Universidad de Cádiz
Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL

SENSOR DE VISIÓN ARTIFICIAL PARA LA MEJORA DE LA AUTOMATIZACIÓN DE PROCESOS INDUSTRIALES EN EL SIMULADOR FACTORY IO

AUTOR: ALBERTO DOMÍNGUEZ LORCA

Cádiz, octubre 2024

Universidad de Cádiz
Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL

SENSOR DE VISIÓN ARTIFICIAL PARA LA MEJORA DE LA AUTOMATIZACIÓN DE PROCESOS INDUSTRIALES EN EL SIMULADOR FACTORY IO

DIRECTOR: DANIEL SÁNCHEZ MORILLO
AUTOR: ALBERTO DOMÍNGUEZ LORCA

Cádiz, octubre 2024

Agradecimientos

En primer lugar, quisiera agradecer al director del Trabajo Fin de Grado, el profesor Daniel Sánchez Morillo. La asesoría, paciencia y apoyo permanentemente ofrecido por él fueron imprescindibles para terminar con éxito el presente trabajo. Tanto sus conocimientos como sus orientaciones no solamente me ayudaron a resolver las dificultades técnicas, sino que además me hicieron un mejor profesional y y alcanzar un mayor rigor científico.

A mis profesores de la Universidad de Cádiz, gracias por haber compartido su conocimiento a lo largo de estos años. Su enseñanza ha sido una parte importante de mi formación y les agradezco por ello mi evolución como ingeniero y persona.

Siempre agradeceré a mis amigos que me han acompañado en este viaje, por su compañía, las risas compartidas, los momentos de desahogo y por siempre estar ahí.

Finalmente, a mi familia, y en especial a mi madre y mi padre, quiero dedicarles un agradecimiento profundo. Su amor, comprensión y apoyo incondicional han sido una fuente constante de motivación. Sin ustedes, nada de esto habría sido posible.

En estos momentos tan importantes es difícil no tener en uno mismo la mezcla de emociones; orgullo por lo que se ha logrado, nostalgia por lo que se dejó y expectativa por lo que vendrá. Pero lo más importante es que siento un gran agradecimiento hacia todos. Este logro es el resultado del esfuerzo conjunto, el apoyo recibido y la confianza depositada en mí.

Resumen

Este proyecto se centrará en la integración de un sensor de visión artificial en el simulador de procesos industriales Factory IO ® (Real Games). El fin último es el de proporcionar una alternativa a los sensores convencionales de proximidad (inductivos, capacitivos u ópticos) y facilitar un nuevo elemento didáctico que pueda ser empleado en las sesiones prácticas de las asignaturas relacionadas con esta materia. El sensor utilizará técnicas de aprendizaje profundo (DL) para detectar y clasificar objetos.

Este trabajo fin de grado contempla la automatización de un proceso industrial concreto, orientad a la clasificación de paquetes. Se afronta la automatización del proceso empleando un autómata programable y siguiendo las directrices software del estándar IEC-61131-3. En una primera aproximación, se resuelve el escenario empleando sensores de proximidad convencionales. Posteriormente, se procede al entrenamiento de un modelo de inteligencia artificial basado en Deep Learning, empleando imágenes sintéticas extraídas del simulador. El modelo será capaz de diferenciar características de los objetos a clasificar en el sistema. Con ello, se modelará un sensor virtual, que se comunicará con el simulador hiperrealista empleando servicios web. La imagen en tiempo real será adquirida por el sensor a través del software Open Broadcaster Software (OBS). Finalmente, se discutirán y compararán los resultados de ambas aproximaciones.

Resume

This project will focus on the integration of a computer vision sensor into the industrial process simulator Factory IO ® (Real Games). The ultimate goal is to provide an alternative to conventional proximity sensors (inductive, capacitive, or optical) and to introduce a new didactic element that can be used in practical sessions for subjects related to this field. The sensor will use deep learning (DL) techniques to detect and classify objects.

This final degree project involves the automation of a specific industrial process aimed at package classification. The process automation is approached using a programmable logic controller (PLC) and following the software guidelines of the IEC-61131-3 standard. In the first stage, the scenario is solved using conventional proximity sensors. Subsequently, an artificial intelligence model based on Deep Learning is trained using synthetic images extracted from the simulator. The model will be able to differentiate characteristics of the objects to be classified within the system. A virtual sensor will be modeled, which will communicate with the hyper-realistic simulator using web services. The real-time image will be acquired by the sensor through the Open Broadcaster Software (OBS). Finally, the results of both approaches will be discussed and compared.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Desde su comienzo, la automatización industrial ha procurado mejorar los procesos de producción mediante el empleo de tecnologías que posibiliten la realización autónoma y eficaz de tareas. Además, la inteligencia artificial ha ampliado enormemente las fronteras de lo que es posible en términos de automatización y toma de decisiones al simular la inteligencia humana mediante algoritmos y sistemas informáticos.

La combinación de la automatización industrial y la inteligencia artificial ha creado un nuevo concepto en el campo de la manufactura, donde los sistemas de producción pueden no solo llevar a cabo tareas repetitivas y programadas, sino también aprender, adaptarse y tomar decisiones por sí mismos en tiempo real. La creación de sistemas mucho más inteligentes ha sido impulsada por esta asociación entre la automatización y la inteligencia artificial, lo que los capacita para detectar las necesidades del mercado, optimizar los recursos y garantizar la calidad del producto final.

En este proyecto, se examinará la intersección entre la automatización industrial y la inteligencia artificial, observando cómo estas dos disciplinas juntas pueden dar lugar a resultados sorprendentes, como se revelará más adelante.

Asimismo, en este proyecto se utilizarán herramientas de simulación, sensores, actuadores y software de programación para promover el entendimiento y funcionamiento de la automatización industrial y la inteligencia artificial. Esto permitirá generar e implementar un sensor novedoso basado en inteligencia artificial, específicamente centrado en visión artificial.

Este tipo de sensores basados en visión artificial pueden detectar, reconocer y rastrear objetos con precisión y rapidez, lo que los hace muy útiles a la hora de mejorar una amplia gama de aplicaciones industriales. Este sensor no solo ampliará las capacidades de detección y reconocimiento de objetos en la simulación, sino que también proporcionará una plataforma para explorar y desarrollar soluciones más avanzadas en el campo de la automatización industrial.

Capítulo 2

Automatización Industrial

Anteriormente se ha hablado sobre la automatización industrial.Pero ¿qué es en realidad la automatización industrial?

Figura 2.1: Automatización Industrial.Fuente: [1]

Durante el siglo pasado, la automatización industrial ha desempeñado un papel fundamental en la evolución de la fabricación y producción, y sigue siendo crucial en la revolución industrial actual. La automatización industrial ha cambiado radicalmente la forma en que se diseñan, fabrican y entregan productos en todo el mundo, desde que Henry Ford introdujo la automatización en la línea de ensamblaje de automóviles hasta la actualidad con la Industria 4.0.

La automatización industrial, en su núcleo, se trata del uso de tecnologías avanzadas para llevar a cabo tareas y procesos de forma autónoma o semiautónoma en entornos industriales, con el fin de mejorar la eficiencia, la productividad y la calidad. Este avance ha sido resultado de una combinación de progresos.

La automatización industrial tiene beneficios extensos y profundos. Ha posibilitado una mayor flexibilidad en la fabricación, lo que permite producir de manera personalizada y bajo demanda, además de mejorar la eficiencia y la productividad. También, al eliminar tareas peligrosas y repetitivas, ha mejorado considerablemente las condiciones laborales y ha creado nuevas oportunidades de innovación y crecimiento económico.

No obstante, la automatización industrial también conlleva desafíos importantes como la reorganización del mercado laboral, la demanda de nuevas habilidades y formación, además de inquietudes sobre la ciberseguridad y privacidad de datos. Es crucial abordar proactivamente y de manera equilibrada los desafíos que plantea la automatización a medida que avanza, para asegurar una distribución justa y equitativa de los beneficios de esta revolución tecnológica en toda la sociedad..

En pocas palabras, la automatización industrial es un fenómeno complejo que ha cambiado significativamente la forma en que se produce y fabrica a lo largo del tiempo. Ha sido un impulsor fundamental para el progreso y la innovación en la economía global, desde los primeros días de la producción en masa hasta la era actual de la Industria 4.0.

2.1. Industria 4.0 y Tecnologías habilitadoras

Como cuenta en su artículo [2] el término Industria 4.0,tuvo su origen en Alemania pero ha ganado una amplia aceptación en toda Europa, representa mucho más que una simple etiqueta; es un concepto que abarca un cambio profundo en la forma en que se organizan y controlan las actividades dentro de la cadena de valor de la industria. Este enfoque revolucionario se extiende a lo largo de todo el ciclo de vida del producto, desde su concepción y diseño hasta su fabricación, distribución y servicio postventa.

Figura 2.2: Industria 4.0

La denominación Fábrica Inteligente e Internet Industrial se utilizan de manera intercambiable con Industria 4.0, ya que todos estos términos hacen referencia a la misma idea fundamental: La convergencia de las tecnologías de la información y comunicación con los procesos tradicionales de fabricación. Básicamente, consiste en dar a las fábricas una mayor inteligencia y la capacidad de optimizarse por sí mismas, aprovechando la interconexión en tiempo real de máquinas, sensores y

sistemas de información.

La implementación de la Industria 4.0 tiene un papel central el concepto de Internet de las cosas (IoT). Esto significa que los dispositivos y equipos industriales pueden recopilar, analizar y compartir datos de forma automatizada, lo cual facilita la toma de decisiones más rápida y precisa en todos los niveles de la cadena de valor.

En síntesis, la Industria 4.0 es una revolución en el sector de la manufactura, impulsada por avances significativos en tecnologías de la información, informática y software. La forma en que operan las empresas está siendo redefinida por esta transformación digital, la cual promueve la eficiencia, flexibilidad y competitividad en un mercado global cada vez más exigente.

[3] muestra los tres niveles de integración que establece el paradigma de la industria 4.0:

- Integración Vertical: Este concepto se centra en la integración de sistemas que abarcan diversos niveles jerárquicos dentro de una organización, desde los niveles más elementales como los actuadores o sensores, pasando por niveles intermedios de control y fabricación, hasta llegar a los niveles superiores de gestión de la producción, ejecución y/o planificación. Esta integración a múltiples niveles facilita, promueve y flexibiliza los procesos propios de la fabricación.

- Integración horizontal:

  Este término se refiere a la integración de sistemas entre los agentes y sistemas implicados en diversas etapas de los procesos de fabricación y planificación empresarial. Esto implica un intercambio fluido de materiales, energía, información y productos, tanto dentro de una sola empresa como entre diferentes entidades a lo largo de toda la cadena de valor

- Integración circular:

  Este tipo de integración busca unificar tanto la integración vertical como horizontal, incorporando al usuario final y abarcando todo el ciclo de vida del producto. Este enfoque busca cerrar el ciclo de producción para lograr una digitalización completa desde las primeras etapas de diseño hasta la entrega al cliente final y los servicios asociados.

Para comprender plenamente el impacto y el alcance de la Industria 4.0, es crucial examinar las tecnologías habilitadoras que hacen posible esta transformación radical en el panorama industrial. Estas tecnologías están en constante evolución y están revolucionando la forma en que se diseñan, fabrican y gestionan los productos. Algunas de las tecnologías clave que impulsan la Industria 4.0 incluyen:

- Internet de las Cosas - IOT:
  Desde el punto de vista de [4] el Internet de las cosas (IoT) implica la conexión en red de todos los objetos cotidianos, los cuales suelen estar equipados con algún tipo de inteligencia. En este contexto, Internet puede ser vista como una plataforma para dispositivos que se comunican electrónicamente y comparten información y datos específicos con su entorno. De esta manera, la IoT puede considerarse como una evolución del Internet tradicional, añadiendo una interconexión más amplia, una mayor capacidad de percepción de la información y servicios inteligentes más completos.

Figura 2.3: IoT.Fuente: [5]

- Sistemas Ciberfísicos (CPS)
  Un sistema ciberfísico emerge de la fusión entre ordenadores y redes con un proceso físico. Se caracteriza por ser "sistemas que se construyen a partir de la integración completa entre la computación y los componentes físicos". De acuerdo con la National Science Foundation, los avances en los Sistemas Ciberfísicos (CPS por sus siglas en inglés) tienen el potencial de superar a los sistemas integrados tradicionales en términos de capacidad, adaptabilidad, escalabilidad, resiliencia, seguridad y uso.

- Ciberseguridad
  La ciberseguridad industrial se centra en las acciones y protocolos diseñados para salvaguardar los sistemas, las redes y los datos empleados en entornos industriales, tales como plantas de manufactura, infraestructuras críticas y sistemas de control industrial.

- Cloud Computing

  [6], se refiere al Cloud Computing como un conjunto de servicios ofrecidos a través de internet, mediante aplicaciones configuradas por medio de la convergencia de hardware y software en centros de datos alrededor del mundo.

  Es decir,es una tecnología que te permite acceder de forma remota a recursos de computación a través de servicios de internet bajo demanda.

- Edge Computing

  Según IBM, Edge Computing es un modelo informático distribuido que acerca las aplicaciones empresariales a las fuentes de datos, como dispositivos IoT o servidores periféricos locales. Esta cercanía a los datos en su origen puede proporcionar importantes beneficios empresariales, tales como la obtención más rápida de información, tiempos de respuesta mejorados y una mayor disponibilidad de ancho de banda.

  La diferencia entre el Cloud Computing y el Edge Computing es que el cloud computing se encarga de ejecutar las cargas de trabajo en los servidores remotos de las nubes, mientras que el edge computing se dedica a la ejecución de esas cargas en los dispositivos ubicados en el extremo de la red, más cerca del usuario final o de donde se generan los datos.

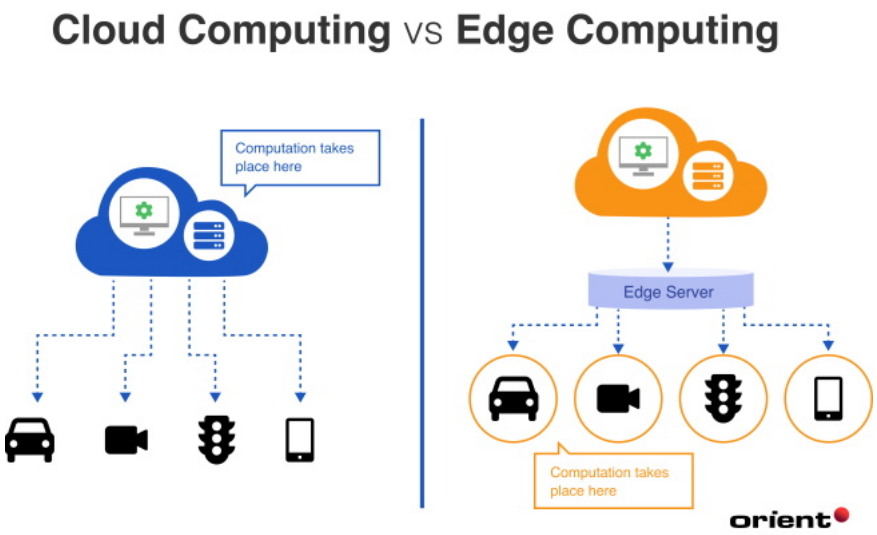


Figura 2.4: Diferencia entre Edge y Cloud Computing.Fuente:[7]

- Blockchain

  Como bien se dicta en [8] el Blockchain es como un gran libro de cuentas compartido donde se registran todas las transacciones. Cada vez que ocurre una transacción, se añade al registro, junto con la cantidad y otros detalles. Este "libro"no está guardado en un solo lugar, sino que está distribuido en muchos ordenadores conectados a la red, llamados nodos. Todos estos nodos están interconectados formando una red descentralizada y utilizan un protocolo común para comunicarse y mantener la integridad de la información.

Pero este proyecto se va a centrar principalmente en la inteligencia artificial y en la visión artificial.

2.1.1. Inteligencia artificial

[9] habla de la Inteligencia Artificial (IA) como una disciplina de las ciencias de la computación que ha captado considerable interés en la actualidad, gracias a su amplio rango de aplicaciones. La exploración de métodos que nos permitan entender la inteligencia y desarrollar modelos y simulaciones en este ámbito ha motivado a numerosos científicos a adentrarse en esta área de investigación.

Además también comenta que la intuición del destacado matemático inglés Alan Turing se atribuye al punto de partida del concepto y los criterios de desarrollo de la IA. McCarthy, fue el que acuñó el término ”Inteligencia Artificial”. Reconocidos científicos e investigadores del campo de las ciencias computacionales, como Marvin Minsky, Nathaniel Rochester, Claude Shannon, Herbert Simon y Allen Newell participaron en este encuentro. De esta reunión surgió el primer esbozo de lo que hoy en día denominamos Inteligencia Artificial, a pesar de la existencia previa de algunos trabajos relacionados con el tema.

¿Pero qué es exactamente la Inteligencia Arificial?

De acuerdo con [10], la Inteligencia Artificial (IA) se define como la capacidad de las máquinas para utilizar algoritmos, aprender de los datos y aplicar ese aprendizaje en la toma de decisiones, similar a un ser humano. No obstante, los dispositivos basados en IA no requieren descanso como las personas y pueden analizar grandes cantidades de datos al mismo tiempo. Las intervenciones humanas en las mismas tareas cometen significativamente más errores que las máquinas que realizan tareas basadas en IA.

Figura 2.5: Inteligencia artificial. Fuente:[11]

La noción de que las computadoras o programas informáticos puedan aprender y tomar decisiones es de suma importancia y requiere nuestra atención, especialmente porque sus capacidades están creciendo de manera exponencial con el tiempo. Gracias a estas capacidades, los sistemas de inteligencia artificial ahora pueden llevar a cabo muchas tareas que anteriormente eran exclusivas de los humanos

Las tecnologías basadas en IA ya están siendo empleadas para ayudar a los seres humanos a experimentar mejoras significativas y lograr una mayor eficiencia en prácticamente todos los aspectos de la vida. Sin embargo, el rápido avance de la IA también nos obliga a estar alerta para detectar y abordar posibles desventajas, tanto directas como indirectas, que puedan surgir debido a la proliferación de la inteligencia artificial.

2.1.2. Visión artificial

La informática avanza hacia la creación de computadoras más veloces, expertas y autónomas. Una de las metas más ambiciosas es dotar a las computadoras de la capacidad de interactuar con su entorno de manera similar a los humanos, a través de los sentidos.

Dotar a una computadora con capacidades sensoriales es un desafío debido a la necesidad de elementos adicionales al microprocesador, como sensores, tarjetas adaptadoras de señal y el ruido ambiental. A pesar de los obstáculos, existe un interés creciente en dar a las computadoras la habilidad de ver, mientras que otros sentidos humanos no despiertan el mismo interés.

La visión artificial engloba los procesos y componentes que otorgan capacidad visual a una máquina. Consiste en deducir automáticamente la estructura y propiedades de un mundo tridimensional, posiblemente dinámico, a partir de una o varias imágenes bidimensionales. Esto incluye propiedades geométricas y materiales, como forma, tamaño, color, iluminación y textura.

La visión, tanto para humanos como para computadoras, implica dos fases principales: Tomar una foto y después analizarla. A pesar de su complejidad, el ojo humano ya ha resuelto la fase de captura de imágenes con la cámara de vídeo. En la siguiente etapa, se realiza la interpretación de imágenes para identificar objetos y extraer información de ellos.

No obstante, el cerebro humano es quien tiene la tarea de interpretar imágenes complejas. Los ordenadores tienen que simplificar las imágenes para hacer más fácil su interpretación, lo cual implica disminuir la complejidad de la trama y eliminar el ruido. Se realiza esto a través de algoritmos matemáticos de preprocesamiento, los cuales calculan nuevas intensidades luminosas para los píxeles.

El preprocesamiento lleva mucho tiempo de cálculo, especialmente con imágenes de alta resolución, lo que impide la visualización en tiempo real por parte de los sistemas informáticos. Se deben estudiar métodos con la menor carga computacional posible para lograr la visión en tiempo real.

La visión por computadora está creciendo constantemente y enfrenta desafíos cada vez más complicados. Entre las áreas de investigación se encuentran la visión tridimensional, la visión en movimiento y la segmentación de objetos en entornos no controlados, entre otros. [12].

2.1.3. Aprendizaje automático

Como bien dicta [13] El aprendizaje automático (Machine Learning o ML) se refiere a un enfoque dentro de la inteligencia artificial que permite a las computadoras aprender y mejorar a partir de la experiencia sin ser programadas explícitamente para realizar tareas específicas. Este concepto fue pionero en 1959 por Arthur Samuel, un investigador en el campo que definió el aprendizaje automático como un çampo de estudio que da a las computadoras la capacidad de aprender sin ser programadas explícitamente". Esta habilidad es crucial para el desarrollo de sistemas inteligentes que puedan adaptarse a nuevas situaciones o con nuevos datos, mejorando así su rendimiento con el tiempo.

El ML se basa en el uso de datos para entrenar algoritmos capaces de modelar la relación entre las entradas y salidas de un sistema. Los algoritmos de ML pueden analizar grandes cantidades de datos, detectar patrones ocultos y utilizar esta información para hacer predicciones, clasificaciones o incluso generar nuevos datos. Esto se logra a través de diversos métodos y técnicas, como la regresión, redes neuronales, árboles de decisión, máquinas de vectores de soporte, entre otros.

El proceso de aprendizaje en ML generalmente se divide en varias etapas clave como preparación de datos, donde se limpiarán y organizaran datos; aprendizaje, donde el algoritmo aprende a partir de un conjunto de datos de entrenamiento; evaluación, donde se miden la precisión y eficacia del modelo; y finalmente implementación, donde se utiliza el modelo previamente entrenado para hacer predicciones o tomar decisiones en un ambiente real.

Las aplicaciones de ML son amplias y diversas. Por ejemplo, en medicina se utiliza para direccionar enfermedades desde imágenes médicas, prever el avance de la enfermedad o tratar a los

pacientes en función del caso individual. En la industria se utiliza para optimizar los procesos de producción, predecir el mantenimiento de maquinarias o mejorar la calidad de productos. En el sector financiero es esencial en todo tipo de fraudes prevenciones, potencial predicciones y manejo de riesgos. En el sector tecnológico constituye una parte esencial en todo tipo de asistentes virtuales tan diversos como motores recomendatorios y sistemas de reconocimiento en voz y texto.

2.1.4. Aprendizaje Profundo

[13] también destaca el aprendizaje profundo o Deep Learning, considerándolo como un subconjunto especializado y autónomo del aprendizaje automático. Este enfoque se distingue principalmente por el uso de redes neuronales artificiales, conocidas como redes neuronales profundas. Estas redes se llaman "profundas"debido al número de capas que tienen, lo que les permite aprender y representar características extremadamente complejas y abstractas a partir de los datos.

En la imagen 2.6 se puede apreciar la diferencia entre una red neuronal y una red neuronal profunda:

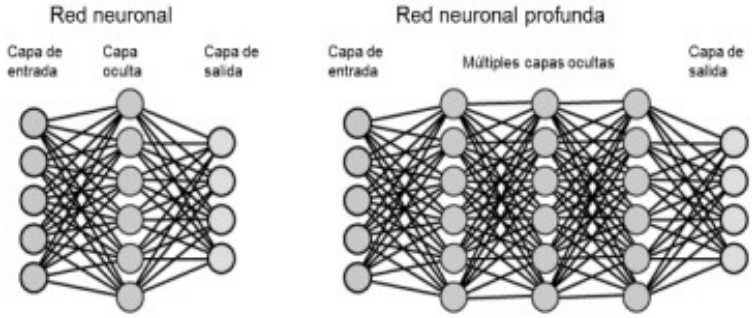


Figura 2.6: Red neuronal convencional vs red neuronal profunda.Fuente:[14]

donde se observa claramente que el nivel de profundidad de las redes de Deep Learning es mucho mayor.

El aprendizaje profundo ha transformado varios campos de la inteligencia artificial gracias a su capacidad para manejar grandes volúmenes de datos y descubrir patrones que son inaccesibles para los métodos tradicionales. A medida que las redes neuronales se vuelven más profundas, pueden aprender representaciones jerárquicas de los datos, donde las capas inferiores capturan características básicas (como bordes en imágenes), mientras que las capas superiores las combinan en representaciones más complejas (como objetos completos).

Una de las principales ventajas del aprendizaje profundo es que permite aprender a partir de datos no estructurados, como imágenes, textos o audio, con poco o ningún esfuerzo humano sustancial para diseñar manualmente las características. Esto contrasta con los enfoques convencionales de aprendizaje.

Por otro lado, el aprendizaje profundo ha demostrado ser altamente efectivo en tareas de reconocimiento de patrones, como la visión por computadora, el procesamiento del lenguaje natural, la

traducción automática, la generación de texto, el reconocimiento de voz y la conducción autónoma, entre muchas otras. La capacidad de estas redes para procesar y comprender datos complejos ha llevado a avances significativos en áreas como la medicina, donde se utilizan para el diagnóstico de enfermedades, o en la industria, donde se aplican en sistemas de recomendación y análisis predictivo.

En el aprendizaje profundo, la salida de cada capa en la red se utiliza como entrada para la siguiente, lo que permite a la red capturar información en diferentes niveles de abstracción. Esto es particularmente útil en tareas donde los datos presentan una estructura jerárquica, como en las imágenes, donde los píxeles se combinan para formar bordes, los bordes se combinan para formar texturas, y así sucesivamente hasta que se identifican objetos completos.

El proceso de aprendizaje de las redes neuronales, conocido como entrenamiento, puede ser supervisado o no supervisado. En el **aprendizaje supervisado**, la red se entrena utilizando grandes cantidades de datos etiquetados, donde cada entrada está asociada con una solución deseada o salida conocida. El objetivo es ajustar los pesos de la red para que la salida generada sea lo más parecida posible a la salida real. Este tipo de aprendizaje es común en problemas de clasificación y predicción, y se pueden distinguir varios enfoques, como el aprendizaje por corrección de error, por refuerzo y el aprendizaje estocástico.

En contraste, en el **aprendizaje no supervisado**, la red recibe un conjunto de patrones sin información sobre las salidas. Aquí, el objetivo es ajustar los pesos para que la red descubra patrones o estructuras ocultas en los datos, siendo útil en tareas como la agrupación de datos similares o la detección de anomalías, donde los datos no se ajustan a patrones predefinidos. Además, existe el aprendizaje semi-supervisado, que combina una pequeña cantidad de datos etiquetados con grandes volúmenes de datos no etiquetados, permitiendo aprovechar lo mejor de ambos enfoques.

Otro aspecto importante del aprendizaje profundo es el uso de técnicas de regularización y optimización, que son cruciales para entrenar redes profundas sin caer en el sobreajuste. Métodos como "dropout" (desactivación de un cierto número de neuronas al azar), normalización por lotes ("batch normalization") y una correcta inicialización de pesos, junto con optimizadores avanzados como "Adam" o "RMSprop", han permitido que estas redes aprendan de manera más eficiente y efectiva.

En resumen, el aprendizaje profundo es una rama del aprendizaje automático que ha transformado el enfoque para resolver problemas complejos. Su capacidad para aprender de manera autónoma representaciones complejas y características jerárquicas a partir de grandes volúmenes de datos lo convierte en una herramienta indispensable en la inteligencia artificial actual. A medida que las capacidades computacionales continúan avanzando, el aprendizaje profundo sigue abriendo nuevas posibilidades en diversos sectores, impulsando innovaciones que antes estaban más allá del alcance de los métodos tradicionales.

2.1.5. Redes Neuronales

Anteriormente se ha hablado sobre redes neuronales, redes neuronales profundas...etc. Pero ¿que es una red neuronal?

Las redes neuronales no son solo mecanismos para replicar ciertas habilidades humanas, como la memoria y la capacidad de asociación de ideas.
Las dificultades que escapan a los algoritmos tienen la misma característica: requieren experiencia.

Los humanos resuelven estos problemas basándose en su propia experiencia. Por eso, una forma de abordarlas es diseñando sistemas que imiten esta destreza humana. Así pues, las redes neuronales representan un modelo artificial del cerebro humano que se basa en el hecho mejor conocido del sistema capaz de aprender a partir de la experiencia.

Para ver de una forma más visual como funciona una red neuronal y cuales son sus componentes [15] propociona en su proyecto una imagen como la que podemos ver en la figura 2.7:

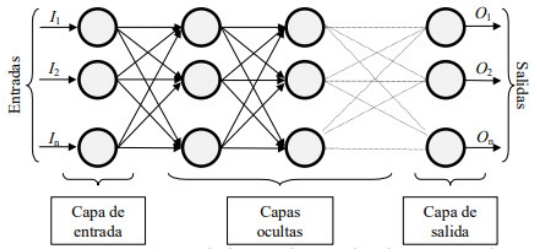


Figura 2.7: Red neuronal. Fuente: [15]

Las neuronas interconectadas y organizadas en un mínimo de tres capas forman la red neuronal, aunque este número puede ser diferente. La información entra desde la capa de entrada, que después se procesa en la capa oculta hasta que produce resultados en la capa de salida. Cabe resaltar que la capa oculta puede tener varias capas.

Como bien comenta [16] la propiedad más importante que tienen las redes neuronales artificiales es su capacidad para aprender a partir de un conjunto de datos de entrenamiento. Esto significa que pueden identificar un modelo que se ajuste a los datos proporcionados. Este proceso de aprendizaje, que también se conoce como entrenamiento de la red, puede ser supervisado o no supervisado como ya se ha explicado anteriormente.

Una vez completado el entrenamineto, se establecen los pesos de conexión en la red neuronal. El siguiente paso es verificar si la red neuronal puede abordar nuevos problemas generales para los que ha sido entrenada. Para este propósito, se requiere otro conjunto de datos, llamado conjunto de validación o conjunto de prueba.

Cada ejemplo en el conjunto de validación contiene valores de variables de entrada más su solución correspondiente, pero esta solución no tiene que ser proporcionada a la red neuronal. Luego, para cada ejemplo de validación, la solución calculada por la red se compara con la solución conocida.

En cuanto a las topologías de redes neuronales que existen se van a destacar las más relevantes y comunes según la perspectiva de este proyecto:

- Redes Neuronales Feedforward (Perceptrones Multicapa - MLP): representan una de las formas más elementales de red neuronal. En ella existen tres tipos de capas que son: la capa de entrada, las capas ocultas y la capa de salida. Los datos ingresan en la capa de entrada; pasan por las capas ocultas donde se procesan y transforman y por último llegan a la capa de salida que produce el resultado. Durante el entrenamiento, los pesos y sesgos se ajustan con el fin de minimizar el error en las predicciones mediante un proceso denominado retropropagación. Se

trata de estructuras versátiles que pueden usarse para realizar tareas tales como clasificación de imágenes o predicción de cifras.

- Redes Neuronales Convolucionales (CNN):funcionan como redes especializadas que procesan imágenes. Utilizan capas convolucionales para aplicar filtros que detectan atributos como bordes y texturas, y capas de pooling que mantienen las características más importantes mientras reducen la dimensionalidad. Luego, la información se envía a capas completamente conectadas con fines como la generación de la clasificación final de la imagen. Gracias a su capacidad para capturar relaciones espaciales en datos visuales y manejar grandes cantidades de información, las CNN son efectivas en tareas de visión por computadora, como el reconocimiento y la detección de objetos.

- Redes Neuronales Recurrentes (RNN):son una forma de red que ha sido desarrollada para evaluar una secuencia ambiental, de Las RNN son más bien sistemas que están conformados por diferentes tipos de conexiones que permiten que la información se almacene y utilice en pasos posteriores; de este modo, tienen "memoria" de lo que han procesado previamente. Gracias a esto, pueden comprender las dependencias temporales y otras relaciones dentro de estas serie. Sin embargo, las RNN pueden tener dificultades para manejar secuencias largas, lo que ha llevado a que se desarrollen variantes como LSTM y GRU que permiten recordar información durante períodos mayores.

- Redes de propagación hacia atrás (Backpropagation) : El término "Backpropagation"se refiere a cómo el error se propaga hacia atrás a través de la red neuronal, es decir, desde la capa de salida hacia las capas anteriores. Este es un proceso que permite ajustar los pesos de las conexiones entre las neuronas ocultas durante el entrenamiento.

  Ajustar los pesos en las conexiones neuronales no solo afecta la entrada global, sino que también influye en la activación de las neuronas y, por extensión, en su salida. Por lo tanto, es fundamental considerar cómo varía la función de activación debido a los cambios en los valores de los pesos. Esta variación en la función de activación en respuesta a los cambios en los pesos se conoce comúnmente como Sensibilidad de la Función de Activación.

El uso de redes neuronales tiene diversas ventajas en las que podemos destacar:

- Aprendizaje Adaptativo: La habilidad para adquirir habilidades a través de la capacitación o experiencia previa.

- Auto-organización: Mediante un proceso de aprendizaje, una red neuronal puede construir su propia forma o representación de los datos que recibe.

- Tolerancia a fallos: Aunque sufra algunos daños, una red neuronal puede seguir funcionando (en cierto grado) porque tiene una cierta redundancia.

- Operación en tiempo real: Los cálculos neuronales son paralelos e implican el uso de hardware específico que soporta este tipo de comparación.

2.2. Simuladores de procesos industriales. Factory IO

2.2.1. Conceptos

Factory I/O es un software de automatización en tiempo real diseñado para construir y simular sistemas industriales, utilizando las tecnologías de automatización más comunes. Esta herramienta ofrece una simulación interactiva con gráficos de alta calidad y sonido, creando un entorno industrial realista[17].

La característica innovadora de Factory I/O radica en su tecnología que permite la creación rápida y sencilla de sistemas industriales en 3D mediante un proceso de arrastrar y soltar. Una vez construidos, estos sistemas pueden ser controlados en tiempo real al conectarse con equipos externos como PLCs, microcontroladores, FPGA, entre otros.

Factory I/O se destaca como una valiosa herramienta educativa para la formación de futuros técnicos e ingenieros en una variedad de programas y cursos, que incluyen automatización industrial, Mecatrónica, Ingeniería Eléctrica, Ingeniería Mecánica, Instrumentación, y muchos otros.

Para conocer de una manera más cercana todo lo que engloba Factory IO es necesario llegar a conocer primero el entorno de Fcatory IO:
Pantalla Principal

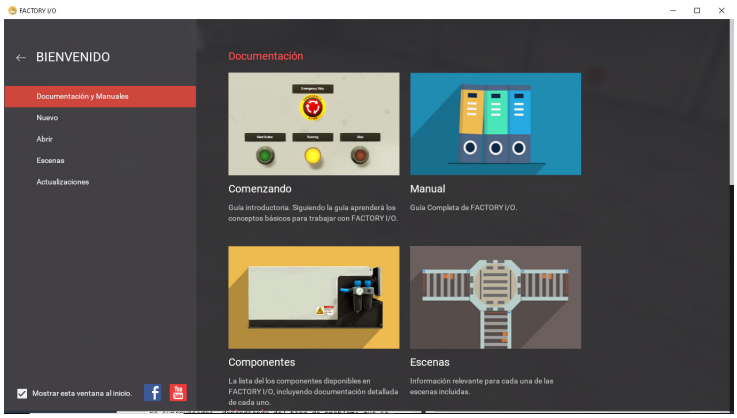


Figura 2.8: Pantalla Principal Factory IO.Fuente:Autor

Al iniciar Factory IO, se abre la pantalla principal dividida en dos partes:
A la derecha se lovaliza una variedad de documentación y manuales sobre Factory IO, que pueden ser de gran utilidad para resolver diversas consultas.

A la izquierda de la pantalla se encuentra el menú que contiene diversas opciones como crear una nueva escena, abrir una nueva escena o abrir una escena prediseñada.

Espacio virtual

El espacio virtual no es más que el entorno donde se desarrollan y se llevan a cabo de una manera visual y gráfica los procesos industriales que queremos simular.

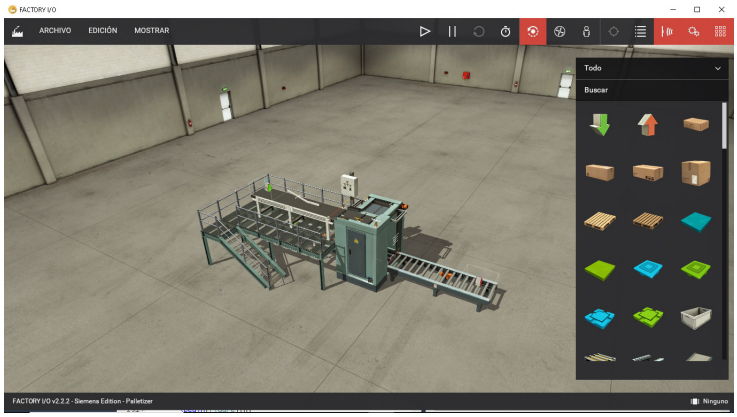


Figura 2.9: Escenario Factory IO.Fuente:Autor

Consiste en una nave en la que se puede ir añadiendo componentes tales como cintas transportadoras,sensores de todo tipo , generadores de objetos...mediante el uso del menú de la derecha.

Figura 2.10: Paleta Factory IO.Fuente:Autor

Barra de Tareas

En la pantalla donde aparece la escenografía también ocupa su lugar la barra de tareas la cual consta de los siguientes comandos:

- "Archivo":

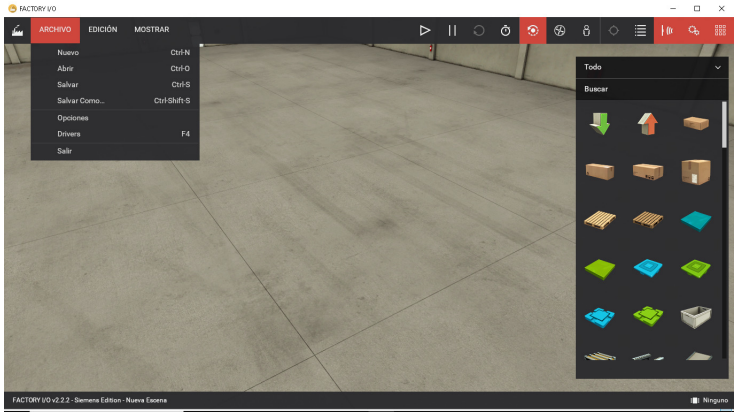


Figura 2.11: Archivo Factory IO.Fuente:Autor

Este menú permitirá realizar diversas acciones como guardar, crear, abrir o cerrar un programa. Dentro de las opciones disponibles, se puede acceder a una pantalla de configuración donde se pueden ajustar aspectos relacionados con el audio, el vídeo, los controles, las licencias, entre otros.
En esta sección ese encuentran los drivers, donde se configuran las conexiones con los distintos controladores,sensores y actuadores utilizados en el proyecto.

- "Edición":

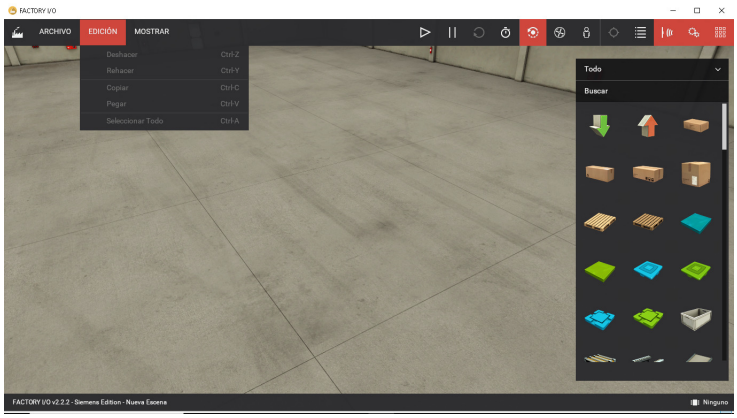


Figura 2.12: Edición Factory IO.Fuente:Autor

Dentro de este menú desplegable se encuentran opciones como copiar, pegar o deshacer, entre otras. Estas son tareas habituales que se encuentran presentes en la mayoría de los programas y permiten realizar acciones básicas de edición de manera rápida y sencilla.

- "Mostrar":

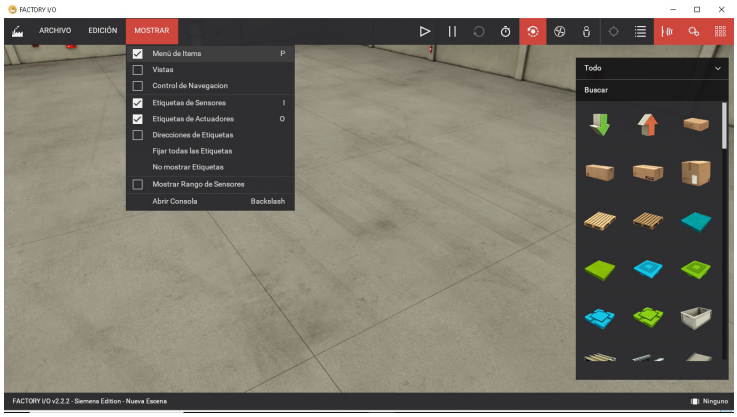


Figura 2.13: "Mostrar"Factory IO.Fuente:Autor

Desde esta pestaña, se tiene la posibilidad de seleccionar qué menús se desean mostrar o ocultar. Esto brinda flexibilidad al usuario para personalizar la interfaz según sus preferencias y necesidades específicas.

- Botones de control del proceso:

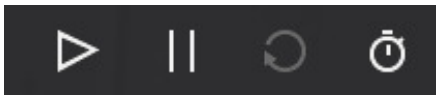


Figura 2.14: Botones de control del proceso

Con estos cuatro botones se puede controlar la simulación de manera eficaz, otorgando al usuario el poder de iniciar, detener, restablecer al estado inicial o ajustar la velocidad temporal de la simulación. Estas funciones ofrecen una experiencia interactiva y personalizable, permitiendo una mayor flexibilidad y control sobre el proceso de simulación.

- Botones de visualización del proceso:

Figura 2.15: Botones de visualización del proceso.Fuente:Autor

Estos botones controlan la perspectiva desde la cual se visualiza el proceso en Factory IO. Este software ofrece tres tipos distintos de cámaras para proporcionar una experiencia visual variada y adaptada a las necesidades del usuario.

La primera una cámara orbital, la segunda una cámara “mosca” que sería una vista similar a la que se tiene desde un dron, y la última sería una cámara de primera persona en la que se vería lo que un operario de lesa industria vería.

- Botones Multifuncionales:

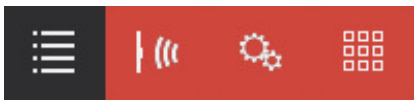


Figura 2.16: Botones Multifuncionales.Fuente:Autor

Estos botones facilitan el acceso a diferentes opciones de visualización en Factory IO. El primero muestra un listado de cámaras guardadas por el usuario, mientras que los siguientes dos muestran u ocultan las etiquetas de los sensores y actuadores. El último botón despliega la paleta con todos los componentes disponibles.

Una parte imprescindible son los ”DRIVERS” dentro de ”ARCHIVO”. En esta sección es donde se van a conectar todas las variables del programa cada item con su correspondiente, y se va a seleccionar el driver que en este caso será OPC Client DA/UA 2.17.

Figura 2.17: Drivers.Fuente:Autor

2.2.2. Sensores y Actuadores

Sensores

Los sensores que se puede usar en Factory IO se pueden clasificar de la siguiente manera como bien aparece en el manual de Factory IO [18]:

- Sensor Capacitivo:Sensor de proximidad utilizado para la detección cercana de cualquier material. Está equipado con un LED que indica la presencia de un objeto dentro de su alcance. El valor de salida puede ser digital o analógico según la configuración seleccionada.

- Sensor Difuso:Sensor fotoeléctrico difuso que puede detectar cualquier objeto sólido.

- Sensor Inductivo:Sensor de proximidad utilizado para la detección cercana de materiales conductores. Está equipado con un LED que indica la presencia de un objeto dentro de su alcance. El valor de salida puede ser digital o analógico, según la configuración seleccionada.

- Matriz de luces (emisor y receptor):Conjunto de haces luminosos paralelos utilizados para crear un campo de detección. Para garantizar la sincronización de ambos dispositivos (emisor y receptor), deben estar correctamente alineados y uno frente al otro. Una vez garantizada esta alineación (indicada por un LED verde), se pueden interrumpir todos los haces sin romper la sincronización. Puede configurarse para trabajar en modo Numérico, Digital o Analógico.

- Sensor retrorreflectante y reflector:A diferencia de los demás sensores, el catadióptrico necesita un reflector. Para funcionar correctamente, debe estar alineado con el reflector. Está equipado con dos LED que indican la alineación correcta (verde) y el estado de detección (amarillo).

- Sensor de visión:El sensor de visión reconoce las materias primas, las tapas de los productos y las bases de los productos, así como sus respectivos colores.Este sensor puede configurarse para detectar más de un tipo de pieza seleccionando la configuración adecuada:

  - Todo digital: devuelve cuatro entradas digitales que indican qué elemento se ha detectado
  - Todo Numérico: devuelve un valor que codifica el elemento detectado
  - Todo ID: devuelve un valor único (aleatorio) que identifica el elemento detectado. Se puede utilizar de forma similar a los lectores de códigos de barras o RFID.

- Lector RFID:El lector RFID puede utilizarse para leer/escribir datos de etiquetas RFID. Implementa un diseño de comunicación de disparo único en el que el controlador envía un comando al lector y recibe una respuesta.

- Codificador incremental:Varias piezas incluyen un codificador incremental que puede activar haciendo clic con el botón derecho en la pieza y seleccionando Usar codificador en el Menú contextual. Este codificador utiliza dos bits (A y B) para producir dos formas de onda que están 90 grados fuera de fase (en cuadratura).

- Operadores: Varios sensores con distintas funcionalidades como: Parada de emergencia, Selector, Potenciómetro,etc.

Actuadores

Los actuadores con los que cuenta Factory IO [18] se clasifican en :

- Emisor:Emite un elemento que se utilizará en una escena (por ejemplo, caja de cartón, paleta, etc.). Mientras un elemento todavía esté dentro del volumen del emisor, no se emiten más elementos. Puede elegir qué parte o base emitir, el tiempo entre emisiones, la cantidad de elementos a emitir y si se debe tener en cuenta la posición y/u orientación aleatoria. Un emisor se puede habilitar o deshabilitar activando o desactivando su etiqueta.
- Agente de mudanzas:Elimina uno o más elementos de la escena (por ejemplo, caja de cartón, paleta, tapa del producto) cuando se cruzan con el volumen del eliminador. Puede habilitar o deshabilitar un eliminador activando o desactivando su etiqueta.
- Piezas de carga pesada:Esta categoría incluye todas las piezas adecuadas para el manejo de cargas pesadas. El equipo es robusto, ancho, de baja altura y opera a baja velocidad.Incluye entre otros:transportador de rodillos,transportador de carga,parada de rodillo,rodillo libre...
- Piezas de carga ligera:La categoría Piezas de carga ligera incluye piezas adecuadas para manipular cargas ligeras. Diseñadas para ejecutar una tarea rápidamente y mantenerse al día con un alto flujo de trabajo, estas piezas son livianas y operan a alta velocidad.Entre otros:transportadores de correa,transportador de correa curva,puerta del transportador de correa,transportador de correa inclinada...
- Dispositivos de advertencia: Entre los que se incluyen sirena de alarma,luz de pila y luz de alerta.

2.2.3. API Web

Una de las áreas que más expansión está teniendo en la Web en los últimos años son las aplicaciones web.

Las aplicaciones web permiten una amplia variedad de funcionalidades gracias a su versatilidad. Se permite la generación automática de contenido, la creación de páginas personalizadas según el perfil del usuario, el desarrollo del comercio electrónico y la interacción con sistemas de gestión empresarial a través de una interfaz web. El mundo digital actual considera esenciales estas aplicaciones, ya que facilitan el acceso y la gestión de datos desde cualquier lugar con conexión a internet como lo señala [19] en su artículo.

Las aplicaciones web forman parte de las arquitecturas cliente/servidor, en las que un ordenador actúa como cliente para solicitar servicios y otro como servidor esperando y respondiendo a esas peticiones. En este escenario, el cliente, que puede ser un navegador web, pide recursos o servicios al servidor, que puede tratarse de una computadora remota o un sistema centralizado. Así, el servidor procesa estas solicitudes y envía las respuestas correspondientes al cliente para permitir la interacción entre el usuario y la aplicación a través de la web.

La distribución eficiente de tareas y recursos entre los clientes y servidores involucrados es fundamental en el funcionamiento de las aplicaciones web modernas, por lo tanto, este modelo de arquitectura.

Es es un conjunto de protocolos y herramientas que permiten la comunicación entre diferentes sistemas o servicios a través de la web específicamente una API WEB. Se usan solicitudes

HTTP(protocolo de transferencia de hipertexto) para la comunicación, permitiendo intercambiar datos y ejecutar acciones específicas entre aplicaciones o servicios web.

En este proyecto se va a centrar la atención en Factory IO por lo que es conveniente saber más acerca de su API Web.

Según dicta el manuel de Factory IO [20] el servidor web utiliza códigos de respuesta HTTP convencionales para indicar el éxito o el fracaso de una solicitud. En general, los códigos en el 2xx rango indican éxito, los códigos en el 4xx rango indican un problema con la solicitud y los códigos en el 5xx rango indican un error en Factory I/O.

Para activar la API Web en Factory IO es necesario primeramente abrir la ”consola”. Esta se encuentra ubicada en el menú desplegable ”Mostrar” como se ve en la imagen 2.18.Y luego se debe escribir el código **”app.web_server = True”**.

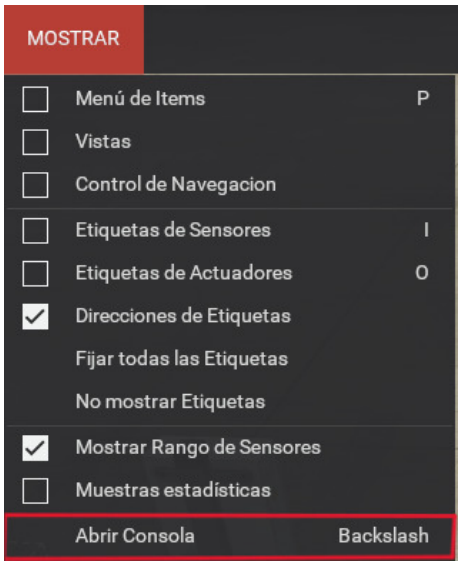


Figura 2.18: Consola_Factory IO. Fuente:Autor

De forma predeterminada, el servidor escucha en el puerto 7410, que se puede cambiar con el **”app.web_server_urlcomando”**.

2.3. Autómatas Programables

2.3.1. Generalidades

[21] define al Autómata programable (API) o PCL como un dispositivo electrónico de control con un cableado interno independiente del proceso que va a ser controlado. Un programa específico (software) se encuentra en este equipo para adaptarse al proceso, el cual contiene la secuencia de operaciones a realizar.

Se define la secuencia de operaciones sobre las señales de entrada y salida del proceso, que están conectadas directamente a los bornes de conexión del autómata. De elementos digitales pueden venir las señales de entrada, en tanto que las señales de salida consisten en órdenes digitales todo o nada o en señales analógicas expresadas como tensión o corriente que se transmiten a los indicadores y actuadores del proceso.

El autómata opera las señales de salida en función del programa de control almacenado previamente en la memoria, tomando como referencia el estado de las señales de entrada.

A través de la unidad de programación, este programa se incorpora al autómata y permite funciones adicionales como simulación, monitorización y control del mismo.

[21] diferencia claramente los bloques en los que se puede dividir un autómata programable o PLC:

- Unidad central de procesos o de control (CPU):

  Supervisa el estado de las entradas y accede a la memoria del programa para obtener la secuencia de instrucciones que debe ejecutar ejecutar. Con base en estas instrucciones, genera las señales de salida u órdenes que serán enviadas al proceso. Durante la ejecución del programa, las instrucciones son procesadas de manera secuencial, una tras otra.

  Además de ejecutar las instrucciones del programa, la unidad de control también se encarga de actualizar continuamente los temporizadores y contadores internos que hayan sido programados. Estos elementos son fundamentales para controlar el tiempo y llevar a cabo operaciones que requieran seguimiento o medición de intervalos en el proceso.

- Memorias Internas:

  La memoria del autómata contiene tanto los datos como las instrucciones necesarias para llevar a cabo la tarea de control. Dentro de esta memoria interna se guardan los datos intermedios de cálculo y las variables internas que no tienen una influencia directa en las salidas. Además, esta memoria actúa como un registro de los últimos estados leídos de las señales de entrada y de las señales de salida enviadas. Esto permite al autómata mantener un seguimiento detallado de los estados pasados y presentes del proceso controlado, lo que facilita la toma de decisiones y el control efectivo durante la ejecución del programa.

- Memorias de programa:

  Contiene la secuencia de operaciones necesarias para manipular las señales de entrada y producir las señales de salida correspondientes, además de los parámetros de configuración del autómata. Por lo tanto, realizar modificaciones en el sistema de control generalmente implica simplemente actualizar el contenido de esta memoria. Esta capacidad ofrece una flexibilidad esencial para adaptar y ajustar el funcionamiento del autómata según las exigencias específicas del proceso controlado.

- Interfaces de entrada y salida:

  Facilitan la comunicación entre el autómata y la planta. Conectadas a los bornes de señales de proceso y al bus interno del autómata, estas interfaces tienen la tarea de adaptar las señales utilizadas en el proceso a las que son manejadas internamente por la máquina. Este proceso asegura la compatibilidad y eficiencia en el manejo de las señales de entrada y salida.

- Fuente de alimentación:

  Proporciona, a partir de una tensión exterior, las tensiones necesarias para el buen funcionamiento de los distintos circuitos electrónicos del sistema.

En la imagen 2.19 se puede ver una representación en bloques de las partes de un autómata.

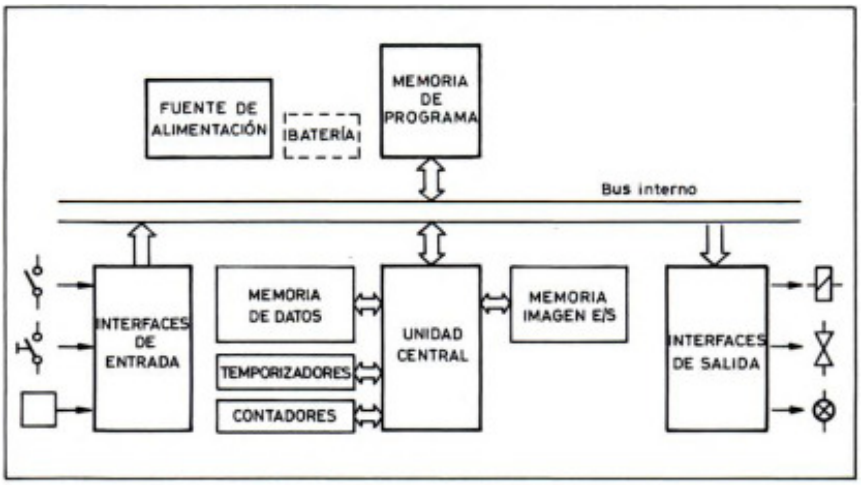


Figura 2.19: Bloques de un autómata. Fuente:[21]

2.3.2. Tipos de autómatas

Se pueden distinguir varios tipos de autómatas programables:

- Autómatas finitos (AF): Los autómatas finitos son representaciones sencillas de máquinas abstractas que cuentan con un número limitado de estados, una entrada también limitada y un conjunto de reglas de transición que indican cómo se produce el cambio entre los estados en respuesta a la entrada. Se utilizan mucho en la teoría de la computación para representar sistemas de control, procesadores de lenguaje y protocolos de comunicación.

- Autómatas de pila (AP): Los autómatas de pila son una versión ampliada de los autómatas finitos que emplean una pila o estructura de datos similar para guardar información temporal. Disponen de un número limitado de estados, una entrada con límite finito y una pila que posibilita el acceso restringido a datos previos. Los autómatas de pila son beneficiosos para el análisis de la sintaxis de los lenguajes formales y para desarrollar compiladores y analizadores léxicos.

- Autómatas celulares (AC):Los autómatas celulares son modelos discretos en los que un espacio de dos o tres dimensiones está formado por células, cada una con su propio estado. Los estados de las células cambian en pasos discretos siguiendo reglas locales simples que se basan en el estado de las células cercanas. Se emplean autómatas celulares en distintos campos, como la simulación de sistemas físicos, la modelización de fenómenos naturales y la generación computacional de gráficos.

- Autómatas de estado finito (FSM):Los autómatas de estado finito son modelos matemáticos que describen sistemas con la capacidad de encontrarse en uno entre un conjunto limitado de estados en cualquier instante. Se usan estos autómatas en diversas aplicaciones, como el diseño de circuitos digitales, la planificación de rutas en sistemas de navegación y la modelización de sistemas de control.

- Autómatas de Mealy y de Moore:Estos tipos de autómatas de estado finito difieren en cómo producen salidas como respuesta a las entradas. En un autómata de Mealy, las salidas se relacionan con las transiciones entre estados; por otro lado, en un autómata de Moore, únicamente se asocian con los estados. Se utilizan estos autómatas en el diseño de circuitos secuenciales y la implementación de sistemas de control.

Aquí hay solo unos pocos ejemplos de los diferentes tipos de autómatas que se usan en informática y otras disciplinas. La elección del tipo adecuado de autómata depende del problema que se esté enfrentando y de los requisitos específicos del sistema, ya que cada tipo tiene sus propias características y aplicaciones.

2.3.3. Programación IEC 61131-3. CODESYS

Los PLCs, como se explicó anteriormente, son dispositivos electrónicos que permiten la programación y el control de procesos en tiempo real. Son ampliamente utilizados en varios sectores industriales. Debido a la gran demanda mundial de PLC, hay muchos fabricantes que ofrecen estos dispositivos con su propio lenguaje de programación. Algunos de los fabricantes más reconocidos de PLC incluyen Festo, Mitsubishi, Siemens, Schneider Electric y otros.

Se estableció un estándar internacional, la norma IEC 1131, con el objetivo de regular todo lo relacionado con las tecnologías PLC, especialmente los lenguajes de programación según la norma IEC 61131-3.

Los lenguajes del estándar IEC 61131-3 se pueden clasificar en dos categorías principales: Lenguaje gráfico y lenguaje textual[22].

Dentro del lenguaje gráfico se pueden destacar:

- Diagrama de Funciones Secuenciales (SFC):Este lenguaje de programación se compone de etapas y transiciones. Mientras la etapa esté activa, se ejecutan las instrucciones asociadas, pudiendo cada etapa estar inactiva o activa.

- Diagrama de Bloques de Funciones(FBD):es un lenguaje gráfico que se emplea en ausencia de ciclos. Se compone de varios elementos funcionales conectados gráficamente, tales como la aritmética booleana y otros tipos de elementos.

- Diagrama de Contactos(LD):es otro lenguaje gráfico muy común en la programación de PLC. Se caracteriza por su similitud con el diseño de circuitos de lógica cableada, lo que permite desarrollar la misma lógica tanto en el diseño de circuitos como en la elaboración de programas para controladores lógicos.

En cuento a la programación textual se encuentran:

- Texto Estructurado(ST):tiene una estructura distintiva, clara y ordenada, utilizando instrucciones similares a comandos y expresiones matemáticas para facilitar la implementación de algoritmos complejos.Es adecuado para aplicaciones que requieren operaciones complejas, estructuras de datos avanzadas y manipulación detallada de variables. Permite la programación de bucles, condiciones y funciones, proporcionando un alto grado de flexibilidad y potencia para el desarrollo de aplicaciones de control.

- Lenguaje de Lista de Instrucciones(IL):es un lenguaje textual de bajo nivel que utiliza una secuencia de instrucciones simples, similares a las de un lenguaje ensamblador, para programar PLCs. Cada instrucción representa una operación básica, como lógica booleana o aritmética.

Actualmente, hay muchos paquetes de software de programación para PLC, desarrollados por fabricantes específicos para su uso exclusivo con sus dispositivos. No obstante, existen programas informáticos elaborados para integrar varios fabricantes. Es crucial considerar que no todos los paquetes de software admiten la programación en todos los lenguajes del estándar IEC-61131-3.

Codesys es un software destacado que permite el uso de los lenguajes del estándar IEC 61131-3 para la programación de PLC. Codesys también brinda la opción de programar con una extensa gama de fabricantes de PLC, como Bosch, Schneider Electric, Festo, IFM y otros más. Esto es especialmente útil en entornos industriales donde se utilizan equipos de diferentes fabricantes, facilitando así la programación y la interoperabilidad entre dispositivos y marcas variadas.

Para adentrarnos más en el mundo de Codesys es necesario explicar el entorno de este software brevemente:

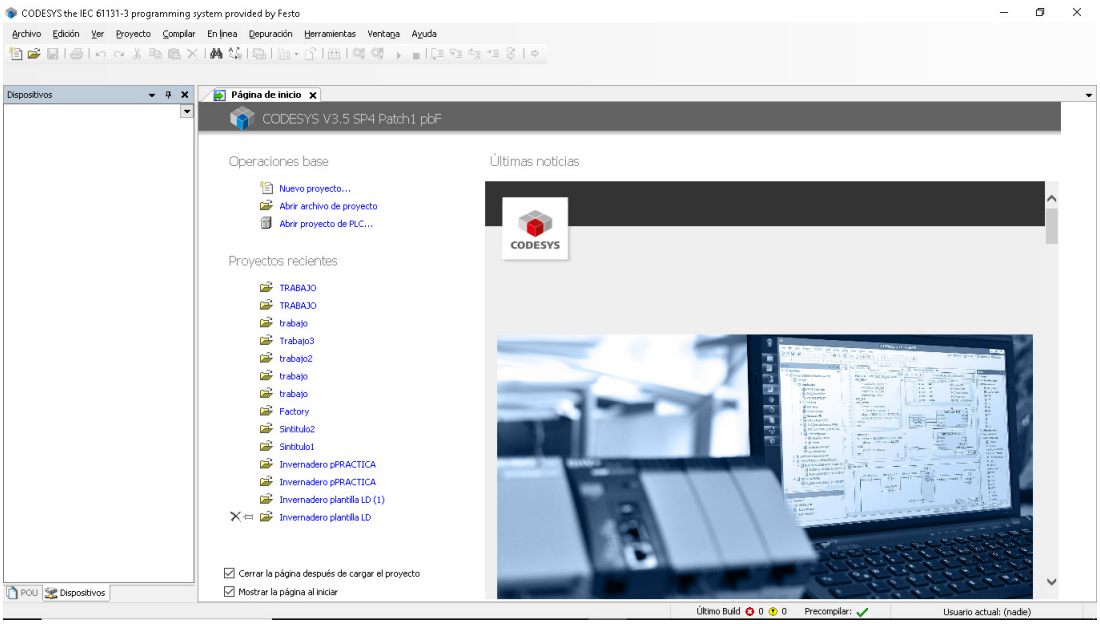


Figura 2.20: Pantalla de inicio codesys. Fuente: Autor

En la parte superior de esta primera pantalla principal de codesys nos aparecen distintos menús como los típicos menús de ”Archivo”,”Edición”,”Ver”,”Proyecto”...Y también nos aparece el menú ”compilar”cuando se quiera corroborar que nuestro proyecto, síntasis, etc está todo correcto. Otro menú importante es el menú ”En Linea” el cuál permite estar en contacto con el PLC.Todo estos menús se puede ver en la imagen 2.21.

Figura 2.21: Menús Codesys-FUente:Autor

A la hora de crear un nuevo proyecto aparecerá la siguiente ventana representada en la figura 2.22

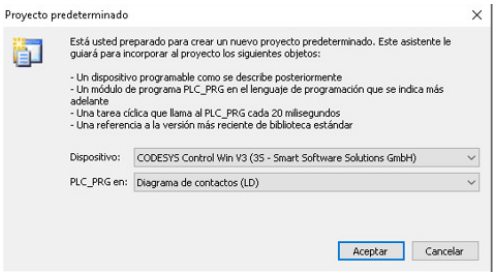


Figura 2.22: Pantalla de dispositivo y lenguaje. Fuente: Autor

En esta ventana se solicitará el dispositivo que se va a utilizar y el lenguaje de programación propio del proyecto.

Una vez se ha avanzado aparecerá el área de trabajo mostrado en la figura 2.23 el cuál consta de las siguientes secciones de interés:

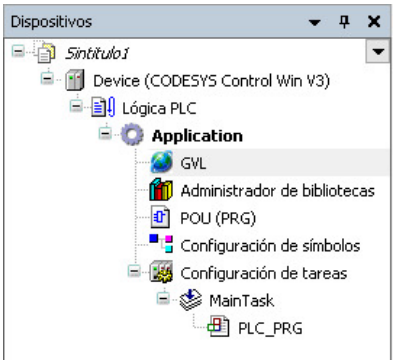


Figura 2.23: Área de trabajo. Fuente: Autor

- Dispositivo (Device) : el cual ya hemos seleccionado previamente.
- GVL: Archivo donde se encuentran declaradas todas las variables globales.
- Administración de bibliotecas: donde pueden agregarse o borrar bibliotecas.
- POU : es una unidad de funcionamiento del programa y puede ser un programa ,una función o un bloque se función.
- Configuración de Símbolos: Permite que se expongan variables y otros elementos del proyecto para que puedan ser utilizados o monitoreados por sistemas externos, a través de protocolos de comunicación como OPC (OLE for Process Control) o OPC UA (Unified Architecture). En este caso será OPC UA/DU.
- Configuración de tareas: es donde se va a llamar a nuestra tarea o nuestro POU.
- Tarea (Main Task): Nos permite darle prioridad y decidir que tipo de tarea va a realizar (cíclica,evento...).

Si se selecciona el área de trabajo principal (seleccionando PLC_PRG) aparecerá la siguiente pantalla mostrada en la figura 2.24.

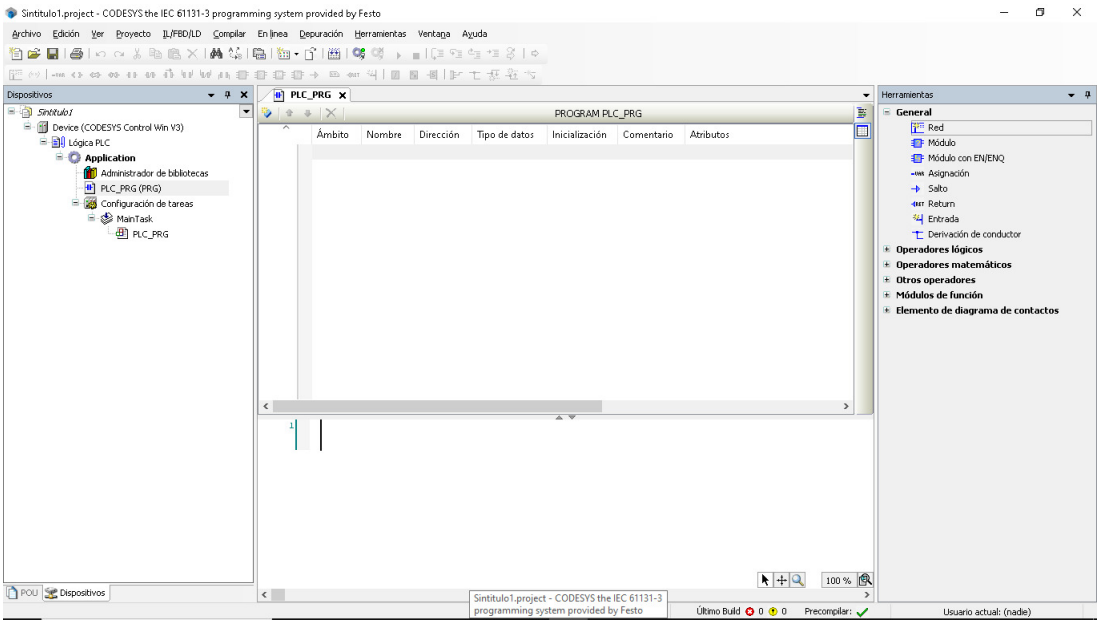


Figura 2.24: PLC_PRG. Fuente: Autor

En esta pantalla es donde se va a realizar la programación principal del programa ya sea en código de escalera (LD), Texto estructurado(ST), etc. En ella se puede distinguir en la parte superior y en la parte lateral dos menús de herramientas.

En el menú superior mostrado en la imagen 2.25 se encuentran los comandos de ”guardar”, ”compilar”,”archivo”,”iniciar sesieon”...etc

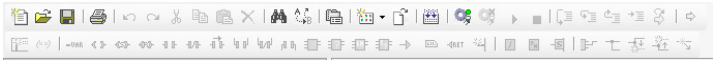


Figura 2.25: Menú Herramientas superior. Fuente: Autor

En el menú lateral podemos encontrar todo tipo de elementos para formar nuestro programa, ya sean bobinas,contactos,lineas de unión...etc como bien se aprecia en la imagen 2.26

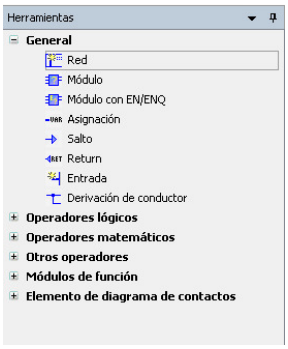


Figura 2.26: Menú Herramientas Lateral. Fuente: Autor

Una parte fundamental es la configuración de símbolos de la que ya se ha hablado antes. Ahí es donde se seleccionan las variables locales (GVL) con el fin de hacerlas visibles en Factory IO.Se muestra está sección en la siguiente imagen 2.27:

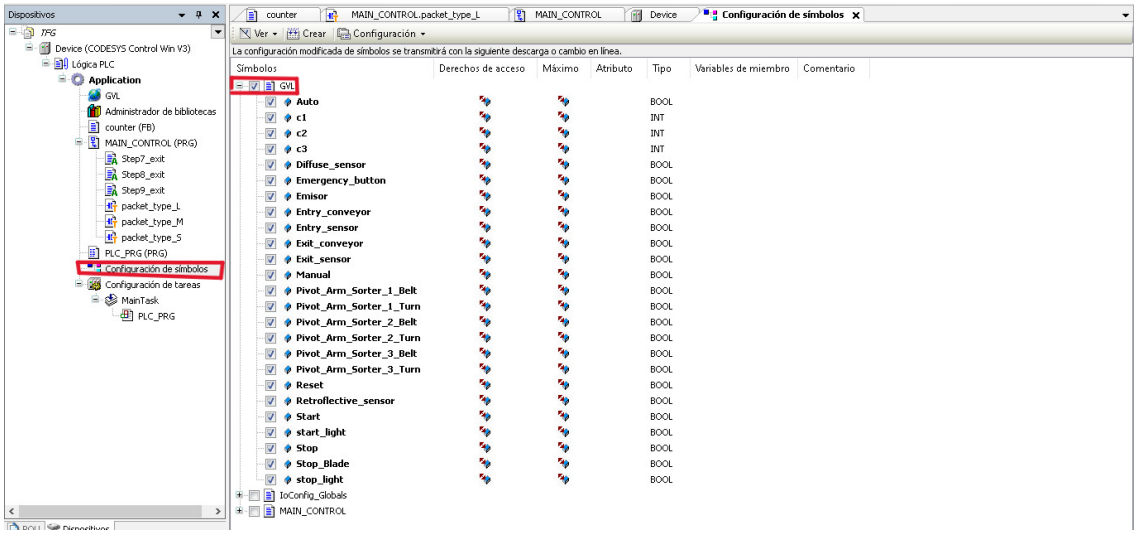


Figura 2.27: Configuración de símbolos. Fuente: Autor

Una ves finalizada la primera toma de contacto con el entorno de Codesys es necesario tener en cuenta las comunicaciones industriales.

2.4. Comunicaciones Industriales

Las comunicaciones industriales son un componente esencial en el entorno de la automatización y control de procesos industriales. Estas tecnologías permiten la transmisión de datos y la interconexión de dispositivos y sistemas dentro de entornos industriales, facilitando la supervisión, el control y la gestión eficiente de los procesos de producción.

En un mundo cada vez más interconectado, las comunicaciones industriales juegan un papel crucial en la optimización de la productividad, la mejora de la calidad del producto, la reducción de costos operativos y la garantía de la seguridad en el lugar de trabajo. Desde la transmisión de datos entre sensores y actuadores hasta la integración de sistemas de control distribuido y la implementación de soluciones de Internet Industrial de las Cosas (IoT), las comunicaciones industriales abarcan una amplia gama de tecnologías y protocolos diseñados para satisfacer las necesidades específicas de cada aplicación industrial.

2.4.1. OPC

OPC es un estándar que se basa en tecnologías como OLE, COM y DCOM de Windows. Ofrece una interfaz abierta que simplifica el intercambio estandarizado de datos entre aplicaciones y dispositivos que controlan procesos de automatización. La fundación OPC (OPC Foundation) se

estableció en 2009 para mantener y actualizar este estándar.

Con OPC, se pueden extraer datos de campo y comunicarlos con varias aplicaciones de automatización, control y gestión como SCADA, DCS y ERP en un formato estándar independiente del fabricante del hardware. OPC trabaja bajo una arquitectura cliente/servidor(constituido por proveedor y el consumidor).

El servidor OPC es una aplicación que se encarga de la comunicación con los instrumentos de campo para intercambiar información del proceso. Las aplicaciones cliente OPC reciben estos datos para su procesamiento. Además de la comunicación local OPC en un mismo equipo, esta tecnología posibilita el intercambio de datos a través de una red, lo que permite la comunicación entre aplicaciones distintas ejecutadas en computadoras diferentes. [23].

Existen cuatro tipos de servidores OPC definidos según [24]:

- **Servidor OPC DA**:Utiliza datos de acceso especialmente configurados para la transmisión instantánea de datos en tiempo real.
- **Servidor OPC HDA**:Se enfoca en el acceso a datos históricos y proporciona al cliente HDA los datos correspondientes.
- **Servidos OPC A&E server**:Se centra en la especificación de Alarmas y Eventos, permitiendo la transmisión de datos desde un dispositivo hacia el cliente OPC A&E.
- **Servidor OPC UA**:Se fundamenta en la especificación de la Arquitectura Unificada, lo que habilita a los servidores OPC para operar con una amplia variedad de tipos de datos.

2.4.2. Servidor OPC Codesys

El servidor CODESYS OPC tiene como función principal facilitar el intercambio de datos (lectura/escritura) con el controlador, lo cual es útil para visualizaciones o programas de registro de datos de proceso. Se integra en la configuración del sistema de desarrollo CODESYS IEC 61131-3. Este servidor es un programa adicional diseñado para Microsoft Windows. Se necesita una licencia para su uso, la cual se puede obtener a través de un dispositivo USB (CODESYS Key) o mediante el SoftContainer instalado en la PC que ejecuta el servidor CODESYS OPC[25].

Para conectarnos con Codesys al PLC virtual es necesario ejecutar en Windows ”Control CODesys Win V3” como aparece el la imagen 2.28 y se selecciona:

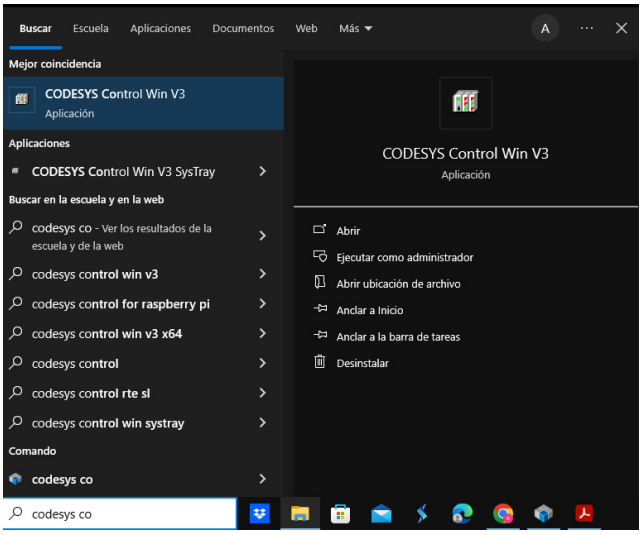


Figura 2.28: Codesys_Control.Fuente: Autor

O bien en la pestaña de iconos ocultos de Windows, seleccionar ”CODESYS Control Win SysTray” y inicializarlo mediante ”Start” como bien explica la figura 2.29.

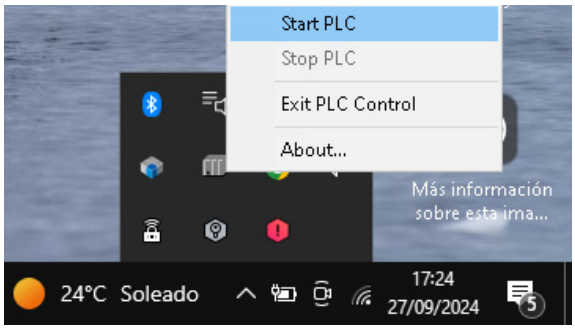


Figura 2.29: OPC Codesys COntrol Win. Fuente:Autor

Posteriormente se regresa a la pantalla principal de Codesys y se selecciona la opción de "Device (CODESYS Control Win V3 x64)".Luego, la pasarela Gateway-1 se coloca en estado activo como aparece en la figura 2.30.

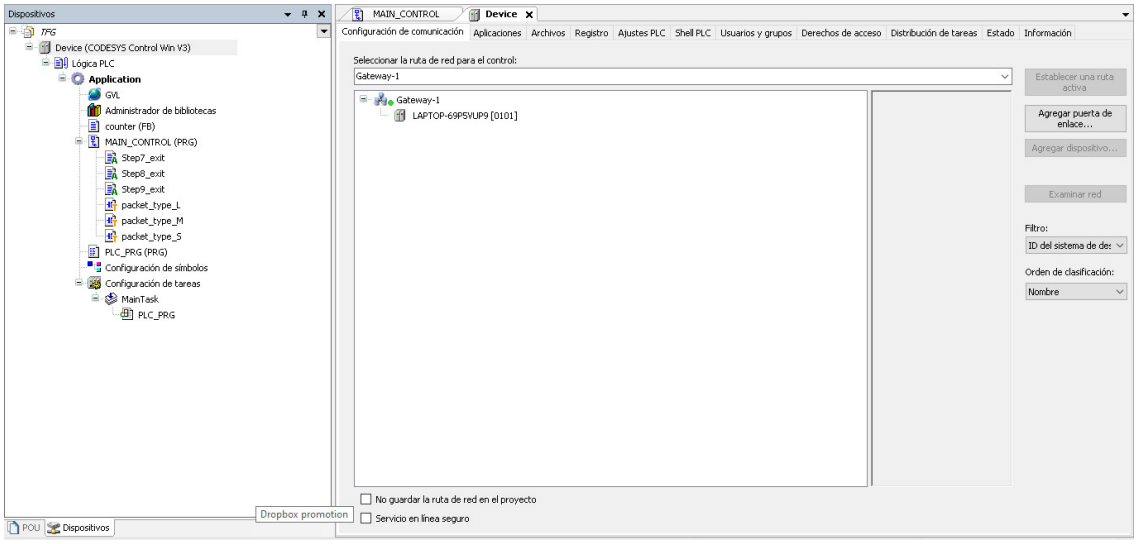


Figura 2.30: Codesys_Control_2.Fuente: Autor

Finalmente se selecciona ”En Linea”-”Iniciar la sesión” y deberá conectarse con el PLC virtual.

Ahora será necesario conectar el ”OPC Configurator”. Para ello se busca de nuevo en Windows y se selecciona o bien se busca directamente en la carpeta de descarga de Codesys.
Click derecho en Server y se selecciona ”Append PLC” como en la imagen 2.31.

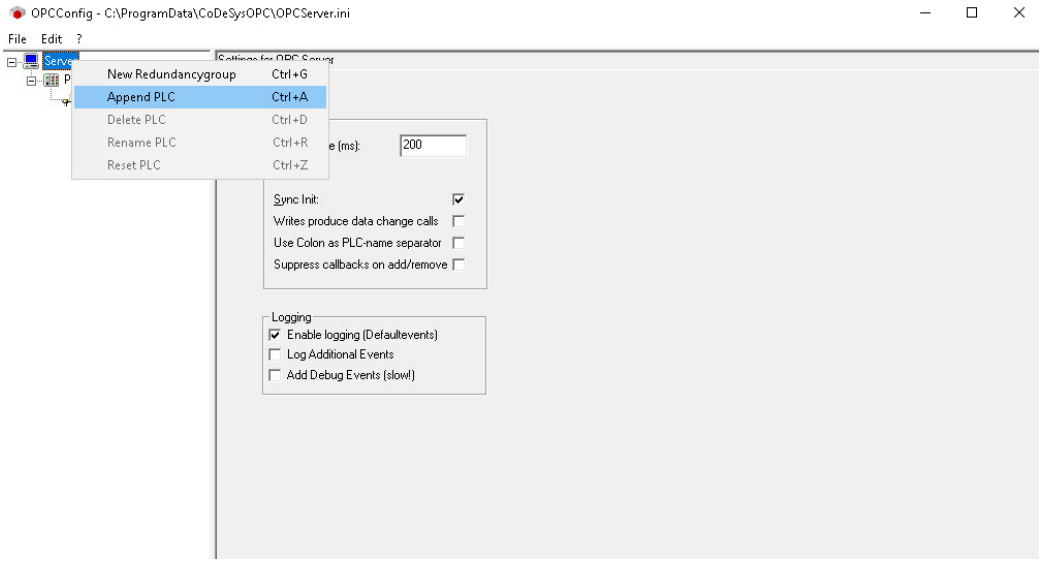


Figura 2.31: Codesys_OPC.Fuente: Autor

No se modifican las opciones que vienen por defecto y consecutivamente se hace click ”Conecction”-”Edit”y en la ventana de PLC Address se debe dejar las opciones tal y como aparecen en la figura 2.32.Es posible y recomendable cambiar el ”PLC_name” y usar el mismo nombre que tiene el

dispositivo en la sección de ”Device”.

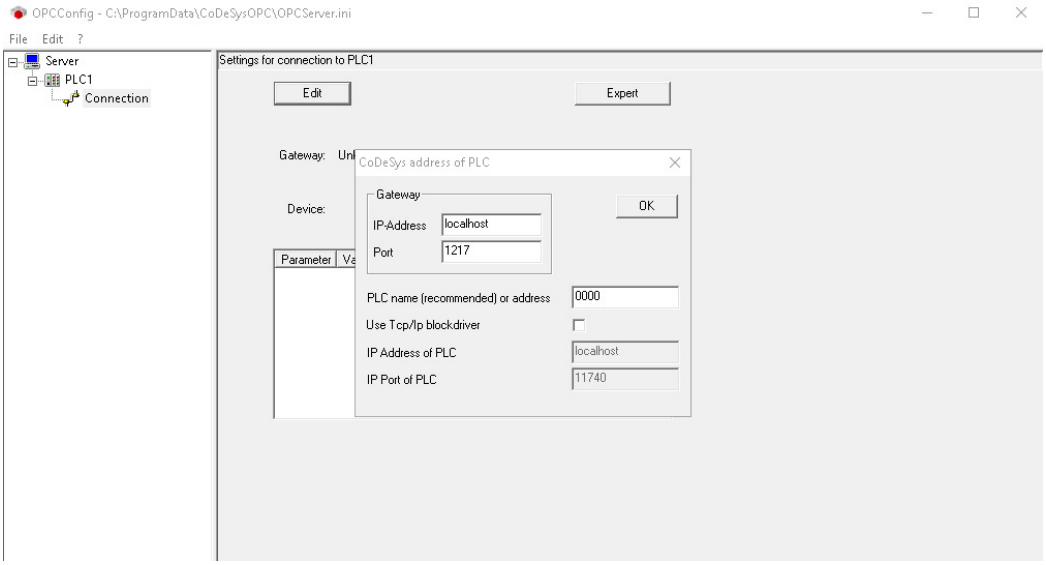


Figura 2.32: Codesys_OPC_2.Fuente: Autor

Aquí se puede observar que se muestra la IP y el puerto de Codesys. Se dejan las opciones por defecto y se pulsa ”OK”.

2.4.3. Cliente OPC en Factory IO

Anteriormente se ha configurado el programa de Codesys y OPC Codesys. Ahora se va a configurar el OPC de Factory IO.

En la escena principal de Factory IO se selecciona ” Archivo” y luego ”Drivers”. En la ventana desplegable, se elige ”OPC Client DA/UA” y después se pulsa en ”Configuration”.En la imagen 2.33 se muestra como hacerlo.

Figura 2.33: Factory_OPC.Fuente: Autor

En esta ventana, hacemos clic en el botón ”Muestra Servidores” y luego seleccionamos nuestro servidor, en este caso ”Codedys.OPC.DA” como bien indica la figura 2.34. Después, hacemos clic en el botón ”Muestra” para que encuentre todas nuestras variables.

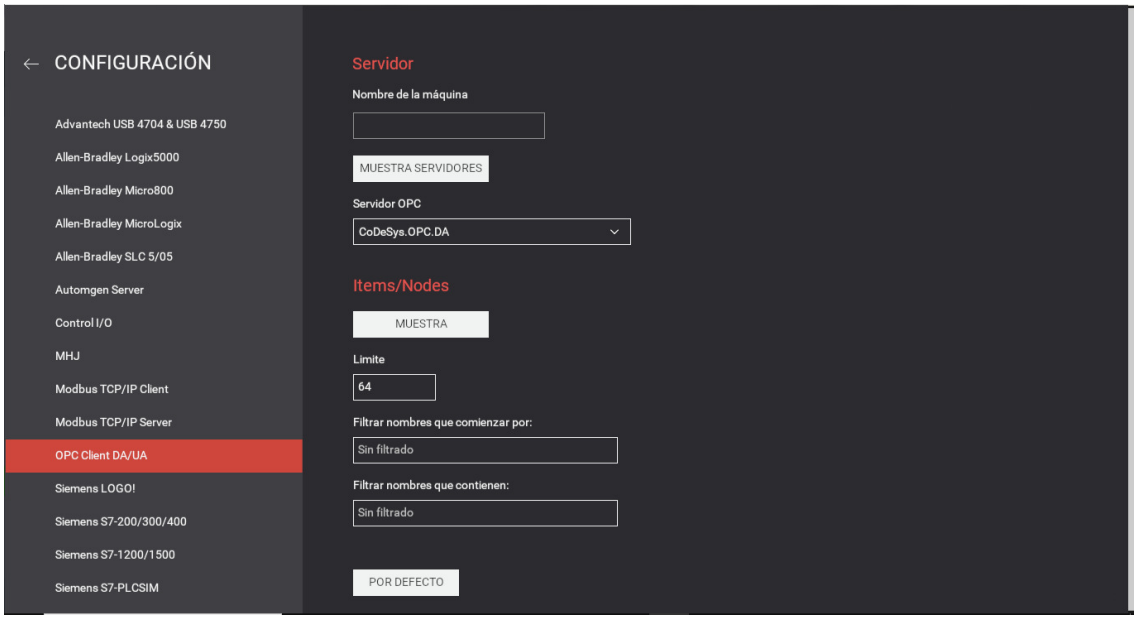


Figura 2.34: Factory_OPC_2.Fuente: Autor

Lo siguiente será arrastrar cada variable de Factory I/O hacia su correspondiente variable del recuadro central para que quede algo parecido a la figura **??**.

Figura 2.35: Conexión de variables. Fuente Autor

Capítulo 3

Descripción del proceso sometido a estudio

3.1. Descripción general

El objetivo principal de este proyecto es desarrollar e implementar un sensor de visión artificial para su integración en un simulador de procesos industriales, con el fin de mejorar la funcionalidad y reducir costos en comparación con los sensores de proximidad convencionales, que son comunes en los entornos industriales. Este proyecto se enfoca en la automatización de un proceso específico de clasificación de paquetes, utilizando un autómata programable y cumpliendo con las directrices del estándar IEC-61131-3.

Este proyecto se llevará a cabo en dos fases principales. La primera fase se implementará en el simulador Factory IO® (Real Games), que reproduce un escenario virtual en el que las cajas se clasificarán según su tamaño (grande, mediano, pequeño). Inicialmente, se utilizarán sensores de proximidad convencionales, como sensores retroreflectivos o difusos, para resolver este proceso.

En la segunda fase del trabajo, se desarrollará y entrenará un modelo de inteligencia artificial basado en aprendizaje profundo utilizando imágenes sintéticas extraídas del simulador. Se empleará el lenguaje Python, junto con el entorno Jupyter Notebook, para entrenar y validar los modelos, y para implementar el sistema en tiempo real. Permitirá la detección y clasificación de objetos con alta precisión, reemplazando varios sensores de distancia convencionales por un único sistema de visión artificial. Este sistema contará con servicios web para integrar el sensor virtual con el simulador y utilizará el software Open Broadcaster Software (OBS) para capturar imágenes en tiempo real.

Finalmente, se compararán los resultados obtenidos de ambas aproximaciones: el uso de sistemas tradicionales frente al nuevo enfoque de visión artificial, definiendo las características de mejora y reduciendo costos.

3.2. Entradas y Salidas

3.2.1. Proceso con sensores de visión convencionales

En la tabla 3.1 se muestran todas las entradas y salidas del proceso de automatización relacionado con el uso de los sensores convencionales como son el sensor retroreflectivo y el sensor difuso.

Country List		
Nombre	Entrada/Salida	Tipo
Pivot Arm Sorter 1 Turn	Salida	Boolean
Pivot Arm Sorter 1 Belt	Salida	Boolean
Pivot Arm Sorter 2 Turn	Salida	Boolean
Pivot Arm Sorter 2 Belt	Salida	Boolean
Pivot Arm Sorter 3 Turn	Salida	Boolean
Pivot Arm Sorter 3 Belt	Salida	Boolean
Entry conveyor	Salida	Boolean
Exit conveyor	Salida	Boolean
Stop Blade	Salida	Boolean
Counter1	Salida	Integer
Counter2	Salida	Integer
Counter3	Salida	Integer
Stop Light	Salida	Boolean
Start Light	Salida	Boolean
Diffuse sensor	Entrada	Boolean
Retroreflective sensor	Entrada	Boolean
Exit sensor	Entrada	Boolean
Entry sensor	Entrada	Boolean
Manual	Entrada	Boolean
Start	Entrada	Boolean
Stop	Entrada	Boolean
Emitter	Entrada	Boolean
FACTORY I/O (Running)	SYSTEM TAGS	Boolean
FACTORY I/O (Paused)	SYSTEM TAGS	Boolean
FACTORY I/O (Reset)	SYSTEM TAGS	Boolean

Cuadro 3.1: Entradas y Salidas Proceso con sensores convencionales

3.2.2. Proceso con sensor visión artificial

En la tabla 3.2 se muestran las entradas y salidas del proceso de automatización relacionado con el uso del sensor de visión artificial.

Country List		
Nombre	Entrada/Salida	Tipo
Pivot Arm Sorter 1 Turn	Salida	Boolean
Pivot Arm Sorter 1 Belt	Salida	Boolean
Pivot Arm Sorter 2 Turn	Salida	Boolean
Pivot Arm Sorter 2 Belt	Salida	Boolean
Pivot Arm Sorter 3 Turn	Salida	Boolean
Pivot Arm Sorter 3 Belt	Salida	Boolean
Entry conveyor	Salida	Boolean
Exit conveyor	Salida	Boolean
Optical Sensor	Entrada	Boolean
Counter1	Salida	Integer
Counter2	Salida	Integer
Counter3	Salida	Integer
Stop Light	Salida	Boolean
Start Light	Salida	Boolean
Vision sensor	Entrada	Integer
Exit sensor	Entrada	Boolean
Entry sensor	Entrada	Boolean
Manual	Entrada	Boolean
Start	Entrada	Boolean
Stop	Entrada	Boolean
Emitter	Entrada	Boolean
FACTORY I/O (Running)	SYSTEM TAGS	Boolean
FACTORY I/O (Paused)	SYSTEM TAGS	Boolean
FACTORY I/O (Reset)	SYSTEM TAGS	Boolean

Cuadro 3.2: Entradas y Salidas Proceso con sensor visión artificial

3.3. Proceso con sensores de proximidad convencionales

Como bien se mencionó con anterioridad en este proceso se usarán dos tipos de sensores como son el sensor retroreflectivo y el sensor difuso. Estos será implementados en la escena como se aprecia en la figura 3.1.

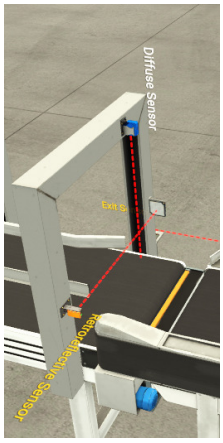


Figura 3.1: Sensores Convencionales. Fuente:Autor

La sequencia que se seguirá en todo el proceso industrial será la siguiente:

1. Inicializar – Botón marcha. Las dos cintas se ponen en ON. Esta inicialización se encuentra reflejada en la imagen 3.2:

Figura 3.2: Puesto de Inicialización. Fuente:Autor

2. Si un paquete llega a Entry_Sensor:

- Subir hoja.
- esperar 2s a que paquete llegue a hoja.
- parar cinta Entry_Conveyor.

3. Clasificar paquete (S,M,L) y esperar 1s. Esta clasificación se llevará a cabo dependiendo de los valores de los sensores difuso y retroreflectivo de la siguiente forma:

DS	RRS	OUTPUT
0	1	S
1	1	M
1	0	L

Cuadro 3.3: Salida sensores convencionales.Fuente:Autor

4. Bajar hoja, activar cinta Entry_Conveyor y activar pivote correspondiente dependiendo si el paquete es S, M o L.

5. Cuando el sensor de rampa se desactive, esperar 1s y resetear pivotes.En la imagen 3.2 se puede comprobar el número de paquetes que se han ido clasificado de cada uno de los tipos mediante los contadores.

6. Paquete de ruta 'S' Paquete de ruta 'M' Paquete de ruta 'L'.

7. Repetir hasta botón de paro.

3.4. Proceso con sensor de visión artificial

En este caso el uso de los sensores convencionales se sustituye por el uso del sensor de visión artitifical. Simularemos este sensor haciendo uso de un sensor de visión el cual de valores de 0, 1 o 2 ya sea el paquete tipo L, M o S respectivamente.

Este sensor recibirá la información que se ha obtenido una vez se hayan procesado los frames que va capturando OBS mediante programación con Python.Para que no haya ninguna interferencia no deseada el sensor se colocará "escondido" en la escena (es decir, no saldría en la cámara configurada) como se ve en la imagen 3.3:

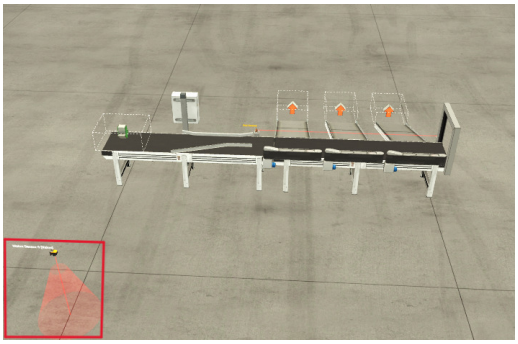


Figura 3.3: Vision Sensor. Fuente:Autor

La secuencia que se seguirá es practicamente idéntica:

1. Inicializar – Botón marcha. Las dos cintas se ponen en ON. Ya se ha visto en la figura 3.2 donde está ese botón de marcha.

2. Si un paquete llega a Optical_Sensor 3.4:

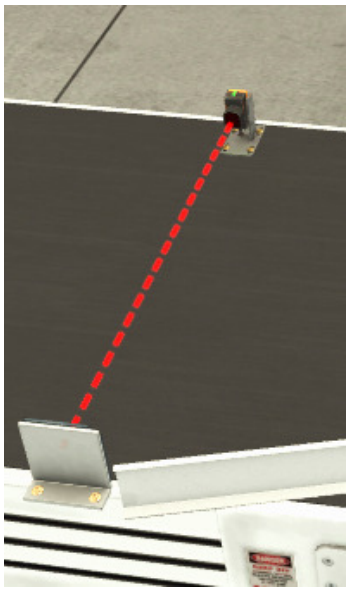


Figura 3.4: Optical Sensor

   - Se para Entry_conveyor.
   - Se llama al modelo entrenado para que procese las imágenes necesarias para clasificar el paquete mediante ese modelo entrenado.
   - esperar 3s a que el paquete sea clasificado.

3. Clasificar paquete (S,M,L). Esta clasificación se llevará a cabo dependiendo del valor del sensor de visión artificial de la siguiente forma:

AVS	OUTPUT
0	L
1	M
2	S

Cuadro 3.4: Salida Vision Sensor.Fuente:Autor

4. Activar cinta Entry_Conveyor y activar pivote correspondiente dependiendo si el paquete es S, M o L.

5. Cuando el sensor de rampa se desactive, esperar 1s y resetear pivotes.Al igual que en el proceso con los sensores convencionales el número de los disintos tipos de paquetes almacenados se puede ver en las variables contadores que están reflejadas en la imagen 3.2.

6. Paquete de ruta 'S' Paquete de ruta 'M' Paquete de ruta 'L'.

7. Repetir hasta botón de paro.

Para la clasificación en tiempo real se hará uso de la cámara virtual que posee OBS. Esta se encargará de ir capturando frames los cuales serán procesados mediante Python para ser posteriormente clasificados y enviados al sensor de visión ("Vision sensor") del que se ha hablado anteriormente.

Dependiendo del valor del valor "Vision sensor",los cuales se han mencionado anteriormente en la tabla 3.4, los paquetes se dirigirán a una u otra salida dependiendo del tipo de paquete que sea.

Capítulo 4

Síntesis del automatismo con GRAFCET

4.1. Introducción a la metodología GRAFCET

El GRAFCET representa un estándar internacional para la especificación de controladores lógicos en sistemas automatizados. Proporciona una forma de representar tanto sistemas secuenciales como simultáneos, lo que significa que puede describir la secuencia de eventos y las acciones concurrentes en un sistema.

Es utilizado como un lenguaje de alto nivel para describir, especialmente en el área del control industrial. Mediante GRAFCET, se puede definir el comportamiento de sistemas embebidos en distintos niveles de detalle, lo que posibilita una descripción clara, inequívoca y completa del sistema y su operación. La fiabilidad y la precisión son críticas en entornos industriales para el diseño, implementación y análisis de sistemas de control[26].

Este lenguaje de programación esta compuesto por los siguientes componentes [27]:

- Etapa: Un estado en el que se encuentra el sistema automatizado es representado por una etapa en GRAFCET. En un momento dado, cada etapa define una condición específica del sistema. Normalmente, se utiliza un doble cuadrado para indicar el inicio del proceso en las etapas iniciales.
- Acción asociada: Durante cada etapa, se pueden realizar una o más acciones que indican las tareas a completar mientras el sistema está en esa etapa. [26] señala que las acciones pueden ser continuas o puntuales.

  Dentro de las acciones continuas destacan:

  1. Básica: la acción se ejecuta siempre y cuando la etapa este activa.
  2. Condicional: la acción se ejecuta solo si la condición impuesta se cumple.
  3. Temporizada:la acción está sometida a una condición temporal-temporización con respecto a una etapa,retraso en la activación,retraso en la desactivación...

Las acciones puntuales son una novedad en la herramienta Graceft y cuentan con:

1. En la activación:la acción se realiza una sola vez cuando la etapa se activa.
2. En la desactivación:la acción se realiza una sola vez cuando la etapa se desactiva.
3. Al franqueo:ésta es una acción asociada al franqueo de una transición.
4. En el evento: acciones asociadas a cualquier evento puntual.

- Transición: Las condiciones que deben cumplirse para que el sistema pase de una etapa a otra se definen mediante una transición. Se evalúan estas condiciones junto con el estado actual del sistema (la etapa actual) para decidir si se debe hacer una transición. Los eventos que pueden estar incluidos en las condiciones de transición son la activación de un botón, la detección de un objeto por parte del sensor o el vencimiento del temporizador. Si se cumplen todas las condiciones de transición, el sistema avanzará a la siguiente etapa tal como está definido en el GRAFCET.

En la imagen 4.1 se puede apreciar los tres componentes fundamentales ya mencionados anteriormente sobre la programación de GRACEFT.

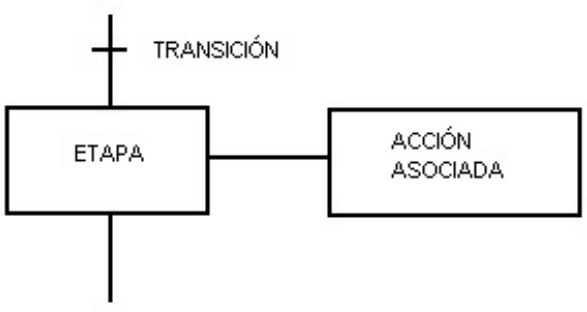


Figura 4.1: Componentes Grafcet.Fuente:[27]

4.2. Proceso con sensores de proximidad

En la imagen 4.2 se muestra la programación del primer escenario (sin sensor de visión artificial) mediante el lenguaje Graceft.

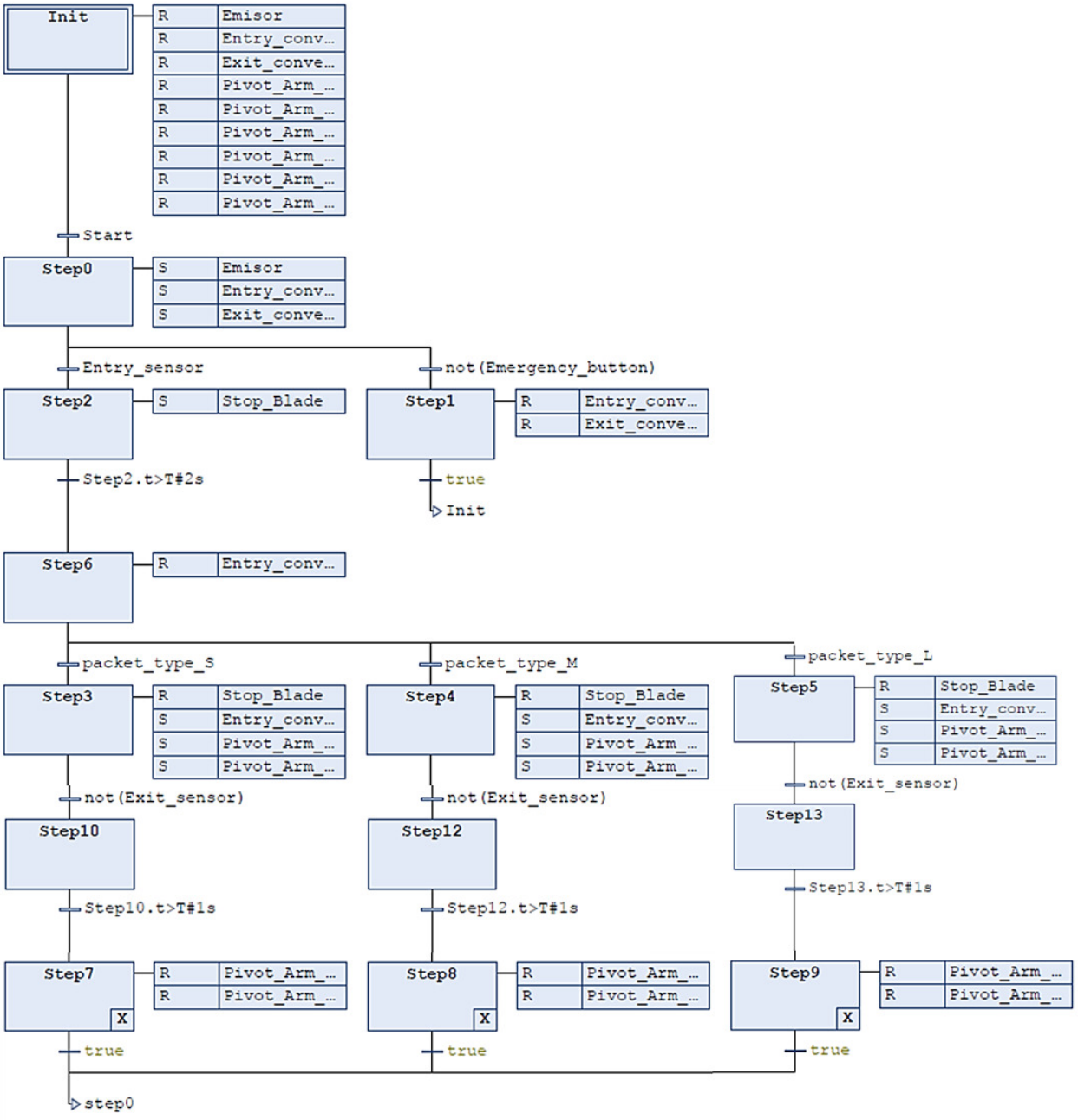


Figura 4.2: Grafcet_1.Fuente:Autor

La configuración de los contadores y la detección del tipo de paquete se han programado mediante lenguaje de texto estructurado (ST) y lenguaje de diagramas de escalera (LADDER) respectivamente:

4.2.1. Contadores

Se ha creado el bloque de funciones "counter (FB) " el cual es un contador de paquetes que toma como entrada el tipo de paquete (packet_type) y, dependiendo del valor de este, incrementa uno de los tres contadores de salida (counter1, counter2, o counter3). La función esta representada en la figura 4.3:

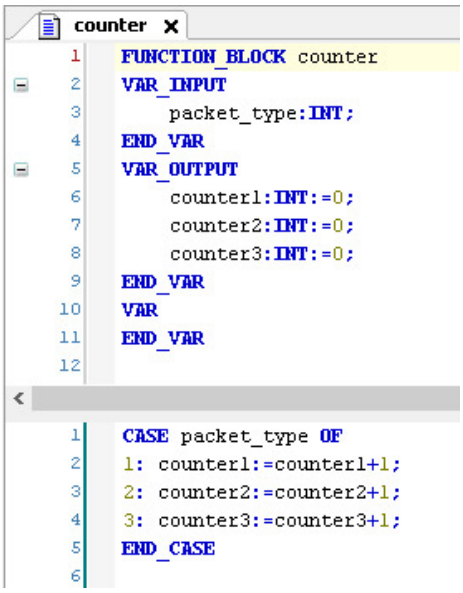


Figura 4.3: counter(FB).Fuente:Autor

La parte superior define la estructura del bloque de funciones llamado contador. Esta parte se ha creado para contar los paquetes de diferentes tipos, tiene entradas y salidas que manejan los valores y el comportamiento del contador.

1. FUNCTION_BLOCK counter: Declara el bloque de funciones llamado **counter**. Este bloque será llamado desde otros programas o funciones para ejecutar la lógica interna que cuenta los paquetes según su tipo.

2. VAR_INPUT: Define las variables que son entradas en los bloques de funciones; estos son los valores que el bloque recibe para su procesamiento.

   En esta situación, **packet_type** es un entero (INT) que representa el tipo de paquetes que se están procesando, siendo la única variable aquí. Esta variable determina qué contador incrementará su valor.

3. VAR_OUTPUT: Define las variables de salida que son devueltas por los bloques de funciones después de ejecutar su lógica:
   - **counter1**: Un contador donde se registran todos los paquetes que pertenecen al tipo 1; su valor inicial sería 0.
   - **counter2**: Cuenta los paquetes del tipo 2, también inicializado en 0.
   - **counter3**: Cuenta los paquetes del tipo 3, igualmente inicializado en 0.
4. VAR: No se han definido variables internas adicionales en este bloque (este bloque podría usarse si fueran necesarias variables internas además de las de entrada o salida).

En la parte inferior del código se encuentra la lógica del conteo, que se basa en una estructura CASE. Esta estructura toma como entrada la variable **packet_type** y dependiendo de su valor decide cual uno de los tres contadores definidos en las salidas debe ser incrementado.

- CASE packet_type OF: El bloque CASE evalúa el valor de **packet_type**. Este es el valor que indica qué tipo de paquete está siendo procesado. Los posibles valores son 1, 2, o 3, lo que corresponde a distintos tipos de paquetes.
- counter := counter + 1;: Si el valor de packet_type es 1, significa que se ha identificado un paquete del tipo 1,2 o 3 dependiendo del contador. Por ende, se aumenta el contador correspondiente en uno. Estos contadores llevan la cuenta de cuántos paquetes de cada tipo han pasado a través del sistema.

Las tres líneas representadas en la figura 4.4 invocan la función counter (FB) tres veces, cada vez con un tipo de paquete distinto (packet_type 1, 2, o 3). Cada llamada incrementa el contador correspondiente (c1, c2, c3) del programa principal, manteniendo un registro actualizado de cuántos paquetes de cada tipo han sido procesados.

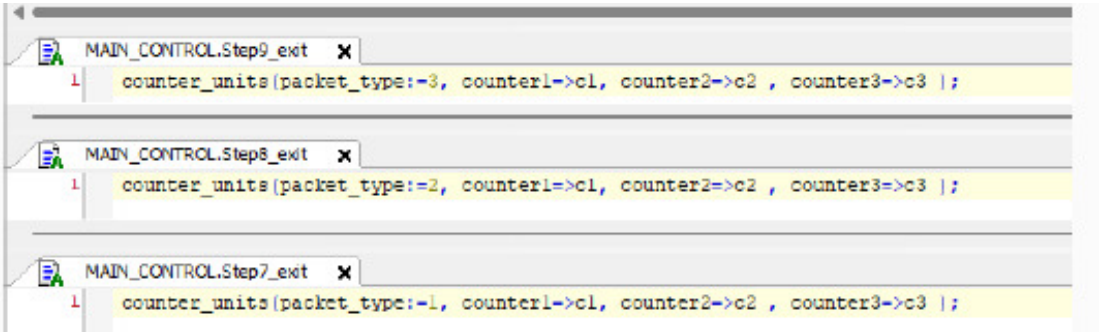


Figura 4.4: Steps counter.Fuente:Autor

4.2.2. Detección del tipo de paquete

Como se ha mencionado anteriormente la detección del tipo de paquete se basa en los valores de dos sensores "retroreflective_sensor_1" y "diffuse_sensor_1".

Para ello se ha programado cada uno de los casos dentro del **MAIN_CONTROL(PRG)** de la siguiente manera representada en la imagen 4.5:

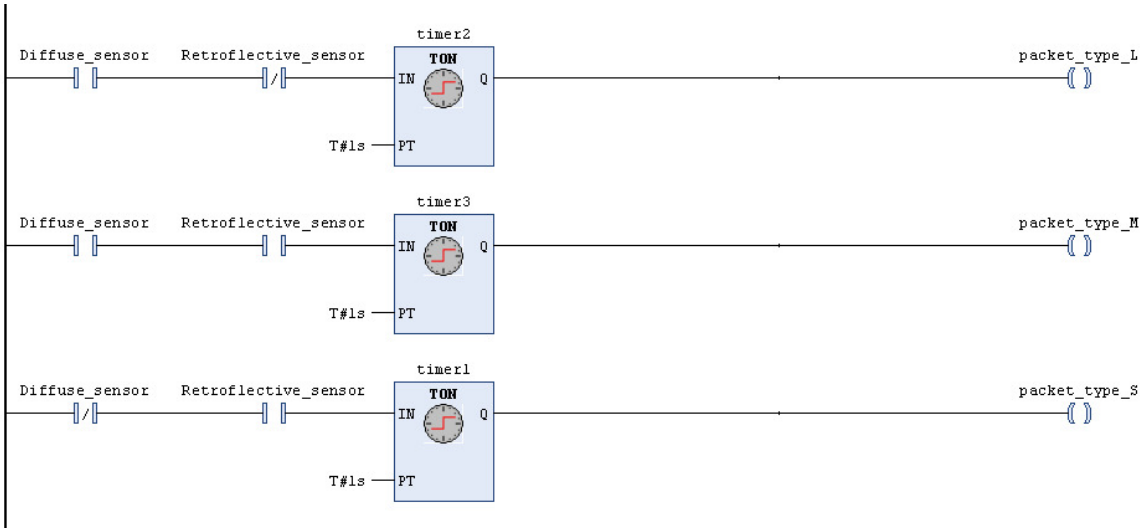


Figura 4.5: Clasificación paquetes.Fuente:Autor

Se puede observar que para cada caso mencionado anteriormente se activa la bobina correspondiente a cada tipo de paquete pasado un segundo después de la activación de los sensores.

4.3. Proceso con sensor de visión artificial

En la imagen 4.6 se muestra la programación del segundo escenario (con sensor de visión artificial) mediante el lenguaje Graceft.

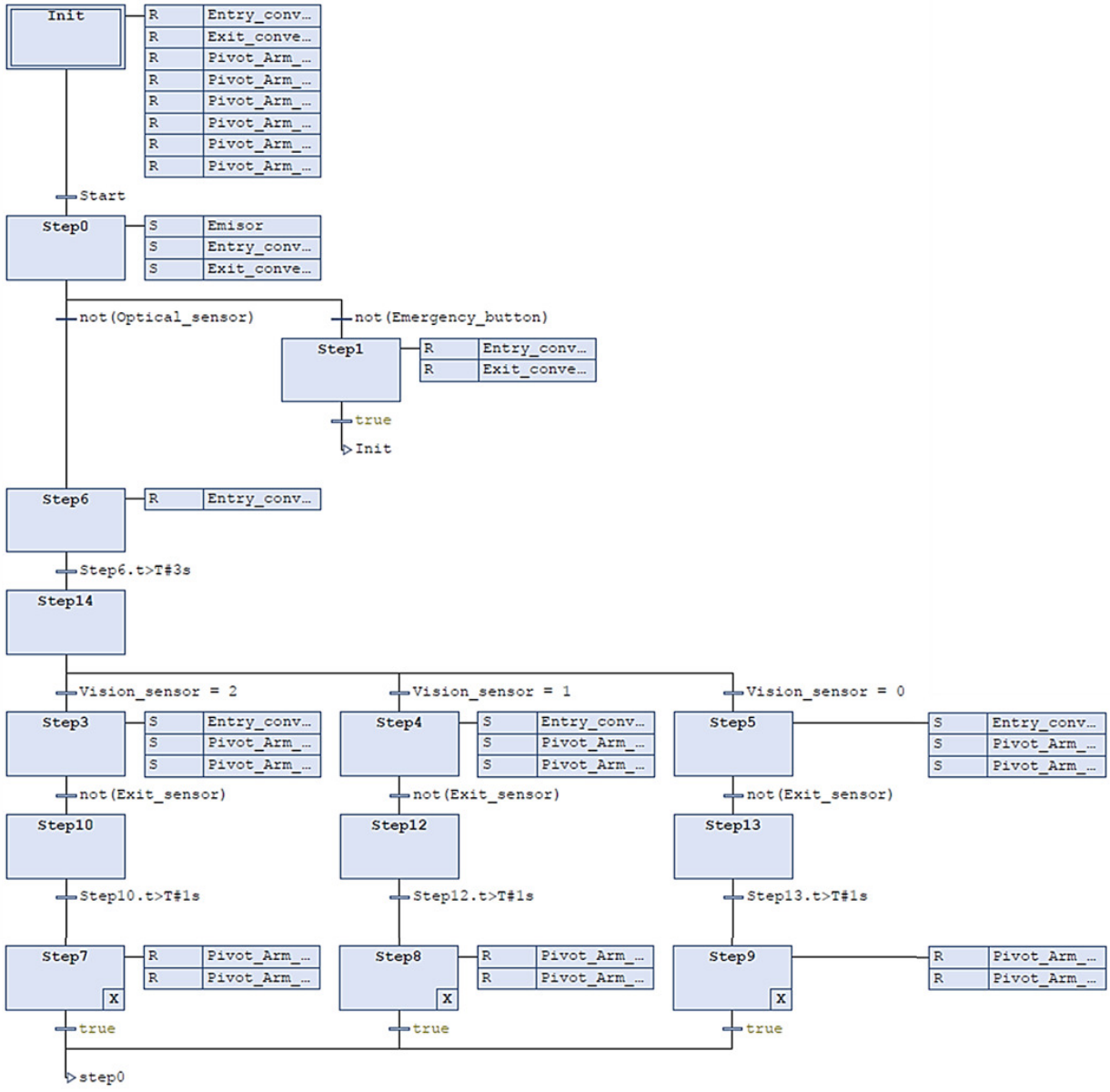


Figura 4.6: Graceft_Sensor de Visión. Fuente: Autor

La programación de los contadores se mantiene igual que en el proceso con los sensores difuso y retroflectivo.

En este caso, el uso del lenguaje ST ya no es necesario. Se sustituyen las partes del código afectadas por la rutina en lenguaje Python que se encarga de clasificar en tiempo real los paquetes en función de las imágenes recibidas desde OBS, simulando la captura de imágenes de un sensor de visión artificial ficticio.

La condición de la trifurcación se debe a los distintos valores que puede adoptar "Vision_sensor".

Capítulo 5

Entrenamiento y validación del modelo de clasificación de imágenes

Se va a exponer como ha sido el entrenamiento del modelo de clasificación de los paquetes mediante la programación en Python. Cabe destacar que para todos estos pasos ha sido necesario usar una camará fija en Factory como aparece en la figura 5.1, llamada "Prueba TFG" ya que así será mucho más fácil que el modelo no tenga errores y que tanto el entrenamiento como la validación se realicen con la mayor precisión posible.

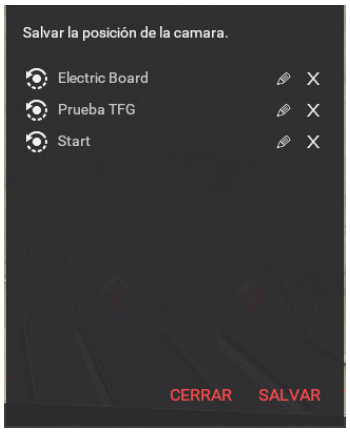


Figura 5.1: Cámara Fija:Fuente:Autor

5.1. Creación del conjunto de imágenes

Para el procesamiento de las imágenes se han creado dos funciones principales; 'Created_Mask' y 'Process_dir'.

La función 'create_mask' recibe imagen de un frame capturado, y realiza las siguientes operaciones reflejadas en código en el Anexo A:

a. Convertir la imagen original, de tamaño 1920 x 1080 al espacio de color LAB, para poder segmentar (detectar) la caja.

b. Aplicar umbrales a los canales para detectar la caja en la imagen.

c. Rellenar huecos en la máscara resultante y eliminar objetos pequeños que puedan confundirse con la caja y todo lo detectado en la parte derecha de la imagen (solo nos interesa la primera cinta).

d. Calcular las propiedades del paquete y marcarlo con un rectángulo en la imagen.

e. Recortar ese cuadrado de la imagen, pasarlo a grises y redimensionarlo a 224 x 224 pixeles (será el tamaño de entrada de nuestro modelo de clasificación).

f. Guarda el recorte para entrenar el modelo.Los recortes de los tipos de paquetes están representados en la figura 5.2.

(a) S (b) M (c) L

Figura 5.2: Imágenes de los paquetes S, M y L.

A modo más visual se ha propocionado el proceso general de una forma esquemática en la figura 5.4.

Se ha realizado un pequeño código a modo de ejemplo para que se vea visualmente lo que hace la función ”Create_Mask”. Esta representación viene reflejada en la figura

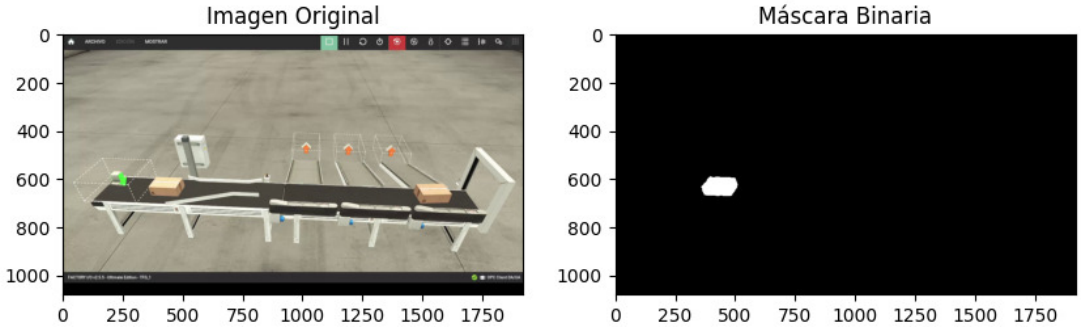


Figura 5.3: Representación de ”Create_Mask”.Fuente:Autor

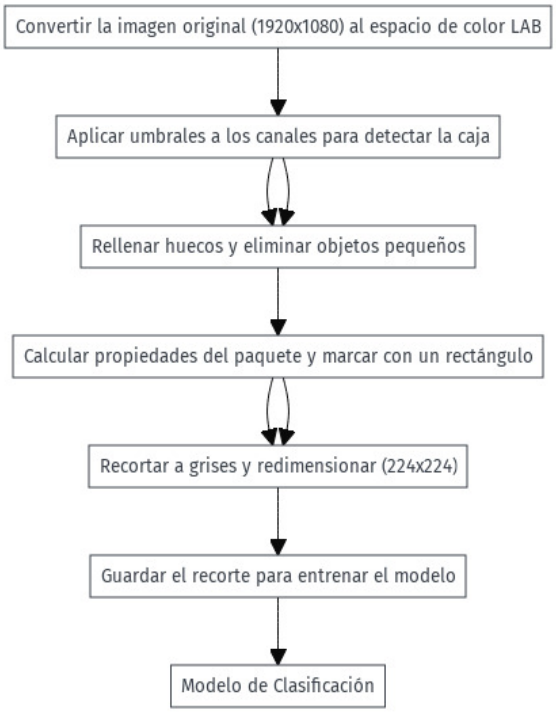


Figura 5.4: Esquema Creación de imágenes. Fuente: Autor

Una vez realizado el código anterior usamos el archivo 'Process_dir' para recorrer los frames que se han guardado para procesar cada imagen, extraer los recortes y guardarlos llamando a la función createMask.

Estos recortes será guardados en una carpeta llamada 'recortes' y serán utilizadas posteriormente para entrenar el modelo.

Para entender mejor como funciona el archivo 'Process_dir' se va a desglosar el código del Anexo B en varias partes:

a. Importar módulos necesarios:

- **os**: Este módulo se utiliza para interactuar con el sistema operativo, como comprobar si existen directorios o listar archivos.
- **nbformat**: Se utiliza para leer cuadernos Jupyter (.ipynb).

b. Especificar la ruta del notebook:
La variable "notebook_path" contiene la ruta del cuaderno donde está definida la función "create_mask" . El cuaderno se encuentra en el escritorio dentro de una carpeta llamada «Entrenamiento».

c. Leer el contenido del notebook:
El archivo del cuaderno se abre y su contenido se lee con **nbformat** mediante la función "with open () as f". Esto permite tener acceso a ejecutar el código dentro del cuaderno.

d. Ejecutar las celdas de código del notebook:

- La búsqueda de celdas con código Python (celdas de código) se produce en el código y se ejecutan una a una utilizando ”exec”.
- Si alguna celda tiene un error de sintaxis se ignora y se muestra un mensaje diciendo que hubo un error de sintaxis.
- En caso de cualquier otro tipo de error se imprime en la consola.

e. Verificar si la función ”create_mask” existe:

Después de ejecutar las celdas, el código verifica si la función ”create_mask” se cargó correctamente. Si no encuentra esa función, lanza un error indicando que no se pudo encontrar.

f. Lista de directorios con imágenes:

La variable ”directorios_imagenes” contiene tres rutas a carpetas con imágenes (las carpetas llamadas S, M y L). Estas carpetas se encuentran en un escritorio en el directorio «Entrenamiento».

g. Procesar imágenes en los directorios:

- El código comprueba primero si el directorio existe para cada directorio. En caso de que no exista, se muestra un mensaje y se procesa el siguiente directorio.
- Si el directorio existe, se listan todos los archivos, pero sólo los que realmente pertenecen a imágenes, es decir, los que terminan con extensiones como .png, .jpg o .jpeg.

h. Aplicar la función create_mask a cada imagen:

- A continuación, el código genera la ruta completa del archivo para cada imagen encontrada e intenta aplicar la función ”create_mask” pasando dos parámetros: ruta de la imagen y valor 0.
- Si la función se ejecuta correctamente imprime un mensaje que indica que la imagen ha sido procesada.
- Si se produce algún error al crear la máscara con la función ”create_mask”, imprime un mensaje de error informando de lo que ha ido mal.

5.2. Entrenamiento del modelo

Una vez se hayan descargado y almacenado los recortes en la carpeta correspondiente (la cual contiene los recortes en subcarpetas divididas según su tamaño ya sean S, M o L) se lleva a cabo el entrenamiento de una red convolucional. Con el procedimiento anterior se han obtenido 40 imágenes de cada tipo.

Este entrenamiento esta recogido en el archivo DL y el código correspondiente se ve reflejado en el Anexo C. Los pasos a realizar son los siguientes:

a. Configuración de Parámetros y Preparación de Datos:

- Se define el directorio donde están almacenadas las imágenes.
- Se establecen los parámetros característicos de las imágenes.
- Se muestra el número total de imágenes en cada carpetas y posteriormente se muestra el numero total de imágenes.
- Se usa 'ImageDataGenerator' para el procesamiento y augmentación de imágenes especificando diferentes parámetros.
- Se crea el generador de datos para el conjunto de entrenamiento y para el conjunto de validación.

b. Definición y Compilación del Modelo:

- Se construye un modelo secuencial usando los siguientes tipos de capas con un orden establecido para su desarrollo óptimo como aparece en la figura 5.5:
  a) **Convolucional**:extraen características de las imágenes. Cada capa tiene un conjunto de filtros que se aplican a las imágenes.
  b) **MaxPooling**:Se reduce la dimensión espacial de las imágenes para conservar las características más importantes.
  c) **Flatten**:Convierte las características 2D en un vector 1D.
  d) **Dropout**:Se utiliza para prevenir el sobreajuste al .ªpagar.ªleatoriamente algunas neuronas durante el entrenamiento.
  e) **Dense**:ayudan a clasificar la imagen en una de las clases.
- Se compila el modelo con un optimizador 'Adam', una función de perdida 'categorical_crossentropy'y una métrica de 'Accuracy'.

c. Entrenamiento del Modelo: En esta fase, el modelo se entrena con los datos preparados:

- **Entrenamiento**: Se ajusta el modelo usando los generadores de datos de entrenamiento y validación durante un número definido de épocas (hasta 100 en este caso).
- **Visualización del Progreso**: Se grafica la pérdida y precisión tanto para el entrenamiento como para la validación para monitorear el desempeño del modelo a lo largo del tiempo.

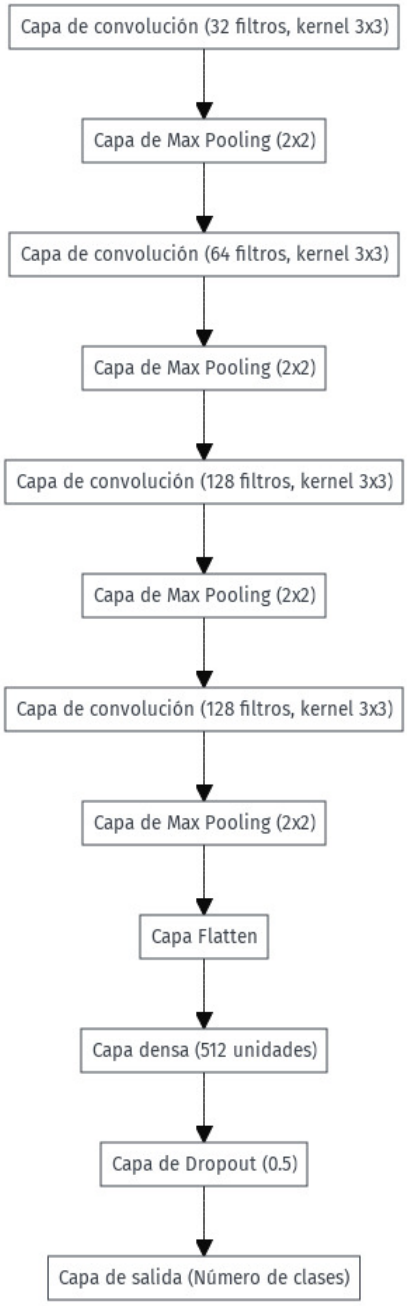


Figura 5.5: Diagrama red neuronal.Fuente: Autor

En la figura 5.6 se aprecia la evolución de la pérdida y la precisión tanto de la validación como del entrenamiento del modelo.

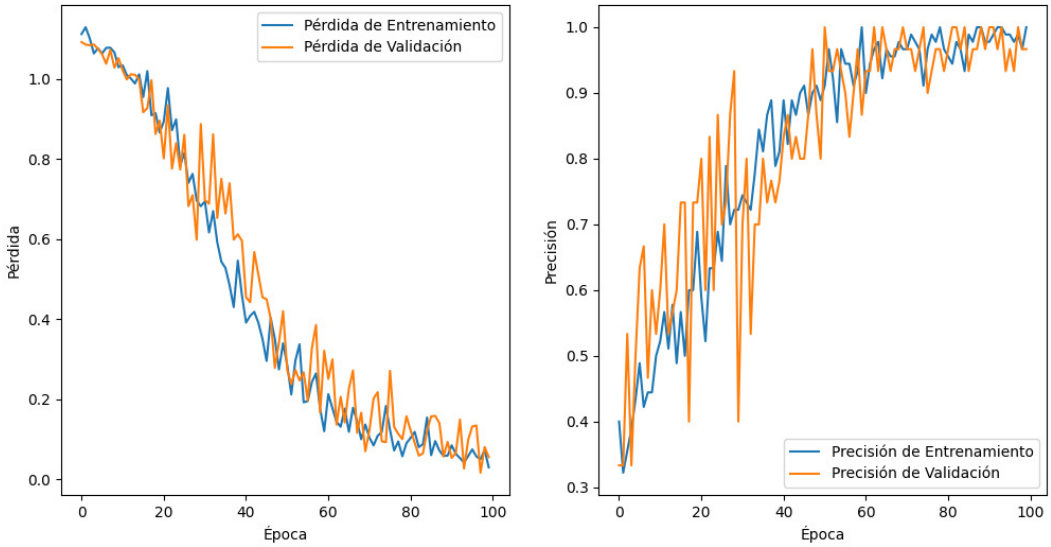


Figura 5.6: Validación y Entrenamiento. Fuente: Autor

Se han elegido 100 épocas para el entrenamiento con objeto de evitar el sobreentrenamiento. Se aprecia en la figura 5.6 como a partir de las 100 épocas, las pérdidas en el conjunto de validación aumentan, lo que supone una muestra de "overfitting" o sobreentrenamiento del modelo, indeseado por mermar la capacidad de generalización del sistema.

d. Evaluación y Predicción: Finalmente, el modelo se evalúa y se realiza una predicción:

- **Evaluación**: Se evalúa el rendimiento del modelo en el conjunto de entrenamiento y de validación, calculando la precisión de cada uno de ellos.

  Esta evaluación ha dado como resultado una precisión del 100 % tanto en el conjunto de entrenamiento como en el conjunto de validación como se muestra en la imagen 5.7.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 830ms/step - accuracy: 1.0000 - loss: 0.0779
Precisión en el conjunto de validación: 100.00%
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 604ms/step - accuracy: 1.0000 - loss: 0.0221
Precisión en el conjunto de entrenamiento: 100.00%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 172ms/step
```

Figura 5.7: Resultados. Fuente:Autor

- **Predicción**: realiza una predicción sobre una imagen nueva, cargándola, preprocesándola y usando el modelo para predecir su clase.

5.3. Estrategia de validación

La estrategia de validación es una metodología usada en redes neuronales para evaluar el desempeño de un modelo durante el entrenamiento. Hay varias estrategias de validación como:

- Validación Cruzada (Cross-Validation): Aquí, el conjunto de datos se divide en varios subconjuntos o "pliegues". El modelo se entrena en algunos de estos pliegues y se rota para validar los otros, asegurando que cada pliegue se utilice tanto para entrenamiento como para validación.
- Validación con un conjunto de validación separado:Hay tres particiones del conjunto de datos: el conjunto de entrenamiento (train), el conjunto de validación (validation) y el conjunto de prueba (test). El conjunto de validación se utiliza para ajustar los hiperparámetros del modelo, mientras que el conjunto de prueba se utiliza para evaluar el rendimiento final.

Pero en este caso la estrategia de validación utilizada es:

- División Train/Test: Se considera en la mayoría de los casos como una forma rápida y sencilla de evaluar modelos; es fácil de ejecutar y requiere menos tiempo de cálculo en comparación con la validación cruzada, lo que la hace especialmente útil para modelos complejos o conjuntos de datos grandes. En este caso, se ha utilizado un 75 % de los datos para el entrenamiento y un 25 % para la validación. Además, al mantener los conjuntos completamente separados, este enfoque permite desarrollar una comprensión clara de cómo el modelo se generaliza en datos no vistos, evitando el sobreajuste durante la validación. Por lo tanto, es útil cuando se necesita una estimación rápida del rendimiento, especialmente en las primeras fases del proceso de desarrollo del modelo.

5.4. Métricas de rendimiento

Las métricas de rendimiento son medidas utilizadas para cuantificar qué tan bien se está desempeñando el modelo. Estas pueden variar según el problema en cuestión (clasificación, regresión, etc.). Algunas métricas comunes son:

- Exactitud (Accuracy): Es la proporción del número de predicciones correctas con respecto al número total de predicciones realizadas. Resulta útil en problemas de clasificación donde las clases están equilibradas.
- Precisión (Precision):Se refiere a la cantidad de verdaderos positivos frente al total de predicciones positivas efectuadas. Esta medida resulta crucial cuando el costo asociado a un falso positivo es alto.
- Recall (Sensibilidad o Tasa de Verdaderos Positivos):se puede definir como la razón de verdaderos positivos con respecto a la suma de los verdaderos positivos y falsos negativos. Es clave en la solución de problemas que envuelven una alta penalización si se deja pasar una instancia positiva (falsos negativos).
- Error Cuadrático Medio (MSE, Mean Squared Error):Promedio de los cuadrados de los errores de predicción. Es una métrica común en problemas de regresión.

En este caso se ha estimado oportuno el uso de la mérica de ‘Accuracy’ como bien se mencionó anteriormente en la parte de entrenamiento, debido a que la situación es referida a un problema de clasificación multiclase.

5.5. Verificación del modelo resultante

En el archivo 'Rutina modelo_entrenado.h5' se realiza la verificación de si el modelo cumple con las expectativas esperadas.Este archivo de programación viene mostrado en el Anexo D y se va a explicar a continuación:

Primeramente se ha llevado a cabo un flujo general que sigue la siguiente secuencia:

a. Carga una imagen de un paquete:Mediante la función "cargar_imagen(image_path)" se carga una imagen y se convierte a un array numpy.Se trabaja con una imagen en formato RGB.

b. Convierte la imagen a un espacio de color LAB:La imagen cargada en RGB se convierte al espacio de color LAB mediante la función "color.rgb2lab(RGB)", que separa la luminosidad (I) y los valores cromáticos (A y B). Este espacio es útil para segmentación de color.

c. Genera una máscara binaria:Haciendo uso de la función definida "crear_mascara" se crea una máscara binaria comparando los valores de los canales LAB contra umbrales predefinidos para cada canal. Solo los píxeles cuyos valores estén dentro de esos rangos serán considerados "verdaderos" (1), mientras que el resto serán "falsos" (0).

d. Procesa la máscara:Se procesa la máscara binaria para cerrar agujeros pequeños en la máscara usando operaciones morfológicas("cv2.morphologyEx", tanto "OPEN" como "CLOSE"), eliminar objetos pequeños que no sean relevantes("remove_small_objects", menos de 500 pixeles en este caso) y eliminar la mitad derecha de la imagen, manteniendo solo la mitad izquierda como región de interés("BW_clean[:, cols // 2:] = 0").

e. Recorta la imagen:Se identifican las regiones conectadas en la máscara y se calcula una Bounding Box (mediante "prop.bbox") que encierra al objeto detectado. La imagen se recorta según esta caja("croppedImg = RGB[bb[0]:bb[2], bb[1]:bb[3]]"). Luego, la imagen se convierte a escala de grises ("cv2.cvtColor") y se redimensiona a 224x224 píxeles para que sea compatible con el modelo de clasificación("cv2.resize").

f. Preprocesa la imagen:La imagen recortada se duplica en 3 canales para simular una imagen RGB, ya que el modelo espera una imagen con 3 canales("np.stack([image_np] * 3, axis=-1)"). Luego se expande una dimensión adicional para simular un batch de tamaño 1("np.expand_dims(image_np_rgb, axis=0)").

g. Carga del Modelo Preentrenado:Se carga un modelo preentrenado en formato .h5 que se utilizará para la clasificación del paquete("tf.keras.models.load_model(model_path)").

h. Clasificación del Paquete:Se utiliza el modelo para predecir la clase a la que pertenece la imagen del paquete. La predicción se realiza usando el método "predict()" de TensorFlow y se retorna la clase con mayor probabilidad:

i. Clasificación General:Si se detecta y recorta correctamente el paquete, la imagen procesada se envía al modelo para su clasificación. Si no se detecta ninguna región válida (por ejemplo, si la máscara no tiene objetos), el programa clasifica el paquete como tipo 3.

El punto de entrada al programa es la funcion "main()". En ella se especifica la ruta donde esta guardado el modelo entrenado y la ruta de la imagen que se va a usar de prueba para clasificar.Posteriormente si sigue el flujo indicado anteriormente.

En conclusión esta función carga un modelo preentrenado,carga una imagen la cual va a ser preporcesada y clasifica el tipo de paquete que aparece en la imagen devolviendo el tipo de paquete del que se trata.

Capítulo 6

Proceso mejorado con Sensor de Visión Artificial

En apartados anteriores se ha comentado como configurar correctamente los software Factory IO y Codesys, y como funciona de manera general el proceso de clasificación con el sensr de visión artificial. En este apartado se van a desgolsar más aspectos importantes para explicar el funcionamiento de este proceso.

Se ofrece a continuación la imagen 6.1, que proporciona una representación visual de las conexiones entre los distintos softwares, facilitando así su comprensión.

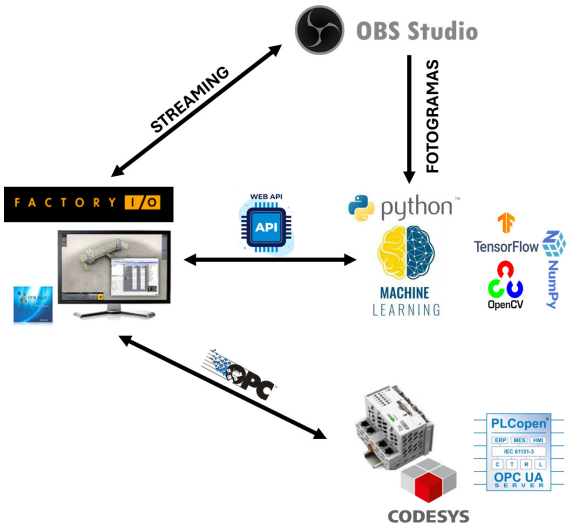


Figura 6.1: Conexiones entre software. Fuente: Autor

6.1. Configuración OBS

Una ves se ejecute OBS aparecerá el menú principal del software.Es necesario crear una captura de ventana que capture el escenario de Factory IO y así poder iniciar la cámara virtual.Los pasos que se seguirán serán los siguientes:

a. En la sección de escena crear una nueva escena mediante el botón "+" y nombrarla (en este caso ha sido nombrada como "Prueba 1")6.2.

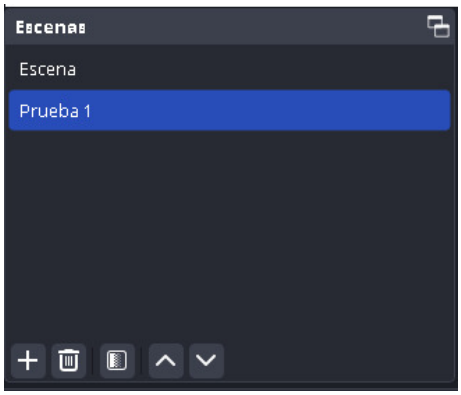


Figura 6.2: Escena OBS.Fuente:Autor

b. Una vez creada la escena es necesario añadir una fuente de captura de pantalla. Para ello se presiona en el botón "+"en la sección de "Fuente" y se selecciona "Captura de ventana".Le asignamos un nombre (en este caso se llama "Captura de ventana")6.3.

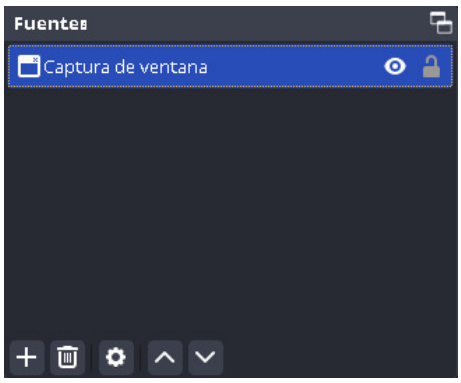


Figura 6.3: Fuente OBS:Fuente Autor

c. Posteriormente se designa la fuente que vamos a utilizar y configuramos las propiedades en el boton ”Propiedades” como aparece en la figura 6.4.

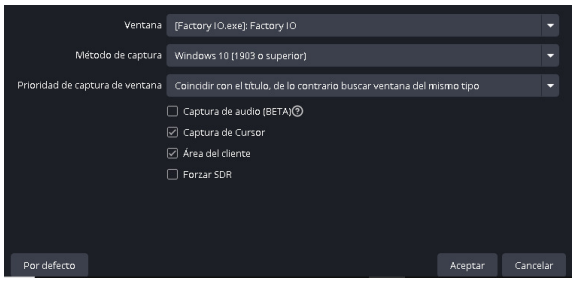


Figura 6.4: Propiedades OBS.Fuente:Autor

d. Se configura la sección de transiciones de escena como se muestra en la figura 6.5.

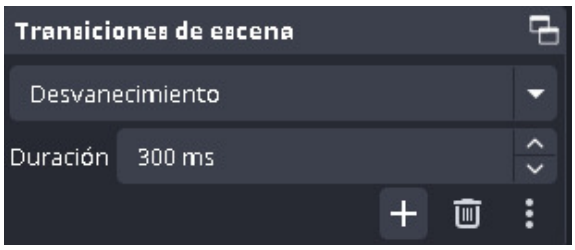


Figura 6.5: Transiciones OBS.Fuente:Autor

e. FInalmente en la sección de controles de OBS se selecciona la opción de ”Iniciar cámara virtual” como el la imagen 6.6.

Figura 6.6: Controle OBS:Fuente:Autor

6.2. ID del Sensor

Para el posterior programa que se encargará de detectar y clasificar los paquetes en tiempo real, es necesario hacer uso del sensor de visión (Vision sensor) que está “escondido” en la escena. Este sensor recibirá los valores suministrados en tiempo real por el modelo de aprendizaje profundo entrenado, lo que le permitirá tomar decisiones pertinentes en la clasificación dentro de Factory

IO. Estos valores serán enviados desde el programa Python a Factory IO empleando una API Web disponible en el simulador de procesos. Esta API permite leer y escribir valores de los sensores y actuadores del simulador, facilitando así la integración y el control en tiempo real.

FACTORY I/O (Running)	ff50b1f0-8822-4da8-9aa0-494295588138	496	Bit	Input	False
FACTORY I/O (Paused)	da924a2d-62e9-4c80-baa0-fe0ad78c1557	497	Bit	Input	False
FACTORY I/O (Reset)	e4505270-488d-4b63-bc85-b2b7818c08c9	498	Bit	Input	False
FACTORY I/O (Run)	e47c3d07-4e02-4216-8207-916cf001cf46	496	Bit	Output	False
FACTORY I/O (Pause)	9f9d0304-5339-492a-90ce-0301b8e373b9	497	Bit	Output	False
FACTORY I/O (Reset)	b6b4ce36-373d-415b-b4ed-03d8a30760ea	498	Bit	Output	False
Exit conveyor	72cbb310-9191-44ca-8d6a-d34b71e13556	2	Bit	Output	True
Emitter	49587079-3269-4e9e-b48f-0c92c360bdf5	0	Bit	Output	True
Sorter 1 belt	321e070a-5476-46a8-bf74-204e430df1de	5	Bit	Output	False
-	f370e574-a3af-463b-a88f-b957466015af	9	Bit	Output	False
Sorter 1 turn	6041e284-03dd-4433-a187-d99071807933	4	Bit	Output	False
Entry conveyor	0252a684-9e02-4099-b08c-43f7fe464e57	1	Bit	Output	True
Sorter 2 turn	8284e907-1eac-44ba-beca-9f300647a5d1	7	Bit	Output	False
-	e68d415b-c089-458d-b108-7aabfb67da13	12	Bit	Output	False
-	95179229-1d09-4c17-897d-010c54eb99ee	6	Bit	Output	False
Sorter 3 turn	50a69e5d-b014-433a-9e9d-e4ec1ac31bd9	10	Bit	Output	False
Sorter 3 belt	62eb2b6e-832a-4ed9-9851-cdf40b157b93	11	Bit	Output	False
Sorter 2 belt	d8c2d4a3-2115-404f-8f0a-2db9eda477a2	8	Bit	Output	False
Exit Sensor	23b13d6e-b68a-4cd9-9981-27bb42daa521	2	Bit	Input	True
Remover 1	05680a85-7afe-4934-87a8-dbcb79522eca	13	Bit	Output	True
Stop	50c9cdee-d88f-4da1-ac60-235c974d1c0e	3	Bit	Input	True
Remover 2	11d68032-6a40-45aa-a5a8-861d39870ad5	14	Bit	Output	True
Emergency stop	5d8020c0-3870-4f05-8315-f160a9e47854	0	Bit	Input	True
Start light	1e8fce83-b1cc-45d1-9b54-d760f7cdd2f2	16	Bit	Output	False
Start	69290690-1b53-4039-b488-88a22aa44e7b	1	Bit	Input	False
Remover 3	b591a9ef-6e04-4e64-8c8b-66f084add929	15	Bit	Output	True
Stop light	9401e17c-d971-4357-abd0-5690ee8a4c33	17	Bit	Output	False
Manual	01058105-1fb7-4daa-a1ff-8781f73453d4	5	Bit	Input	True
Auto	b576e297-24a0-4f9b-a691-236c2019d6ae	6	Bit	Input	False
Optical Sensor	7989882f-96ab-45ad-a67f-bb5cbb1c8d90	4	Bit	Input	True
FACTORY I/O (Time Scale)	69e4f019-cb25-413d-ad53-920d2e0660d0	240	Float	Input	0
FACTORY I/O (Camera Position)	1ffb5ace-5272-4771-a1e2-8d7efc96ac32	240	Int	Output	0
Counter 1	47ea5311-9a0f-4548-ba0e-bc998a54b967	0	Int	Output	0
Counter 2	c586e831-f400-4dc1-b204-1fe189df9b4c	1	Int	Output	0
Vision sensor	998d6a68-bc8f-4f1c-bc10-77301c84177f	0	Int	Input	0
Counter 3	017cd76b-9cf0-4155-a694-fb341f3a52d8	2	Int	Output	0

Figura 6.7: Lista Componentes Factory IO.Fuente:Autor

Para poder forzar valores desde Python a este sensor mediante la API Web es necesario conocer la ID del sensor.Para ello se ha realizado un programa expuesto en el Anexo E en python que localiza las ID de todos los componentes de la escena y así poder obtener también el ID del propio sensor de vision como aparece en la imagen 6.7.

El programa funciona de la siguiente forma:

a. Solicitud a la API:Usa el comando ”requests.get(url)” para enviar una solicitud a la api cuya url se especifica en el código. Esta url contiene el puerto 7410 que como se mencionó con anterioridad es el que viene por defecto en Factory IO.

b. Convertir la respuesta a formato JSON:Mediante el comando ”response.json()” se convierte la respuesta HTTP (que se espera esté en formato JSON) a un objeto Python.

c. Mostrar los datos en columnas: Usando un bucle for se recorre cada ”etiqueta” (tag) de la lista. Para cada ”etiqueta”, verifica si es un diccionario con la función ”isinstance(tag, dict))”. Mediante ”tag.get('clave', 'N/A')” se obtiene el valor de las claves name, id, address...

6.3. Resultados Finales

Una vez establecida la cámara ”Prueba TFG” en Factory IO, configurados los softwares Codesys,Factory IO y OBS como se ha indicado se procederá a activar el código encargado que

capturar los frames en tiempo real de la escena, procesarlos,realizar la clasificación de los paquetes en el programa y enviar los datos que vaya capturando de la clasificación al sensor "Visión sensor" para que este desvíe el paquete al ramal oportuno.

Este código esta recogido en el Anexo F y se va a divir la explicación en dos partes principales:

6.3.1. Definición de funciones

Primeramente vamos a describir las funciones definidas que serán usadas posteriormente en la función principal. La mayoría de estas ya se han explicado a groso modo en el apartado de Çomprobación de entrenamiento".

- cargar_imagen(image_path): Abre la imagen desde la ruta dada y la convierte en un array de Numpy, devolviendo solo tres canales RGB, es decir, rojo, verde y azul, ignorando otros como el canal alfa si está presente.

- crear_mascara(I): La imagen se convierte al espacio de color LAB (que es luminancia y dos canales de crominancia), y luego se aplican umbrales en cada canal por separado. Esto proporciona una segmentación de la imagen, donde solo se seleccionan los píxeles que cumplen las condiciones dadas, y se crea una imagen binaria (fondo = 0, objeto = 1) que muestra las regiones de interés, como un paquete.

- procesar_mascara(BW): Se realizan operaciones morfológicas (dilatar y erosionar la imagen) para rellenar agujeros y eliminar el ruido en la máscara. Los objetos pequeños que probablemente sean ruido en la imagen se eliminan, y la mitad derecha de la imagen se ajusta a cero para concentrarse en un área específica, probablemente en la parte izquierda, donde es probable que esté situado un paquete, por ejemplo.

- recortar(RGB, BW_clean): Localiza el área de enfoque presente en la imagen utilizando los componentes conectados de la máscara limpia. Extrae el área de interés (caja delimitadora) y córtala de la imagen fuente. Convierte la imagen recortada a escala de grises y redimensiónala a 224x224 píxeles, que es el tamaño de imagen requerido para que el modelo de clasificación la procese.

- preparar_entrada_modelo(croppedImgResized): Convierte la imagen recortada en un array de números decimales (floats) y duplica el canal de escala de grises tres veces para crear una imagen de color (RGB) con tres canales, tal como lo requiere el modelo. Además, agrega una dimensión extra al array para crear un lote de tamaño 1, que es el formato esperado por TensorFlow.

- cargar_modelo_h5(model_path): Carga el modelo preentrenado en formato .h5, que es el formato estándar para modelos de Keras/TensorFlow.

- realizar_prediccion(model, image_np): Emplea el modelo para estimar la clase de la imagen procesada. El modelo proporciona un vector de probabilidades para cada clase, y se elige la clase con la mayor probabilidad.

- clasificar(croppedImgResized, model_path): Clasifica el paquete utilizando el modelo cargado. Si no se detecta paquete, se clasifica como "tipo 3" (indicando la ausencia de un paquete).

- enviar_valor_factory_io(sensor_id, value_to_force): Utiliza una API para comunicar el resultado de la clasificación a un sensor virtual en Factory IO, el cual en este caso será el "Vision sensor". Se envía el valor de clasificación mediante una solicitud HTTP (PUT) a una API local teniendo como parámetros de entrada el ID del sensor y el valor a forzar.

  En términos generales sigue el siguiente proceso:

  a. Se define la URL del API y se prepara una solicitud JSON.
  b. Se utiliza "requests.put" para enviar el valor al sensor.
  c. Maneja errores de conexión y respuesta no exitosa.

- leer_valor_sensor(nombre_sensor): Obtiene el valor actual de un sensor específico en Factory IO. En este caso el valor que va a obtener será el de Optical sensor. Y el proceso en términos generales es:

  a. Se define la URL del API para leer los valores de sensores.
  b. Se realiza una solicitud GET que incluye el nombre del sensor.
  c. Si la respuesta es exitosa, extrae el valor del sensor de la respuesta JSON.
  d. Maneja errores de conexión y situaciones en las que no se encuentra el valor del sensor.

6.3.2. Función principal (main())

La función principal sigue el siguiente flujo:

a. Se establecen las rutas tanto del modelo como de la imagen. La ruta de la imagen o "ruta de grabación" será la ruta donde se irán sobreescribiendo los frames que se vayan detectando.

b. Se inicia la captura de video de la cámara virtual de OBS que en este caso el índice es 1. Este índice puede cambiar dependiendo del dispositivo utilizado.

c. Se define el intervalo de espera entre iteraciones (en este caso es de 1 segundo).

d. Se define el ID del sensor al que se enviará el valor (vision sensor) y el nombre del sensor al cual se le leerá el valor (Optical Sensor).

e. Se envía un valor inicial al sensor de vision de Factory para que comience el bucle con el valor 3,es decir, con ausencia de paquete mediante la función "enviar_valor_factory_io".

f. Se establece un bucle que se ejecuta hasta 1000 veces, donde se realizan las siguientes acciones en cada iteración:

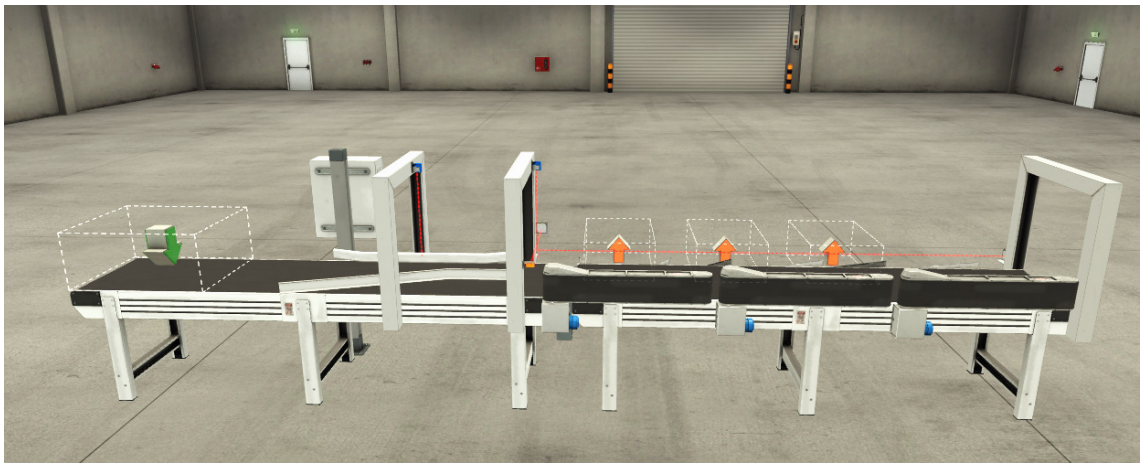
   - Establece la resolución de la cámara
   - Lee el valor del sensor óptico (Optical sensor). Si el sensor esta en "False" continua y procesa la imagen.
   - Se procesa la imagen de la misma forma que se ha comentado ya en apartados anteriores mediante ese flujo de procesamiento de la imagen y usando las funciones definidas anteriormente.
   - Se llama a clasificar para realizar la clasificación de esa imagen según el tipo de paquete.
   - Mediante la función "enviar_valor_factory_io" se envia el valor clasificado al sensor de Factory IO .

- Realiza una pausa de 1 segundo entre iteraciones mediante la función ” time.sleep(intervalo)”. Donde intervalo se ha definido antes como ya se comentó.

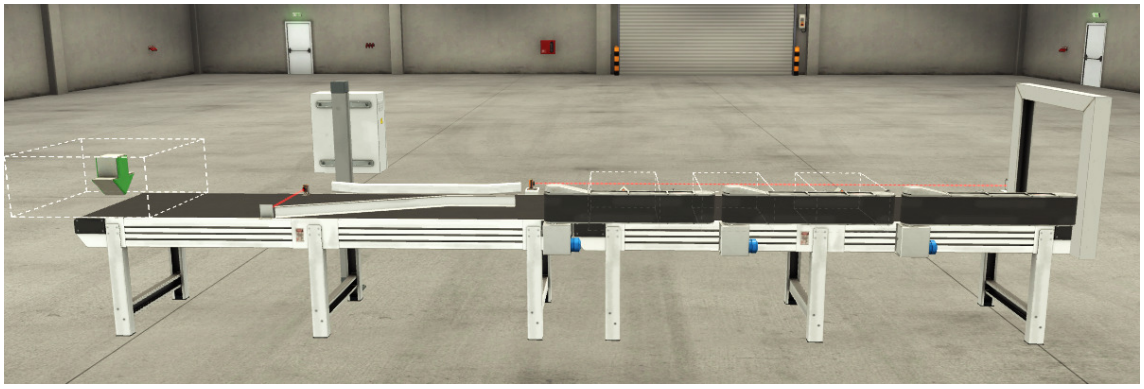
g. Una vez salga del bucle se libera la captura de video y se restaura el valor del sensor al finalizar.

En resumen esta función se encarga de gestionar el flujo general de la aplicación, asegurando que las imágenes sean capturadas, procesadas, clasificadas y enviadas a Factory IO de manera continua.

En las figuras 6.8 y 6.9 se puede apreciar la reducción en el escenario B de sensores y estrcuturas.

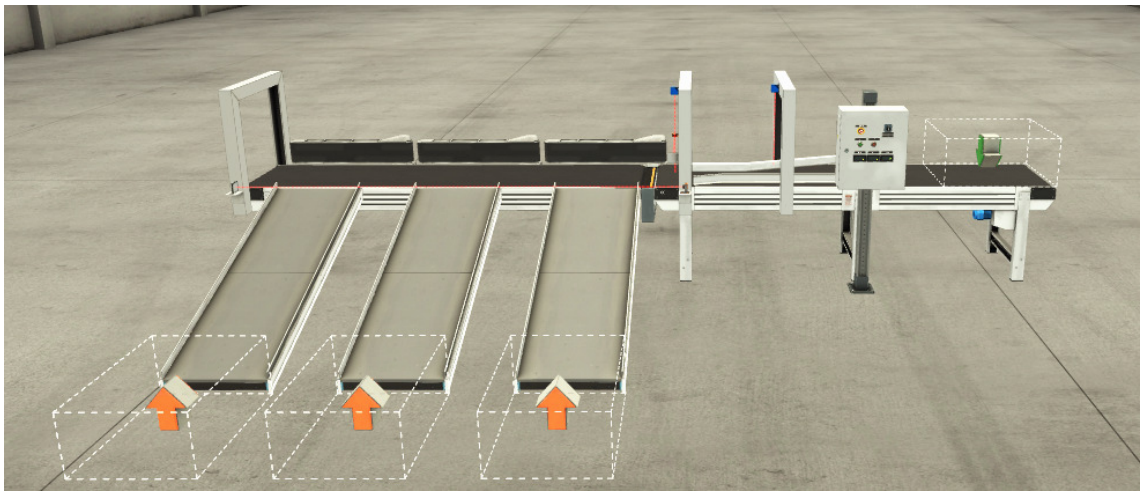


Escena 1 A)

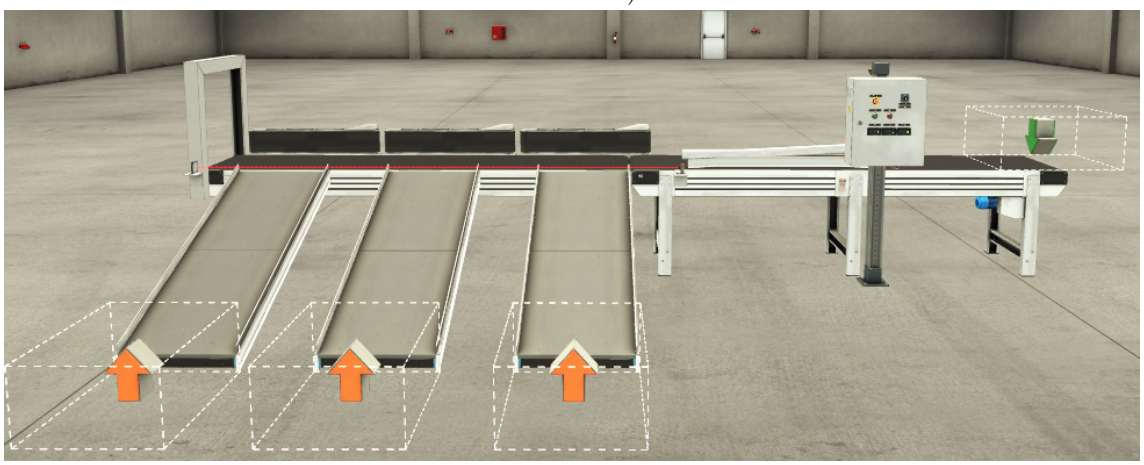


Escena 2 A)

Figura 6.8: Comparación de Escena 1 A) y Escena 2 A)



Escena 1 B)



Escena 2 B)

Figura 6.9: Comparación de Escena 1 B) y Escena 2 B)

Capítulo 7

Planificación y Presupuesto

7.1. Planificación

La siguiente propuesta describe las etapas y actividades imprescindibles para el desarrollo del presente proyecto "Sensor de visión artificial para la mejora de la automatización de procesos industriales en el simulador Factory IO". La duración total estimada es de 18 semanas y 300 horas de trabajo.

Fase	**Descripción**	**Duración (semanas)**	**Horas estimadas**
1. Estudios de antecedentes	Investigación sobre los sensores de visión artificial, el simulador Factory IO, y las tecnologías asociadas a la automatización industrial.	2	40
2. Diseño de escenarios	Configuración y diseño del entorno de simulación, elección de procesos industriales y componentes necesarios.	1	20
3. Automatización del proceso 1	Implementación del primer proceso con sensores convencionales. Programación y configuración en Factory IO.	3	50
4. Entrenamiento de los modelos de IA	Obtención de datos sintéticos y entrenamiento de los modelos. Evaluación y ajuste de parámetros.	4	60
5. Programación del modelo final	Integración del modelo entrenado en el simulador y ajuste de conexiones con el sensor virtual.	2	30
6. Automatización del proceso 2	Implementación del segundo proceso con el sensor de visión artificial. Comparación con sensores convencionales.	2	40
7. Integración de resultados	Comparación de los resultados de ambos sistemas. Ajuste y optimización de la solución final.	1	20
8. Redacción de la memoria	Elaboración del informe final, documentación técnica y análisis de resultados.	3	40
Total		**18 semanas**	**300 horas**

Cuadro 7.1: Planificación del proyecto

7.1.1. Diagrama de Gantt

Esta herramienta será usada en nuestro proyecto para planificar, controlar y gestionar todas las fases del proyecto de manera visual y organizada, garantizando que se cumplan los plazos y se utilicen eficientemente los recursos disponibles.En la figura 7.1 se muetsra el diagrama de Gantt.

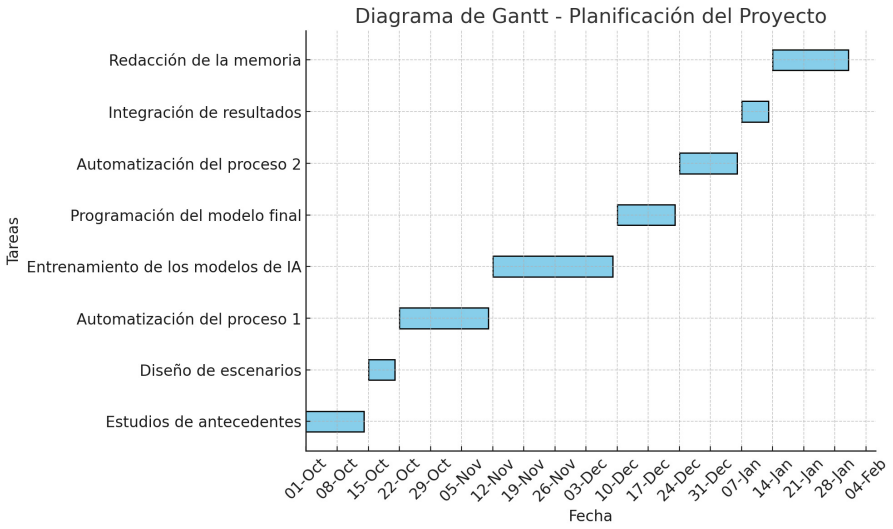


Figura 7.1: Diagrama de Gantt.Fuente:Autor

7.2. Presupuesto

El presupuesto del proyecto incluye el coste de los materiales, software y las horas de trabajo necesarias para completar las fases descritas.

Concepto	**Descripción**	**Cantidad**	**Precio unitario (€)**	**Total (€)**
Hardware				
Ordenador	PC con capacidad para ejecutar simulaciones y entrenar modelos de IA	1	1.500 €	1.500 €
Software				
Licencia de Factory IO	Simulador de procesos industriales	1	300 €	300 €
Licencia de CODESYS	Software de programación de autómatas	1	200 €	200 €
Licencia de Deep Learning	Software para el entrenamiento y evaluación de los modelos	1	200 €	200 €
Mano de obra	300 horas de trabajo a 30 €/h	300	30 €	9.000 €
Total general				**11.200 €**

Cuadro 7.2: Presupuesto del proyecto

Capítulo 8

Conclusiones y mejoras

El análisis de estas conclusiones destacaun avance notable en el proceso de incorporación de u sensor de aprendizaje profundo para la visión artificial en el entorno de simulación Factory IO. Este método supone un avance respecto a la tecnología tradicional, permite la localización y clasificación de objetos en el espacio tridimensional. Los resultados de la simulación realizada han demostrado ser válidos, por lo que la implantación de este sensor estaba justificada, ya que podía discernir las características esenciales de los objetos objeto de reconocimiento en tiempo comprimido, mejorando así los procesos industriales.

Ambos procesos, el que funciona con sensores clásicos y el que utiliza un sensor de visión, han demostrado un buen rendimiento de clasificación. Sin embargo, al tratar de captar el proceso 2 (el que utiliza visión artificial), es evidente que este proceso es mucho más lento. Esto se debe a que en él intervienen varias aplicaciones, que aumentan la carga de trabajo del sistema y lo ralentizan.

Una mejora que se podría haber implementado es la utilización exclusiva del sensor de visión, no siendo necesario el empleo del sensor óptico. Además, podría intentarse la clasificación de los paquetes sin necesidad de detener la cinta. Sin embargo, por cuestiones computacionales, el rendimiento de mi equipo informático no ha facilitado lograr este objetivo.

Los dos aspectos beneficiosos del sistema de visión artificial son su flexibilidad y adaptabilidad, ya que el sistema puede diseñarse para ajustarse a distintos casos u objetos de la fábrica simulada. No obstante, aunque los resultados del proyecto han sido positivos, hay que tener en cuenta sus limitaciones. La simulación de IO de fábrica no recrea por completo el funcionamiento del entorno realista; por ejemplo, aspectos como la iluminación, la degradación de las piezas y las diferencias entre objetos influirían en el funcionamiento del sistema en la práctica. Por tanto, sería razonable realizar esas pruebas en condiciones reales para comprobar la eficacia de la solución ofrecida.

Con relación a las propuestas de mejora, se sugiere mejorar la forma en que se enseñan los sensores de visión artificial. Aporta buenos resultados, pero aun así, incluir conjuntos de entrenamiento adicionales y un ajuste más exhaustivo de varios hiperparámetros quizá podría ayudar a aumentar la eficacia de la detección, localización y clasificación de objetos en el espacio.

También sería muy beneficioso que el sistema pudiera identificar paquetes en cualquier ángulo respecto a la cámara. Esto aumentaría las capacidades funcionales del sistema, haciéndolo más útil en diferentes áreas operativas sin tener que realizar cambios constantes en el posicionamiento

del dispositivo. También se podría pensar en mejorar las capacidades del sensor para que pueda realizar tareas más complejas, como la inspección de defectos en los productos o la vigilancia de movimientos, aunque el software Factory IO no permite esta operativa en sus versiones actuales.

En síntesis, se han alcanzado todos los objetivos propuestos en la concepción de este trabajo fin de grado:

a. Se ha realizado una revisión de los aspectos clave relacionados con la visión artificial, los sensores industriales, la inteligencia artificial y más concretamente el aprendizaje profundo, y las tendencias en automatización industrial, conducentes a la actual industria 4.0.

b. Se ha descrito el proceso industrial seleccionado y se ha presentado la selección de hardware y software adecuado para la automatización de este y para el sistema de visión artificial.

c. Se ha automatizado el proceso propuesto en sus dos variantes, empleando la metodología GRAFCET y el entorno de programación CODESYS, completamente compatible con el estándar IEC 61131-3.

d. Se ha desarrollado e integrado un sensor de visión artificial virtual en Factory IO. Para ello, se ha generado un conjunto de imágenes virtuales para el entrenamiento de un modelo de aprendizaje profundo.

e. Se ha implementado y puesto en marcha un modelo basado en aprendizaje profundo capaz de clasificar automáticamente tres tipos de paquetes que circulan por el escenario virtual. Ello ha supuesto dotar al simulador de un nuevo sensor virtual.

f. Se han adquirido y trabajado conceptos técnicos muy variados, que van desde la programación de autómatas virtuales, el procesado de imágenes digitales, y el desarrollo de modelos de inteligencia artificial.

g. Se han adquirido destrezas en programación de autómatas industriales, en técnicas de visión y en lenguaje Python.

h. Se han identificado los aspectos a mejorar y los trabajos futuros.

Por todo ello, se considera alcanzado el objetivo principal de este proyecto de desarrollar e implementar un sensor de visión artificial para su uso en un simulador de procesos industriales, demostrando que puede proporcionar mejoras en funcionalidad y coste frente a los sensores de proximidad, habituales en los entornos industriales.

Anexos

Anexos A

Segmentación y Recorte Automático de Imágenes por Color(Create_Mask.py)

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2lab, rgb2gray
from skimage.morphology import disk, closing, remove_small_objects
from skimage.measure import regionprops, label
from skimage import img_as_ubyte
from pathlib import Path

def create_mask(ruta, plot):
    """
    Crea una máscara a partir de una imagen en función de umbrales de color,
    recorta las regiones detectadas y guarda la imagen recortada y redimensionada.

    Parámetros:
    ruta (str): Ruta del archivo de imagen.
    plot (bool): Si True, muestra la imagen original y la máscara binaria.

    Retorna:
    list or int: Lista de imágenes recortadas y redimensionadas si se detecta al menos
    un objeto,0 si no se detecta ningún objeto."""

    # Verificar si la ruta proporcionada es un archivo válido
    if not Path(ruta).is_file():
        print(f"La ruta {ruta} no es válida.")
        return 0

    # Leer la imagen desde la ruta proporcionada usando OpenCV
    RGB = cv2.imread(ruta) #Esta función de OpenCV se utiliza para leer una
    # imagen desde el disco.
    if RGB is None:
        print(f"No se pudo leer la imagen en la ruta {ruta}.")
        return 0

    # Convertir la imagen de BGR (formato predeterminado de OpenCV)
    # a RGB (formato esperado)
    RGB = cv2.cvtColor(RGB, cv2.COLOR_BGR2RGB)

    # Definir el tamaño al que se redimensionarán las imágenes recortadas
    tamano = (224, 224)

    # Convertir la imagen RGB al espacio de color LAB, que es más adecuado
    #para segmentación por color
    I = rgb2lab(RGB)

    # Definir los umbrales para cada canal en el espacio LAB para segmentar
    #el objeto deseado
    channel1_min, channel1_max = 0.000, 100.000  # Umbrales para el canal L
    channel2_min, channel2_max = -18.016, 23.049 # Umbrales para el canal a
    channel3_min, channel3_max = 14.411, 31.572  # Umbrales para el canal b


```

```
    # Crear una máscara binaria basada en los umbrales de color definidos
    sliderBW = ((I[:,:,0] >= channel1_min) & (I[:,:,0] <= channel1_max) &
                (I[:,:,1] >= channel2_min) & (I[:,:,1] <= channel2_max) &
                (I[:,:,2] >= channel3_min) & (I[:,:,2] <= channel3_max))

    # Convertir la máscara binaria a un formato adecuado para procesamiento (0 y 255)
    BW = (sliderBW * 255).astype(np.uint8)
        # Rellenar huecos en la máscara binaria usando operaciones de cierre
    BW_filled = closing(BW > 0, disk(3))
    # Eliminar objetos pequeños basados en un tamaño mínimo para reducir el ruido
    BW_filled = remove_small_objects(BW_filled, min_size=500)

    # Establecer a cero la mitad derecha de la máscara para
    #enfocar el análisis en la mitad izquierda
    BW_filled[:, BW_filled.shape[1]//2:] = 0

    # Mostrar la imagen original y la máscara binaria si el parámetro 'plot' es True
    if plot:
        plt.figure(figsize=(10, 5))
        plt.subplot(1, 2, 1)
        plt.imshow(RGB)
        plt.title('Imagen Original')
        plt.subplot(1, 2, 2)
        plt.imshow(BW_filled, cmap='gray')
        plt.title('Máscara Binaria')
        plt.show()

    # Etiquetar las regiones en la máscara binaria para identificar diferentes objetos
    labeled_bw = label(BW_filled)
    # Obtener propiedades de las regiones etiquetadas
    props = regionprops(labeled_bw)


    # Lista para almacenar los recortes de las regiones detectadas
    recortes = []

    if len(props) > 0:
        # Iterar sobre cada región detectada en la lista de propiedades 'props'
        for k, prop in enumerate(props):
            # Obtener la caja delimitadora (bounding box) de la región actual
            bb = prop.bbox
            minr, minc, maxr, maxc = bb  # Desempaquetar la caja
            #delimitadora en coordenadas (minr, minc, maxr, maxc)

            # Recortar la región de la imagen original usando la caja delimitadora
            cropped_img = RGB[minr:maxr, minc:maxc]

            # Convertir el recorte a escala de grises si es una imagen en color (RGB)
            cropped_img_gray = rgb2gray(cropped_img)
```

```
            # Redimensionar la imagen recortada al tamaño deseado (224x224 píxeles)
            cropped_img_resized =
            cv2.resize(cropped_img_gray, tamano, interpolation=cv2.INTER_AREA)

            # Crear un directorio para guardar los recortes si no existe ya
            path_original = Path(ruta)  # Convertir la ruta proporcionada
            #en un objeto Path

            output_dir = path_original.parent / 'recortes'  # Definir el
            subdirectorio 'recortes'
            dentro del directorio padre

            output_dir.mkdir(exist_ok=True)  # Crear el directorio 'recortes'
            #si no existe

            # Generar el nombre del archivo para el recorte
            nombre_recorte = f"{path_original.stem}_recorte_{k+1}{path_original.suffix}"
            filepath = output_dir / nombre_recorte  # Definir la
            #ruta completa para guardar el recorte

            # Guardar la imagen recortada y redimensionada en el directorio 'recortes'
            cv2.imwrite(str(filepath), img_as_ubyte(cropped_img_resized))

            # Añadir la imagen recortada y redimensionada a la lista de recortes
            recortes.append(cropped_img_resized)

            # Retornar la lista de imágenes recortadas y redimensionadas
            return recortes
    else:
        # Mostrar mensaje si no se detecta ningún objeto en la imagen
        print('Paquete no detectado')
        return 0



# Ejemplo de uso: Llamada a la función para procesar una imagen específica
#y mostrar resultados

#Cambiar dirección si es necesario

create_mask(r'C:\Users\alber\Desktop\Entrega_TFG\Entrenamiento\...
            ...\Frames\M\ezgif-frame-001.png', True)

```

Anexos B

Procesamiento Automático de Frames y Recorte de Imágenes(Process_Dir.py)

```python
import os
import nbformat

# Ruta al notebook que contiene la función create_mask
notebook_path = r'C:\Users\alber\Desktop\Entrega_TFG\Entrenamiento\Create_Mask.ipynb'

# Leer el contenido del notebook
with open(notebook_path, 'r', encoding='utf-8') as f:
    notebook_content = nbformat.read(f, as_version=4)

# Extraer y ejecutar solo celdas de código válidas
for cell in notebook_content.cells:
    if cell.cell_type == 'code':
        # Intentar ejecutar el código de la celda en el espacio global
        try:
            exec(cell.source, globals())
        except SyntaxError as e:
            print(f'Se omitió una celda con error de sintaxis: {e}')
        except Exception as e:
            print(f'Error al ejecutar una celda: {e}')

# Verificar que la función create_mask está disponible
if 'create_mask' not in globals():
    raise ImportError("No se encontró la función 'create_mask' en el notebook.")

# Lista de directorios que contienen las imágenes
directorios_imagenes = [
    r'C:\Users\alber\Desktop\Entrega_TFG\Entrenamiento\Frames\S',
    r'C:\Users\alber\Desktop\Entrega_TFG\Entrenamiento\Frames\M',
    r'C:\Users\alber\Desktop\Entrega_TFG\Entrenamiento\Frames\L'
]

# Procesar imágenes en cada uno de los directorios especificados
for directorio_imagenes in directorios_imagenes:
    # Verificar si el directorio existe
    if not os.path.isdir(directorio_imagenes):
        print(f'El directorio {directorio_imagenes} no existe.')
        continue

    # Iterar sobre todos los archivos en el directorio actual
    for archivo in os.listdir(directorio_imagenes):
        # Filtrar archivos para procesar solo imágenes con extensiones específicas
        if archivo.lower().endswith(('.png', '.jpg', '.jpeg')):
            # Construir la ruta completa del archivo de imagen
            ruta_imagen = os.path.join(directorio_imagenes, archivo)

            try:
                # Llamar a la función create_mask con la ruta de la imagen y
                #el parámetro 0
                create_mask(ruta_imagen, 0)
                print(f'Procesado: {ruta_imagen}')
            except Exception as e:
                # Imprimir un mensaje de error si algo sale mal
                #durante el procesamiento
                print(f'Error al procesar {ruta_imagen}: {e}')
```

Anexos C

Entrenamiento Modelo (DL_2.py)

```python
# Importar las bibliotecas necesarias
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, BatchNormalization,
ReLU, Flatten, Dense, Softmax, MaxPooling2D, Dropout
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import matplotlib.pyplot as plt
import numpy as np
import os
from tensorflow.keras import layers
from tensorflow.keras.optimizers import Adam

# Directorio donde están almacenadas las imágenes
data_dir = r'C:\Users\alber\Desktop\Entrega_TFG\Entrenamiento\Recortes'

# Parámetros
image_size = (224, 224)  # Tamaño de las imágenes que se redimensionarán
batch_size = 32  # Tamaño del lote para el entrenamiento y la validación
num_classes = 3  # Número de clases en la clasificación

# Obtener el número total de imágenes en cada carpeta
num_images_s = len(os.listdir(os.path.join(data_dir, 'S')))
num_images_m = len(os.listdir(os.path.join(data_dir, 'M')))
num_images_l = len(os.listdir(os.path.join(data_dir, 'L')))

# Mostrar la cantidad de imágenes en cada carpeta
print(f"Imágenes en 'S': {num_images_s}")
print(f"Imágenes en 'M': {num_images_m}")
print(f"Imágenes en 'L': {num_images_l}")

# Total de imágenes
total_images = num_images_s + num_images_m + num_images_l
print(f"Total de imágenes: {total_images}")

# Preparar los generadores de datos para entrenamiento y validación
datagen = ImageDataGenerator(
    rescale=1./255,  # Normaliza los píxeles de las imágenes al rango [0, 1]
    validation_split=0.25,  # Usamos el 25% de los datos para validación
    horizontal_flip=True,  # Realiza un volteo horizontal aleatorio
    width_shift_range=30./224,  # Traducción horizontal aleatoria de hasta 30 píxeles
    height_shift_range=30./224  # Traducción vertical aleatoria de hasta 30 píxeles
)
```

```
# Crear el generador de datos para el conjunto de entrenamiento
train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=image_size,  # Redimensiona las imágenes al tamaño especificado
    batch_size=batch_size,  # Tamaño del lote para entrenamiento
    subset='training',  # Subconjunto de entrenamiento (75% de los datos)
    class_mode='categorical'  # Las etiquetas están en formato categórico
)

# Crear el generador de datos para el conjunto de validación
validation_generator = datagen.flow_from_directory(
    data_dir,
    target_size=image_size,  # Redimensiona las imágenes al tamaño especificado
    batch_size=batch_size,  # Tamaño del lote para validación
    subset='validation',  # Subconjunto de validación (25% de los datos)
    class_mode='categorical'  # Las etiquetas están en formato categórico
)

# Definir el modelo
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),  # Capa
    #de convolución con 32 filtros y tamaño de kernel 3x3
    MaxPooling2D((2, 2)),  # Capa de max pooling para reducir la dimensión espacial
    Conv2D(64, (3, 3), activation='relu'),  # Capa de convolución con 64 filtros
    MaxPooling2D((2, 2)),  # Otra capa de max pooling
    Conv2D(128, (3, 3), activation='relu'),  # Capa de convolución con 128 filtros
    MaxPooling2D((2, 2)),  # Otra capa de max pooling
    Conv2D(128, (3, 3), activation='relu'),  # Capa de convolución con 128 filtros
    MaxPooling2D((2, 2)),  # Otra capa de max pooling
    Flatten(),  # Aplana las características 2D en un vector 1D
    Dense(512, activation='relu'),  # Capa densa totalmente conectada con 512 unidades
    Dropout(0.5),  # Capa de dropout para evitar el sobreajuste
    Dense(num_classes, activation='softmax')  # Capa de salida con
    #el número de clases (softmax para clasificación)
])

# Compilar el modelo
model.compile(
    optimizer=Adam(learning_rate=1e-4),  # Optimizador Adam con
    #tasa de aprendizaje ajustada
    loss='categorical_crossentropy',  # Función de pérdida para clasificación categórica
    metrics=['accuracy']  # Métrica para evaluar el rendimiento del modelo
)

# Entrenar el modelo
history = model.fit(
    train_generator,
    epochs=100,  # Número máximo de épocas para entrenar
    validation_data=validation_generator,  # Datos de
    #validación para monitorear el rendimiento
)
```

```
# Graficar el progreso del entrenamiento
plt.figure(figsize=(12, 6))

# Gráfico de la pérdida de entrenamiento y validación
plt.subplot(1, 2, 1)
# Pérdida en el conjunto de entrenamiento
plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
# Pérdida en el conjunto de validación
plt.plot(history.history['val_loss'], label='Pérdida de Validación')
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.legend()

# Gráfico de la precisión de entrenamiento y validación
plt.subplot(1, 2, 2)
# Precisión en el conjunto de entrenamiento
plt.plot(history.history['accuracy'], label='Precisión de Entrenamiento')
# Precisión en el conjunto de validación
plt.plot(history.history['val_accuracy'], label='Precisión de Validación')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend()
plt.show()

# Evaluar el modelo en el conjunto de validación
eval_result = model.evaluate(validation_generator)
print(f"Precisión en el conjunto de validación: {eval_result[1] * 100:.2f}\%")

# Evaluar el modelo en el conjunto de entrenamiento
train_eval_result = model.evaluate(train_generator)
print(f"Precisión en el conjunto de entrenamiento: {train_eval_result[1] * 100:.2f}\%")

# Predicción en una imagen de validación
from tensorflow.keras.preprocessing import image

# Leer y preparar una imagen para la predicción
img_path = validation_generator.filepaths[0]  # Ruta a la
#primera imagen del conjunto de validación
img = image.load_img(img_path, target_size=image_size)  # Cargar la
#imagen y redimensionarla
img_array = image.img_to_array(img) / 255.0  # Convertir la imagen
#a un array y normalizar
img_array = np.expand_dims(img_array, axis=0)  # Añadir una dimensión de lote

# Realizar la predicción
predictions = model.predict(img_array)  # Obtener las predicciones del modelo
predicted_class = np.argmax(predictions, axis=1)  # Obtener la clase
#con mayor probabilidad

# Imprimir el índice de la clase predicha
print(f"Índice de la etiqueta predicha: {predicted_class[0]}")
```

Anexos D

Comprobación de funcionamiento del modelo (Rutina modelo_entrenamo.h5.py)

```python
import numpy as np
from PIL import Image
from skimage import color, measure
from skimage.morphology import remove_small_objects
import cv2
import tensorflow as tf  # Para cargar y usar el modelo .h5

# Función para cargar la imagen
def cargar_imagen(image_path):
    """Carga y preprocesa la imagen."""
    RGB = np.array(Image.open(image_path))  # Abre la imagen usando PIL
    #y la convierte a un array numpy.
    return RGB[:, :, :3]  # Retorna solo los 3 canales RGB, ignorando
    #un posible canal alfa.

# Función para crear una máscara binaria basada en umbrales
def crear_mascara(I):
    """Crea una máscara binaria basada en los umbrales."""
    # Umbrales predefinidos para cada canal en el espacio de color LAB
    min_max_values = {
        'channel1': (0.000, 100.000),
        'channel2': (-18.016, 23.049),
        'channel3': (14.411, 31.572)
    }
    channel1Min, channel1Max = min_max_values['channel1']
    channel2Min, channel2Max = min_max_values['channel2']
    channel3Min, channel3Max = min_max_values['channel3']

    # Retorna una máscara binaria que identifica los píxeles que cumplen
    #los umbrales en los 3 canales LAB.
    return ((I[:, :, 0] >= channel1Min) & (I[:, :, 0] <= channel1Max) &
            (I[:, :, 1] >= channel2Min) & (I[:, :, 1] <= channel2Max) &
            (I[:, :, 2] >= channel3Min) & (I[:, :, 2] <= channel3Max))

# Función para procesar la máscara
def procesar_mascara(BW):
    """Procesa la máscara para rellenar agujeros y eliminar objetos pequeños."""
    # Convierte la máscara binaria a tipo uint8 para procesamiento con OpenCV
    BW_filled = np.uint8(BW)

    # Realiza una operación de cierre (dilatar y luego erosionar)
    #para eliminar agujeros pequeños
    BW_filled = cv2.morphologyEx(BW_filled, cv2.MORPH_CLOSE, np.ones((3, 3), np.uint8))

    # Aplica una operación de apertura (erosionar y luego dilatar) para suavizar bordes
    BW_filled = cv2.morphologyEx(BW_filled, cv2.MORPH_OPEN, np.ones((3, 3), np.uint8))
```

```
1
2       # Elimina objetos pequeños de la imagen binaria (menores a 500 píxeles)
3       BW_clean = remove_small_objects(BW_filled.astype(bool), min_size=500)
4
5       # Elimina la mitad derecha de la máscara, estableciendo a 0 todos los
6       #valores en esa región
7       cols = BW_clean.shape[1]
8       BW_clean[:, cols // 2:] = 0  # Eliminar la mitad derecha de la máscara
9       return BW_clean
10
11
12  # Función para recortar la imagen basada en la región de interés
13  def recortar(RGB, BW_clean):
14      """Recorta la región de interés y guarda la imagen recortada."""
15      # Obtiene las propiedades de las regiones conectadas en la máscara limpia
16      props = measure.regionprops(np.uint8(BW_clean))
17
18      if len(props) > 0:
19          for prop in props:
20              # Extrae la caja delimitadora (BoundingBox) de la región de interés
21              bb = prop.bbox  # Obtener la BoundingBox
22
23              # Recorta la imagen original según las coordenadas de la BoundingBox
24              croppedImg = RGB[bb[0]:bb[2], bb[1]:bb[3]]
25
26              # Si la imagen recortada tiene 3 canales (RGB),
27              #la convierte a escala de grises
28              croppedImgGray = cv2.cvtColor(croppedImg, cv2.COLOR_RGB2GRAY)...
29              ...if croppedImg.shape[2] == 3 else croppedImg
30
31              # Redimensiona la imagen recortada a 224x224
32              #(dimensión esperada por el modelo)
33              croppedImgResized = cv2.resize(croppedImgGray, (224, 224))
34
35              return croppedImgResized  # Devuelve la imagen
36              #recortada y redimensionada
37      else:
38          print('Paquete no detectado')  # Si no hay regiones de interés
39          return None
40          # Función para preparar la imagen recortada para el modelo
41
42
```

```python
def preparar_entrada_modelo(croppedImgResized):
    """Prepara la imagen recortada para la entrada del modelo."""
    # Convierte la imagen a un array numpy con tipo de dato float32
    image_np = np.array(croppedImgResized, dtype=np.float32)

    # Duplica el canal de grises para obtener una imagen RGB (3 canales)
    image_np_rgb = np.stack([image_np] * 3, axis=-1)  # Replicar el
    #canal de grises 3 veces

    # Expande las dimensiones para simular un batch de tamaño 1
    #(añade la dimensión de batch)
    image_np_rgb = np.expand_dims(image_np_rgb, axis=0)

    return image_np_rgb  # Devuelve la imagen lista para el modelo
    # Función para cargar el modelo en formato .h5
def cargar_modelo_h5(model_path):
    """Carga el modelo .h5."""
    model = tf.keras.models.load_model(model_path)  # Carga el modelo
    #preentrenado en formato .h5
    return model



# Función para realizar la predicción usando el modelo .h5
def realizar_prediccion(model, image_np):
    """Realiza la predicción usando el modelo .h5."""
    predictions = model.predict(image_np)  # Realiza la predicción
    return np.argmax(predictions, axis=1)  # Retorna la clase con mayor probabilidad



# Función para clasificar el paquete
def clasificar(croppedImgResized, model_path):
    """Clasifica el paquete"""
    if croppedImgResized is not None:
        # Prepara la imagen para el modelo
        image_np = preparar_entrada_modelo(croppedImgResized)

        # Carga el modelo
        model = cargar_modelo_h5(model_path)

        # Realiza la predicción
        predicted_class = realizar_prediccion(model, image_np)

        # Obtiene la clase predicha
        tipo_paquete = predicted_class[0]
    else:
        # Si no se detecta paquete, se clasifica como "tipo 3"
        tipo_paquete = 3
    return int(tipo_paquete)  # Retorna la clase del paquete
```

```python
# Función principal
def main():
    # Ruta del modelo y de la imagen
    model_path = 'C:/Users/alber/Desktop/Entrega_TFG/Entrenamiento/...
    ...'/modelo_entrenado.h5'
    image_path = 'C:/Users/alber/Desktop/Entrega_TFG/Entrenamiento/...
    ...'/FramesPrueba/frame_15.png'

    # Flujo de procesamiento de la imagen
    RGB = cargar_imagen(image_path)  # Cargar la imagen
    I = color.rgb2lab(RGB)  # Convertir a espacio LAB
    BW = crear_mascara(I)  # Crear la máscara binaria
    BW_clean = procesar_mascara(BW)  # Procesar la máscara
    croppedImgResized = recortar(RGB, BW_clean)  # Recortar la región de interés

    # Clasificar el paquete
    tipo_de_paquete = clasificar(croppedImgResized, model_path)
    print(f'Paquete de tipo {tipo_de_paquete}')

# Punto de entrada al programa
if __name__ == "__main__":
    main()
```

Anexos E

Lista Componentes Factory IO (Componentes.py)

```python
import requests

try:
    # URL de la API para obtener las etiquetas (tags)
    url = 'http://localhost:7410/api/tags'
    response = requests.get(url)
    response.raise_for_status()  # Lanza un error si la solicitud falla

    # Obtener la respuesta JSON
    tags = response.json()

    # Verificar si la respuesta es una lista
    if isinstance(tags, list):
        # Imprimir encabezados
        print(f"{'Name':<30} {'ID':<40} {'Address':<10}...
        ...{'Type':<10} {'Kind':<10} {'Value'}")
        print("=" * 100)

        # Imprimir cada etiqueta en columnas
        for tag in tags:
            if isinstance(tag, dict):
                print(f"{tag.get('name', 'N/A'):<30} {tag.get('id', 'N/A'):<40}...
                ...{tag.get('address', 'N/A'):<10}{tag.get('type', 'N/A'):<10} ...
                ...{tag.get('kind', 'N/A'):<10} {tag.get('value', 'N/A')}")
            else:
                print(f"Elemento inesperado en la lista: {tag}")
    else:
        print(f"Respuesta inesperada: {tags}")

except requests.exceptions.RequestException as e:
    print(f"Error al hacer la solicitud: {e}")
```

Anexos F

Clasificación en tiempo real con sensor de visión artificial (Proceso_Final.py)

```python
import numpy as np
from PIL import Image
from skimage import color, measure
from skimage.morphology import remove_small_objects
import cv2
import os
import tensorflow as tf  # Para cargar y usar el modelo .h5
import time
import requests  # Para interactuar con la API web de Factory IO

# Función para cargar la imagen
def cargar_imagen(image_path):
    """Carga y preprocesa la imagen."""
    RGB = np.array(Image.open(image_path))  # Abre la imagen usando PIL
    #y la convierte a un array numpy.
    return RGB[:, :, :3]  # Retorna solo los 3 canales RGB,
    #ignorando un posible canal alfa.

# Función para crear una máscara binaria basada en umbrales
def crear_mascara(I):
    """Crea una máscara binaria basada en los umbrales."""
    # Umbrales predefinidos para cada canal en el espacio de color LAB
    min_max_values = {
        'channel1': (0.000, 100.000),
        'channel2': (-18.016, 23.049),
        'channel3': (14.411, 31.572)
    }
    channel1Min, channel1Max = min_max_values['channel1']
    channel2Min, channel2Max = min_max_values['channel2']
    channel3Min, channel3Max = min_max_values['channel3']

    # Retorna una máscara binaria que identifica los píxeles
    #que cumplen los umbrales en los 3 canales LAB.
    return ((I[:, :, 0] >= channel1Min) & (I[:, :, 0] <= channel1Max) &
            (I[:, :, 1] >= channel2Min) & (I[:, :, 1] <= channel2Max) &
            (I[:, :, 2] >= channel3Min) & (I[:, :, 2] <= channel3Max))

# Función para procesar la máscara
def procesar_mascara(BW):
    """Procesa la máscara para rellenar agujeros y eliminar objetos pequeños."""
    # Convierte la máscara binaria a tipo uint8 para procesamiento con OpenCV
    BW_filled = np.uint8(BW)

    # Realiza una operación de cierre (dilatar y luego erosionar)
    #para eliminar agujeros pequeños
    BW_filled = cv2.morphologyEx(BW_filled, cv2.MORPH_CLOSE, np.ones((3, 3), np.uint8))
```

```
    # Aplica una operación de apertura (erosionar y luego dilatar)
    #para suavizar bordes
    BW_filled = cv2.morphologyEx(BW_filled, cv2.MORPH_OPEN, np.ones((3, 3), np.uint8))

    # Elimina objetos pequeños de la imagen binaria (menores a 500 píxeles)
    BW_clean = remove_small_objects(BW_filled.astype(bool), min_size=500)

    # Elimina la mitad derecha de la máscara, estableciendo a 0
    #todos los valores en esa región
    cols = BW_clean.shape[1]
    BW_clean[:, cols // 2:] = 0  # Eliminar la mitad derecha de la máscara
    return BW_clean

    # Función para recortar la imagen basada en la región de interés
def recortar(RGB, BW_clean):
    """Recorta la región de interés y guarda la imagen recortada."""
    # Obtiene las propiedades de las regiones conectadas en la máscara limpia
    props = measure.regionprops(np.uint8(BW_clean))

    if len(props) > 0:
        for prop in props:
            # Extrae la caja delimitadora (BoundingBox) de la región de interés
            bb = prop.bbox  # Obtener la BoundingBox

            # Recorta la imagen original según las coordenadas de la BoundingBox
            croppedImg = RGB[bb[0]:bb[2], bb[1]:bb[3]]

            # Si la imagen recortada tiene 3 canales (RGB),
            #la convierte a escala de grises
            croppedImgGray = cv2.cvtColor(croppedImg, cv2.COLOR_RGB2GRAY)...
            ...if croppedImg.shape[2] == 3 else croppedImg

            # Redimensiona la imagen recortada a 224x224
            #(dimensión esperada por el modelo)
            croppedImgResized = cv2.resize(croppedImgGray, (224, 224))

            return croppedImgResized  # Devuelve la imagen recortada y redimensionada
    else:
        print('Paquete no detectado')  # Si no hay regiones de interés
        return None

# Función para preparar la imagen recortada para el modelo
def preparar_entrada_modelo(croppedImgResized):
    """Prepara la imagen recortada para la entrada del modelo."""
    # Convierte la imagen a un array numpy con tipo de dato float32
    image_np = np.array(croppedImgResized, dtype=np.float32)
```

```
    # Duplica el canal de grises para obtener una imagen RGB (3 canales)
    image_np_rgb = np.stack([image_np] * 3, axis=-1)  # Replicar el canal
    #de grises 3 veces

    # Expande las dimensiones para simular un batch de tamaño 1
    #(añade la dimensión de batch)
    image_np_rgb = np.expand_dims(image_np_rgb, axis=0)

    return image_np_rgb  # Devuelve la imagen lista para el modelo

# Función para cargar el modelo en formato .h5
def cargar_modelo_h5(model_path):
    """Carga el modelo .h5."""
    model = tf.keras.models.load_model(model_path)  # Carga el modelo preentrenado
    #en formato .h5
    return model



# Función para realizar la predicción usando el modelo .h5
def realizar_prediccion(model, image_np):
    """Realiza la predicción usando el modelo .h5."""
    predictions = model.predict(image_np)  # Realiza la predicción
    return np.argmax(predictions, axis=1)  # Retorna la clase con mayor probabilidad



# Función para clasificar el paquete
def clasificar(croppedImgResized, model_path):
    """Clasifica el paquete"""
    if croppedImgResized is not None:
        # Prepara la imagen para el modelo
        image_np = preparar_entrada_modelo(croppedImgResized)

        # Carga el modelo
        model = cargar_modelo_h5(model_path)

        # Realiza la predicción
        predicted_class = realizar_prediccion(model, image_np)

        # Obtiene la clase predicha
        tipo_paquete = predicted_class[0]
    else:
        # Si no se detecta paquete, se clasifica como "tipo 3"
        tipo_paquete = 3
    return int(tipo_paquete)  # Retorna la clase del paquete
```

```python
# Función para enviar el valor a Factory IO
def enviar_valor_factory_io(sensor_id, value_to_force):
    """Envía el valor de clasificación al sensor de Factory IO usando la API web."""
    url = 'http://localhost:7410/api/tag/values-force'  # URL de la API de Factory IO
    headers = {'Content-Type': 'application/json'}

    # Estructura del payload
    payload = [
        {
            "id": sensor_id,
            "value": value_to_force
        }
    ]

    try:
        # Hacer la petición POST para enviar los datos
        response = requests.put(url, json=payload, headers=headers)
        if response.status_code != 200:
            print(f'Error al enviar el valor al sensor. Código de estado:...
            ...{response.status_code}')
    except Exception as e:
        print(f'Error de conexión: {e}')

#Función para leer el valor de un sensor
def leer_valor_sensor(nombre_sensor):

    url = 'http://localhost:7410/api/tag/values/by-name'  # URL del endpoint
    headers = {
        "Content-Type": "application/json"
    }

    try:
        # Hacer la solicitud GET para obtener el valor del sensor
        response = requests.get(url, json=[nombre_sensor], headers=headers)

        # Verificar que la respuesta sea exitosa
        if response.status_code == 200:
            valores_etiquetas = response.json()
            if valores_etiquetas:
                # Devolver el valor del sensor (suponiendo que solo pedimos un sensor)
                return valores_etiquetas[0]
            else:
                print("No se encontraron valores para el sensor.")
                return None
        else:
            print(f"Error al obtener el valor del sensor:...
            ...{response.status_code}, {response.text}")
            return None
    except requests.exceptions.ConnectionError as e:
        print(f"Error de conexión: {e}")
        return None
    except Exception as e:
        print(f"Ocurrió un error: {e}")
        return None
```

```python
# Función principal
def main():
    # Ruta del modelo y de la imagen
    #Cambiar la dirección si es necesario.
    model_path = 'C:/Users/alber/Desktop/Entrega_TFG/Entrenamiento/modelo_entrenado.h5'
    ruta_grabacion = 'C:/Users/alber/Desktop/Entrega_TFG/Entrenamiento/trama0.png'

    # Iniciar captura de video desde la cámara
    cap = cv2.VideoCapture(1)  #Cambiar el número dependiendo de la cámara
    intervalo = 1  # Intervalo de espera entre iteraciones (en segundos)
    sensor_id = '998d6a68-bc8f-4f1c-bc10-77301c84177f'  # ID del sensor
    #al que se enviará el valor
    nombre_sensor = "Optical sensor"

    # Enviar un valor inicial al sensor de Factory IO
    enviar_valor_factory_io(sensor_id, 3)

    for i in range(1000):  # Bucle de procesamiento
        cap.set(3, 1920)  # Establecer resolución de la cámara
        cap.set(4, 1080)

        # Leer el valor del sensor óptico
        optical_sensor = leer_valor_sensor(nombre_sensor)

        if  optical_sensor:  # Si el sensor está en False, procesa la imagen
            ret, frame = cap.read()  # Capturar un frame de la cámara
            if not ret:
                print("Error al capturar el frame.")
                continue

            # Flujo de procesamiento de la imagen
            RGB = frame[:, :, :3]  # Cargar la imagen en formato RGB
            cv2.imwrite(ruta_grabacion, RGB)  # Guardar la imagen
            RGB = cargar_imagen(ruta_grabacion)  # Cargar la imagen guardada
            I = color.rgb2lab(RGB)  # Convertir a espacio LAB
            BW = crear_mascara(I)  # Crear la máscara binaria
            BW_clean = procesar_mascara(BW)  # Procesar la máscara
            croppedImgResized = recortar(RGB, BW_clean)  # Recortar la región de interés

            # Clasificar el paquete
            tipo_de_paquete = clasificar(croppedImgResized, model_path)
            print(f'Paquete de tipo {tipo_de_paquete}')

            # Enviar el valor clasificado al sensor de Factory IO
            enviar_valor_factory_io(sensor_id, tipo_de_paquete)

        time.sleep(intervalo)  # Pausar entre iteraciones

    # Liberar la captura de video y restaurar el valor del sensor al finalizar
    cap.release()
    enviar_valor_factory_io(sensor_id, 3)

# Punto de entrada al programa
if __name__ == "__main__":
    main()
```

Bibliografía

[1] Sigma21. *Automatización Industrial: qué es y por qué es tan importante*. 2021. URL: https://www.sicma21.com/automatizacion-industrial-importancia-y-beneficios/ (visitado 11-04-2024).

[2] José Luis del Val Román. "Industria 4.0: la transformación digital de la industria". En: *Valencia: Conferencia de Directores y Decanos de Ingeniería Informática, Informes CODDII*. 2016.

[3] Ana Ayerbe. "La ciberseguridad de la industria 4.0: Un medio para la continuidad del negocio". En: *Economía industrial* 410 (2018), págs. 37-46.

[4] Jordi Salazar y Santiago Silvestre. "Internet de las cosas". En: *Techpedia. České vysoké učení technické v Praze Fakulta elektrotechnická* (2016).

[5] Makenzie Buenning. *¿Qué es la gestión de dispositivos IoT?* 2018. URL: https://www.ninjaone.com/es/blog/que-es-la-gestion-de-dispositivos-iot/ (visitado 11-04-2024).

[6] Luis Felipe Ortiz Clavijo et al. "Computación en la Nube: Estudio de herramientas orientadas a la Industria 4.0". En: *Lámpsakos* 20 (2018), págs. 68-75.

[7] Shannon Jackson-Barnes. "Edge Computing VS Cloud Computing: Differences, Benefits, and Best Practices". En: (2022).

[8] Carlos Alonso Hernández et al. "Blockchain y criptomonedas". En: (2019).

[9] Julio Cesar Ponce Gallegos et al. "Inteligencia artificial". En: (2014).

[10] Lasse Rouhiainen. "Inteligencia artificial". En: *Madrid: Alienta Editorial* (2018), págs. 20-21.

[11] AI Art Generator. "Inteligencia Artificial". En: (2024).

[12] Ana González Marcos et al. "Técnicas y algoritmos básicos de visión artificial". En: *Técnicas y Algoritmos Básicos de Visión Artificial* (2006).

[13] Florelva Rozo-García. "Revisión de las tecnologías presentes en la industria 4.0". En: *Revista UIS Ingenierías* 19.2 (2020), págs. 177-191.

[14] M. Mitchell AWaldrop. "News Feature: What are the limits of deep learning?" En: *Proceedings of the National Academy of Sciences* 116.4 (2019), págs. 1074-1077.

[15] Damián Jorge Matich. "Redes Neuronales: Conceptos básicos y aplicaciones". En: *Universidad Tecnológica Nacional, México* 41 (2001), págs. 12-16.

[16] Rodrigo Salas. "Redes neuronales artificiales". En: *Universidad de Valparaıso. Departamento de Computación* 1.1 (2004), págs. 1-7.

[17] Tomás Contreras Jodra. "Aprendizaje de programación de PLCs con simulaciones en Factory IO." En: (2021).

[18] Factory IO. *Sensores*. 2006. URL: https://docs.factoryio.com/manual/parts/sensors/#incremental-encoder (visitado 11-04-2024).

[19] Sergio Luján-Mora. "Programación de aplicaciones web: historia, principios básicos y clientes web". En: (2002).

[20] Factory IO. *API Web*. 2006. URL: https://docs.factoryio.com/manual/web-api/ (visitado 11-04-2024).

[21] Josep Balcells, José Luis Romeral y José Luis Romeral Martínez. *Autómatas programables*. Vol. 1089. Marcombo, 1997.

[22] Michael Tiegelkamp y Karl-Heinz John. *IEC 61131-3: Programming industrial automation systems*. Vol. 166. Springer, 2010.

[23] Ernesto Granado, Washington Marín y Omar Pérez. "Desarrollo de un laboratorio de sistemas y comunicaciones industriales para la mejora del proceso enseñanza/aprendizaje". En: *Revista de la Facultad de Ingeniería Universidad Central de Venezuela* 25.1 (2010), págs. 33-42.

[24] Andres Felipe Nieto Morales, Andres Mauricio Reyes Porras et al. "Diseño de prácticas virtuales de automatización basado en la conectividad entre Factory I/OY Codesys". En: (2021).

[25] Grupo COdesys. *SERVIDOR OPC CODESYS*. 2023. URL: https://www.codesys.com/products/codesys-runtime/opc-server.html (visitado 20-04-2024).

[26] Oriol Bayó Puxan. "Metodología per implementar automatismes GRAFCET en microprocessadors programats en C". En: (2005).

[27] JOSÉ MANUEL GEA. *GRAFCET*. 2006. URL: https://www.automatas.org/redes/grafcet.htm (visitado 11-04-2024).

references.bib

Acrónimos

AEPD Agencia Española de Protección de Datos

RGPD Reglamento General de Protección de Datos

IBM International Business Machines

PLC Programmable Logic Controller

OLE Object Linking and Embedding

OPC OLE for Process COntrol

COM Component Object Model

DCOM Distributed Component Object Model

SCADA Supervisory Control and Data Acquisition

DCS Distributed Control System

ERP Enterprise Resource Planning

GRAFCET Graphe Fonctionnel de Commande Etapes-Transitions

HTTP Hypertext Transfer Protocol

FPGA Field Programable Gate Array