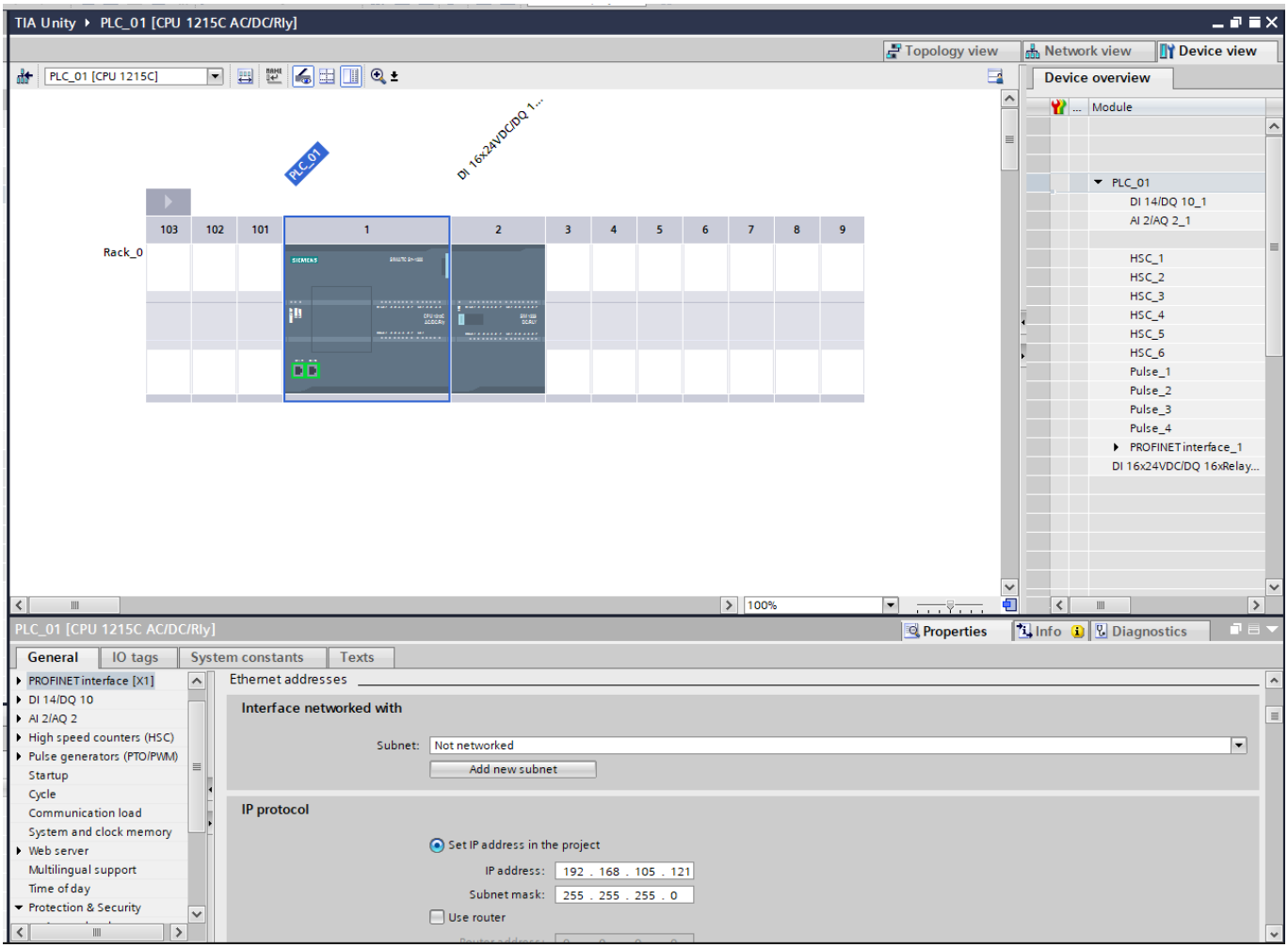


Figura 82 Vista general del proyecto

Configuración de TIA Portal para el envío y recepción de datos.

En su configuración básica, el PLC procesa las señales de entrada provenientes de dispositivos de campo físicos, como sensores o botones, los cuales se encuentran cableados directamente al módulo de entradas digitales (si son señales discretas). Sin embargo, el objetivo de este manual es que el PLC también responda ante señales generadas de forma virtual, las cuales pueden llegar a sustituir de

manera funcional a las conexiones físicas. Para lograrlo, se aprovecha el protocolo de comunicación PROFINET, integrado de forma nativa en el controlador.

Mediante la comunicación PROFINET, es posible enviar y recibir datos hacia un segmento específico de la memoria del PLC, donde la información se almacena y procesa como variables internas del sistema, permitiendo la interacción entre el entorno físico y el virtual.

El segmento de memoria donde se almacenará la información será un bloque de datos (*Data Block o DB).* La transferencia de información entre el entorno virtual y el PLC se realizará exclusivamente mediante estos bloques, que actúan como pasarelas de comunicación. Para una correcta identificación y gestión de variables, se deben crear dos bloques de datos: uno destinado al envío de información y otro a la recepción.

De acuerdo con la documentación técnica de Siemens, es importante que estos bloques no estén optimizados, ya que la optimización modifica la estructura interna del DB y puede impedir el acceso correcto desde aplicaciones externas como Unity. [10]

En la Figura 83, se observa el procedimiento para crear un bloque de datos. Dentro de la carpeta de *Program blocks,* se da doble click en la opción “Add new block”; en la ventana emergente, seleccionar el ícono que tiene por nombre *Data block;* colocar como nombre “DB_Enviar_Datos”.

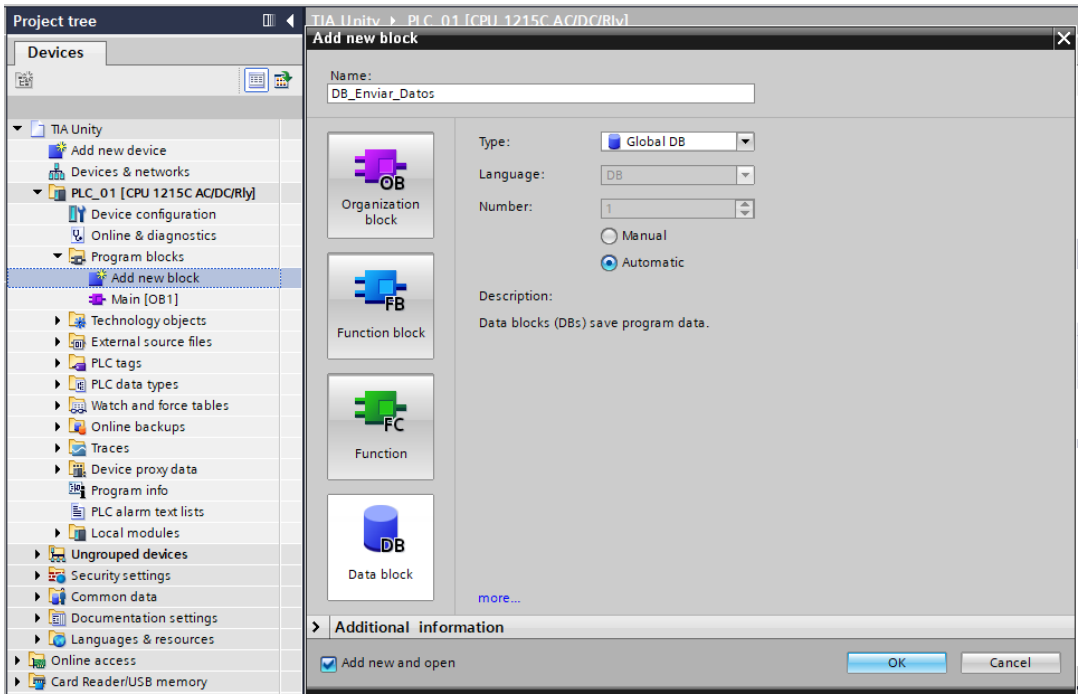


Figura 83 Program blocks -> DB -> DB_Enviar_Datos

Al finalizar, la ventana principal cambiará a una tabla, ver Figura 84.

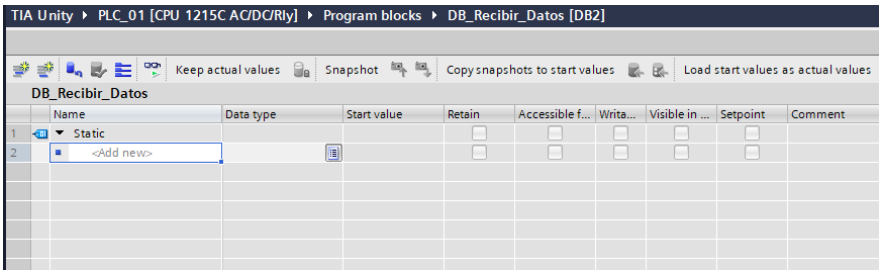


Figura 84 DB_Enviar_Datos

Realizar el mismo procedimiento para crear “DB_Recibir_Datos”.

Cuando ambos bloques de datos han sido creados, se procede a quitar la optimización de cada uno; seleccionar cualquiera de los bloques (ubicado en la carpeta *Program blocks*), dar click derecho e ir a las propiedades, como en la Figura 85. Una ventana emergente aparecerá (Figura 86); buscar la pestaña de “Attributes” y desmarcar la opción “Optimized blocks access”; confirmar en la pestaña de advertencia.

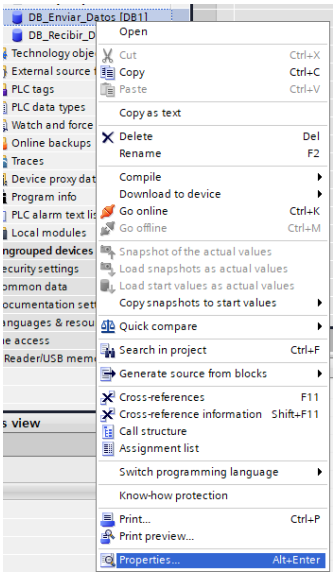


Figura 85 Program blocks -> DB_Enviar_Datos -> Properties

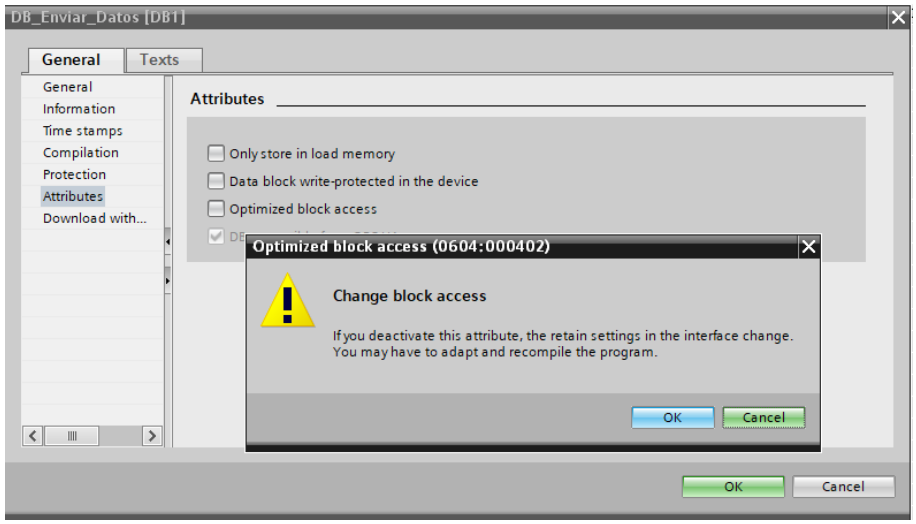


Figura 86 General -> Attributes -> (desactivar) Optimatized block access -> Ok

En ocasiones TIA Portal llega a modificar la configuración de protección del PLC: En la pestaña “Protection & Security”, Acces level debe ser “Full Access (no protection)”, ver Figura 87.

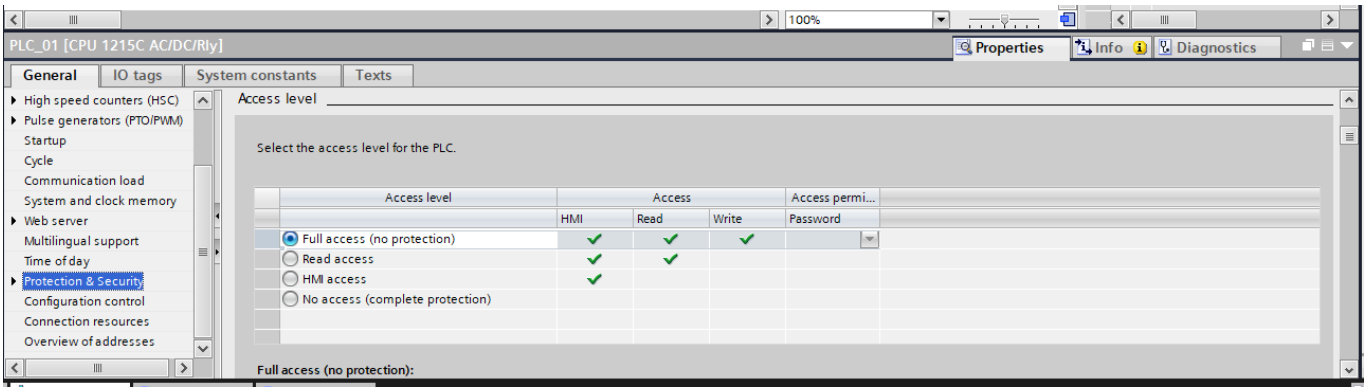


Figura 87 CPU -> General -> Protection & Security -> Access level -> Full access (no protection)

Sobre la misma pestaña, desplazándose hacia la parte inferior, en el apartado “Connection mechanism” habilitar la casilla “Permit Access with PUT/GET communication from remote partner” (Figura 88).

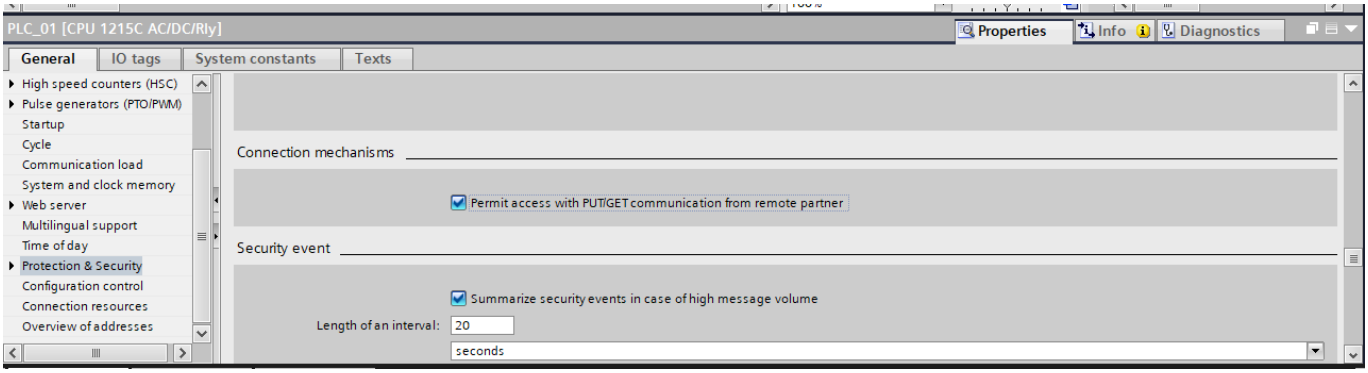


Figura 88 CPU -> General -> Protection & Security -> Connection mechanism -> Permit Access with PUT/GET communication from remote partner

Direccionamiento de entradas y salidas

Como se había mencionado, el Laboratorio de Automatización Industrial tiene una infraestructura definida, por tanto, los dispositivos de entrada y salida (sensores, botones, luces y pistones), ya cuentan con un direccionamiento específico. En la Tabla 1 y en la Tabla 2, se encuentran los direccionamientos correspondientes. Este direccionamiento está presente en las 10 estaciones de trabajo:

En la Tabla se encuentran las señales conectadas al CPU del PLC.

Tabla 1 Direccionamiento de PLC S7-1200 LAI

Etiqueta	*Identificador*	*Dirección*	*Descripción*
a0	I	%I0.0	Sensor Reed a0
a1	I	%I0.1	Sensor Reed a1
b0	I	%I0.2	Sensor Reed b0
b1	I	%I0.3	Sensor Reed b1
c0	I	%I0.4	Sensor Reed c0
c1	I	%I0.5	Sensor Reed c1
Botón1	I	%I1.0	Botón pulsador NA
Botón2	I	%I1.1	Botón pulsador NA
Botón3	I	%I1.2	Botón selector NA
Botón4	I	%I1.3	Botón enclavado NA
1 EV-14	Q	%Q0.0	A+
1 EV-12	Q	%Q0.1	A-
2 EV-14	Q	%Q0.2	B+
3 EV-14	Q	%Q0.3	C+

Para el caso de la Tabla 2, las señales corresponden al módulo de expansión del PLC. Este módulo es habilitado gracias a una caja con bornes para la conexión de cables con terminal de banana.

Tabla 2 Direccionamiento del módulo de expansión

Etiqueta	*Identificador*	*Dirección*	*Descripción*
BSecundario1	I	%I2.0	Botón selector NA
BSecundario2	I	%I2.1	Botón pulsador NA
BSecundario3	I	%I2.2	Botón pulsador NA
BSecundario4	I	%I2.3	Botón pulsador NA
BSecundario5	I	%I2.4	Botón enclavado NA
Luz_Verde	Q	%Q2.0	Indicador
Luz_Ambar	Q	%Q2.1	Indicador
Luz_Roja	Q	%Q2.2	Indicador

Cuando se instala la caja con bornes (Figura 89), se puede conectar una botonera iluminada (botones con indicadores iluminados, ver Figura 90) y una botonera externa (con cinco botones en diferentes configuraciones, ver Figura 91).

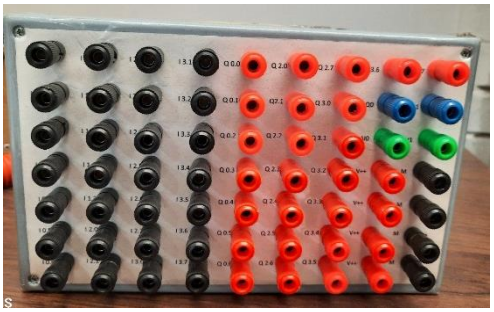


Figura 89 Caja con bornes o caja de conexiones

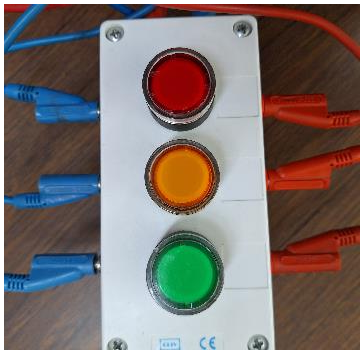


Figura 90 Botonera iluminada

Figura 91 Botonera externa

VIII. Ejercicio 1: Encendido y apagado de un indicador luminoso

El ejercicio se desarrolla en tres etapas. Primera etapa: programación en TIA Portal; segunda etapa: modificación de los scripts que interactúan para el encendido y apagado del indicador luminoso (botones y luz); tercera etapa: enlace de Unity y TIA Portal.

Etapa 1: Programación en TIA Portal.

Dentro del archivo de TIA Portal que se generó, abrir la *Default tag table.* Colocar las variables que se muestran en la Figura 92 y asignar las direcciones de acuerdo a la *Tabla 2*.

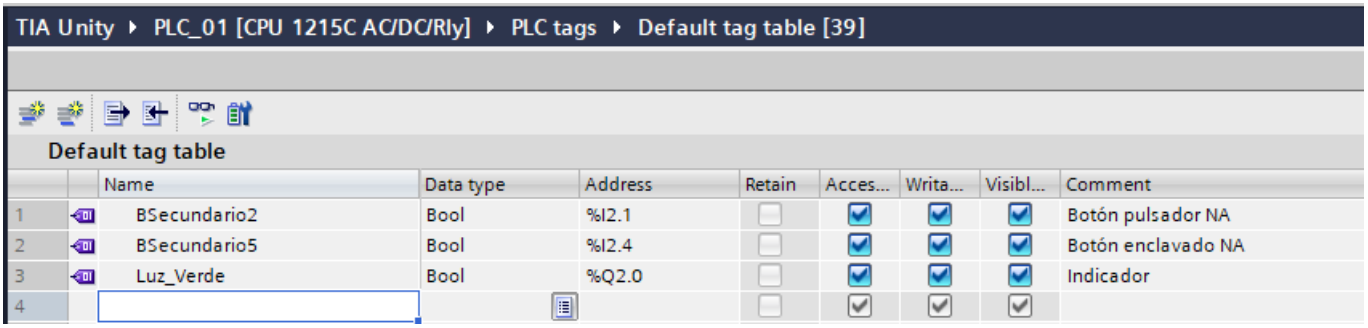


Figura 92 PLC_01 -> PLC tags -> Default tag table

De la Figura 92 se tiene:

- *BSecundario2*: Botón pulsador (botonera externa) para arranque del sistema
- *Bsecundario5*: Botón enclavado (botonera externa) para detener el proceso
- *Luz_Verde*: Indicador luminoso verde (botonera iluminada) como actuador o salida a controlar.

Ahora es importante colocar las señales que se enviará a Unity. El bloque de datos que se creó para el envío de señales de activación de los botones (Start_Fisico y Stop_Fisico) y del indicador luminoso (*Luz_Verde*). En la Figura 93 se observa el DB “Enviar datos”.

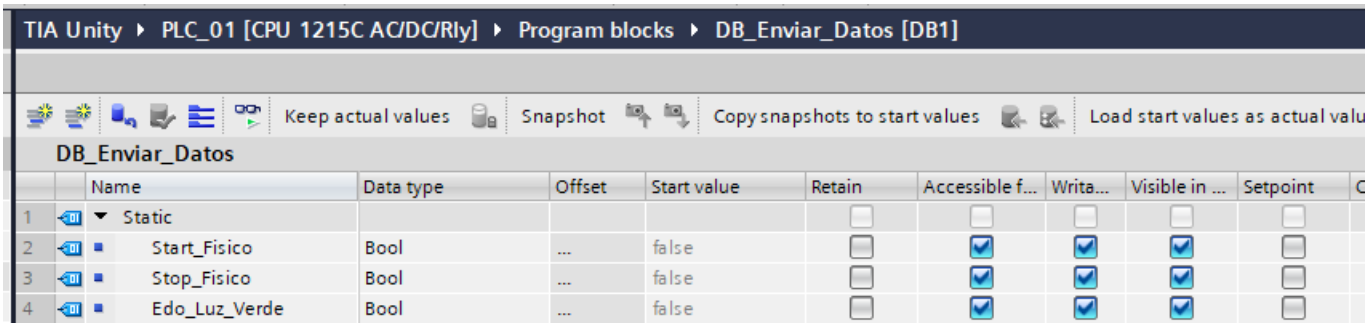


Figura 93 Program blocks -> DB_Enviar_Datos

A continuación, escribir las variables que se recibirán del ambiente virtual. Las variables serán de arranque (*Start_Virtual_M1*) y de paro (*Stop_Virtual_M1*). Las variables se colocarán en el DB “Recibir datos”, cómo en la Figura 94.

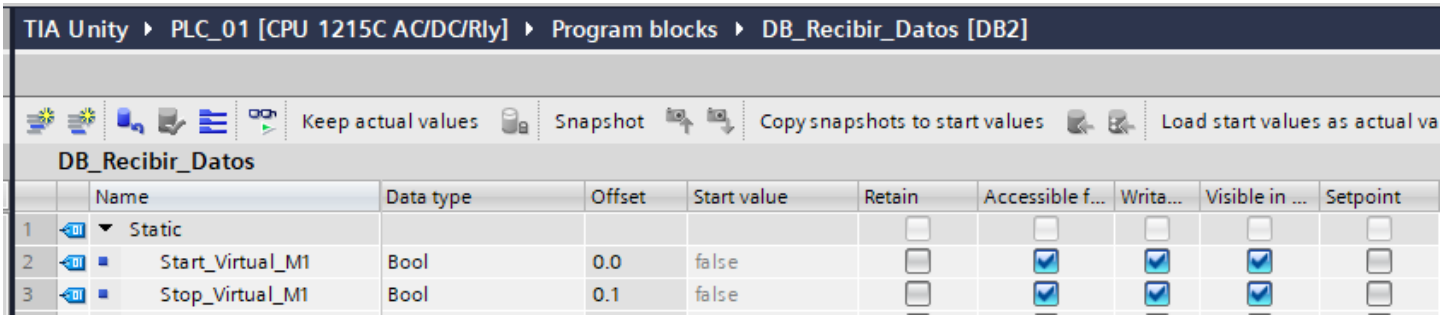


Figura 94 Program blocks -> DB_Recibir_Datos

Al terminar de ingresar los datos, compilar el programa para que los bloques de datos sean cargados a la memoria del proyecto y se puedan ocupar posteriormente. El botón se encuentra en la parte superior izquierda, en la Figura 95.

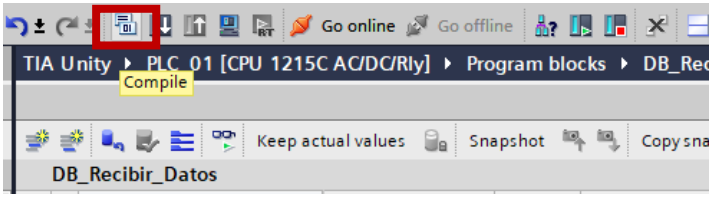


Figura 95 Botón "Compile"

Finalmente, colocar la lógica de arranque y paro con autoenclavamiento. Debido a que el actuador o se activará de forma física o de forma virtual, colocar un peldaño paralelo para la activación del sistema. Para detener la ejecución, colocar al paro en serie. Colocar una salida en paralelo para la señal *Edo_Luz_Verde,* que se encuentra dentro del *DB_Enviar_Datos*. Enviar las señales BSecundario2 y BSecundario5 al entorno virtual. El proyecto se debe observar como en la Figura 96.

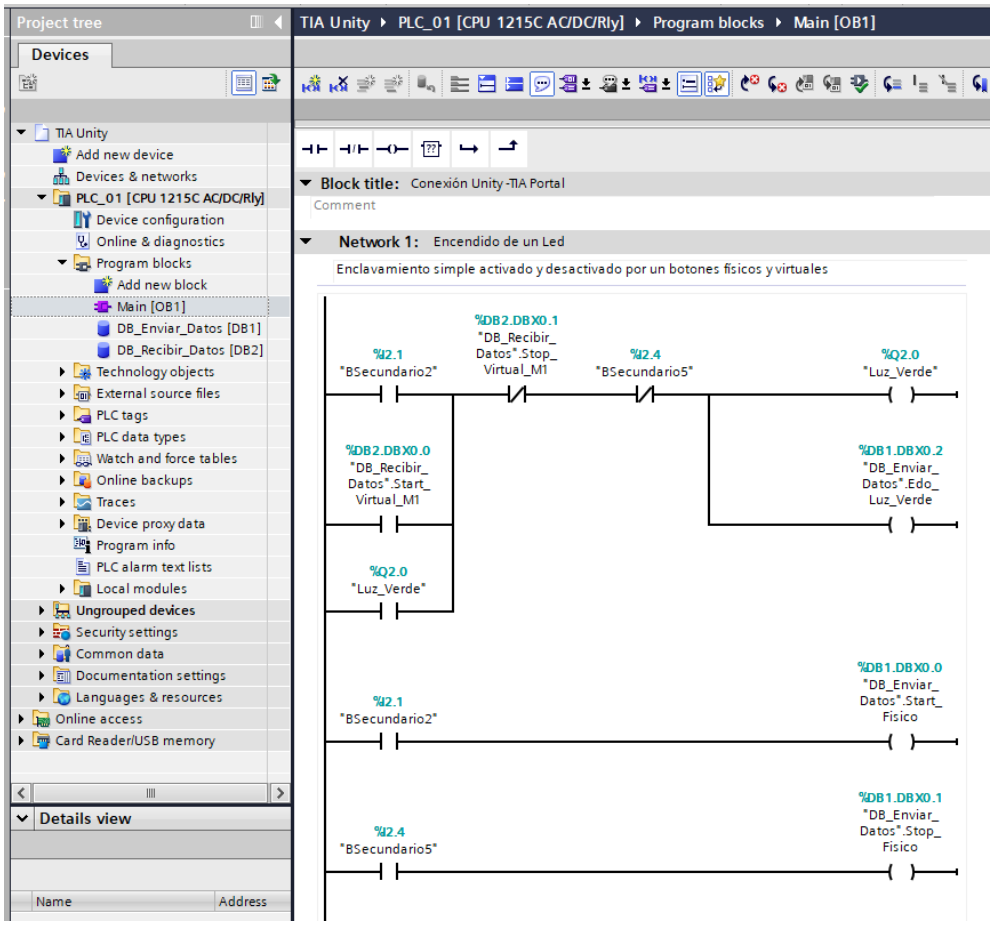


Figura 96 Programación para activar y desactivar un indicador luminoso

Etapa 2: Modificación de scripts

Antes de configurar el script *Luz_Verde*, abrir el script *PLC_Comunicacion* (creado en el apartado *Acondicionamiento de Unity*); dentro del método público y antes de la variable privada *PLC*, declarar la siguiente variable:

- public Luz_Verde luzVerd; →Referencia al script *Luz_Verde*

Posterior a la variable *PLC;*

- public bool bp2M1 = false; →Referencia al estado del botón pulsador que se encuentra en la parte inferior de la botonera virtual, inicia en falso.
- private bool enclavado = false; →Referencia al estado del botón enclavado de la botonera, inicia en falso.

Enclavado_Reset.cs | BotonPulsadorB1.cs | Luz_Verde.cs | PLC_Comunicacion.cs

Archivos varios

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using S7.Net;
5
6   public class PLC_Comunicacion : MonoBehaviour
7   {
8
9       public Luz_Verde luzVerd;   // Encendido de led Verde
10
11      private Plc plc;
12      public bool bp2M1 = false;
13      private bool enclavado = false;
14
15      void Start()
16      {
17          plc = new Plc(CpuType.S71200, "127.0.0.1", 0, 1);
18
19          try
20          {
21              plc.Open();
22              Debug.Log(" Conectado al PLC.");
23          }
24          catch
25          {
26              Debug.LogError(" No se pudo conectar al PLC.");
27          }
28      }
29
30      void Update()
31      {
32          if (plc != null && plc.IsConnected)
33          {
34              try
35              {
36                  // Leer señal desde el PLC (ej: DB1.DBX0.0)
37                  bool StartVirtual = (bool)plc.Read("DB1.DBX0.0");
38                  bool StopVirtual = (bool)plc.Read("DB1.DBX0.1");
39
40                  if (StartVirtual)
41                  {
42                      EncenderIV();
43                  }
44
45                  if (StopVirtual)
46                  {
47                      ApagarIV();
48                  }
```

Figura 97 PLC_Comunicacion, Parte 1

En la Figura 97 se aprecia parte del código.

Después del bloque *void Start(),* colocar el bloque *void Update();* este bloque se ejecutará cíclicamente. Dentro del bloque, colocar:

-bloque *if,* verificando que el sistema esté enlazado (plc != null && plc.IsConnected)
-bloque *try* realizará una actualización constante de las variables que contenga.

Dentro del bloque *try* se colocarán las direcciones de las variables que se alojan en los bloques de memoria de TIA Portal y se guardarán en variables alternas para que Unity las opere, la sintaxis es la siguiente:
bool StartVirtual = (bool)plc.Read("DB1.DBX0.0");
→bool StartVirtual: variable de referencia en Unity
→(bool)plc.Read(): tipo de dato (en TIA Portal) y función de lectura (S7.Net)
→ DB1.DBX0.0: Referencia a la dirección del bloque de memoria en TIA Portal siendo que:
- DB1: Data Block #1
-DBX: Byte de memoria
-0.0: Dirección del Bit dentro del Byte

La sintaxis también se aplica para la variable StopVirtual.

Posteriormente a la lectura de las variables provenientes del PLC, definir las condiciones que permitirán activar las funciones asignadas a cada botón.

- Si *StartVirtual* es verdadera (botón presionado), se ejecuta el método EncenderIV().

- Si *StopVirtual* es verdadera (botón presionado), se ejecuta el método ApagarIV().

En la Figura 98 se presenta la continuación del código correspondiente al script *PLC_Comunicacion.*

Opcionalmente, se pueden enviar mensajes a la consola de Unity para visualizar el estado recibido de los botones y facilitar el diagnóstico de comunicación.

```
48                  }
49
50                  Debug.Log("Señal 1 desde PLC: " + StartVirtual);
51                  Debug.Log("Señal 2 desde PLC: " + StopVirtual);
52              }
53              catch
54              {
55                  Debug.LogWarning(" Fallo al leer del PLC.");
56              }
57          }
58      }
59
60      void OnApplicationQuit()
61      {
62          if (plc != null && plc.IsConnected)
63          {
64              plc.Close();
65          }
66      }
67
68      public void EncenderIV()
69      {
70          luzVerd.EncenderLuzPLC();
71      }
72
73      public void ApagarIV()
74      {
75          luzVerd.ApagarLuzPLC();
76      }
77
78      public void EscribirBP2M1() //Señal virtual de paro
79      {
80          bp2M1 = !bp2M1;
81          plc.Write("DB2.DBX0.0", bp2M1);
82      }
83
84      public void EscribirBEM1() //Señal virtual de paro
85      {
86          enclavado = !enclavado;
87          plc.Write("DB2.DBX0.1", enclavado);    //Paro virtual
88      }
89
90  }
91
```

Figura 98 PLC_Comunicacion, Parte 2

Continuando con el código, si en algún momento el bloque *try* deja de recibir datos del PLC, el bloque *catch* desplegará un mensaje en la consola indicando “Fallo al leer del PLC.”

Como medida de seguridad, se escribe el bloque “OnApplicationQuit()”; al dejar de ejecutar la escena de trabajo, Unity detendrá la comunicación.

Bloque EncenderIV(): llamada al script *Luz_verde* para activar el indicador luminoso (*EncenderLuzPLC()*).

Bloque ApagarIV(): llamada al script *Luz_verde* para desactivar el indicador luminoso (*ApagarLuzPLC()*).

Finalmente, los estados de los botones virtuales deberán ser enviados a los bloques de datos de TIA Portal, por lo tanto:

Bloque EscribirBP2M1(): al entrar a este bloque, se niega el estado del botón (bp2M1) para informar que se ha registrado un cambio en su estado y se coloca el siguiente código: “plc.Write(“DB2.DBX0.0”, bp2M1);”

- Plc.Write: método de escritura (S7.Net)
- DB2: Data Block #2
- DBX: Byte de memoria
- 0.0: Dirección del Bit dentro del Byte
- Bp2M1: contiene el valor del estado de la variable (verdadero o falso)

La sintaxis también se aplica para el bloque EscribirBEM1(); botón enclavado.

Modificación al script Luz_Verde

Abrir el script Luz_Verde. Agregar dos nuevas variables:

→ private bool EstadoVirtual = false; variable de estado para activación virtual.

→ private bool EstadoFisico = false; variable de estado para activación física.

Dentro del bloque *EncenderLuz(),* eliminar la instrucción previa y colocar el siguiente código:

→ EstadoVirtual = true; La luz se activó por el botón virtual.

```
Enclavado_Reset.cs | BotonPulsadorB1.cs | Luz_Verde.cs | PLC_Comunicacion.cs
Archivos varios
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Luz_Verde : MonoBehaviour
6  {
7
8      public GameObject lVerde;
9      public bool luz;
10
11     private bool EstadoVirtual = false;
12     private bool EstadoFisico = false;
13
14     public void EncenderLuz()
15     {
16         EstadoVirtual = true;
17         EstadoLuz();
18     }
19
20     public void ApagarLuz()
21     {
22         EstadoVirtual = false;
23         EstadoLuz();
24     }
25
26     public void EncenderLuzPLC()
27     {
28         EstadoFisico = true;
29         EstadoLuz();
30     }
31
32     public void ApagarLuzPLC()
33     {
34         EstadoFisico = false;
35         EstadoLuz();
36     }
37
38     public void EstadoLuz()
39     {
40         if (EstadoVirtual == true || EstadoFisico == true)
41         {
42             lVerde.SetActive(true);
43         }
44
45         else if (EstadoVirtual == false || EstadoFisico == false)
46         {
47             lVerde.SetActive(false);
48         }
49     }
50 }
51
```

Figura 99 Script Luz_Verde modificado

→ EstadoLuz(); Ejecutar el método.

Dentro del bloque *ApagarLuz(),* eliminar la instrucción previa y colocar el siguiente código:

→ EstadoVirtual = false; La luz se desactivó por el botón virtual.

→ EstadoLuz(); Ejecutar el método.

Crear el bloque EncenderLuzPLC() y colocar:

→ EstadoFisico = true; La luz se activó desde TIA Portal.

→ EstadoLuz(); Ejecutar el método.

Crear el bloque ApagarLuzPLC() y colocar:

→ EstadoFisico = false; La luz se desactivó desde TIA Portal.

→ EstadoLuz(); Ejecutar el método.

Finalmente, crear el bloque *EstadoLuz(),* dentro de él, colocar los siguientes casos:

Si alguna de las variables *EstadoVirtual* o *EstadoFisico* son verdaderas, activar luz verde (SetActive = true).

Si alguna de las variables *EstadoVirtual* p *EstadoFisico* son falsas, desactivar luz verde (SetActive = false).

La Figura 99 es una captura completa al script Luz_Verde.

Modificación al script del botón pulsador

Al activar el evento del botón pulsador, el estado debe ser enviado al script *PLC_Comunicacion*, por tanto, lo primero que se necesita hacer es generar la referencia entre scripts. Agregar la siguiente línea de código antes del bloque *void Start(),* como se muestra en la Figura 100:

→ public PLC_Comunicacion Comunicación; referencia al script y asociación de una variable.

```
...do_Reset.cs | BotonPulsadorB1.cs | Luz_Verde.cs | PLC_Comunicacion.cs
...hivos varios
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class BotonPulsadorB1 : MonoBehaviour
6  {
7      public float distancia = 1f;          // Cuánto se mueve el botón
8      public float distancia2 = 1f;
9      public float velocidad = 40f;           // Qué tan rápido se mueve
10
11     private Vector3 posicionInicial;
12     private Vector3 posicionPresionada;
13
14     public PLC_Comunicacion Comunicacion;
15
16     private bool presionado = false;
17
18     // Start is called before the first frame update
19     void Start()
```

Figura 100 Referencia al script PLC_Comunicacion

```
24
25      // Update is called once per frame
26      public void Pulso()
27      {
28          if (Input.GetButtonDown("Fire1"))
29          {
30              presionado = true;
31              Comunicacion.EscribirBP2M1();
32          }
33
34          if (Input.GetButtonUp("Fire1"))
35          {
36              presionado = false;
37              Comunicacion.EscribirBP2M1();
38          }
```

Figura 101 Llamada a función EscribirBP2M1 contenida en el script PLC_Comunicacion

Dentro del bloque Pulso, cuando se interactúa con el mouse, referenciar al método *EscribirBP2M1* de la variable comunicación, cómo se muestra en la Figura 101.

No olvidar que, en el Anexo de este manual, se encuentran los códigos de los scripts completos.

Modificación al script del botón enclavado

Las modificaciones a este script son muy similares a las que se realizaron en el botón pulsador.

→ Referencia al script PLC_Comunicacion: public PLC_Comunicacion CComunicacion; Figura 102.

→ Dentro del bloque Enclavar, al interactuar con el mouse, referenciar al método *EscribirBEM1* de la variable comunicación, cómo se muestra en la Figura 103.

Enclavado_Reset.cs | BotonPulsadorB1.cs | Luz_Verde.cs | PLC_Comunicacion.cs

Archivos varios

```
4
5       public class Enclavado_Reset : MonoBehaviour
6       {
7           public float distancia = 1f;      // Movimiento en Y
8           public float distancia2 = 1f;     // Movimiento en X
9           public float velocidad = 10f;     // Velocidad del movi
10
11          private Vector3 posicionInicial;
12          private Vector3 posicionPresionada;
13          private bool enclavado = false;
14
15          public PLC_Comunicacion CComunicacion;
16
17          void Start()
18          {
```

Figura 102 Referencia al script PLC_Comunicacion

```
22
23      public void Enclavar()
24      {
25
26          // Detectar clic con Fire1 y raycast sobre este objeto
27          if (Input.GetButtonDown("Fire1"))
28          {
29              enclavado = !enclavado;
30              CComunicacion.EscribirBEM1();
31          }
32
```

Figura 103 Llamada a función EscribirBEM1 contenida en el script PLC_Comunicacion

Etapa 3: Enlace de entornos

Antes de establecer el enlace entre ambos sistemas, es importante recordar el concepto fundamental de un gemelo digital:

-se trata de dos interpretaciones de un mismo sistema, uno de características virtuales y el otro de elementos físicos, que operan de manera equivalente-.

Además, los gemelos digitales “se utilizan durante el ciclo de vida del producto para simular, predecir y optimizar el producto y el sistema de producción antes de invertir en prototipos y activos físicos” (Gemelo Digital | Siemens Software, 2025). [11]

Bajo este enfoque, el proyecto podrá operar en dos modalidades:

- ➔ Totalmente virtual
- ➔ Con sistema físico

En los apartados siguientes se describen los pasos necesarios para la puesta en marcha de cada configuración.

Gemelo digital totalmente virtual

Para esta modalidad se debe utilizar un software complementario llamado NetToPLCsim, el cual es de licencia libre y puede descargarse desde la red. [12]

Enseguida, se presentan los pasos para establecer la comunicación requerida. Es indispensable respetar el orden indicado, ya que un cambio en la secuencia puede impedir la correcta transmisión bidireccional de datos:

- Ejecutar PLCSIM y preparar la tabla de simulación para facilitar la activación y desactivación de señales.
- Iniciar NetToPLCsim. La dirección de trabajo es 127.0.0.1, mientras que IP asignada a PLCSIM será 192.168.105.121 (este valor puede modificarse según los requerimientos del usuario).
- Ejecutar la escena de Unity.

➔ *Uso de NetToPLCsim:*

Ejecutar NetToPLCsim como administrador (Figura 104) para que pueda mantener un dominio sobre el puerto 102 de comunicación y generar una conexión TCP/ IP. La herramienta permite acceder a la red generada por la aplicación PLCSim, haciendo uso del protocolo PROFINET desde una red local, generada directamente en la computadora en la que se ejecuta la simulación mediante la interfaz de red.

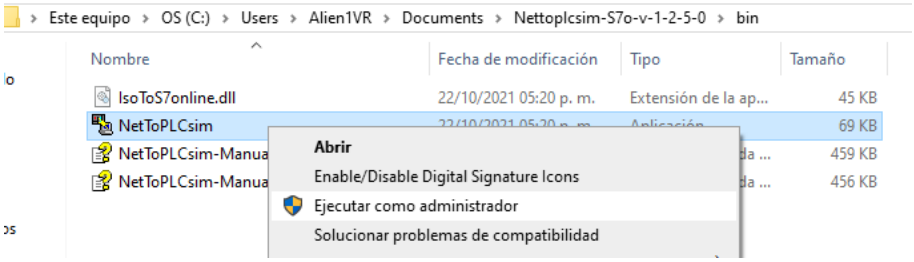


Figura 104 Ejecutar NetToPLCsim como administrador

La configuración de la aplicación es sencilla, se requiere de un nombre de red (PLC#001), una IP de trabajo (127.0.0.1) y la IP de operación del PLCSIM (192.168.105.121). Finalmente, configurar el Rack y Slot de trabajo del CPU, en este caso será 0 y 1, respectivamente. Dar click en el botón “OK”, en la Figura 105 se observa todo el procedimiento.

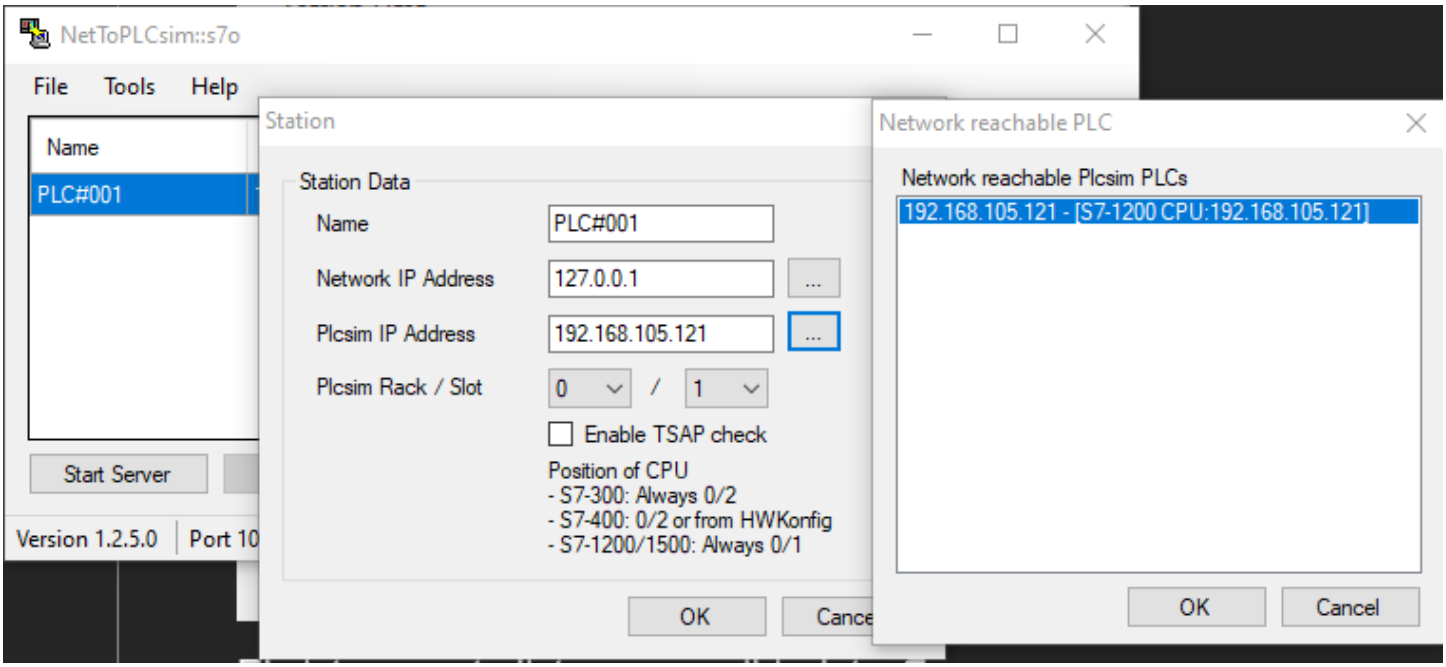


Figura 105 Configuración de NetToPLCsim

Dar click en “Start Server”, con ello el estatus cambiará a “Running”, Figura 106.

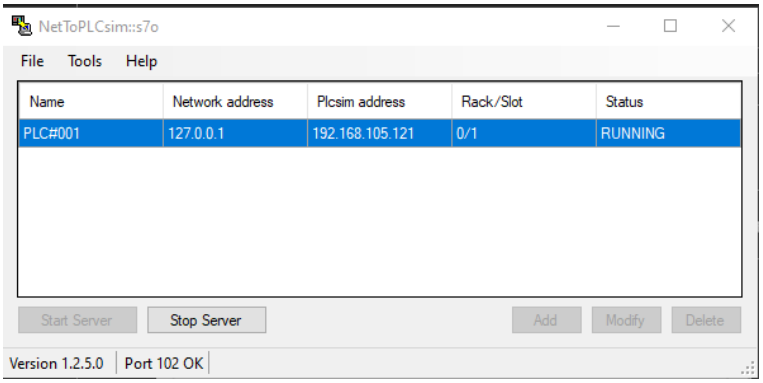


Figura 106 Inicio del servidor

Para obtener un cambio, activar la señal de entrada BSecundario2 de la Tabla Sim. Las señales *Luz_Verde* y *EDO_Luz_Verde* se vuelven verdaderas. En el entorno de Unity, el indicador verde de la torreta se activa.

Ahora, TIA Portal y Unity están comunicándose y tienen un reflejo de sus acciones, en la Figura 107 se aprecian ambos sistemas.

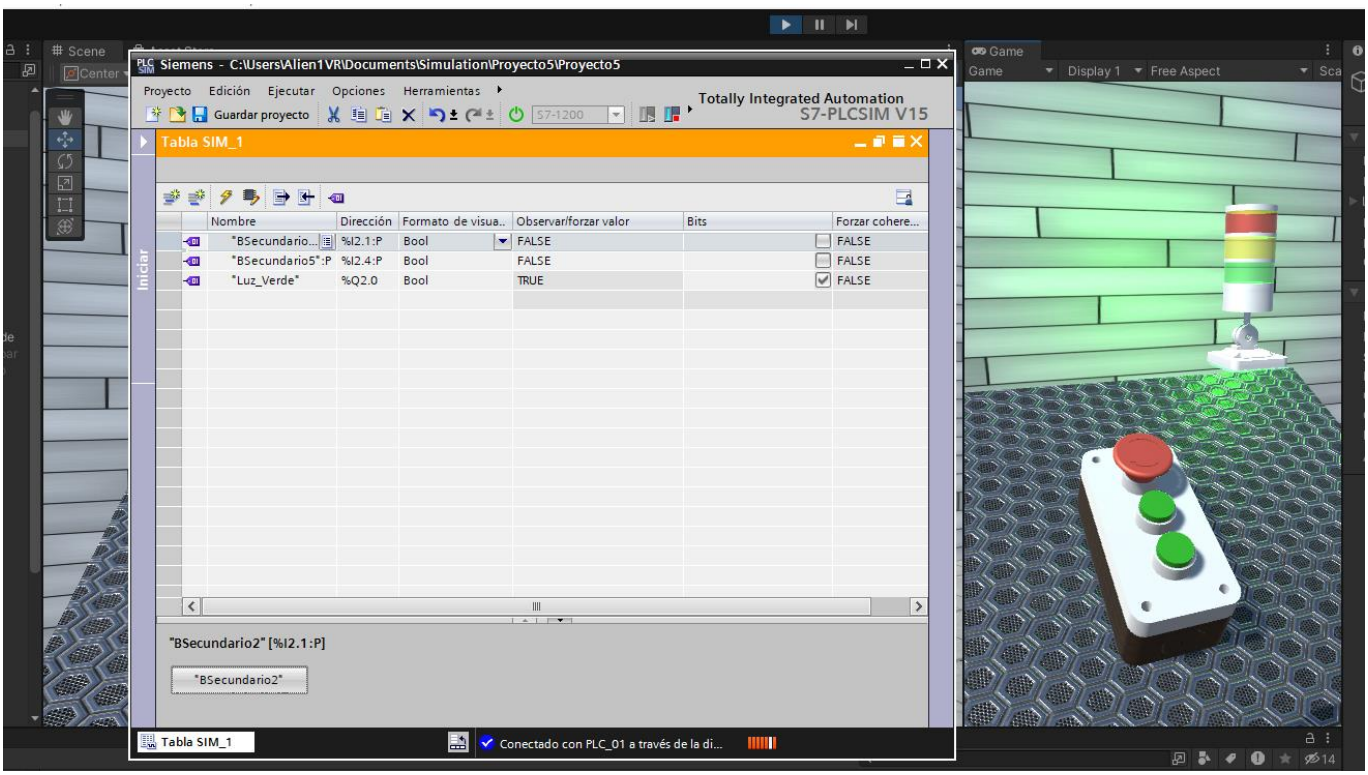


Figura 107 Activación de sistema desde TIA Portal

Para comprobar la escritura, en la Tabla Sim se agregan las variables del DB_Recibir_Datos (DB2.DBX0.0 y DB2.DBX0.1), Figura 108:

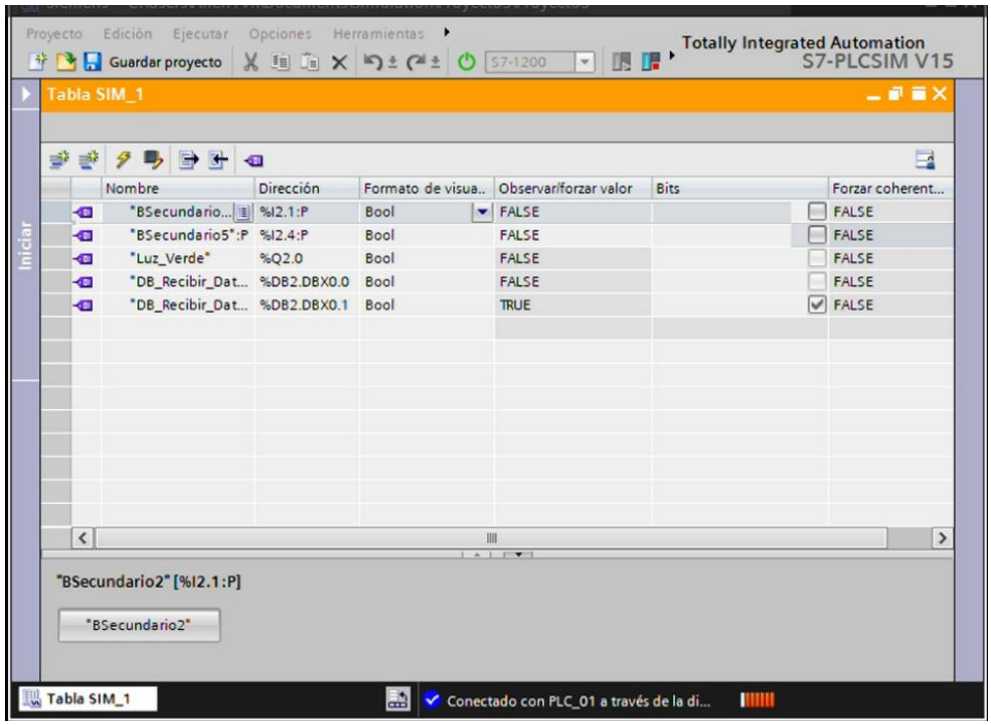


Figura 108 Señales Start_Virtual y Stop_Virtual

Desde Unity se presiona el botón enclavado (Figura 109). En la consola de Unity se aprecia el intercambio de datos; en la Figura 108, se observa que la señal DB2.DBX0.1 (paro virtual), se activa.

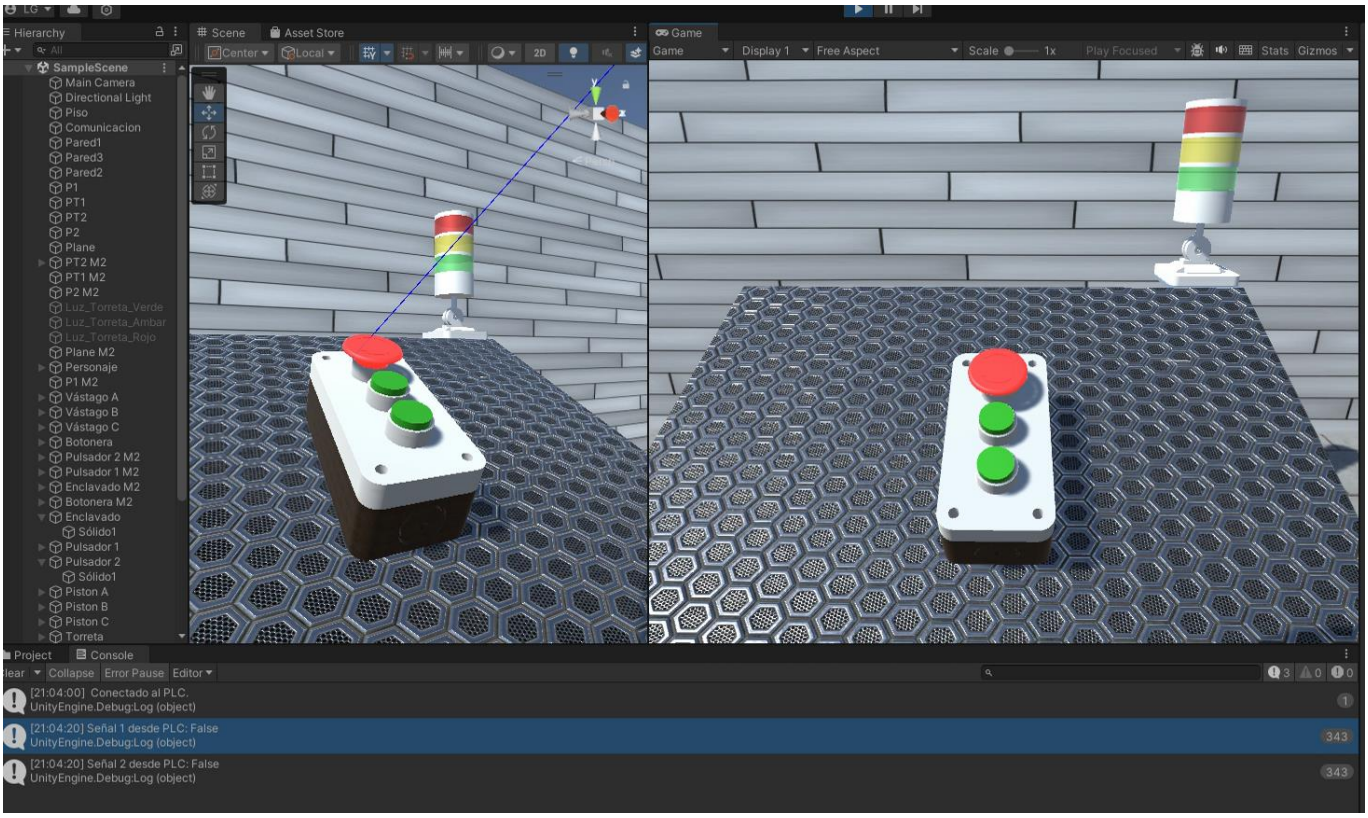


Figura 109 Activación de botón enclavado desde Unity

Gemelo digital con equipo físico

El procedimiento es más sencillo que de forma simulada, sin embargo, se requerirá de un cable Ethernet.

- ➔ Conectar un cable Ethernet del puerto RJ45 de la computadora al puerto de conexión del PLC; dado que la IP de trabajo es 192.168.105.121, el PLC de trabajo se ubica en el Gabinete 1.
- ➔ Cargar la rutina de control al PLC y ponerlo en modo ejecución (RUN).
- ➔ Debido a que el PLC que se trabaja no cuenta con una torreta, conectar los botones iluminados y la botonera externa mediante la caja de conexiones
- ➔ Verificar que la computadora de trabajo esté en el segmento de red del PLC.
- ➔ Cambiar la IP 127.0.0.1 a 192.168.105.121 en el script *PLC_comunicacion* (Figura 110). Ejecutar la escena de Unity.

```
void Start()
{
    plc = new Plc(CpuType.S71200, "192.168.105.121", 0, 1);   //Simulado: "127.0.0.1" Físico "192.168.105.121"
```

Figura 110 Cambio de IP

Para verificar si el equipo de cómputo está conectado al nodo de red en una conexión por Ethernet, es necesario entrar a las configuraciones de red del panel de control (Figura 111). La IP debe estar dentro del segmento, en caso contrario, asignarla manualmente y guardar los cambios.

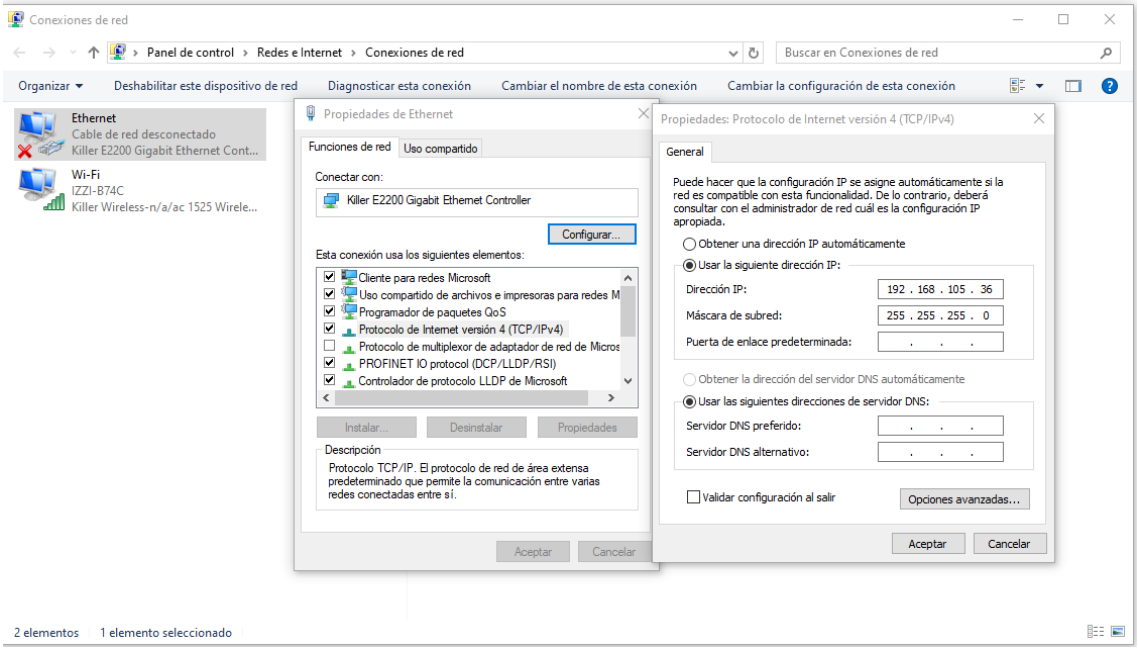


Figura 111 Ingreso al segmento de red (192.168.105.###)

Colocar TIA Portal en modo monitoreo y activar el botón pulsador de la botonea externa. El monitoreo debe ser similar al de la Figura 112

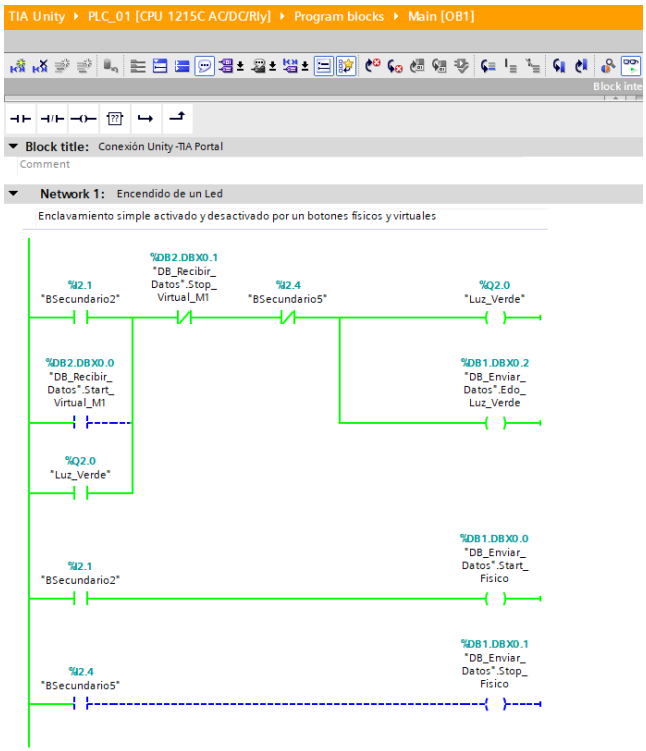


Figura 112 Activación de señal %I2.1 (BSecundario2)

En la escena de Unity, el indicador luminoso verde se enciende (Figura 113).

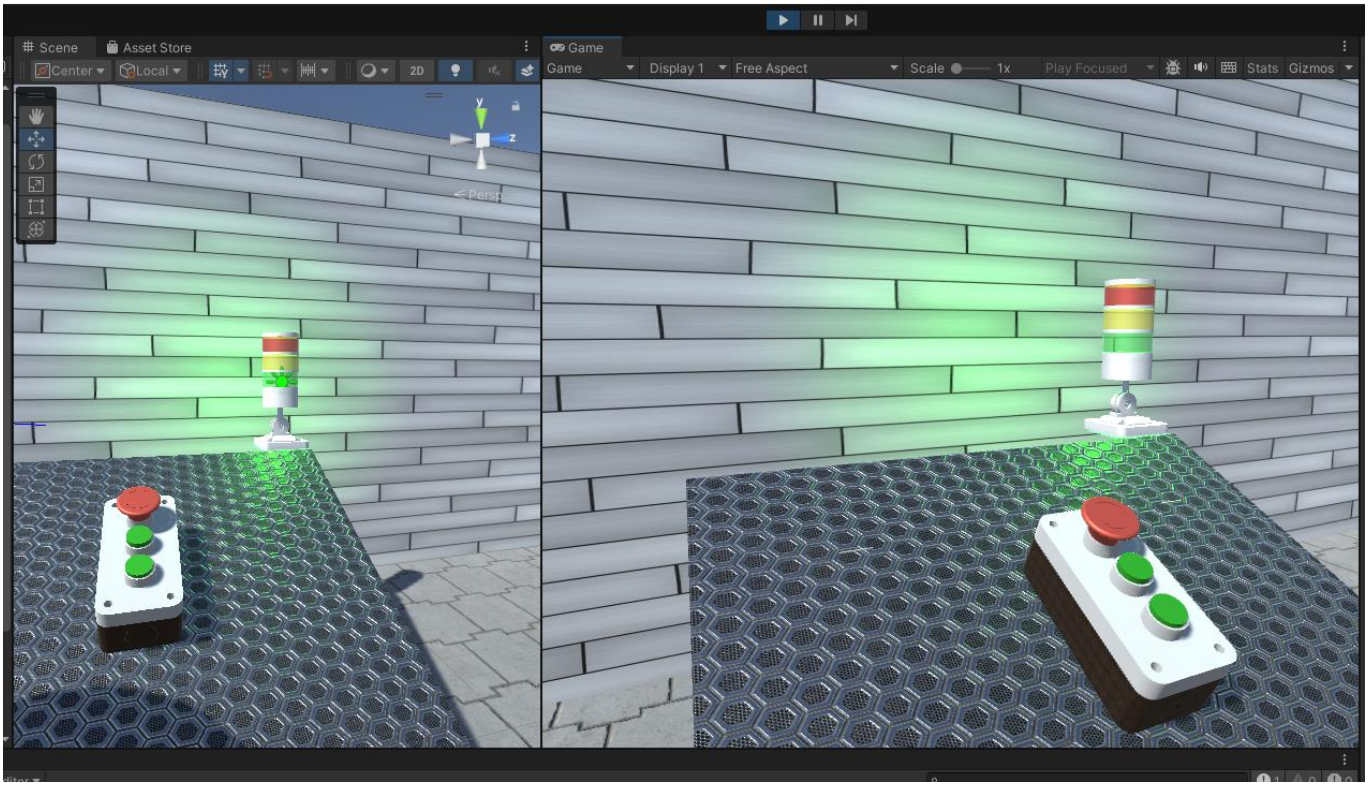


Figura 113 Activación de indicador verde mediante señal física

Al accionar el botón pulsador inferior de mesa neumática virtual, activa el sistema; en la Figura 114 se aprecia la interacción de la señal. Como resultado, el actuador físico y virtual responden tanto al ambiente físico como al virtual, Figura 115.

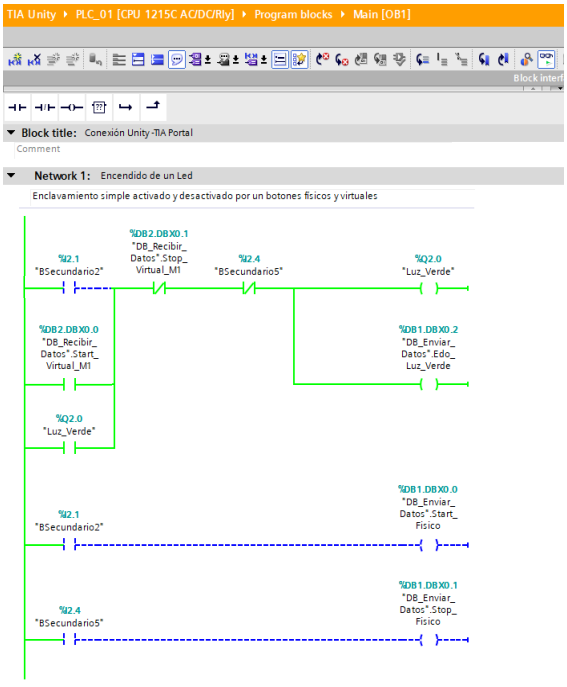


Figura 114 Activación desde el entorno virtual

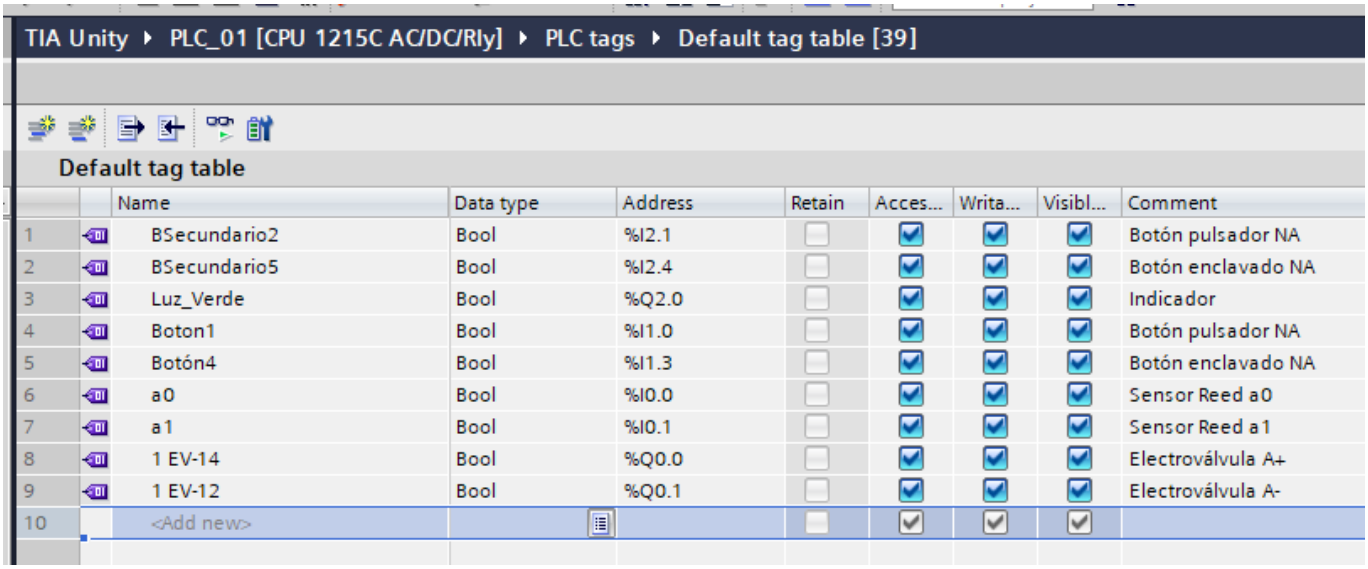
Figura 115 Indicador luminoso verde encendido

IX. Ejercicio 2: Activar y desactivar el vástago de un pistón neumático

El ejercicio se desarrolla en tres etapas. Primera etapa: lógica de programación en TIA Portal; segunda etapa: modificación de los scripts que interactúan para la extensión y retroceso del pistón de doble efecto (botones y sensores); tercera etapa: enlace de Unity y TIA Portal.

Etapa 1: Modificación de TIA Portal

Ya que el encendido y apagado del indicador luminoso sucede al mismo tiempo, abrir el proyecto del ejercicio anterior y actualizar las direcciones conforme a lo que se muestra en la Figura 116.



	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Comment
1	BSecundario2	Bool	%I2.1	☐	☑	☑	☑	Botón pulsador NA
2	BSecundario5	Bool	%I2.4	☐	☑	☑	☑	Botón enclavado NA
3	Luz_Verde	Bool	%Q2.0	☐	☑	☑	☑	Indicador
4	Boton1	Bool	%I1.0	☐	☑	☑	☑	Botón pulsador NA
5	Botón4	Bool	%I1.3	☐	☑	☑	☑	Botón enclavado NA
6	a0	Bool	%I0.0	☐	☑	☑	☑	Sensor Reed a0
7	a1	Bool	%I0.1	☐	☑	☑	☑	Sensor Reed a1
8	1 EV-14	Bool	%Q0.0	☐	☑	☑	☑	Electroválvula A+
9	1 EV-12	Bool	%Q0.1	☐	☑	☑	☑	Electroválvula A-
10	<Add new>			☐	☑	☑	☑	

Figura 116 Tags actualizados en el Default tag table

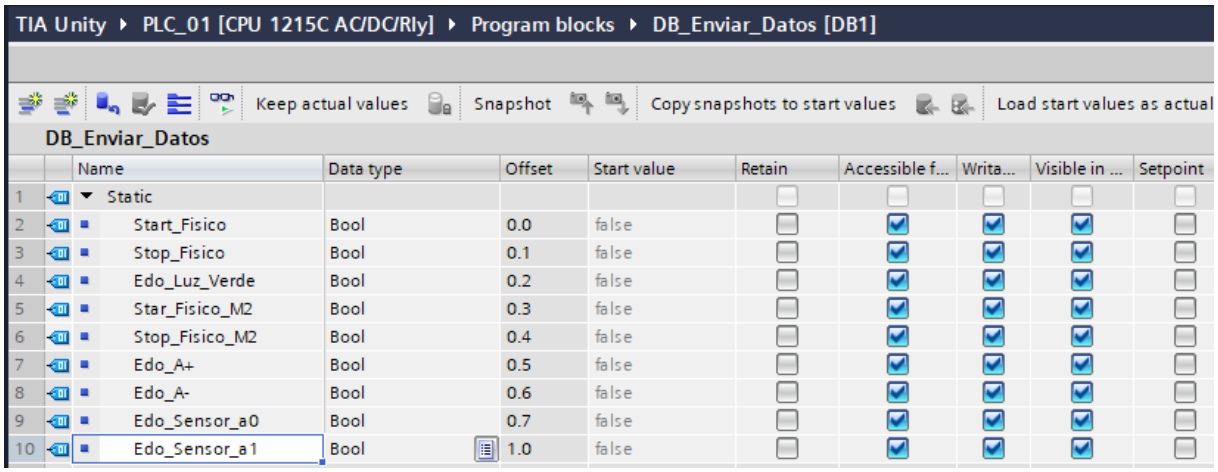
De la Figura 116 se tiene:

- *Boton1*: Botón pulsador (botonera de la mesa neumática) para arranque del sistema.
- *Boton4*: Botón enclavado (botonera de la mesa neumática) para detener el proceso.
- *a*0: Sensor de posición (posición retraída).
- *a1:* Sensor de posición (posición extendida).
- *1 EV-14:* Pilotaje 14 de la electroválvula del pistón A (movimiento A+).
- *1 EV-12:* Pilotaje 12 de la electroválvula del pistón A (movimiento A-).

Las señales a enviar a Unity son los estados de la electroválvula y los sensores.

- Edo_A+ → Pilotaje activo
- Edo_A- → Pilotaje activo
- Edo_Sensor_a0 → Estado del sensor
- Edo_Sensor_a1 → Estado del sensor

Actualizar las variables a enviar en el DB “Enviar datos”, Figura 117.



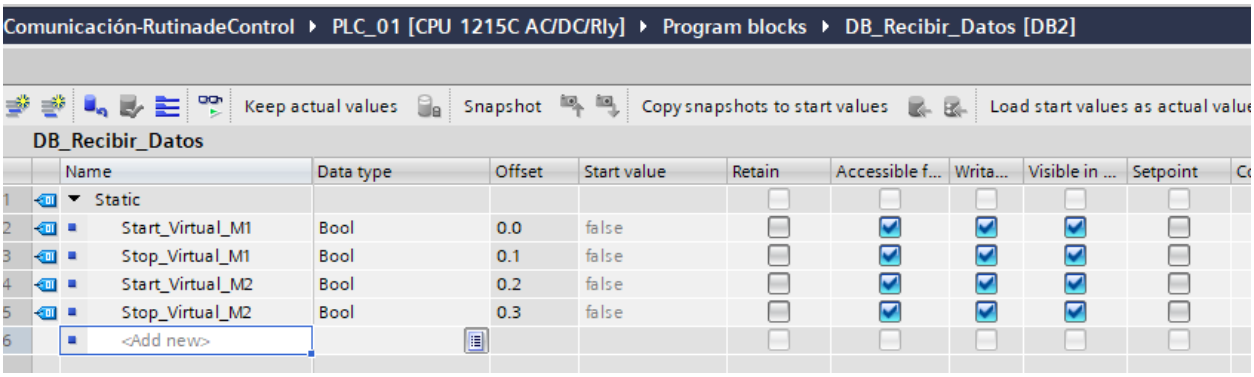
	Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint
1	Static								
2	Start_Fisico	Bool	0.0	false		☑	☑	☑	
3	Stop_Fisico	Bool	0.1	false		☑	☑	☑	
4	Edo_Luz_Verde	Bool	0.2	false		☑	☑	☑	
5	Star_Fisico_M2	Bool	0.3	false		☑	☑	☑	
6	Stop_Fisico_M2	Bool	0.4	false		☑	☑	☑	
7	Edo_A+	Bool	0.5	false		☑	☑	☑	
8	Edo_A-	Bool	0.6	false		☑	☑	☑	
9	Edo_Sensor_a0	Bool	0.7	false		☑	☑	☑	
10	Edo_Sensor_a1	Bool	1.0	false		☑	☑	☑	

Figura 117 Tags actualizados en el DB “Enviar datos”

Las variables a recibir de Unity son los estados de los botones.

- Start_Virtual_M2 → botón pulsador
- Stop_Virtual_M2 → botón enclavado

Actualizar las variables a recibir en el DB “Recibir datos”, Figura 118.



	Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint
1	Static								
2	Start_Virtual_M1	Bool	0.0	false		☑	☑	☑	
3	Stop_Virtual_M1	Bool	0.1	false		☑	☑	☑	
4	Start_Virtual_M2	Bool	0.2	false		☑	☑	☑	
5	Stop_Virtual_M2	Bool	0.3	false		☑	☑	☑	
6	<Add new>								

Figura 118 Tags actualizados en el DB “Recibir datos”

Al terminar de actualizar los bloques, compilar el programa.

En un Network nuevo, colocar la lógica de activación (tanto virtual como física) para la que bobina *1 EV-14* se energice; como restricción, la señal “a1”. En paralelo a la salida, colocar la bobina *Edo_A+.*

Enviar directamente las señales de los sensores (una por cada peldaño) al entorno virtual.

Para que retorne el pistón, colocar la señal del botón de paro (tanto virtual como física) como señal de activación para la bobina *1 EV-12*; como restricción la señal “a0”. En paralelo a la salida, colocar la bobina *Edo_A+.*

En la Figura 119 se muestra el programa completo en TIA Portal.

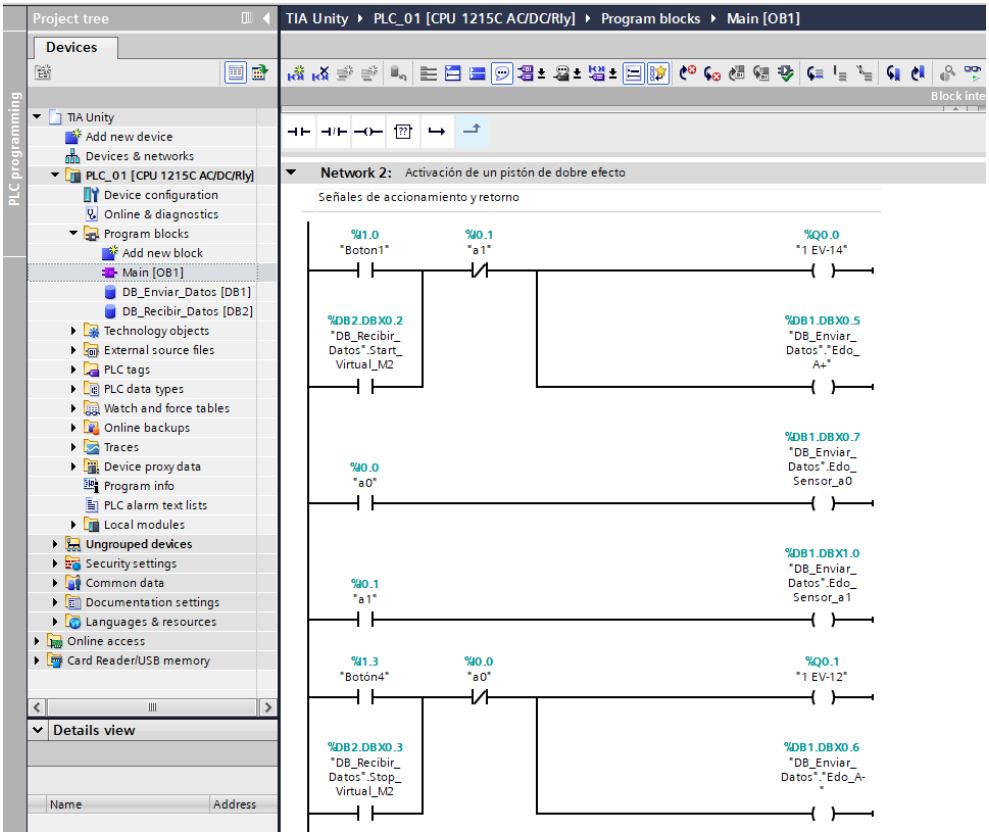


Figura 119 Lógica de activación de un pistón de doble efecto

Etapa 2: Modificación de scripts

La primera configuración será en el script *PLC_Comunicacion*.

```
6   public class PLC_Comunicacion : MonoBehaviour
7   {
8
9       public Luz_Verde luzVerd;   // Encendido de led Verde
10      public GameObject Sa0;      //Luz Sensor a0
11      public GameObject Sa1;      //Luz Sensor a1
12      public MovimientoPositivo PosicionVastago;
13
14      private Plc plc;
15      private bool bp2M1 = false;
16      private bool enclavado = false;
17      private bool bp2M2 = false;
18      private bool enclavadoM2 = false;
19      private bool bCentralM1 = false;
20
21      void Start()
22      {
23          plc = new Plc(CpuType.S71200, "127.0.0.1", 0, 1);
```

Figura 120 Declaración de variables

Dentro del método público y después de la variable *Luz_Verde*, declarar las siguientes variables públicas:

-public GameObject Sa0; → Referencia al Point light del sensor *a0*
-public GameObject Sa1; → Referencia al Point light del sensor *a1*.
-public MovimientoPositivo PosicionVastago; → Referencia al script *MovimientoPositivo*

Después de la variable *enclavado,* colocar las siguientes variables privadas*;*

- private bool bp2M2 = false; →Referencia al estado del botón pulsador que se encuentra en la parte inferior de la botonera virtual (mesa 2), inicia en falso.
- private bool enclavadoM2 = false; →Referencia al estado del botón enclavado de la botonera (mesa 2), inicia en falso.

La Figura 120 muestra parte del código; *bCentralM1* no es necesario colocarla.

```
36      void Update()
37      {
38          if (plc != null && plc.IsConnected)
39          {
40              try
41              {
42                  // Leer señal desde el PLC (ej: DB1.DBX0.0)
43                  bool StartVirtual = (bool)plc.Read("DB1.DBX0.0"); //Indicador Verde
44                  bool StopVirtual = (bool)plc.Read("DB1.DBX0.1");    //Indicador Verde
45
46                  bool StartVirtualM2 = (bool)plc.Read("DB1.DBX0.3"); // Activación de sistema; trabajo
47                  bool StopVirtualM2 = (bool)plc.Read("DB1.DBX0.4");  //Paro de sistema; reposo
48                  bool Sensora0 = (bool)plc.Read("DB1.DBX0.7");       //Sensor de reposo
49                  bool Sensora1 = (bool)plc.Read("DB1.DBX1.0");       //Sensor de trabajo
50
```

Figura 121 Variables a leer

Dentro del bloque *try* (ver Figura 121) y posterior a la variable *StopVirtual*, colocar las variables a leer de TIA Portal:

- bool StartVirtualM2 = (bool)plc.Read(“DB1.DBX0.3”); → Botón pulsador de mesa neumática.
- bool StopVirtualM2 = (bool)plc.Read(“DB1.DBX0.4”); → Botón enclavado de mesa neumática.
- bool Sensora0 = (bool)plc.Read(“DB1.DBX0.7”); → Sensor de efecto Reed.
- bool Sensora1 = (bool)plc.Read(“DB1.DBX1.0”); → Sensor de efecto Reed.

Archivos varios

```
52
53                  if (StartVirtual)
54                  {
55                      EncenderIV();
56                  }
57
58                  if (StopVirtual)
59                  {
60                      ApagarIV();
61                  }
62
63                  if (StartVirtualM2)
64                  {
65                      Expulsion();
66                  }
67
68                  if (StopVirtualM2)
69                  {
70                      Retroceso();
71                  }
72
73                  if (Sensora0 == true)
74                  {
75                      Sa0.SetActive(true);
76                  }
77
78                  if (Sensora0 == false)
79                  {
80                      Sa0.SetActive(false);
81                  }
82
83
84                  if (Sensora1 == true)
85                  {
86                      Sa1.SetActive(true);
87                  }
88
89                  if (Sensora1 == false)
90                  {
91                      Sa1.SetActive(false);
92                  }
93
94                  Debug.Log("Señal 1 desde PLC: " + StartVirtual);
95                  Debug.Log("Señal 2 desde PLC: " + StopVirtual);
96              }
97              catch
98              {
99                  Debug.LogWarning(" Fallo al leer del PLC.");
100             }
101         }
102     }
103
```

Figura 122 Activación de métodos

Posterior a la lectura de las variables, definir las condiciones que permitirán activar las funciones que activan los eventos; posterior a la condición *StopVirtual,* utilizar la Figura 122 como referencia.

- Si *StartVirtualM2* es verdadera (botón presionado), se ejecuta el método Expulsion().

- Si *StopVirtualM2* es verdadera (botón presionado), se ejecuta el método Retroceso().

- Si *Sensora0* es igual a verdadero (sensor activo), se ejecuta Sa0.SetActive(true).

- Si *Sensora0* es igual a falso (sensor desactivado), se ejecuta Sa0.SetActive(false).

- Si *Sensora1* es igual a verdadero (sensor activo), se ejecuta Sa1.SetActive(true).

- Si *Sensora1* es igual a falso (sensor desactivado), se ejecuta Sa1.SetActive(false).

Opcionalmente, se pueden enviar mensajes a la consola de Unity para visualizar el estado recibido de los botones y facilitar el diagnóstico de comunicación.

```
112     public void EncenderIV()    // Indicador verde On
113     {
114         luzVerd.EncenderLuzPLC();
115     }
116
117     public void ApagarIV()      // Indicador verde Off
118     {
119         luzVerd.ApagarLuzPLC();
120     }
121
122     public void Expulsion()
123     {
124         PosicionVastago.OnPulsadorPressed();
125     }
126
127     public void Retroceso()
128     {
129         PosicionVastago.ToggleEnclavado();
130     }
131
132     public void EscribirBP2M1() //Señal virtual de paro mesa 1
133     {
134         bp2M1 = !bp2M1;
135         plc.Write("DB2.DBX0.0", bp2M1);
136     }
137
138     public void EscribirBEM1() //Señal virtual de paro mesa 1
139     {
140         enclavado = !enclavado;
141         plc.Write("DB2.DBX0.1", enclavado);    //Paro virtual
142         plc.Write("DB2.DBX0.5", enclavado); //Paro de semáforo
143     }
144
145     public void EscribirBP2M2() //Señal virtual de paro mesa 2
146     {
147         bp2M2 = !bp2M2;
148         plc.Write("DB2.DBX0.2", bp2M2);
149     }
150
151     public void EscribirBEM2() //Señal virtual de paro mesa 2
152     {
153         enclavadoM2 = !enclavadoM2;
154         plc.Write("DB2.DBX0.3", enclavadoM2);    //Paro virtual
155     }
156
157 }
158
```

Figura 123 Activación de scripts y envío de datos

Posterior al bloque ApagarIV(); colocar los siguientes bloques (observar la Figura 123 como referencia):

Bloque Expulsion(): llamada al script *PosicionVastago* para activar la función del pulsado presionado (*OnPulsadorPressed()*).

Bloque Retroceso(): llamada al script *PosicionVastago* para activar la función del enclavado presionado (*ToggleEncalvado()*).

Posterior al bloque EscribirBEM1(), enviar los estados del resto de botones virtuales:

Bloque EscribirBP2M2(): al entrar a este bloque, se niega el estado del botón (bp2M2) para informar que se ha registrado un cambio en su estado y se coloca el siguiente código: plc.Write(“DB2.DBX0.2”, bp2M2);

Bloque EscribirBEM2(): al entrar a este bloque, se niega el estado del botón (enclavadoM2) para informar que se ha registrado un cambio en su estado y se coloca el siguiente código: plc.Write(“DB2.DBX0.3”, enclavadoM2);

Modificación al script Activar_MOVPOS

Abrir el script Activar_MOVPOS. Agregar una nueva variable, como se muestra en la Figura 124:

→ public PLC_Comunicacion EnviarDato; llamado al script de comunicación.

Dentro del bloque *PulsoPos(),* Figura 125*:*

→ Cuando se da click con el botón derecho del mouse; EnviarDato.EscribirBP2M2();

→ Cuando se deja de dar click con el botón derecho del mouse; EnviarDato.EscribirBP2M2();

```
10
11      private Vector3 posicionInicial;
12      private Vector3 posicionPresionada;
13
14      public MovimientoPositivo movimiento;
15      public PLC_Comunicacion EnviarDato;
16
17      private bool presionado = false;
18
```

Figura 124 Conexión al script PLC_Comunicacion

```
25
26      // Update is called once per frame
27      public void PulsoPos()
28      {
29          if (Input.GetButtonDown("Fire1"))
30          {
31              presionado = true;
32              movimiento.OnPulsadorPressed();
33              EnviarDato.EscribirBP2M2();
34          }
35
36          if (Input.GetButtonUp("Fire1"))
37          {
38              presionado = false;
39              movimiento.OnPulsadorReleased();
40              EnviarDato.EscribirBP2M2();
41          }
42
43          // Movimiento suave
44          if (presionado)
```

Figura 125 Envío de estado del botón pulsador

Modificación al script Activar_MOVNEG

Abrir el script Activar_MOVPOS. Agregar una nueva variable, como se muestra en la Figura 126:

- ➔ public PLC_Comunicacion DatoEntrada; llamado al script de comunicación.

Dentro del bloque *EnclavarNeg(),* Figura 125*:*

- ➔ Cuando se da click con el botón derecho del mouse; DatoEntrada.EscribirBEM2();

```
10
11      private Vector3 posicionInicial;
12      private Vector3 posicionPresionada;
13      private bool enclavado = false;
14
15      public MovimientoPositivo movimiento;
16      public PLC_Comunicacion DatoEntrada;
17
18      void Start()
19      {
20          posicionInicial = transform.position;
21          posicionPresionada = posicionInicial - new Vector3(distancia2, distancia, 0);
22      }
23
24      public void EnclavarNeg()
25      {
26
27          // Detectar clic con Fire1 y raycast sobre este objeto
28          if (Input.GetButtonDown("Fire1"))
29          {
30              enclavado = !enclavado;
31              movimiento.ToggleEnclavado();
32              DatoEntrada.EscribirBEM2();
33          }
34
35          // Movimiento según estado enclavado
36          if (enclavado)
```

Figura 126 Envío de estado del botón enclavado

El script *MovimientoPositivo* no presenta modificaciones.

Etapa 3: Enlace de entornos

Para el enlace de entornos, se puede elegir la opción de funcionamiento más conveniente:

- ➔ Totalmente virtual
- ➔ Con sistema físico

En cualquier caso, los resultados serán:

Gemelo digital totalmente virtual

No olvidar seguir el orden:

- Arrancar PLCSIM (con su Tabla Sim)
- Ejecutar como administrador NetToPLCsim
- Ejecutar la escena de Unity

En Unity, activar el botón pulsador inferior de la Mesa virtual 2, como se muestra en la Figura 127

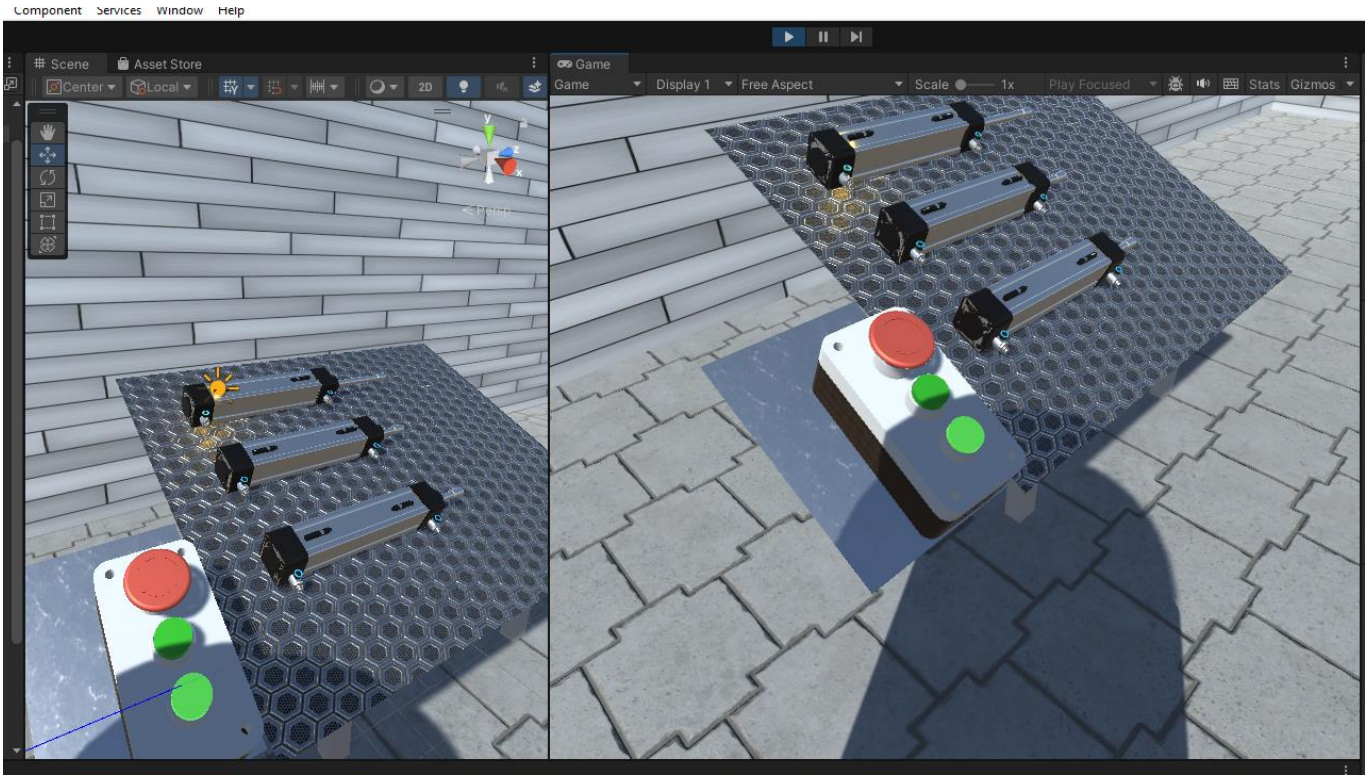


Figura 127 Expulsión del vástago del pistón A

Ya que se está simulando el entorno, el estado del sensor *a0* no cambiará hasta ser desactivado de forma manual en la Tabla Sim; el sensor *a1*, presenta la misma situación. En la Figura 128 se muestra la desactivación de la señal *a0* y la activación de la señal *a1.*

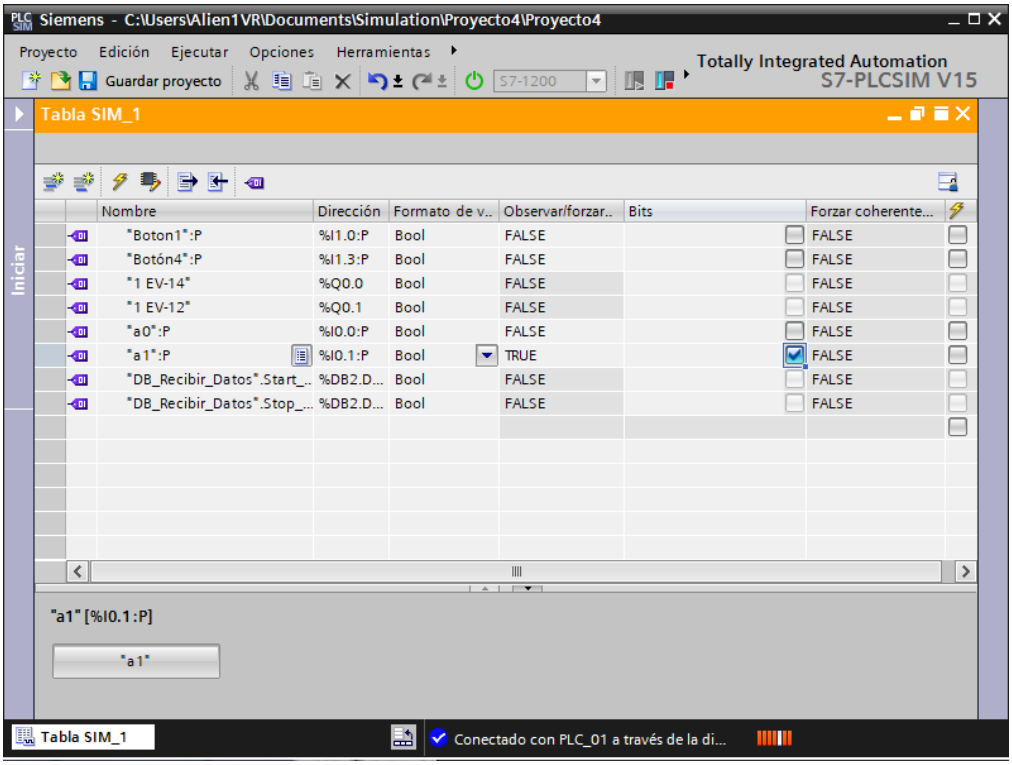


Figura 128 Señal a1 verdadera

Con esta acción, en la Figura 129 el sensor *a1* se enciende

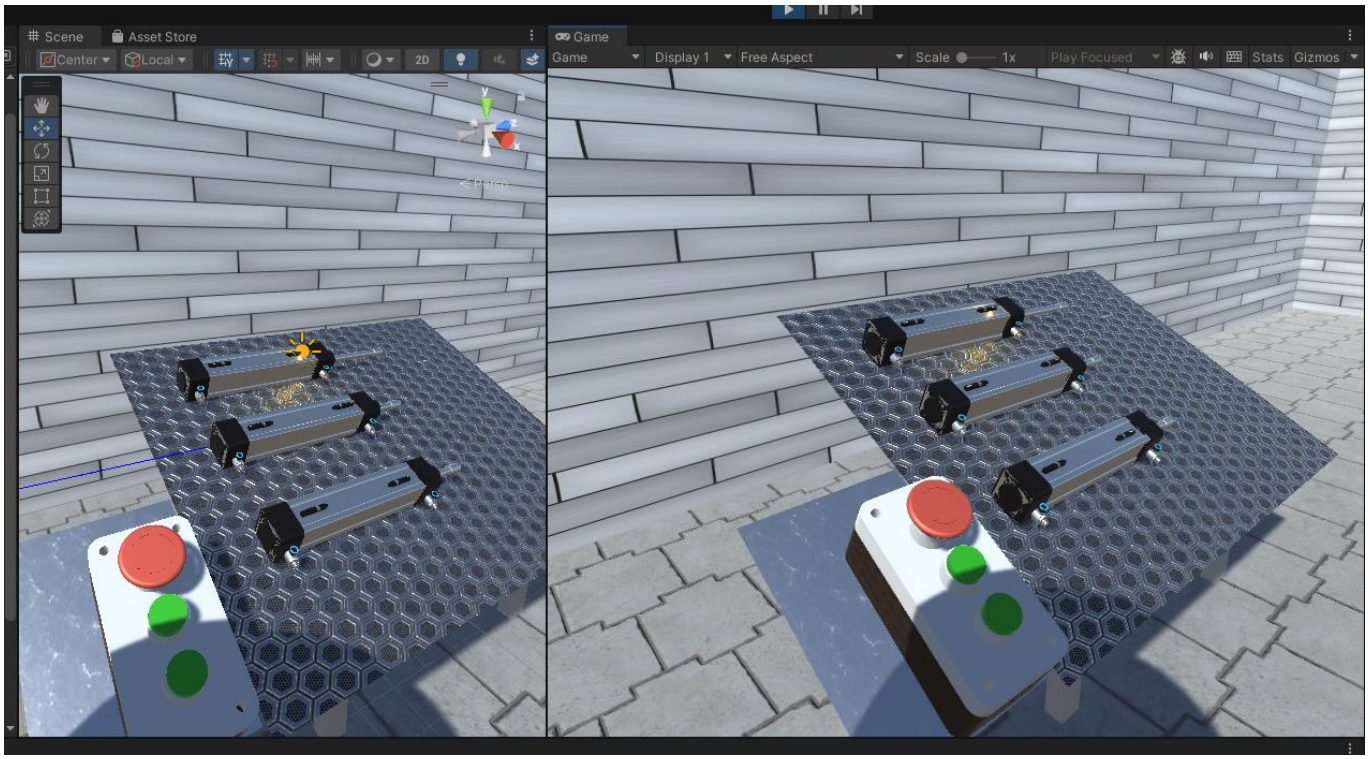


Figura 129 Vástago extendido y señal a1 verdadera

Desde la Tabla Sim, se activa el botón enclavado y al mismo tiempo, se modifican los estados de sensores (Figura 130); la respuesta en Unity se aprecia en la Figura 131

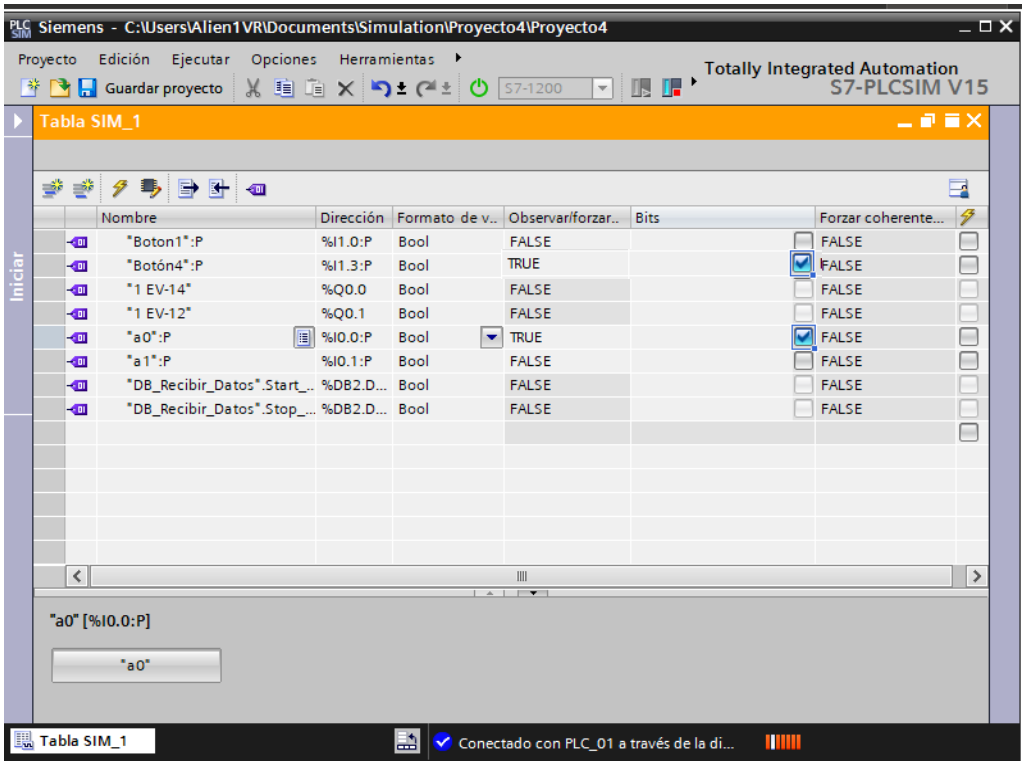


Figura 130 Activación de botón enclavado y de sensor a0

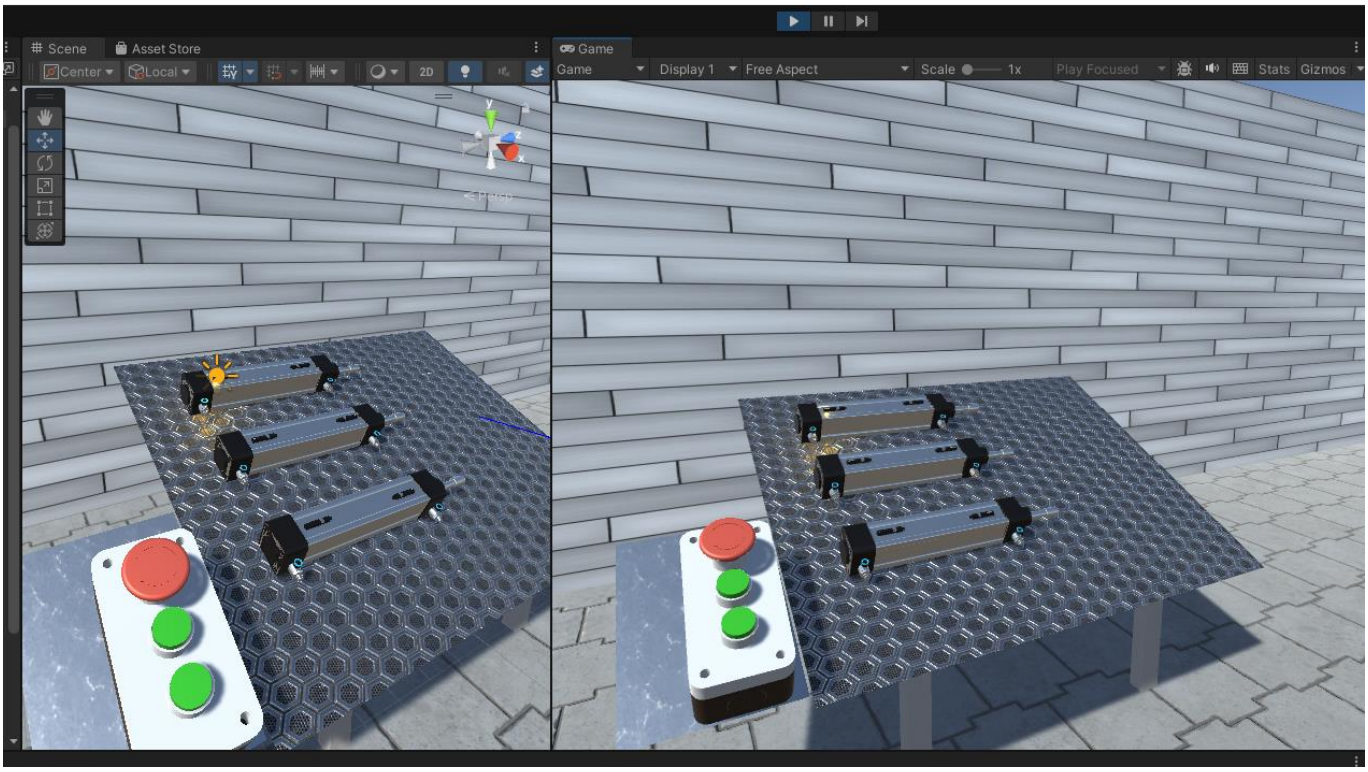


Figura 131 Retorno de vástago y activación de señal a0

Gemelo digital sistema físico

- Conectar el PLC con la computadora mediante un cable Ethernet
- Colocar la IP del PLC físico.
- Ejecutar la escena de Unity

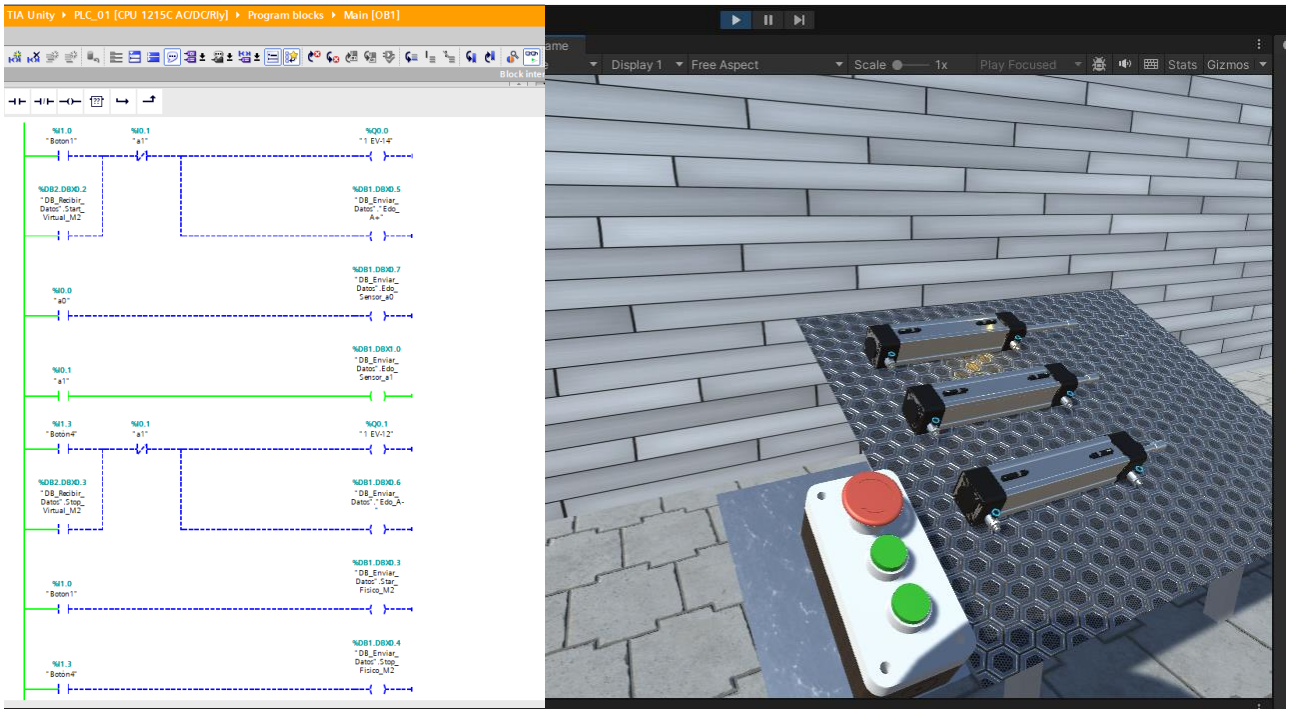


Figura 132 Activación por botón físico

Al presionar el botón físico, se expulsa el vástago; ahora los sensores responderán por sí mismos. En la Figura 132 se muestra el monitoreo de TIA Portal y al mismo tiempo la reacción del entorno virtual.

La Figura 133 muestra el comportamiento del equipo físico.

Figura 133 Activación del actuador A (Movimiento A+)

Para el retroceso, activamos desde el ambiente virtual, como se muestra en la Figura 134.

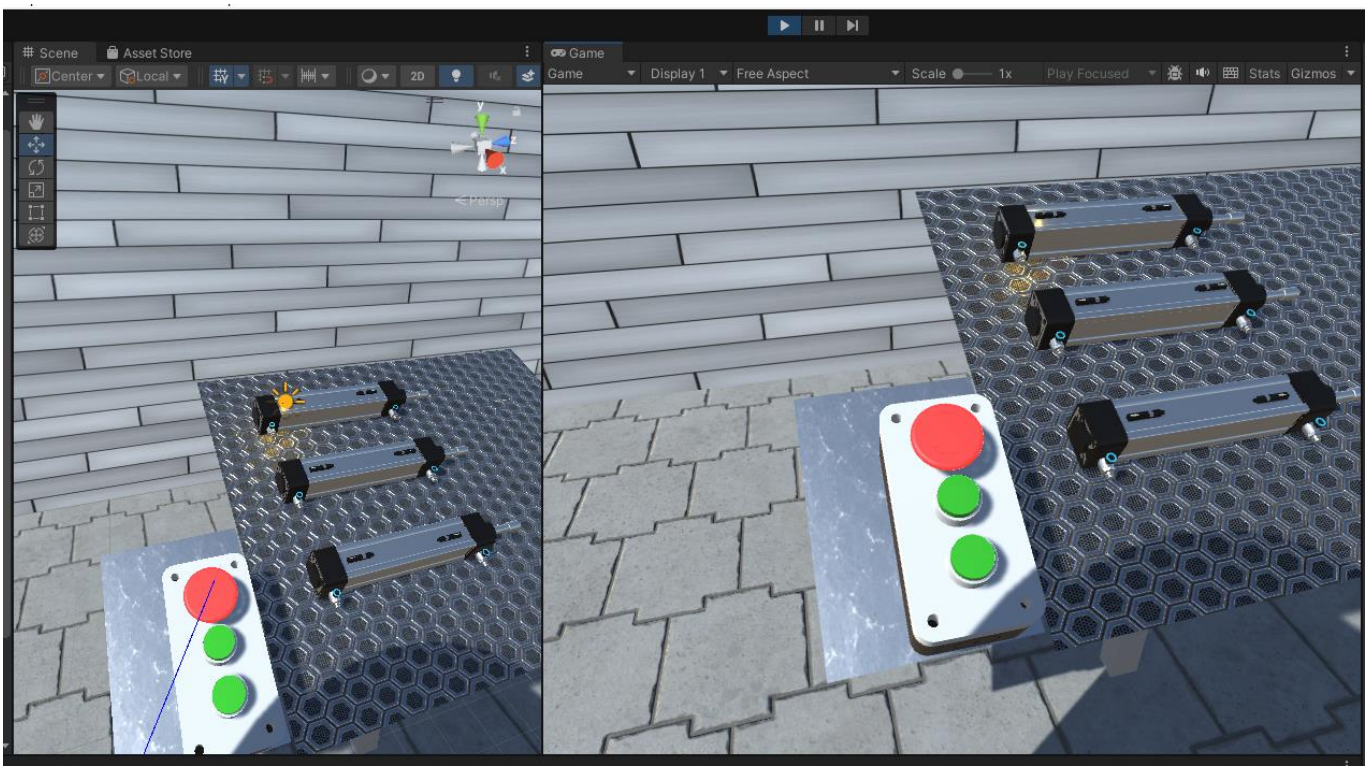


Figura 134 Retroceso del vástago desde Unity

En la Figura 135 se muestra el momento exacto de la activación virtual en TIA Portal, mientras que del lado derecho se observa el actuador en su posición de reposo.

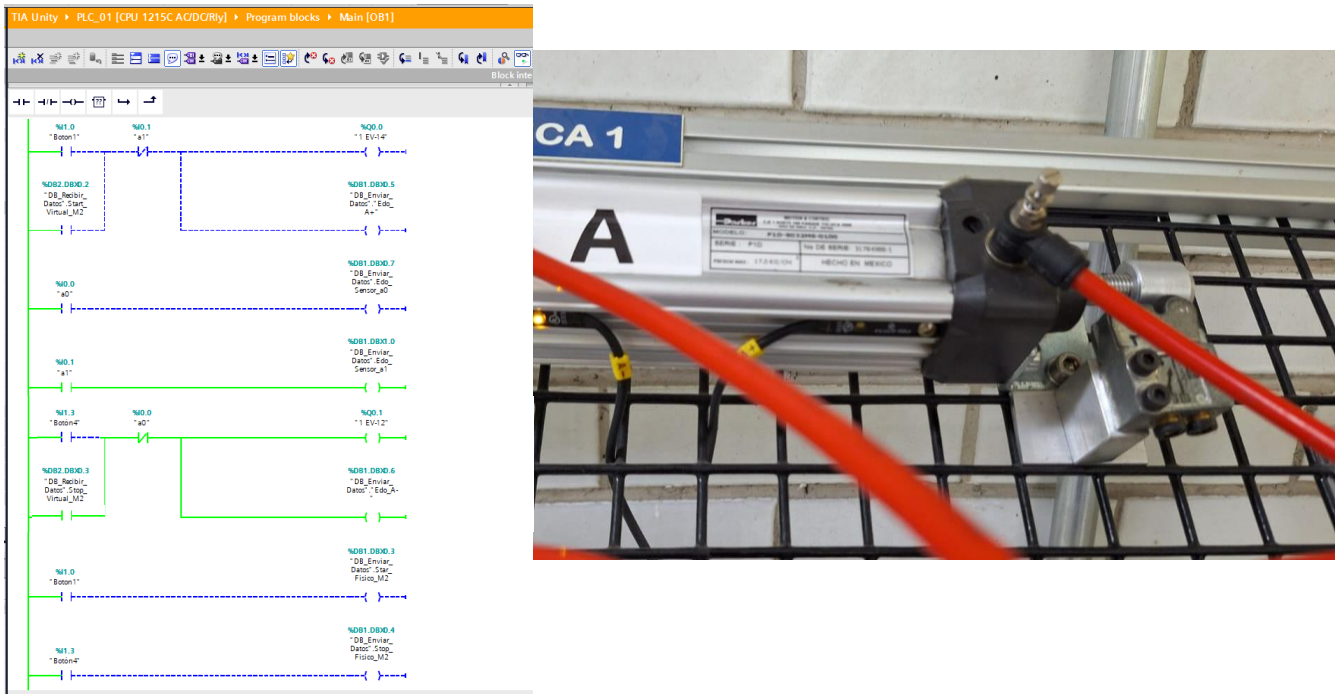


Figura 135 Monitoreo en TIA Portal y actuador físico

Para concluir, las posibilidades que pueden desarrollarse mediante una adecuada inversión de tiempo y, especialmente, el interés por los sistemas automatizados, son amplias. Como ejemplo de ello, la Figura 136 muestra uno de los resultados que pueden alcanzarse al integrar correctamente los elementos vistos en este manual y en la Figura 137, su accionamiento en el equipo físico.

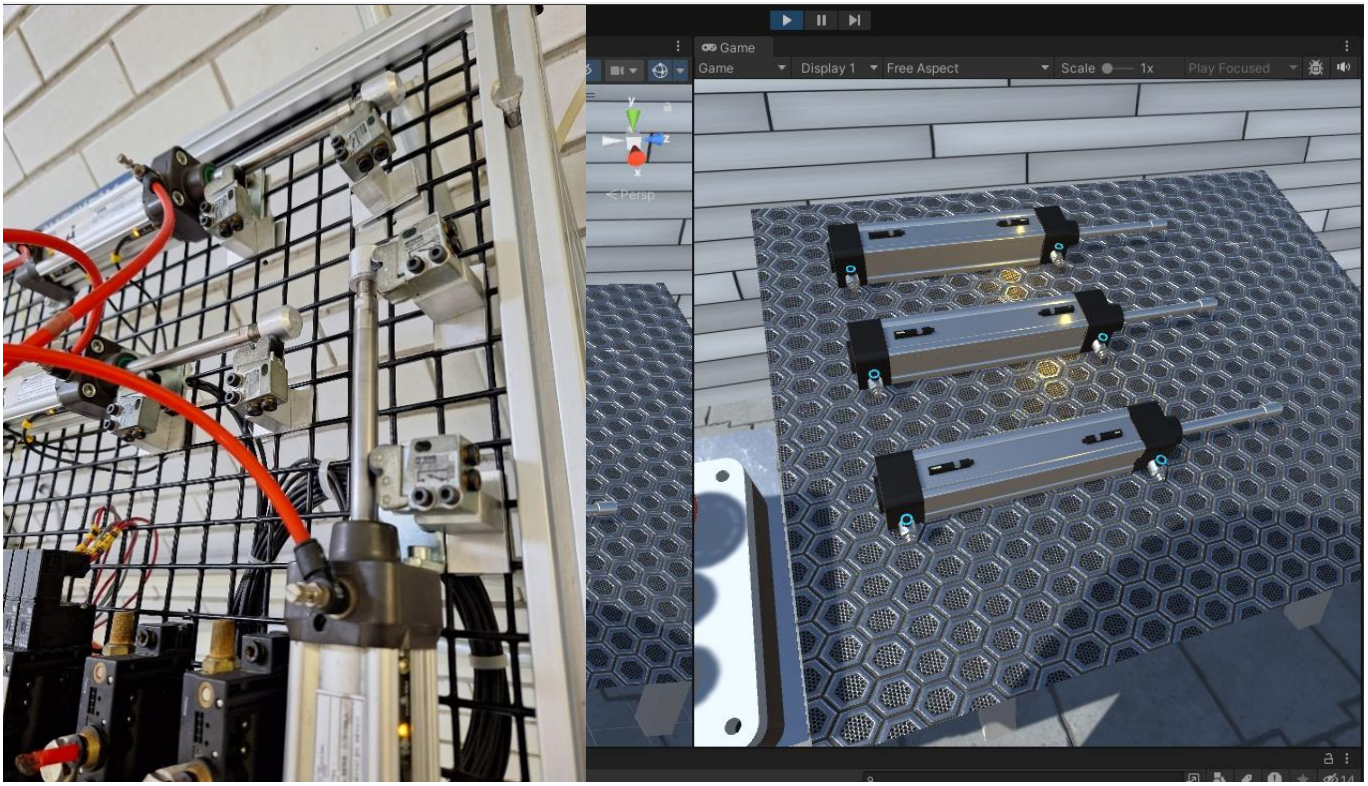


Figura 136 Activación de los pistones A, B y C

X. Referencias:

[1] Poliigon - PBR Textures, Models and HDRIs. (s. f.). Poliigon. Recuperado de: https://www.poliigon.com/ [Accedido en 15-04-2025].

[2] Unity AssetStore. (s. f.). Unity. Recuperado de: https://unity.com/es/pages/introduction-to-asset-store [Accedido en 14-04-2025].

[3] Technologies, U. (s. f.). Unity - Manual: ¿Cómo importo modelos desde mi aplicación 3D? Recuperado de: https://docs.unity3d.com/es/530/Manual/HOWTO-importObject.html#:~:text=Archivos%203D%20exportados,pueden%20encontrar%20en%20muchas%20aplicaciones [Accedido en 14-04-2025].

[4] Douglasdeal. (mayo 2023). *Como MOVER una CAMARA en UNITY 3D con el MOUSE [FUNCIONA] Para JUEGO FPS #1,* [Archivo de Video]. YouTube. Recuperado de: https://www.youtube.com/watch?v=yhDq2vHQDiE&list=PLfsT5XTEhZzEYZuPWFyraszuGnwOK62K2 [Accedido en 17-04-2025].

[5] Douglasdeal. (mayo 2023). *Como hacer MOVER un PERSONAJE en UNITY 3d [Funciona] primera persona FPS #2,* [Archivo de Video]. YouTube. Recuperado de: https://www.youtube.com/watch?v=MSynlrqQXKU&list=PLfsT5XTEhZzEYZuPWFyraszuGnwOK62K2&index=3 [Accedido en 17-04-2025].

[6] Multiverso Sensorial. (octubre 2021). *Como hacer un interruptor de luz en Unity,* [Archivo de Video]. YouTube. Recuperado de: https://www.youtube.com/watch?v=9DmPRqUFACA&list=PLfsT5XTEhZzEYZuPWFyraszuGnwOK62K2&index=5 [Accedido en 30-04-2025].

[7] DansterDev. (agosto 2021). *Como INTERACTUAR con objetos ¡RAPIDO Y SENCILLO! Unity 2021,* [Archivo de Video]. YouTube. Recuperado de: https://www.youtube.com/watch?v=oaWsRmDTLBo&list=PLfsT5XTEhZzEYZuPWFyraszuGnwOK62K2&index=6 [Accedido en 01-05-2025].

[8] Luis Antonio Aceves Argueta. (octubre 2021). *Objetos transparentes en Unity*, [Archivo de Vídeo]. YouTube. Recuperado de: https://www.youtube.com/watch?v=48jSajugR4c. [Accedido en 16-04-2025].

[9] S7NetPlus. (2023). GitHub - S7NetPlus/s7netplus: S7.NET+ -- A .NET library to connect to Siemens Step7 devices. GitHub. Recuperado de https://github.com/S7NetPlus/s7netplus. [Accedido en 01-11-2024].

[10] Siemens. (noviembre 2024). *Diferencia entre el acceso estándar y el acceso optimizado a los bloques*. Recuperado de: https://support.industry.siemens.com/cs/document/67655611/diferencia-entre-el-acceso-est%C3%A1ndar-y-el-acceso-optimizado-a-los-bloques?dti=0&lc=es-MX. ID 67655611. [Accedido en 20-09-2025].

[11] Gemelo digital | Siemens Software. (2025). Siemens Digital Industries Software. Recuperado de: https://www.sw.siemens.com/es-ES/technology/digital-twin/. [Accedido en 22-09-2025].

[12] SourceForge. (2025). NetToPLCsim - Network extension for Plcsim. NetToPLCsim. Recuperado de https://nettoplcsim.sourceforge.net/. [Accedido en 05-05-2025].

XI. Anexo

Se colocan todos los scripts utilizados en el manual:

1. Código del script de movimiento de la cámara del personaje (*Movimiento_Personaje*).

Movimiento_Personaje.cs | Mirada_Camara.cs

Archivos varios | Movimiento_Personaje

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Movimiento_Personaje : MonoBehaviour
6   {
7       public CharacterController Controlador;
8
9       public float Velocidad = 15f;
10      public float Gravedad = -10;
11
12      public Transform EnElPiso;
13      public float DistanciaDelPiso;
14      public LayerMask MascaraDelPiso;
15
16      Vector3 VelocidadAbajo;
17      bool EstaEnELPiso;
18
19      // Start is called before the first frame update
20      void Start()
21      {
22
23      }
24
25      // Update is called once per frame
26      void Update()
27      {
28          EstaEnELPiso = Physics.CheckSphere(EnElPiso.position, DistanciaDelPiso, MascaraDelPiso);
29
30          if (EstaEnELPiso && VelocidadAbajo.y < 0)
31          {
32              VelocidadAbajo.y = -2;
33          }
34
35          float x = Input.GetAxis("Horizontal");
36          float z = Input.GetAxis("Vertical");
37
38          Vector3 mover = transform.right * x + transform.forward * z;
39          Controlador.Move(mover * Velocidad * Time.deltaTime);
40
41          VelocidadAbajo.y += Gravedad * Time.deltaTime;
42
43          Controlador.Move(VelocidadAbajo * Time.deltaTime);
44      }
45  }
46
```

2. Código del script para la mirada en la cámara del personaje (*Mirada_Camara*).

Esquema del documento

Movimiento_Personaje.cs | Mirada_Camara.cs

Archivos varios | Mirada_Camara

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Mirada_Camara : MonoBehaviour
6   {
7       public float Velocidad = 100f;
8       float RotacionX = 0f;
9
10      public Transform Jugador;
11      // Start is called before the first frame update
12      void Start()
13      {
14          Cursor.lockState = CursorLockMode.Locked;
15      }
16
17      // Update is called once per frame
18      void Update()
19      {
20          float MauseX = Input.GetAxis("Mouse X") * Velocidad * Time.deltaTime;
21          float MauseY = Input.GetAxis("Mouse Y") * Velocidad * Time.deltaTime;
22
23          RotacionX -= MauseY;
24          RotacionX = Mathf.Clamp(RotacionX, -90f, 90f);
25
26          transform.localRotation = Quaternion.Euler(RotacionX, 0f, 0f);
27          Jugador.Rotate(Vector3.up * MauseX);
28      }
29  }
30
```

3. Código del script del RayCast (interacción con los botones).

RayCast.cs

Archivos varios — RayCast

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

public class RayCast : MonoBehaviour
{
    public int rango;

    private bool bandera;

    void Update()
    {
        RaycastHit hit;
        if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hit, rango))
        {
            if (hit.collider.GetComponent<Luz_Verde>() == true)
            {
                if(Input.GetButtonDown("Fire1"))
                {
                    if (hit.collider.GetComponent<Luz_Verde>().luz == true && bandera == true)
                    {
                        hit.collider.GetComponent<Luz_Verde>().EncenderLuz();
                    }

                    if (hit.collider.GetComponent<Luz_Verde>().luz == true && bandera == false)
                    {
                        hit.collider.GetComponent<Luz_Verde>().ApagarLuz();
                    }
                }
            }

            if (hit.collider.GetComponent<BotonPulsadorB1>() == true)
            {
                hit.collider.GetComponent<BotonPulsadorB1>().Pulso();
                bandera = true;
            }

            if (hit.collider.GetComponent<Enclavado_Reset>() == true)
            {
                hit.collider.GetComponent<Enclavado_Reset>().Enclavar();
                bandera = false;
            }

            if (hit.collider.GetComponent<Activar_MOVPOS>() == true)
            {
                hit.collider.GetComponent<Activar_MOVPOS>().PulsoPos();
            }

            if (hit.collider.GetComponent<Activar_MOVNEG>() == true)
            {
                hit.collider.GetComponent<Activar_MOVNEG>().EnclavarNeg();
            }

            if (hit.collider.GetComponent<Boton_Central_M1>() == true)
            {
                hit.collider.GetComponent<Boton_Central_M1>().PulsoCentralM1();
            }
        }
    }

    private void OnDrawGizmos()
    {
        Gizmos.color = Color.blue;
        Gizmos.DrawRay(Camera.main.transform.position, Camera.main.transform.forward * rango);
    }
}
```

4. Código del script del efecto Hover mediante el RayCast (*Luminosidad*).

Luminosidad.cs

Archivos varios — Luminosidad

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

public class Luminosidad : MonoBehaviour
{
    public Color hoverColor = Color.yellow;       // Color principal al hacer hover
    public Color emissionColor = Color.yellow;    // Color de emisión al hacer hover

    private Color originalColor;
    private Color originalEmission;
    private Renderer objRenderer;
    private bool isHovered = false;
    private Material instancedMaterial;

    void Start()
    {
        objRenderer = GetComponent<Renderer>();
        if (objRenderer != null)
        {
            // Se clona el material para no afectar a otros objetos
            instancedMaterial = new Material(objRenderer.material);
            objRenderer.material = instancedMaterial;

            originalColor = instancedMaterial.color;
            originalEmission = instancedMaterial.GetColor("_EmissionColor");
        }
    }

    void Update()
    {
        // Raycast desde la cámara hacia el puntero del mouse
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        if (Physics.Raycast(ray, out RaycastHit hit))
        {
            if (hit.transform == transform)
            {
                if (!isHovered)
                {
                    ApplyHoverColor();
                    isHovered = true;
                }
                return;
            }
        }

        if (isHovered)
        {
            ResetColor();
            isHovered = false;
        }
    }

    void ApplyHoverColor()
    {
        instancedMaterial.color = hoverColor;
        instancedMaterial.EnableKeyword("_EMISSION");
        instancedMaterial.SetColor("_EmissionColor", emissionColor);
    }

    void ResetColor()
    {
        instancedMaterial.color = originalColor;
        instancedMaterial.SetColor("_EmissionColor", originalEmission);
        instancedMaterial.DisableKeyword("_EMISSION");
    }
}
```

5. Código del script que anima el botón pulsador inferior de la Mesa 1 (*BotonPulsadorB1*).

Luz_Verde.cs | BotonPulsadorB1.cs

Archivos varios | BotonPulsadorB1

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

public class BotonPulsadorB1 : MonoBehaviour
{
    public float distancia = 1f;         // Cuánto se mueve el botón
    public float distancia2 = 1f;
    public float velocidad = 40f;           // Qué tan rápido se mueve

    private Vector3 posicionInicial;
    private Vector3 posicionPresionada;

    public PLC_Comunicacion Comunicacion;

    private bool presionado = false;

    // Start is called before the first frame update
    void Start()
    {
        posicionInicial = transform.position;
        posicionPresionada = posicionInicial - new Vector3(distancia2, distancia, 0);
    }

    // Update is called once per frame
    public void Pulso()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            presionado = true;
            Comunicacion.EscribirBP2M1();
        }

        if (Input.GetButtonUp("Fire1"))
        {
            presionado = false;
            Comunicacion.EscribirBP2M1();
        }

        // Movimiento suave
        if (presionado)
        {
            transform.position = Vector3.Lerp(transform.position, posicionPresionada, Time.deltaTime * velocidad);
        }
        else
        {
            transform.position = Vector3.Lerp(transform.position, posicionInicial, Time.deltaTime * velocidad);
        }
    }
}

```

6. Código del script que anima el botón enclavado de la Mesa 1 (*Enclavado_Reset*).

Enclavado_Reset.cs | BotonPulsadorB1.cs

Archivos varios | Enclavado_Reset

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

public class Enclavado_Reset : MonoBehaviour
{
    public float distancia = 1f;      // Movimiento en Y
    public float distancia2 = 1f;     // Movimiento en X
    public float velocidad = 10f;     // Velocidad del movimiento

    private Vector3 posicionInicial;
    private Vector3 posicionPresionada;
    private bool enclavado = false;

    public PLC_Comunicacion CComunicacion;

    void Start()
    {
        posicionInicial = transform.position;
        posicionPresionada = posicionInicial - new Vector3(distancia2, distancia, 0);
    }

    public void Enclavar()
    {

        // Detectar clic con Fire1 y raycast sobre este objeto
        if (Input.GetButtonDown("Fire1"))
        {
            enclavado = !enclavado;
            CComunicacion.EscribirBEM1();
        }

        // Movimiento según estado enclavado
        if (enclavado)
        {
            transform.position = Vector3.Lerp(transform.position, posicionPresionada, Time.deltaTime * velocidad);
        }
        else
        {
            transform.position = Vector3.Lerp(transform.position, posicionInicial, Time.deltaTime * velocidad);
        }
    }
}
```

7. Código del script que enciende y apaga un Point Light (*Luz_Verde*).

Luz_Verde.cs | BotonPulsadorB1.cs

Archivos varios

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

public class Luz_Verde : MonoBehaviour
{

    public GameObject lVerde;
    public bool luz;

    private bool EstadoVirtual = false;
    private bool EstadoFisico = false;

    public void EncenderLuz()
    {
        EstadoVirtual = true;
        EstadoLuz();
    }

    public void ApagarLuz()
    {
        EstadoVirtual = false;
        EstadoLuz();
    }

    public void EncenderLuzPLC()
    {
        EstadoFisico = true;
        EstadoLuz();
    }

    public void ApagarLuzPLC()
    {
        EstadoFisico = false;
        EstadoLuz();
    }

    public void EstadoLuz()
    {
        if (EstadoVirtual == true || EstadoFisico == true)
        {
            lVerde.SetActive(true);
        }

        else if (EstadoVirtual == false || EstadoFisico == false)
        {
            lVerde.SetActive(false);
        }
    }
}

```

8. Código del script que anima al botón pulsador inferior de la Mesa 2 (*Activar_MOVPOS*).

Activar_MOVPOS.cs | Enclavado_Reset.cs

Archivos varios | Activar_MOVPOS

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

public class Activar_MOVPOS : MonoBehaviour
{
    public float distancia = 1f;         // Cuánto se mueve el botón
    public float distancia2 = 1f;
    public float velocidad = 40f;           // Qué tan rápido se mueve

    private Vector3 posicionInicial;
    private Vector3 posicionPresionada;

    public MovimientoPositivo movimiento;
    public PLC_Comunicacion EnviarDato;

    private bool presionado = false;

    // Start is called before the first frame update
    void Start()
    {
        posicionInicial = transform.position;
        posicionPresionada = posicionInicial - new Vector3(distancia2, distancia, 0);
    }

    // Update is called once per frame
    public void PulsoPos()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            presionado = true;
            movimiento.OnPulsadorPressed();
            EnviarDato.EscribirBP2M2();
        }

        if (Input.GetButtonUp("Fire1"))
        {
            presionado = false;
            movimiento.OnPulsadorReleased();
            EnviarDato.EscribirBP2M2();
        }

        // Movimiento suave
        if (presionado)
        {
            transform.position = Vector3.Lerp(transform.position, posicionPresionada, Time.deltaTime * velocidad);
        }
        else
        {
            transform.position = Vector3.Lerp(transform.position, posicionInicial, Time.deltaTime * velocidad);
        }
    }

}
```

9. Código del script que animación el botón enclavado de la Mesa 2 (*Activar_MOVNEG*).

Activar_MOVNEG.cs | Activar_MOVPOS.cs | Enclavado_Reset.cs

Archivos varios — Activar_MOVNEG

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

public class Activar_MOVNEG : MonoBehaviour
{
    public float distancia = 1f;      // Movimiento en Y
    public float distancia2 = 1f;     // Movimiento en X
    public float velocidad = 10f;     // Velocidad del movimiento

    private Vector3 posicionInicial;
    private Vector3 posicionPresionada;
    private bool enclavado = false;

    public MovimientoPositivo movimiento;
    public PLC_Comunicacion DatoEntrada;

    void Start()
    {
        posicionInicial = transform.position;
        posicionPresionada = posicionInicial - new Vector3(distancia2, distancia, 0);
    }

    public void EnclavarNeg()
    {

        // Detectar clic con Fire1 y raycast sobre este objeto
        if (Input.GetButtonDown("Fire1"))
        {
            enclavado = !enclavado;
            movimiento.ToggleEnclavado();
            DatoEntrada.EscribirBEM2();
        }

        // Movimiento según estado enclavado
        if (enclavado)
        {
            transform.position = Vector3.Lerp(transform.position, posicionPresionada, Time.deltaTime * velocidad);
        }
        else
        {
            transform.position = Vector3.Lerp(transform.position, posicionInicial, Time.deltaTime * velocidad);
        }
    }
}

```

10. Código del script que anima el cambio de posición del vástago A; posición de extensión / trabajo y de retroceso / reposo.

MovimientoPositivo.cs

Archivos varios — MovimientoPositivo

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

public class MovimientoPositivo : MonoBehaviour
{
    public Vector3 positionA;       // Posición inicial (A)
    public Vector3 positionB;       // Posición final (B)
    public float speed = 5f;        // Velocidad del movimiento

    public bool pulsadorPressed = false;   // Control desde UI o input
    public bool enclavadoActive = false;   // Control desde UI o input

    private Vector3 targetPosition;

    void Start()
    {
        transform.position = positionA;     // Iniciar en A
        targetPosition = positionA;
    }

    void Update()
    {
        // Lógica de movimiento
        if (pulsadorPressed)
        {
            targetPosition = positionB;
        }
        else if (enclavadoActive)
        {
            targetPosition = positionA;
        }

        // Movimiento suave hacia la posición objetivo
        transform.position = Vector3.MoveTowards(transform.position, targetPosition, speed * Time.deltaTime);
    }

    // Métodos para conectar con los botones de UI o eventos
    public void OnPulsadorPressed()
    {
        pulsadorPressed = true;
    }

    public void OnPulsadorReleased()
    {
        pulsadorPressed = false;
    }

    public void ToggleEnclavado()
    {
        enclavadoActive = !enclavadoActive;
    }
}
```

11. Código del script para el intercambio de datos (PLC_Comunicación). *Parte I:*

PLC_Comunicacion.cs*

Archivos varios — PLC_Comunicacion

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using S7.Net;

public class PLC_Comunicacion : MonoBehaviour
{

    public Luz_Verde luzVerd;   // Encendido de led Verde
    public GameObject Sa0;      //Luz Sensor a0
    public GameObject Sa1;      //Luz Sensor a1
    public MovimientoPositivo PosicionVastago;

    private Plc plc;
    private bool bp2M1 = false;
    private bool enclavado = false;
    private bool bp2M2 = false;
    private bool enclavadoM2 = false;
    private bool bCentralM1 = false;

    void Start()
    {
        plc = new Plc(CpuType.S71200, "127.0.0.1", 0, 1);

        try
        {
            plc.Open();
            Debug.Log(" Conectado al PLC.");
        }
        catch
        {
            Debug.LogError(" No se pudo conectar al PLC.");
        }
    }

    void Update()
    {
        if (plc != null && plc.IsConnected)
        {
            try
            {
                // Leer señal desde el PLC (ej: DB1.DBX0.0)
                bool StartVirtual = (bool)plc.Read("DB1.DBX0.0"); //Indicador Verde
                bool StopVirtual = (bool)plc.Read("DB1.DBX0.1");    //Indicador Verde

                bool StartVirtualM2 = (bool)plc.Read("DB1.DBX0.3"); // Activación de sistema; trabajo
                bool StopVirtualM2 = (bool)plc.Read("DB1.DBX0.4");  //Paro de sistema; reposo
                bool Sensora0 = (bool)plc.Read("DB1.DBX0.7");       //Sensor de reposo
                bool Sensora1 = (bool)plc.Read("DB1.DBX1.0");       //Sensor de trabajo
```

Parte II

Archivos varios — PLC_Comunicacion

```
52
53              if (StartVirtual)
54              {
55                  EncenderIV();
56              }
57
58              if (StopVirtual)
59              {
60                  ApagarIV();
61              }
62
63              if (StartVirtualM2)
64              {
65                  Expulsion();
66              }
67
68              if (StopVirtualM2)
69              {
70                  Retroceso();
71              }
72
73              if (Sensora0 == true)
74              {
75                  Sa0.SetActive(true);
76              }
77
78              if (Sensora0 == false)
79              {
80                  Sa0.SetActive(false);
81              }
82
83
84              if (Sensora1 == true)
85              {
86                  Sa1.SetActive(true);
87              }
88
89              if (Sensora1 == false)
90              {
91                  Sa1.SetActive(false);
92              }
93
94              Debug.Log("Señal 1 desde PLC: " + StartVirtual);
95              Debug.Log("Señal 2 desde PLC: " + StopVirtual);
96          }
97          catch
98          {
99              Debug.LogWarning(" Fallo al leer del PLC.");
100         }
101     }
102 }
103
```

Parte III

Archivos varios — PLC_Comunicacion

```
103
104     void OnApplicationQuit()
105     {
106         if (plc != null && plc.IsConnected)
107         {
108             plc.Close();
109         }
110     }
111
112     public void EncenderIV()    // Indicador verde On
113     {
114         luzVerd.EncenderLuzPLC();
115     }
116
117     public void ApagarIV()      // Indicador verde Off
118     {
119         luzVerd.ApagarLuzPLC();
120     }
121
122     public void Expulsion()
123     {
124         PosicionVastago.OnPulsadorPressed();
125     }
126
127     public void Retroceso()
128     {
129         PosicionVastago.ToggleEnclavado();
130     }
131
132     public void EscribirBP2M1() //Señal virtual de paro mesa 1
133     {
134         bp2M1 = !bp2M1;
135         plc.Write("DB2.DBX0.0", bp2M1);
136     }
137
138     public void EscribirBEM1() //Señal virtual de paro mesa 1
139     {
140         enclavado = !enclavado;
141         plc.Write("DB2.DBX0.1", enclavado);    //Paro virtual
142         plc.Write("DB2.DBX0.5", enclavado); //Paro de semáforo
143     }
144
145     public void EscribirBP2M2() //Señal virtual de paro mesa 2
146     {
147         bp2M2 = !bp2M2;
148         plc.Write("DB2.DBX0.2", bp2M2);
149     }
150
151     public void EscribirBEM2() //Señal virtual de paro mesa 2
152     {
153         enclavadoM2 = !enclavadoM2;
154         plc.Write("DB2.DBX0.3", enclavadoM2);    //Paro virtual
155     }
156
157 }
158
```