Proyecto Fin de Máster
Doble Máster de Ingeniería Industrial y Electrónica, Robótica y Automática

Sincronización posicional entre gemelo digital y entorno real mediante redes neuronales

Autor: Francisco Javier Cebolla Verdugo

Tutor: Juan Manuel Escaño González

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Proyecto Fin de Máster
Doble Máster de Ingeniería Industrial y Electrónica, Robótica y Automática

Sincronización posicional entre gemelo digital y entorno real mediante redes neuronales

Autor:
Francisco Javier Cebolla Verdugo

Tutor:
Juan Manuel Escaño González
Profesor Titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Proyecto Fin de Máster: Sincronización posicional entre gemelo digital y entorno real mediante redes neuronales

Autor: Francisco Javier Cebolla Verdugo
Tutor: Juan Manuel Escaño González

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En agradecimiento total hacia todas las personas que me han apoyado durante los años que he estado realizando mis estudios, empezando por mi familia, amigos, compañeros y profesores, en particular Juan Manuel por haberme apoyado desde el principio en realizar el trabajo completo.

F. Javier Cebolla Verdugo
Sevilla, 2024

Resumen

En estos últimos años, ha habido un gran avance respecto al campo de la inteligencia artifical mediante el uso de redes neuronales artificiales, debido a la gran complejidad que pueden tomar y solucionar problemas que hasta ahora eran muy complejos, y a la creación de entornos virtuales para simular procesos complejos. El proyecto trata, mediante el uso de esta herramienta aplicada a visión, crear una red neuronal que pueda identificar la posición de una bandeja dentro de una célula de fabricación, y su dirección a través de los carriles que esta tiene, todo ello para transmitirlo a su gemelo digital en el mundo virtual.

Abstract

In recent years, there has been significant progress in the field of artificial intelligence through the use of artificial neural networks, due to their ability to take on and solve problems that were previously very complex, and the creation of virtual environments to simulate intricate processes. The project aims, through the application of this tool to vision, to create a neural network capable of identifying the position of a tray within a manufacturing cell and its direction along the rails it follows, all in order to transmit this information to its digital twin in the virtual world.

Índice

1 Introducción

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

Claude Shannon, 1948

En el momento actual que nos encontramos, la digitalización, la Industria 4.0 y la convergencia de tecnologías ha transformado la manera en la que se diseñan, realizan y operan los sistemas industriales. La adopción de gemelos digitales se ha convertido en una pieza central de esta transformación, prometiendo una simbiosis sin precedentes entre los mundos físico y digital. La realización de tales sistemas avanzados es compleja, especialmente cuando se entrelazan disciplinas y tecnologías dispares en un único proyecto. La motivación detrás de este trabajo de fin de máster nace de la aspiración de realizar una implementación completa de un gemelo digital, un proyecto que encapsula la creación, la comunicación, y la colaboración multidisciplinaria.

Este capítulo sienta las bases de nuestra trabajo, delineando la justificación y la necesidad de abordar los obstáculos que nos encontramos en la comunicación. Se discutirá la relevancia de los gemelos digitales y la importancia crítica de las comunicaciones eficientes para su operatividad en tiempo real. Además, se examinarán los desafíos actuales que presenta la implementación de estas tecnologías en un entorno con recursos limitados y la complejidad que conlleva la integración de sistemas heterogéneos. A través de este trabajo, se busca crear un concepto que pueda ser aplicado en la célula de fabricación en la que se ha trabajado para

1.1 Motivación del problema

La implementación de proyectos multidisciplinarios representa el mundo actual en el que nos encontramos. La confluencia de distintas áreas de conocimiento, como la impresión 3D, la sensorización avanzada, el procesamiento de datos, y la inteligencia artificial, conforma un desafío de integración. Este trabajo se sitúa en la conclusión de un proyecto de este tipo, enfocándose en la sinergia entre la creación física de componentes y la comunicación interdispositivos para alimentar y entrenar un gemelo digital funcional.

El proceso de dar vida a un gemelo digital implica no solo la replicación virtual de un objeto físico, sino también la captación y procesamiento continuo de datos que alimentan y actualizan su estado en un entorno simulado. En este contexto, la complejidad de los sistemas contemporáneos, caracterizados por su heterogeneidad y la necesidad de interoperabilidad, se convierte en uno de los desafíos más significativos. En el laboratorio, nos enfrentamos a limitaciones de recursos, donde dependiendo de los activos que tengamos, pueda ser incompatible la comunicación o ser más mucho más complejo debido a la limitación en cuanto a versatilidad que tiene un PLC respecto a un ordenador convencional o una Rapsberry Pi, enfrentando y superando las discrepancias que pueden surgir de la incompatibilidad o ausencia de dispositivos específicos.

La integración de sistemas dispares plantea una problemática donde la ausencia de un procedimiento estandarizado evidencia la necesidad de un diseño que, actualmente, llegue a ser funcional bajo una serie de condicionantes, pero con más recursos, se podría optimizar y llegar a realizar un trabajo mucho más flexible. Este trabajo pretende abordar dicha complejidad, proponiendo soluciones que permitan la comunicación

fluida y eficiente entre distintos dispositivos, fundamentales para la alimentación de datos en tiempo real del gemelo digital. La relevancia de este enfoque radica en la premisa de que la calidad y la sincronización de la información son cruciales para la representación precisa y la funcionalidad del gemelo digital.

Por consiguiente, la necesidad de este proyecto emana de la propia arquitectura del sistema multidisciplinar; donde la obtención de datos no se limita a los dispositivos actuadores, sino que se extiende a una red de sensores y nodos interconectados. La comunicación entre estos elementos es la piedra angular que permite que los procesos no solo funcionen, sino que lo hagan de manera coherente y alineada con las expectativas de una simulación y una operación predictiva. Al abordar estos desafíos, este trabajo busca alcanzar una integración funcional y guiar futuros proyectos en la creación de gemelos digitales eficientes y realistas.

1.2 Antecedentes

En este capítulo, se examinará el contexto histórico de los dos elementos principales del proyecto: las comunicaciones industriales y los gemelos digitales. Este análisis proporcionará una comprensión más profunda de cómo estos componentes han evolucionado a lo largo del tiempo y su relevancia en el contexto del trabajo, debido a que han sufrido grandes cambios en un plazo de tiempo muy corto con respecto a otros campos de la ingeniería. A continuación, se presenta un resumen detallado de la historia y evolución de cada una de las partes del proyecto.

1.2.1 Historia de las Comunicaciones Industriales y los PLC

La historia de las comunicaciones industriales está intrínsecamente ligada al desarrollo de los Controladores Lógicos Programables (PLC). Antes de la invención de los PLC, el control de procesos industriales se llevaba a cabo mediante complejos sistemas de relés electromecánicos que ocupaban grandes espacios y requerían mantenimiento continuo. La automatización cableada era inflexible y la resolución de averías, así como las modificaciones en el proceso, eran tareas arduas y prolongadas.

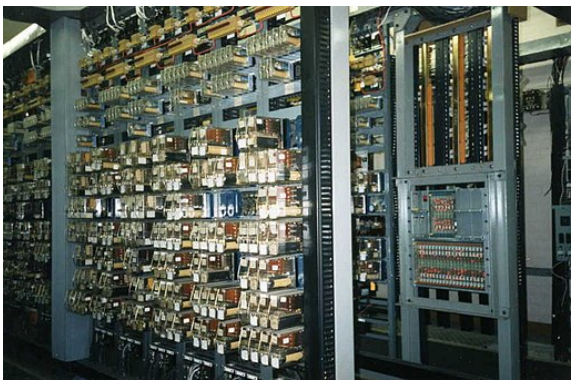

Figura 1.1 Ordenador basado en relés. Fuente: Wikipedia Commons [1].

En 1968, Dick Morley y su equipo en Bedford Associates introdujeron el Modicon 084, el primer prototipo funcional de un PLC, un avance que simplificaría enormemente los sistemas de control industrial. Sin embargo, el verdadero éxito llegó con el Modicon 184, el primer PLC comercialmente exitoso y ampliamente adoptado en la industria.

Con el establecimiento de los PLC, surgió una nueva necesidad: la comunicación entre estos controladores y otros dispositivos en la planta de producción. En respuesta a esto, en 1979, se creó Modbus, uno de los primeros protocolos de comunicación industrial. Modbus permitió la transmisión de información entre dispositivos y PLCs, facilitando así la automatización de la producción en masa.

El último gran avance en el ámbito de las comunicaciones industriales ha sido el desarrollo del protocolo OPC UA (Open Platform Communications Unified Architecture). OPC UA, concebido en la década de 2000, es un protocolo de comunicaciones industrial multidimensional que no solo transmite datos, sino que también facilita la interoperabilidad y la integración segura de máquinas y sistemas dentro de la industria, apoyando la creación de sistemas de automatización más complejos y flexibles.

Figura 1.2 PLC Modicon 184. Fuente: Blog 330 Ohms[2].

1.2.2 Historia y evolución de los Gemelos Digitales

Los gemelos digitales, concebidos como réplicas virtuales de entidades físicas, han evolucionado desde meras representaciones hasta convertirse en sistemas dinámicos capaces de simular en tiempo real el estado y la historia de sus contrapartes físicas. Aunque el concepto fue teorizado en 1991 y formalmente introducido por Michael Grieves en 2002, ha sido a través de la influencia de la NASA y el auge de la Industria 4.0 que los gemelos digitales han encontrado su identidad y aplicación estratégica.

Roberto Saracco, en su exploración de la evolución de los gemelos digitales en la manufactura, destaca el cambio de un modelo estático CAD a una interacción dinámica que modela comportamientos y perfiles, como en los asistentes de voz. Esta adaptabilidad se refleja en la noción de gemelos virtuales, que representan entidades que pueden no existir físicamente pero que interactúan en el espacio cibernético y físico, lo que subraya la creciente complejidad y la función expansiva de los gemelos digitales en la manufactura moderna.

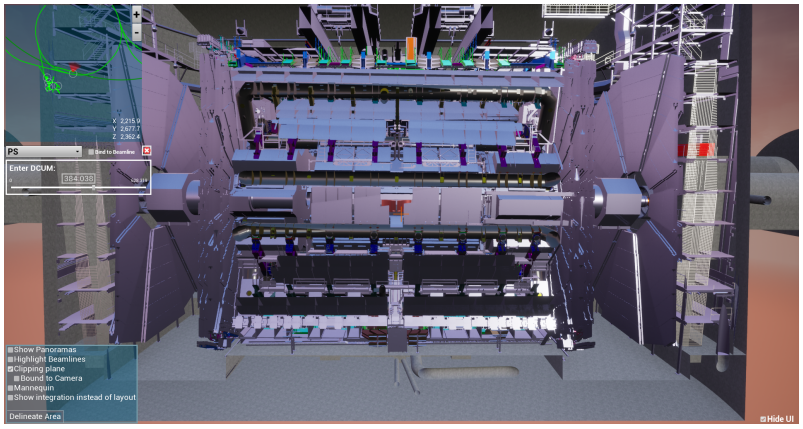

Figura 1.3 Ejemplo de gemelo digital. Fuente: X[3].

La integración del IoT y la recopilación de datos en tiempo real a través del "shadowing" permiten que los gemelos digitales proporcionen una "inteligencia" enriquecida, no solo para instancias específicas, sino también para modelos genéricos, abriendo puertas a nuevos servicios y soporte de operaciones. Este avance ha transformado la manufactura, donde el análisis de datos y la interacción digital- física potencian la eficiencia y la innovación. A continuación enfatizamos en los dos conceptos y en sus aplicaciones en la industria.

1.3 Gemelos Digitales

En este apartado vamos a enumerar parte de los beneficios que se obtienen de la implementación de gemelos digitales en la industria, y del marco normativo en el que se encuentra.

1.3.1 Aplicaciones de Gemelos Digitales en la Industria

La tecnología de gemelos digitales está en constante evolución y mejora, revolucionando la forma en que las industrias diseñan, desarrollan y mantienen sus productos. Esta innovación está siendo impulsada por líderes del sector que buscan mejorar la eficiencia, la seguridad y la sostenibilidad.

Rolls-Royce, en colaboración con la Defence Science and Technology Agency (DSTA) de Singapur, está utilizando gemelos digitales para optimizar el mantenimiento de motores aeroespaciales. Mediante el uso de la Visión Artificial, mejoran la eficiencia de las inspecciones y la ejecución del mantenimiento predictivo.

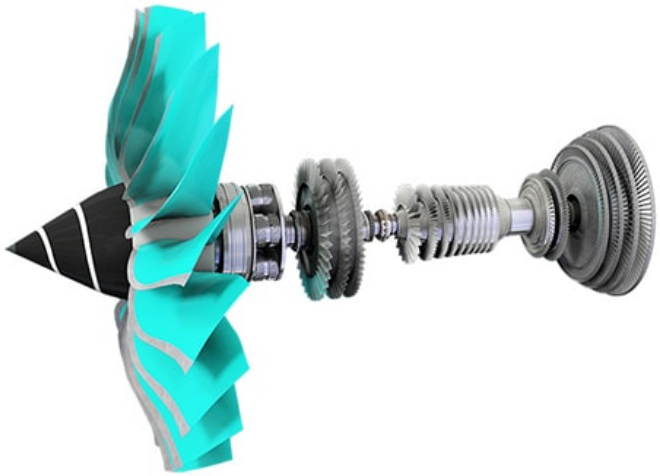

Figura 1.4 Gemelo digital de un motor Rolls Royce. Fuente: Web Rolls Royce[4].

La marca de coches Volvo está aplicando gemelos digitales a lo largo de todo el ciclo de vida de sus vehículos. Utilizan entornos virtuales inmersivos para la colaboración en diseño y desarrollo, lo que les permite simular y probar diversos aspectos del vehículo, desde su aerodinámica hasta la experiencia del usuario, reduciendo costos y acelerando el proceso de desarrollo.

Figura 1.5 Gemelo digital de un Volvo XC40 en Unity. Fuente: Youtube[5].

Estos no son los únicos ejemplos de la aplicación práctica de los gemelos digitales. Otras empresas están explorando cómo esta tecnología puede mejorar la planificación urbana, el diseño de infraestructuras y la experiencia del cliente. Unity, por ejemplo, está facilitando a industrias como la aeroespacial y la automotriz la creación de gemelos digitales interactivos, que no solo se utilizan para el diseño y el mantenimiento, sino

también para mejorar la experiencia del cliente y la planificación estratégica. A continuación se entrará a definir los métodos de comunicación y cómo funcionan mediante el modelo de capa.

1.4 Comunicaciones y modelo de capa

1.4.1 Modelo de Capa

El modelo de capa de comunicaciones es un marco conceptual que describe cómo los datos son transmitidos a través de una red de computadoras. Este modelo se divide en varias capas, cada una de las cuales se encarga de un aspecto específico del proceso de comunicación. A continuación se presenta una descripción de las diferentes capas del modelo OSI (Open Systems Interconnection), que es uno de los modelos de capa más conocidos y utilizados:

- **Capa Física:** Esta es la capa más baja del modelo OSI y se encarga de la transmisión y recepción de los bits sin procesar a través de un medio físico. Incluye especificaciones de cables, tarjetas de red, repetidores y demás hardware relacionado.
- **Capa de Enlace de Datos:** Esta capa proporciona los medios funcionales y procedurales para transferir datos entre entidades de red y detectar y corregir errores que puedan ocurrir en la capa física. Se divide en dos subcapas: la subcapa de control de acceso al medio (MAC) y la subcapa de control lógico de enlace (LLC).
- **Capa de Red:** Su función principal es determinar cómo los datos se enrutan desde el origen hasta el destino a través de una o más redes. Utiliza direcciones lógicas para identificar dispositivos y enrutadores para encontrar la mejor ruta.
- **Capa de Transporte:** Esta capa garantiza que los datos lleguen de manera completa y sin errores al destinatario. Proporciona control de flujo, segmentación y reensamblaje, y manejo de errores mediante protocolos como TCP y UDP.
- **Capa de Sesión:** Gestiona y controla las conexiones entre computadoras. Establece, administra y finaliza sesiones entre aplicaciones locales y remotas.
- **Capa de Presentación:** Se encarga de la traducción de datos entre el formato de la red y el formato que entiende la aplicación. También maneja la compresión y el cifrado de datos.
- **Capa de Aplicación:** Esta es la capa más alta del modelo OSI y proporciona servicios de red directamente a las aplicaciones del usuario. Incluye protocolos como HTTP, FTP, SMTP y otros que permiten la comunicación de datos entre aplicaciones.

Cada una de estas capas desempeña un papel crucial en el proceso de comunicación, asegurando que los datos se transmitan de manera eficiente y confiable desde la fuente hasta el destino.

Modbus y OPC UA son protocolos de comunicación que se ubican principalmente en las capas superiores del modelo OSI, como veremos a continuación.

1.4.2 Protocolos de comunicación

En el ámbito de los sistemas y redes industriales, los protocolos de comunicación desempeñan un papel crucial, actuando como conjuntos de reglas y procedimientos que permiten a los dispositivos y sistemas transmitir información de manera eficiente y coherente. Un protocolo de comunicación es esencialmente un lenguaje común que los dispositivos utilizan para interactuar entre sí. Para ser efectivo, un protocolo debe asegurar la integridad de los datos transmitidos, garantizar la compatibilidad entre diferentes sistemas y dispositivos, y proporcionar medidas de seguridad adecuadas para proteger la información intercambiada. La elección del protocolo de comunicación adecuado es fundamental en entornos industriales, ya que permite la integración fluida de máquinas y sistemas, facilita la automatización de procesos y mejora la eficiencia operativa. En la actualidad, los dos protocolos de comunicación más usados son los siguientes:

OPC UA (Unified Architecture):

- Protocolo de comunicación industrial abierto, independiente de la plataforma y seguro.
- Brinda acceso a datos en tiempo real y alarmas de dispositivos y sistemas industriales, sin restricciones de ubicación o fabricante.

- Basado en una arquitectura orientada a servicios, utilizando un modelo de publicación-suscripción para comunicaciones eficientes y escalables.
- Compatible con una amplia gama de tipos de datos y sistemas, tanto antiguos como modernos.
- Incorpora características avanzadas de seguridad, incluyendo encriptación, autenticación y control de acceso.

Modbus:

- Protocolo de comunicación industrial desarrollado en la década de 1970.
- Ampliamente utilizado en sistemas de control industrial, manufactura, control de procesos y automatización de edificios.
- Su arquitectura cliente-servidor posibilita la comunicación directa entre dispositivos y sistemas.
- Compatible con una diversidad de dispositivos industriales, como controladores programables, sensores y actuadores.
- A diferencia de OPC UA, carece de características de seguridad integradas.

Al elegir un protocolo para la implementación de gemelos digitales, es crucial considerar la seguridad, la compatibilidad y la capacidad de manejar diversos tipos de datos. OPC UA, con su enfoque en la seguridad y la versatilidad, puede ser más adecuado para entornos que requieren una integración y comunicación complejas entre una variedad de dispositivos y sistemas. Adicional a lo anterior, junto con la implementación de comunicación y gemelo digital, se necesitará de un proceso de redes neuronales que realicen el entrenamiento para la sincronización del sistema, como veremos a continuación.

1.5 Introducción a las Redes Neuronales

Las redes neuronales representan una piedra angular en el desarrollo de la inteligencia artificial moderna. Inspiradas en el funcionamiento del cerebro humano, estas redes son sistemas de algoritmos que intentan reconocer patrones y procesar datos de manera análoga a como lo hace nuestro cerebro. Su flexibilidad y capacidad para aprender a partir de grandes cantidades de datos las hacen herramientas poderosas en una variedad de aplicaciones, desde el reconocimiento de voz hasta la predicción de tendencias en datos financieros.

Una red neuronal se compone de nodos, o "neuronas", organizados en capas. Cada nodo recibe entradas, realiza cálculos simples y pasa su salida a los nodos de la siguiente capa. La "profundidad" de estas redes, es decir, el número de capas ocultas entre la entrada y la salida, es lo que define al aprendizaje profundo (deep learning), un subcampo del aprendizaje automático que ha ganado notoriedad por su capacidad para realizar tareas complejas de manera eficaz.

En este trabajo, exploraremos el potencial de las redes neuronales en el marco del aprendizaje por refuerzo, una forma de aprendizaje automático donde un agente aprende a tomar decisiones mediante la experimentación y la recepción de recompensas o penalizaciones. Este enfoque será aplicado para entrenar al gemelo digital y comprobar la dificultad o el uso del mismo en el entorno virtual, para poder extraer información del gemelo para poder llegar a aplicarse en el entorno real.

1.5.1 El problema del aprendizaje

El aprendizaje es un proceso intrínseco a la inteligencia, tanto natural como artificial. En el contexto de la inteligencia artificial, el aprendizaje se refiere a la capacidad de un sistema para mejorar su desempeño en una tarea específica a través de la experiencia. Este proceso es central en áreas como el aprendizaje automático (machine learning) y el aprendizaje profundo (deep learning), donde los algoritmos ajustan sus parámetros internos, generalmente pesos en una red neuronal, para minimizar una función de coste o maximizar una función de recompensa.

En el aprendizaje supervisado, se proporciona al sistema un conjunto de ejemplos etiquetados, y el objetivo es aprender una función general que mapee las entradas a las salidas deseadas. Sin embargo, el aprendizaje no supervisado intenta encontrar patrones y estructuras ocultas en datos no etiquetados. Por otro lado, el aprendizaje por refuerzo se centra en cómo los agentes deben tomar acciones en un entorno para maximizar alguna noción de recompensa acumulativa.

El "problema del aprendizaje" abarca varios desafíos, como el sobreajuste, donde un modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a datos nuevos; la selección de características, que implica determinar qué información es relevante para la tarea; y la escalabilidad, ya que los algoritmos deben ser capaces de aprender eficientemente a medida que la cantidad de datos crece. Además, está el desafío de la transferencia de aprendizaje, donde un sistema debe aplicar lo aprendido en un contexto a tareas relacionadas pero diferentes.

Se muestra en el siguiente apartado el contexto en el que este proyecto se encuentra, ya que tiene una relación estrecha con un proyecto al mismo, del que partimos para poder realizar este nuevo trabajo.

1.6 Célula de fabricación

En este proyecto se usará la una de las células de fabricación existentes en las instalaciones de la Universidad de Sevilla, dentro del laboratorio de la Escuela Técnica Superior de Ingeniería, ubicada en la planta baja del departamento de Ingeniería de Sistemas y Automática. Esta célula manufacturera, que se compone de varias cintas y de distintas estaciones de trabajo que se explicarán con más detalle en el capítulo 3, crea un complejo sistema que reúne todas las características necesarias para simular un entorno de manufactura, ideales para desarrollar proyectos piloto como el aquí desarrollado y realizar un modelo digital completo y complejo con múltiples funcionalidades y una complejidad suficiente.

Este proyecto, como veremos a continuación, es el punto de partida para el control y creación de un modelo digital, que tenga las mismas propiedades y mecanismos que la célula de fabricación en el mundo real, y que como veremos en la siguiente sección, trata de obtener datos en tiempo real mediante el uso de visión de la célula de fabricación que vamos a usar en el proyecto.

1.7 Relación con trabajo de detección mediante deep learning

El desarrollo de este proyecto tiene fundamentos sólidos en trabajos anteriores sobre visión por computadora y deep learning. Un aspecto crítico abordado anteriormente es la detección de objetos utilizando redes neuronales profundas, lo que es central para la alimentación de datos hacia el gemelo digital de nuestro sistema actual.

Se han explorado técnicas como YOLO (You Only Look Once) y SSD (Single Shot Multibox Detector) para el reconocimiento y seguimiento de objetos en tiempo real, que son esenciales para la captura de datos en ambientes dinámicos. Estas tecnologías permiten la estimación precisa de la posición y velocidad de objetos, lo que es crucial para la sincronización y precisión de los gemelos digitales en simulaciones virtuales.

> “El proyecto trata, mediante el uso de herramientas de inteligencia artificial aplicadas a la visión, de analizar varios tipos de redes neuronales que identifican la posición de objetos dentro de una célula de fabricación y su dirección a través de los carriles, transmitiendo esta información a su gemelo digital en el mundo virtual.”

En el contexto de nuestra investigación, estas aplicaciones no solo refuerzan la capacidad de interacción entre el mundo físico y el digital, sino que también mejoran la eficacia con la que se realizan las simulaciones y predicciones de comportamiento en entornos controlados.

La integración de estos métodos avanzados de aprendizaje profundo y seguimiento de objetos fortalece la precisión de los modelos virtuales, facilitando un desarrollo más ágil y eficiente de soluciones en la manufactura digital. Estos avances representan un paso significativo hacia la optimización de procesos en tiempo real y la reducción de errores en la fase de diseño y prueba de nuevos sistemas.

1.7.1 Primera fase del proyecto de integración con gemelo digital

Para comprender el proyecto global que se intenta realizar en este trabajo, se describe a continuación parte de las fases previas del proyecto de captación de datos y entrenamiento de redes neuronales artificiales, que nos ayudará a realizar la continuación del mismo.

Definición del problema a resolver

La primera fase del proyecto, tras la definición de materiales a usar en el mismo y la definición del problema a resolver, se obtuvo los datos iniciales de la célula de fabricación, como son las distancias dentro del mismo, longitud de las cintas de transporte, ubicación de los botones de referencia para ubicar el sistema de referencia

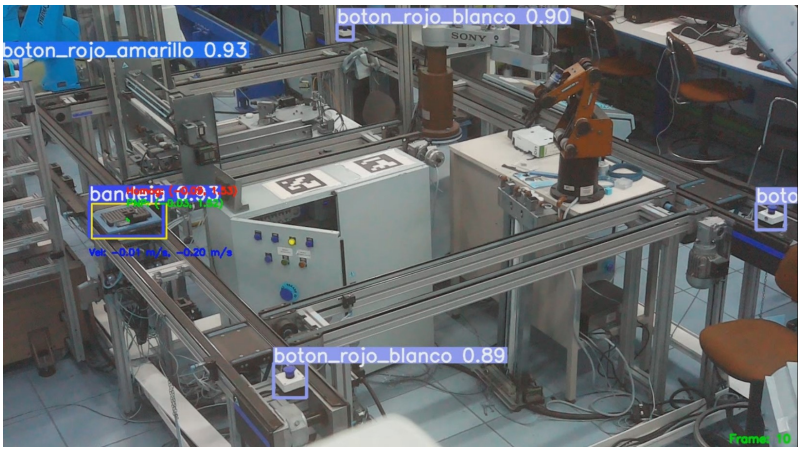

Figura 1.6 Obtención de datos en tiempo real de la célula de fabricación.

del proyecto, tamaño de la bandeja y de los objetos a ubicar encima de ella. Todos estos datos son importantes de conocer debido a que no solo se usan en el proyecto previo, si no que son datos que tenemos que usar para definir y concretar el diseño en el enotorno virtual del gemelo.

Creación de datos para su entrenamiento

Con los datos de posición y velocidad ya procesados y validados, la segunda fase del proyecto abordó el problema del entreno supervisado. En esta etapa se creó la base de datos tras la captura de muchas imágenes en el laboratorio para indicar a la red neuronal que patrón era una bandeja, por lo que se necesitó de un gran número de imágenes y variaciones de la misma para poder lograr el objetivo.

Creación de la estructura de la red neuronal

En la siguiente etapa, una vez ya recopilado toda la base de datos de imágenes de bandejas a través de la cámara, probamos los tipos de redes neuronales que vamos a modelar para obtener la localización de las mismas en nuevas imágenes no usadas para el entrenamiento..

Retroalimentación y Mejora Continua

Los resultados de estos entrenamientos iniciales fueron analizados para identificar áreas de mejora y ajustar el modelo. Esto creó un ciclo de retroalimentación continuo donde los se añadieron más imágenes para lograr mejor ajuste y precisión durante la validación del entrenamiento de la red neuronal.

Conclusión del proyecto

Una vez se entrenó al modelo y se identificaba correctamente, mediante triangulación y óptica se obtuvo la dirección del movimiento, posición y velocidad de la bandeja, datos necesarios para poder alimentar a este trabajo. Este trabajo continúa en este punto, en el que ya tenemos unos datos que pueden ser usados para multitud de problemas, en particular para nuestro trabajo, para el seguimiento de una referencia móvil.

2 Norma ISO 23247

En el marco de la revolución digital, uno de los conceptos que ha experimentado un avance significativo es el de los **gemelos digitales**. Este término se refiere a la creación de réplicas digitales precisas de entidades físicas, que pueden ir desde objetos individuales hasta sistemas complejos y entornos completos. Estos modelos digitales dinámicos, actualizados en tiempo real, permiten simular, predecir y visualizar cambios en sus contrapartes físicas.

El concepto de gemelo digital, inicialmente desarrollado por la NASA para la industria aeroespacial, ha ganado relevancia en los últimos años gracias a la convergencia de tecnologías avanzadas como el Internet de las Cosas (IoT), Big Data, Inteligencia Artificial (IA), aprendizaje automático y la computación en la nube. Estos avances han permitido su aplicación en diversos sectores, mejorando la eficiencia, flexibilidad y adaptabilidad de los procesos industriales.

2.1 Norma ISO 23247

Las normas ISO, especialmente la serie **ISO 23247:2021**, establecen los principios generales para el desarrollo y aplicación de gemelos digitales en la fabricación. Estas normas abarcan aspectos clave como la precisión en la representación de datos, la comunicación efectiva entre el modelo digital y su contraparte física, y la integridad y seguridad de los datos. Además, enfatizan la importancia de la extensibilidad y la granularidad de los modelos digitales, permitiendo su adaptación y escalabilidad en distintos entornos de fabricación.

2.1.1 Estructura de la Norma ISO 23247

ISO 23247-1:2021 - Principios Generales

Esta parte proporciona una visión general y los principios fundamentales del marco de gemelos digitales. Enfatiza la precisión en la representación de datos y la necesidad de una comunicación eficaz entre el modelo digital y su contraparte física. Se describen aplicaciones como el control en tiempo real, el mantenimiento predictivo, y el análisis fuera de línea, destacando los beneficios de los gemelos digitales en la mejora de la eficiencia y la gestión de riesgos.

- **Concepto de Gemelo Digital**: Un gemelo digital es una representación digital precisa de un elemento de fabricación observable (OME), con sincronización entre el elemento y su representación digital. El gemelo digital puede existir a lo largo de todo el ciclo de vida del producto, aprovechando datos del entorno virtual y físico para mejorar el rendimiento del sistema de fabricación.

- **Aplicaciones de Gemelos Digitales**: Incluyen el control en tiempo real, análisis fuera de línea, mantenimiento predictivo, verificación de la salud, diseño de ingeniería, control de producción y vigilancia por video. Específicamente, las aplicaciones de control en tiempo real permiten monitorizar los OMEs y realizar ajustes en el proceso de fabricación según sea necesario. El análisis fuera de línea permite comparar los gemelos digitales de múltiples OMEs para determinar tendencias y cambios, y hacer recomendaciones para futuros procesos de fabricación.

- **Beneficios**: Mejoras en la planificación y validación en bucle, aseguramiento de la programación de producción, comprensión mejorada de los elementos de fabricación, gestión dinámica de riesgos, trazabilidad de partes y ensamblajes, y trazabilidad de procesos. La planificación en bucle puede utilizarse para resecuenciar y ajustar dinámicamente un proceso de fabricación durante la producción en respuesta a excepciones en el piso de fábrica.

- **Elementos de Fabricación Observables**: Personal, equipos, materiales, procesos, instalaciones, ambiente, productos y documentos de soporte. Cada uno de estos elementos puede tener gemelos digitales que modelen aspectos específicos, como la disponibilidad y nivel de certificación del personal, o las características de estado y rendimiento de los equipos.
- **Principios Generales**: Se detallan aspectos como la precisión, comunicación, adquisición de datos, análisis de datos, integridad de los datos, extensibilidad, granularidad, identificación, gestión, ciclo de vida del producto, seguridad, simulación y sincronización. La precisión asegura que el gemelo digital describa el estado de su OME con el nivel de fidelidad adecuado, mientras que la comunicación y sincronización aseguran que el gemelo digital y el OME se actualicen mutuamente de manera continua.

ISO 23247-2:2021 - Arquitectura de Referencia

Define una arquitectura de referencia para la implementación de gemelos digitales. Esta parte especifica los componentes y las interacciones necesarias para crear un entorno de gemelos digitales cohesivo. Incluye modelos de referencia basados en dominios y entidades, y proporciona una vista funcional que detalla las entidades funcionales necesarias para la implementación de gemelos digitales.

- **Modelos de Referencia**: Basados en dominios y entidades, incluyendo el dominio del usuario, dominio del gemelo digital, dominio de comunicación del dispositivo y el dominio de fabricación observable. Estos modelos de referencia facilitan la organización y estructura de los componentes necesarios para crear gemelos digitales efectivos.
- **Vista Funcional**: Incluye entidades funcionales como la comunicación de dispositivos, entidades de gemelos digitales y entidades de usuarios, abarcando la recolección de datos, control de dispositivos, simulación, análisis y acceso a recursos. Las entidades funcionales también incluyen la visualización y presentación de datos, la sincronización de datos en tiempo real, y el almacenamiento y archivo histórico de datos.
- **Objetivos y Alcances**: Detalla objetivos como la sincronización, visualización, presentación, archivo histórico, análisis de datos, simulación y optimización. Estos objetivos aseguran que los gemelos digitales puedan proporcionar una visión integral y actualizada de los procesos de fabricación, facilitando la toma de decisiones informada y la optimización continua de los procesos.

ISO 23247-3:2021 - Atributos de Información

Establece los atributos de información básicos necesarios para los elementos de fabricación observables. Incluye detalles sobre los datos necesarios para representar digitalmente estos elementos, asegurando que la información sea precisa y consistente para facilitar la interoperabilidad entre sistemas.

- **Atributos Básicos**: Datos sobre personas, equipos, materiales, procesos, instalaciones, ambientes, productos y documentos de soporte. Estos atributos aseguran que cada elemento de fabricación observable esté representado de manera precisa y completa en el gemelo digital.
- **Modelado de Datos**: Guías para el modelado de datos, asegurando precisión y consistencia para facilitar la interoperabilidad entre sistemas. Métodos para la adquisición, análisis, integridad y gestión de datos. El modelado de datos incluye la definición de formatos de datos, estándares de comunicación y protocolos de sincronización.

ISO 23247-4:2021 - Requisitos Técnicos para el Intercambio de Información

Establece los requisitos técnicos para el intercambio de información entre las entidades dentro de la arquitectura de referencia. Proporciona detalles sobre los protocolos y métodos de comunicación necesarios para asegurar una integración eficaz. Incluye ejemplos específicos y casos de uso que demuestran la aplicación práctica del marco de gemelos digitales en la fabricación.

- **Intercambio de Información**: Protocolos de comunicación, formatos de datos y métodos de sincronización para asegurar la integridad y seguridad de los datos. Los protocolos de comunicación deben ser robustos y seguros, permitiendo la transferencia eficiente de grandes volúmenes de datos en tiempo real.
- **Casos de Uso y Ejemplos**: Ejemplos específicos como la optimización de operaciones de remoción de material y la predicción del desgaste de componentes, demostrando la aplicación práctica del marco de gemelos digitales en la fabricación. Estos casos de uso incluyen la integración de sensores y sistemas de monitoreo para proporcionar una visión precisa del estado actual y la tasa de desgaste de componentes, permitiendo el mantenimiento preventivo y la optimización continua de los procesos.

2.1.2 Implementación y Desafíos

La implementación de gemelos digitales según las normas ISO 23247 requiere un enfoque sistemático que incluya la selección de métodos de identificación, estándares y tecnologías para la recopilación de

datos, control de OMEs, representaciones digitales, y comunicación entre niveles. Algunos de los desafíos incluyen la integración de sistemas heterogéneos, la gestión de grandes volúmenes de datos, y la necesidad de mantener una sincronización precisa entre los modelos digitales y sus contrapartes físicas. La norma destaca la importancia de la precisión, la seguridad, y la extensibilidad en la creación de gemelos digitales.

Detalles Adicionales

- **Sincronización**: La sincronización entre el gemelo digital y el OME debe ser continua y precisa, utilizando métodos basados en eventos o en tiempo. Esto asegura que el gemelo digital refleje con exactitud el estado actual del OME.
- **Simulación y Predicción**: Los gemelos digitales deben ser capaces de simular diferentes escenarios y predecir futuros estados del OME. Esto es esencial para la planificación de mantenimiento y la mejora continua de los procesos de fabricación.
- **Seguridad de los Datos**: La seguridad es un aspecto crítico. Los gemelos digitales deben comunicarse únicamente con recursos autorizados y asegurar la integridad y confidencialidad de los datos intercambiados.
- **Modelado Jerárquico**: Los gemelos digitales deben ser capaces de modelar cualquier nivel de la jerarquía funcional y basada en roles definida en la norma IEC 62264-1.
- **Extensibilidad y Escalabilidad**: Los gemelos digitales deben ser extensibles para adaptarse a nuevas aplicaciones y escalables para manejar diferentes tamaños y complejidades de sistemas de fabricación.
- **Ciclo de Vida del Producto**: Los gemelos digitales deben soportar la continuidad de la información a lo largo del ciclo de vida del producto, incluyendo diseño, planificación, fabricación y mantenimiento.
- **Casos de Uso Detallados**: Los anexos de las normas proporcionan casos de uso detallados que ilustran cómo implementar gemelos digitales en diferentes escenarios de fabricación. Por ejemplo, la optimización de operaciones de remoción de material y la predicción del desgaste de componentes.
- **Interoperabilidad**: La interoperabilidad entre sistemas heterogéneos es crucial para el éxito de la implementación de gemelos digitales. Las normas proporcionan guías sobre cómo asegurar que diferentes sistemas y componentes puedan trabajar juntos de manera efectiva.

3 Arquitectura del sistema

En este capítulo, se va a abordar los elementos tanto físicos como digitales que han sido necesarios para nuestro proyecto. Se detallan en particular las especificaciones de cada uno ya que son elementos necesarios para que el proyecto pueda ser realizado correctamente.

3.1 Dispositivos Hardware

3.1.1 Raspberry Pi 4

La Raspberry Pi 4 representa un salto cualitativo en la serie de microordenadores Raspberry Pi, ofreciendo mayores capacidades y flexibilidad. Aunque actualmente no es la versión más potente, ya que recientemente ha salido la quinta generación de la marca, este modelo sigue teniendo unas caracterísiticas más que notables para ser un dispositivo de bajo costo. Se detallan sus características y el propósito específico que cumple en el sistema:

Figura 3.1 Raspberry Pi 4 con módulo de cámara integrado.

Procesador: Equipada con un procesador ARM Cortex-A72 de 64 bits y una frecuencia de reloj que puede alcanzar los 1.5 GHz.

Memoria RAM: Con 8 GB de RAM LPDDR4, la Raspberry Pi 4 está preparada para soportar aplicaciones que requieren un gran volumen de memoria.

Conectividad: La conectividad robusta es fundamental para nuestra Raspberry Pi 4, que incluye puertos USB 2.0 y 3.0, HDMI, Ethernet Gigabit y, en este modelo, conectividad inalámbrica Wi-Fi y Bluetooth. Esta

multitud de opciones de conexión permite una integración flexible y confiable dentro de la infraestructura que se pretende realizar.

Almacenamiento: Utilizamos una tarjeta microSD de 128 GB para el almacenamiento del sistema operativo y los datos.

Sistema Operativo: Se ha optado por Raspbian debido a su estabilidad, soporte comunitario y optimización para el hardware de la Raspberry Pi.

Limitaciones: Se ha constatado que a la hora de realizar inferencia sobre redes neuronales, la Raspberry no ha conseguido realizar el procesamiento en un tiempo decente, por lo que, no se va a iniciar el gemelo digital en el mismo.

3.1.2 Ordenador de alta gama

El portátil utilizado en este proyecto es un dispositivo de alta gama diseñado para manejar tareas de computación intensiva y aplicaciones gráficas avanzadas. Se detallan sus características más destacadas:

Procesador: Equipado con un Intel® Core™ i7-8750H, el portátil ofrece una frecuencia base de 2.2 GHz y puede alcanzar velocidades significativamente mayores, alrededor de 4.0 GHz.

Memoria RAM: Cuenta con 16 GB de memoria DDR4 a 2133MHz.

Almacenamiento: Para el almacenamiento de datos, el portátil combina un disco duro de 1TB que opera a 5400 RPM con un disco SSD NVMe de 256GB.

Gráficos: El portátil incluye una tarjeta gráfica Nvidia Geforce GTX1070 con 8GB de memoria GDDR5.

Sistema Operativo y Software: El sistema operativo usado es Windows 10.

Este equipo proporciona una potencia y una flexibilidad excepcionales para tareas que van desde el desarrollo de software hasta el procesamiento de imágenes y vídeos, lo que lo convierte en la mejor opción tanto para realizar cualquier trabajo intensivo que se vaya a necesitar.

3.1.3 Serie PLC Modicon M340

La serie PLC Modicon M340 de Schneider Electric representa una solución avanzada en automatización y control para aplicaciones industriales. Destaca por su flexibilidad, rendimiento y capacidad de integración con una amplia gama de módulos y dispositivos, incluyendo procesadores dedicados y una plataforma de módulos Modicon X80, además de módulos adicionales para aplicaciones específicas.

CPS 2000

El módulo CPS 2000, parte de la serie Modicon X80, es un componente crucial para los PAC Modicon M580 y M340. Este módulo de alimentación maneja una tensión primaria de 100V a 240V CA y proporciona una alimentación secundaria de 16,8 W a 24 V CC. Con una corriente de tensión secundaria de 2,5 A a 3,3 V CC y una disipación de potencia máxima de 8,5 W, es ideal para aplicaciones de procesos medianos y grandes. Certificado por normas internacionales, este módulo es robusto y de alta calidad, con clasificación IP20 y un peso de 0,3 kg.

P34 2020

El módulo de procesador P34 2020 de la serie Modicon M340 es esencial para los PAC de Schneider Electric. Soporta hasta 4 racks, con una configuración máxima de 1024 E/S digitales y 256 E/S analógicas en configuraciones de multirrack. Con 4096 KB de RAM, dos conectores RJ45 para Ethernet Modbus/TCP y enlace serie, y una función de duplicación automática de datos, este módulo combina rendimiento y flexibilidad. Su diseño compacto y peso de 0.205 kg lo hacen compatible con una amplia gama de aplicaciones industriales.

DDI 6402K

El módulo BMXDDI6402K, con 64 canales de entrada discreta y un suministro de 24V CC, opera eficientemente en un rango de temperatura de 0°C a +60°C. Su robustez se ve reforzada por una resistencia de aislamiento superior a 10 MOhm a 500 V CC y una potencia disipada de 4,3 W. Su tiempo de respuesta va de 4 ms a 7 ms, lo que garantiza una operación eficiente y rápida.

DDO 6402K

El módulo BMXDDO6402K es un módulo discreto de salida de transistor con 64 canales de salida y un suministro de 24V DC. Diseñado para condiciones ambientales exigentes, funciona en un rango de temperatura de 0°C a +60°C. Este módulo asegura un funcionamiento seguro y eficiente, con protecciones contra diversas contingencias eléctricas y un tiempo medio entre fallos (MTBF) impresionante.

AMI 0410

El módulo BMXAMI0410 de entrada analógica aislada cuenta con 4 canales y se adapta a varios rangos de tensión o corriente. Capaz de funcionar en un rango de temperatura de 0 °C a +60 °C, ofrece un ambiente único para la implementación de módulos de E/S en arquitecturas de automatización. Con clasificación IP20, peso de 0,143 kg y cumplimiento de normativas internacionales, este módulo es ideal para una variedad de aplicaciones industriales.

Todos los dispositivos se pueden ver en la Figura **??**.

Figura 3.2 PLC Modicon M340 usado en el proyecto.

3.2 Dispositivos Software

3.2.1 Unity

Unity es un motor de desarrollo de juegos ampliamente reconocido y utilizado en la industria del videojuego y en aplicaciones de realidad virtual y aumentada. Proporciona un entorno integrado donde los desarrolladores pueden construir experiencias interactivas en 2D y 3D. La plataforma destaca por su versatilidad y capacidad de exportar juegos y aplicaciones a múltiples plataformas, incluyendo Windows, macOS, iOS, Android y consolas de juegos. Unity también es popular en sectores fuera del entretenimiento, como la educación, la arquitectura y la ingeniería, debido a su capacidad para crear simulaciones detalladas y entornos interactivos.

Además de su amplio soporte para gráficos, físicas y sistemas de audio, Unity se integra con una variedad de herramientas y servicios que potencian su utilidad. Esto incluye la integración con sistemas de control de versiones, soporte para realidad aumentada y realidad virtual, y herramientas para la monetización y análisis de juegos y aplicaciones. Su entorno de desarrollo flexible y el soporte para scripts en C# lo hacen accesible tanto para principiantes como para profesionales experimentados.

3.2.2 Machine Control Expert - Schneider

EcoStruxure Control Expert de Schneider es un software diseñado para el desarrollo, configuración y mantenimiento de sistemas de automatización industrial. Esta herramienta se centra en proporcionar una plataforma eficiente para la programación de controladores lógicos programables (PLC) y otros dispositivos de automatización. Ofrece un entorno de programación intuitivo, facilitando la implementación de soluciones de control complejas.

Además, Machine Control Expert soporta una amplia gama de funciones, incluyendo la simulación de procesos, diagnósticos en tiempo real y opciones avanzadas de configuración. Esta flexibilidad lo convierte en una solución ideal para una variedad de aplicaciones industriales, desde la manufactura hasta el manejo de procesos y maquinaria.

3.2.3 Pycharm

Pycharm es un entorno de desarrollo integrado (IDE) específicamente diseñado para la programación en Python. Ofrece herramientas avanzadas para la edición de código, depuración, análisis de código, integración con sistemas de control de versiones y soporte para desarrollo web con Django.

3.2.4 Visual Studio

Visual Studio, desarrollado por Microsoft, es un poderoso IDE que soporta múltiples lenguajes de programación como C#, C++, y Visual Basic. Es ampliamente utilizado para el desarrollo de aplicaciones de escritorio, móviles y web, así como para servicios en la nube.

3.2.5 OPC UA Simulation Server

El OPC UA Simulation Server es una herramienta de simulación que permite a los desarrolladores probar y depurar aplicaciones de automatización y control en un entorno virtual. Proporciona una forma eficiente de simular datos y comportamientos de dispositivos en redes de automatización industrial.

3.2.6 Protocolos de comunicación

Los protocolos de comunicación usados en este proyecto, como describimos en el inicio de la memoria, serán tanto el protocolo Modbus, como OPC UA, y TCP/IP para realizar la transmisión de datos entre distintos programas y dispositivos.

3.2.7 Lenguajes de programación

C#

C# (pronunciado como C Sharp) es un lenguaje de programación moderno, orientado a objetos, desarrollado por Microsoft como parte de su plataforma .NET. Lanzado en 2000, C# fue diseñado para combinar la robustez y la eficiencia de C++ con la simplicidad de Visual Basic. Este lenguaje es conocido por su claridad sintáctica, lo que facilita la escritura de código limpio y comprensible.

C# es ampliamente utilizado en el desarrollo de software para Windows, aplicaciones web (a través de ASP.NET), juegos (usando Unity), y en aplicaciones móviles (a través de Xamarin). Su integración con el entorno de desarrollo de .NET y su soporte para programación funcional y concurrente lo convierten en un lenguaje versátil y poderoso para una amplia gama de aplicaciones.

Python

Python es un lenguaje de programación de alto nivel, interpretado, lanzado por primera vez en 1991 por Guido van Rossum. Se caracteriza por su énfasis en la legibilidad del código y su sintaxis que permite a los programadores expresar conceptos en menos líneas de código que serían posibles en otros lenguajes. Python es utilizado en una amplia gama de aplicaciones, desde desarrollo web hasta ciencia de datos, inteligencia artificial y aprendizaje automático.

La comunidad de Python es una de sus mayores fortalezas, ofreciendo una vasta cantidad de paquetes y módulos de software libre que amplían su funcionalidad. Python es apreciado por su versatilidad y capacidad de integración, siendo una elección popular tanto para prototipos rápidos como para sistemas de producción a gran escala.

Ladder (LD)

Ladder Logic (LD) es un lenguaje de programación gráfico utilizado en la programación de controladores lógicos programables (PLC). Inspirado en los diagramas de relés eléctricos, Ladder Logic fue uno de los primeros lenguajes desarrollados para PLCs y sigue siendo ampliamente utilizado en la automatización industrial. Su representación gráfica de circuitos eléctricos facilita la comprensión y el diagnóstico por parte de los electricistas y técnicos.

Ladder Logic es especialmente efectivo para aplicaciones de control secuencial y es altamente valorado por su simplicidad y facilidad de uso. A pesar de su sencillez, es un lenguaje poderoso que puede ser utilizado para construir sistemas de control complejos, siendo una herramienta esencial en la industria de la automatización.

Structured Text (ST)

Structured Text (ST) es un lenguaje de programación de alto nivel diseñado para programar PLCs. Es uno de los cinco lenguajes de programación definidos en la norma IEC 61131-3 para la automatización de procesos

industriales. ST es similar en sintaxis a Pascal, C y otros lenguajes de alto nivel, lo que lo hace accesible para programadores con experiencia en estos lenguajes.

Structured Text es adecuado para aplicaciones que requieren cálculos complejos, algoritmos de control y tareas que no se adaptan bien a lenguajes gráficos como Ladder Logic. Su capacidad para manejar datos complejos y estructuras de control lo hace particularmente útil en aplicaciones de automatización avanzadas.

3.3 Célula de Fabricación

Se va a realizar el uso de la célula de fabricación, la cual se compone de los siguientes elementos:

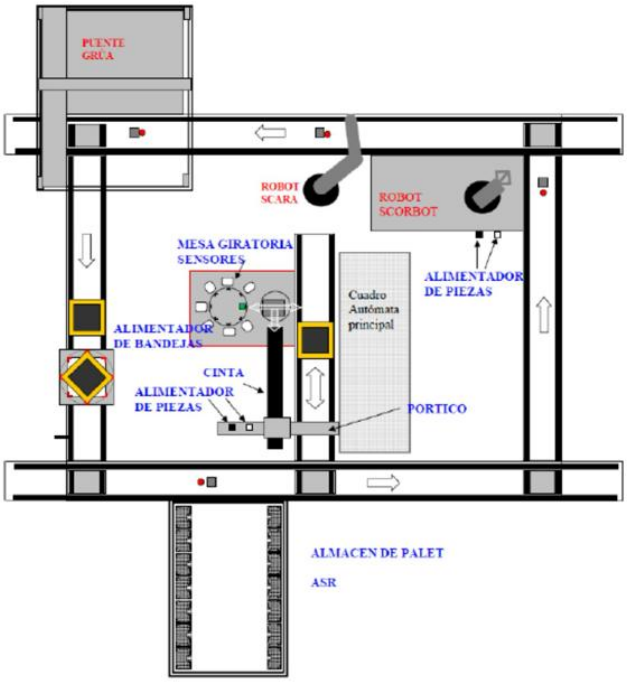

Figura 3.3 Plano general de la célula de fabricación.

- **Pórtico para manejo de piezas:** Sistema de pórtico que incluye una estructura sobre raíles con un brazo mecánico o robótico para trasladar piezas que se encuentren encima de las bandejas.
- **Mesa de Trabajo 1 y Mesa de Trabajo 2:** Espacios equipados para realizar manipulación de los objetos que se encuentran encima de las bandejas, modificaciones o inspecciones de las mismas.
- **Cuadro del Alimentador de Bandejas:** Panel de control para un sistema que organiza y suministra los palets que se situan encima de las bandejas para la producción.
- **Alimentador de Piezas:** Dispositivo que suministra las piezas a la línea de producción.
- **Mesa Giratoria con Sensores:** Mesa equipada con sensores para posicionar y realizar modificaciones en las piezas.
- **Cintas Transportadora de Piezas:** Sistema de transporte continuo para el traslado de piezas a la mesa giratoria, o a la cinta de bandejas.
- **Almacén de Pallet (ASR):** "Automated Storage and Retrieval System" utilizado para el almacenamiento y recuperación automatizados de pallets o bandejas.
- **Cuadro Autómata Principal:** Panel de control central en el que se encuentra el PLC y demás controladores para controlar y/o comunicar los dispositivos.

Se muestra en la figura 3.4 una imagen en perspectiva de la célula en el que se puede observar las dimensiones de la misma.

Figura 3.4 Plano general de la célula de fabricación.

4 Metodología y aplicación de algoritmos

En este capítulo se va a tratar por partes el proceso que se ha realizado para realizar este trabajo. Se va a dividir en función de la metodología empleada, por lo que se empezará por los primeros pasos del proyecto hasta la conclusión final del mismo:

4.1 Generación de datos Online/Offline

La generación y recopilación de datos son fundamentales en el desarrollo, entrenamiento y operación de gemelos digitales. Esta sección se enfoca en comparar los dos enfoques principales para el desarrollo del mismo: la generación de datos en tiempo real (online) y la generación de datos de manera offline o simulada. A continuación, se presentan las ventajas y desventajas de cada método.

4.1.1 Datos en Tiempo Real (Online)

Ventajas:

- *Actualización Continua:* Los datos online ofrecen información actualizada constantemente, lo que es crucial para la toma de decisiones en tiempo real y el monitoreo dinámico de sistemas.
- *Precisión Operacional:* Al reflejar las condiciones actuales del sistema, los datos online permiten un análisis más preciso del comportamiento y el rendimiento del sistema.
- *Respuesta Rápida:* Facilita la detección y respuesta inmediata a problemas o cambios en el sistema, mejorando la eficiencia operativa.
- *Mejora del Aprendizaje Automático:* Los datos actualizados mejoran la precisión de los modelos de aprendizaje automático, permitiendo ajustes y mejoras en tiempo real.

Desventajas:

- *Dependencia de la Infraestructura:* Requiere una infraestructura robusta y confiable para la recopilación y transmisión de datos. En este caso, se ha encontrado funcionamiento deficiente en algunos sensores de la célula de fabricación y en una de las cintas de la misma, por lo que el entrenamiento y la captación de datos puede verse afectado por estos problemas.
- *Vulnerabilidad a Fallas:* Los sistemas online son susceptibles a interrupciones, lo que puede afectar la recopilación de datos.
- *Errores de precisión:* Debido a que la posición de las bandejas se estima en función de una única cámara, en puntos donde existe oclusión del objeto la estimación de la posición puede verse afectada y variar considerablemente respecto a la real.

4.1.2 Datos Offline o Simulados

Ventajas:

- *Control y Repetibilidad:* Los datos offline permiten un mayor control sobre las condiciones experimentales y son ideales para pruebas repetibles.

- *Seguridad:* Menor riesgo de interrupción o de comprometer la integridad del sistema en operación.
- *Análisis en Profundidad:* Ofrecen la oportunidad de realizar análisis más detallados y depurar los ensayos en función de las necesidades o del enfoque necesario.

Desventajas:

- *Verosimilitud de Datos:* Los datos pueden no reflejar las condiciones actuales del sistema, lo que limita su aplicabilidad en situaciones dinámicas.
- *Limitaciones en la Predicción:* Pueden no ser suficientes para entrenar modelos de aprendizaje automático destinados a operar en condiciones de tiempo real.
- *Falta de Interactividad:* No permiten una respuesta inmediata a cambios en el sistema o entorno.

Se va a optar en este trabajo por obtener datos en tiempo real para valorar los límites y valores que se obtienen de la Raspberry, pero se va a realizar una simulación de los datos para obtener unos datos menos dependientes del sistema real.

Para ello, se va a realizar el siguiente procedimiento:

1. **Generación de Trayectoria para la Caja:** El proceso inicia con la generación de una trayectoria básica para una caja, la cual se mueve en línea recta partiendo de un punto de origen definido. La trayectoria se caracteriza por un movimiento secuencial en los ejes X e Y, donde en cada instante solo una dimensión está sujeta a cambios en la posición, emulando así un movimiento rectilíneo.Para añadirle más realismo a la trayectoria, se le añadirá ruido blanco en la dirección en la que no se mueve la cinta.
2. **Incorporación de Parámetros Dinámicos:** A la trayectoria básica se le añaden parámetros dinámicos clave, tales como la velocidad de desplazamiento de la caja, el intervalo temporal entre las mediciones (*timespan*), y las dimensiones específicas de las cintas transportadoras, tanto en sus tramos cortos como largos. Esta ampliación de parámetros busca refinar la simulación para que refleje con mayor precisión el proceso de captura de datos, similar al obtenido mediante sistemas de videovigilancia.
3. **Simulación de Zonas de Detección de Sensores:** Posteriormente, se procede a establecer zonas de detección virtual para simular la presencia y funcionamiento de sensores. En este modelo, si la trayectoria de la caja intersecta alguna de estas zonas, se activa el sensor correspondiente. Esta fase es crucial para abordar y resolver desafíos asociados con la sincronización y latencia, permitiendo la captura simultánea y sincronizada de la trayectoria de la caja y la activación de los sensores.
4. **Integración de Actuadores en la Simulación:** Finalmente, se define y sincroniza la actuación de diversos elementos mecánicos, como retenedores y pistones, con la trayectoria simulada de la caja y el estado de los sensores. La inclusión de estas zonas de actuación asegura una simulación integrada y coordinada, reflejando la interacción entre la caja, los sensores y los actuadores en un entorno controlado.

Con esto obtenemos los datos necesarios para alimentar al gemelo digital para su entrenamiento posterior. El código que realiza esta tarea de obtener los datos, uno para el servidor python y otro para la generación de datos del PLC.

4.2 Diseño del gemelo digital

Una vez que obtenemos los datos con los que se va a alimentar nuestro gemelo digital, hay que realizar el entorno de trabajo del mismo en Unity, el cual va a ser el motor virtual para nuestro gemelo.

Unity permite a los usuarios diseñar y desarrollar de manera intuitiva. En el contexto de configurar una estación de trabajo, el proceso comienza por definir y configurar los objetos dentro del entorno de Unity. Estos objetos pueden ser creados directamente en el propio Unity o importados desde otros entornos de diseño, como programas CAD (Diseño Asistido por Computadora), ofreciendo así una gran flexibilidad en el diseño.

El primer paso es la selección o creación del objeto que se desea manipular o estudiar en la estación de trabajo. Este objeto puede ser un modelo simple o un diseño complejo, dependiendo de las necesidades del proyecto. Una vez seleccionado o creado, el objeto se introduce en el entorno de Unity, donde se puede manipular su posición, orientación, y escala para ajustarlo a las necesidades específicas de la simulación. En este caso, se va usar un modelo simplificado de la célula de trabajo desarrollado por Denim, departamento de la universidad de Sevilla, el cual tiene los elementos básicos con los que vamos a trabajar.

- *Cintas de transporte:* Se definen las cintas de transporte en la misma escala que en la célula de fabricación, no se añade la cinta transportadora central ya que vamos a trabajar únicamente con las bandejas y no con los objetos que tiene encima de la misma.
- *Tanque de aire:* Se diseña el tanque de aire que elevará los pistones para el cambio de cintas.
- *Bandejas:* Se crea el modelo a escala de las bandejas que vamos a usar.
- *Botones de emergencia:* Se crea el modelo a escala de los botones de parada de emergencia.
- *Sensores y retenedores:* Se crea el modelo de los sensores colocados por toda la célula.

En la figura se muestra la escena completa del gemelo digital en Unity. En él podemos observar el nivel de detalle del gemelo,es cual tiene las mismas propiedas que la célula de fabricación a la que pretende representar.

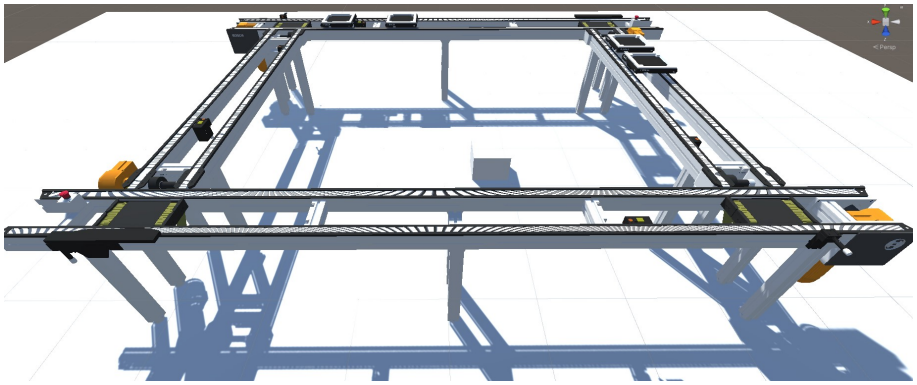

Figura 4.1 Plano general del gemelo digital diseñado.

4.2.1 Lógica interna

Respecto a la lógica primaria del gemelo, se dispone de los siguientes apartados en los que se detallan por módulos las características que debe de tener integrada el gemelo digital:

- *Automatización:* El módulo se compone de una automatización por estados para el funcionamiento simple del sistema, con lógica básica programable y con velocidad constante para las cintas.
- *Comunicación:* El módulo se compone con distintos scripts para comunicar el gemelo mediante distintos métodos, al cual se le añadirá los específicos para este trabajo.
- *Energía:* Módulo por el cual se calcula la energía suministrada y consumida por el sistema al completo.
- *Filtering:* Módulo para comunicar en C# y Python, aunque se usará uno específico apropiado para el trabajo.
- *Inicialización:* Módulo para iniciar los parámetros iniciales del sistema.
- *Modelos:* Módulo en el que se tienen en cuenta los actuadores, sensores y objetos para el sistema.

4.3 Comunicaciones

En este apartado, vamos a abordar el desafío de establecer las comunicaciones eficientes y seguras en el contexto de sistemas automatizados y redes industriales. Se va a usar la siguiente arquitectura para realizar la comunicación entre todos los dispositivos necesarios:

- *Comunicación PLC-Gemelo Digital:* Para realizar la comunicación entre PLC y Gemelo digital, vamos a usar el protocolo MODBUS TCP/IP, esto es debido a que el propio PLC Modicon M340 está configurado para poder escuchar y ser un esclavo en el Puerto 502 de nuestra IP, es por ello que, si usamos como maestro a nuestro gemelo digital, podemos recibir por este puerto tanto los sensores como los actuadores de una manera rápida y eficaz.

- *Comunicación Raspberry Pi 4-Gemelo Digital:* Para realizar la comunicación entre PLC y la raspberry, vamos a implementar el protocolo OPC UA para tener por un lado, un protocolo de comunicación distinto al anterior, y por otro, para implementar el método de la seguridad intrínseco a OPC UA. Por un lado, la comunicación podrá ser realiza directamente a través de los datos recibidos por la cámara de la rapsberry pi 4, o se podrá realizar mediante el guardado de los mismos y posteriormente usados como base de datos. La única diferencia será que una es en tiempo real y la otra no.

De esta manera, se tienen interconectados mediante el gemelo digital funcionando en Unity la comunicación tanto del PLC como la comunicación obtenida por la Raspberry, obteniendo en el mismo toda la información necesaria para realizar el control y el entrenamiento del modelo.

4.3.1 Comunicación OPC UA

Para la comunicación entre la Rapsberry Pi 4 y el gemelo digital, se realizarán los siguientes pasos clave:

1. El servidor, en este caso será la Raspberry Pi 4. Alternativamente, en ausencia de dicha unidad, se puede implementar un script en el ordenador local que opere de manera autónoma (offline). En este se recopilarán los identificadores únicos de cada bandeja (ID), la posición y la velocidad de cada bandeja en cada dirección en los momentos en los que se obtienen los datos.
2. En escenarios donde la cantidad de bandejas en el sistema experimente un incremento, se procederá a la generación dinámica de nuevos nodos en la red. Este enfoque asegura que cada bandeja, identificada de manera única, mantenga asociados sus respectivos parámetros de velocidad y posición en los ejes X e Y, facilitando así un seguimiento detallado y una representación precisa dentro del gemelo digital.
3. La iteración de este método de comunicación se adaptará de manera flexible a la cantidad de elementos presentes en el sistema. El proceso continuará en un ciclo iterativo, garantizando una actualización constante de los estados de los elementos, o hasta que el sistema concluya su operación, ya sea por un cierre programado o por la desactivación del sistema.
4. En lo que respecta al cliente, este rol será asumido por el gemelo digital desarrollado en Unity. Esta entidad se conectará al servidor mediante el puerto designado 48484, en casos donde el servidor resida en el mismo equipo local. Este esquema de conexión facilita el proceso de entrenamiento ya que se requerirá de mucho tiempo de entrenamiento para el sistema.

Este proceso es necesario ya que el aprendizaje del mismo dependerá de la posición donde se encuentre la bandeja detectada por la cámara, por lo que

4.3.2 Comunicación MODBUS TCP/IP

Para el método de comunicación MODBUS, se va a emplear Unity Pro XL (actualmente denominado EcoStruxure Control Expert) y el gemelo digital en Unity, se realizará de la siguiente manera:

1. Se obtendrá la dirección IP del PLC dentro de la red local de la universidad para la comunicación entre maestro y esclavo. Alternativamente, en el caso de no poder usar el servidor de la universidad, se simulará la conexión de ambos mediante la modificación de los valores de manera virtual.
2. Se guardarán en memoria todos los actuadores, sensores y valores que pueden ser útiles para la comunicación entre el gemelo digital y el PLC.
3. El esclavo, en este caso el servidor, enviará los datos que sean requeridos por el maestro, que en nuestro caso será el gemelo digital.
4. Se almacenarán en memoria los registros que se vayan a utilizar en el gemelo digital después de comprobar el correcto funcionamiento de ambos.

4.3.3 Sincronización de datos

La coherencia temporal entre los distintos componentes del sistema es de vital importancia para garantizar la precisión y la relevancia de los datos utilizados en la simulación y el control del sistema. La sincronización adecuada de los tiempos en estos dispositivos es esencial para asegurar que los datos recopilados y utilizados sean congruentes y reflejen fielmente el estado real del sistema en cualquier instante dado.

La sincronización temporal implica la alineación de los marcos temporales en los que opera cada componente del sistema. Esto incluye asegurar que Unity, la cámara y el PLC operen dentro de un intervalo de tiempo coordinado, lo que permite una integración efectiva de los datos recopilados. En particular, la inferencia de la red neuronal para la captación de la posición y velocidad de las bandejas es un tiempo a tener en cuenta para la sincronización precisa. El uso de una tasa de refresco correcto y consistente es fundamental para evitar la dispersión de la información y garantizar la congruencia de los datos recibidos.

Como ejemplo a tener en cuenta en situaciones de oclusión, donde la visibilidad de los elementos puede verse comprometida, es crucial tener un entendimiento claro de la posición y el estado de dichos elementos. Esto puede lograrse mediante la utilización de datos redundantes, como la velocidad de la bandeja antes de la oclusión o la confirmación de su posición al activar un sensor. Estos datos proporcionan un medio para inferir la posición de la bandeja incluso cuando no está directamente visible, asegurando así la continuidad y la fiabilidad de la información utilizada para la simulación y el control del sistema. Adicional a lo anterior, el modelo digital nos ayuda a obtener esta información redudante y procesarla en un entorno en el cual podemos validar los datos o podemos tener en cuenta si la información que recibimos no es congruente por el estado en el que debería de estar si los datos que recibimos por la cámara o por el PLC no son redudantes o carecen de lógica.

4.4 Entrenamiento del Gemelo Digital

El proceso de entrenamiento del gemelo digital es un componente crucial para asegurar su funcionamiento óptimo y su capacidad para simular con precisión el comportamiento del sistema físico real y del objetivo final del gemelo. Este apartado se dedica a explorar los diferentes aspectos involucrados en el entrenamiento del gemelo digital, desde los métodos de entrenamiento hasta las herramientas específicas utilizadas, como el plugin ML-Agents de Unity.

4.4.1 Métodos de Entrenamiento

El entrenamiento del gemelo digital puede adoptar diversas formas, dependiendo de los objetivos específicos del sistema y la naturaleza de las tareas que se espera que realice. Entre los métodos más comunes se encuentran:

- **Aprendizaje Supervisado:** Donde el modelo aprende a partir de un conjunto de datos etiquetados, intentando predecir la etiqueta correcta para nuevos datos basándose en ejemplos anteriores.
- **Aprendizaje No Supervisado:** Este enfoque implica que el modelo intente identificar patrones y estructuras dentro de los datos sin la guía de etiquetas predefinidas.
- **Aprendizaje por Refuerzo:** En este método, el gemelo digital aprende a realizar tareas mediante la experimentación y la obtención de recompensas, ajustando sus acciones en base a las recompensas recibidas para maximizar algún concepto de recompensa acumulativa.

4.4.2 Plugin de Unity ML-Agents

El plugin ML-Agents de Unity es una herramienta que permite la integración del aprendizaje automático dentro de entornos de simulación en Unity. Este plugin facilita la implementación de técnicas de aprendizaje por refuerzo, aprendizaje supervisado, y otros métodos de IA, permitiendo que los agentes dentro del gemelo digital aprendan y se adapten a partir de la interacción con el entorno virtual.

4.4.3 Elementos Necesarios para el Entrenamiento

Para un entrenamiento efectivo, es esencial contar con ciertos componentes que permitan la recopilación de datos, la interacción con el entorno y la evaluación del rendimiento de los agentes. Estos incluyen:

- **Sensores:** Que proporcionan al gemelo digital información sobre su entorno, permitiéndole percibir y reaccionar de manera informada.
- **Actuadores:** Los mecanismos mediante los cuales el gemelo digital puede influir en su entorno, ejecutando acciones basadas en sus decisiones de aprendizaje.
- **Observaciones:** Datos recopilados durante la simulación que sirven como entrada para el proceso de aprendizaje, incluyendo el estado del entorno, la posición de los objetos, entre otros.
- **Recompensas:** En el contexto del aprendizaje por refuerzo, las recompensas son fundamentales para guiar el aprendizaje del agente, indicando el éxito o fracaso de las acciones realizadas.

4.4.4 Consideraciones Adicionales

Al implementar el entrenamiento del gemelo digital, es importante tener en cuenta factores como la precisión de la simulación, la representatividad de los datos de entrenamiento, y la capacidad computacional disponible. Estos factores pueden influir significativamente en la eficacia del entrenamiento y en la fidelidad del gemelo digital respecto al sistema real como veremos en el apartado resultados.

Mostramos a continuación, la arquitectura del sistema en la Figura 4.2 , en la que se muestra el algoritmo a lograr y las comunicaciones entre los distintos sistemas usados en el proyecto. En el siguiente capítulo, entraremos más en detalle de la red neuronal y el gemelo digital creado y su funcionamiento.

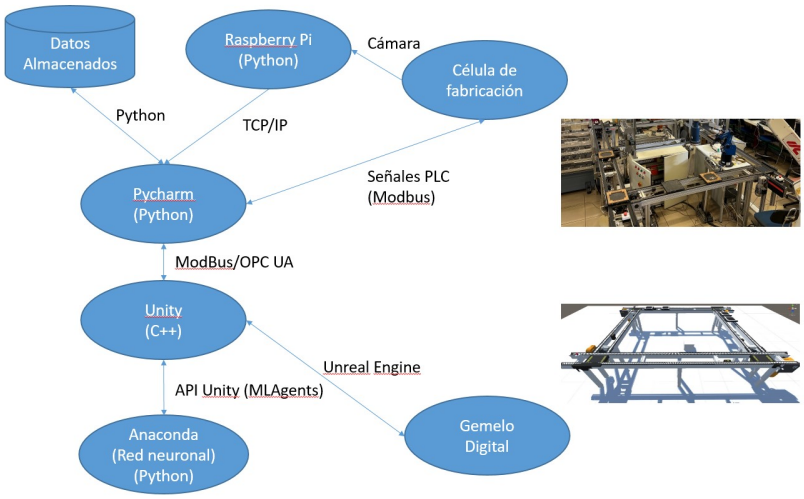

Figura 4.2 Diagrama de flujo de arquitectura del sistema.

5 Gemelo Digital: Diseño y funcionamiento en el proyecto

En este capítulo, exploraremos en profundidad los aspectos técnicos y metodológicos que subyacen a la implementación y operación del gemelo digital. Este enfoque nos permitirá comprender no solo las interacciones entre los componentes del sistema, sino también las estrategias de entrenamiento adoptadas para optimizar su funcionamiento y el entorno de trabajo en el que vamos a desarrollar el trabajo.

5.1 Comunicaciones

Para realizar las comunicaciones entre los dispositivos e Unity, se ha optado por dividir el funcionamiento entre programas. Para la comunicación con el servidor de datos almacenados, como veremos a continuación, se realizará en el software PyCharm. Para la comunicación con los sensores y actuadores reales, se usará la aplicación Unity Pro XL. Para el registro de datos del entrenamiento y el propio entrenamiento, se realizará en Anaconda 3.

5.1.1 Ruido y Sensibilidad

Una de las principales consideraciones al establecer la comunicación entre el sistema físico y el gemelo digital es la gestión del ruido y la sensibilidad de los datos. En entornos reales, los datos suelen estar contaminados con ruido, lo que puede conducir a interpretaciones erróneas y decisiones subóptimas por parte del gemelo digital. La implementación de filtros y algoritmos de suavizado dentro de Unity ayuda a mitigar estos efectos, asegurando que los datos reflejen con precisión el estado del sistema.

En la obtención de los datos de la cámara, la velocidad se encuentra con unos valores de velocidad no constante, por lo que se realiza un promedio de las mismas para que en el gemelo digital se pueda obtener un vector constante de velocidad para poder simular la bandeja con esta velocidad promedio. El error al obtener la velocidad de la bandeja a través de la cámara se debe a múltiples factores, siendo la variación de tamaño de la bandeja respecto a la cámara, es por ello que ya que la cinta de trabajo tiene actualmente una velocidad constante, promediamos y obtenemos que la velocidad de desplazamiento se encuentra en 0.2 m/s, tanto para las cintas largas como las cortas.

Podemos observar en las figuras 5.1 y 5.2 como los datos que observamos a través de la cámara no son completamente correctos, por lo que un tratamiento directo sin filtrar dichas posiciones y velocidad pueden hacer que el entrenamiento sea un fracaso únicamente por los datos de entrada. A pesar de lo anterior, tampoco es necesario tener unos datos completamente simulados, en los que no haya ningún tipo de error, debido a que su implementación en la realidad puede hacer que si no tiene unos datos ideales (por ejemplo, no teniendo en cuenta el rozamiento), puede hacer que el gemelo digital tome decisiones correctas en la simulación, pero dicha red neuronal en la célula de fabricación no actué correctamente.

También respecto a los sensores, la modificación de las distancias de los sensores en la célula de fabricación hace que sea más complicado trasladar inequívocamente la posición del las bandejas al pasar por los sensores. Esta modificación es posible de subsanar midiendo y comprobando que las medidas físicas y las medidas en el entorno virtual sean idénticas o al menos a escala. Tenemos que tener en cuenta que no se ha hecho a la escala por lo que es posible cierta diferencia entre ambas. Por ello, lo óptimo es realizar una transformación de las posición de las bandejas en las esquinas, con nuestra referencia real, y trasladar esas posiciones al

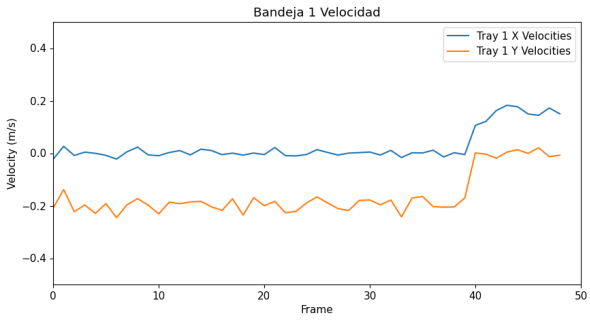

Figura 5.1 Estimación de la velocidad a través de la cámara.

entorno de Unity, así tendremos en cuenta que a pesar de que no tengan la misma escala, sean comparables las posiciones del entorno con el mundo real.

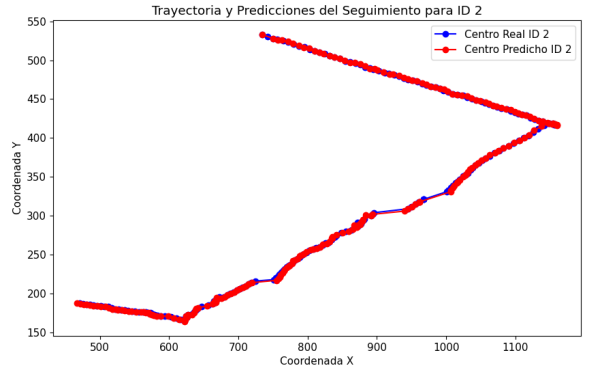

Figura 5.2 Estimación de la posición a través de la cámara.

Por último, uno de los problemas con Unity es la simulación de los golpes y las colisiones respecto al mundo real. En la célula de fabricación es prácticamente improbable un descarrilamiento de las bandejas a través de las cintas transportadoras, sin embargo en nuestro modelo en Unity, es posible que descarrille debido a que no se encuentra encajado dentro de las cintas transportadoras como tal. Esto es un problema ya que existe la posibilidad de que al simular un movimiento la bandeja no quede lo suficientemente correcta como para poder levantarse en el elevador y dejaría de funcionar correctamente el entorno virtual.

5.1.2 Comunicación con Base de Datos Generada

Para tener suficientes dato como para realizar un entrenamiento a una red neuronal, debido a que el sistema es lo suficientemente complejo como para requerir de una gran cantidad de información para poder realizar un adecuado entrenamiento. Es por ello que no solo basta con generar trayectorias en el mundo real y trasladarlas a al gemelo para que entrene, si no que debemos de, una vez teniendo en cuenta las trayectorias que puede tener nuestra célula de fabricación, entrenar todos los tipos de casos que puedan ocurrir.

En el caso actual, la célula únicamente tiene un sentido de dirección de movimiento a través de las cintas, por lo que teniendo en cuenta la trayectoria y la velocidad que permanece invariante, el caso se simplifica. Aún así, es posible añadir un variador de frecuencia para aumentar o disminuir la velocidad de las bandejas dentro de la célula, por lo que en ese caso, se deberá de tener en cuenta las posibles variaciones de velocidad en la trayectoria a recopilar y generar nuevas trayectorias o durante más tiempo para poder entrenar a la red neuronal con muchos más datos.

En la 5.3, podemos ver cual sería finalmente la trayectoria que se obtiene filtrada que recibiríamos de la cámara, teniendo en cuenta con una zona sombreada en amarillo cuál sería la zona en la que los senores se activaría, para así tener en cuenta realimentando con los datos que se puedan obtener del PLC, si es correcto o no la posición estimada por la cámara.

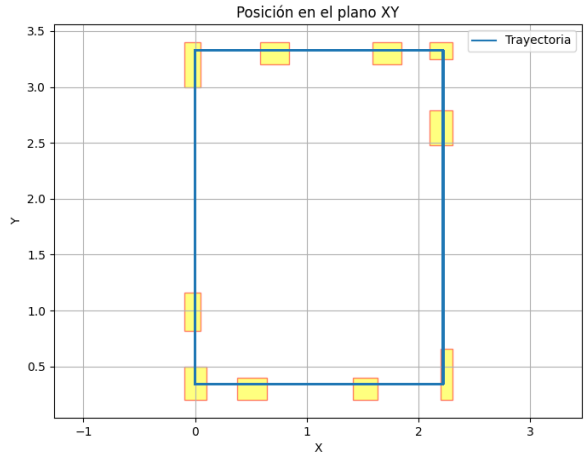

Figura 5.3 Movimiento filtrado de la bandeja y la zona de sensores.

Con esto, somos capaces de generar trayectorias y salidas de sensores como las tomadas en las figuras 5.4 y **??**, en este caso, para la realización del entrenamiento, requerimos de una gran cantidad de tiempo dando vueltas sobre la célula de fabricación, por lo que generamos un archivo .json creado por un script en el que le indicamos el número de vueltas que queremos realizar, de cara a tener simulado varias horas de funcionamiento seguido sin parar de una manera cómoda y sencilla.

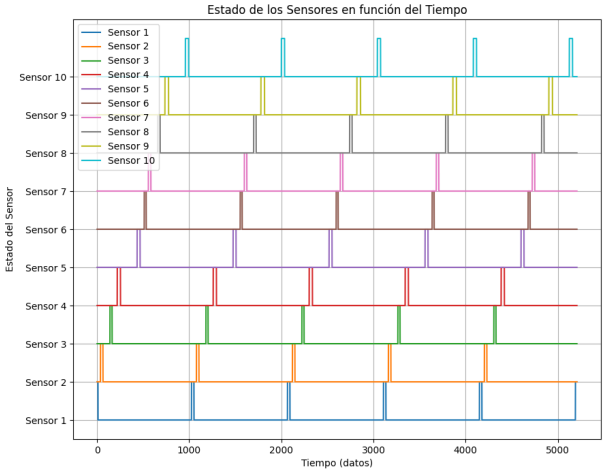

Figura 5.4 Estado de los sensores en función de la posición en 5 vueltas.

Esto es necesario ya que el entrenamiento con datos obtenidos únicamente a través de la cámara sería inviable, debido a que habría que filtrar muchos más datos y el tiempo requerido en la universidad y del uso de las instalaciones sería mucho mayor, además de los fallos que pueda tener la célula de fabricación.

5.2 Entrenamiento

El entrenamiento del gemelo digital en Unity es un proceso iterativo que involucra la calibración de observaciones, actuadores y sistemas de recompensas para afinar el comportamiento y las respuestas del modelo. Al no ser una automatización con lógica básica, la automatización de los procesos mediante una red neuronal va a requerir de diversas pruebas y métodos para que se consiga el objetivo principal de la red neuronal, que es el seguimiento de la célula de fabricación en condiciones idénticas.

5.2.1 Objetivo del entrenamiento

El objetivo del entrenamiento que se va a realizar en este trabajo es el seguimiento y actuación autonómo por parte de una red neuronal entrenada mediante aprendizaje con refuerzo, mediante la entrada de la posición y velocidad de la bandeja a través de los datos capturados por la cámara, y las posiciones de los pistones que elevan o bajan a la bandeja de las cintas transportadoras. El objetivo es que la diferencia entre ambas bandejas sean mínima, para que se realice un seguimiento de la misma y se conozca en todo momento en donde se encuentra en la célula de fabricación.

Como objetivos secundarios de la red neuronal, sería el correcto funcionamiento de los pistones y de la velocidad de las cintas transportadoras, por lo que esta red debe ser capaz de coordinar ambas actividades por sí sola,por un lado, no tener problemas a la hora de cambiar de cintas transportadores (elevar o bajar pistones) en función de los sensores y de los estados que se tienen.

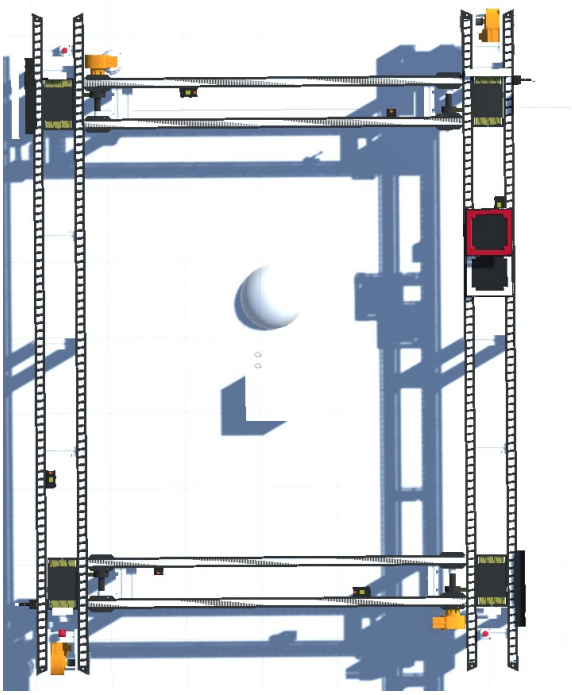

Figura 5.5 El objetivo es que ambas bandejas se encuentren una encima de otra.

5.2.2 Elementos del gemelo

En esta sección vamos a describir todos los elementos de los que se compone el gemelo digital, y como podemos interactuar con los mismos y que información nos pueden proporcionar y cómo los hemos configurado para el trabajo. Los elementos a destacar son los siguientes:

- **Cinta Transportadora Larga** El módulo completo de la cinta se descompone en 6 partes en concreto:

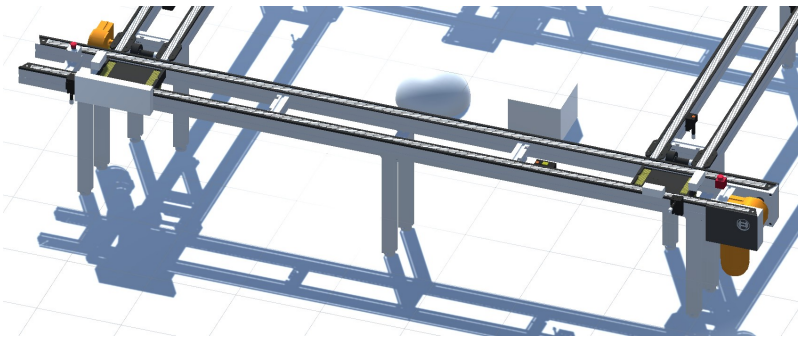

Figura 5.6 Imagen de la cinta transportadora larga.

– Cinta: Se compone de agrupaciones de rectángulos clonados a lo largo de la cinta transportadora, que al entrar en contacto con un objeto, lo desplazan en la dirección del movimiento de la cinta, creando el movimiento de las bandejas.

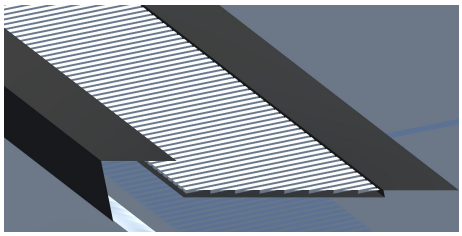

Figura 5.7 Imagen ampliada de la cinta transportadora.

– Módulo de parada: El módulo correspondiente al sensor-actuador que nos encontramos en la célula de fabricación, se trata de una pestaña integrada con un sensor, que se eleva para parar las bandejas que puedan entrar en el pistón, se trata de un método para no sobrealimentar los pistones y llevando a la colisión de los mismos en la parte móvil. Se eleva lo suficiente para poder para la bandeja por debajo de la misma, ya que tiene una ranura por debajo de la misma. En la figura 5.8 se pueden observar de color amarillo el módulo de parada y el sensor inductivo.

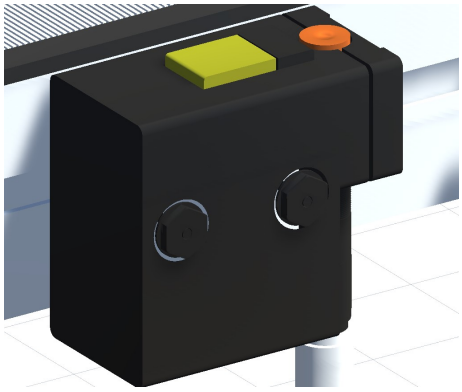

Figura 5.8 Imagen del módulo de sensor-actuador en la cinta.

– Botón de emergencia: Botón de emergencia para parar todos los actuadores del gemelo en caso de pulsarse. Adicionalmente al que se muestra en la figura 5.9, se incluye uno de color rojo, indicando cual es la referencia inicial que usamos para ubicar las bandejas en la célula de fabricación, por lo que se usará dicho botón para la referencia real en metros.

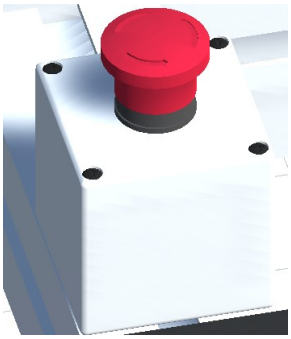

Figura 5.9 Imagen del botón de emergencia.

– Sensor de fuerza: Este sensor actúa como final de carril del la cinta corta pero se encuentra integrada en la cinta transportadora larga. Se encuentra en el borde exterior para que en el caso de

que se dirija una bandeja desde la cinta corta, al tener contacto con la misma, empuje el cabezal y se separe cierta altura respecto a un sensor inductivo, el cual detecta de manera continuada una pieza de metal incrustada en el cabezal del sensor. Es por esto que este sensor funciona de manera invertida a los anteriores, la señal se encuentra activa hasta que es empujada por una bandeja, y la señal se vuelve inactiva. Finalmente al abandonar la bandeja la posición de colisión, tiene un resorte para volver a su posición inicial, para que no se encuentre inactiva y pueda volver a funcionar de nuevo. Se puede ver en detalle en la figura 5.10.

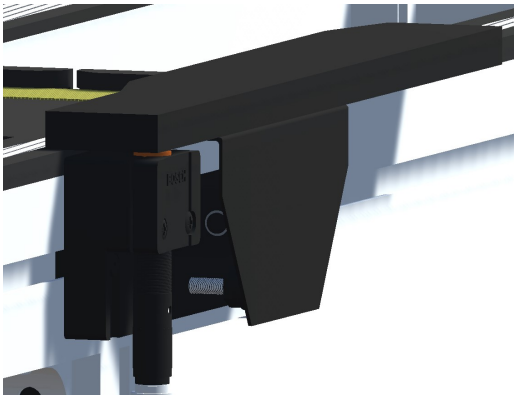

Figura 5.10 Imagen del sensor de fuerza.

- Sensor inductivo: Este sensor, integrado en el final de carril opuesto al que se encuentra en en el sensor de fuerza, nos permite detectar el borde de la bandeja ya que tiene una chapa metálica para poder ser detectada por este sensor. Como ventaja respecto al sensor de fuerza, este sensor nos permite detectar la posición sin tener ningún impacto con la misma, además, es posible usarlo en ambos sentidos, a diferencia del sensor de fuerza, que no podemos conocer con exactitud el punto de la esquina de la bandeja, ya que al querer elevar o bajar el pistón, es crucial que esté alineado tanto la bandeja como el pistón.

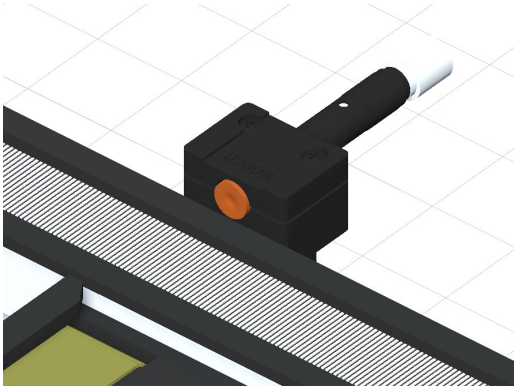

Figura 5.11 Imagen del sensor inductivo de final de carril.

- Pistones: Se encuentran dos pistones por cada cinta transportadora larga, ambos se encuentran con una pequeña cinta propia para mover la bandeja de la cinta transportadora corta a la larga. Sólo puede elevar una bandeja y tiene en el borde unos soportes para encajar la bandeja. Tiene la altura suficiente para que en caso de encontrarse en abajo la bandeja pueda pasar por encima de ella, pero si se eleva, no se puede subir ninguna bandeja.

- **Cinta Transportadora Corta**: Al igual que la cinta transportadora larga, consta de varios elementos que son reutilizados en el gemelo digital. Podemos ver la imagen en la figura 5.13.
  - Cinta transportadora corta: En este caso la diferencia es la longitud respecto a la cinta anteriormente definida.

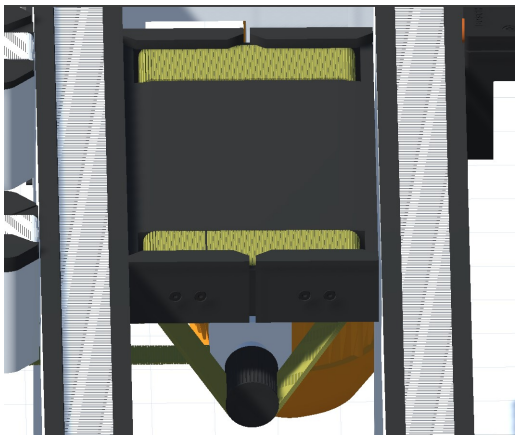

Figura 5.12 Imagen de uno de los dos pistones de la cinta.

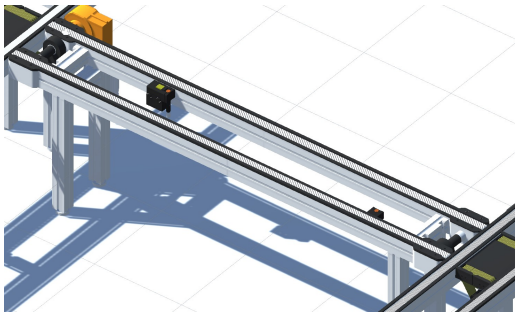

Figura 5.13 Imagen general de la cinta transportadora corta.

- Módulo de parada de emergencia. Se encuentra el mismo módulo sensor-actuador, uno en cada cinta, con la misma funcionalidad que en la cinta transportadora larga.
- Sensor inductivo: Se encuentra en este caso en el interior de la cinta transportadora, en el lado opuesto al sensor-actuador, para indicar nuevamente la salida de una bandeja para dar continuidad a la elevación de una nueva sobre el pistón.

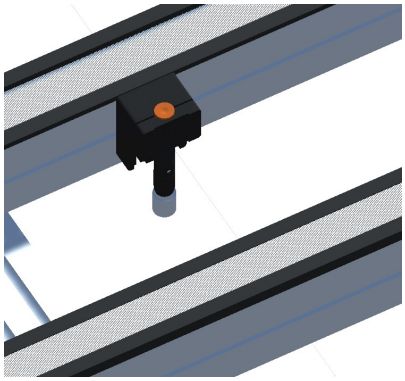

Figura 5.14 Imagen del sensor inductivo dentro de la cinta.

- **Tanque de Aire**: Contiene las componentes para dar presión a los actuadores que tenemos en el gemelo. No se encuentra físicamente representado pero es la parte que pretende representar el aire a presión que necesitamos para elevar los pistones y los paradores que alimentan a la cinta, por lo que en caso de que no hubiera presión, no se podría actuar sobre estos.
- **Bandejas**: Son los elementos móviles de la célula de fabricación que alojarán las piezas que se muevan por las cintas transportadoras. En este caso, se han adaptado para tener chapas metálicas a lo largo

de los puntos clave para que los sensores inductivos detecten la caja, tanto por la parte inferior de la caja, como se muestra en la figura 5.15, como por la superior y lateral, donde podemos ver también las chapas metálicas de menor tamaño.

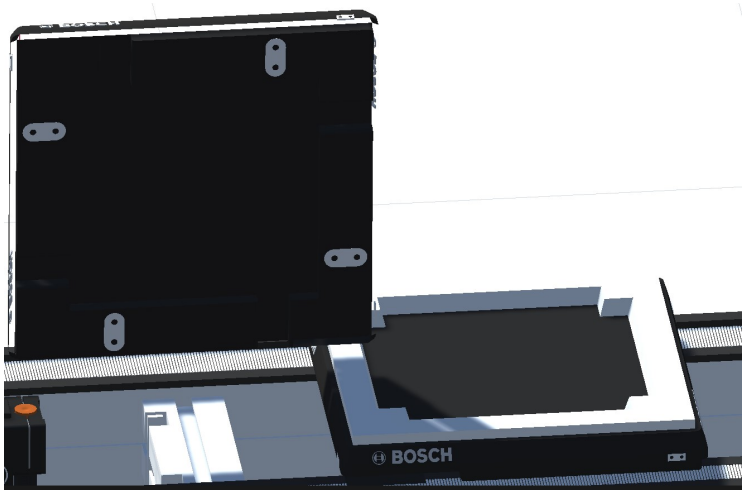

Figura 5.15 Imagen las bandejas parte superior e inferior.

Observaciones

Una vez que tenemos descritas todas las partes de las que se compone el gemelo digital, se procede a enumerar las observaciones con las que vamos a trabajar para que la red neuronal sea capaz de realizar el seguimiento de la bandeja en tiempo real. Este es el número máximo de variables posibles, pero se verá en apartados posteriores que es posible simplificar el proceso.

Como observaciones posibles serían las siguientes:

- Vector posición bandeja referencia: Vector de dimensión 3.
- Vector posición bandeja controlada: Vector de dimensión 3.
- Estado de los 4 pistones: Variables booleanas.
- Velocidad cinta: Variable tipo float.
- Estado de los 10 sensores: Variables tipo booleana.

Actuadores

Los actuadores en este caso quedarán definidos de la siguiente manera:

- Motores de las 4 cintas descritas en la célula de fabricación.
- Los 4 actuadores que elevarán y bajarán los pistones en la célula de fabricación.

Método de Recompensas

El método de recompensas es un componente crítico del entrenamiento, especialmente en enfoques de aprendizaje por refuerzo. Dentro de Unity, establecemos un sistema de recompensas que valora las acciones del gemelo digital en función de su efectividad y precisión. Este sistema motiva al gemelo digital a optimizar sus decisiones para maximizar la acumulación de recompensas, guiándolo hacia comportamientos que mejoran el rendimiento y la eficiencia del sistema simulado.

Este capítulo ha proporcionado una visión general del proceso de configuración y entrenamiento del gemelo digital en Unity, destacando la importancia de una comunicación efectiva, el manejo del ruido, y la implementación de un sistema de entrenamiento robusto para optimizar el funcionamiento del gemelo digital.

6 Resultados

En este capítulo se presentan los resultados alcanzados en la implementación y optimización del gemelo digital. A lo largo del desarrollo, se destacó la importancia de manejar adecuadamente el ruido, problemas en la física con Unity y las comunicaciones efectivas para asegurar una simulación correcta y obtener resultados favorables.

La implementación práctica del gemelo digital permitió identificar y rectificar errores en las simulaciones, mejorar y realizar simplificaciones en el programa, y facilitar la implementación de pruebas con redes neuronales para realizar entrenamiento y funcionamientos más complejos que una automatización mediante lógica programada básica. Estos avances proporcionarán en futuras líneas de investigación problemas que tiene asociado el entrenamiento mediante recompensas y tenerlos en cuenta para poder avanzar en otras líneas, y así tener un modelo de gemelo digital más completo y complejo.

6.1 Simplificación del sistema

En primer lugar, inicialmente se pensó realizar el trabajo con el modelo completo de la célula de fabricación, pero para realizar el seguimiento de una bandeja en tiempo real con la misma ubicación en la célula y en Unity, se ha tomado la elección de realizar sin el alimentador de bandejas ni el almacenamiento de piezas pequeñas que suponen un mayor procesamiento de datos, y añadimos variables complejas que no aportan en principio ayuda al entrenamiento de la red neuronal. Es por ello, que se decide ubicar, simplificando el añadir el alimentador de bandejas, una posición fija el inicio de los entrenamientos para la bandeja, simulando así el inicio del entrenamiento que se correspondería con la primera bandeja que introduciría en el sistema el alimentador de bandejas.

Debido a la complejidad del sistema a entrenar, y debido a que existen muchos factores con los que se ha realizado muchas pruebas y se ha ido, mediante prueba y error, mejorando e intentando resolver los problemas que nos hemos encontrado, se va a realizar una lista de pruebas e intentos de mejora, que con las pruebas y tras comprobar que los resultados no han sido los esperados o ha sido infructuoso el resultado obtenido, se detallan estos mismos ya que se quiere constar de la sucesiva mejora continua que se ha querido tomar en este trabajo, sea un resultados satisfactorio o sea un resultado no óptimo.

6.1.1 Definición de estados del sistema

Para implementar la lógica del sistema.

- L1: Comprende desde el sensor de fuerza de la cinta de transporte larga (sensor3) hasta el sensor de metal de la misma cinta (sensor4). Entra en este estado al desactivarse el sensor3 y sale al activarse el sensor4.
- L12S0:Comprende desde el sensor de metal anterior (sensor4) hasta el sensor de metal de la siguiente cinta (sensor6). Entra en este estado al activarse el sensor4 y sale al activarse el sensor6. No usamos el sensor5 en este caso ya que se usa para la activación del pistón nº3. Consideramos este estado de transición ya la bandeja pasa de una cinta a otra, siendo posible invertir el sentido debido a la configuración de los sensores, caso no posible el otros pistones debido al sensor de fuerza.

- S0:Comprende desde el sensor de meta de la cinta de transporte corta (sensor6) hasta el sensor de metal de la misma cinta (sensor7). Entra en este estado al activarse el sensor7 y sale al desactivarse el sensor7.
- S02L0:Comprende desde el sensor de fuerza de metal de transporte corta (sensor7) hasta el sensor de fuerza siguiente (sensor8). Entra en este estado al activarse el sensor7 y sale al desactivarse el sensor8. Este estado no es invertible al tener un sensor de fuerza en el pistón nº4.
- L0:Comprende desde el sensor de fuerza de la cinta de transporte larga (sensor8) hasta el sensor de metal de la misma cinta (sensor9). Entra en este estado al desactivarse el sensor8 y sale al activarse el sensor9.
- L02S1:Comprende desde el sensor de metal de la cinta de transporte larga (sensor9) hasta el sensor de metal de la siguiente cinta (sensor1). Entra en este estado al activarse el sensor9 y sale al activarse el sensor1.
- S1:Comprende desde el sensor de metal de la cinta de transporte corta (sensor1) hasta el sensor de metal de la misma cinta (sensor2). Entra en este estado al activarse el sensor1 y sale al activarse el sensor2.
- S12L1:Comprende desde el sensor de metal de la cinta de transporte corta (sensor2) hasta el sensor de fuerza del pistón2 (sensor2). Entra en este estado al activarse el sensor2 y sale al desactivarse el sensor3.

Se muestra a continuación, dividido por zonas, la visualización gráfica de los estados de las cintas de transporte y sus estados de transición:

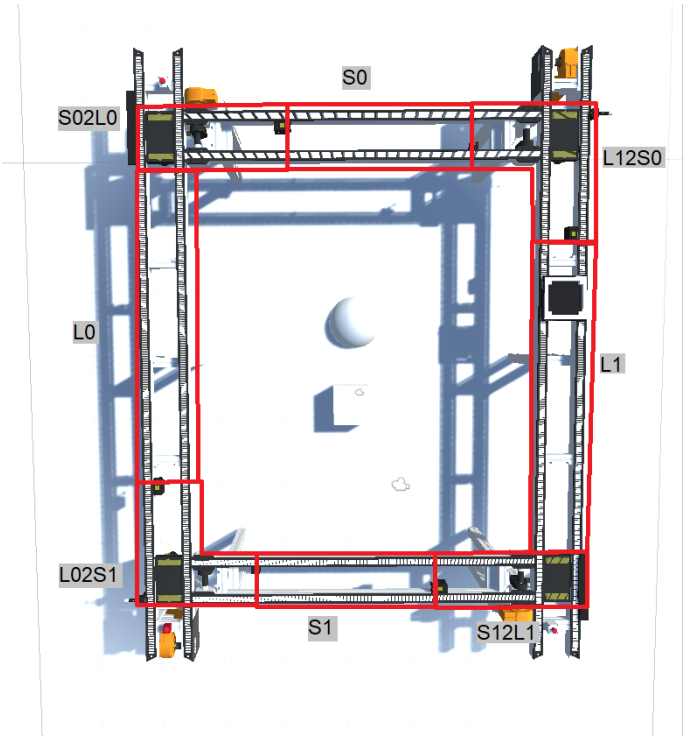

Figura 6.1 Localización de los estados del sistema.

6.2 Problemas encontrados

En esta sección, detallaremos los problemas y errores encontrados a la hora de realizar los entrenamientos y el funcionamiento del modelo en Unity. Procedemos en este caso, a realizar una lista completa de los errores y de su decisión tomada para resolver el problema:

6.2.1 Final de cintas de transporte

Al comenzar el movimiento de las bandejas a través de las bandejas, nos encontramos en las cintas de transporte, en particular las cintas de transporte que son de mayor longitud, que no se define un final de carril o un sistema de anclaje del pistón elevador para que la bandeja se encuentre alineada con el pistón. El problema de no realizar este alineamiento son varios, entre otros:

- **Imprecisión en el entrenamiento de la red neuronal:** Debido al realizar el movimiento con una velocidad considerable, la elevación de la bandeja es un momento crucial en el gemelo, es por ello que una detección no sincronizada o no detectada por el sensor, junto con una toma de decisiones imprecisas por la red neuronal mientras se entrena hace que la elevación del pistón se realice sin estar alineada, por ello, se eleva una parte de la bandeja mientras que otra no, haciendo que la simulación de la misma sea una colisión en la que prácticamente en 90 % de las veces, resulte en un reseteo de la simulación al salirse la bandeja de los carriles de movimiento que se tienen establecidos.
- **Descarrilamiento continuo de la bandeja** Como hemos comentado anteriormente, la elevación de la bandeja es un momento complejo que necesita que se encuentre perfectamente alineado con el pistón en cuestión, por ello, como se ha descrito anteriormente, en el entrenamiento se realizan elevaciones parciales de la bandeja que, en el mejor de los casos es pequeño, pero en la mayoría de las veces resulta de un descarrilamiento del bandeja. Esto se produce inicialmente en la cinta de transporte larga, pero, en el mejor de los casos, la bandeja, en algunos casos es posible que avance a la cinta de transporte corta torcida, realizando un avance no perpendicular que hace que el sistema detecte como movimiento correcto, ya que el entrenamiento del mismo hace que si avanza la bandeja se recompensa positivamente, pero sin embargo al llegar al final de la cinta corta, no se coloque ya que queda torcida. Es por ello que finalmente se reseteará el entrenamiento y será infructuoso, sin tener una causa evidente ya que la red neuronal previamente, en sus pasos previos, se le ha recompensado positivamente por el avance de la bandeja.
- **Complejidad a la hora de retroceder la bandeja** Para resolver el problema de descarrilamiento, se optó en un inicio en añadir la posibilidad de retroceder el movimiento de la bandeja hacia atrás, en el caso de que la bandeja haya avanzado demasiado y el pistón no se haya elevado, pero se descarta esta opción rápidamente ya que como hemos comentado, se reitera en la posibilidad de imprecisión al entrenar y elevar la bandeja, y vuelve a existir la posibilidad de descarrillar la bandeja,por ello, se decide no implementar esta solución.

Como solución a estos problemas evidentes, añadimos un objeto inmóvil que realice de sujeción de final de carril en este caso, tal y como ocurre en la célula de fabricación ya que la bandeja no avanza indefinidamente a pesar de que la bandeja se encuentre con el pistón en la posición baja. No se realiza exactamente el mismo objeto que en la célula de fabricación, si no que usamos un objeto simple para que impacte contra la bandeja, con las propiedades que detallamos a continuación:

- **Bounciness(Rebote)=0**: Se crea un material que tenga como propiedad física rebote igual a 0, por el hecho de que el impacto de la misma haría que la bandeja se desplazase en el sentido contrario al movimiento de la bandeja, haciendo que volvamos a tener el problema de imprecisión en el elevamiento de la bandeja y el pistón de manera conjunta.
- **Dynamic friction(fricción dinámica)=0,01:** Se necesita de este final de carril que durante el traslado de la bandeja desde el pistón elevador a la cinta de transporte de menor longitud, no exista apenas rozamiento, por que esto hace que el movimiento de la misma sea más forzado, e incluso llegar a descarrillar la bandeja al crearse tensión entre ambos objetos. Por ello la fricción del objeto debe de ser mínima.
- El resto de parámetros se pueden dejar estándar o menores a lo estándar generado por Unity, ya que los dos anteriores son los más relevantes para nuestro funcionamiento en la elevación del pistón y la bandeja.

6.2.2 Sistema de recompensas

Para el sistema de recompensas, se han realizado multitud de intentos de recompensas y tomas de decisiones, por lo que de definimos a continuación los errores o resultados negativos que hemos tenido con los sistemas de recompensas definidos para el modelo de red neuronal.

6.2.3 Acelerar el movimiento hacia el objetivo

Se tomó en un inicio un intento de mejorar el movimiento hacia el objetivo, realizando pequeña tareas intermedias como sistema de recompensa una recompensa significativa inversamente proporcional al tiempo tardado en completar esta tarea. Esta tarea en cuestión, se trataría de llevar la bandeja desde el sensor antes del pistón al sensor posterior al pistón, en este caso se trataría de permanecer el menor tiempo posible en los estados transición. La fórmula en cuestión es la siguiente:

$$\text{Recompensa\_por\_tiempo} = \frac{\text{Recompensa\_fija}}{\text{tiempo\_transcurrido}}$$

Esta fórmula de objetivo, tiene sus ventajas y sus inconvenientes.

Por un lado, tenemos que tener un balance correcto entre recompensas fijas y recompensas continuas, por lo que encontrar un valor correcto de recompensa fija depende de cuánto obtiene de recompensa el sistema en función de la distancia objetivo a recorrer, ya que queremos minimizar la distancia entre la bandeja referencia y la que podemos mover, y esto hace que, en algunos casos, es necesario ir a una menor velocidad para maximizar la recompensar por distancia, que maximizar la recompensa por tiempo, a costa de que ambas bandejas no se encuentren alineadas. Es por ello que, en los casos que la distancia entre ambas bandejas, como un reinicio del sistema, pueda llegar a ser interesante añadir una recompensa por mejorar la velocidad del sistema, en los casos en las que ambas bandeja están alineadas descompensa el entrenamiento y se obtiene que prima la velocidad a la alineación.

Por todo lo anterior, se opta por no continuar recompensando a la red neuronal con este tipo de funcionalidades, y se comentarán en el código principal o será eliminado del mismo para mayor claridad al optimizarse el mismo.

6.2.4 Recompensas continuas

En este apartado se va a analizar los problemas referentes al cruzamiento de sistemas de recompensas, el cual a la hora de realizar tenemos que tener en cuenta los siguientes puntos:

- **frame rate (tasa de imágenes)**: Este es el número de veces por segundo en el que se realizan las ejecuciones de "update()" dentro de Unity, por lo que a mayor número de ejecuciones por segundo, más procesamiento necesitamos del mismo. En nuestro caso, el frame rate partirá de 60.
- **Decision period (período de decisión):** Es el número de veces que espera la red neuronal para tomar una decisión. Este número es importante ya que cada X tiempo tomará una decisión sobre una o varias variables, por lo que a un número muy pequeño, se tomarán muchas decisiones las cuales no tendrán un impacto positivo o negativo sobre el sistema, y no aprenderá del mismo; si es mayor, tardará en tomar una decisión que es necesaria tomarla, por lo que hay que encontrar un valor equilibrado. El número de veces que tomará una decisión sobre una variable continua en este caso será:

  $$\text{decisiones\_por\_segundo} = \frac{\text{frame\_rate}}{\text{decision\_period}}$$

  En este caso, optaremos por un valor de 10 para el período de decisión, teniendo unas 6 decisiones por segundo. Esta velocidad de decisión es elevada dependiendo de cuál sea, por ejemplo, en el caso de los pistones, esta velocidad se deberá reducir ya que muchas decisiones sobre elevar o bajar sin control hace que la bandeja rebote encima sin control, perdiendo la alineación y por lo tanto, produciéndose el descarrilamiento y el reinicio del entrenamiento.

Es por ello que llegar a un compromiso en la recompensa continua, con respecto a las recompensas discretas, ya que el método de entrenamiento lo que comprende es la puntuación recibida y en función de lo obtenido, realizará ciertos patrones, por ello, si únicamente tiene en cuenta un tipo de recompensa, no aprenderá a distinguir entre recompensas por distancia y recompensas por acciones discretas correctas.

6.2.5 Complejidad de la red neuronal

Realizar la red neuronal más compleja no ha tenido mejores resultados que los obtenidos con redes neuronales más simples. Añadir más capas ocultas, más neuronas por capa, o realizar más tiempo de entrenamiento no supone incrementar la mejora del sistema. Es por ello que se añade un elemento complejo ya que al

intentar simplificar o añadir nuevas variables de observación del sistema o eliminarlas,se tiene que probar de nuevo varias configuraciones de redes neuronales. Por ello, encontrar el punto óptimo entre complejidad, optimización y rendimiento se vuelve muy complicado y tedioso.

Un ejemplo claro sería los hiperparámetros escogidos para el agente `ConveyorBeltAgent` utilizando el algoritmo `PPO`, que fueron configurados de la siguiente manera:

Hiperparámetros Iniciales:

```
Tamaño de lote (batch_size): 16
Tamaño del búfer (buffer_size): 1024
Tasa de aprendizaje (learning_rate): 3.0e-4
Beta (beta): 5.0e-4
Épsilon (epsilon): 0.2
Lambda (lambd): 0.95
Número de épocas (num_epoch): 5
Programación de la tasa de aprendizaje (learning_rate_schedule): lineal
Configuración de la Red:
Unidades ocultas: 64
Número de capas: 4
Memoria:
Uso recurrente: Sí
Longitud de secuencia: 8
Tamaño de la memoria: 64
```

Sin embargo, estos valores iniciales no proporcionaron resultados satisfactorios, por lo que se realizó una modificación en la configuración, aumentando significativamente el tamaño del lote, el tamaño del búfer, y ajustando los parámetros de la memoria y las épocas de entrenamiento:

Hiperparámetros Modificados:

```
Tamaño de lote (batch_size): 256
Tamaño del búfer (buffer_size): 16384
Número de épocas (num_epoch): 10
Memoria:
Longitud de secuencia: 256
Tamaño de la memoria: 2048
Unidades ocultas (hidden_units): 256
Número de capas (num_layers): 8
```

Estas modificaciones no solucionaron los problemas subyacentes, lo que llevó a reconsiderar y modificar el sistema de recompensas. Este ajuste implica un desafío significativo debido a la interacción entre la complejidad de la hiperparametrización y el sistema de recompensas. Al enfrentar este problema, se genera una amplia malla de resultados que requiere una evaluación continua para identificar la configuración óptima que puede efectivamente resolver el problema en cuestión.

6.2.6 Ajuste recompensa continua o por objetivos

Como se ha descrito en la sección "Recompensas continuas", se requiere de encontrar un equilibrio entre las recompensas continuas o por objetivos. Por un lado, tenemos que priorizar minimizar las distancia entre las bandejas, por otro lado, debemos de comprometernos a realizar un movimiento continuo, es por ello, que para calcular la ponderación de las recompensas, realizamos una estimación de la siguiente manera:

$$\text{máx}(\text{recompensa\_continua}\,[ud/s]) > \text{máx}(\text{recompensa\_discreta}\,[ud/s]) \tag{6.1}$$

$$\text{recompensa\_discreta\_media}\,[ud/s] \approx \text{recompensa\_continua\_media}\,[ud/s] \tag{6.2}$$

$$\text{recompensa\_discreta}\,[ud/s] < \text{recompensa\_continua}\,[ud/s] \tag{6.3}$$

Debido a estas restricciones, se proponen distintas funciones de recompensas en función de la distancia euclídea entre ambas bandejas. Conociendo que el la distancia mínima sería 0, en caso de perfecto alineamiento, y 10, sería la distancia máxima entre las esquinas opuestas.

Enumeramos las funciones probadas durante el entrenamiento:

Función parabólica

$$y = -0.1x^2 + 10$$

Ventajas:

- Promueve que el agente se mantenga cerca del vértice de la parábola, que es el punto máximo, fomentando así un comportamiento centrado en el objetivo óptimo.

Desventajas:

- El decremento cuadrático puede ser demasiado punitivo para la distancia máxima, disuadiendo al agente de tomar acciones que inicialmente parecen perjudiciales pero que a largo plazo pueden ser beneficiosas.

Función lineal

$$y = -x + 10$$

Ventajas:

- Proporciona un gradiente constante, ofreciendo una retroalimentación consistente al agente sobre cómo las acciones afectan la distancia al objetivo.
- Simple y computacionalmente eficiente de evaluar.

Desventajas:

- El incentivo es lineal, por lo que no ajusta la intensidad de la recompensa basada en la proximidad al objetivo; es decir, la penalización por estar lejos del objetivo no aumenta.
- Puede fomentar una exploración insuficiente al no ofrecer recompensas incrementales a medida que el agente se acerca al objetivo.
- Puede no ajustarse correctamente con la compensación de recompensas discretas.

Función hiperbólica

$$y = \frac{1}{x + 0.1}$$

Ventajas:

- A medida que el agente se aleja del origen, la penalización aumenta, lo que refuerza fuertemente el retorno al punto de inicio o cercano a él.

Desventajas:

- La recompensa puede volverse insignificante a medida que la distancia aumenta, lo cual podría desmotivar al agente si el objetivo está lejos.
- Potencial para causar divisiones por cero o valores infinitamente grandes si x se aproxima a cero. Se resuelve el problema con la función propuesta ya que 1/x tendría no resuelve este problema.

Función exponencial

$$y = 10e^{-0.5x}$$

Ventajas:

- Provee una penalización que es exponencialmente más severa a medida que el agente se aleja del origen, incentivando un regreso rápido a la alineación.

Desventajas:

- Similar a la función hiperbólica, la penalización puede volverse demasiado severa rápidamente, potencialmente desalentando la exploración adecuada.

Función polinómica

$$y = -0.002x^4 + 0.032x^3 - 0.1x^2 - 1.2x + 10$$

Ventajas:

- La función es compleja y puede modelar múltiples comportamientos o fases dentro de un mismo entorno, ofreciendo recompensas diferenciadas basadas en la posición del agente.
- Posibilidad de ajustar valores intermedios más estables usando un exponente mayor y ajustando polinómicamente en función de nuestras necesidades.

Desventajas:

- La complejidad de la función puede hacer que sea difícil para el agente aprender la política óptima debido a los múltiples máximos y mínimos locales.
- Requiere un cálculo más intensivo y una mayor comprensión del entorno para optimizar efectivamente.

Observamos en la figura 6.2 las diferentes curvas, observando la recompensa máxima y la mínima en función de la distancia euclídea.

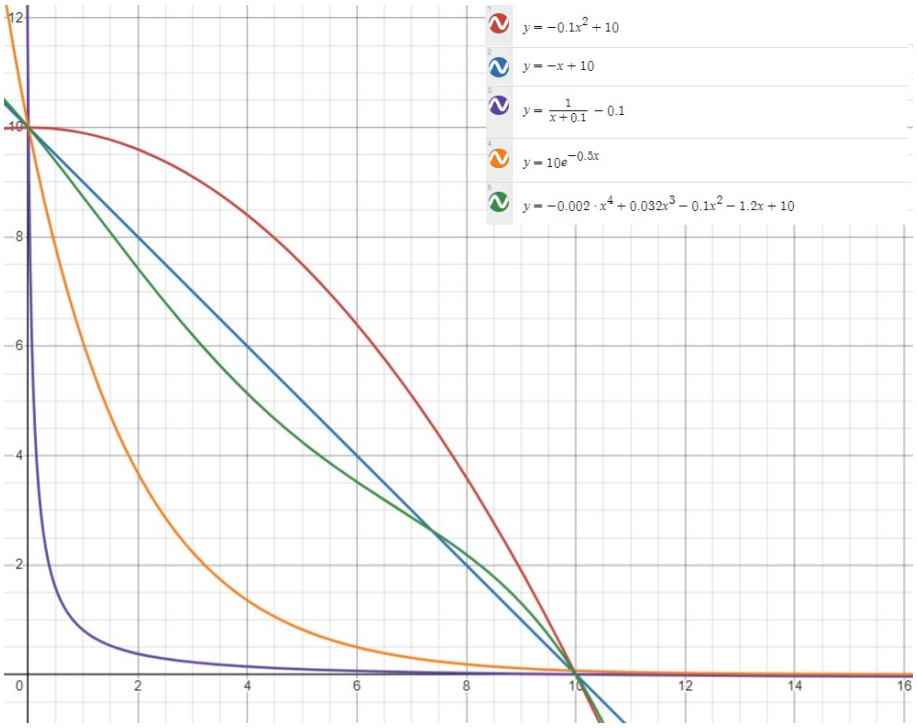

Figura 6.2 Imagen de las funciones.

En el caso de las recompensas por objetivos, será en los momentos de decisión de elevar o bajar el pistón, cuando se recompensará o penalizará al sistema. Debido a la recompensa continua tomada en los casos anteriores, es necesario ajustar dichos valores ya que el valor medio de las mismas depende de la curva en cuestión. Por ejemplo, si tomamos la curva parabólica, se le concederá mucho más valor medio de puntuación al agente, que en el caso de la exponencial, es por ello que tenemos que ponderar esta recompensa, teniendo en cuenta la curva.

6.2.7 Entrenamiento

El entrenamiento supone uno de los mayores problemas a la hora de realizar el objetivo a lograr. Este depende mucho de las condiciones de inicio del sistema, de los errores cometidos previamente, los reinicios realizados y del movimiento de la celda, junto con los posibles valores a tomar de los actuadores.

Debido a que la obtención de los valores de entrenamiento no son posibles de obtener con Unity, y mediante el paquete Mlagent tampoco es posible, debemos de usar el paquete Tensorboard para obtener los datos de entrenamiento del sistema. Debido a la incompatibilidad de realizar el entrenamiebnto y la obtención de datos dentro del mismo entorno del ordenador, se crea un nuevo entorno en Anaconda 3 para la obtención de los mismos. A continuación mostramos el código a realizar en nuestra entorno para tener en el puerto 6006 de nuestro ordenador, los dato que se obtienen para registrar cómo se encuentra el entrenamiento del mismo.

Código 6.1 Activación del entorno y ejecución de TensorBoard.

```
# Activar el entorno de Conda
conda activate entorno_tensorboard_entrenamiento

# Ejecutar TensorBoard para ver los resultados generales
tensorboard --logdir results --port 6006

# Ejecutar TensorBoard para un entrenamiento específico en una ruta de red
tensorboard --logdir Z:\2TFM\ManufacturingCell-main\DigitalTwin2\config\
    results\MyTrainingRun10\ConveyorBeltAgent --port 6006
```

Para el funcionamiento del entrenamiento de la red neuronal, se deben de realizar los siguientes pasos:

1. Deberemos de iniciar el funcionamiento del script simular la obtención de los datos en tiempo real de la posición de la bandeja en la célula de fabricación, código en el anexo A.2. Este código se estará ejecutando durante toda la sesión de entrenamiento, por lo que mínimo deberá tener una duración simular para que siga teniendo datos de entrada.
2. Se ejecuta, en una nueva ventana de Anaconda 3, el código necesario para el funcionamiento del entrenamiento en esta ventana se encuentra en el código 6.2:
3. Se ejecuta el modo juego dentro de Unity para que se conecte el módulo de entrenamiento en el puerto 5001 con los datos que debe de obtener y controlar de Unity.

Con esto, se empieza a entrenar el sistema para que vaya modificando y actuando sobre los actuadores. Mientras se va ejecutando, los valores obtenidos se van almacenando dentro del puerto 6006 con Tensorboard en donde se muestran en distintas ventanas el rendimiento del entrenamiento, mientras que en Unity podemos ir viendo y observando como se ejecuta el entrenamiento.

Código 6.2 Activación del entorno de entranmiento de la red neuronal.

```
# Activar el funcionamiento de código en Python
set PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION=python

# Buscamos la carpeta en donde se encuentra el archivo con los hiperparámetros
     de entrenamiento
cd Z:\2TFM\ManufacturingCell-main\DigitalTwin2\config

# Ejectuamos el entrenamiento con el archivo de los hiperparámetros y forzamos
     para sobreescribir.
mlagents-learn Z:\2TFM\ManufacturingCell-main\DigitalTwin2\config\
    training_configv2.yaml --run-id=MyTrainingRun10 --force
```

6.2.8 Problemas en los resultados en gemelo digital

Finalmente, tras muchos intentos para obtener una red neuronal que sea capaz de lograr alcanzar la bandeja de referencia, comprobamos que no es posible debido a que el entrenamiento en el mismo hace que la ejecución de la red neuronal de entrenamiento, junto con el modelo del sistema, no sea capaz de tener unos resultados coherentes y un entrenamiento en un tiempo prudencial. Debido a la tasa de observaciones que queremos tener, el entrenamiento de la red neuronal duraría 1 semana, sin ninguna mejora en la misma durante más

de un día entrenando, por lo que se da por resultado insatisfactorio en el gemelo debido al rendimiento. Mostramos una captura de pantalla en el que vemos que los frames por segundo que puede trabajar el sistema, y el gran número de elementos en movimiento hace inviable el entrenamiento en este entorno.

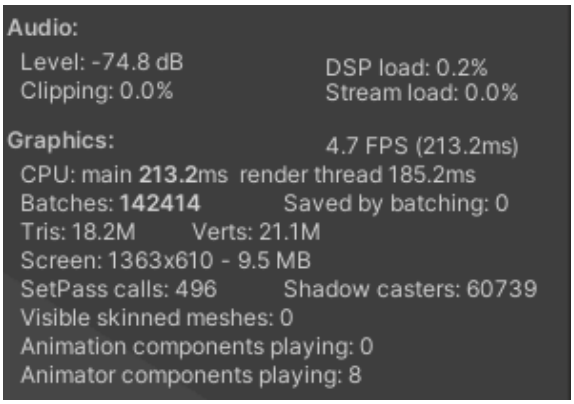

Figura 6.3 Datos obtenido de "stats" de Unity en ejecución sin entrenamiento.

Es por esto que se decide, a partir de los datos mínimos imprescindibles como es el recorrido del sistema y sus dimensiones, crear un entorno prácticamente vacío para mejorar mucho el rendimiento del ordenador y experimentar con más posibilidades de entrenamiento de redes neuronales, los cuales no han sido posibles en el entorno del gemelo.

6.3 Alternativa simplificada en nuevo entorno mínimo y nuevos cambios

Se opta por realizar el siguiente entorno en Unity, eliminando todos los elementos que no son estrictamente necesarios para el desarrollo del objetivo principal que sería el seguimiento de la bandeja de referencia.

Por ello, se eliminan los siguientes problemas del anterior entorno:

- Eliminación de elementos decorativos del gemelo digital.
- Eliminación de pistones, control considerado en automático por el sistema ya que tiene una lógica simple (detección del sensor).
- Eliminación de cintas, ya que su representación son miles de pequeños bloques en movimiento y renderizándose por completo.
- Eliminación de impactos de objetos y de gravedad.
- Eliminación de comunicaciones externas a Unity, simulando su movimiento en el mismo para menor consumo de recursos.

Se señala con elementos alargados el centro de los elevadores, simulando las esquinas de la célula de fabricación, manteniendo siempre la misma escala que el gemelo digital. Se muestra a continuación una imagen del mismo.

En este caso, como en el anterior, la bandeja roja sería la referencia, la bandeja dorada en este caso sería la bandeja a controlarse, y tendríamos las 4 esquinas señaladas como los vértices de las cintas en cuestión.

Se intenta asemejar lo máximo posible al código y entorno del gemelo para que, en el caso de que funcione,se pueda usar la red neuronal entrenada en este entorno en el gemelo digital, con el objetivo de que la red neuronal entrenada sea capaz de funcionar idénticamente en el modelo completo.

Además, añadimos dos nuevas funcionalidades no vistas hasta el momento en mlagent previamente, que serán los módulos "demostration recorder" y "curriculum".

6.3.1 Demostration recorder

En este módulo, mediante el uso de la función heurística, implementar en una demostración el resultado de las observaciones y las acciones que queremos que realice la red neuronal.

En este caso, sería un entrenamiento supervisado o guiado, ya que le enseñamos a la red neuronal lo que queremos tratar de conseguir. Teniendo un número elevado de episodios con una parametrización correcta

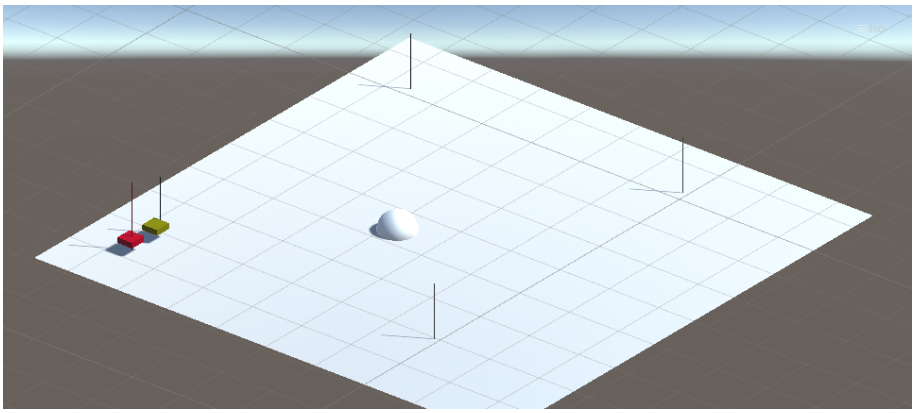

Figura 6.4 Vista del gemelo digita mínimo.

con acciones sobre la distancia entre objetivos. Esto facilitará en las primeras etapas de la red a tener una guía con la cuál comparar sus resultados.

6.3.2 Curriculum

Con el entrenamiento currículum, conseguimos realizar las fases de entrenamiento por etapas. El objetivo es, mediante entrenamientos progresivos, cada vez más complejo en el tiempo, ir entrenamiendo a la red por partes para lograr que fase a fase, aprenda el objetivo final del modelo.

Este entrenamiento junto con las demostraciones, hace que la red neuronal aprenda de una manera más efectiva, ya que el entrenamiento no supervisado y sin etapas hemos comprobado que no ha tenido buenos resultados.

6.3.3 Distancia sobre cintas

Con el objetivo de mejorar el entrenamiento, tomaremos finalmente la distancia entre objetivos medida sobre el patrón de movimiento que tienen las bandejas, ya que la distancia euclídea es más sencilla y consume menos recursos,pero en las esquinas deja de funcionar correctamente, la distancia se incrementa, problema aún más complejo para la red neuronal. Por ello, medimos la distancia objetivo sobre carriles con dos signos, ya que no es lo mismo que la bandeja de referencia se encuentre por delante de la bandeja a controlar, ya que el objetivo será esperar a la bandeja de referencia, que la bandeja de referencia se encuentre por delante, cuyo objetivo será alcanzar y seguir su trayectoria.

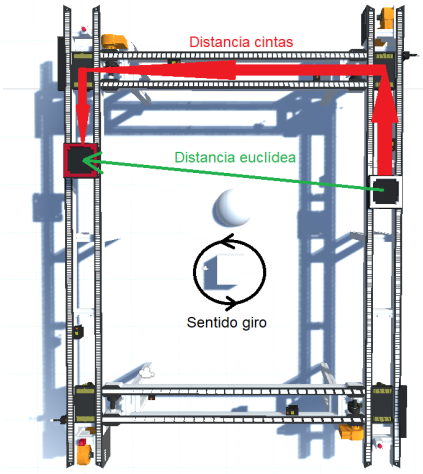

Figura 6.5 Posibles mediciones de distancia objetivo.

6.3.4 Resultado final

Como resultado final, modificando los hiperparámetros tras varios intentos, conseguimos finalmente entrenar una red neuronal que imita correctamente la demostración realizada mediante heurística. Se muestra a continuación los datos obtenidos mediante tensorboard:

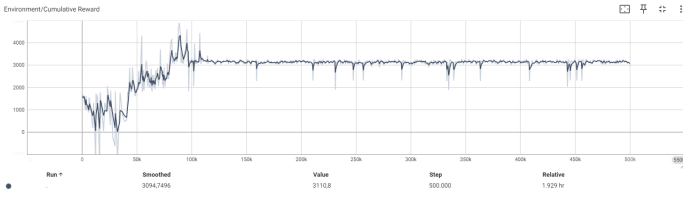

Figura 6.6 Recompensa acumulada en el entrenamiento.

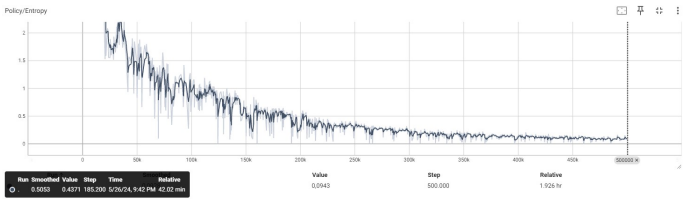

Figura 6.7 Entropía del entreno.

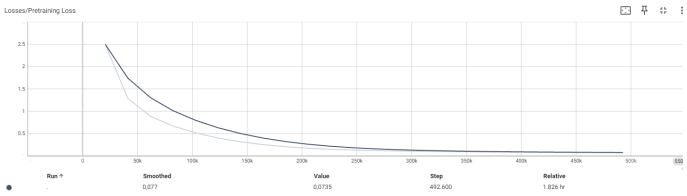

Figura 6.8 Pérdidas sobre el preentrenamiento.

Como vemos los daen tos muestreados, el entrenamiento tiende a un régimen permanente en el que la entropía y las pérdidas se minimizan, y la recompensa se mantiene constante y en aumento.

El entrenamiento final que ha obtenido resultados ha sido mediante los siguientes puntos críticos necesarios:

- Demostración de patrón de movimiento mediante heurística durante varios episodios
- Disminución progresiva de la distancia objetivo entre ambas bandejas.
- Aumento progresivo del tiempo dentro del rango objetivo entre ambas bandejas.
- Parametrización de los hiperparámetros como una salida continua, a pesar de ser discreta, debido al alto número de observaciones y acciones tomadas (10 observaciones/acciones por segundo)
- Curriculum de 5 fases, en el que se realizan lo anteriormente descrito entorno a las variables de distancia y tiempo.
- Observaciones de dimensión 7: Distancia entre bandejas, velocidades de ambas bandejas sobre los carriles y posición (x,y) de ambas bandejas.
- Actuación limitada a speedAction, variable que controla la velocidad de las cintas de transporte.
- Lógica lineal en las recompensas, con una recompensa fija de 3 menos la distancia objetivo medida, siendo máximo la mitad de la longitud completa de los carriles.

7 Conclusiones

Este capítulo sintetiza los logros principales obtenidos durante la realización del proyecto, destacando las contribuciones significativas al campo de la visión por computadora aplicada al desarrollo de gemelos digitales y la comunicación entre varios dispositivos y diversas fuentes de información para el entrenamiento del gemelo, proporcionando una evaluación crítica del trabajo realizado y sus posibles aplicaciones.

Se quiere resaltar que, a pesar del trabajo realizado y de sus posibles líneas de investigación futuras, vemos que es un campo del que se requiere un conocimiento avanzado del problema a solventar. Debido a la complejidad del problema en cuestión, se ha tenido que realizar más simplificaciones de las ideadas en una primera planificación del problema, todo esto adaptado a un entorno como es Unity, en el cual se desarrollan otros tipos de entornos y estando pensado en un primer inicio para el desarrollo de videojuegos, se pueden realizar simulaciones muy logradas y que tienen una muy buena base para realizar el trabajo que hemos hecho en el mismo.

Si bien hemos realizado en este trabajo dos vertientes importantes, que son la comunicación y sincronización de elementos externos dentro de un gemelo digital y el entrenamiento mediante redes neuronales para el funcionamiento autónomo y aprendizaje del mismo, es posible la mejora de la misma en otros aspectos más automatizables que los propuestos en este trabajo.

Tal como se ha descrito en la memoria de este trabajo, en el primer apartado del trabajo se ha resuelto satisfactoriamente la comunicación de todos los elementos de los que disponía, por lo que esta parte se considera resulta y sin problemas para avanzar. En contraparte, la segunda parte del trabajo, en la que realizamos la red neuronal para automatizar procesos, vemos que ha sido mucho más complejo que la primera parte, se requiere de mucha más dedicación de tiempo debido a que la automatización mediante recompensas es un aspecto diseñado en un primer momento para crear inteligencias artificales para videojuegos y agentes que puedan interactuar con elementos, por lo que en un primer momento para este tipo de aplicaciones pueden ser complejo el cómo programar y realizar el tratamiento de recompensar/penalizar al agente que manipula los elementos de fabricación, pudiendo ser más simple, la simplificación de la toma de decisiones a distintos agentes que controlen únicamente una variable en función de los datos que se obtienen, más que obtener un agente más complejo que intente comprender el funcionamiento completo de la célula de fabricación.

A pesar de haber conseguido el objetivo logrado, no ha sido posible tratar de corregir los problemas inherentes que hemos detectado en el gemelo digital, por lo que se tendrá que partir del mismo como un modelo preeentrenado,por lo que se complicaría más el problema de aprendizaje, ya que deberíamos de realizar de nuevo la demostración en el mismo junto con el curriculum, objetivo fuera finalmente del alcance del proyecto. Sin embargo, hemos demostrado la posibilidad del mismo en un entorno más simple, por lo que queda claro que es posible realizarlo y que solventaría parte de los problemas que tenemos en el primer proyecto.

Como nuevas líneas de investigación del trabajo, se pueden trabajar tanto en el campo de la comunicación, como en la automatización y entrenamiento de las redes neuronales, entre otros, podrían ser alguno de los siguientes:

- Automatización independiente de cada motor de cada cinta.
- Mejora en la seguridad y la transmisión de datos con el gemelo digital.
- Implementación de seguimiento de dos o más bandejas en la célula de fabricación.
- Detección de errores en los sensores de la célula de fabricación.

- Seguimiento e identificación de nuevas bandejas con piezas en zona superior.
- Movimiento inverso de las cintas de trabajo.
- Implementación de la célula de fabricación al completo y no versión simplificada.
- Modificación y búsqueda de puntos críticos en la célula de fabricación.
- Mantenimiento predictivo de la célula de fabricación de sus elementos móviles.

Estos son algunos posibles avances del trabajo planteado, posibilidades que una vez perfeccionado el gemelo digital en este caso y resuelto varios de los problemas encontrados en la versión actual del gemelo digital de la célula de fabricación, suponga una mejor base para crear nueva redes neuronales artificiales más complejas y con más aplicación en un futuro que la actual creada en el trabajo.

Con esto, terminamos de realizar un trabajo muy complejo dividido en dos trabajos independientes que, en conjunto, han requerido de gran trabajo, que han supuesto la adquisición de un gran conocimiento respecto a campos tan variados, desde el manejo de impresión 3D, realizando redes neuronales para distintos motivos, desde la visión artificial, como la automatización de procesos industriales, que han acabado dando lugar al entrenamiento dentro de un gemelo digital de la célula de fabricación ubicada en las instalaciones de la universidad.

Apéndice A
Códigos en Python

A.1 Creación de trayetoria

```
import json
import matplotlib.pyplot as plt


# Función para calcular la velocidad basada en la diferencia de posición
def calcular_velocidad(pos_actual, pos_anterior):
    return [pos_actual[0] - pos_anterior[0], pos_actual[1] - pos_anterior[1]]


# Función para graficar la posición X y Y por separado
def graficar_posiciones_separadas(datos_id1,datos_id2):
    # Extraer tiempos, posiciones X y Y para ambos IDs
    tiempos_id1 = list(range(len(datos_id1)))
    posiciones_x_id1 = [d['position'][0] for d in datos_id1]
    posiciones_y_id1 = [d['position'][1] for d in datos_id1]

    tiempos_id2 = list(range(len(datos_id2)))
    posiciones_x_id2 = [d['position'][0] for d in datos_id2]
    posiciones_y_id2 = [d['position'][1] for d in datos_id2]

    # Crear gráficas para la posición X
    plt.figure(figsize=(8, 6))
    plt.plot(tiempos_id1, posiciones_x_id1, label='Posición X ID 1')
    plt.plot(tiempos_id2, posiciones_x_id2, label='Posición X ID 2')
    plt.title('Posición X en función del tiempo')
    plt.xlabel('Tiempo (frames)')
    plt.ylabel('Posición X')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Crear gráficas para la posición Y
    plt.figure(figsize=(8, 6))
    plt.plot(tiempos_id1, posiciones_y_id1, label='Posición Y ID 1')
    plt.plot(tiempos_id2, posiciones_y_id2, label='Posición Y ID 2')
    plt.title('Posición Y en función del tiempo')
    plt.xlabel('Tiempo (frames)')
    plt.ylabel('Posición Y')
```

```
    plt.legend()
    plt.grid(True)
    plt.show()


def graficar_datos(datos):
    # Filtrar los datos por ID
    datos_id1 = [d for d in datos if d['id'] == 1]
    datos_id2 = [d for d in datos if d['id'] == 2]

    # Extraer posiciones y velocidades para cada ID
    posiciones_x_id1 = [d['position'][0] for d in datos_id1]
    posiciones_y_id1 = [d['position'][1] for d in datos_id1]
    velocidades_y_id1 = [d['velocity'][1] for d in datos_id1]

    posiciones_x_id2 = [d['position'][0] for d in datos_id2]
    posiciones_y_id2 = [d['position'][1] for d in datos_id2]
    velocidades_y_id2 = [d['velocity'][1] for d in datos_id2]

    # Crear la figura para la trayectoria
    plt.figure(figsize=(8, 6))
    plt.plot(posiciones_x_id1, posiciones_y_id1, label='ID 1')
    plt.plot(posiciones_x_id2, posiciones_y_id2, label='ID 2')
    plt.title('Trayectoria en el plano XY')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Crear la figura para la velocidad
    plt.figure(figsize=(8, 6))
    plt.plot(range(len(velocidades_y_id1)), velocidades_y_id1, label='Velocidad
        Y ID 1')
    plt.plot(range(len(velocidades_y_id2)), velocidades_y_id2, label='Velocidad
        Y ID 2')
    plt.title('Velocidad en el eje Y')
    plt.xlabel('Tiempo (frames)')
    plt.ylabel('Velocidad Y')
    plt.legend()
    plt.grid(True)
    plt.show()

# Inicializar datos
datos = []
posicion = [0.1, 0.1]
posicion_anterior = [0.1, 0.1]
longitud = 2.0
altura = 3.0
paso = (longitud + altura) * 2 / 200 # Definir el paso para 200 datos en el per
    ímetro

# Generar trayectoria para el id 1
for i in range(200):
    if i < 50:
        posicion[0] += paso # Mover en x
    elif i < 100:
```

```
        posicion[1] += paso # Mover en y
    elif i < 150:
        posicion[0] -= paso # Mover en x negativo
    else:
        posicion[1] -= paso # Mover en y negativo

    velocidad = calcular_velocidad(posicion, posicion_anterior)
    datos.append({'id': 1, 'position': posicion[:], 'velocity': velocidad})
    posicion_anterior = posicion[:]

# Generar trayectoria para el id 2, empezando después de la primera esquina
for i in range(50, 200):
    if i < 100:
        posicion[0] += paso # Mover en x
    elif i < 150:
        posicion[1] += paso # Mover en y
    elif i < 200:
        posicion[0] -= paso # Mover en x negativo
    elif i < 250:
        posicion[1] -= paso # Mover en y negativo

    velocidad = calcular_velocidad(posicion, posicion_anterior)
    datos.append({'id': 2, 'position': posicion[:], 'velocity': velocidad})
    posicion_anterior = posicion[:]

# Eliminar datos extra para id 2
datos = datos[:400] # Nos aseguramos de tener solo 200 datos para cada ID

# Guardar los datos en un archivo JSON
with open('datos.json', 'w') as archivo:
    json.dump(datos, archivo, indent=4)

# Llamada a la función de graficación
graficar_datos(datos)
# Llamada a la función de graficación para posiciones separadas
# Llamada a la función de graficación para posiciones separadas
datos_id1 = [d for d in datos if d['id'] == 1]
datos_id2 = [d for d in datos if d['id'] == 2]
graficar_posiciones_separadas(datos_id1, datos_id2)
graficar_posiciones_separadas(datos)

print("Archivo 'datos.json' generado con éxito.")
```

A.2 Comunicación servidor en PyCharm

```
from opcua import Server
import time
from opcua import ua
import random
import json

# Inicializar servidor OPC UA
server = Server()
```

```
# Configurar endpoint y nombre
server.set_endpoint("opc.tcp://127.0.0.1:48484/freeopcua/server/")
server.set_server_name("Python OPC UA Simulation Server")
# Establece la política de seguridad en None
# Configuración de política de seguridad para no usar seguridad (None)
server.set_security_policy([ua.SecurityPolicyType.NoSecurity])

# Configurar espacio de nombres
uri = "urn:DESKTOP-O7FSI3H:UnityOPCUAClient"
idx = server.register_namespace(uri)

# Crear una lista para mantener un registro de los IDs existentes
existing_ids = []

node_ids = {}

# Iniciar el servidor
try:
    print("Start server")
    server.start()
    print("Endpoints : ", str(server.get_endpoints()).replace(',', '\n'))
    time.sleep(10)
    first_iteration = 1
    while first_iteration==1:
        # Asumir que 'datos' es una lista de diccionarios que obtienes de
            alguna fuente
        with open('C:/Users/Ceravedi/Desktop/UNIVERSIDAD/Doble_Master/TFMS/
            Segundo TFM/Codigo/datos7/datos7.json', 'r') as archivo:
            datos = json.load(archivo)

            # Dentro del bucle while True:

            for dato in datos:
                id_ = dato['id']
                position = dato['position']
                velocity = dato['velocity']

                # Definir los identificadores de nodo para cada atributo como
                    numéricos
                pos_x_identifier = ua.NodeId(id_ * 1000 + 1, idx)
                pos_y_identifier = ua.NodeId(id_ * 1000 + 2, idx)
                vel_x_identifier = ua.NodeId(id_ * 1000 + 3, idx)
                vel_y_identifier = ua.NodeId(id_ * 1000 + 4, idx)

                # Almacenar en el diccionario
                node_ids[id_] = {
                    'pos_x': pos_x_identifier,
                    'pos_y': pos_y_identifier,
                    'vel_x': vel_x_identifier,
                    'vel_y': vel_y_identifier,
                }

                if id_ not in existing_ids:
                    # Crear un nuevo objeto con sus variables internas
                    obj = server.nodes.objects.add_object(idx, f"Objeto{id_}")

                    # Crear variables individuales para la posición y velocidad
```

```
                    var_pos_x = obj.add_variable(pos_x_identifier, "PosX",
                        position[0])
                    var_pos_y = obj.add_variable(pos_y_identifier, "PosY",
                        position[1])
                    var_vel_x = obj.add_variable(vel_x_identifier, "VelX",
                        velocity[0])
                    var_vel_y = obj.add_variable(vel_y_identifier, "VelY",
                        velocity[1])


                    # Hacer las variables escribibles
                   #var_pos_x.set_writable()
                    #var_pos_y.set_writable()
                    #var_vel_x.set_writable()
                    #var_vel_y.set_writable()

                    existing_ids.append(id_)
                else:
                    # Actualizar datos del objeto existente
                    server.get_node(node_ids[id_]['pos_x']).set_value(ua.Variant(
                        position[0], ua.VariantType.Double))
                    server.get_node(node_ids[id_]['pos_y']).set_value(ua.Variant(
                        position[1], ua.VariantType.Double))
                    server.get_node(node_ids[id_]['vel_x']).set_value(ua.Variant(
                        velocity[0], ua.VariantType.Double))
                    server.get_node(node_ids[id_]['vel_y']).set_value(ua.Variant(
                        velocity[1], ua.VariantType.Double))
                    time.sleep(0.1)

                    #print(f"Objeto ID: {obj.get_browse_name().Name}")
                    #print(f"Posición X: {obj.get_child(f'{idx}:PosX{id_}').
                        get_value()}")
                    #print(f"Posición Y: {obj.get_child(f'{idx}:PosY{id_}').
                        get_value()}")
                    #print(f"Velocidad X: {obj.get_child(f'{idx}:VelX{id_}').
                        get_value()}")
                    #print(f"Velocidad Y: {obj.get_child(f'{idx}:VelY{id_}').
                        get_value()}")
            print("fin de datos, me voy a la cama")

            first_iteration = 1 # 1 no va a acabar nunca, 0 acaba

except Exception as e:
    print(e)
finally:
    # Cerrar el servidor de forma segura
    server.stop()
    print("Adio")
```

Apéndice B
Códigos en C#

B.1 Método de entrenamiento en gemelo digital

```
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Actuators;
using UnityEngine;
using System.Collections.Generic;


public class ConveyorBeltAgent : Agent
{
    private Vector3 posicionInicialBox;
    private Quaternion rotacionInicialBox; // Agrega esta línea
    private Rigidbody rbBox; // Agrega esta línea
    public float speedAction;
    public float[] previousSpeedActions = new float [10];
    public bool avance; // Indicara si debe de avanzar o no
    public bool[] previousSensorStates;

    public bool[] previousPiston1Position = new bool[10];
    public bool[] previousPiston2Position = new bool[10];
    public bool[] previousPiston3Position = new bool[10];
    public bool[] previousPiston4Position = new bool[10];

    public bool triggerpiston1 = false;
    public bool triggerpiston2 = false;
    public bool triggerpiston3 = false;
    public bool triggerpiston4 = false;

    public bool triggerdireccionpiston4 = false;
    public bool triggerdireccionpiston2 = false;

    public bool triggertransicionpiston1 = false;
    public bool triggertransicionpiston2 = false;
    public bool triggertransicionpiston3 = false;
    public bool triggertransicionpiston4 = false;

    private Vector3 originalPositionArribaPiston2;
    private Vector3 originalPositionArribaPiston4;
    private Quaternion originalRotationArribaPiston2;
```

```
    private Quaternion originalRotationArribaPiston4;

    private Dictionary<Transform, (Vector3, Quaternion)> initialState = new
        Dictionary<Transform, (Vector3, Quaternion)>();

    private float temporizador;
    private bool isTimerActive = false;

    // Definiciones para el seguimiento de las posiciones anteriores
    public Vector3[] previousBoxPositions = new Vector3[30];
    public Vector3[] previousReferenceBoxPositions = new Vector3[10];

    public GameObject box; // CAJA (5)
    public GameObject referenceBox; // CAJA REFERENCIA
    public InductiveSensor[] inductiveSensors;

    public GameObject arribaPiston2;
    public GameObject arribaPiston4;

    public ChangePistonModule piston1;
    public ChangePistonModule piston2;
    public ChangePistonModule piston3;
    public ChangePistonModule piston4;

    public enum ConveyorLocation
    {
        LongBeltConveyor1,
        LongBeltConveyor0,
        ShortBeltConveyor0,
        ShortBeltConveyor1,
        OutOfConveyor
    }

    enum State
    {
        L0, L1, S0, S1, L02S1, S02L0, S02L1, L12S0, S12L0, S12L1
    }
    private State currentState;

    void Start()
    {
        rbBox = box.GetComponent<Rigidbody>();
        currentState = State.L1; // Estado inicial
        posicionInicialBox = box.transform.position;
        rotacionInicialBox = box.transform.rotation;
        previousSensorStates = new bool[inductiveSensors.Length];

        // Almacenar el estado inicial de arribaPiston2 y todos sus hijos
        AddToInitialStateDict(arribaPiston2.transform);
        // Almacenar el estado inicial de arribaPiston4 y todos sus hijos
        AddToInitialStateDict(arribaPiston4.transform);

        // Otras inicializaciones...
        //originalPositionArribaPiston2 = arribaPiston2.transform.position;
        //originalPositionArribaPiston4 = arribaPiston4.transform.position;
        //originalRotationArribaPiston2 = arribaPiston2.transform.rotation;
        //originalRotationArribaPiston4 = arribaPiston4.transform.rotation;
```

```
    InvokeRepeating("LogCumulativeReward", 2f, 2f); // Inicia después de 2
        segundos y repite cada 2 segundos
}

public override void CollectObservations(VectorSensor sensor)
{
    // Añade la observación de la posición de la caja y la caja referencia
    Vector3 boxPos = box.transform.localPosition;
    Vector3 referenceBoxPos = referenceBox.transform.localPosition;
    sensor.AddObservation(boxPos);
    sensor.AddObservation(referenceBoxPos);

    // Determina y añade la ubicación de la caja y la caja de referencia en
         la cinta transportadora
    ConveyorLocation boxLocation = DetermineConveyorLocation(boxPos);
    ConveyorLocation referenceBoxLocation = DetermineConveyorLocation(
        referenceBoxPos);
    sensor.AddObservation((float)boxLocation);
    sensor.AddObservation((float)referenceBoxLocation);
    sensor.AddObservation((bool)piston1.IsPistonElevated());
    sensor.AddObservation((bool)piston2.IsPistonElevated());
    sensor.AddObservation((bool)piston3.IsPistonElevated());
    sensor.AddObservation((bool)piston4.IsPistonElevated());
    sensor.AddObservation((int)speedAction);


    // Añadir la observación de cada sensor inductivo al vector de
        observaciones
    for (int i = 0; i < inductiveSensors.Length; i++)
    {
        // Obtiene el estado crudo del sensor inductivo
        bool currentSensorState = inductiveSensors[i].IsMetalDetected();

        // Determina si el sensor actual debe ser tratado como especial
        bool isSpecialSensor = (i == 3 || i == 8); // Asume que los sensores
             especiales son el 4º y 9º

        // Añade la observación, invierte la lógica para los sensores
            especiales directamente
        sensor.AddObservation((isSpecialSensor ? !currentSensorState :
            currentSensorState) ? 1.0f : 0.0f);

        // Asigna el estado crudo a previousSensorStates
        previousSensorStates[i] = currentSensorState;
    }

}

public override void OnActionReceived(ActionBuffers actionBuffers)
{
    Vector3 boxPos = box.transform.localPosition;
    Vector3 referenceBoxPos = referenceBox.transform.localPosition;
    // Distancia eliminando componente en altura
    Vector2 boxPos2 = new Vector2(boxPos.x, boxPos.z);
    Vector2 referenceBoxPos2 = new Vector2(referenceBoxPos.x,
        referenceBoxPos.z);
```

```
// Iniciar acciones de los pistones
int beltDirectionActionPiston1 = actionBuffers.DiscreteActions[0];
bool pistonPositionActionPiston1 = actionBuffers.DiscreteActions[1] ==
    1;

int beltDirectionActionPiston2 = actionBuffers.DiscreteActions[2];
bool pistonPositionActionPiston2 = actionBuffers.DiscreteActions[3] ==
    1;

int beltDirectionActionPiston3 = actionBuffers.DiscreteActions[4];
bool pistonPositionActionPiston3 = actionBuffers.DiscreteActions[5] ==
    1;

int beltDirectionActionPiston4 = actionBuffers.DiscreteActions[6];
bool pistonPositionActionPiston4 = actionBuffers.DiscreteActions[7] ==
    1;

int discreteSpeedAction = actionBuffers.DiscreteActions[8];

if (StepCount >= MaxStep - 1)
{
    ReiniciarTriggers();
}


speedAction = discreteSpeedAction*0.25f; // Ajusta el valor al rango de
     0 a +4
ShortBeltConveyorModel.Parameters.Speed = speedAction;
LongBeltConveyorModel.Parameters.Speed = speedAction;

//Zona de recompensas
if (speedAction > 0) AddReward(0.05f); // avanzar es positivo
if (Mathf.Abs(speedAction - previousSpeedActions[1]) > 2) AddReward(-3f)
    ; //no quiero movimientos bruscos
if (Mathf.Abs(speedAction - previousSpeedActions[1]) == 0) AddReward
    (0.2f);
if (speedAction != 0 && (Mathf.Abs(previousBoxPositions[0].x -
    previousBoxPositions[1].x) < 0.001 && Mathf.Abs(previousBoxPositions
    [0].z - previousBoxPositions[1].z) < 0.001)) AddReward(-1.5f); // No
     quiero que se quede quiero y sume puntos
int cambios = 0;
//Zona recompensas por estados
switch (currentState) // Dar recompensas en función del estado en el
    que se encuentre
{
    case State.L1: // L1 , ESTADO INICIAL
        ResetTriggersL1();

        // Comprueba la condición específica del pistón 3 sin repetir la
             comprobación de elevación
        if (!triggerpiston3)
        {
            if (piston3.IsPistonElevated())
            {
                // Establece la posición del pistón solo si la acción es
                    'false', es decir, bajar el pistón
```

```
                if (!pistonPositionActionPiston3)
                {
                    piston3.SetPistonPosition(pistonPositionActionPiston3)
                        ;
                    AddReward(30.0f);

                }
                else
                {
                    AddReward(-4.0f);
                }
            }
        }
        break;   // FIN L1


    case State.L12S0: //TRANSICIÓN DE L1 A S0

        if (!isTimerActive)
        {
            temporizador = Time.time; // Inicia el temporizador
            isTimerActive = true;
        }
        piston3.SetBeltDirection(beltDirectionActionPiston3 - 1);
        if (previousSensorStates[5] && !triggerpiston3)
        {
            cambios = ContarCambios(previousPiston3Position);
            AddReward(cambios > 2 ? -1.0f : 10.0f);

            //piston3.SetBeltDirection(beltDirectionActionPiston3 - 1);
            piston3.SetPistonPosition(pistonPositionActionPiston3);

            // Evalúa la condición para la recompensa positiva
            if (piston3.IsPistonElevated() /*&&
                beltDirectionActionPiston3 == -1*/)
            {
                AddReward(30.0f);
                triggerpiston3 = true; // Activa el trigger para evitar
                    futuras ejecuciones
            }
            else
            {
                AddReward(-4.0f); // Aplica una penalización en todos los
                     demás casos
            }
        }

        break;  //FIN TRANSICIÓN DE L1 A S0

    case State.S0: // S0
        triggertransicionpiston1 = false;
        triggertransicionpiston2 = false;
        triggertransicionpiston3 = false;
        triggertransicionpiston4 = false;

        // Lógica para el pistón 4
```

```
                if (!triggerpiston4) // Verifica si el trigger para el pistón 4
                    aún no se ha activado
                {
                    if (piston4.IsPistonElevated() == false)
                    {
                        if (pistonPositionActionPiston4) // Si la acción es
                            elevar el pistón 4
                        {
                            piston4.SetPistonPosition(pistonPositionActionPiston4)
                                ; // Eleva el pistón 4
                            AddReward(30.0f); // Añade una recompensa positiva
                            triggerpiston4 = true; // Activa el trigger para
                                evitar futuras recompensas/acciones
                        }
                        else // Si la acción no es elevar el pistón 4
                        {
                            AddReward(-4.0f); // Penalización por no elevar el
                                pistón cuando se esperaba
                        }
                    }
                }


                // Lógica para el pistón 3
                if (!triggerpiston3) // Verifica si el trigger para el pistón 3
                    aún no se ha activado
                {
                    if (piston3.IsPistonElevated() == false)
                    {
                        if (pistonPositionActionPiston3) // Si la acción es
                            elevar el pistón 3
                        {
                            piston3.SetPistonPosition(pistonPositionActionPiston3)
                                ; // Eleva el pistón 3
                            AddReward(30.0f); // Añade una recompensa positiva
                            triggerpiston3 = true; // Activa el trigger para
                                evitar futuras recompensas/acciones
                        }
                        else // Si la acción no es elevar el pistón 3
                        {
                            AddReward(-4.0f); // Penalización por no elevar el
                                pistón cuando se esperaba
                        }
                    }
                }

                // Calcular la recompensa basada en el tiempo si el contador est
                    á activo
                if (isTimerActive)
                {
                    float tiempoensegundos = Time.time - temporizador; // Calcula
                         el tiempo transcurrido
                    Debug.Log("Tiempo en segundos: " + tiempoensegundos); //
                        Muestra el tiempo transcurrido en la consola de Unity
                    //AddReward(2000 / tiempoensegundos); // Añade la recompensa
                        basada en el tiempo
                    isTimerActive = false; // Desactiva el contador
```

```
            }

            break; // FIN S0

        case State.S02L0: //TRANSICIÓN S0 A L0
            triggerpiston3 = false;
            triggerpiston4 = false;


            // Lógica para la dirección del pistón 4 basada en el estado del
                 sensor 8 y el trigger
            if (previousSensorStates[8] && !triggerdireccionpiston4)
            {
                if (beltDirectionActionPiston4 == -1)
                {
                    piston4.SetBeltDirection(beltDirectionActionPiston4);
                    AddReward(10.0f);
                    triggerdireccionpiston4 = true; // Activa el trigger para
                         evitar futuras recompensas/acciones
                }
                else if (beltDirectionActionPiston4 == 0)
                {
                    AddReward(-0.5f); // Penalización si la dirección de la
                        cinta es 0 cuando el sensor está activo
                }
            }



            // Lógica para la transición del pistón 4 basada en el estado
                del sensor 8 y el trigger
            if (!previousSensorStates[8] && !triggertransicionpiston4) //
                Verifica si el sensor 8 está desactivado y el trigger de
                transición del pistón 4 aún no se ha activado
            {
                cambios = ContarCambios(previousPiston4Position);
                AddReward(cambios > 2 ? -0.5f : 5.0f);

                piston4.SetBeltDirection(beltDirectionActionPiston4);
                piston4.SetPistonPosition(pistonPositionActionPiston4);

                // Otorga una recompensa basada en si el pistón 4 se eleva o
                    no, y activa el trigger según sea necesario
                if (!piston4.IsPistonElevated())
                {
                    AddReward(30.0f); // Recompensa por elevar el pistón 4
                        cuando el sensor 8 está desactivado
                    triggertransicionpiston4 = true; // Evita futuras
                        ejecuciones de este bloque
                }
                else
                {
                    AddReward(-4.0f); // Penalización si el pistón 4 está
                        elevado cuando el sensor 8 está desactivado
                }
            }
```

```
                break;

            case State.L0: //L0
                triggerpiston2 = false;
                triggerpiston3 = false;
                triggerpiston4 = false;
                triggertransicionpiston1 = false;
                triggertransicionpiston2 = false;
                triggertransicionpiston3 = false;
                triggertransicionpiston4 = false;

                if (piston1.IsPistonElevated() && !triggerpiston1) // Verifica
                    si el pistón 1 está elevado y el trigger aún no ha sido
                    activado
                {
                    // Si la acción es bajar el pistón 1 y aún no se ha activado
                        el trigger
                    if (!pistonPositionActionPiston1)
                    {
                        piston1.SetPistonPosition(false); // Baja el pistón 1
                        //triggerpiston1 = true; // Activa el trigger para evitar
                             futuras ejecuciones
                        AddReward(30.0f); // Añade una recompensa positiva
                    }
                    else
                    {
                        AddReward(-4.0f); // Penalización si el pistón 1 está
                            elevado pero la acción no es bajarlo
                    }
                }
                break;

            case State.L02S1: //TRANSICIÓN L0 A S1

                piston1.SetBeltDirection(beltDirectionActionPiston1 - 1);
                if (previousSensorStates[0] && !triggerpiston1)
                {
                    cambios = ContarCambios(previousPiston1Position);
                    AddReward(cambios > 2 ? -0.5f : 3.0f);

                    //piston1.SetBeltDirection(beltDirectionActionPiston1 - 1);
                    piston1.SetPistonPosition(pistonPositionActionPiston1);

                    // Evalúa la condición para la recompensa positiva
                    if (piston1.IsPistonElevated() /*&&
                        beltDirectionActionPiston1 == -1*/)
                    {
                        AddReward(30.0f);
                        triggerpiston1 = true; // Activa el trigger para evitar
                            futuras ejecuciones
                    }
                    else
                    {
                        AddReward(-4.0f); // Aplica una penalización en todos los
                             demás casos
                    }
                }
```

```
            break;

        case State.S1: // ESTADO S1

            triggerpiston1 = false;
            triggerpiston2 = false;
            triggerpiston3 = false;
            triggerpiston4 = false;
            triggertransicionpiston1 = false;
            triggertransicionpiston2 = false;
            triggertransicionpiston3 = false;
            triggertransicionpiston4 = false;

            // Lógica para el pistón 2
            if (!triggerpiston2) // Verifica si el trigger para el pistón 2
                aún no se ha activado
            {
                if (piston2.IsPistonElevated() == false)
                {
                    if (pistonPositionActionPiston2) // Si la acción es
                        elevar el pistón 2
                    {
                        piston2.SetPistonPosition(pistonPositionActionPiston2)
                            ; // Eleva el pistón 2
                        AddReward(30.0f); // Añade una recompensa positiva
                        triggerpiston2 = true; // Activa el trigger para
                            evitar futuras recompensas/acciones
                    }
                    else // Si la acción no es elevar el pistón 2
                    {
                        AddReward(-4.0f); // Penalización por no elevar el
                            pistón cuando se esperaba
                    }
                }
            }

            // Lógica para el pistón 1
            if (!triggerpiston1) // Verifica si el trigger para el pistón 1
                aún no se ha activado
            {
                if (piston1.IsPistonElevated() == false)
                {
                    if (pistonPositionActionPiston1) // Si la acción es
                        elevar el pistón 1
                    {
                        piston1.SetPistonPosition(pistonPositionActionPiston1)
                            ; // Eleva el pistón 1
                        AddReward(30.0f); // Añade una recompensa positiva
                        triggerpiston1 = true; // Activa el trigger para
                            evitar futuras recompensas/acciones
                    }
                    else // Si la acción no es elevar el pistón 1
                    {
                        AddReward(-4.0f); // Penalización por no elevar el
                            pistón cuando se esperaba
```

```
                }
            }
        }

        break;

    case State.S12L1: // TRANSICIÓN S1 A L1
        ResetTriggersS12L1();

        // Lógica para la dirección del pistón 2 basada en el estado del
             sensor 3 y el trigger
        if (previousSensorStates[3] && !triggerdireccionpiston2)
        {
            if (beltDirectionActionPiston2 == -1)
            {
                piston2.SetBeltDirection(beltDirectionActionPiston2);
                triggerdireccionpiston2 = true; // Activa el trigger para
                     evitar futuras recompensas/acciones
            }
            else if (beltDirectionActionPiston2 == 0)
            {
                AddReward(-0.5f); // Penalización si la dirección de la
                    cinta es 0 cuando el sensor está activo
            }
        }

        // Lógica para la transición del pistón 2 basada en el estado
            del sensor 3 y el trigger
        if (!previousSensorStates[3] && !triggertransicionpiston2) //
            Verifica si el sensor 3 está desactivado y el trigger de
            transición del pistón 2 aún no se ha activado
        {
            cambios = ContarCambios(previousPiston2Position);
            AddReward(cambios > 2 ? -0.2f : 1.0f);

            piston2.SetBeltDirection(beltDirectionActionPiston2);
            piston2.SetPistonPosition(pistonPositionActionPiston2);

            // Otorga una recompensa basada en si el pistón 2 se eleva o
                no, y activa el trigger según sea necesario
            if (!piston2.IsPistonElevated())
            {
                AddReward(30.0f); // Recompensa por elevar el pistón 2
                    cuando el sensor 3 está desactivado
                triggertransicionpiston2 = true; // Evita futuras
                    ejecuciones de este bloque
            }
            else
            {
                AddReward(-4.0f); // Penalización si el pistón 2 está
                    elevado cuando el sensor 3 está desactivado
            }
        }

        break; // FIN TRANSICIÓN S1 A L1

    case State.S02L1: // Inhabilitado
```

```
            break;


        case State.S12L0: // Inhabilitado
            break;

    }

    // Calcula la distancia en X y Z, teniendo en cuenta los límites
    //float distanceX = Mathf.Abs(boxPos2.x - referenceBoxPos2.x) / (Mathf.
        Abs(-2 - -64) + 7); // El +7 es para ajustar por el tamaño de la
        bandeja
    //float distanceZ = Mathf.Abs(boxPos2.z - referenceBoxPos2.z) / (Mathf.
        Abs(17 - -39) + 7);
    // Define los límites del área de juego en las coordenadas X y Z
    float maxX = 64; // Asume que este es el límite máximo en X
    float maxZ = 39; // Asume que este es el límite máximo en Z
    float minX = -2; // Asume que este es el límite mínimo en X
    float minZ = -39; // Asume que este es el límite mínimo en Z

    // Calcula las dimensiones máximas del área de juego
    float maxWidth = maxX - minX;
    float maxHeight = maxZ - minZ;

    // Calcula la máxima distancia euclidiana posible dentro del área de
        juego
    float limite = Mathf.Sqrt(Mathf.Pow(maxWidth, 2) + Mathf.Pow(maxHeight,
         2));

    // Calcula la distancia euclidiana (norma del vector diferencia) entre
        boxPos2 y referenceBoxPos2
    float distanciaEuclidiana = (boxPos2 - referenceBoxPos2).magnitude;

    // Ajusta la empinadez de la curva de recompensa elevando la inversa de
         la distancia euclidiana a una potencia deseada
    float exponente = 0.5f; // Ajusta este valor para cambiar la empinadez
        de la curva de recompensa
    float reward = 10 * Mathf.Max(0, 1 - Mathf.Pow(distanciaEuclidiana /
        limite, exponente))-2.0f;
    AddReward(reward);

    //AddReward(combinedReward);
    Debug.Log($"Recompensa añadida por distancia: {reward}");

    // Imprimir la recompensa acumulativa hasta el momento
    //Debug.Log($"Recompensa acumulativa: {GetCumulativeReward()}");


    // FIN ZONA DE RECOMPENSAS
    UpdateDatos(speedAction, pistonPositionActionPiston1,
        pistonPositionActionPiston2, pistonPositionActionPiston3,
        pistonPositionActionPiston4);

    //RESETEO POR SI FALLA ALGO
    if (Input.GetKeyDown(KeyCode.R)) { isTimerActive = false; AddReward
        (-9900f); EndEpisode(); }
}
```

```
public override void Heuristic(in ActionBuffers actionsOut)
{
    var discreteActionsOut = actionsOut.DiscreteActions;
    // Pistón 1
    if (Input.GetKey(KeyCode.A)) { piston1.SetBeltDirection(1); Debug.Log("
        Pistón 1: cinta izquierda"); }
    if (Input.GetKey(KeyCode.D)) { piston1.SetBeltDirection(-1); Debug.Log(
        "Pistón 1: cinta derecha"); }
    if (Input.GetKey(KeyCode.W)) { piston1.SetPistonPosition(true); Debug.
        Log("Pistón 1: sube"); }
    if (Input.GetKey(KeyCode.S)) { piston1.SetPistonPosition(false); Debug.
        Log("Pistón 1: baja"); }

    // Pistón 2
    if (Input.GetKey(KeyCode.F)) { piston2.SetBeltDirection(-1); Debug.Log(
        "Pistón 2: cinta izquierda"); }
    if (Input.GetKey(KeyCode.H)) { piston2.SetBeltDirection(1); Debug.Log("
        Pistón 2: cinta derecha"); }
    if (Input.GetKey(KeyCode.T)) { piston2.SetPistonPosition(true); Debug.
        Log("Pistón 2: sube"); }
    if (Input.GetKey(KeyCode.G)) { piston2.SetPistonPosition(false); Debug.
        Log("Pistón 2: baja"); }

    // Pistón 3
    if (Input.GetKey(KeyCode.J)) { piston3.SetBeltDirection(-1); Debug.Log(
        "Pistón 3: cinta izquierda"); }
    if (Input.GetKey(KeyCode.L)) { piston3.SetBeltDirection(1); Debug.Log("
        Pistón 3: cinta derecha"); }
    if (Input.GetKey(KeyCode.I)) { piston3.SetPistonPosition(true); Debug.
        Log("Pistón 3: sube"); }
    if (Input.GetKey(KeyCode.K)) { piston3.SetPistonPosition(false); Debug.
        Log("Pistón 3: baja"); }

    // Pistón 4
    if (Input.GetKey(KeyCode.Z)) { piston4.SetBeltDirection(-1); Debug.Log(
        "Pistón 4: cinta izquierda"); }
    if (Input.GetKey(KeyCode.C)) { piston4.SetBeltDirection(1); Debug.Log("
        Pistón 4: cinta derecha"); }
    if (Input.GetKey(KeyCode.V)) { piston4.SetPistonPosition(true); Debug.
        Log("Pistón 4: sube"); }
    if (Input.GetKey(KeyCode.X)) { piston4.SetPistonPosition(false); Debug.
        Log("Pistón 4: baja"); }


    if(Input.GetKey(KeyCode.Alpha1)) { speedAction = 1; Debug.Log($"Speed
        Action set to {speedAction}"); }
    if (Input.GetKey(KeyCode.Alpha2)) { speedAction = 2; Debug.Log($"Speed
        Action set to {speedAction}"); }
    if (Input.GetKey(KeyCode.Alpha3)) { speedAction = 3; Debug.Log($"Speed
        Action set to {speedAction}"); }
    if (Input.GetKey(KeyCode.Alpha4)) { speedAction = 4; Debug.Log($"Speed
        Action set to {speedAction}"); }
    if (Input.GetKey(KeyCode.Alpha5)) { speedAction = 5; Debug.Log($"Speed
        Action set to {speedAction}"); }

    // Puedes agregar más controles para otros pistones aquí
```

```
        ShortBeltConveyorModel.Parameters.Speed = speedAction;
        LongBeltConveyorModel.Parameters.Speed = speedAction;

        if (Input.GetKeyDown(KeyCode.R)) EndEpisode();

    }
    public override void OnEpisodeBegin()
    {
        // Restablece la posición de la bandeja, la bandeja de referencia, y
            cualquier otro estado del entorno
        //ResetEnvironment();
        ChangeState(State.L1);
        box.transform.position = posicionInicialBox;
        box.transform.rotation = rotacionInicialBox;
        // Restablece la posición y rotación de los objetos a sus estados
            originales
        //arribaPiston2.transform.position = originalPositionArribaPiston2;
        //arribaPiston4.transform.position = originalPositionArribaPiston4;
        //arribaPiston2.transform.rotation = originalRotationArribaPiston2;
        //arribaPiston4.transform.rotation = originalRotationArribaPiston4;
        ReiniciarTriggers();

        foreach (var kvp in initialState)
        {
            kvp.Key.position = kvp.Value.Item1;
            kvp.Key.rotation = kvp.Value.Item2;
            // Restablecer otros estados físicos si es necesario, como Rigidbody
            var rb = kvp.Key.GetComponent<Rigidbody>();
            if (rb != null)
            {
                rb.velocity = Vector3.zero;
                rb.angularVelocity = Vector3.zero;
            }
        }

        if (rbBox != null) { rbBox.velocity = Vector3.zero; rbBox.
            angularVelocity = Vector3.zero;}

        // Asegúrate de que los pistones estén en la posición 'false' y en la
            dirección '1'
        piston1.SetPistonPosition(false); // Establece la posición del pistón 1
             a 'false'
        piston1.SetBeltDirection(1); // Establece la dirección del pistón 1 a
            '1'

        piston2.SetPistonPosition(false); // Establece la posición del pistón 2
             a 'false'
        piston2.SetBeltDirection(1); // Establece la dirección del pistón 2 a
            '1'

        piston3.SetPistonPosition(false); // Establece la posición del pistón 3
             a 'false'
        piston3.SetBeltDirection(1); // Establece la dirección del pistón 3 a
            '1'

        piston4.SetPistonPosition(false); // Establece la posición del pistón 4
             a 'false'
```

```
        piston4.SetBeltDirection(1); // Establece la dirección del pistón 4 a
            '1'
    }

    void FixedUpdate()
    {
        if (inductiveSensors != null)
        {
            int[] sensorEdgeResults = AnalyzeSensorEdges(inductiveSensors, ref
                previousSensorStates);
            HandleStateChanges(sensorEdgeResults);
        }
    }
    void Update()
    {
        if (box.transform.localPosition.y < -10)
        {
            isTimerActive = false;
            EndEpisode();
        }
    }


    private ConveyorLocation DetermineConveyorLocation(Vector3 position)
    {
        float longBeltZ1 = -39f;
        float longBeltZ0 = 17f;
        float shortBeltX0 = -2f;
        float shortBeltX1 = -64f;

        float zMargin = 1f;
        float xMargin = 2f;

        if (position.z > longBeltZ1 - zMargin && position.z < longBeltZ1 +
            zMargin) { return ConveyorLocation.LongBeltConveyor1; }
        else if (position.z > longBeltZ0 - zMargin && position.z < longBeltZ0 +
             zMargin) { return ConveyorLocation.LongBeltConveyor0; }
        else if (position.x > shortBeltX0 - xMargin && position.x < shortBeltX0
             + xMargin) { return ConveyorLocation.ShortBeltConveyor0; }
        else if (position.x > shortBeltX1 - xMargin && position.x < shortBeltX1
             + xMargin) { return ConveyorLocation.ShortBeltConveyor1; }

        return ConveyorLocation.OutOfConveyor;
    }

    private void HandleStateChanges(int[] sensorEdgeResults)
    {
        switch (currentState)
        {
            case State.L0:
                //Debug.Log("L0");
                if (sensorEdgeResults[9] == 1 /*&& speedAction > 0*/) {
                    ChangeState(State.L02S1); Debug.Log("Cambio de estado a
                    L02S1"); };
                // Aquí, implementa las acciones específicas para el estado L0
                break;
            case State.L1:
```

```
            //Debug.Log("L1");
            if (sensorEdgeResults[4] == 1 /*&& speedAction > 0*/ ) {
                ChangeState(State.L12S0); Debug.Log("Cambio de estado a
                L12S0"); }
            // Aquí, implementa las acciones específicas para el estado L1
            break;
        case State.S0:
            //Debug.Log("S0");
            if (sensorEdgeResults[6] == 1 /* && speedAction < 0*/) {
                ChangeState(State.S02L1); Debug.Log("Cambio de estado a
                S02L1"); }
            if (sensorEdgeResults[7] == 1 /*&& speedAction > 0*/) {
                ChangeState(State.S02L0); Debug.Log("Cambio de estado a
                S02L0"); }
            // Aquí, implementa las acciones específicas para el estado S0
            break;
        case State.S1:
            //Debug.Log("S1");
            if (sensorEdgeResults[1] == 1 /*&& speedAction < 0*/) {
                ChangeState(State.S12L0); Debug.Log("Cambio de estado a
                S12L0"); }
            if (sensorEdgeResults[2] == 1 /*&& speedAction > 0*/) {
                ChangeState(State.S12L1); Debug.Log("Cambio de estado a
                S12L1"); }
            break;

        case State.L02S1:
            // Debug.Log("L02S1");
            if (sensorEdgeResults[1] == 1 /*&& speedAction > 0*/) {
                ChangeState(State.S1); Debug.Log("Cambio de estado a S1"); }
                 //
            break;

        case State.S02L0:
            //Debug.Log("S02L0");
            if (sensorEdgeResults[8] == 1 /*&& speedAction > 0*/) {
                ChangeState(State.L0); Debug.Log("Cambio de estado a L0"); }
                 //
            break;

        case State.S02L1:
            //Debug.Log("S02L1");
            if (sensorEdgeResults[4] == 1 /*&& speedAction < 0*/) {
                ChangeState(State.L1); ; Debug.Log("Cambio de estado a L1");
                 } //
            break;

        case State.L12S0:
            //Debug.Log("L12S0");
            if (sensorEdgeResults[6] == 1 /*&& speedAction > 0*/) {
                ChangeState(State.S0); Debug.Log("Cambio de estado a S0"); }
                //
            break;

        case State.S12L0:
            //Debug.Log("S12L0");
```

```
            if (sensorEdgeResults[9] == 1 /*&& speedAction < 0*/) {
                ChangeState(State.L0); Debug.Log("Cambio de estado a L0"); }
                 //
            break;

        case State.S12L1:
            //Debug.Log("S12L1");
            if (sensorEdgeResults[3] == 1 /*&& speedAction > 0*/) {
                ChangeState(State.L1); Debug.Log("Cambio de estado a L1"); }
                 //
            break;

    }
}



//////////////////////////////////////FUNCIONES AUXILIARES
    ////////////////////////////////////////////////////////

public void UpdateDatos(float speedAction, bool pistonPositionActionPiston1
    , bool pistonPositionActionPiston2, bool pistonPositionActionPiston3,
    bool pistonPositionActionPiston4) // Función para actualizar los datos
    y almacenar los últimos
{
    // Desplaza los valores hacia adelante en el array para hacer espacio
        al valor más reciente en el índice 0
    for (int i = previousSpeedActions.Length - 1; i > 0; i--)
    {
        previousSpeedActions[i] = previousSpeedActions[i - 1];
    }

    // Inserta la velocidad actual en el índice 0
    previousSpeedActions[0] = speedAction;

    // Actualiza las posiciones anteriores de 'box'
    for (int i = previousBoxPositions.Length - 1; i > 0; i--)
    {
        previousBoxPositions[i] = previousBoxPositions[i - 1];
    }
    previousBoxPositions[0] = box.transform.localPosition;
    // Actualiza las posiciones anteriores de 'referenceBox'
    for (int i = previousReferenceBoxPositions.Length - 1; i > 0; i--)
    {
        previousReferenceBoxPositions[i] = previousReferenceBoxPositions[i -
             1];
    }
    previousReferenceBoxPositions[0] = referenceBox.transform.localPosition;

    // Actualizar los estados anteriores de los pistones directamente con
        los valores booleanos
    for (int i = previousPiston1Position.Length - 1; i > 0; i--)
    {
        previousPiston1Position[i] = previousPiston1Position[i - 1];
        previousPiston2Position[i] = previousPiston2Position[i - 1];
        previousPiston3Position[i] = previousPiston3Position[i - 1];
        previousPiston4Position[i] = previousPiston4Position[i - 1];
```

```
        }

        // Inserta los estados actuales de los pistones en el índice 0
        previousPiston1Position[0] = pistonPositionActionPiston1;
        previousPiston2Position[0] = pistonPositionActionPiston2;
        previousPiston3Position[0] = pistonPositionActionPiston3;
        previousPiston4Position[0] = pistonPositionActionPiston4;

        bool isStagnant = true; // Asumimos que está estancado hasta que se
            demuestre lo contrario
        float tolerance = 0.001f; // Tolerancia de diferencia en la posición
        // Comprobar si todas las posiciones en previousBoxPositions son
            iguales
        for (int i = 1; i < previousBoxPositions.Length; i++)
        {
            if (Mathf.Abs(previousBoxPositions[i].x - previousBoxPositions[0].x)
                 > tolerance ||
                Mathf.Abs(previousBoxPositions[i].y - previousBoxPositions[0].y)
                     > tolerance ||
                Mathf.Abs(previousBoxPositions[i].z - previousBoxPositions[0].z)
                     > tolerance)
            {
                isStagnant = false; // Se encontró una diferencia mayor que la
                    tolerancia, por lo tanto, no está estancado
                break;
            }
        }

        if (isStagnant)
        {
            AddReward(-9900f);
            isTimerActive = false;// Aplicar penalización
            EndEpisode(); // Reiniciar episodio
        }

    }



    void ChangeState(State newState)
        {
            currentState = newState;
        }

    public static int[] AnalyzeSensorEdges(InductiveSensor[] sensors, ref bool
        [] previousStates)
    {
        int[] edgeResults = new int[sensors.Length];

          for (int i = 0; i < sensors.Length; i++)
          {
             bool currentSensorState = sensors[i].IsMetalDetected();

              if (!previousStates[i] && currentSensorState) { edgeResults[i] =
                  1; }
                else if (previousStates[i] && !currentSensorState) { edgeResults
                    [i] = -1; }
```

```
                else { edgeResults[i] = 0; }

                previousStates[i] = currentSensorState;
            }

            return edgeResults;
        }
    void LogCumulativeReward()
    {
        Debug.Log($"Recompensa acumulativa: {GetCumulativeReward()}");
    }

    private int ContarCambios(bool[] estados)
    {
        int cambios = 0;
        for (int i = 1; i < estados.Length; i++)
        {
            if (estados[i] != estados[i - 1])
            {
                cambios++;
            }
        }
        return cambios;
    }

    private void ResetTriggersL1()
    {
        triggerpiston1 = false;
        triggerpiston2 = false;
        //triggerpiston3 = false;
        triggerpiston4 = false;
        triggertransicionpiston1 = false;
        triggertransicionpiston2 = false;
        triggertransicionpiston3 = false;
        triggertransicionpiston4 = false;
    }
    private void ResetTriggersL12S0()
    {
        triggerpiston1 = false;
        triggerpiston2 = false;
        //triggerpiston3 = false;
        triggerpiston4 = false;
        triggertransicionpiston1 = false;
        triggertransicionpiston2 = false;
        triggertransicionpiston3 = false;
        triggertransicionpiston4 = false;
    }
    private void ResetTriggersS0()
    {
        triggerpiston1 = false;
        triggerpiston2 = false;
        //triggerpiston3 = false;
        triggerpiston4 = false;
        triggertransicionpiston1 = false;
        triggertransicionpiston2 = false;
        triggertransicionpiston3 = false;
        triggertransicionpiston4 = false;
```

```
    }
    private void ResetTriggersS02L0()
    {
        triggerpiston1 = false;
        triggerpiston2 = false;
        //triggerpiston3 = false;
        triggerpiston4 = false;
        triggertransicionpiston1 = false;
        triggertransicionpiston2 = false;
        triggertransicionpiston3 = false;
        triggertransicionpiston4 = false;
    }
    private void ResetTriggersL0()
    {
        triggerpiston1 = false;
        triggerpiston2 = false;
        //triggerpiston3 = false;
        triggerpiston4 = false;
        triggertransicionpiston1 = false;
        triggertransicionpiston2 = false;
        triggertransicionpiston3 = false;
        triggertransicionpiston4 = false;
    }
    private void ResetTriggersL02S1()
    {
        triggerpiston1 = false;
        triggerpiston2 = false;
        //triggerpiston3 = false;
        triggerpiston4 = false;
        triggertransicionpiston1 = false;
        triggertransicionpiston2 = false;
        triggertransicionpiston3 = false;
        triggertransicionpiston4 = false;
    }
    private void ResetTriggersS1()
    {
        triggerpiston1 = false;
        triggerpiston2 = false;
        //triggerpiston3 = false;
        triggerpiston4 = false;
        triggertransicionpiston1 = false;
        triggertransicionpiston2 = false;
        triggertransicionpiston3 = false;
        triggertransicionpiston4 = false;
    }
    private void ResetTriggersS12L1()
    {
        triggerpiston1 = false;
        triggerpiston2 = false;
        triggerpiston3 = false;
        //triggerpiston4 = false;
        //triggertransicionpiston1 = false;
        //triggertransicionpiston2 = false;
        //triggertransicionpiston3 = false;
        //triggertransicionpiston4 = false;
    }
```

```
    void ReiniciarTriggers()
    {
        // Reinicia todos los triggers a false
        triggerpiston1 = false;
        triggerpiston2 = false;
        triggerpiston3 = false;
        triggerpiston4 = false;
        triggertransicionpiston1 = false;
        triggertransicionpiston2 = false;
        triggertransicionpiston3 = false;
        triggertransicionpiston4 = false;
    }

    private void AddToInitialStateDict(Transform transform)
    {
        // Añade el objeto actual al diccionario
        initialState[transform] = (transform.position, transform.rotation);

        // Recorre cada hijo y añádelo también
        foreach (Transform child in transform)
        {
            AddToInitialStateDict(child);
        }
    }

}
```

B.2 Método de entrenamiento en entorno simplificado

```
using System;
using System.Collections;
using System.Collections.Generic;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Actuators;
using UnityEngine;

public class ConveyorBeltAgentSimulacion : Agent
{
    public GameObject BandejaReferencia;
    public GameObject BandejaControlada;
    private float speedAction = 5f;

    private Vector3 posicionAnteriorBandejaControlada;
    private Vector3 posicionAnteriorBandejaReferencia;

    public MoverBandejaControlada moverBandejaControlada; // Referencia al
        script MoverBandejaControlada
    private float[] speedValues = {
            0.0f, 0.25f, 0.5f, 0.75f, 1.0f, 1.25f, 1.5f, 1.75f,
            2.0f, 2.25f, 2.5f, 2.75f, 3.0f, 3.25f, 3.5f, 3.75f,
            4.0f, 4.25f, 4.5f, 4.75f, 5.0f
        };
```

```
    public Vector3[] corners = new Vector3[]
    {
        new Vector3(-60.98f, 3f, 56.62f), // Inferior Izquierda
        new Vector3(-60.98f, 3f, 1f),  // Inferior Derecha
        new Vector3(2f, 3f, 1f),       // Superior Derecha
        new Vector3(2f, 3f, 56.62f)    // Superior Izquierda
    };

    private const float totalConveyorLength = 239.2f;

    private int decisionInterval = 10; // Intervalo de decisión en pasos
    private int stepCounter = 0; // Contador de pasos

    private float requiredTimeInSeconds;
    private float tolerance;
    private float timeWithinRange = 0f;

    public enum ConveyorLocation
    {
        LongBeltConveyor1,
        LongBeltConveyor0,
        ShortBeltConveyor0,
        ShortBeltConveyor1,
        OutOfConveyor
    }

    void Start()
    {
        InvokeRepeating("LogCumulativeReward", 2f, 2f);
        posicionAnteriorBandejaControlada = BandejaControlada.transform.
            position;
        posicionAnteriorBandejaReferencia = BandejaReferencia.transform.
            position;
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        Vector3 BandejaControladaPosicion = BandejaControlada.transform.
            position;
        Vector3 BandejaReferenciaPosicion = BandejaReferencia.transform.
            position;

        float distanciaCarriles = CalculateDistance(BandejaControladaPosicion,
            BandejaReferenciaPosicion);
        float distanciaCarriles_metros = distanciaCarriles / 22.045f;
        float distanciaCarriles_metros_normalizada = distanciaCarriles / (239.2
            f*0.5f);
        sensor.AddObservation(distanciaCarriles);

        // Calcular velocidades manualmente
        Vector3 desplazamientoBandejaControlada = BandejaControladaPosicion -
            posicionAnteriorBandejaControlada;
        Vector3 desplazamientoBandejaReferencia = BandejaReferenciaPosicion -
            posicionAnteriorBandejaReferencia;
        float velocidadBandejaControlada = desplazamientoBandejaControlada.
            magnitude;
```

```
        float velocidadBandejaReferencia = desplazamientoBandejaReferencia.
            magnitude;

        // Añadir las magnitudes de las velocidades como observaciones
        sensor.AddObservation(velocidadBandejaControlada);
        sensor.AddObservation(velocidadBandejaReferencia);
        sensor.AddObservation(BandejaControladaPosicion.x);
        sensor.AddObservation(BandejaControladaPosicion.z);
        sensor.AddObservation(BandejaReferenciaPosicion.x);
        sensor.AddObservation(BandejaReferenciaPosicion.z);

        // Mostrar las velocidades en la consola
        //Debug.Log($"Velocidad de BandejaControlada: {
            velocidadBandejaControlada}");
        //Debug.Log($"Velocidad de BandejaReferencia: {
            velocidadBandejaReferencia}");
        // Mostrar la distancia en la consola
        //Debug.Log($"Distancia Carriles (metros): {distanciaCarriles_metros}");


        // Actualizar posiciones previas
        posicionAnteriorBandejaControlada = BandejaControladaPosicion;
        posicionAnteriorBandejaReferencia = BandejaReferenciaPosicion;

        // Actualizar los parámetros del currículo
        //requiredTimeInSeconds = Academy.Instance.EnvironmentParameters.
            GetWithDefault("required_time_in_seconds", 0.1f);
        //tolerance = Academy.Instance.EnvironmentParameters.GetWithDefault("
            distance_tolerance", 0.05f);

        //Debug.Log($"Parametros del curriculo - Tiempo requerido: {
            requiredTimeInSeconds}, Tolerancia: {tolerance}");
    }

    public override void OnActionReceived(ActionBuffers actionBuffers)
    {
        int discreteSpeedAction = actionBuffers.DiscreteActions[0];
        //Debug.Log($" SOY DISCRETESPEEDACTION:" + discreteSpeedAction);

        // Mapear la acción discreta al rango de velocidad continua de 0 a 11
        speedAction = speedValues[discreteSpeedAction];
        //Debug.Log($" SOY SPEEDACTION:" + speedAction);

        Vector3 BandejaControladaPosicion = BandejaControlada.transform.
            localPosition;
        Vector3 BandejaReferenciaPosicion = BandejaReferencia.transform.
            localPosition;

        float distanciaCarriles = CalculateDistance(BandejaControladaPosicion,
            BandejaReferenciaPosicion);
        float distanciaCarriles_metros = distanciaCarriles / 22.045f;

        // Calcular la recompensa basada en la distancia en los carriles
        float reward = distanciaCarriles_metros >= 0 ? 2 -
            distanciaCarriles_metros : 2 + distanciaCarriles_metros;
        //Debug.Log($"DISTANCIA CARRILES: {distanciaCarriles_metros}");
        AddReward(reward);
```

```
        // Actualizar la speedAction en MoverBandejaControlada
        moverBandejaControlada.SetSpeedAction(speedAction);
        //Debug.Log($"Valor de SPEEDACTION:" + speedAction);

         //Lógica de recompensa y finalización del episodio basado en el currí
             culo
        //if (Mathf.Abs(distanciaCarriles_metros) <= tolerance)
        //{
        //    timeWithinRange += Time.deltaTime*decisionInterval;
        //    Debug.Log($"TIEMPO SEGUIMIENTO BANDEJAS:" + timeWithinRange);

        //    AddReward(2f); // Pequeña recompensa por estar dentro de la
            distancia objetivo
        //}
        //else
        //{
        //    timeWithinRange = 0f;
        //    AddReward(-0.1f); // Penalización por estar fuera de la distancia
            objetivo
        //}

        //if (timeWithinRange >= requiredTimeInSeconds)
        //{
        //    AddReward(1500.0f);
        //    Debug.Log($"Recompensa acumulativa antes de finalizar el episodio:
             {GetCumulativeReward()}");
        //    EndEpisode();
        //}
    }

    public override void OnEpisodeBegin()
    {
        // Mostrar el número de episodios completados
        Debug.Log("Episodios completados: " + CompletedEpisodes);

        Vector3 randomPosition = corners[UnityEngine.Random.Range(0, corners.
            Length)];
        moverBandejaControlada.SetRandomPosition(randomPosition);

        // Actualizar posiciones previas
        posicionAnteriorBandejaControlada = BandejaControlada.transform.
            localPosition;
        posicionAnteriorBandejaReferencia = BandejaReferencia.transform.
            localPosition;

        //// Actualizar los parámetros del currículo
        //requiredTimeInSeconds = Academy.Instance.EnvironmentParameters.
            GetWithDefault("required_time_in_seconds", requiredTimeInSeconds);
        //tolerance = Academy.Instance.EnvironmentParameters.GetWithDefault("
            distance_tolerance", tolerance);
        //Debug.Log($"Parametros del curriculo - Tiempo ambas juntas en
            segundos: {requiredTimeInSeconds}, Distancia entre bandejas en
            metros: {tolerance}");

    }
```

```
    public override void Heuristic(in ActionBuffers actionsOut)
    {
        var discreteActionsOut = actionsOut.DiscreteActions;
        int selectedSpeedIndex = 0;

        Vector3 BandejaControladaPosicion = BandejaControlada.transform.
            localPosition;
        Vector3 BandejaReferenciaPosicion = BandejaReferencia.transform.
            localPosition;
        float distanciaCarriles = CalculateDistance(BandejaControladaPosicion,
            BandejaReferenciaPosicion);
        float distanciaCarriles_metros = distanciaCarriles / 22.045f;

        if (distanciaCarriles_metros < 0 && distanciaCarriles_metros>-0.5)
        {
            speedAction = 1f;
        }
        else if (distanciaCarriles_metros <= -0.5 && distanciaCarriles_metros
            >-1)
        {
            speedAction = 0f;
        }
        else if (distanciaCarriles_metros < -1)
        {
            speedAction = 0f;
        }
        else if (distanciaCarriles_metros >= 0 && distanciaCarriles_metros <
            0.1f)
        {
            speedAction = 1.5f;
        }
        else if (distanciaCarriles >= 0.1f && distanciaCarriles_metros < 0.3f)
        {
            speedAction = 2f;
        }
        else if (distanciaCarriles_metros >= 0.3f && distanciaCarriles_metros <
             0.5f)
        {
            speedAction = 3f;
        }
        else if (distanciaCarriles_metros >= 0.5f && distanciaCarriles_metros <
             0.8f)
        {
            speedAction = 3.5f;
        }
        else if (distanciaCarriles_metros >= 0.8f)
        {
            speedAction = 5f;
        }

        selectedSpeedIndex = Array.IndexOf(speedValues, speedAction);
        if (selectedSpeedIndex == -1) selectedSpeedIndex = 0; // Fallback to a
            default value if not found

        discreteActionsOut[0] = selectedSpeedIndex;

        if (Input.GetKeyDown(KeyCode.R))
```

```
        {
            Debug.Log($"Recompensa acumulativa antes de finalizar el episodio: {
                GetCumulativeReward()}");
            EndEpisode();
        }
    }

    void FixedUpdate()
    {
        // Verificar si el episodio ha alcanzado el número máximo de pasos
        if (StepCount >= MaxStep - 1)
        {
            Debug.Log($"Recompensa acumulativa al alcanzar MaxStep: {
                GetCumulativeReward()}");
        }

        stepCounter++;
        if (stepCounter >= decisionInterval)
        {
            RequestDecision();
            stepCounter = 0;
        }
    }

    private ConveyorLocation DetermineConveyorLocation(Vector3 position)
    {
        float longBeltZ1 = 1f;
        float longBeltZ0 = 56f;
        float shortBeltX0 = 2f;
        float shortBeltX1 = -60f;

        float zMargin = 1f;
        float xMargin = 2f;

        if (position.z > longBeltZ1 - zMargin && position.z < longBeltZ1 +
            zMargin) { return ConveyorLocation.LongBeltConveyor1; }
        else if (position.z > longBeltZ0 - zMargin && position.z < longBeltZ0 +
             zMargin) { return ConveyorLocation.LongBeltConveyor0; }
        else if (position.x > shortBeltX0 - xMargin && position.x < shortBeltX0
             + xMargin) { return ConveyorLocation.ShortBeltConveyor0; }
        else if (position.x > shortBeltX1 - xMargin && position.x < shortBeltX1
             + xMargin) { return ConveyorLocation.ShortBeltConveyor1; }

        return ConveyorLocation.OutOfConveyor;
    }

    public float CalculateDistance(Vector3 start, Vector3 end)
    {
        ConveyorLocation startLocation = DetermineConveyorLocation(start);
        ConveyorLocation endLocation = DetermineConveyorLocation(end);

        if (startLocation == endLocation)
        {
            return CalculateSameConveyorDistance(start, end, startLocation,
                totalConveyorLength);
        }
```

```
        return CalculateDifferentConveyorDistance(start, end, startLocation,
            endLocation, totalConveyorLength);
    }

    private float CalculateSameConveyorDistance(Vector3 start, Vector3 end,
        ConveyorLocation location, float totalConveyorLength)
    {
        float directDistance = Vector3.Distance(start, end);
        float indirectDistance = totalConveyorLength - directDistance;

        bool isDirect = false;

        switch (location)
        {
            case ConveyorLocation.LongBeltConveyor1:
                isDirect = start.x <= end.x;
                break;
            case ConveyorLocation.ShortBeltConveyor0:
                isDirect = start.z <= end.z;
                break;
            case ConveyorLocation.LongBeltConveyor0:
                isDirect = start.x >= end.x;
                break;
            case ConveyorLocation.ShortBeltConveyor1:
                isDirect = start.z >= end.z;
                break;
            default:
                throw new ArgumentException("Invalid conveyor location");
        }

        if (isDirect)
        {
            return Mathf.Abs(directDistance) < Mathf.Abs(indirectDistance) ?
                directDistance : indirectDistance;
        }
        else
        {
            float negativeDistance = -directDistance;
            return Mathf.Abs(negativeDistance) < Mathf.Abs(indirectDistance) ?
                negativeDistance : indirectDistance;
        }
    }

    private float CalculateDifferentConveyorDistance(Vector3 start, Vector3 end
        , ConveyorLocation startLocation, ConveyorLocation endLocation, float
        totalConveyorLength)
    {
        float directDistance = 0f;

        float distanceToNextCornerStart = DistanceToNextCorner(start,
            GetNextCornerIndex(startLocation));
        float distanceFromPreviousCornerEnd = DistanceFromPreviousCorner(end,
            GetPreviousCornerIndex(endLocation));

        directDistance += distanceToNextCornerStart;
        directDistance += distanceFromPreviousCornerEnd;
```

```
        int startCornerIndex = GetNextCornerIndex(startLocation);
        int endCornerIndex = GetPreviousCornerIndex(endLocation);

        while (startCornerIndex != endCornerIndex)
        {
            int nextCornerIndex = (startCornerIndex + 1) % 4;
            directDistance += Vector3.Distance(corners[startCornerIndex],
                corners[nextCornerIndex]);
            startCornerIndex = nextCornerIndex;
        }

        float indirectDistance = totalConveyorLength - directDistance;
        indirectDistance = -indirectDistance; // Hacer que la distancia
            indirecta sea negativa

        return Mathf.Abs(directDistance) < Mathf.Abs(indirectDistance) ?
            directDistance : indirectDistance;
    }


    private int GetNextCornerIndex(ConveyorLocation location)
    {
        switch (location)
        {
            case ConveyorLocation.LongBeltConveyor1:
                return 2;
            case ConveyorLocation.ShortBeltConveyor0:
                return 3;
            case ConveyorLocation.LongBeltConveyor0:
                return 0;
            case ConveyorLocation.ShortBeltConveyor1:
                return 1;
            default:
                EndEpisode();
                throw new ArgumentException("Invalid conveyor location");
        }
    }

    private int GetPreviousCornerIndex(ConveyorLocation location)
    {
        switch (location)
        {
            case ConveyorLocation.LongBeltConveyor1:
                return 1;
            case ConveyorLocation.ShortBeltConveyor0:
                return 2;
            case ConveyorLocation.LongBeltConveyor0:
                return 3;
            case ConveyorLocation.ShortBeltConveyor1:
                return 0;
            default:
                EndEpisode();
                throw new ArgumentException("Invalid conveyor location");
        }
    }

    private float DistanceToNextCorner(Vector3 point, int cornerIndex)
```

```
    {
        return Vector3.Distance(point, corners[cornerIndex]);
    }

    private float DistanceFromPreviousCorner(Vector3 point, int cornerIndex)
    {
        return Vector3.Distance(point, corners[cornerIndex]);
    }

    void LogCumulativeReward()
    {
        //Debug.Log($"Recompensa acumulativa: {GetCumulativeReward()}");
    }

    void Update()
    {
        // Debug or visualize the current position
        //Debug.Log("Posición actual: " + BandejaControlada.transform.
            localPosition);
    }
}
```

B.3 Código para mover la bandeja de referencia

```
using System.Collections;
using UnityEngine;

public class MoverBandejaReferencia : MonoBehaviour
{
    private Vector3 OrigenReferencia = new Vector3(-67.28f, 3f, 56.62f);

    public Vector3[] OrigenEsquinas = new Vector3[]
    {
        new Vector3(-60.98f, 3f, 56.62f), // Inferior Izquierda
        new Vector3(-60.98f, 3f, 1f),  // Inferior Derecha
        new Vector3(2f, 3f, 1f),       // Superior Derecha
        new Vector3(2f, 3f, 56.62f)    // Superior Izquierda
    };

    public Vector3[] RealEsquinas = new Vector3[]
    {
        new Vector3(0.01f, 0f, 0.34f), // Inferior Izquierda
        new Vector3(2.22f, 0f, 0.34f), // Inferior Derecha
        new Vector3(2.22f, 0f, 3.33f), // Superior Derecha
        new Vector3(0.0f, 0f, 3.33f)   // Superior Izquierda
    };

    private float escalaY = 20.867f; // Calculado previamente
    private float escalaX = -25.113f; // Calculado previamente

    private int currentCornerIndex = 0;
    private Vector3 ejemploPosicionReal;

    void Start()
```

```
    {
        ejemploPosicionReal = RealEsquinas[0];
        StartCoroutine(MoveAlongSquarePath());
    }

    IEnumerator MoveAlongSquarePath()
    {
        while (true)
        {
            Vector3 targetPosition = RealEsquinas[(currentCornerIndex + 1) %
                RealEsquinas.Length];
            while (ejemploPosicionReal != targetPosition)
            {
                ejemploPosicionReal = Vector3.MoveTowards(ejemploPosicionReal,
                    targetPosition, 0.1f);
                PosicionarEnUnity(ejemploPosicionReal);
                yield return new WaitForSeconds(1f);
            }

            currentCornerIndex = (currentCornerIndex + 1) % RealEsquinas.Length;
        }
    }

    // Método para posicionar el objeto en Unity basado en las coordenadas del
        mundo real
    public void PosicionarEnUnity(Vector3 posicionReal)
    {
        // Aplicar la transformación
        Vector3 posicionUnity = TransformarCoordenadasMundoRealAUnity(
            posicionReal);

        // Ajustar la posición al nuevo origen
        transform.position = OrigenReferencia + posicionUnity;
    }

    // Transforma las coordenadas del mundo real a las coordenadas de Unity
    private Vector3 TransformarCoordenadasMundoRealAUnity(Vector3 posicionReal)
    {
        // Convertir las coordenadas reales a las de Unity
        return new Vector3(posicionReal.z * escalaY, 0, posicionReal.x *
            escalaX);
    }

    void Update()
    {
        // Debug o visualización de la posición actual
        //Debug.Log("Posición actual en el mundo real: " + ejemploPosicionReal);

    }
}
```

B.4 Código para mover la bandeja de control

```
using UnityEngine;
```

```
public class MoverBandejaControlada : MonoBehaviour
{
    private Vector3 posicionInicialBandejaControlada = new Vector3(-60.98f, 3f,
        20f);

    public Vector3[] OrigenEsquinas = new Vector3[]
    {
        new Vector3(-60.98f, 3f, 56.62f), // Inferior Izquierda
        new Vector3(-60.98f, 3f, 1f),  // Inferior Derecha
        new Vector3(2f, 3f, 1f),       // Superior Derecha
        new Vector3(2f, 3f, 56.62f)    // Superior Izquierda
    };

    private enum ConveyorLocation
    {
        LongBeltConveyor1,
        LongBeltConveyor0,
        ShortBeltConveyor0,
        ShortBeltConveyor1,
        OutOfConveyor
    }

    private ConveyorLocation currentConveyor;
    private bool firstRun = true;
    public float speedAction = 1.0f;
    private float tolerance = 0.1f;

    void Start()
    {
        if (firstRun)
        {
            transform.position = posicionInicialBandejaControlada;
            firstRun = false;
            currentConveyor = DetermineInitialConveyor(transform.position);
        }
    }

    void Update()
    {
        MoveToNextCorner();
    }

    public void SetRandomPosition(Vector3 newPosition)
    {
        transform.position = newPosition;
        currentConveyor = DetermineInitialConveyor(newPosition);
    }

    public void SetSpeedAction(float newSpeed)
    {
        speedAction = newSpeed;
        //Debug.Log($"SpeedAction updated to: {speedAction}");
    }

    private ConveyorLocation DetermineInitialConveyor(Vector3 position)
    {
```

```
        float x = position.x;
        float z = position.z;

        // Verificar si está dentro del rango de ShortBeltConveyor1
        if (IsWithinRange(x, OrigenEsquinas[0].x) && z <= OrigenEsquinas[0].z
            && z >= OrigenEsquinas[1].z)
        {
            //Debug.Log($"Position {position} is within ShortBeltConveyor1 range
                .");
            return ConveyorLocation.ShortBeltConveyor1;
        }
        // Verificar si está dentro del rango de LongBeltConveyor1
        else if (IsWithinRange(x, OrigenEsquinas[1].x) && z >= OrigenEsquinas
            [1].z && z <= OrigenEsquinas[2].z)
        {
            //Debug.Log($"Position {position} is within LongBeltConveyor1 range
                .");
            return ConveyorLocation.LongBeltConveyor1;
        }
        // Verificar si está dentro del rango de ShortBeltConveyor0
        else if (IsWithinRange(x, OrigenEsquinas[2].x) && z >= OrigenEsquinas
            [2].z && z <= OrigenEsquinas[3].z)
        {
            //Debug.Log($"Position {position} is within ShortBeltConveyor0 range
                .");
            return ConveyorLocation.ShortBeltConveyor0;
        }
        // Verificar si está dentro del rango de LongBeltConveyor0
        else if (IsWithinRange(x, OrigenEsquinas[3].x) && z >= OrigenEsquinas
            [3].z && z <= OrigenEsquinas[0].z)
        {
            //Debug.Log($"Position {position} is within LongBeltConveyor0 range
                .");
            return ConveyorLocation.LongBeltConveyor0;
        }
        else
        {
            //Debug.Log($"Position {position} is OutOfConveyor.");
            return ConveyorLocation.OutOfConveyor;
        }
    }

    private bool IsWithinRange(float value, float center)
    {
        return Mathf.Abs(value - center) <= 1f;
    }

    private bool IsAtPosition(Vector3 currentPosition, Vector3 targetPosition)
    {
        return Vector3.Distance(currentPosition, targetPosition) <= tolerance;
    }

    private void MoveToNextCorner()
    {
        Vector3 targetPosition = Vector3.zero;

        switch (currentConveyor)
```

```
        {
            case ConveyorLocation.ShortBeltConveyor1:
                targetPosition = OrigenEsquinas[1];
                break;
            case ConveyorLocation.LongBeltConveyor1:
                targetPosition = OrigenEsquinas[2];
                break;
            case ConveyorLocation.ShortBeltConveyor0:
                targetPosition = OrigenEsquinas[3];
                break;
            case ConveyorLocation.LongBeltConveyor0:
                targetPosition = OrigenEsquinas[0];
                break;
            case ConveyorLocation.OutOfConveyor:
                //Debug.Log("OutOfConveyor");
                return; // No hacer nada si está fuera de la cinta
        }

        // Multiplicar por Time.timeScale
        transform.position = Vector3.MoveTowards(transform.position,
            targetPosition, speedAction * Time.timeScale * 0.03f);

        if (IsAtPosition(transform.position, targetPosition))
        {
            //Debug.Log($"Reached target position: {targetPosition}");
            UpdateConveyorState();
        }
    }

    private void UpdateConveyorState()
    {
        switch (currentConveyor)
        {
            case ConveyorLocation.ShortBeltConveyor1:
                currentConveyor = ConveyorLocation.LongBeltConveyor1;
                break;
            case ConveyorLocation.LongBeltConveyor1:
                currentConveyor = ConveyorLocation.ShortBeltConveyor0;
                break;
            case ConveyorLocation.ShortBeltConveyor0:
                currentConveyor = ConveyorLocation.LongBeltConveyor0;
                break;
            case ConveyorLocation.LongBeltConveyor0:
                currentConveyor = ConveyorLocation.ShortBeltConveyor1;
                break;
            case ConveyorLocation.OutOfConveyor:
                // No hacer nada
                break;
        }

        //Debug.Log($"Updated Conveyor State: {currentConveyor}");
    }
}
```

B.5 Hiperparámetros para entrenamiento

```
behaviors:
  ConveyorBeltAgentSimulacion:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 20480
      learning_rate: 1.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 3
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
      gail:
        strength: 1.0
        gamma: 0.99
        demo_path: "Z:/2TFM/Simple_carriles/Simplificacion_carriles/Assets/
            Demonstrations/DemoSim7VARv1.demo" # Ruta al archivo de demostración
        learning_rate: 0.0003
        use_actions: true
        use_vail: true
    behavioral_cloning:
      strength: 1.0
      demo_path: "Z:/2TFM/Simple_carriles/Simplificacion_carriles/Assets/
          Demonstrations/DemoSim7VARv1.demo" # Ruta al archivo de demostración
      steps: 0
      batch_size: 1024
      num_epoch: 3
      samples_per_update: 0
    max_steps: 500000
    time_horizon: 128
    summary_freq: 200
    checkpoint_interval: 50000 # Guardar un punto de control cada 50000 pasos
    threaded: false

environment_parameters:
  required_time_in_seconds:
    curriculum:
      - name: Lesson0
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 0.5
        value: 5
```

```
      - name: Lesson1
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 1.0
        value: 10
      - name: Lesson2
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 2.0
        value: 15
      - name: Lesson3
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 4.0
        value: 20
      - name: Lesson4
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 8.0
        value: 40
      - name: Lesson5
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 16.0
        value: 80
  distance_tolerance:
    curriculum:
      - name: Lesson0
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 0.5
        value: 0.7
      - name: Lesson1
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 1.0
```

```
        value: 0.5
      - name: Lesson2
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 2.0
        value: 0.4
      - name: Lesson3
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 4.0
        value: 0.3
      - name: Lesson4
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 8.0
        value: 0.3
      - name: Lesson5
        completion_criteria:
          measure: reward
          behavior: ConveyorBeltAgentSimulacion
          signal_smoothing: true
          min_lesson_length: 30
          threshold: 16.0
        value: 0.25
```

Índice de Figuras

Índice de Códigos

Bibliografía

[1] Wikipedia contributors, "Modbus," https://en.wikipedia.org/wiki/Modbus, Wikipedia, The Free Encyclopedia, 2023, Última edición: 2023.

[2] B. . Ohms, "¿qué es un plc?" 5 2021, accesible 17/06/2024. [Online]. Available: https://blog.330ohms.com/2021/05/14/que-es-un-plc/

[3] CERN, "Cern en twitter," consultado: 17 junio, 2024. [Online]. Available: https://x.com/CERN/status/1722964403006386420

[4] Rolls-Royce, "How digital twin technology can enhance aviation," 7 2019, accesible 17/06/2024. [Online]. Available: https://www.rolls-royce.com/media/our-stories/discover/2019/how-digital-twin-technology-can-enhance-aviation.aspx

[5] M. Nanda, "Digital twin example in automotive - volvo," Toobler, Jul 2023, accesible 2023-11-12. [Online]. Available: https://www.toobler.com/blog/9-examples-of-digital-twin-technologies-for-industries/

[6] J. S. Rinaldi, *Industrial Ethernet*. Research Triangle Park, NC: ISA, 2005.

[7] M. Grieves, *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*. Florida Institute of Technology: White Paper, 2014.

[8] M. Walker, C. Bissell, and J. Monk, "The plc: a logical development," *Measurement + Control*, 2010.

[9] C. Miskinis, "The mysterious history of digital twin technology and who created it," https://www.challenge.org/insights/digital-twin-history/, March 2019.

[10] R. Saracco. (2022, May) Digital twins: Evolution in manufacturing. IEEE Digital Reality. [Online]. Available: https://digitalreality.ieee.org/images/files/pdf/2022may-ebook-digitaltwins-manufacturing2.pdf

[11] Unity, "Reimagine the automotive lifecycle with real-time 3d | unity," 5 2021, accesible 17/06/2024. [Online]. Available: https://www.youtube.com/watch?v=mj7TYBisWB8

[12] "ISO 23247-1:2021 Automation systems and integration – Digital twin framework for manufacturing," International Organization for Standardization, 10 2021, first edition. [Online]. Available: https://www.iso.org/standard/75053.html

[13] Rolls-Royce plc, "Rolls-royce and dsta collaborate on digital technology," Jul 2019, accesible 2023-11-12. [Online]. Available: https://www.rolls-royce.com/media/press-releases/2019/23-07-2019-rr-and-dsta-collaborate-on-digital-technology.aspx

[14] U. Technologies. (2022) What is a digital twin? building a smarter, safer, and more sustainable reality. [Online]. Available: https://resources.unity.com/automotive-transportation-manufacturing-content/what-is-a-digital-twin