ESCUELA DE INGENIERÍAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial.

TRABAJO FIN DE GRADO

Implementación de un sistema robótico de Pick and Place a través de visión y computación acelerada.

Autor: María del Carmen Salas Casas.

Titulación: Ingeniería Electrónica Industrial.

Tutor: José Ramón Salinas Vázquez.

Área de Conocimiento: Tecnología Electrónica.

Departamento: Tecnología Electrónica.

Málaga, 4 de septiembre de 2023

Agradecimientos

Quería comenzar dando las gracias a la persona que ha estado a mi lado en este proyecto, dándole forma y aconsejándome en cada paso, mi tutor José Ramón Salinas. Gracias José Ramón por tu cercanía, tu dedicación, tu entusiasmo, y sobre todo gracias por transmitirme algo tan bonito como es la pasión por lo que hacemos.

Gracias a la empresa Grupo PREMO y a todas las personas que la conforman, así como a Francisco Sánchez por recomendarme a José Ramón como tutor e impulsarme a conocer dicha empresa.

Gracias a mi hermano mayor por haber sido para mí todo un referente y a mi hermano pequeño por su confianza plena aun cuando ni yo misma la tenía. Gracias a mis amigos por escucharme y comprenderme siempre. Gracias a mi familia por celebrar mis éxitos como si fueran suyos propios. Gracias a mi pareja por ser mi mayor apoyo, por entenderme, respetarme y acompañarme en cada proceso. Y en especial gracias a mis padres por hacer de mí la persona que soy hoy y por mostrarme su compresión y apoyo incondicional, por darme aliento y serenidad cuando más lo he necesitado.

Por último, agradecer a una de las personas más especiales y que más ha confiado en mí a lo largo de esta aventura que es la universidad, aunque desgraciadamente no ha podido acompañarme hasta el final siempre ha estado y estará ahí. Gracias, mamica.

Resumen

En este documento se recoge tanto el desarrollo como los resultados obtenidos en la ejecución del presente Trabajo Final de Grado.

Dicho trabajo consiste en desarrollar un sistema de visión artificial capaz de detectar y clasificar dos tipos de piezas diferentes por medio de la computación acelerada, concretamente las piezas 3DC11LP y 3DC14EMR-ULP desarrolladas por la empresa PREMO. Dichas piezas deben ser detectadas sobre el plano de trabajo, identificadas y posteriormente recogidas y orientadas para ser adecuadamente almacenadas en su blíster correspondiente, en caso de tratarse de la pieza 3DC14EMR-ULP o la cara superior de la pieza 3DC11LP. Si la pieza a recoger es identificada como la cara inferior de la pieza 3DC11LP, esta será directamente desechada con el fin de que sea reubicada en el plano y volver a realizar el proceso de detección y clasificación con el propósito de empaquetarla adecuadamente.

Posteriormente, el proyecto incluirá el desarrollo de un rastreador de piezas a fin de observar la rapidez de detección de las piezas mediante el uso de la visión y computación acelerada. Esto permitirá realizar el proceso de detección en un tiempo significativamente menor en comparación con los sistemas de visión convencionales y reconocer también posibles factores limitantes en cuanto al tiempo de detección de las piezas y la velocidad de los movimientos ejecutados por el brazo robótico empleado. Asimismo, con vistas a realizar movimientos articulares en el proceso del rastreo, se aplica la función de cinemática inversa para obtener la posición de las articulaciones del brazo robótico que se requieren para alcanzar la posición destino. De este modo dejarán los movimientos del brazo de ser prácticamente un factor limitante en términos de tiempo en dicha aplicación, pudiendo evaluar adecuadamente de esta manera la velocidad de funcionamiento del sistema.

Finalmente, se incluye una cinta transportadora al clasificador de piezas con el fin de integrar el sistema rastreador en dicha aplicación.

Todo ello resulta posible gracias a la creación de dos redes neuronales personalizadas: una para la detección de los dos tipos de piezas empleadas y otra para la distinción entre la cara superior e inferior de la pieza 3DC11LP y a la programación necesaria para satisfacer los requisitos y criterios previamente establecidos. Esto incluye los movimientos requeridos para que el brazo robótico recoja y deposite las piezas identificadas en sus respectivos blísteres, al tiempo que garantiza la orientación correcta y el cumplimiento de los criterios de recogida.

Debe tenerse en cuenta la ubicación de la cámara, la cual se encuentra incorporada en el extremo del brazo robótico para disponer de mayor precisión de las imágenes cuando se requiera.

Palabras clave

Visión Artificial, Aprendizaje Profundo, Redes Neuronales Convolucionales (CNN), Jetson Nano, NVIDIA, robots colaborativos, robots equipados con sistemas de visión, robótica guiada por visión.

UNIVERSIDAD
DE MÁLAGA

Abstract

This document presents both the development and the results obtained in the execution of this Final Degree Project.

The project aims to develop an artificial vision system capable of detecting and classifying two different types of components, namely, 3DC11LP and 3DC14EMR-ULP, developed by PREMO company, utilizing accelerated computing. These components must be detected on the work surface, identified, and subsequently picked up and oriented for proper storage in their respective blister packaging, in the case of the 3DC14EMR-ULP part or the upper face of the 3DC11LP part. If the component to be picked up is identified as the lower face of the 3DC11LP component, it will be promptly discarded for repositioning on the work surface, and the detection and classification process will be repeated with the aim of proper packaging.

Subsequently, the project will include the development of a part tracker to monitor the speed of parts detection through vision and accelerated computing. This will make it possible to carry out the detection process in a significantly shorter time compared to conventional vision systems and to identify possible limiting factors in terms of the detection time of the parts and the speed of the movements performed by the robotic arm used. In addition, to perform the joint movements in the tracking process, the inverse kinematics function is used to determine the position of the joints of the robotic arm required to reach the target position. In this manner, the movements of the robotic arm are practically no longer a limiting factor in terms of time in this application, enabling an appropriate assessment of the system's operating speed.

Finally, a conveyor belt is incorporated into the parts classifier to integrate the tracking system into this application.

All of this has become possible thanks to the creation of two custom-made neural networks: one for detecting the two types of parts and the other for distinguishing between the upper and lower face of the 3DC11LP component, and the programming required to meet the predetermined criteria and specifications. This involves the movements necessary for the robotic arm to pick up and deposit the identified parts into their respective blisters, while ensuring correct orientation and compliance with the pick-up criteria.

The position of the camera needs consideration, as it is integrated at the end of the robotic arm for more precise imaging, if necessary.

Keywords

Computer Vision, Deep Learning, Convolutional Neural Networks (CNN), Jetson Nano, NVIDIA, collaborative robots, robots equipped with vision systems, vision guided robotics.

Índice

Índice de figuras

Índice de tablas

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Memoria descriptiva

1. Introducción

En los últimos años la automatización de los procesos industriales ha sufrido un significativo avance debido a la situación del mercado actual, globalizado y extremadamente competitivo, por la que las empresas se ven obligadas a luchar continuamente para alcanzar una mayor rentabilidad, eficiencia y calidad en el control de sus procesos. Este gran cambio está siendo posible gracias al desarrollo tecnológico que está cambiando radicalmente el modo en el que interactuamos con el mundo que nos rodea. Una de las principales manifestaciones de este cambio es la creciente implementación de robots en la industria, especialmente en procesos repetitivos y fatigosos para el ser humano, lo cual produce una mejora de la eficiencia, garantiza una mayor seguridad, un aumento de la productividad y la calidad y, por ende, una optimización de los costes.

En Europa el uso de robots industriales está en constante crecimiento. Los 27 Estados miembros de la Unión Europea (UE) instalaron un total de casi 72.000 unidades en el año 2022, lo que supone un 6% más que el año anterior. Estos son resultados preliminares, presentados por la Federación Internacional de Robótica (IFR). Marina Bill, presidenta de la IFR, afirma que los cinco principales países de la UE causantes de este aumento son Alemania, Francia, Italia, Polonia y España, representando estos países alrededor del 70% de los robots industriales instalados en 2022 en la UE [1].

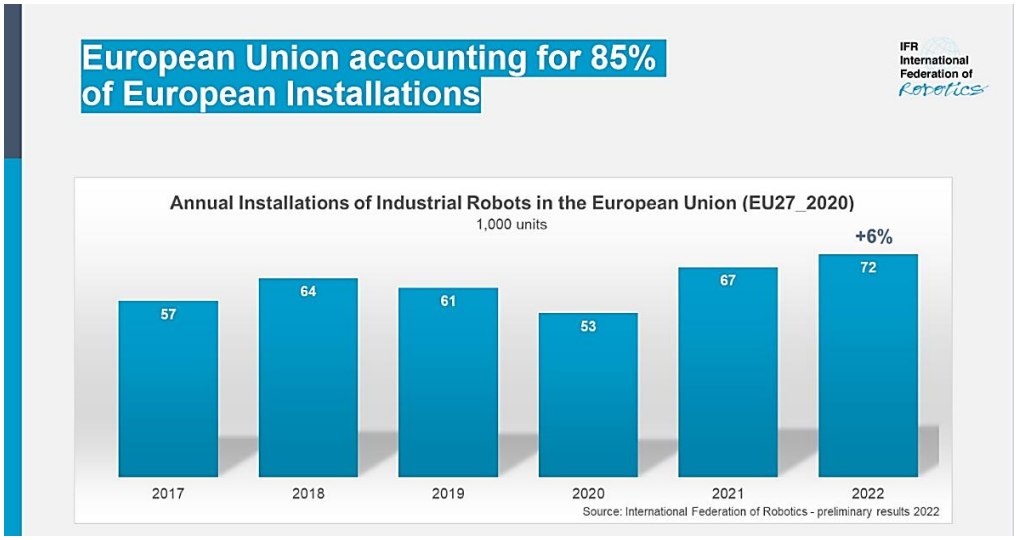

Figura 1. *Instalaciones anuales de robots industriales en la Unión Europea (UE)* [1].

Este incremento también se refleja a nivel mundial donde la cifra asciende a 531.060 unidades en todo el mundo en el año 2022, lo que representa un incremento del 2% en comparación con el año anterior como se puede observar en la ***Figura 2***.

En este sentido es donde se destaca una de las áreas más prometedoras de la industria, la robótica colaborativa, donde humanos y robots trabajan en armonía con el fin de optimizar los procesos y mejorar la eficiencia. Además, si a las ventajas aportadas por la robótica se le suman la implementación de la visión artificial en los robots, se obtienen los **sistemas VGR *(Vision Guided Robotic Systems)*** [2] que aportan una mayor flexibilidad en la automatización de procesos mediante el uso de robótica.

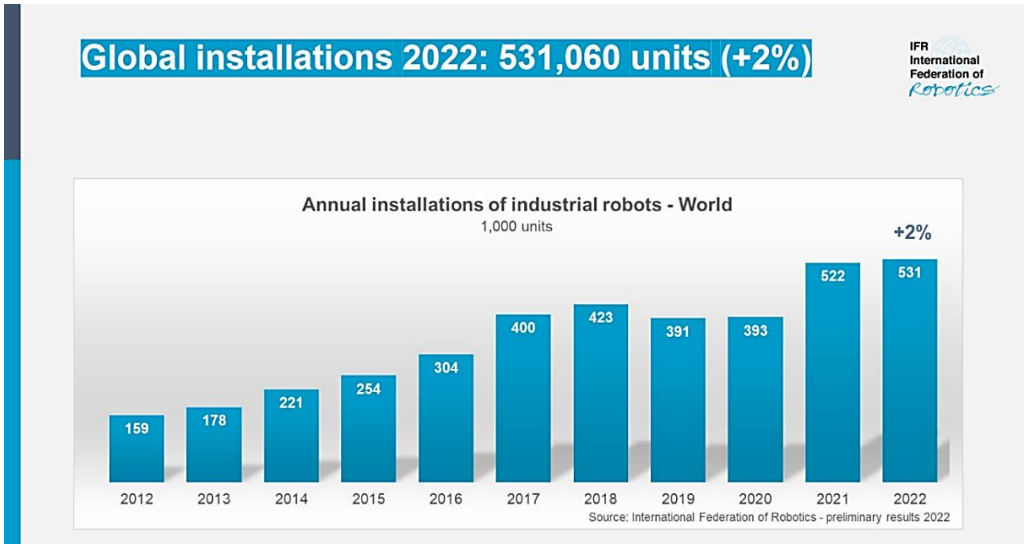

Figura 2. *Instalaciones anuales de robots industriales en el mundo* [1].

Los sistemas VRG otorgan a los robots la capacidad de visión, lo cual proporciona un mayor grado de libertad a la robótica, al permitir operar en entornos más versátiles sin la necesidad de trabajar en espacios predefinidos o depender de ubicaciones precisas preestablecidas para determinados elementos o piezas. Una de las funciones más destacadas dentro de los sistemas VRG es el ***Bin Picking*** [2] que permite la selección y recolección de piezas amontonadas aleatoriamente en un contenedor, utilizando para ello un sistema de visión que reconozca y localice la pieza, y un sistema robótico para su extracción y posterior ubicación, aunque presenta dificultades para recoger la totalidad de las piezas situadas en el contenedor. Dichos sistemas emplean cámaras de visión 3D y pinzas diseñadas especialmente para cada producto por lo que la implementación de tales sistemas acarrea costes significativos.

1.1. Motivación

En la actualidad, nos encontramos en la era de la inmediatez, donde la rapidez y la eficiencia son aspectos fundamentales en cualquier acción que se lleve a cabo. Es precisamente en este contexto donde surge la necesidad del desarrollo del presente proyecto.

La producción de componentes electromagnéticos constituye una industria crítica en la actualidad, ya que dichos componentes se emplean en una amplia variedad de aplicaciones, desde dispositivos electrónicos personales hasta sistemas de transporte y energía.

Por otro lado, el empaquetado preciso y eficiente de estos componentes puede garantizar una reducción de los costes de la empresa, permitiendo un ahorro significativo. Es por ello, que se opta por integrar la visión artificial al brazo robótico con el propósito de realizar la detección de piezas que posteriormente serán recogidas y colocadas con la orientación adecuada en su correcta ubicación. Dicha aportación resulta en una reducción de costes y una mayor flexibilidad al incrementar la autonomía de los robots al dotarlos de visión. Estos sistemas, al prescindir de la necesidad de trabajar en un espacio altamente controlado, con ubicaciones

precisas previamente establecidas, pueden omitir el uso de sistemas de colocación de piezas altamente precisos, lo cual conlleva una reducción de costes.

En definitiva, la incorporación de la visión en los robots brinda versatilidad, una reducción de costes y una menor necesidad de intervención humana en los procesos, ya que los sistemas de visión permiten a los robots operar con mayor autonomía y flexibilidad, lo cual otorgar una mayor precisión y eficiencia en los procesos de producción.

1.2. Objetivo

El objetivo del presente proyecto es desarrollar un sistema de detección y clasificación de piezas mediante visión y computación acelerada que disminuya considerablemente el tiempo de detección respecto a los sistemas convencionales, quedando como factor limitante los movimientos del robot. Se parte de un concepto de sistema clasificador de piezas que emplea visión convencional y un brazo manipulador encargado de realizar la tarea de recoger las piezas y depositarlas en un contenedor.

El sistema desarrollado en el presente proyecto estará constituido por dos redes neuronales, una para la detección de piezas y otra para la detección de la cara superior e inferior de la pieza 3DC11LP, además de la programación necesaria que satisfaga los requisitos y criterios previamente establecidos, donde se incluyen los movimientos requeridos por el brazo robótico para la recogida de las piezas detectadas y posterior colocación en un blíster, el criterio de recogida de las piezas, la orientación de estas, etc.

Por otra parte, el proyecto incluirá el desarrollo de un rastreador de piezas a fin de observar la rapidez de detección de las piezas mediante el uso de la visión y computación acelerada. Esto permitirá realizar el proceso de detección en un tiempo significativamente menor en comparación con los sistemas de visión convencionales y reconocer también posibles factores limitantes en cuanto al tiempo de detección de las piezas y la velocidad de los movimientos ejecutados por el brazo robótico empleado. Asimismo, con vistas a realizar movimientos articulares en el proceso del rastreo, se aplica la función de cinemática inversa para obtener la posición de las articulaciones del brazo robótico que se requieren para alcanzar la posición destino. De este modo dejarán los movimientos del brazo de ser prácticamente un factor limitante en términos de tiempo en dicha aplicación, pudiendo evaluar adecuadamente de esta manera la velocidad de funcionamiento del sistema.

Finalmente, se incluye una cinta transportadora al clasificador de piezas con el fin de integrar el sistema rastreador en dicha aplicación.

Todo ello, estando la cámara incorporada en el extremo del brazo robótico para disponer de mayor precisión de las imágenes cuando se requiera. En lo que respecta al aspecto de visión, los objetivos principales son:

- En primer lugar, localizar la ubicación exacta de las piezas 3DC11LP y 3DC14EMR-ULP en el blíster para posteriormente depositarlas adecuadamente. Esto es realizado a una altura determinada.

- Reconocer la pieza más cercana al centro de la imagen e identificar el centro de la pieza seleccionada. Esto es realizado a una altura determinada, diferente a la anterior. Por lo que el sistema consta de dos alturas diferentes en las que es necesario obtener la relación entre los píxeles de la imagen y los milímetros en la realidad. Dichas alturas fueron escogidas tras probar empíricamente varias y comprobar que las alturas seleccionadas eran adecuadas para el objetivo del proyecto.
- Identificar el tipo de pieza (3DC11LP o 3DC14EMR-ULP) y en caso de ser identificada la pieza 3DC11LP, detectar si está posicionada en el plano por su cara superior o inferior.
- Determinar la orientación de la pieza situada en el plano, en caso de detectar la pieza 3DC14EMR-ULP o la cara superior de la pieza 3DC11LP.

En términos de la parte de robótica, el controlador del robot debe ser capaz de:

- Permitir que el robot realice los movimientos adecuados para la recogida y posicionado de las piezas.
- Realizar los movimientos rotatorios necesarios de la articulación J6 del brazo robótico para una correcta orientación de la pieza a la hora de su posicionado en el blíster.
- Poner en marcha la cinta transportadora según sea necesario.

De este modo, el sistema presenta la capacidad de:

- detectar la pieza más cercana al centro de la imagen,
- identificar su centro,
- reconocer su cara superior o inferior (en el caso de la pieza 3DC11LP),
- determinar su orientación,
- recogerla y,
- depositarla en el blíster adecuado con la orientación apropiada o, en caso de tratarse de la cara inferior de la pieza 3DC11LP, desecharla a un contenedor para que posteriormente sea reubicada en el plano y volver a realizar el proceso de detección y clasificación con el fin empaquetarla adecuadamente.

Esta mejora en el proceso de automatización es especialmente útil en líneas de producción que requieran de la identificación de múltiples tipos de piezas o herramientas, ya que permite reducir drásticamente los tiempos de detección, lo que a su vez aumenta la eficiencia del proceso productivo.

2. Estado del arte

En busca de alcanzar el objetivo de disminuir notoriamente el tiempo empleado en la detección de piezas, es necesario el uso de redes neuronales, con el fin de que, tras un entrenamiento, puedan identificar las piezas en tiempo real.

En el presente apartado, por tanto, se procede a hacer un repaso histórico de la evolución de las redes neuronales y de su importancia en la actualidad, así como de las diferentes tecnologías de procesamiento acelerado por GPU. Además, se va a mostrar, entre otros temas, el desarrollo y la labor de los robots colaborativos en la industria.

2.1. Inteligencia Artificial

Las Redes Neuronales Artificiales, ANN (*Artificial Neural Networks*) [3], se inspiran en el funcionamiento de las redes neuronales biológicas del cerebro humano y manifiestan cualidades similares. Por ejemplo, que, por medio de un proceso de aprendizaje, "aprenden" de la experiencia a partir de ejemplos previos extrapolándolo posteriormente a ejemplos nuevos, teniendo la capacidad de abstraer las características principales de una serie de datos. Aunque existen diferentes tipos de redes neuronales, el presente proyecto se centra en las redes neuronales convolucionales que se describen en el apartado *3.1 Redes Neuronales Convolucionales (CNN)*.

Desde 1936 con Alan Turing, quien estudio por primera vez el cerebro como forma de ver el mundo de la computación y desde 1943, cuando el neurofisiólogo Warren McCulloch y el matemático Walter Pitts (los primeros teóricos en concebir los fundamentos de la computación neuronal) lanzaron una teoría sobre la forma en la que trabajan las neuronas, hasta el día de hoy, las redes neuronales han experimentado un constante desarrollo. A continuación, se mencionan los hechos más destacados.

Entre 1957 y 1958, el científico Frank Rosenblatt publicó lo que hasta la fecha fue el mayor trabajo de investigación en computación neuronal. Su trabajo consistió en el desarrollo de un elemento llamado **Perceptrón,** el cual es considerado como la red neuronal más antigua. Este consiste en un sistema clasificador de patrones capaz de identificar patrones geométricos y abstractos. Posteriormente, en 1959 Bernard Widrow y Marcian Hoff desarrollaron un elemento llamado **Adaline** (Adaptative Linear Neuron) con el fin de reconocer patrones binarios y **Madaline**, la cual fue la primera red neuronal aplicada a un problema real. Todo ello dio paso a la aparición del **Perceptrón Multicapa** en 1965, que consiste en una Red Neuronal Artificial compuesta por múltiples capas que le otorgan la capacidad de solucionar problemas más complejos e identificar patrones no lineales [3].

Por otro lado, en 1969 se produjo una fuerte crisis con relación a las Redes Neuronales que frenó en gran parte su desarrollo hasta el año 1982. Este hecho se debe a Minsky y Papert, pertenecientes al Laboratorio de Investigación de Electrónica del MIT (*Massachusetts Institute Technology*), que comenzaron un trabajo de crítica hacia el Perceptrón el cual dio como resultado el libro *Perceptrons*, que consistía en un análisis matemático del concepto del Perceptrón [3].

Para encontrarnos con el siguiente gran avance es necesario trasladarse al año 1986, año en el que fue publicado el trabajo "*Learning representations by back-propagating errors*" por David Rumelhart, Geoffrey Hinton y Ronald Williams. En dicho trabajo, los autores explican y formalizan el concepto básico de *Backpropagation* que fue previamente presentado en 1974 por Paul Werbos y reinventado en 1982 por David Parker [3].

En el año 1989 aparece la primera **Red Neuronal Convolucional (CNN)** creada por Yann LeCun y ya en el nuevo milenio, concretamente en 2006, gracias a Geoffrey Hinton y su equipo tuvo lugar el comienzo del ***Deep Learning*** [4] o aprendizaje profundo que consiste en un tipo de *Machine Learning* relacionado con algoritmos inspirados en la estructura cerebral humana y redes neuronales artificiales, haciendo uso de redes neuronales multicapa y un gran conjunto de datos etiquetados para llevar a cabo el aprendizaje. A diferencia de las redes anteriores, esta utiliza una asignación inteligente de pesos mediante un preentrenamiento de las capas de la red de manera no supervisada.

En el año 2012 el informático canadiense Alex Krizhevsky junto a su equipo introdujo una red neuronal convolucional profunda llamada **AlexNet**. Y dos años más tarde, en el 2014, Ian Goodfellow un ingeniero, informático y ejecutivo estadounidense, propuso las **redes generativas adversativas (GANs debido a sus siglas en inglés, *Generative Adversarial Networks*)** que consisten en una clase de algoritmos de inteligencia artificial empleados en el aprendizaje no supervisado que permite la generación de imágenes realistas y datos sintéticos por medio de un sistema de dos redes neuronales [5].

Por otro lado, el año 2016 representa un punto de inflexión en cuanto a las capacidades de las redes neuronales, ya que el programa informático **AlphaGo** desarrollado por Google DeepMind logró ganar en el juego de mesa Go a uno de los mejores jugadores profesionales de dicha disciplina, concretamente a Lee Sedol [5].

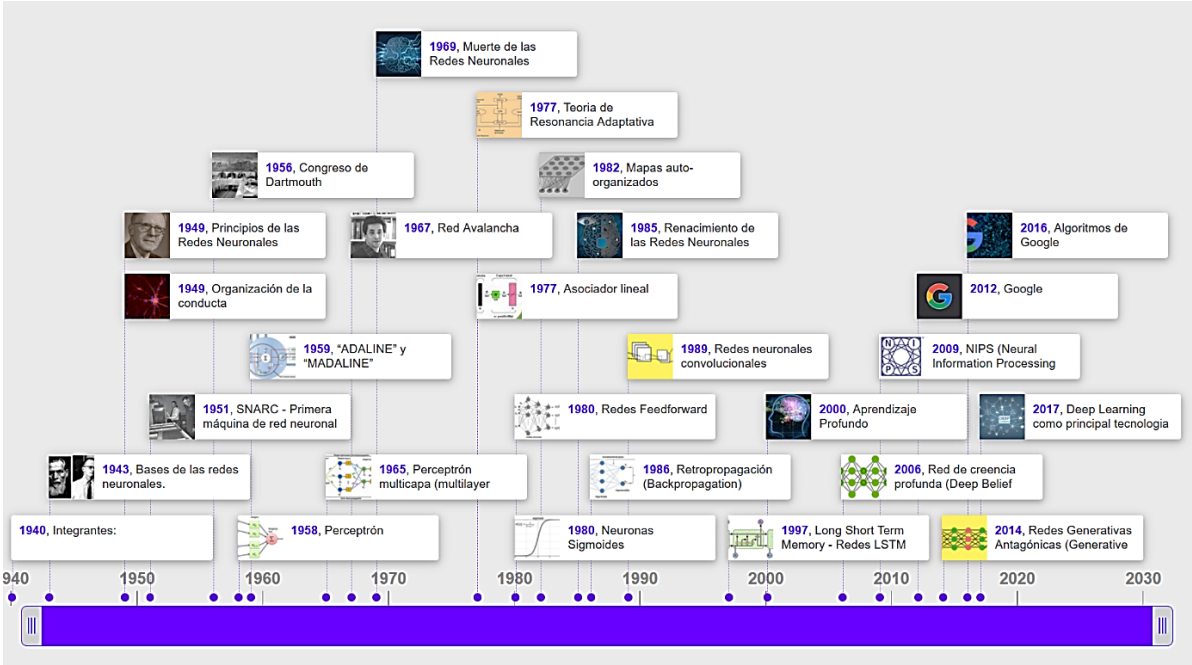

Figura 3. *Eje cronológico de la evolución de las redes neuronales hasta 2017 [5].*

En los últimos años, los avances de las redes neuronales en diversas áreas han resultado extraordinarios. Por un lado, en el campo de la medicina se han empleado las redes neuronales para el diagnóstico de enfermedades como el Alzheimer y el cáncer de piel. Por otro lado, en el área de la visión por computadora se han alcanzado grandes progresos en cuanto a la clasificación y detección de objetos tanto en imágenes como en videos. A medida que las redes neuronales han evolucionado en el tiempo, se percibe un uso creciente de autoencoders en diversas aplicaciones donde, entre otras, destaca su papel en la detección de anomalías. A su vez, se han logrado mejoras en el campo del procesamiento del lenguaje, posibilitando la generación de conversaciones entre robots y humanos, y la traducción automática de textos. Las *Transformers Networks*, como GPT-3, que consisten en un tipo de arquitectura de red neuronal desarrollada en el artículo "*Attention is All You Need*" publicado en el año 2017 por empleados de Google han revolucionado el procesamiento del lenguaje dando lugar a una gran diversidad de textos dotados de coherencia [6].

2.1.1. Redes neuronales convolucionales (CNN)

Las **redes neuronales convolucionales (CNN debido a sus siglas en inglés, *Convolutional Neural Network)*** [7] consisten en una clase de arquitectura de redes neuronales profundas donde las "neuronas" se organizan en campos receptivos de forma muy similar a como lo hacen las neuronas en la corteza visual primaria (V1) del cerebro. Dichas redes son desarrolladas principalmente para el procesamiento y análisis de datos estructurados, como por ejemplo imágenes tomando estas como entradas, asociándole importancias (pesos) a determinados elementos dentro de la imagen a fin de distinguir unos de otros.

Las CNN aprenden a identificar objetos en las imágenes gracias a un entrenamiento previo con una cantidad importante de "muestras", es decir, la red va a ser capaz de reconocer un elemento determinado gracias a que lo ha "visto" anteriormente una gran cantidad de veces.

El procedimiento de aprendizaje de las redes neuronales convolucionales comienza del siguiente modo: las CNN procesan las imágenes tomando sus píxeles como entrada. En caso de tratarse de una imagen en blanco y negro, sería necesario emplear 784 neuronas si se tratase de una imagen de 28x28 píxeles de alto y ancho. Sin embargo, si la imagen es a color es necesario el uso de tres canales RGB (*red, green, blue*) por lo que serían necesarias para una imagen de 28x28 píxeles 2352 neuronas (28x28x3) en este caso. Estas neuronas constituyen por tanto la capa de entrada.

A continuación, resulta de especial utilidad realizar la normalización de los datos de la capa de entrada, procesando los valores de los píxeles dividiéndolos entre 255 con el fin de que se encuentren en un rango de 0 a 1.

Seguidamente, se continúa hacia el proceso distintivo de las redes neuronales convolucionales, las convoluciones. Dicho proceso consiste en tomar grupos de píxeles adyacentes de la imagen de entrada con el fin de realizar el producto escalar contra una pequeña matriz denominada kernel, por lo que se le estaría dando con este procedimiento importancia a la posición de los píxeles dentro de la imagen. Por otro lado, el kernel en las redes neuronales convolucionales es

considerado como el filtro aplicado a una imagen con el fin de extraer las características importantes o los patrones de esta [8].

El kernel se desplaza por la imagen de modo que, al aplicar el producto escalar, genera una nueva matriz de salida la cual representa una capa de neuronas ocultas. En caso de tratarse de una imagen a color, sería necesario aplicar tres kernels, uno por cada canal de color, los cuales se suman para formar una única salida. Cada uno de los filtros o kernels, extraerán unas determinadas características de la imagen que serán reflejadas en el mapa de características correspondiente.

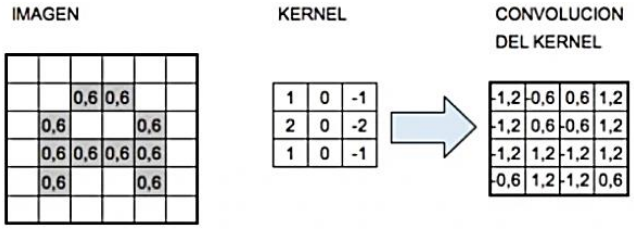

Figura 4. *Operación de convolución [7].*

En definitiva, llegados a este punto, las redes neuronales convolucionales procesan imágenes en grupos de píxeles y aplicando convoluciones por medio de filtros se extraen características específicas de la imagen, como bordes o texturas. Cada filtro o kernel, dependiendo de sus parámetros de configuración, destaca diversos aspectos de la imagen, permitiendo que la red identifique objetos, texturas, bordes y patrones.

2.1.1.1. Arquitectura convolucional

A continuación, se presentan las capas que conforman la arquitectura de una red neuronal convolucional de clasificación, ya que se trata de la arquitectura de mayor relevancia en el contexto de este proyecto. Las capas se dividen inicialmente en una capa de entrada, un conjunto de capas ocultas y una capa final de salida [9]:

- **Capa de entrada:** esta capa recibe la imagen de entrada que puede ser a color (canales RGB) o en blanco y negro.
- **Capas ocultas:** las capas ocultas hacen referencia a las capas intermedias de una red neuronal, las cuales se encuentran entre la capa de entrada y de salida. Dichas capas resultan esenciales para el procesamiento y la transformación de los datos de entrada.

  Las capas ocultas en una CNN incluyen:

  - **Capa convolucional:** la presente capa aplica los filtros (kernels) a la imagen de entrada con objeto de como se ha comentado anteriormente, destacar diferentes características de la imagen.
  - **Capa de activación:** tras cada capa convolucional, se aplica una función de activación como por ejemplo **ReLU (*Rectified Linear Activation*)**, $max(0, x)$.

Dicha función, presenta una clara ventaja frente a la **función sigmoide** y es que no sufre del problema de degradado de fuga y, por lo tanto, el aprendizaje puede llegar a ser más eficaz.

- **Capa de agrupación:** estas capas presentan la capacidad de reducir la dimensionalidad del mapa de características al muestrear y seleccionar las regiones de mayor importancia. Es posible destacar la técnica del Max-Pooling la cual consiste en dividir la entrada en un conjunto de regiones que no se superpongan, generando el valor máximo de activación para cada región en todas las profundidades de modo que, al condensar cada región en un solo punto, se reduce el tamaño de la imagen.

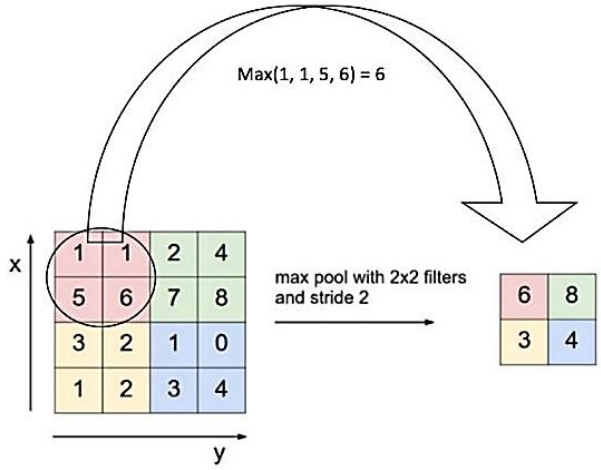

Figura 5. *Técnica Max-Pooling [10].*

En el ejemplo anterior, se aplica Max-Pooling de 2x2 consiguiendo de este modo reducir la salida a la mitad.

- **Capa densa (Fully Connected Layer)**: tras varias capas convolucionales, de activación y Max-Pooling, es necesario una capa en la que todas las neuronas estén conectadas a todas las neuronas de la capa anterior con el fin de combinar los aspectos extraídos de las capas anteriores y emplearlos en la toma de decisiones.

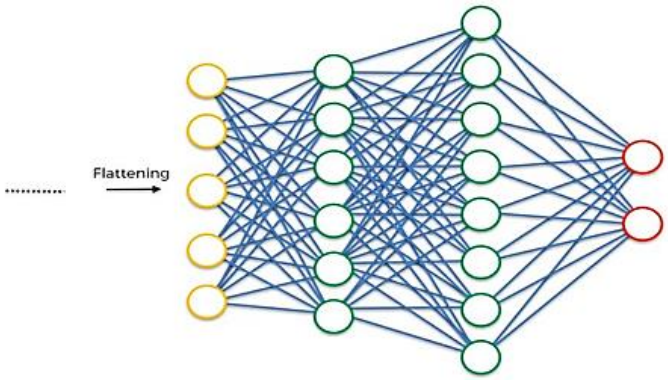

Figura 6. *Capa densa de una CNN [10].*

- **Capa SoftMax:** finalmente, se encuentra la capa donde se aplica una función llamada SoftMax, empleada normalmente en problemas de clasificación, cuya labor consiste en transformar las salidas de la red en una distribución de probabilidades. Esto permite identificar el tipo de clase que presenta mayor probabilidad para una entrada determinada.

- **Capa de salida:** la capa de salida devuelve la predicción final de la red, que puede ser una clasificación, una máscara en el caso de segmentación semántica o un valor de regresión.

La imagen que se presenta a continuación de una **CNN de clasificación** puede resultar de gran utilidad para profundizar en la comprensión y facilitar la visualización del contenido y los conceptos de una CNN que se han expuesto anteriormente.

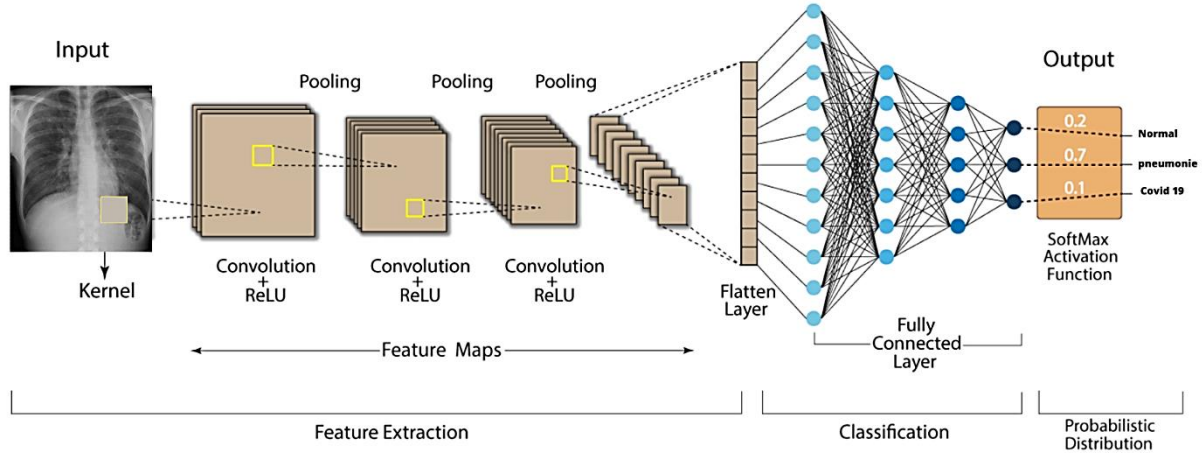

Figura 7. *Arquitectura de una CNN [11].*

2.2.Tecnologías de Procesamiento Acelerado por GPU

En la actualidad, en el corazón de la revolución tecnológica, las tecnologías de procesamiento de datos han evolucionado, permitiendo un procesamiento más rápido y eficiente en una amplia gama de aplicaciones. A continuación, se habla de la computación acelerada por GPU y CUDA:

2.2.1. Computación acelerada por GPU

La computación acelerada por GPU [12] consiste en el empleo de una unidad de procesamiento gráfico (GPU) junto a una unidad central de procesamiento (CPU) con el propósito de acelerar la velocidad y mejorar el rendimiento de diversas aplicaciones como por ejemplo aquellas vinculadas con el aprendizaje profundo, el análisis y la ingeniería.

Dicha técnica fue impulsada por la empresa NVIDIA, quienes en torno al año 2007 introdujeron los aceleradores de GPU, año en el que CUDA impulsó una nueva línea de las GPU de NVIDIA que extendió la computación acelerada a una amplia gama de aplicaciones científicas e industriales [13].

La aceleración de las aplicaciones de software es posible debido a que, por medio de la computación acelerada por GPU, las tareas más exigentes en términos de cálculos son asignadas a la GPU, mientras que el resto del código es ejecutado en la CPU.

Con el fin de simplificar este concepto, resulta útil conocer las diferencias entre la CPU y la GPU a la hora de procesar las tareas. Por un lado, mientras que la CPU dispone de un número pequeño de núcleos optimizados para realizar tareas secuencialmente, la GPU por otro lado, dispone de una arquitectura en paralelo con un gran número de núcleos más pequeños y eficaces capaces de resolver múltiples tareas al mismo tiempo, por lo que la GPU presenta una capacidad de procesamiento en paralelo muy superior a la de la CPU.

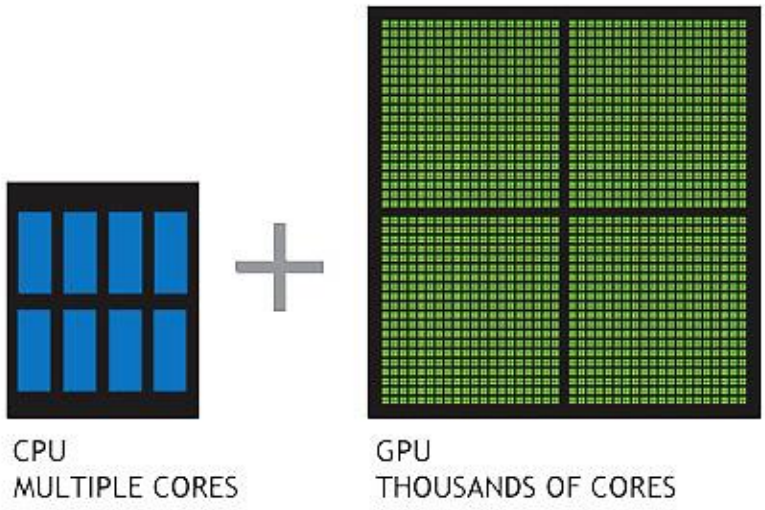

Figura 8. *Computación acelerada por GPU [12].*

Para concluir el presente apartado, se podría decir en definitiva que la computación acelerada por GPU se beneficia de la potencia aportada por la GPU para realizar cálculos y tareas de alta intensidad en paralelo, lo cual permite acelerar significativamente la velocidad y por tanto aumentar el rendimiento de diversas aplicaciones.

2.2.2. CUDA

CUDA™ son las siglas de ***Compute Unified Device Architecture*** [14] que hacen referencia a una arquitectura de cálculo paralelo desarrollada por NVIDIA, la cual aprovecha los beneficios aportados por la GPU con el fin de incrementar el rendimiento y la velocidad del sistema. Es por ello por lo que aprovecha los beneficios que brindan las GPUs frente a las CPUs, haciendo uso del paralelismo que surge de sus múltiples núcleos, los cuales permiten ejecutar simultáneamente un número elevado de hilos. Su primera versión del kit de desarrollo (SDK) fue lanzada en febrero del año 2007, siendo solamente compatible con sistemas Linux y Windows, aunque posteriormente, cuando se lanzó la versión 2.0 se extendió CUDA también al sistema operativo Mac OS.

El procesamiento CUDA procede del siguiente modo:

1. En primer lugar, los datos contenidos en la memoria principal son copiados en la memoria GPU.
2. La CPU se encarga de ordenarle el proceso pertinente a la GPU.
3. La GPU ejecuta el proceso en paralelo en sus múltiples núcleos.
4. Finalmente, el resultado obtenido tras realizar el proceso en la GPU es copiado en la memoria principal.

Por otro lado, comparado con otras clases de computación en GPU que emplean APIs gráficas, CUDA ofrece numerosas ventajas, a parte de las ya mencionadas, gracias a sus características. A continuación, se exponen las ventajas ofrecidas por CUDA:

- Es posible realizar **lecturas dispersas**, es decir, permite acceder a cualquier posición de la memoria.
- Posibilita **transferencias de datos más rápidas** desde y hacia la GPU.
- Brinda al programador una sección de **memoria compartida** entre hilos (*threads*) que debido a su tamaño rapidez puede ser empleada como memoria caché.
- Permite trabajar con **operaciones a nivel de bit y enteros.**

Sin embargo, pese a sus ventajas, también presenta ciertas limitaciones como:

- No admite el renderizado de texturas.
- En modo de precisión simple CUDA no soporta ni números desnormalizados ni NaNs (*Not-a-Number*).
- Presenta restricciones en la programación a la hora de utilizar la recursividad, variables estáticas dentro de funciones, etc.
- Para una mayor eficiencia es recomendable trabajar con miles de hilos en total.

- Es posible que en ciertas ocasiones se produzcan cuellos de botella entre la CPU y la GPU debido a los anchos de banda de los buses y sus latencias.

2.2.2.1. Arquitectura CUDA.

En la arquitectura clásica de las GPUs es posible encontrar dos clases diferentes de procesadores: los procesadores de vértices y los de fragmentos, los cuales se encargan de tareas diferentes. Este hecho puede dar lugar a un desequilibrio de carga entre ambos procesadores y a la disparidad entre sus repertorios de instrucciones. En consecuencia, la inminente necesidad en la evolución de la arquitectura de las tarjetas gráficas con miras de alcanzar una arquitectura unificada en la que no exista distinción entre ambos tipos de procesadores dio lugar a la llegada de la arquitectura CUDA, en la que todos sus núcleos utilizan recursos muy similares y el mismo repertorio de instrucciones [15].

En la ***Figura 9*** se aprecia la arquitectura de una tarjeta gráfica compatible con CUDA, concretamente la arquitectura FERMI de NVIDIA, en la que es posible observar ocho unidades de ejecución llamadas *Streaming Multiprocessors* (SM), las cuales se encuentran interconectadas entre sí por medio de una zona de memoria común. Dichas unidades a su vez se componen de unos núcleos de cómputo, encargados de ejecutar las instrucciones, denominados *Streaming Processors* (SP) o núcleos CUDA, 32 núcleos por cada SM lo que significa un total de 256 núcleos de procesamiento en el presente caso. Este tipo de diseño de hardware facilita en gran medida la programación de los núcleos de la GPU empleando un lenguaje de alto nivel.

Figura 9. *Seis equipos de GPUs con arquitectura FERMI de NVIDIA [15].*

Aunque en el texto se refleja únicamente la arquitectura FERMI, NVIDIA ha desarrollado a lo largo de los años múltiples arquitecturas como, por ejemplo: la arquitectura Kepler, Maxwell, Pascal, Volta y Turing.

Por todo lo comentado anteriormente, en la actualidad, CUDA destaca como un enfoque líder en la programación de GPUs, posibilitando el acceso a la potencia de procesamiento masivo de dichas unidades y logrando una alta eficiencia del sistema.

2.3.Robots colaborativos en la industria

La **industria 4.0** o también conocida como **Cuarta Revolución Industrial**, hace referencia a una transformación tecnológica y digital de la industria en la que los dispositivos se comunican de forma autónoma entre ellos a lo largo de la cadena de valor, permitiendo a los dispositivos interactuar entre ellos dando lugar a la vinculación entre el mundo físico y el digital. Dicha etapa es sustentada por las siguientes tecnologías [16]:

- **Robots autónomos:** cada vez los robots se están tornado más flexibles, cooperativos y autónomos. Esto les permite interactuar entre ellos, trabajar de manera segura junto a los humanos e incluso aprender de ellos.
- **Big data y análisis:** hace referencia al análisis de conjuntos de datos caracterizados por su volumen, velocidad a la que tienen que ser procesados y variedad de datos estructurados y no estructurados.
- **Ciberseguridad:** cada vez es más necesario proteger de posibles amenazas informáticas los sistemas industriales y las líneas de producción debido a la evolución hacia una industria inteligente.
- **Simulación:** concede la posibilidad de representar el mundo físico en un modelo virtual, el cual puede incluir productos, máquinas, procesos y personas en tiempo real. Esto permite realizar pruebas antes de poner la máquina o el proceso en marcha, lo cual ayuda a prevenir averías y a optimizar el proceso.
- **Realidad aumentada:** se trata de sistemas que integran el modelado, la simulación y la virtualización, permitiendo complementar el mundo real con elementos virtuales.
- **Internet de las cosas (Internet of things, IoT):** otorga una mayor facilidad en la toma de decisiones en base a la información recogida del entorno por la tecnología. Además, permite una comunicación multidireccional entre personas, máquinas y productos.
- **Sistemas de integración horizontales y verticales:** consienten la integración de tecnologías operacionales con tecnologías de la información y la comunicación.
- **Fabricación adictiva:** partiendo de un modelo virtual previo, permite la producción de elementos por medio de la superposición de capas de diferentes materiales.
- **Computación en la nube:** brinda almacenamiento, acceso y uso de servicios informáticos en línea.

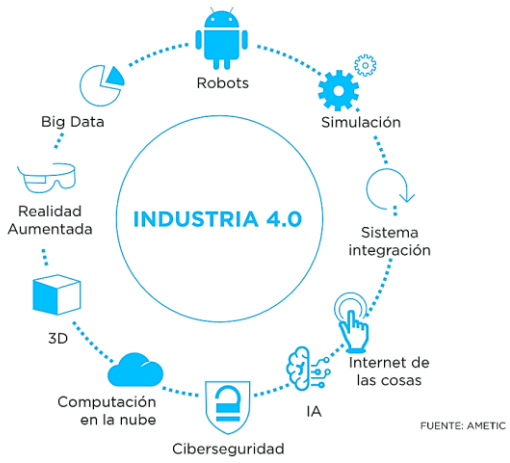

Figura 10. *Tecnologías fundamentales de la Industria 4.0 [16].*

- **Inteligencia artificial:** en base al desarrollo de algoritmos, permite a las computadoras procesar datos a una velocidad inusual y concede a las máquinas la capacidad de comprender, percibir, actuar y aprender automáticamente.

La **Industria 4.0** [17] que representa una revolución tecnológica, conlleva por tanto significativos avances en el mundo de la robótica por el que las industrias han cambiado la forma en la que afrontan sus procesos productivos.

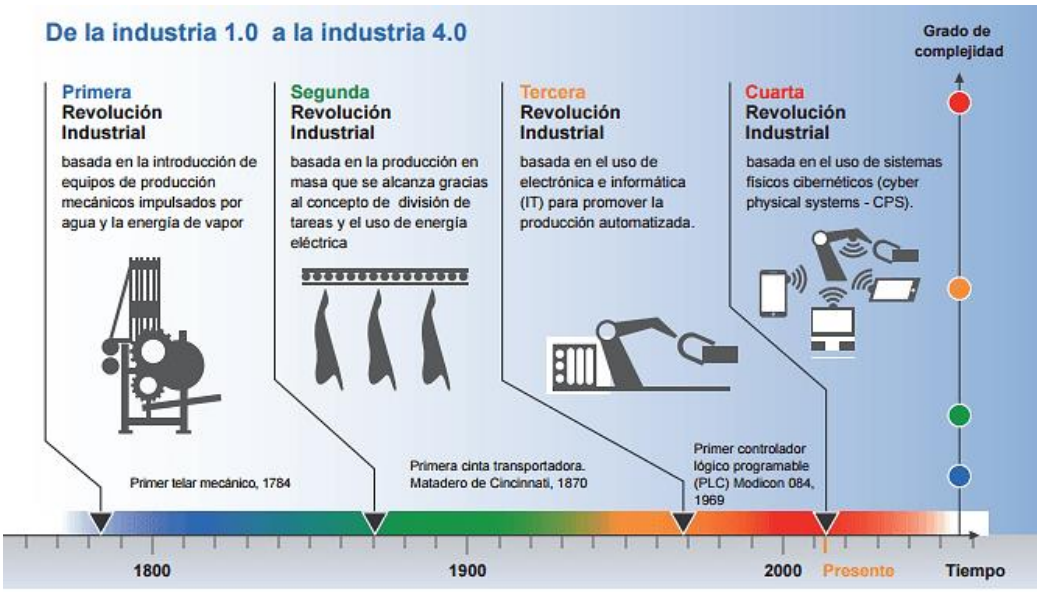

Figura 11. *De la Industria 1.0 a la industria 4.0 [17].*

En este contexto, los **robots colaborativos**, también llamados *cobots*, han emergido como una solución relevante que presenta numerosas ventajas en contraste con los robots tradicionales, y su incorporación dentro del marco de la Industria 4.0 potencia aún más sus beneficios.

Los robots colaborativos [18] son aquellos robots que permiten fusionar la productividad de los sistemas automatizados con la destreza y flexibilidad de los manuales, lo cual les permite operar en colaboración con los humanos con total seguridad, asistiéndoles en diversas tareas y procesos. Es por ello por lo que la presencia de los *cobots* es cada vez más común en los sistemas de producción modernos.

2.3.1. Capacidades de los robots colaborativos

Los robots colaborativos han sido desarrollados con características particulares que los diferencian de los tradicionales.

Aunque la interacción directa entre el robot y el humano pueda ser a su vez la mayor ventaja o un factor limitante de los sistemas colaborativos, dependiendo de cómo afecten los factores humanos como el estrés mental y la ergonomía al proceso, los *cobots* presentan las siguientes capacidades [18]:

- **Adaptabilidad:** presentan la capacidad de ajustarse o responder de manera flexible a diversas situaciones o entornos.
- **Seguridad:** presentan la capacidad de trabajar conjuntamente en armonía con operadores, sin riesgo para la salud física y mental del operario.
- **Movilidad:** presentan la capacidad de aportar facilidades a la hora de mover el *cobot* por la planta de producción.
- **Conectividad:** presentan la capacidad de comunicarse con los operadores y otros robots dentro del entorno de trabajo, recopilando y proporcionando a su vez información.
- **Actuación:** presentan la capacidad de desarrollar trayectorias de manera fluida y segura.

La convergencia de los factores mencionados anteriormente ha incrementado significativamente el número de robots colaborativos implantados en la industria en comparación con los robots tradicionales. Todo ello debido en gran parte a la sinergia de las demandas cambiantes de la producción, los grandes avances tecnológicos y los beneficios aportados por la colaboración entre máquinas y robots.

3. Herramientas

En el presente capítulo se reúnen las herramientas tanto de hardware como de software empleadas en el desarrollo del proyecto. En el apartado de hardware se muestran los dispositivos y/o recursos físicos y tangibles que han sido necesarios para llevar a cabo dicho proyecto. Por otro lado, en el apartado de software se exponen los distintos programas empleados a lo largo del desarrollo, así como el entorno de programación, las librerías y aplicaciones empleadas.

3.1. Hardware

Con la finalidad de llevar a cabo el objetivo del proyecto se emplea el siguiente soporte físico:

- **Portátil:** en primer lugar, para llevar a cabo la comparación del rendimiento en la detección de anomalías con y sin el uso de una unidad de procesamiento gráfico (GPU), llevada a cabo en el punto 4.1.1, se emplea un portátil con GPU integrada, concretamente un **portátil MSI** con 16 gigabytes (GB) de RAM y GPU **NVIDIA GeForce GTX 1650 Ti with Max-Q Design**.
- **Brazo robótico:** con el propósito de recoger las piezas detectadas y posteriormente depositarlas en su blíster correspondiente, se emplea un robot colaborativo de seis grados de libertad (*xArm 6*) desarrollado por *UFactory* y distribuido en Europa por *PREMO Robotics* como *pArm6* [19]. Este posee una carga útil de 5 kilogramos (kg) y una repetibilidad de ± 0,1 milímetros (mm), siendo su peso de 12 kg y poseyendo un alcance de hasta 700 mm.

Figura 12. *Brazo robótico colaborativo de seis grados de libertad pArm6 [19].*

Se seleccionó dicho brazo robótico debido a su disponibilidad en la empresa PREMO [20] ubicada en Málaga, además de ofrecer un rendimiento estándar que lo iguala a otros brazos robóticos de 6 ejes a un menor precio.

- **Efector de vacío:** el brazo robótico mencionado en el párrafo anterior dispone de una ventosa como efector final. Esto es posible gracias al efector de vacío desarrollado por *UFactory* y distribuido por *PREMO Robotics* con el nombre *pArm6 Vacuum Gripper* [21]. Este posee un suministro de aire incorporado capaz de soportar una carga útil de hasta 5 kg siendo su peso de 610 gramos (g).

Figura 13. *Efector de vacío pArm6 Vaccum Gripper [21].*

- **Ventosa:** con el fin de recoger piezas de pequeño tamaño, se requiere el uso de una ventosa adaptada a dichas condiciones. Para ello, se emplea un soporte para la ventosa adecuado a sus dimensiones. El vacío se realiza gracias a un tubo neumático de 6 mm de diámetro exterior que se encuentra conectado en un extremo al efector de vacío y en el otro al soporte de la ventosa por medio de un racor en ambos casos.

  Debe tenerse en cuenta el uso de arandelas siempre que sea posible, ya que esto evita pérdidas de aire, mejorando con ello la succión de las piezas.

Figura 14. *Sistema de succión instalado.*

- **Cámara Web y soporte:** para realizar la detección de piezas por medio de la visión artificial se emplea una cámara **Logitech C210** [22] con una resolución de video de 640x480 píxeles, es decir, 0,3 megapíxeles (Mpx), además, de presentar una resolución de captura de imágenes de hasta 1,3 Mpx y una velocidad de captura de 30 *Frames* por segundo (FPS). Presenta una longitud del cable de 1,5 metros (m) y un peso de 227 g.

Figura 15. *Webcam Logitech C210 [22].*

Se seleccionó dicha cámara debido a su bajo coste y fácil conexión mediante USB, además de que el objetivo del proyecto es realizarlo por medio de una cámara de baja resolución para así observar las limitaciones que pueden surgir.

Por otro lado, con objeto de sujetar la cámara con mayor facilidad por el extremo del brazo robótico, se emplea un soporte impreso en 3D diseñado por medio del software de diseño asistido por computadora SolidWorks [23].

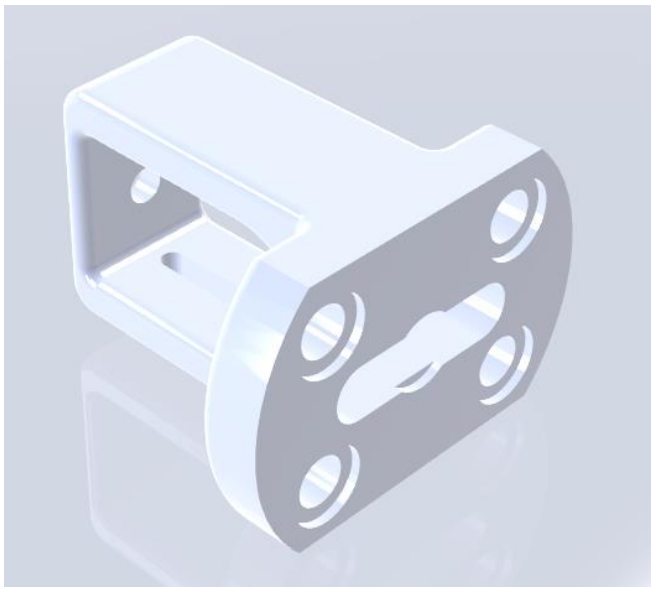

Figura 16. *Soporte de la cámara.*

Finalmente, el sistema presenta la apariencia que se muestra en la figura siguiente:

Figura 17. *Cámara sujeta al soporte.*

- **Jetson Nano con su carcasa y ventilador:** con el fin de ejecutar la red neuronal para la detección de dos clases de piezas de la empresa PREMO, y realizar la distinción entre las caras de la pieza 3DC11LP, se emplea una de las plataformas de desarrollo de NVIDIA, concretamente el kit de desarrollo de Jetson Nano. Dicho kit, está diseñado específicamente para aplicaciones de inteligencia artificial (IA) y permite ejecutar varias redes neuronales en paralelo para aplicaciones como la detección de objetos, el procesamiento del lenguaje, la segmentación y clasificación de imágenes y todo ello a un bajo precio respecto a otros modelos.

  Es conveniente añadir al kit una carcasa con la finalidad de proporcionar una mayor seguridad a la placa a la hora de sufrir posibles golpes, además de un ventilador con vistas a evitar sobrecalentamientos a la hora de la utilización de la Jetson. En el apartado ***¡Error! No se encuentra el origen de la referencia.***, se entra en detalle del por qué se selecciona el kit de desarrollo mencionado.

Figura 18. *Kit de Jetson Nano con su carcasa y ventilador.*

- **Cable de alimentación para Jetson Nano:** debido a la falta del cable de alimentación en el kit de desarrollo de Jetson Nano, se emplea uno compatible de la marca CARGADOR ESP [24] de 5V de voltaje de salida, una corriente de 4A y, por tanto, una potencia eléctrica de 20W. La clavija del cargador presenta un diámetro exterior de 5,5 mm e interior de 2,1 mm.

Figura 19. *Cable de alimentación [24].*

- **Tarjeta microSD y un lector para leerla:** se emplea una tarjeta microSD de 256 GB de la empresa SanDisk para flashear en ella la imagen del sistema operativo basado en Ubuntu. Además, debido a la carencia de un lector de tarjetas por parte del portátil empleado para la descarga del sistema operativo, es necesario emplear un lector externo de tarjetas microSD de la marca *Aqprox*.

Figura 20. *Tarjeta microSD y lector de tarjetas microSD.*

- **Piezas para detectar y blísteres donde depositarlas:** con objeto de cumplir los objetivos del proyecto, es necesario seleccionar dos tipos de piezas diferentes para así poder detectarlas por medio de la red neuronal previamente entrenada, estas piezas son la 3DC11LP (3DCoil) [25] y la 3DC14EMR-ULP (3DCoil Ultra-Low-Profile) [26] fabricadas por la empresa PREMO. Esta última presenta la característica de ser muy finita y estar moldeada, mientras que la otra pieza no.

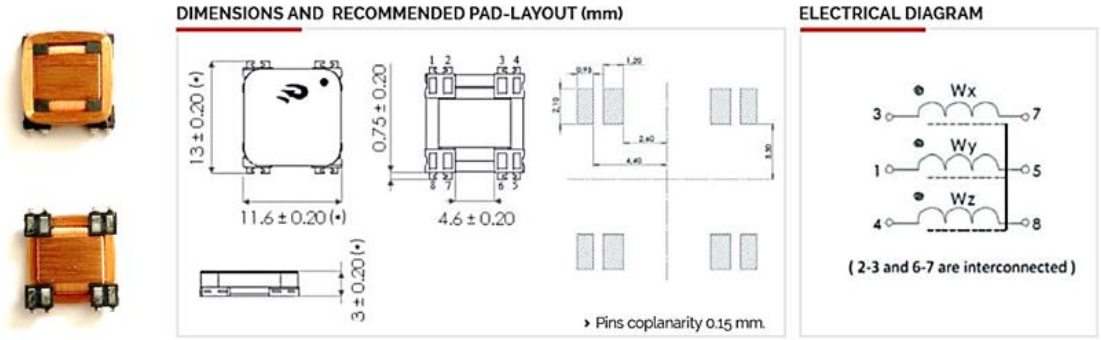

Figura 21. *Pieza 3DC11LP.*

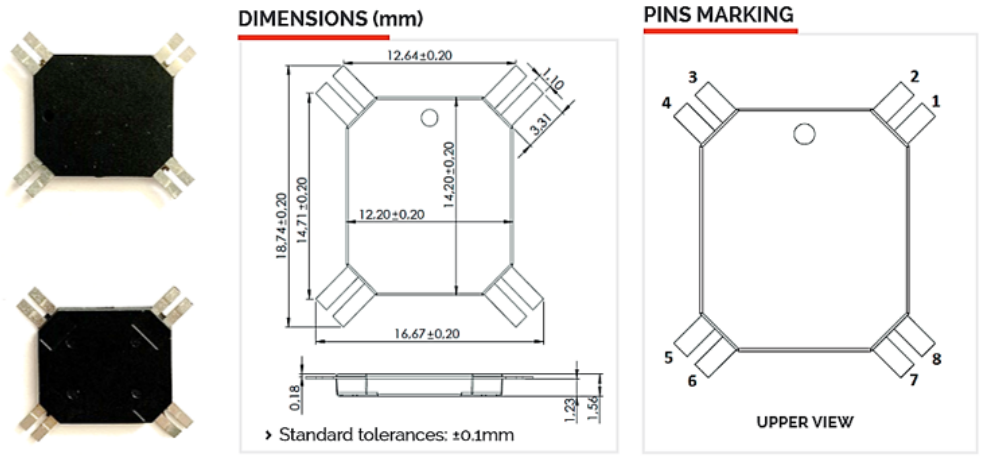

Figura 22. *Pieza 3DC14EMR-ULP.*

Para su posterior almacenamiento es necesario emplear dos blísteres, uno por cada tipo de pieza. El blíster empleado para el almacenamiento de la pieza 3DC11LP posee 8 filas y 10 columnas, siendo la distancia entre los centros de dos filas contiguas de 30,09 mm, mientras que la distancia entre los centros de dos columnas contiguas es de 31,37 mm. Mientras que el blíster empleado en el almacenamiento de la pieza 3DC14EMR-ULP posee 10 filas y 8 columnas, siendo la distancia entre los centros de dos filas contiguas de 31,37 mm, mientras que la distancia entre los centros de dos columnas contiguas es de 30,09 mm.

- **Cinta transportadora:** para llevar a cabo el seguimiento de las piezas por medio del brazo robótico a través de la visión artificial, se añade una cinta transportadora, ya existente en las instalaciones de PREMO, de velocidad regulable por medio de un potenciómetro, accionada por un motor de corriente continua de 24V con una velocidad máxima de 200 revoluciones por minuto, modelo XD-60GA775.

Figura 23. *Cinta transportadora y su motor.*

3.2.Software

A fin de llevar a cabo el objetivo del proyecto se emplea el siguiente soporte lógico:

- **MATLAB®:** como se ha comentado en el apartado 3.1. Hardware, es necesario en primera instancia realizar una comparación del rendimiento en la detección de anomalías con y sin el uso de una GPU. Para ello se emplea la versión R2022b de MATLAB® dado que el código de detección de anomalías fue desarrollado previamente en dicha plataforma en la versión mencionada, además de por sus amplias prestaciones para la creación de gráficas e histogramas de manera fácil y versátil.
- **Python™ y Visual Studio Code:** como lenguaje de programación se utiliza Python debido a que posee una amplia colección de bibliotecas además de su amplia compatibilidad con el robot colaborativo empleado. Por otra parte, para realizar la edición de código se emplea como editor de código fuente **Visual Studio Code**.
- **OpenCV y CUDA™:** con objeto de realizar la programación de visión computacional se emplea la librería de código abierto **OpenCV** [27], la cual proporciona un marco de trabajo de alto nivel para el desarrollo de aplicaciones de visión por computador en tiempo real: análisis y procesamiento de imágenes, estructuras de datos, análisis estructural, etc. Adicionalmente, al incorporar el software **CUDA** de NVIDIA con OpenCV, se obtiene OpenCV con soporte para aceleración GPU proporcionado por CUDA.
- **Paquete de desarrollo de software (SDK) para Jetson Nano:** debido a que se emplea una Jetson Nano para llevar a cabo el presente proyecto, es necesario emplear un paquete de desarrollo de software (SDK) llamado **JetPack** [28], concretamente la **versión 4.6** [L4T 32.6.1]. Este paquete incluye un sistema operativo basado en Linux, entorno de escritorio Ubuntu, kernel de Linux, un conjunto de bibliotecas para la aceleración de computación por GPU, gráficos, multimedia y visión por computadora. Además, como se observa en el repositorio [29], al emplear una versión de JetPack superior o igual a la 4.2, es posible instalar de manera opcional otra herramienta en Jetson Nano llamada **PyTorch** [30], herramienta en la que se profundiza a continuación.
- **PyTorch:** con el fin de entrenar los modelos se emplea PyTorch, concretamente la **versión 1.6.0** en el presente caso. PyTorch consiste en una librería de aprendizaje automático basada en Python, diseñada con el fin de ejecutar cálculos numéricos haciendo uso de la programación de tensores, permitiendo su ejecución en GPU para acelerar los cálculos. Los tensores consisten en estructuras de datos empleadas en la computación acelerada para representar datos de cualquier número de dimensiones, de modo que permiten realizar cálculos y operaciones matemáticas en paralelo, acelerando de este modo el rendimiento computacional, especialmente empleada en aplicaciones como la visión artificial y el procesamiento del lenguaje natural.
- **TensorRT:** con objeto de ejecutar redes neuronales optimizadas en GPU desde Python o C++ se emplea TensorRT [31] aunque previamente debe exportarse el modelo anteriormente entrenado con PyTorch a un formato compatible con TensorRT que en el presente caso se trata del formato **ONNX (Open Neural Network Exchange)** [32] conocido en español como Intercambio de Redes Neuronales Abiertas que consiste en un ecosistema de inteligencia artificial de código abierto el cual permite el intercambio de modelos entre diferentes marcos de trabajo y herramientas de aprendizaje profundo. TensorRT consiste en un entorno de ejecución y optimización de inferencia de modelos de

aprendizaje profundo de alto rendimiento en hardware NVIDIA, el cual ofrece baja latencia y elevado rendimiento en aplicaciones de inferencia.

- **Librerías:** a la hora de programar, es necesario el uso de librerías para facilitar y simplificar la programación de tareas complejas, aprovechando recursos y funcionalidades comunes previamente resueltas por otros programadores.

  Se emplean librerías como **Numpy (Numerical Python**) [33], especializada en el cálculo numérico y el análisis de datos, dado que a nivel de programación las imágenes son tratadas como matrices. **Matplotlib** [34] para la creación de gráficos en dos dimensiones. **Math** [35], la cual proporciona acceso a funciones matemáticas definidas en el estándar de C. **Time** [36] que proporciona funciones relacionadas con el tiempo. Por otro lado, las librerías **Jetson Inference** [37] y **Jetson Utils** [38], para el desarrollo de aplicaciones de aprendizaje profundo y visión por computadora en dispositivos Jetson de NVIDIA.

  Jetson Utils incluye herramientas como procesamiento acelerado por GPU, interfaz con la cámara, manipulación de imágenes y video e interfaz con el hardware del sistema, mientras que, por otro lado, *Jetson Inference* incluye inferencia de modelos de aprendizaje, análisis de videos en tiempo real, clasificación y detección de objetos, etc. *Jetson Utils* normalmente se construye como un submódulo de *Jetson Inference*, aunque también es posible que sea instalada de manera independiente.

- **Aplicación de interactuación y kit de desarrollo de software (SDK) del brazo robótico:** con objeto de controlar el brazo robótico se emplea la aplicación **UFACTORY Studio**, la cual permite controlar la posición del brazo en tiempo real, grabar movimientos, desarrollar códigos de programación mediante bloques y posteriormente pasarlo a lenguaje Python e incluso establecer parámetros como el peso de la carga, la velocidad de movimiento, la sensibilidad a la hora de detectar colisiones, etc. Además, es necesario instalar el **kit de desarrollo de software (SDK**) **del robot colaborativo**, ya que se trata de un conjunto de herramientas y bibliotecas proporcionadas por el fabricante a fin de poder realizar su configuración y programación de manera efectiva.
- **SolidWorks:** se emplea el citado software de diseño asistido por computadora con el propósito de realizar el diseño en 3D de un soporte que sujete la cámara web al extremo del brazo robótico.

4. Desarrollo

En el presente capítulo se describen los pasos llevados a cabo en el proyecto para alcanzar el objetivo final. Como primer paso se realiza una evaluación preliminar del potencial que podría brindar la GPU, seguido de un análisis de selección del hardware adecuado para la implementación de la inteligencia artificial en el proyecto. Posteriormente, se procede a la preparación y configuración del entorno de la Jetson Nano con objeto de comenzar con la creación de las redes neuronales requeridas. Una vez implementadas dichas redes neuronales en el ámbito de la robótica junto a los códigos de programación necesarios, se obtienen los resultados finales del proyecto.

A continuación, se muestra un diagrama de flujo con la lógica seguida a lo largo del desarrollo del proyecto:

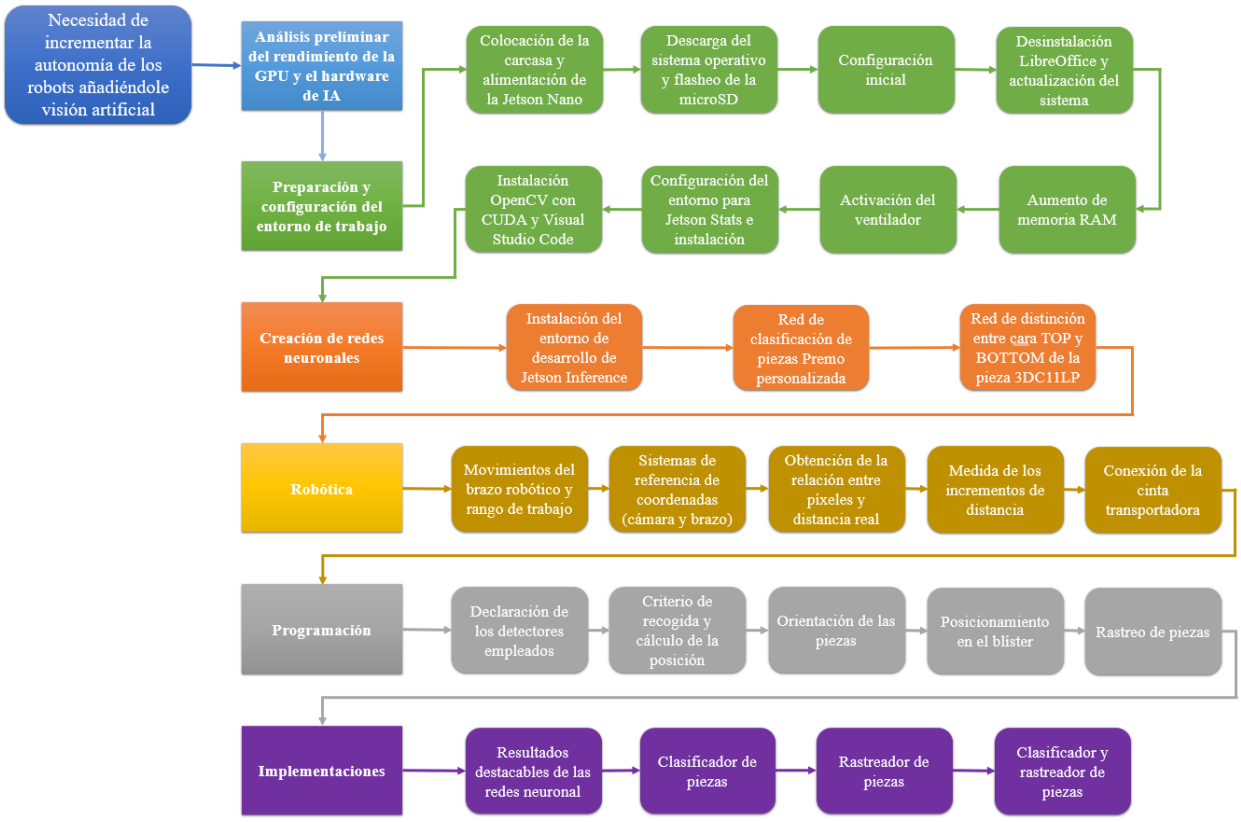

Figura 24. *Diagrama de flujo seguido en el desarrollo del proyecto.*

En el capítulo se presentan en detalle los pasos seguidos en el desarrollo del proyecto y se proporciona en los Anexos información adicional de los procedimientos que así lo requieran por ser muy densos o por su nivel de minuciosidad técnica, lo cual requiere de una descripción más exhaustiva. Se especifica en los apartados correspondientes del presente capítulo aquellos procedimientos que requieren de un Anexo para completar su descripción y asimismo se indica el apartado del Anexo en el que puede encontrarse la información correspondiente a cada procedimiento.

4.1. Análisis preliminar del rendimiento de la GPU y de los dispositivos de inteligencia artificial

En primer lugar, antes de entrar en materia, se realiza un análisis del beneficio aportado por la GPU en el proceso de detección de anomalías en la cara inferior de la pieza 3DCoil y una comparativa de los distintos dispositivos de computación acelerada con el fin de seleccionar la opción óptima en términos de relación calidad-precio para lograr el objetivo planteado.

4.1.1. Comparación del rendimiento en la detección de anomalías con y sin el uso de una unidad de procesamiento gráfico (GPU)

En el presente apartado, se realiza una comparación entre la realización de un proceso como es la detección de anomalías en piezas electromagnéticas con o sin el uso de una unidad de procesamiento gráfico (GPU), mediante el uso de MATLAB®, con el fin de observar la magnitud de los beneficios aportados por el empleo de la computación acelerada por GPU. En dicho procedimiento se emplea el **portátil MSI** con 16 gigabytes (GB) de RAM y GPU **NVIDIA GeForce GTX 1650 Ti with Max-Q Design**, mencionado en el apartado de Hardware.

Para realizar dicha comparativa, se lleva a cabo la detección de anomalías con y sin GPU en la cara inferior de la pieza electromagnética 3DCoil no moldeada, producida por la empresa PREMO. Para ello se instala la versión R2022b de MATLAB®, se buscan las imágenes a las que se les va a realizar el proceso de detección de anomalías, se parte de un código de detección de anomalías ya desarrollado por trabajadores de PREMO [39], se desarrollan los comandos necesarios para realizar la cargar de los datos requeridos, para configurar el uso o no de la GPU, para almacenar los tiempos, y finalmente realizar las gráficas e histogramas con los tiempos obtenidos.

Como podemos ver a continuación, a partir de una imagen de entrada de la pieza, se obtiene una imagen de salida con las anomalías, en caso de tenerlas, marcadas por medio de una circunferencia de color rojo.

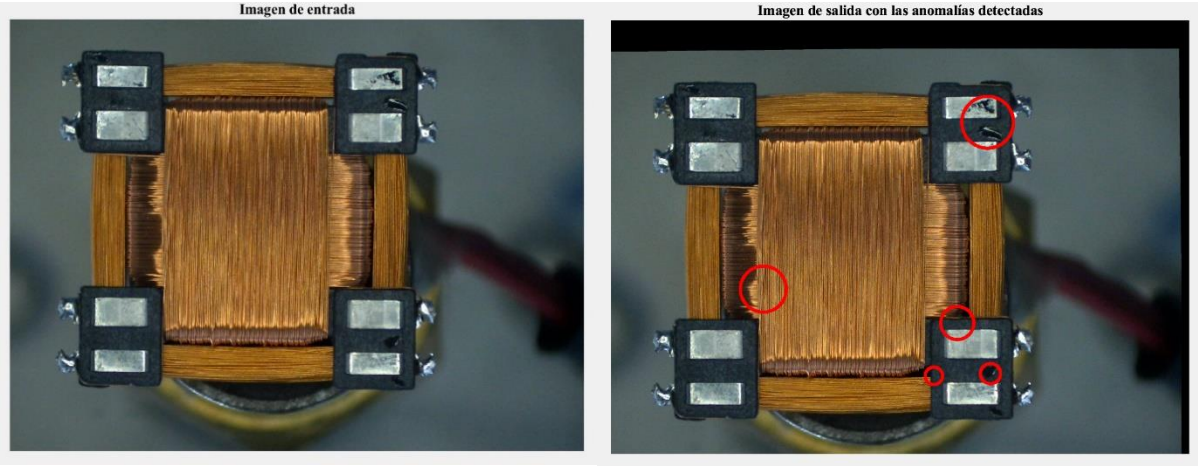

Figura 25. *Comparación de imagen de entrada con la imagen de salida, indicando las anomalías.*

Con objeto de obtener una mayor precisión en los resultados, se analizaron quinientas piezas 3DCoil por cada una de las seis caras de la pieza, aunque debido a que el alcance del presente proyecto no aborda el tema mencionado, se exponen únicamente los resultados de la cara inferior de la pieza, los cuales son más que suficientes para evidenciar las ventajas en términos de tiempos que se obtienen al emplear la GPU (para el resto de las caras, los resultados fueron similares). Una vez obtenidos los valores de tiempo que tardan en analizarse cada una de las quinientas piezas por su cara inferior y los comandos necesarios, recogidos en el Anexo I, se obtiene la gráfica e histograma que se muestran a continuación, los cuales ayudarán a reflejar la mejora aportada por la GPU.

En la ***Figura 26*** y en la ***Figura 27***, se muestra en naranja el tiempo que tarda en analizarse cada una de las quinientas piezas 3DCoil por su cara inferior sin el uso de la GPU, mientras que en azul se reflejan los tiempos de ese mismo análisis empleando ahora la GPU del portátil utilizado en dicha comparativa.

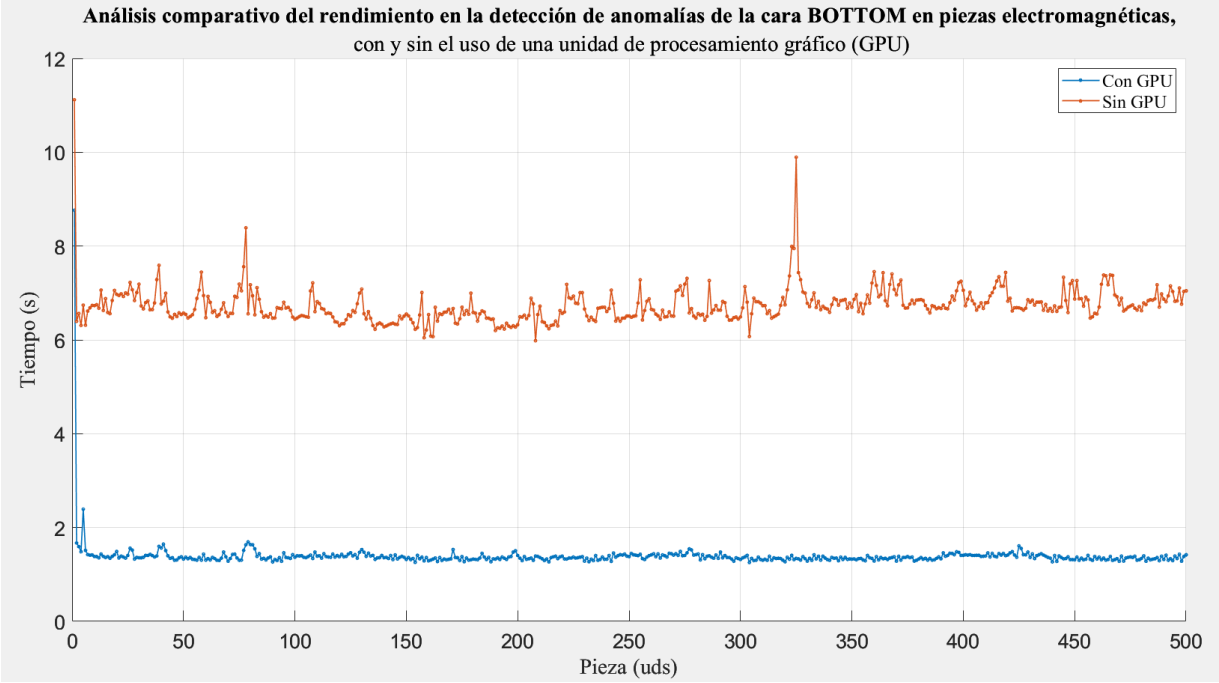

Figura 26*. Gráfica comparativa del rendimiento en la detección de anomalías de la cara BOTTOM en piezas electromagnéticas con y sin el uso de GPU.*

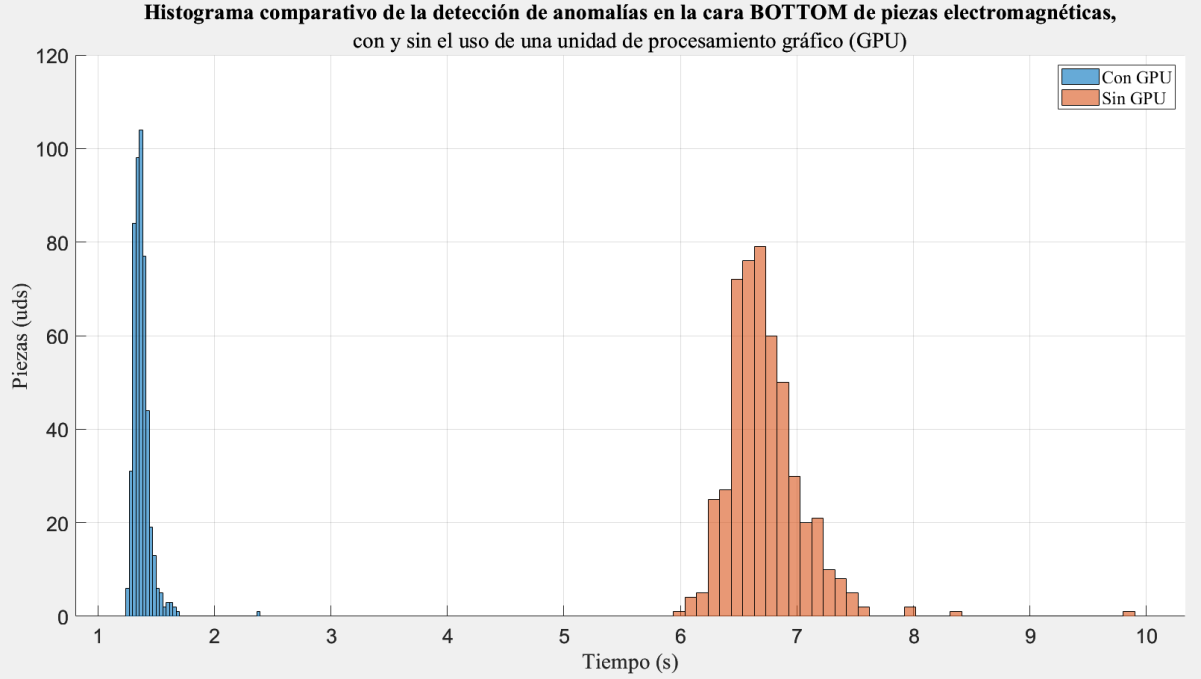

Figura 27. *Histograma comparativo del rendimiento en la detección de anomalías de la cara BOTTOM en piezas electromagnéticas con y sin el uso de GPU.*

En ambas figuras se refleja la misma información, aunque de distinto modo; en la ***Figura 26***, se presenta en forma de gráfica, y en la ***Figura 27***, mediante un histograma, con el propósito de observar una comparación más visual y efectiva.

Por medio de ambas figuras es posible observar los beneficios en términos de tiempo aportados por el uso de la GPU en el procedimiento de detección de anomalías, siendo el peor de los tiempos[1] empleando la GPU (2,36 segundos) muy inferior al mejor de los tiempos sin su uso (5,94 segundos), concluyendo de este modo que es de gran interés el uso de la computación acelerada por GPU para abordar el objetivo propuesto.

Cabe destacar que el peor tiempo registrado al emplear la GPU se trata de una excepción, ya que la mayoría de los tiempos de análisis en la detección de anomalías de las piezas 3DCoil por su cara inferior oscilan alrededor de 1,35 segundos, un segundo inferior al máximo registrado, siendo el mejor tiempo registrado de 1,24 segundos. Por otro lado, también se observa que el peor tiempo sin el empleo de la GPU resulta ser una excepción, dado que la mayoría de los tiempos en dicho caso se encuentran en torno a los 6,63 segundos.

[1] El tiempo empleado en analizar la primera pieza es descartado debido a que al iniciar la ejecución del programa algunos datos son almacenados en la memoria caché por lo que la primera llamada a cualquier función resultará más lenta de lo habitual

4.1.2. Selección del Hardware para la Aceleración Computacional

En el marco de la implementación de la computación acelerada en un proyecto, seleccionar adecuadamente la plataforma de hardware juega un papel esencial en términos de eficiencia y coste. En el presente apartado, se exponen algunas de las tecnologías más punteras en el ámbito de las plataformas de computación acelerada, con el fin de identificar la opción óptima que satisfaga los objetivos planteados en el proyecto.

Pese a que el número de empresas dedicadas al desarrollo de plataformas de computación acelerada está en constante crecimiento debido a la creciente demanda en dicho sector, el proceso de elección partió con la lucha entre tres grandes rivales: **Intel, Google y NVIDIA,** empresas consideradas inicialmente por su prestigio y fácil acceso a su documentación e información.

En primer lugar, la empresa Intel es considerada por su reputación, versatilidad y amplio bagaje en soluciones tecnológicas. Posteriormente, Google llama la atención debido a su renombre e innovaciones en el campo de la inteligencia artificial y el aprendizaje profundo. Aunque tras un análisis exhaustivo, la elección final del hardware recae sobre una de las opciones proporcionadas por NVIDIA, empresa líder en el campo de la computación acelerada. Dicha empresa, además de aportar una amplia gama de opciones en términos de plataformas, con diversas capacidades y características con el propósito de seleccionar la más adecuada según el alcance de cada aplicación, también dispone de una gran variedad de ejemplos y una amplia comunidad de desarrolladores, lo cual brinda un sólido soporte.

Como se ha comentado, NVIDIA ofrece un amplio catálogo de sistemas integrados con Jetson que presentan diversas capacidades, características y precios adaptados a diferentes tipos de aplicaciones [40]. A continuación, se presentan las opciones analizadas junto a sus características:

- **Jetson Orin.**

Dentro de este modelo de Jetson se encuentran las siguientes series:

- La serie **Jetson AGX Orin** de 32 GB o 64 GB que presentan respectivamente 1792 y 2048 núcleos de procesamiento basados en la arquitectura Ampere. Además de 8 y 12 núcleos de CPU respectivamente.
- La serie **Jetson Orin NX** de 8 GB o 16 GB que presentan 1024 núcleos de procesamiento basados en la arquitectura Ampere. Además de 8 núcleos de CPU.

El precio del kit de desarrollo de 32GB de la serie Jetson AGX Orin ronda los 1.800,00 €.

- **Jetson Xavier.**

Dentro del presente modelo, se encuentran las siguientes series de Jetson:

- La serie **Jetson AGX Xavier** (32 y 64 GB) que presenta 512 núcleos de procesamiento basados en la arquitectura Volta junto a 8 núcleos de CPU.

- La serie **Jetson Xavier NX** (8 y 16 GB) que presenta 384 núcleos de procesamiento basados en la arquitectura Volta junto a 6 núcleos de CPU.

Para hacer una estimación del precio, se observa que el módulo de Jetson AGX Xavier de 64 GB presenta un precio aproximado de 1.295,00 €, siendo por otro lado el precio del módulo Jetson Xavier NX de 16 GB de 536,00 € aproximadamente. En cuanto al kit de desarrollo, que resulta una opción más interesante para abordar el presente proyecto, la cifra asciende en la serie NX a los 982,00 € para un kit de 8 GB.

- **Jetson TX2.**

La serie TX2 presenta 256 núcleos de procesamiento basados en la arquitectura Pascal junto a 6 núcleos de CPU.

El kit de desarrollo de 8 GB de dicha serie presenta un precio aproximado de 800,00 €.

- **Jetson Nano.**

Finalmente, la Jetson Nano presenta 128 núcleos de procesamiento basados en la arquitectura Maxwell junto a 4 núcleos de CPU a un precio no muy superior a los 200,00 €.

En la ***Figura 28***, se muestra una tabla en la que aparecen las especificaciones detalladas de todas las series de Jetson anteriormente mencionadas, además, en la ***Figura 29*** se muestra una comparación sobre los tiempos de Kernel de la GPU para procesar imágenes 2K.

	NANO	TX2 NX	XAVIER NX*	ORIN NX 8 GB	ORIN NX 16 GB	AGX XAVIER**	AGX ORIN 32GB	AGX ORIN 64GB
GPU	128 Core Maxwell 0.5 TFLOPs (FP16)	256 Pascal cores 1.3 TFLOPS (FP16)	384 Core Volta 21 TOPs (INT8)	1024 Cores Ampere NVDLA V2 70 TOPS (INT8)	1024 Cores Ampere 2x NVDLA V2 100 TOPS (INT8)	512 Core Volta + NVDLA 10 TFLOPS (FP16) 32 TOPS (INT8)	1792 Cores Ampere 2x NVDLA V2 200 TOPS (INT8)	2048 Cores Ampere 2x NVDLA V2 275 TOPS (INT8)
CPU	4 core ARM A57	6 core Denver and A57 (2x) 2MB L2	6 core Carmel ARM V8 (3x) 2MB L2 + 4MB L3	6 cores A78 ARM V8 1.5MB L2 + 4MB L3	8 cores A78 ARM V8 2MB L2 + 4MB L3	8 core Carmel ARM V8 (4x) 2MB L2 + 4MB L3	8-core A78 ARM V8 2MB L2 + 4MB L3	12 core A78 ARM V8 3MB L2+ 6MB L3
Memory	4 GB 64-bit LPDDR4 25.6 GB/s	4 GB 128-bit LPDDR4 51.2 GB/s	8GB or 16GB 128-bit LPDDR4x 51.2 GB/s	8 GB 128-bit LPDDR5 102.4 GB/S	16 GB 128-bit LPDDR5 102.4 GB/S	32GB or 64 GB 256-bit LPDDR4x 137 GB/s	32 GB 256-bit LPDDR5 204.8 GB/s	64 GB 256-bit LPDDR5 204.8 GB/s
Storage	16 GB eMMC	16 GB eMMC	16 GB eMMC	- (Supports external NVMe)	- (Supports external NVMe)	32 GB eMMC	64 GB eMMC	64 GB eMMC
Encode	4K @ 30 (H.265)	4K @ 60 (H.265)	2x 4K @ 30 (H.265)	1x 4K @ 60 (H.265)	1x 4K @ 60 (H.265)	4x 4K @ 60 (H.265)	1x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)
Decode	4K @ 60 (H.265)	2x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)	6x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)	3x 4K @ 60 (H.265)
Camera	12 (3x4 or 4x2) MIPI CSI-2 D-PHY 1.1 lanes (18 Gbps)	12 lanes (3x4 or 5x2) MIPI CSI-2 D-PHY 1.2 (30 Gbps)	12 lanes (3x4 or 6x2) MIPI CSI-2 D-PHY 1.2 (30 Gbps)	8 lanes MIPI CSI-2 D-PHY 1.2 (20 Gbps)	8 lanes MIPI CSI-2 D-PHY 1.2 (20 Gbps)	16 lanes MIPI CSI-2 | 8 lanes SLVS-EC D-PHY (40 Gbps) C-PHY (59 Gbps)	16 lanes MIPI CSI-2 D-PHY 2.1 (40 Gbps) C-PHY 2.0 (164 Gbps)	16 lanes MIPI CSI-2 D-PHY 2.1 (40 Gbps) C-PHY 2.0 (164 Gbps)
Mechanical	69.6mm x 45mm 260 pin edge connector	69.6mm x 45mm 260 pin connector	69.6mm x 45mm 260 pin edge connector	69.6mm x 45mm 260 pin edge connector	69.6mm x 45mm 260 pin edge connector	100mm x 87mm 699 pin connector	100mm x87 mm 699 pins connector	100mm x87 mm 699 pins connector
Software	JetPack SDK - Unified software release across all Jetson products							

Figura 28. *Especificaciones de los diferentes sistemas integrados para la computación acelerada desarrollados por NVIDIA [41].*

Algorithm and parameters / Jetson model	Jetson Nano	Jetson TX2/TX2i	Jetson Xavier NX	Jetson AGX Xavier
White Balance	0.6	0.24	0.19	0.08
L7 Debayer (window 7×7)	1.95	0.87	0.61	0.40
DFPD Debayer (window 11×11)	4.7	2.06	1.08	0.95
MG Debayer (window 23×23)	12.7	5.9	2.73	2.2
Color Correction with 3×4 matrix	1.7	0.81	0.55	0.25
Resize from 2K to 960×540	10.0	4.3	2.21	1.5
Resize from 2K to 1919×1079	19.8	8.2	4.34	2.4
Gamma (1920×1080)	1.4	0.84	0.42	0.2
JPEG compression (1920×1080, 90%, 4:2:0)	4.3	1.7	1.09	0.62
JPEG compression (1920×1080, 90%, 4:4:4)	6.8	2.6	1.5	0.75
Total for simple camera pipeline (ms)	**9.95**	**4.8**	**2.85**	**1.53**

Figura 29. *Tiempos de kernel de GPU en el procesamiento de imágenes 2K (1920 × 1080, 8/16 bits por canal, milisegundos) [42].*

Finalmente, tras los datos anteriormente expuestos es necesario tomar una decisión sobre la opción más adecuada para abordar el proyecto:

- La **serie Orin** fue descartada por el hecho de poseer uno de los equipos de IA más potentes del mundo para máquinas autónomas y de gran eficiencia energética, por lo que dicha serie proporciona una sucesión de características que excede las necesidades del proyecto a un elevado precio en comparación con otras alternativas.
- Por otro lado, está el modelo **Xavier** en el que la serie Jetson Xavier NX fue preseleccionada, puesto que, para abordar el problema planteado, satisface con creces las necesidades a un menor precio que con la Jetson AGX Xavier que presenta mejores prestaciones, pero como se ha comentado a un mayor precio.
- La **serie Jetson TX2** presenta un menor poder computacional que la serie anteriormente comentada por un precio muy similar, que no supera los 200,00 € de diferencia. Dicha opción es conveniente tenerla en cuenta a la hora de buscar proveedores, pero en caso de encontrar un precio similar al de la serie Xavier NX, esta última tendrá prioridad.
- Por último, NVIDIA ofrece la **Jetson Nano** que se trata un ordenador de IA pequeño. Este presenta el rendimiento y la eficiencia energética necesarios para ejecutar cargas de trabajo de IA modernas, varias redes neuronales en paralelo y datos de proceso de varios sensores de gran resolución al mismo tiempo. Posee un menor poder computacional que el resto, aunque resulta más que suficiente para abordar el tema del proyecto planteado.

Finalmente, se selecciona la plataforma Jetson Nano como hardware para la implementación de la computación acelerada en el proyecto debido a su bajo coste y poder computacional idóneo para alcanzar los objetivos planteados eficientemente.

4.2. Preparación y configuración del entorno de trabajo de la Jetson Nano

En este apartado tiene lugar la preparación y configuración del entorno de trabajo de la Jetson Nano, desde su primer arranque incluyendo una alimentación adecuada, la instalación del sistema operativo, además de realizar ciertas configuraciones con vistas a prevenir errores, hasta la instalación de los paquetes y librerías necesarios, con el propósito de ponerla a disposición para su posterior uso en redes neuronales.

A continuación, se muestran los pasos llevados a cabo para la preparación y configuración de la Jetson Nano en aplicaciones de visión artificial.

4.2.1. Colocación de la carcasa y alimentación

La colocación de la carcasa se trata de un paso adicional y no estrictamente necesario, aunque aporta una mayor protección de la Jetson Nano. Para una mayor comodidad a la hora de trabajar se recomienda prescindir inicialmente de la parte superior de la carcasa.

Por otro lado, en cuanto a la alimentación de la Jetson Nano como indica la Guía de Uso [43], el kit de desarrollo requiere de una fuente de alimentación de 5V capaz de suministrar una corriente de al menos 2 A. Cabe la posibilidad de realizar la alimentación por medio de un puerto Micro-USB 2.0, J28 en la ***Figura 30***, o bien por medio del conector tipo barril, J25.

En el presente caso, se opta por alimentar la placa a través de la toma de alimentación J25 con el fin de superar los 2A que proporciona la fuente de alimentación Micro-USB 2.0 debido a la cantidad de periféricos que serán conectados a la placa en el fututo. Siendo la fuente de alimentación J25 una toma de alimentación para una fuente de 5V y 4A.

Para realizar entonces la alimentación mediante dicha fuente, deben conectarse los pines del conector J48 para desactivar la alimentación a través de Micro-USB y habilitar los 5V y 4A de la toma de alimentación J25.

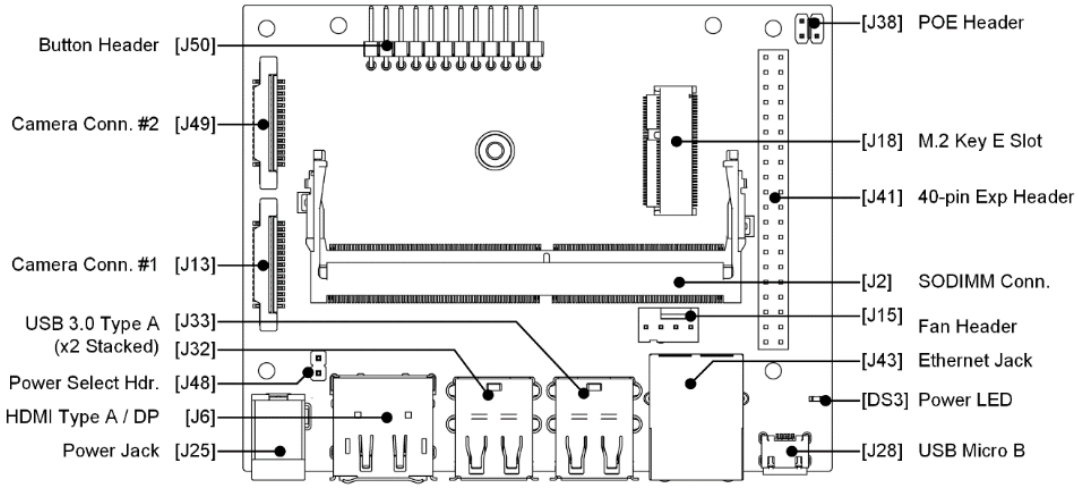

Figura 30. *Módulo del kit de desarrollo y placa portadora versión B01. Vista superior [43].*

4.2.2. Descarga del sistema operativo y flasheo de la tarjeta microSD

Con el objeto de ejecutar las redes neuronales en el kit de desarrollo, en primer lugar, es necesario instalar la imagen del sistema operativo basado en Ubuntu. Para ello, se requiere la descarga de varios elementos, incluyendo la imagen del sistema operativo y una aplicación de software llamada *balenaEtcher* [44], programa recomendado por la página oficial de NVIDIA para flashear dicha imagen en una tarjeta microSD. Es necesario realizar el flasheo de la tarjeta microSD de 256 GB en este caso antes de insertarla en la Jetson Nano, debido a que la imagen del sistema operativo se almacena en dicha tarjeta y no en el almacenamiento interno de la Jetson Nano. Al flashear la tarjeta se escribe la imagen del sistema operativo y las aplicaciones necesarias en ella, lo cual permite que la Jetson Nano arranque y ejecute el software desde la microSD.

NVIDIA ofrece diferentes versiones de la imagen del sistema operativo para Jetson Nano. En el presente caso, se selecciona la versión JetPack 4.6 [45] a pesar de que la última versión disponible para Jetson Nano hasta la fecha de este proyecto es JetPack 4.6.3. La opción es tomada en base a su ya comprobada compatibilidad con el hardware, sus funciones avanzadas de procesamiento de imágenes y un sólido soporte y programa de actualizaciones. Además, es necesario aclarar que para la descarga debe tenerse en cuenta si la Jetson Nano es de 2 GB o de 4 GB, como es el caso.

Una vez descargada la aplicación balenaEtcher y la imagen del sistema operativo, es posible flashear la imagen en la tarjeta microSD. Con el fin de lograrlo, es necesario seguir los pasos que se detallan a continuación:

- En primer lugar, seleccionar la opción "*Flash from File*" en el menú de balenaEtcher.
- A continuación, escoger la opción "*Select target*" para indicar la ubicación de instalación de la imagen del sistema operativo. Es importante prestar atención y seleccionar la tarjeta microSD de 256 GB correctamente y no el disco duro del ordenador.
- Finalmente, presionar "*Flash!*" para iniciar el proceso.

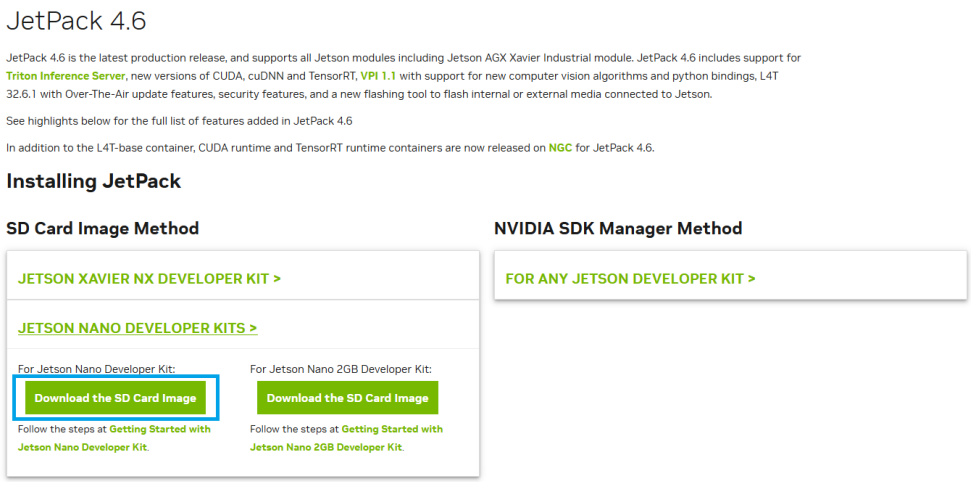

Figura 31. *Descarga de JetPack versión 4.6 desde la página oficial de NVIDIA [45].*

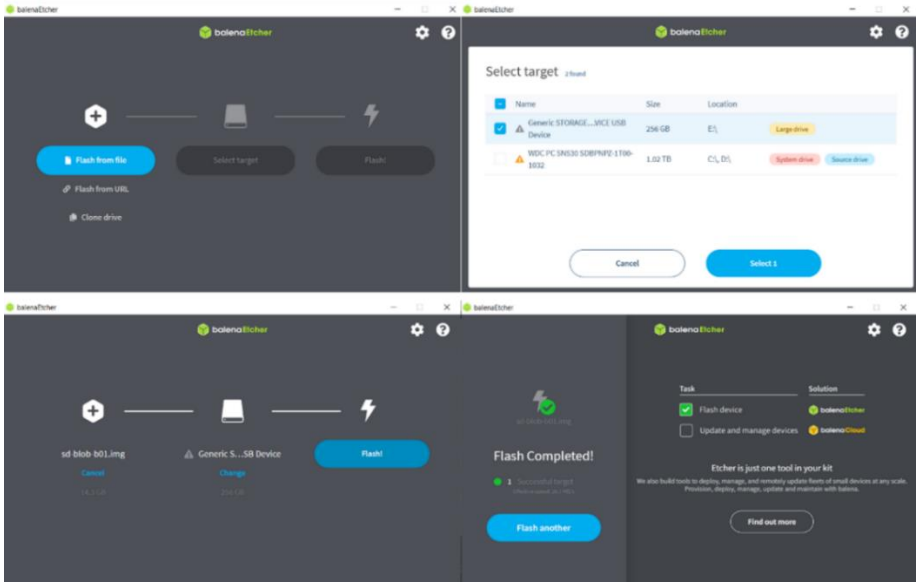

Figura 32. *Procedimiento del flasheo de la tarjeta microSD en balenaEtcher.*

Una vez flasheada la tarjeta, proceso que tiene una duración de veinte minutos, aparece el siguiente mensaje: "*Flash Completed!*", lo cual indica que el proceso ha finalizado y la imagen del sistema operativo ha sido flasheada con éxito en la tarjeta microSD.

4.2.3. Configuración inicial

En este apartado se describe el proceso de encendido y configuración de la Jetson Nano. En primer lugar, se inserta la tarjeta microSD previamente flasheada en la ranura ubicada en la parte inferior de la placa, como se observa en la

Figura 33. Tras conectar la placa a la fuente de alimentación, se procede a conectar los periféricos: una pantalla a través de un cable HDMI, un ratón y un teclado a través de los puertos USB. Finalmente, se presiona el botón de encendido.

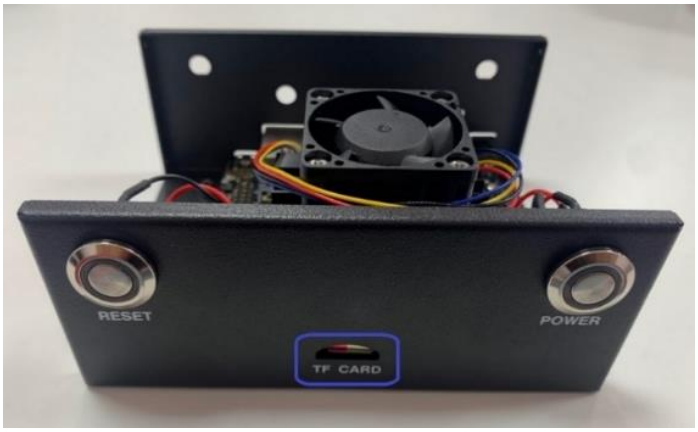

Figura 33. *Jetson Nano con tarjeta microSD portadora del sistema operativo.*

Una vez encendida por primera vez la placa, se realiza la configuración del sistema. En primer lugar, se aceptan los términos y condiciones y se selecciona el idioma, así como la disposición del teclado:

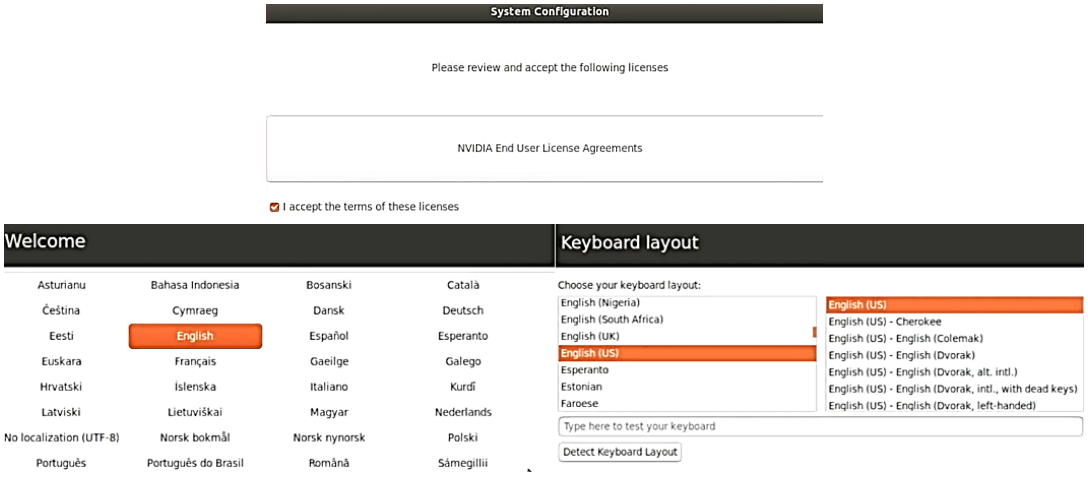

Figura 34. *Configuración inicial del sistema en Jetson Nano I.*

A continuación, se indica el lugar del mundo en el que se encuentra trabajando con la placa, Málaga en este caso. Y posteriormente, se indica el nombre de usuario y una contraseña:

Figura 35. *Configuración inicial del sistema en Jetson Nano II.*

Al avanzar al paso siguiente, se solicita que se determine la cantidad de espacio a asignar para el sistema operativo, teniendo en cuenta los 256 GB totales disponibles, que por defecto se configura al máximo posible. En este punto, se llega a un paso de suma importancia donde se pregunta por el modo de alimentar la placa. Por defecto se selecciona la opción "*MAXN*" con el fin de proporcionar una mayor potencia de alimentación a la Jetson Nano, permitir la conexión de una mayor cantidad de periféricos y asegurar un mejor rendimiento:

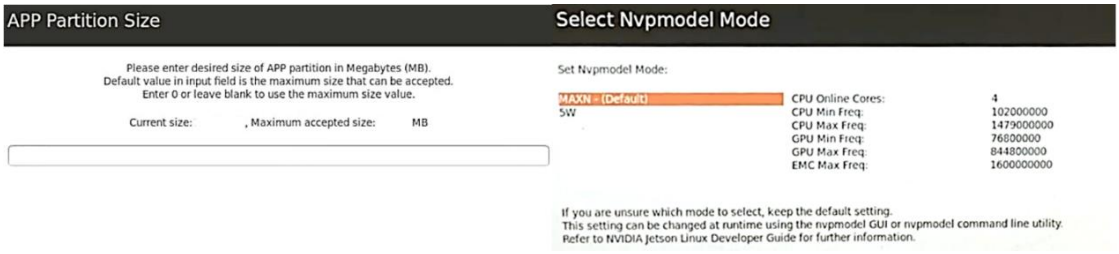

Figura 36. *Configuración inicial del sistema en Jetson Nano III.*

4.2.4. Desinstalación de LibreOffice y actualización del sistema

En esta y las dos próximas secciones, se muestran tres configuraciones que se recomienda realizar con el fin de prevenir posibles errores al ejecutar modelos de Deep Learning o Machine Learning.

Con el fin de obtener la máxima potencia de la placa y trabajar a su máximo rendimiento es necesario liberar espacio, es por ello por lo que se aconseja eliminar el software LibreOffice, puesto que no es necesario su uso desde la Jetson Nano, ver apartado A del Anexo II.

Además, al acceder a la interfaz de la Jetson Nano, es crucial actualizar el sistema operativo, JetPack 4.6 en nuestro caso, permitiendo así limpiar los paquetes obsoletos y garantizar la actualización de los repositorios y paquetes instalados. Para ello, se deben ejecutar los comandos referidos a la actualización del apartado A del Anexo II.

4.2.5. Aumento de la memoria Swap

En este paso, con la ayuda del primer episodio del Curso "Jetson AI Fundamentals" [46], se persigue aumentar la memoria RAM de nuestra Jetson Nano por medio de la memoria Swap, la cual es una extensión temporal utilizada cuando la memoria RAM física está agotada.

Es posible comprobar que por defecto Ubuntu 18.04 de Jetson Nano posee 2 GB de memoria Swap:

```
grupopremo@grupopremo-desktop:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3956        1254        1388          26        1312        2533
Swap:          1978           0        1978
```

Figura 37. *Memoria Swap inicial de la Jetson Nano.*

Finalmente, tras seguir los pasos del apartado B del Anexo II, se reinicia la placa y se vuelve a ejecutar el comando "free -m" a fin de comprobar que efectivamente se ha ampliado el espacio de memoria Swap a 4 GB como se observa en la siguiente figura:

```
grupopremo@grupopremo-desktop:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3956        1061        2236          29         658        2726
Swap:          4095           0        4095
```

Figura 38*. Memoria Swap de Jetson Nano tras su ampliación.*

4.2.6. Activación del ventilador

Siguiendo los pasos del repositorio jetson-fan-ctl [47] y ejecutando los comandos descritos en el apartado C del Anexo II se pone en marcha el ventilador instalado en la Jetson con el fin de evitar un sobrecalentamiento de la placa a la hora de ponerla en funcionamiento.

En primer lugar, es necesario disponer de un ventilador de modulación por ancho de pulsos (PWM) de 5V e instalarlo adecuadamente, por medio de cuatro tornillos, en la placa como se muestra en la ***Figura 39***.

Finalmente, una vez instalado el ventilador y puesto en funcionamiento por medio de los comandos del apartado C del Anexo II, se obtiene un control automático del ventilador según la temperatura de la placa.

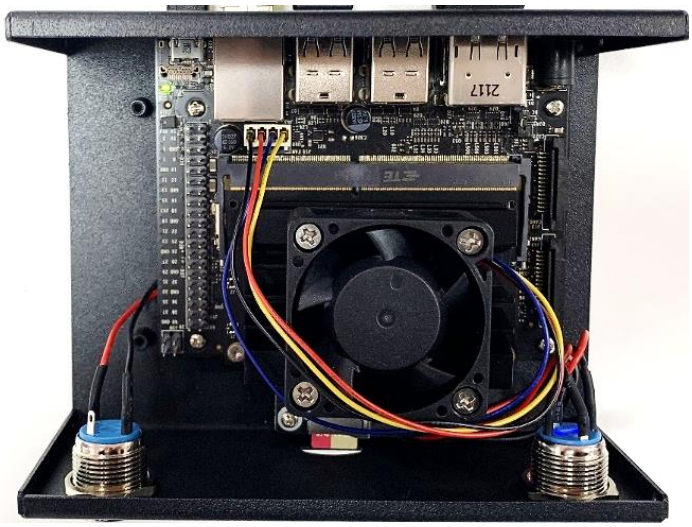

Figura 39. *Ventilador instalado en el kit de desarrollo Jetson Nano.*

4.2.7. Configuración del entorno para Jetson Stats e instalación

El propósito de la presente sección es crear un entorno de soporte para *Jetson Stats* y su correspondiente instalación. *Jetson Stats* consiste en un paquete para monitorear y controlar la plataforma de NVIDIA empleada. Para lograrlo, es necesario seguir los pasos detallados en el apartado D del Anexo II.

Tras seguir los pasos, se accede a una interfaz de monitoreo en tiempo real que muestra información sobre el rendimiento del sistema. Concretamente, a continuación, se muestra la información que se obtiene en las siete pestañas de la interfaz al ejecutar el comando "*jtop*" junto con las correspondientes imágenes de la interfaz:

- En la primera pestaña **(ALL)**, aparecen datos generales sobre el estado de la Jetson incluyendo información sobre la CPU, la GPU, la memoria, el ventilador, el disco y el estado sobre NVPmodel, jetson_clocks, etc. Por otro lado, en la segunda pestaña **(GPU)** se muestra el consumo de la GPU:

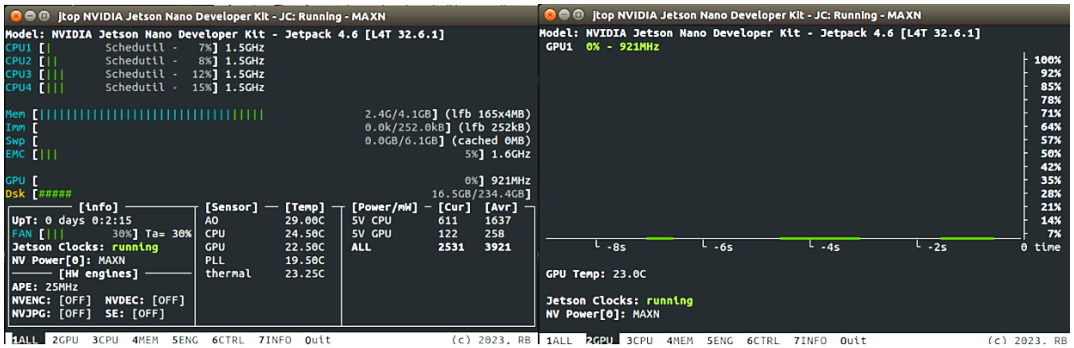

***Figura 40**. Pestañas ALL y GPU en la interfaz de jtop en Jetson Nano.*

- En la tercera pestaña **(CPU)**, se refleja el consumo de la CPU y en la cuarta pestaña **(MEM)** aparece un gráfico de la memoria en tiempo real junto a su consumo:

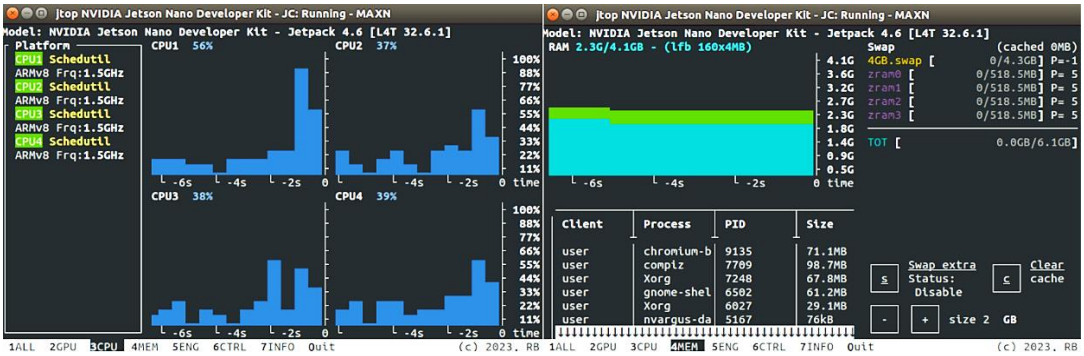

***Figura 41.** Pestañas CPU y MEM en la interfaz de jtop en Jetson Nano.*

- En la quinta **(ENG)** y sexta pestañas **(CTRL)**, aparece respectivamente una lista en tiempo real con el estado de los motores y el control de ventilación, que en el presente caso se encuentra al 25%:

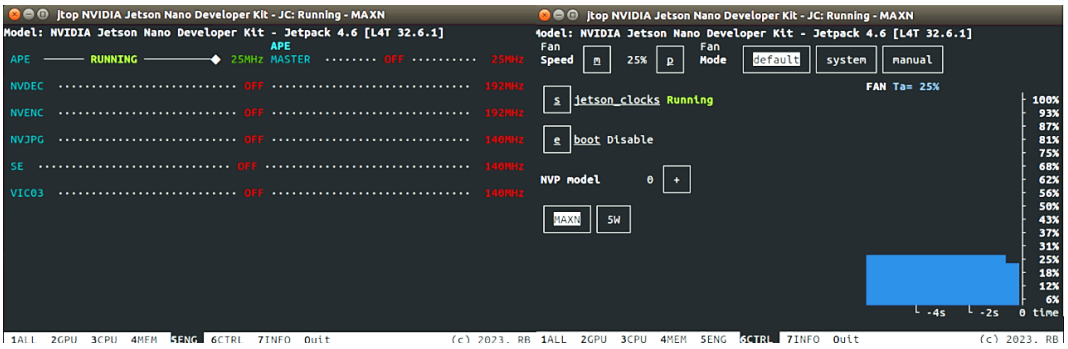

***Figura 42.** Pestañas ENG y CTRL en la interfaz de jtop en Jetson Nano.*

- Finalmente, en la última pestaña **(INFO)** aparece información sobre las librerías que se encuentran instaladas, número de serie, interfaces, etc.:

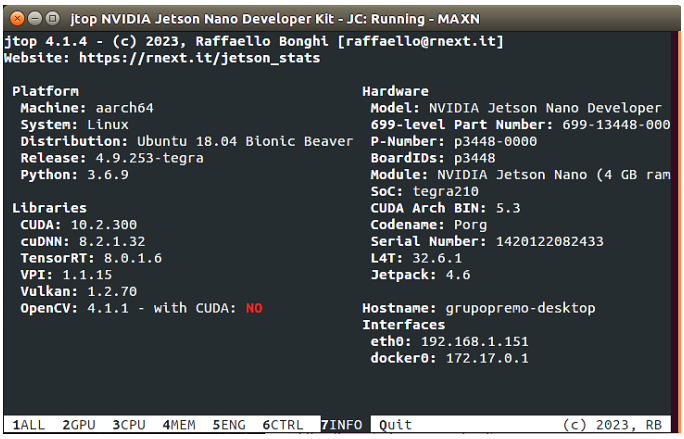

Figura 43. *Pestaña INFO en la interfaz de jtop en Jetson Nano.*

4.2.8. Instalación OpenCV con CUDA y Visual Studio Code

En último lugar, con el fin de completar la preparación y configuración del entorno de trabajo de la Jetson Nano, se realiza la instalación de la biblioteca OpenCV 4.1 con soporte CUDA y el editor de código fuente Visual Studio Code.

Para llevar a cabo la instalación de OpenCV 4.1 con soporte CUDA, es necesario seguir los pasos detallados en el apartado E del Anexo II

. Una vez realizada correctamente la instalación, es posible comprobarla por medio del comando “jtop” mediante el cual se accede a una interfaz donde es posible verificar dicha información en la pestaña “INFO” tal y como se muestra en la ***Figura 44*** .

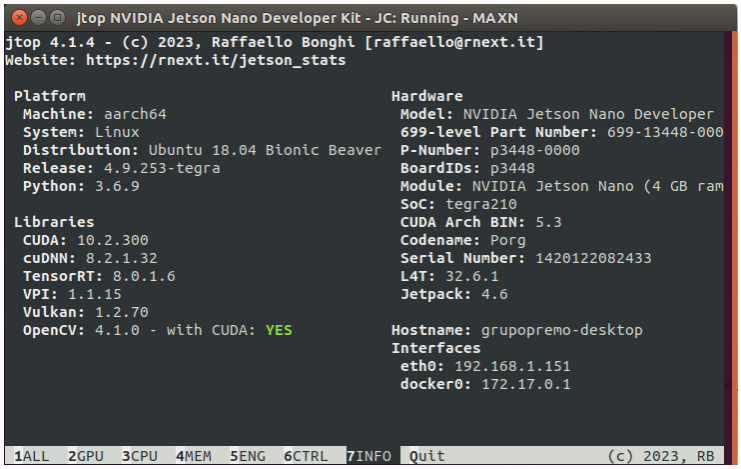

Figura 44. *Información de la interfaz de Jetson Nano: librería OpenCV instalada con soporte CUDA.*

A continuación, se explica el proceso de descarga e instalación de Visual Studio Code en Jetson Nano.

En primer lugar, accedemos a la página oficial de descargas de Visual Studio Code [48] donde aparece un desplegable como el de la ***Figura 45***, el cual proporciona opciones de descarga para diferentes sistemas operativos.

A continuación, se selecciona la opción "otras descargas" y aparecen todas las opciones de descarga para los diferentes sistemas operativos. En este caso, ya que Jetson Nano se trata de una plataforma con una Unidad Central de Procesamiento (CPU) basada en la arquitectura *Advanced RISC Machines* (ARM), se debe seleccionar la opción de descarga para Linux ARM, teniendo en cuenta la versión específica de Linux ARM empleada. Esto implica verificar si se trata de la versión de 32 bits o de 64 bits para así asegurar la compatibilidad con el sistema operativo de la plataforma. En el presente caso, como se refleja en la *Figura 46*, se selecciona la opción para Linux ARM de 64 bits.

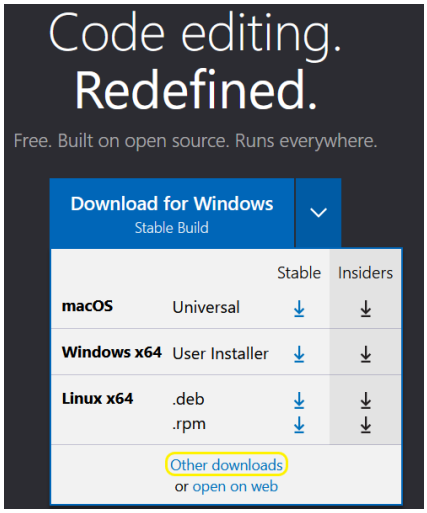

Figura 45. *Opciones de descarga de Visual Studio Code para diferentes sistemas operativos [48]*

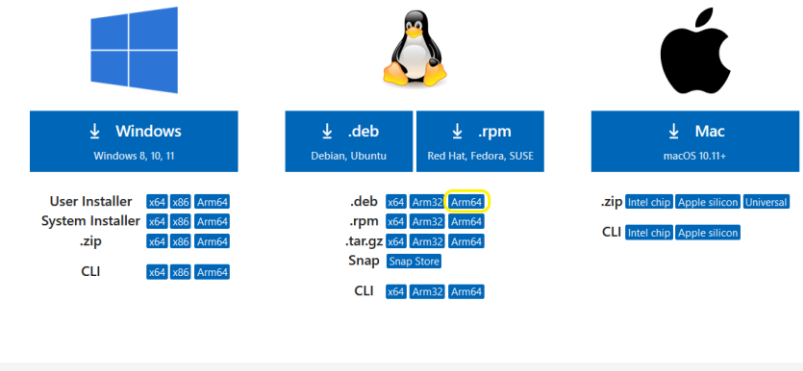

Figura 46*. Todas las opciones de descarga de Visual Studio Code para diferentes sistemas operativos [48]*

Una vez completa la descarga, se selecciona la carpeta de descargas y se clica en el archivo descargado, lo cual abrirá la siguiente ventana:

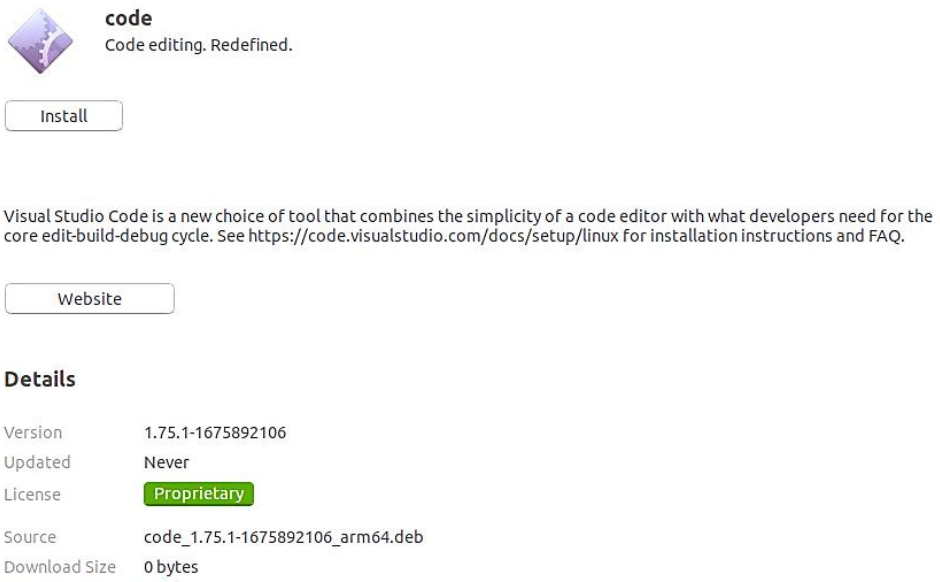

Figura 47. *Ventana de instalación de Visual Studio Code en Jetson Nano.*

Se selecciona el botón de instalación *"Install"*, completándose de este modo la instalación. Finalmente, una vez instalado correctamente Visual Studio Code en la Jetson Nano es necesario terminar de realizar la instalación de las librerías requeridas para la ejecución del proyecto tales como *Numpy*, *DateTime*, el kit de desarrollo del robot, etc. Para la instalación deben seguirse los comandos recogidos en el apartado F del Anexo II, donde también se muestra la verificación de la conexión de la cámara empleada a la Jetson Nano.

4.3.Creación de redes neuronales

En la presente sección se recogen los pasos requeridos en la creación de las dos redes neuronales necesarias para alcanzar el objetivo del presente proyecto; una para identificar las piezas 3DC11LP y las 3DC14EMR-ULP y otra para distinguir entre la cara superior e inferior de la 3DCoil no moldeada. Además, se instala en primer lugar el entorno de desarrollo de *Jetson Inference* con objeto de que quede instalado previamente todo lo necesario antes de comenzar con la creación de las redes neuronales requeridas.

4.3.1. Instalación del entorno de desarrollo de Jetson Inference

En el presente apartado, se realiza la instalación del entorno de desarrollo de *Jetson Inference* con el apoyo del repositorio [49] proporcionado por uno de los desarrolladores de *NVIDIA* y se ejecutan nuestras primeras redes neuronales preentrenadas.

Tras seguir los pasos detallados en el apartado A del Anexo III, se encuentra el entorno de desarrollo de *Jetson Inference* correctamente instalado junto a diversas redes neuronales preentrenadas, listas para ser ejecutadas.

A continuación, en la ***Figura 48*** se muestran los diferentes modelos de inferencia que es posible encontrar en la carpeta "*build*", que se ubica en el siguiente directorio: "/detección-objetos1/jetson-inference/build/aarch64/bin".

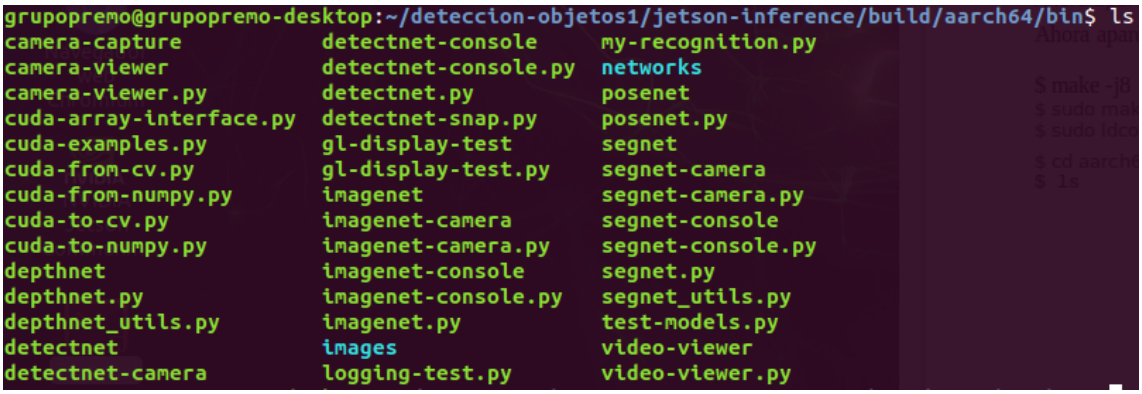

Figura 48. *Contenido de la carpeta "bin" con modelos de inferencia preentrenados.*

En ella se observan diversas redes neuronales preentrenadas con diferentes usos:

- Profundidad monocular (dethNet).
- Detección de objetos (detectNet).
- Reconocimiento de imagen (imageNet).
- Estimación de poses (poseNet).
- Segmentación (segNet).

En vista de ejecutar alguno de los scripts de la figura anterior, que implementa alguna de las funcionalidades mencionadas anteriormente empleando algún modelo de aprendizaje automático como "*depthnet*", "*detectenet*", "*imagenet*", "*posenet*" o "*segnet*", u otro script

desde la terminal de comandos, es necesario la ejecución de un comando específico. Dicho comando se compone de tres elementos:

- En primer lugar, debe indicarse el **intérprete de Python** que se empleará para ejecutar el script, Python 3 en este caso.
- Posteriormente debe indicarse el **nombre del script** que desea ejecutar.
- Finalmente, debe indicarse la **ruta o dirección de la cámara** empleada.

A modo de ejemplo, si se desea ejecutar el script "*detectnet.py*" cuya funcionalidad es realizar la detección de objetos empleando el modelo de inferencia "*detectnet*" con una cámara de video vinculada a la ruta "*/dev/video0*" se ejecutaría el siguiente comando:

```
$ python3 detectnet.py /dev/video0
```

Tras la ejecución del comando anterior, se carga un modelo de red neuronal que ha sido previamente entrenado para realizar la detección de objetos. Como se observa en la ***Figura 49***, este detector genera un recuadro sobre el objeto detectado e indica en su parte superior izquierda la clase del objeto junto con su valor de confianza.

Figura 49. *Ejemplo de detección de objetos modelo detecNet.*

En el ejemplo anterior, se observa que se está detectando una manzana con un 98,8 % de confianza. Es decir, la predicción de la red de que el objeto que está detectando sea una manzana posee un nivel de confianza del 98,8 %.

4.3.2. Red personalizada de clasificación de piezas PREMO

En Jetson Nano, es posible crear una red preentrenada con el fin de detectar objetos. Sin embargo, en el presente apartado se pretende crear y entrenar una red neuronal básica para la detección de dos tipos de piezas, concretamente las piezas 3DC11LP y 3DC14EMR-ULP fabricadas por la empresa PREMO, con el fin de poder emplearla para crear un clasificador de piezas.

El modelo de red empleado en la creación de las redes neuronales es el **SSD (*Single Shot Multibox Detector*)** que consiste en un tipo de arquitectura de red neuronal convolucional

empleada en tareas de detección de objetos tanto en imágenes como en videos. El detector SSD destaca en comparación con otros detectores de disparo único (capacidad del modelo para realizar la detección de objetos en una sola pasada a través de la red neuronal) debido al uso de múltiples capas que mejoran la precisión en la detección de objetos con diferentes escalas. A medida que se profundiza en las capas, el modelo SSD se vuelve más eficaz en la detección de objetos más grandes [50].

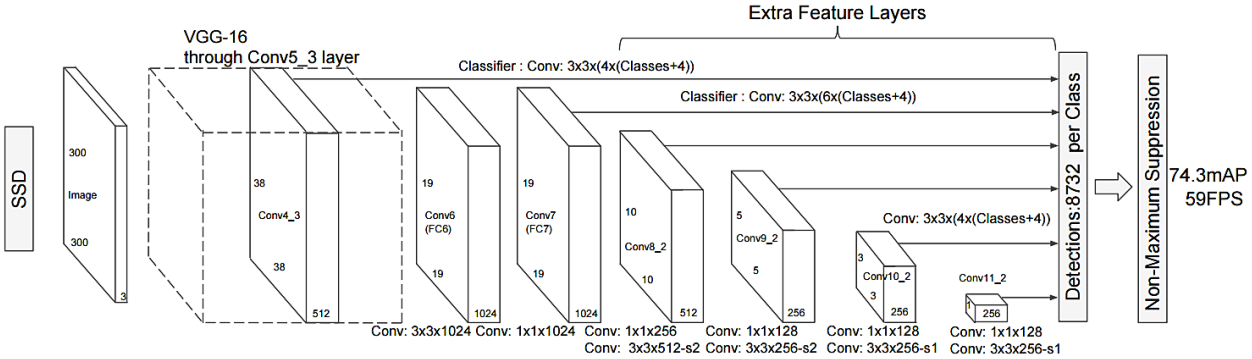

Figura 50. *Arquitectura de la red neuronal SSD [50].*

Por lo general, el modelo SSD comienza con un modelo preentrenado como VGG en Resnet y posteriormente se añaden capas convolucionales que mejoran el manejo de objetos de mayor tamaño. En principio, la arquitectura SSD puede utilizarse con cualquier modelo base de red profunda [50].

Se selecciona dicha arquitectura por estar validada para la detección de objetos y Jetson nos proporciona facilidades para su uso además de por su capacidad para realizar la detección de objetos en tiempo real con un buen equilibrio entre velocidad y precisión.

Llegados a este punto comienza la creación de una red neuronal capaz de detectar dos tipos de piezas. Como se puede observar en el apartado B del Anexo III, para lograr dicho objetivo, en primer lugar, se debe realizar una recopilación de los datos de detección para, posteriormente, realizar el entrenamiento del modelo.

Una vez completada la información de la sección **Recopilación de datos de detección** del apartado B del Anexo III, se recopilan las imágenes de las piezas junto a su etiqueta correspondiente siguiendo los siguientes pasos[2]:

1. En primer lugar, se congela la imagen presionando la opción "*FreezeEdit*".
2. Posteriormente, se establece un cuadro delimitador que abarque el contorno de toda la pieza seleccionada.
3. Una vez delimitada la pieza, se selecciona la etiqueta de la pieza correspondiente (3DC11LP o 3DC14EMR-ULP).

[2] También es posible etiquetar un conjunto de imágenes del que ya se dispone, omitiendo la necesidad de capturar las imágenes desde la cámara. Para ello debe emplear una herramienta de anotación de visión artificial (*Computer Vision Annotation Tool,* CVAT) y exportar el conjunto de datos en Pascal VOC [51].

4. Finalmente, se guarda la información presionando la opción "*Save*" ubicada en la parte inferior derecha de la ventana.

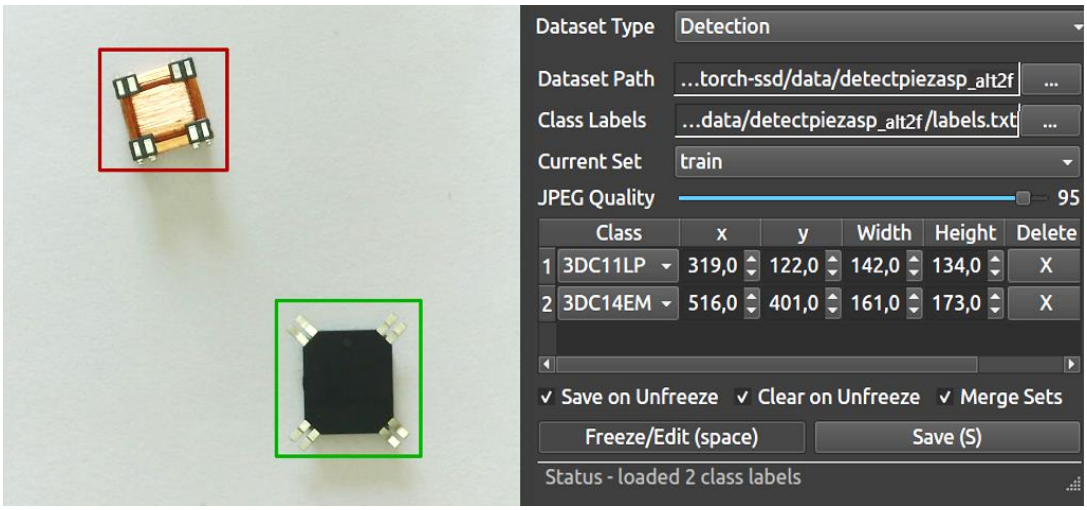

Figura 51. *Recopilación de datos para la red neuronal personalizada de clasificación.*

La recopilación de datos se realizó manualmente empleando la cámara dispuesta en el extremo del brazo robótico cuyo objetivo se encontraba prácticamente paralelo al plano de trabajo. Se tomaron 800 imágenes en el presente caso con sus respectivas etiquetas y a diferentes alturas con objeto de enriquecer la base de datos y aumentar así la precisión del modelo, además, las imágenes fueron tomadas a color con el propósito de obtener más información y mejorar la posterior distinción de las clases.

Una vez recopilada una cantidad suficiente de datos se realiza el entrenamiento del modelo de acuerdo con la sección **Entrenamiento del modelo** del apartado B del Anexo III. Dicho proceso de entrenamiento por medio del script "*train_ssd.py"* puede durar alrededor de 3 horas con la cantidad de imágenes mencionada y la información detallada en el apartado B del Anexo III: el tamaño del lote empleado durante el entrenamiento, la cantidad de subprocesos del cargador de datos y número de épocas que se emplearán para el entrenamiento.

En este punto, tras seguir los pasos detallados en apartado B del Anexo III, es posible emplear la red neuronal para llevar a cabo la detección de las piezas 3DC11LP y 3DC14EMR-ULP en cualquier aplicación o entorno de programación compatible con ONNX.

Una vez guardado el programa en la carpeta deteccion-objetos1/jetson-inference/build/aarch64/bin/detectpiezasp_alt2f, accedemos a dicha carpeta con el fin de ejecutarlo[3]:

```
$ cd detección-objetos1/jetson-
inference/build/aarch64/bin/detectpiezasp_alt2f
$ python3 detectpiezasp_alt2f.py /dev/video0
```

[3] Tras ejecutar el comando de la red neuronal por primera vez, se observará un tiempo de carga mayor en comparación con ejecuciones posteriores debido a diversas tareas que se llevan a cabo en el primer proceso de ejecución.

Una vez ejecutada la red neuronal desde la terminal de comandos, se abrirá una ventana que mostrará la salida visual de la detección y clasificación de los dos tipos de piezas diferentes capturados por la cámara conectada al dispositivo. En caso de que en el área capturada por la cámara aparezca un objeto que no pertenezca a ninguno de los dos tipos de piezas especificados, no será reconocido ni identificado por el sistema. La red neuronal diseñada y entrenada para llevar a cabo la clasificación de los dos tipos de piezas mencionados anteriormente, se enfoca únicamente en la identificación de las dos categorías especificadas:

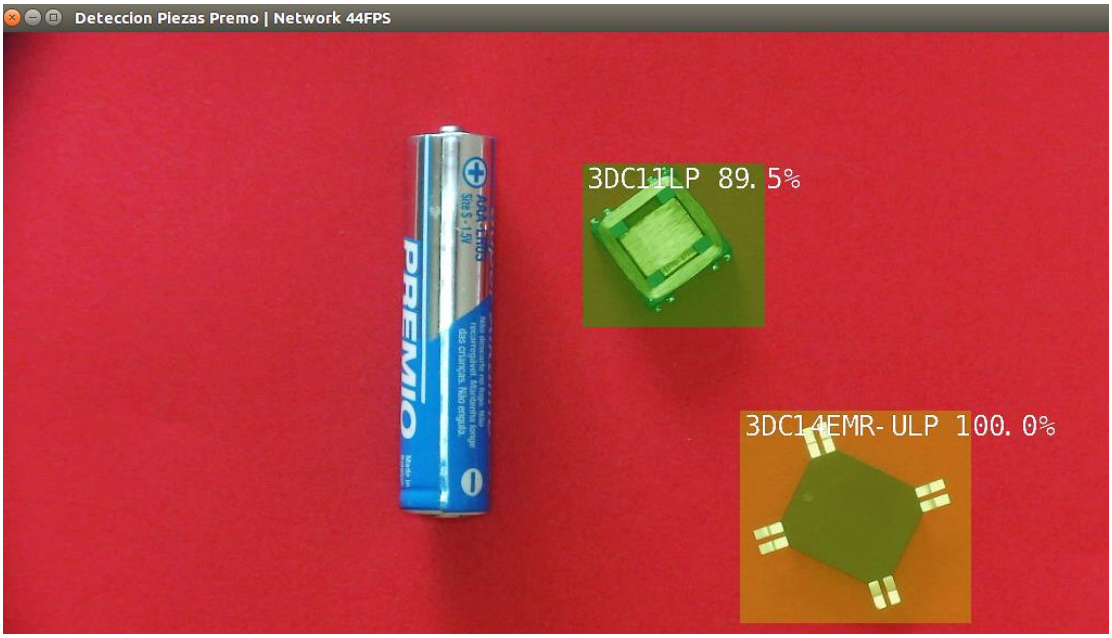

Figura 52. *Red neuronal de detección de piezas Premo, concretamente 3DC11LP y 3DC14EMR-ULP.*

El funcionamiento de la red fue validado manualmente con la cámara de baja resolución empleando 200 piezas de cada tipo, en diferentes días y con diferentes fondos, en condiciones de iluminación ambiente. Las distintas piezas se colocaron sobre el plano de trabajo en distintas posiciones y con distinta orientación empleando un threshold del 50 % en el nivel de confianza, siendo siempre la clasificación aportada por la red correcta. Por lo que se valida la red neuronal "detectpiezasp_alt2f" como buena.

En el caso de la red "detectpiezasp_alt2f", por tanto, tras la realización de numerosas pruebas en diferentes días y con distintos fondos, se comprueba que la red neuronal exhibe una notable precisión en su desempeño sobre todo a la hora de realizar la identificación de la pieza 3DC14EMR-ULP, dado que esta se trata de una pieza de color negro, sin reflejos, la cual presenta mayor facilidad de percepción sobre cualquier fondo con un contraste perceptible.

4.3.3. Red personalizada para la distinción entre la cara TOP y la cara BOTTOM de la pieza 3DC11LP

En el presente apartado, en virtud de los satisfactorios resultados obtenidos en la creación de la red neuronal para la clasificación de piezas, se pretende crear y entrenar una red neuronal de mayor complejidad, ya que, en este caso, la red debe realizar la distinción entre la cara superior e inferior de la pieza 3DC11LP, lo cual podría presentar mayor complejidad, puesto que ambas caras de la pieza son idénticas en cuanto a forma, tamaño y color. Como se puede observar en el apartado C del Anexo III para lograr dicho objetivo, en primer lugar, se debe realizar una recopilación de los datos de detección para posteriormente realizar el entrenamiento del modelo.

Una vez completada la información de la sección **Recopilación de datos de detección** del apartado C del Anexo III, se recopilan las imágenes de las piezas junto a su etiqueta correspondiente siguiendo los siguientes pasos:

1. En primer lugar, se congela la imagen presionando la opción "*FreezeEdit*".
2. Posteriormente, se establece el cuadro delimitador que abarque el contorno de toda la pieza seleccionada.
3. Una vez delimitada la pieza, se selecciona la etiqueta de la pieza correspondiente (TOP o BOTTOM).
4. Finalmente, se guarda la información presionando la opción "*Save*" ubicada en la parte inferior derecha de la ventana.

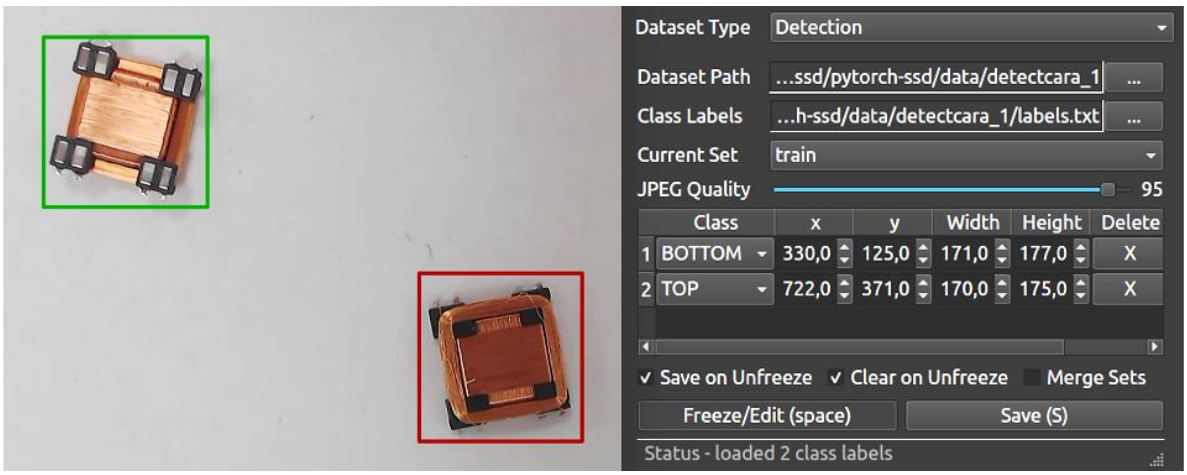

Figura 53. *Recopilación de datos para la red neuronal personalizada para la distinción entre la cara superior e inferior de la pieza 3DC11LP.*

La recopilación de datos se realizó manualmente empleando la cámara dispuesta en el extremo del brazo robótico cuyo objetivo se encontraba paralelo al plano de trabajo. Se tomaron 500 imágenes en el presente caso con sus respectivas etiquetas y a diferentes alturas con objeto de enriquecer la base de datos y aumentar así la precisión del modelo, además, las imágenes fueron tomadas a color con el propósito de obtener más información y mejorar la posterior distinción de las clases.

Una vez recopilada una cantidad suficiente de datos se realiza el entrenamiento del modelo de acuerdo con la sección **Entrenamiento del modelo** del apartado C del Anexo III. Dicho proceso de entrenamiento por medio del script "*train_ssd.py"* puede durar alrededor de 2 horas con la cantidad de imágenes mencionada y la información detallada en el apartado C del Anexo III: el tamaño del lote empleado durante el entrenamiento, la cantidad de subprocesos del cargador de datos y número de épocas que se emplearán para el entrenamiento.

Llegados a este punto, tras seguir los pasos detallados en el apartado C del Anexo III, se tiene la posibilidad de emplear la red neuronal para llevar a cabo la distinción entre la cara superior e inferior de la pieza 3DC11LP en cualquier aplicación o entorno de programación compatible con ONNX.

Una vez guardado el programa en la carpeta *deteccion-objetos1/jetson-inference/build/aarch64/bin/detectcara_1*, se accede a dicha carpeta con el fin de ejecutarlo:

```
$ cd detección-objetos1/jetson-inference/build/aarch64/bin/detectcara_1
$ python3 detectcara_1.py /dev/video0
```

Una vez ejecutada la red neuronal desde la terminal de comandos, se abrirá una ventana que mostrará la salida visual de la detección y distinción entre la cara superior e inferior de la pieza 3DC11LP capturada por la cámara conectada al dispositivo. Al igual que en el caso anterior, en caso de que en el área capturada por la cámara aparezca un objeto que no pertenezca a ninguna de las dos categorías especificadas, no será reconocido ni identificado por el sistema. La red neuronal diseña y entrenada para llevar a cabo la distinción entre la cara *top* y *bottom* de la 3DC11LP se enfoca únicamente en la identificación de las dos categorías especificadas:

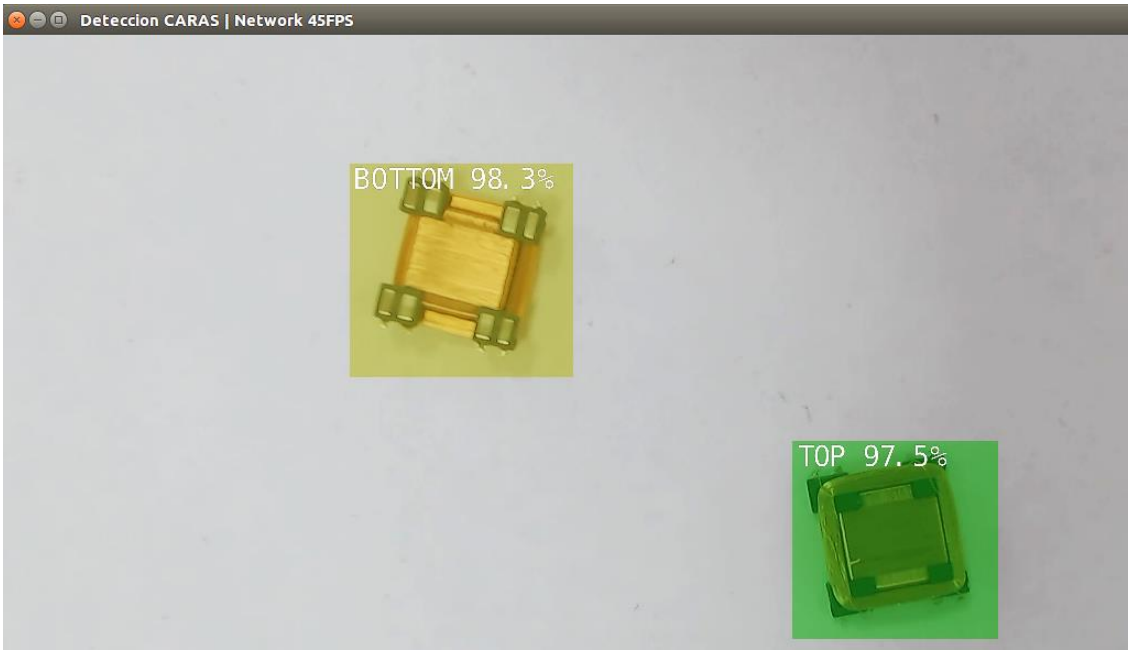

Figura 54. *Red neuronal de detección de la cara superior e inferior de la pieza 3DC11LP.*

En la ***Figura 54***, se aprecia una notable mejoría en la percepción de la pieza 3DC11LP al ser presentada sobre un fondo blanco en comparación con la ***Figura 52***, donde se encuentra sobre un fondo rojo. Este hecho se debe a que, al encontrarse la pieza de color bronce con reflejos, sobre un fondo blanco, logra resaltarla gracias al contraste generado por dichos colores.

El funcionamiento de la red fue validado manualmente en diferentes días, en condiciones de iluminación ambiente, con diferentes fondos y empleando una cámara de alta resolución y otra de baja resolución, utilizando en cada caso 100 piezas 3DC11LP por su cara superior y otras 100 piezas 3DC11LP por su cara inferior.

Las piezas se colocaron por sus distintas caras sobre el plano de trabajo en distintas posiciones y con distinta orientación empleando un threshold del 50 % en el nivel de confianza de la inferencia de la red, y siendo siempre la clasificación aportada por la red correcta en el caso de emplear una cámara de alta resolución. Por otro lado, al emplear una cámara de baja resolución, solamente en 3 ocasiones de las 200 piezas analizadas con dicha cámara se mostró una clasificación errónea, confundiendo la cara superior con la inferior. Esto fue debido a unas condiciones de iluminación desfavorables que, posteriormente, se comprobó que con una cámara de alta resolución y en unas condiciones de iluminación favorables no sucedía dicho error, por lo que se validó la red neuronal "detectcara_1" como buena.

En el caso de la red "detectcara_1", por tanto, tras la realización de numerosas pruebas en diferentes días y con distintos fondos, se comprueba que la red neuronal exhibe una notable precisión en su desempeño, siendo capaz de realizar una clara distinción detectada por la red neuronal entre la cara superior e inferior de la pieza empleando una cámara de alta resolución.

4.3.4. Optimización de los modelos.

Pese a que en los dos apartados anteriores se ha proporcionado una exhaustiva explicación sobre el diseño y la creación de una red neuronal personalizada, en el presente apartado se detallan las tecnologías involucradas (ya descritas en el apartado 3.2. Software) y los métodos empleados en la optimización de los modelos previamente vistos, y ya listos para ser ejecutados, con el fin de permitir la inferencia de alto rendimiento y, por tanto, ejecutar el modelo con un mayor rendimiento y menor tiempo de latencia.

Con la intención de asegurar un respaldo antes de ejecutar la optimización del modelo PyTorch con TensorRT, se realizó una investigación a cerca de la mejoría en términos de tiempos que dicho proceso podía conllevar. Finalmente, se encontraron los siguientes resultados en el documento [52]:

En el documento mencionado, se entrena un modelo usando PyTorch, posteriormente se convierte el modelo al formato ONNX y finalmente se emplea NVIDIA TensorRT para la inferencia.

- **Aceleración con TensorRT:**

Con objeto de comparar el tiempo en PyTorch y TensorRT para el modelo del documento [52], se compara el tiempo de inferencia y el tiempo de detección, es decir, el preprocesamiento más la inferencia más el post-procesamiento. Para ello se indica en dicho documento que la inferencia es ejecutada varias veces con objeto de obtener un tiempo medio:

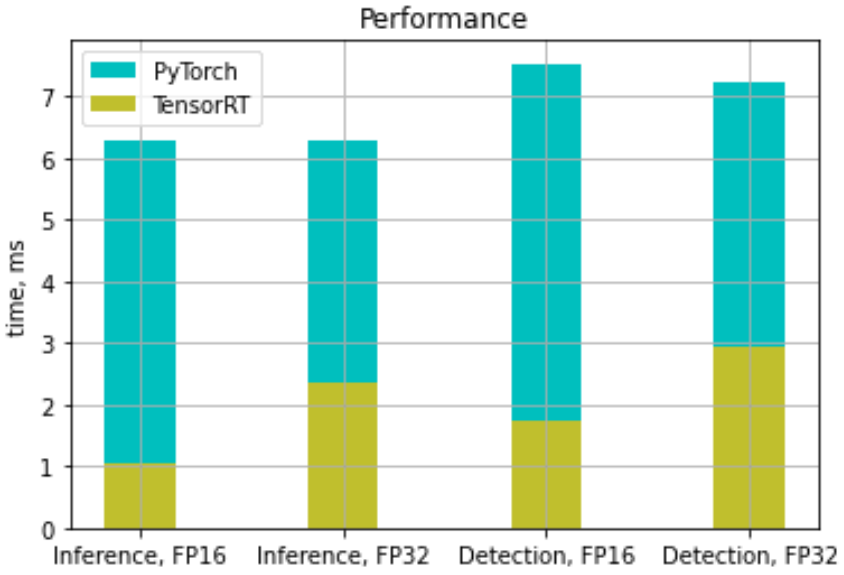

Figura 55. *Resultados obtenidos en la optimización del modelo PyTorch con TensorRT [52].*

En la figura anterior se observa que al realizar la optimización del modelo PyTorch con TensorRT se logra una aceleración del modelo de 2 a 3 veces en el modo FP32 y de 4 a 6 veces en el modo FP16 [52]. Por tanto, se comprueba que realizar una optimización del modelo PyTorch con TensorRT mejora los resultados en términos de tiempo.

Una vez realizada la comprobación anterior, a continuación, se indican las tecnologías y métodos mencionados, acompañados de una breve descripción de los pasos a seguir en la creación de las redes neuronales a fin de proporcionar un contexto, considerando que los pasos han sido detallados anteriormente[4]:

1. En primer lugar, es necesario recopilar una cantidad suficiente de imágenes de las piezas junto a su etiqueta correspondiente, 3DC11LP o 3DC14EMR-ULP en la primera red de detección creada y TOP o BOTTOM en la segunda.
2. Una vez recopilada una cantidad considerable de imágenes con sus respectivas etiquetas, se procede a entrenar el modelo con **PyTorch**.
3. Posteriormente, se convierte el modelo PyTorch previamente entrenado al formato de intercambio **ONNX (*Open Neural Network Exchange)*** que permite el intercambio

[4] La explicación detallada de los pasos y comandos necesarios para realizar la recopilación de datos, el entrenamiento y la ejecución de una red neuronal se encuentra en los puntos 4.3.2, 4.3.3 y en los apartados B y C del Anexo III.

de modelos entre diferentes marcos de trabajo y herramientas de aprendizaje profundo siendo además compatible con TensorRT.

4. **Optimización del modelo empleando recursos aportados por TensorRT:** se realiza un código de programación en el que se importan las bibliotecas **"*jetson.inference*"** y **"*jetson.utils*"** en las que se incluyen la inferencia de modelos de aprendizaje, herramientas, etc. Y posteriormente, se declara la red neuronal de detección, ya sea la red encargada de detectar dos tipos de piezas de la empresa PREMO como la encargada de identificar la cara superior e inferior de la pieza 3DC11LP o ambas. TensorRT finalmente, aplica técnicas como la fusión de capas, la cuantización de pesos, la optimización de GPU y de kernels con el fin de crear un grafo de inferencia altamente optimizado.
5. Finalmente, se ejecuta el modelo desde la terminal de Jetson Nano con el fin de emplear los **motores de ejecución de TensorRT** y aprovechar todos los recursos aportados por la plataforma Jetson empleada, consiguiendo de este modo optimizar el o los modelos, ejecutando el o los modelos a un mayor rendimiento y menor tiempo de latencia.

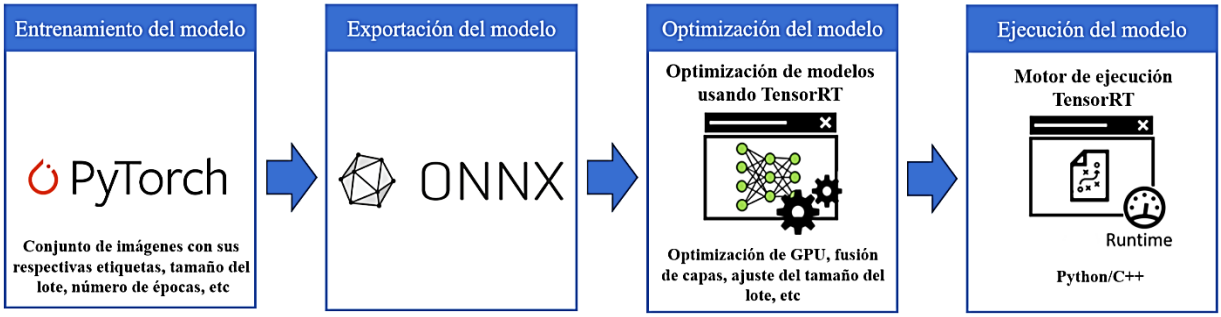

Figura 56. *Ciclo de optimización del modelo PyTorch con TensorRT.*

4.4.Robótica

Como se ha comentado en el apartado *Software*, para llevar a cabo la configuración del brazo robótico se emplea el **kit de desarrollo de software (SDK**) del robot, proporcionado por el propio fabricante. Dicho kit junto con la aplicación **UFACTORY Studio** permite realizar el control del brazo robótico, así como su posición en tiempo real e incluso realizar un código de programación en diagrama de bloques permitiendo pasarlo posteriormente a lenguaje Python. Además, clicando sobre la pestaña de configuración, dentro de la aplicación de UFACTORY, permite el ajuste de diversos parámetros necesarios para la correcta configuración del brazo robótico según su aplicación, por ejemplo:

- Selección del tipo de efector final del brazo robótico: el brazo robótico empleado permite equiparse con una gran variedad de accesorios según sea necesario para la aplicación llevada a cabo, siendo posible indicar el tipo de efector final: pinza, pinza de vacío, pinza BIO, motor lineal o sensor de par de fuerza de seis ejes. Además, permite ajustar la velocidad de apertura y cierre, así como activar la prevención de colisión del efector.
- Peso de la carga: es posible indicar el peso de la carga en kilogramos, siendo este la suma del peso de la carga más el del efector final, permitiendo el modelo empleado un peso de carga útil de hasta 5 kg. En este caso, es necesario indicar la posición del centro de gravedad de la carga en milímetros.
- Sensibilidad ante colisiones: el brazo robótico posee un rango de sensibilidad ante colisiones durante su movimiento. Dicho rango abarca desde el 0, indicando que se ha desactivado la sensibilidad ante las posibles colisiones, hasta el 5 que denota la máxima sensibilidad. El fabricante recomienda establecer una sensibilidad superior a 3.

Una vez realizado el ajuste adecuado de los parámetros del brazo robótico para la aplicación que se desea llevar a cabo, el brazo se encuentra en condiciones de comenzar a trabajar con él.

4.4.1. Movimientos del brazo robótico y rango de trabajo

A la hora de programar los movimientos del brazo robótico en Python se emplean los siguientes tipos de movimientos:

- **Movimiento cartesiano:** es empleado para mover el brazo robótico a una posición específica del espacio de trabajo dada en coordenadas cartesianas (x, y, z) expresadas en milímetros. Para llevar a cabo este movimiento se emplea la siguiente función:

```
xarm.set_position(x, y, z, speed=, mvacc=, wait=, radius=)
```

  A continuación, se explican los parámetros que deben especificarse en la función anterior:

  - **x, y, z:** coordenadas cartesianas objetivo.
  - **Speed:** velocidad a la que el brazo robótico realiza el movimiento.

- **Mvacc:** aceleración del movimiento cartesiano.
- **Wait:** se trata de un parámetro de entrada booleano en el cual debe indicarse True en caso de desear que el movimiento sea completado para continuar o False en caso contrario.
- **Radius:** radio de curvatura en el movimiento.

En definitiva, los parámetros de entrada para realizar un movimiento cartesiano son: las coordenadas cartesianas objetivo, la velocidad del movimiento, la aceleración, la espera y el radio de curvatura.

La velocidad de los movimientos cartesianos se expresa en milímetros por segundo (mm/s), siendo la velocidad máxima de 1000 mm/s.

La aceleración del movimiento se expresa en milímetros por segundo al cuadrado (mm/s^2), siendo su máxima de 50000 mm/ s^2.

- **Movimiento articular:** es empleado para mover el brazo robótico a una posición angular específica de las articulaciones del robot expresada en grados.

```
xarm.set_servo_angle(angle=, speed=, mvacc=, wait=, radius=)
```

A continuación, se explican los parámetros que deben especificarse en la función anterior:

- **Angle:** posición angular objetivo de las articulaciones. Al tratarse de un brazo robótico con seis grados de libertad, deben indicarse la posición angular de sus seis articulaciones.
- **Speed:** velocidad a la que se moverán las articulaciones hasta alcanzar la posición objetivo.
- **Mvacc:** aceleración en el movimiento de las articulaciones.
- **Wait:** se trata de un parámetro de entrada booleano en el cual debe indicarse True si desea que se complete el movimiento para continuar o False en caso contrario.
- **Radius:** radio de curvatura en el movimiento.

En definitiva, los parámetros de entrada para realizar un movimiento articular son: la posición angular destino de las articulaciones, la velocidad del movimiento, la aceleración, la espera y el radio de curvatura.

La velocidad de los movimientos articulares se expresa en grados por segundo (º/s), siendo la velocidad máxima de 180 º/s.

La aceleración del movimiento articular se expresa en grados por segundo al cuadrado (mm/s2), siendo su máxima de 1145 º/ s^2.

En relación con el brazo robótico, este posee un sistema de succión que le permite realizar la recogida de las piezas. Con el fin de activar y desactivas la ventosa, se emplean las siguientes funciones:

```
#Activivacion de la ventosa:
xarm.set_suction_cup(True, wait= ,delay_sec=0)

#Desactivacion de la ventosa:
xarm.set_suction_cup(False, wait= ,delay_sec=0)
```

Los parámetros pasados a la función encargada de realizar el control de la ventosa son los siguientes:

- **True o False:** indica la activación o desactivación de la ventosa. True indica la activación del sistema de succión, es decir, se activa la succión que permite la recogida de la pieza. Por otro lado, False indica la desactivación del sistema de succión.
- **Wait:** se trata de un parámetro de entrada booleano en el cual debe indicarse True en caso de desear que el procedimiento sea completado para continuar o False en caso contrario.
- **Delay_sec:** parámetro que indica el retraso en segundos (s) antes de la activación de la ventosa. Si dicho parámetro se iguala a cero significa que no se producirá ningún retraso antes de la activación de la ventosa.

Por otro lado, también es necesario conocer el rango de trabajo del brazo robótico con el fin de evitar programar movimientos fuera del rango, así como evitar también daños en el brazo al intentar exceder sus capacidades. A continuación, se muestra una imagen donde se aprecia el rango de trabajo del brazo robótico empleado:

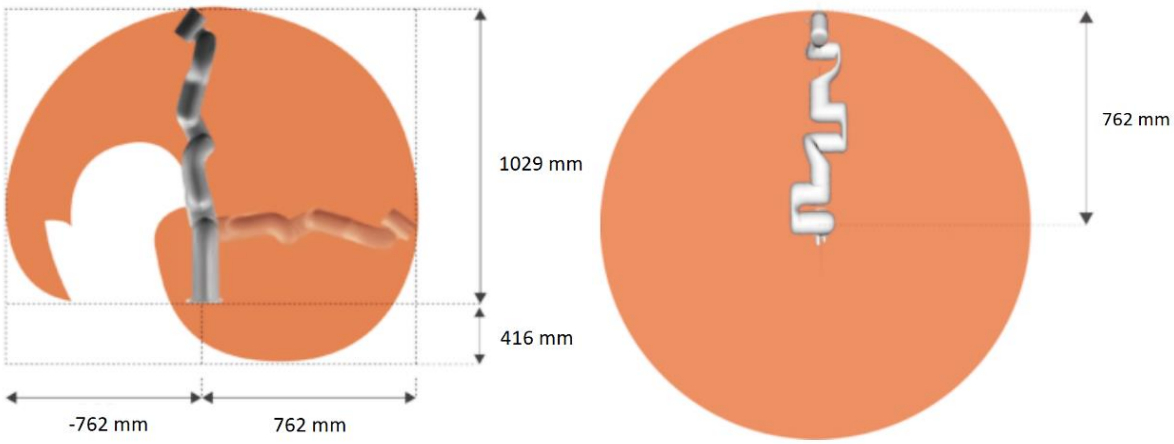

Figura 57. *Rango de movimiento del brazo robótico xArm6. [53]*

4.4.2. Sistema de referencia de coordenadas del brazo robótico y de la cámara

Con objeto de una vez detectada la pieza sobre el plano de trabajo recogerla en la posición adecuada, es necesario conocer el sistema de referencia de coordenadas tanto de la cámara empleada para la visión artificial como del brazo robótico.

El punto de origen del sistema de referencia de la cámara se sitúa en su esquina superior izquierda. De este modo, el eje X de la cámara comienza desde dicha esquina y se extiende hacia la derecha, coincidiendo con el eje Y del robot en dirección opuesta. Por tanto, con objeto de realizar el cálculo de la coordenada Y final del robot para la recogida de la pieza, se debe realizar la suma de la posición actual del eje Y del robot más la distancia negada del eje X captada por la cámara.

Por otro lado, el eje Y de la cámara comienza desde su esquina izquierda superior y se extiende hacia abajo, coincidiendo con el eje X del robot en dirección opuesta. Por tanto, con objeto de realizar el cálculo de la coordenada X final del robot para la recogida de la pieza, se suma la posición actual del eje X del robot más la distancia negada del eje Y captada por la cámara.

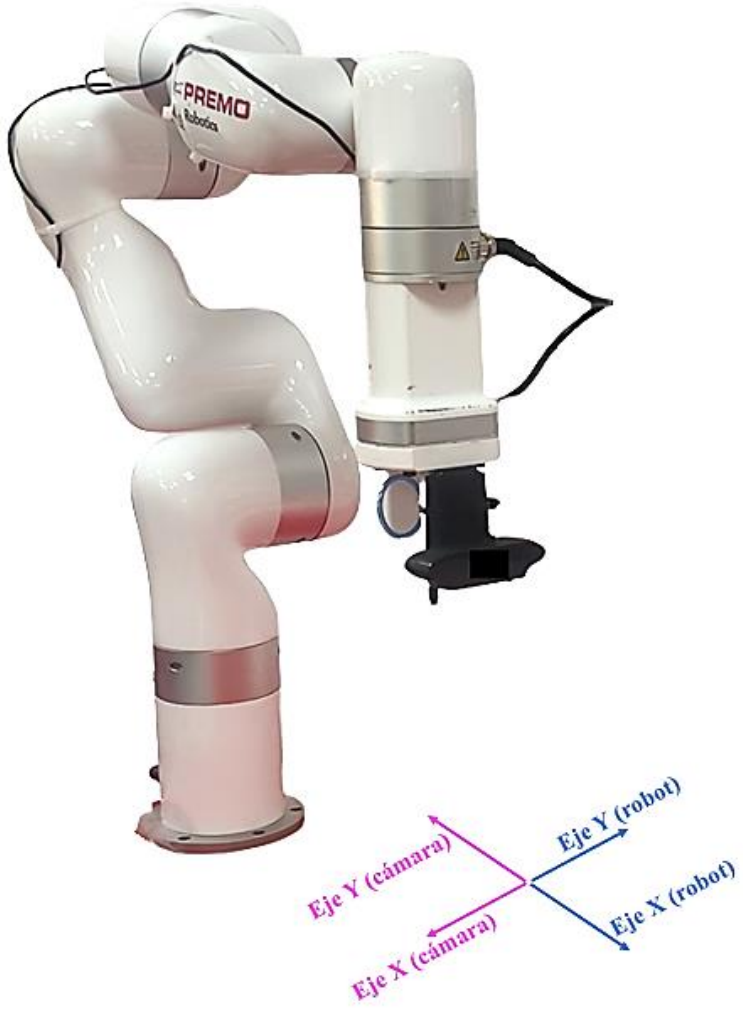

Figura 58. *Sistema de referencia del brazo robótico y de la cámara.*

Seguidamente se muestra el sistema de referencia del brazo robótico y de la cámara desde otra perspectiva, concretamente desde una imagen real capturada por la cámara depositada en el soporte ubicado en el extremo del brazo robótico:

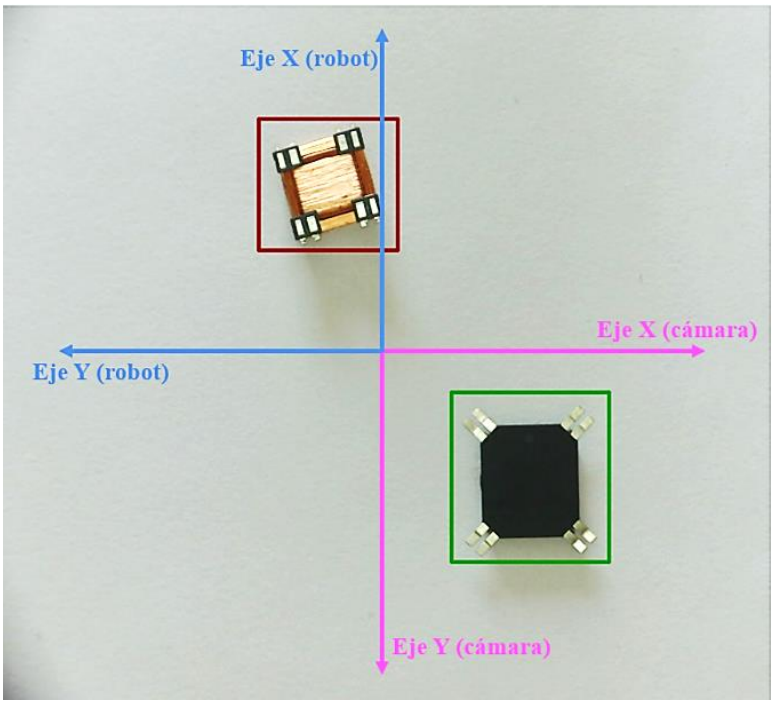

Figura 59. *Sistema de referencia del brazo robótico y de la cámara desde una imagen real capturada por la cámara.*

4.4.3. Obtención de la relación entre los píxeles y la distancia real

Conocer la relación existente entre los píxeles capturados por la cámara y la distancia en la realidad, resulta indispensable para lograr mover con éxito el brazo robótico a la posición de la pieza deseada. Debe tenerse en cuenta que la relación calculada solamente será válida para la altura a la que se calcula su valor, es decir, si se desea obtener la relación entre los píxeles y la distancia real a diferentes alturas, se requiere calcular dicha relación para todas las alturas deseadas y posteriormente almacenarlas como una constante. Cabe resaltar que no se ha empleado la autocalibración ni el uso de marcadores ArUco, elementos empleados en múltiples aplicaciones en el campo de la visión computacional, por no entrar en el alcance del proyecto, aunque sería posible incorporar un proceso de autocalibrado al sistema sin mayor inconveniente. Por dicho motivo se ha realizado la obtención de la relación manualmente.

En el presente proyecto, es necesario obtener la relación para dos alturas diferentes, una altura más cercana al plano de trabajo para almacenar la posición exacta de las piezas 3DC11LP y 3DC14EMR-ULP en su blíster correspondiente y otra a mayor altura para el rastreo y el cálculo de la posición de las piezas.

Para llevar a cabo dicha labor por medio del método empleado en el presente proyecto, es necesario seguir los siguientes pasos:

1. Desde la altura a la que se desea obtener la relación, capturar una imagen de un objeto, preferiblemente rectangular o cuadrado para obtener una mayor precisión en las medidas.
2. Medir en milímetros (mm) las dimensiones del objeto previamente capturado en el paso anterior.
3. A través de una aplicación que permita visualizar las dimensiones en unidades de píxeles de los objetos, obtener las dimensiones en píxeles del objeto capturado en el paso 1. La aplicación empleada para llevar a cabo la presente tarea es: GIMP 2.10.12 [54].
4. Una vez conocidas las dimensiones del objeto tanto en píxeles como en milímetros, se está en disposición de hallar la relación por medio de la siguiente ecuación:

$$relación_{pixeles-distancia\ real} = \frac{distancia\ real\ (mm)}{distancia\ (píxeles)} \qquad (1)$$

A continuación, se detallan los cálculos para las diferentes alturas requeridas. Como se comenta en el apartado del Objetivo, dichas alturas fueron escogidas tras empíricamente probar varias y comprobar que las alturas seleccionadas eran adecuadas para el objetivo del proyecto:

- **Altura en la que el brazo robótico presenta en su coordenada Z un valor de 300,00 mm:**

  Con objeto de llevar a cabo el cálculo de la relación entre píxeles y milímetros en la realidad, se emplea la pieza 3DC11LP.

  1. Se toma una captura de la pieza estando el brazo robótico posicionado a una altura de 300,00 mm en su eje Z.
  2. Se miden las dimensiones de la pieza. Al tratarse de una pieza cuadrada, presenta una longitud de 11,34 mm en cada uno de sus lados.
  3. Por medio de la aplicación GIMP, versión 2.10.12 se obtienen las medidas de la pieza en píxeles. Al tratarse de una pieza cuadrada, desde la altura sobre la que se está operando, presenta una longitud de 68 píxeles en cada uno de sus lados.

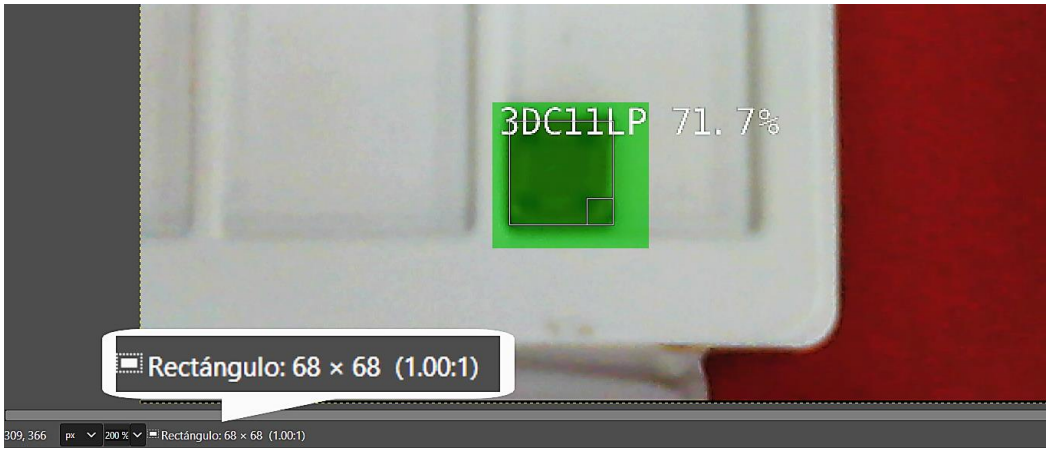

Figura 60. *Obtención de los pixeles de la pieza por medio de la aplicación GIMP (menor altura).*

4. Finalmente, se obtiene la relación entre los píxeles de la imagen y la distancia real en milímetros:

$$relación_{alt1} = \frac{11{,}34\ mm}{68\ píxeles} = 0{,}167 \qquad (2)$$

Por tanto, para una altura en la que el brazo se encuentra en su coordenada Z a 300,00 mm, se obtiene una relación entre los píxeles de la imagen y la distancia en la realidad medida en milímetros de 0,167.

- **Altura en la que el brazo robótico presenta en su coordenada Z un valor de 380,50 mm:**

Con objeto de llevar a cabo el cálculo de la relación entre píxeles y milímetros en la realidad, se emplea la pieza 3DC11LP.

1. Se toma una captura de la pieza estando el robot posicionado a una altura de 380,50 mm en su eje Z.
2. Se miden las dimensiones de la pieza. Al tratarse de una pieza cuadrada, presenta una longitud de 11,34 mm en cada uno de sus lados.
3. Por medio de la aplicación GIMP, versión 2.10.12 se obtienen las medidas de la pieza en píxeles. Al tratarse de una pieza cuadrada, desde la altura sobre la que se está operando, presenta una longitud de 48 píxeles en cada uno de sus lados.

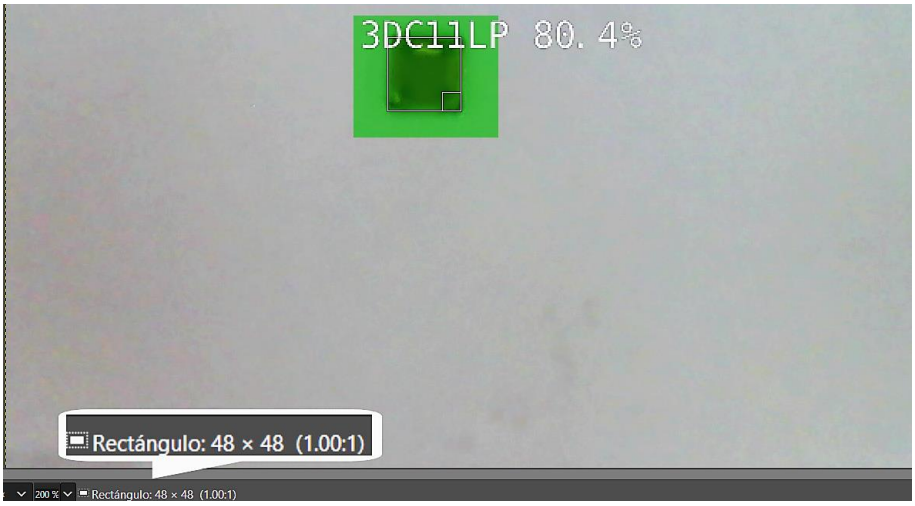

Figura 61. *Obtención de los pixeles de la pieza por medio de la aplicación GIMP (mayor altura).*

4. Finalmente, se obtiene la relación entre los píxeles de la imagen y la distancia real en milímetros:

$$relación_{alt2} = \frac{11{,}34\ mm}{48\ píxeles} = 0{,}236 \qquad (3)$$

Por tanto, para una altura en la que el brazo se encuentra en su coordenada Z a 380,50 mm, se obtiene una relación entre los píxeles de la imagen y la distancia en la realidad medida en milímetros de 0,236.

Cabe mencionar que los valores de las coordenadas del eje Z del brazo robótico mencionados en el presente apartado hacen referencia a la implementación del Clasificador de piezas, aunque las coordenadas del Clasificador y rastreador de piezas se encuentran modificadas con el fin de encontrarse el objetivo de la cámara a la misma altura desde el plano de trabajo, que en este caso es una cinta transportadora. Es por esto por lo que para la aplicación del Clasificador y rastreador de piezas se mantienen las relaciones entre los píxeles de la imagen y los milímetros en la realidad.

Por otro lado, aunque las dimensiones en relación con la pieza 3DC11LP mostradas en el apartado de Hardware, indican que la distancia existente entre dos terminales es de 11,60 mm, en el presente apartado se escoge una distancia de 11,34 mm, puesto que a la hora de obtener las medidas de la pieza en píxeles por medio de la aplicación GIMP se selecciona como referencia las esquinas del recubrimiento de cobre de la pieza, siendo la distancia de una esquina a otra de 11,34 mm.

4.4.4. Medida de los incrementos de distancia

A la hora realizar la recogida de las piezas, en primer lugar, se calcula la posición de la pieza que debe ser recogida y el brazo robótico se mueve hasta ella. En este punto, la cámara queda ubicada encima de la pieza que se desea recoger, aunque la ventosa que es lo que realmente realiza su recogida se encuentra a una determinada distancia del objetivo dicha cámara. Se trata de una distancia fija, concretamente a 51,50 mm en el eje X del robot.

Por otro lado, tras la ejecución de un conjunto de pruebas, se identifican una serie de discrepancias constantes a la hora de la recogida de las piezas. Además, debido a la falta de una alineación perfecta entre el objetivo de la cámara y el eje horizontal del plano de trabajo y el sistema de referencia del robot, se aprecia un error fijo pero distinto en cada cuadrante captado por la cámara.

Con objeto de obtener una solución robusta para el problema planteado, sería necesario llevar a cabo un proceso de ajuste de los ejes de la cámara, el plano de trabajo y el sistema de referencia del robot, lo cual implicaría: modificar el sistema de cogida de la cámara para lograr ese ajuste preciso que se necesita y desarrollar una rutina de calibración, que se sale del alcance de este trabajo.

Para solucionar dicha cuestión vía software, se llevaron a cabo las mediciones necesarias. Estas correcciones serían el resultado de la rutina de calibración mencionada anteriormente. Finalmente, el término “incremento final” se refiere a la suma de la distancia a la que se encuentra la ventosa del objetivo de la cámara más el incremento correspondiente al error cometido en el proceso.

$$\Delta_{final} = Distancia_{cámara-ventosa} + \Delta_{error} \qquad (4)$$

Como se ha comentado, el error cometido en el proceso es distinto para cada cuadrante captado por la cámara debido a la no presencia completa de paralelismo entre el objetivo de la cámara y el plano de trabajo. A continuación, se presenta la disposición de los cuadrantes en una imagen capturada por la cámara:

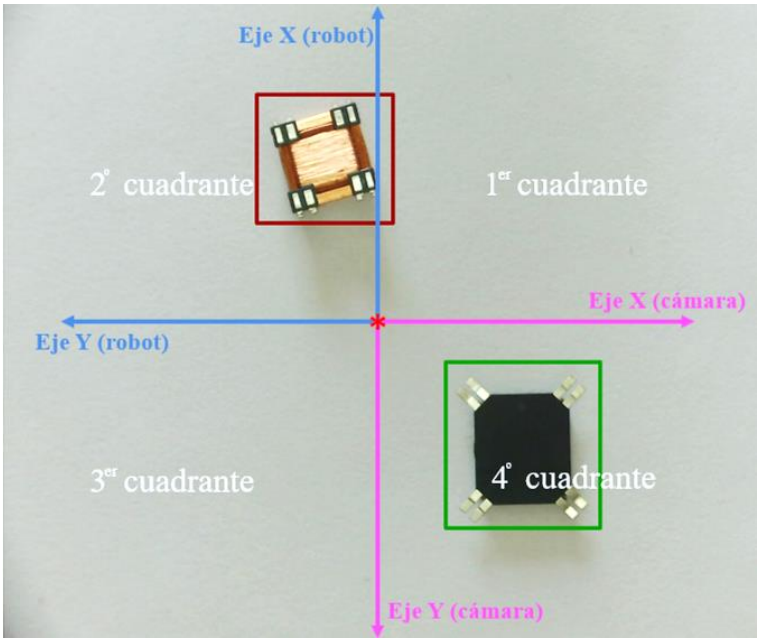

Figura 62. *Sistema de referencia de los cuadrantes de las imágenes captadas por la cámara para los incrementos.*

Concluyendo esta sección, se muestra el resultado de los incrementos calculados en la recogida y deposición de las piezas:

Incrementos finales calculados para la recogida de la pieza 3DC11LP		
	Δx (mm)	Δy (mm)
1er cuadrante	53,50	0,50
2º cuadrante	51,50	3,50
3er cuadrante	49,00	1,50
4º cuadrante	52,00	-1,50

Tabla 1. *Incrementos finales calculados para la recogida de la pieza 3DC11LP.*

Incrementos finales calculados para la recogida de la pieza 3DC14EMR-ULP		
	Δx (mm)	Δy (mm)
1er cuadrante	54,00	3,00
2º cuadrante	50,00	7,00
3er cuadrante	50,00	4,00
4º cuadrante	52,00	0,00

Tabla 2. *Incrementos finales calculados para la recogida de la pieza 3DC14EMR-ULP.*

Incrementos finales calculados para la deposición de la pieza 3DC11LP		
	Δx (mm)	Δy (mm)
1er cuadrante	52,00	-6,50
2º cuadrante	50,00	-11,00
3er cuadrante	56,00	-8,00
4º cuadrante	54,00	-6,00

Tabla 3. *Incrementos finales calculados para la deposición de la pieza 3DC11LP.*

Incrementos finales calculados para la deposición de la pieza 3DC14EMR-ULP		
	Δx (mm)	Δy (mm)
1er cuadrante	52,00	-5,00
2º cuadrante	50,00	-7,00
3er cuadrante	56,00	-4,50
4º cuadrante	54,00	-3,50

Tabla 4. *Incrementos finales calculados para la deposición de la pieza 3DC14EMR-ULP.*

Los diferentes incrementos tanto para la recogida y deposición de las piezas como para los diferentes tipos de piezas, es debido a que la recogida y colocación de las piezas no tienen el punto de referencia a la misma altura. En cuanto a que las piezas 3DC11LP y 3DC14EMR-ULP presenten diferentes incrementos, aunque en la recogida y en la deposición presenten el punto de referencia a la misma altura, se debe a la diferencia de tamaño existente entre ambas piezas, siendo normalmente mayor el error cometido en la pieza 3DC11LP por ser de menor tamaño.

4.4.5. Conexión de la cinta transportadora

Con el propósito de efectuar la implementación del Clasificador y rastreador de piezas, es necesario emplear una cinta transportadora por la que circulen las piezas a fin de visualizar la rapidez en la detección de las piezas aportada por las redes neuronales. En el presente apartado, se detallan las conexiones necesarias y los comandos requeridos para obtener un correcto funcionamiento de la cinta. A continuación, se muestra una imagen en la que aparecen las conexiones necesarias para poner en marcha la cinta transportadora.

Siendo la interpretación de los colores de los cables de las conexiones la siguiente:

- Cable amarillo: conexión a tierra.
- Cable azul: conexión a neutro.
- Cable marrón: conexión a fase:
- Cable del resto de colores: conexiones intermedias.

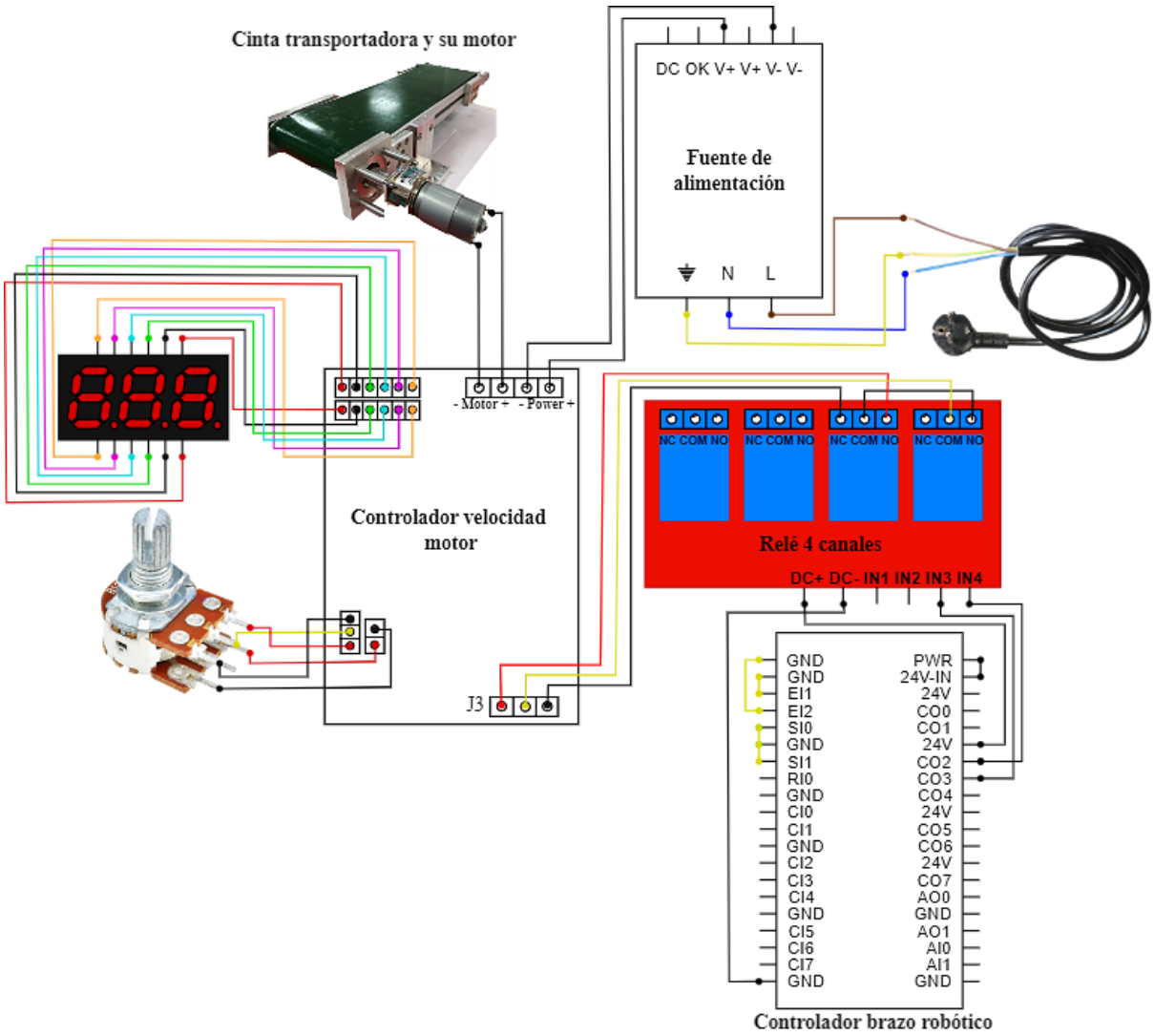

Figura 63. *Conexiones realizadas para poner en marcha la cinta transportadora.*

Tras mostrar las conexiones necesarias que garantizan el adecuado funcionamiento de la cinta, se presentan los comandos básicos para activar y desactivar la cinta, así como que la cinta se desplace en una dirección o en la otra:

- **Activación de la cinta:**

```
xarm.set_cgpio_digital(2,1, delay_sec=0)
```

- **Desactivación de la cinta:**

```
xarm.set_cgpio_digital(2,0, delay_sec=0)
```

- **Dirección de desplazamiento:** la cinta transportadora presenta la posibilidad de desplazarse en ambas direcciones. Tomando como punto de referencia el brazo robótico,

se considera que cuando se desplaza en dirección hacia el brazo robótico, está avanzando y en caso contrario, retrocediendo:

```
#Avanza hacia el brazo:
xarm.set_cgpio_digital(3,1, delay_sec=0)

#Retroceso de la cinta:
xarm.set_cgpio_digital(3,0, delay_sec=0)
```

Los parámetros pasados a la función encargada de configurar un pin digital en la interfaz GPIO del brazo robótico son los siguientes:

- **1º número:** hace referencia a uno de los pines digitales de la interfaz GPIO del brazo robótico. El propósito de cada pin depende de las conexiones realizadas previamente. En el caso presente, el pin número 2 se encarga de activar y desactivar la cinta mientras que el pin 3 es el encargado de permitir el control de la dirección de esta.
- **2º número:** hace referencia a un valor lógico previamente configurado en el pin digital. En el presente caso, el valor lógico 0 para el pin digital 2 implica la desactivación de la cinta y en el caso del pin digital 3, el retroceso de esta.
- **Delay_sec:** parámetro que indica el retraso en segundos antes de ejecutar la acción asignada al pin digital. Si dicho parámetro se iguala a cero significa que no se producirá ningún retraso antes de dicha acción.

4.5.Programación

Con miras a alcanzar los objetivos finales del proyecto se han desarrollado tres códigos, los cuales se encuentran detallados en el Anmexo IV. Dichos códigos resultan imprescindibles para la garantía del correcto funcionamiento de las implementaciones presentadas en el apartado 5.

Con el propósito de mejorar la comprensión del código de programación, se muestra en la ***Figura 64*** y en la ***Figura 65*** respectivamente un diagrama de flujo con la lógica seguida a lo largo del desarrollo del código de programación en la implementación del clasificador de piezas y del clasificador y rastreador de piezas.

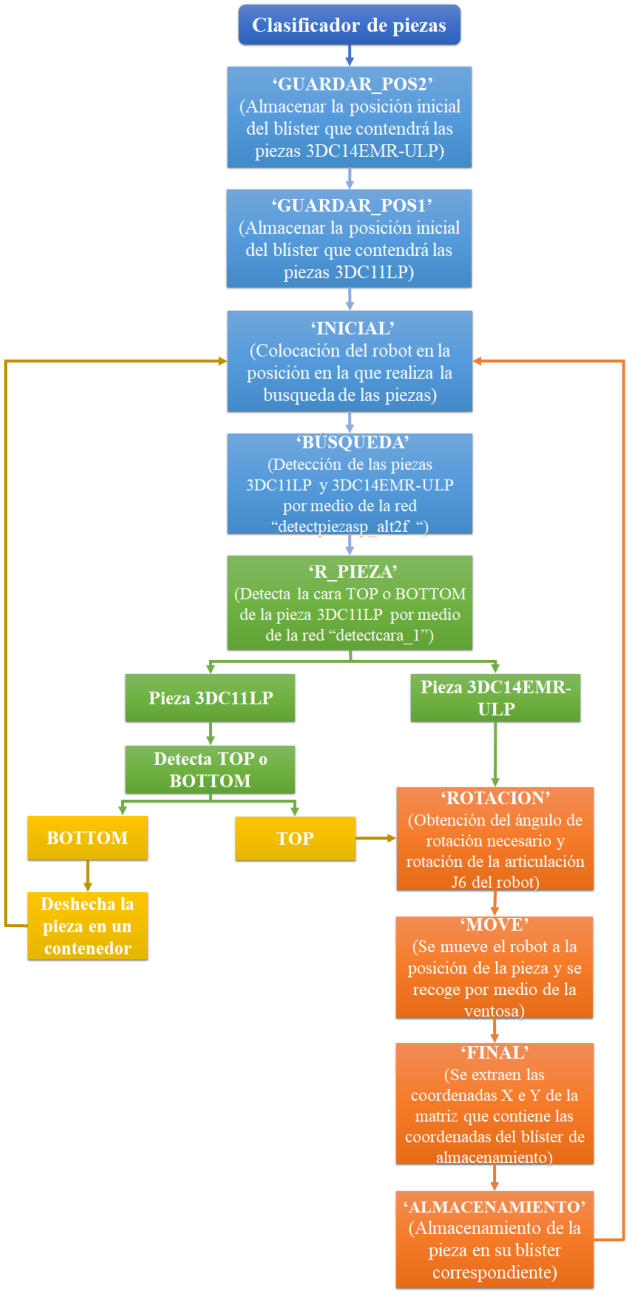

Figura 64. *Diagrama de flujo del código de programación del clasificador de piezas.*

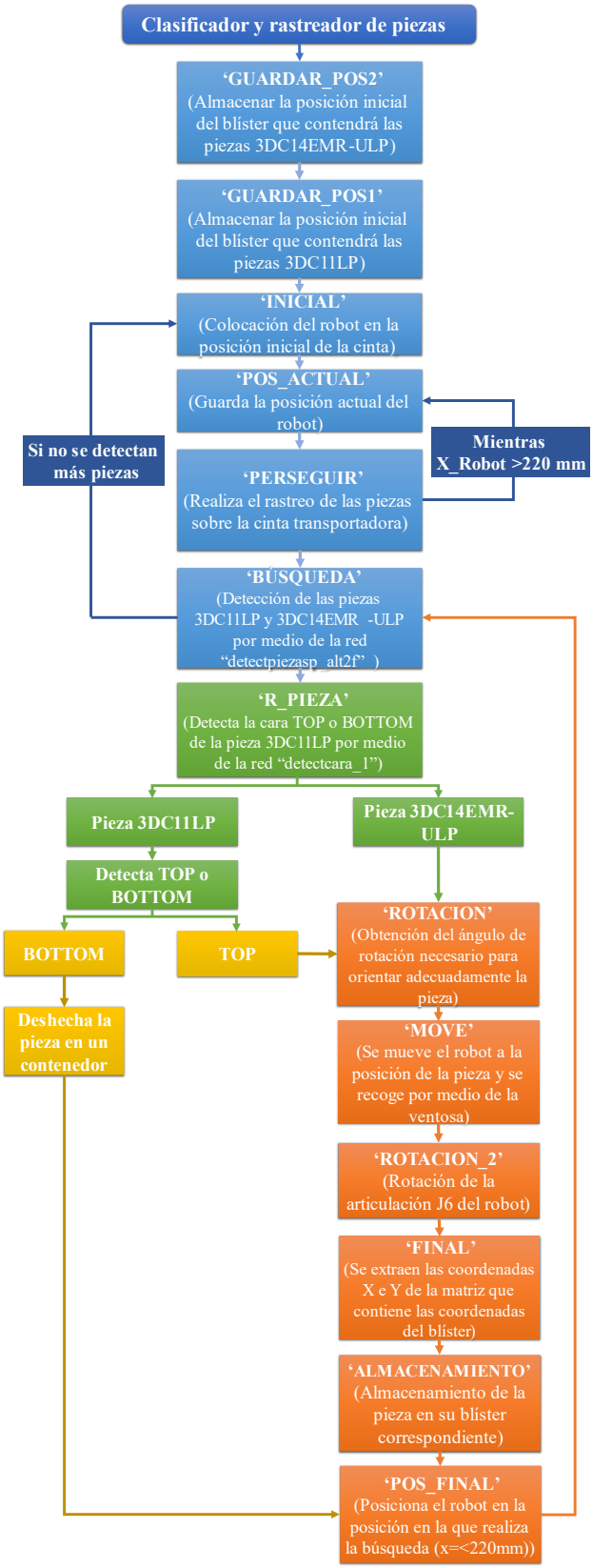

Figura 65. *Diagrama de flujo del código de programación del clasificador y rastreador de piezas.*

A continuación, se exponen los criterios establecidos seguidos de una explicación detallada de gran parte de los códigos desarrollados.

4.5.1. Declaración de los detectores empleados

Para la detección de piezas y de la cara superior e inferior de la pieza 3DC11LP se emplean dos redes neuronales preentrenadas por medio de la biblioteca *Jetson Inference*.

A continuación, se expone la sección de código donde se realiza la declaración del detector de piezas y de las caras de la pieza 3DC11LP con objeto de explicar detalladamente su contenido:

```
#Declaración del detector de piezas
net = jetson.inference.detectNet(argv=['--
model=detectpiezasp_alt2f.onnx','--labels=labels.txt','--input-
blob=input_0','--output-cvg=scores','--output-bbox=boxes'],
threshold=0.5)

#Declaración del detector de caras de la pieza 3DC11LP
net2 = jetson.inference.detectNet(argv=['--model=detectcara_1.onnx','--
labels=labels_c.txt','--input-blob=input_0','--output-cvg=scores','--
output-bbox=boxes'],threshold=0.5)
```

A continuación, se explican los parámetros que deben especificarse en las funciones anteriores:

- **jetson.inference.detectNet:** herramienta proporcionada por la biblioteca *Jetson Inference* para la detección de objetos.
- **argv:** argumento empleado para traspasar una lista de argumentos de línea de comando a la función *jetson.inference.detectNet*. Seguidamente se exponen los argumentos que deben detallarse:
    - **model:** se indica el archivo ONNX que contiene la definición con los pesos de la red neuronal.
    - **- -labels:** se indica el archivo de texto que presenta las etiquetas de las clases de la red neuronal.
    - **--input-blob:** se indica el nombre de la estructura de datos de entrada de la red.
    - **--output-cvg:** se indica el nombre de la estructura de datos de salida para la puntuación de confianza en las detecciones. Esta especifica las puntuaciones de confianza de cada detección.
    - **--output-bbox:** se indica el nombre de la estructura de datos de salida que representa las coordenadas de las cajas delimitadoras en la detección.

- **threshold:** umbral de confianza empleado en las detecciones. Este presenta el objetivo de descartar las detecciones con una puntuación de confianza por debajo del umbral especificado. Siendo más estricto a mayor valor impuesto.

Para el uso de las dos redes neuronales expuestas, es necesario que sean llamadas por medio del nombre con el que han sido definidas. A la red neuronal empleada para la detección de las piezas 3DC11LP y 3DC14EMR-ULP se le ha identificado como "net" y a la encargada de realizar la detección de la cara superior e inferior de la pieza 3DC11LP como "net2".

4.5.2. Criterio de recogida y cálculo de la posición de la pieza detectada

Con el fin de poseer un criterio fijo para la recogida de las piezas, en caso de encontrarse más de una pieza sobre el plano, se define que la pieza a recoger será aquella que se encuentre más cercana al centro de la imagen captada. Para lograr dicho objetivo, se emplea la función *calculo_distancia()* del código de programación, la cual calcula la distancia métrica entre el centro de la imagen y el centro de la pieza, teniendo como parámetros de entrada las distancias en el eje x e y desde el centro de la imagen hasta el centro de la pieza. De este modo se calcula la distancia para cada pieza detectada y se almacena toda su información en una lista llamada "piezas". Esta lista es ordenada por medio de la función *sorted* en función de la distancia previamente calculada en orden ascendente. Finalmente, una vez detectadas todas las piezas del plano y almacenadas estas en la lista "piezas" en orden ascendente en cuanto a su distancia al centro de la imagen, se selecciona la primera pieza de la lista, es decir, la más próxima al centro.

Por otro lado, resulta necesario conocer la posición real sobre el plano de trabajo de la pieza que debe ser recogida con el fin de que el robot reciba las coordenadas adecuadas para coger la pieza. Esta labor se lleva a cabo por medio de la función *calculo_coordenadas()* del código de programación, la cual presenta como valores de retorno las nuevas coordenadas X e Y del robot, en la posición en la que se encuentra la pieza a recoger y como parámetros de entrada las distancias en el eje X e Y desde el centro de la imagen hasta el centro de la pieza y las coordenadas iniciales X e Y del robot. Además, es necesario conocer la relación entre los píxeles de la imagen y los milímetros de la realidad calculada en el punto *4.4.3*, con objeto de pasar las medidas en píxeles capturadas por la cámara a distancia real en milímetros. Una vez que se tienen los parámetros de entrada indicados y la relación entre píxeles y milímetros, se procede a realizar el cálculo de las nuevas coordenadas del robot por medio de las ecuaciones que se exponen a continuación.

$$X_{final} = X_{inicial} + dy \qquad (5)$$

El cálculo de la coordenada X del robot, se realiza por medio de la ecuación (*5*) en la que se exhiben los siguientes términos:

- **X_{final}:** valor de retorno de la función *calculo_coordenadas()*. Consiste en la coordenada X calculada a la que debe moverse el robot.
- **$X_{inicial}$:** parámetro de entrada de la función *calculo_coordenadas()*. Consiste en la coordenada X en la que se encuentra el robot.

- **dy:** parámetro de entrada de la función *calculo_coordenadas()*. Consiste en la distancia existente en el eje Y entre el centro de la imagen y el centro de la pieza. Esta distancia se mide en milímetros por ser calculada previamente la relación píxeles-milímetros en base a dicha unidad métrica. Empleando la relación entre los píxeles de la cámara y los milímetros de la realidad, se obtiene la distancia que se emplea en la ecuación anterior:

$$dy = -dy \cdot relación_{pym} \qquad (6)$$

Como puede observarse en la ecuación (*5*) se realiza una suma de la coordenada X inicial del robot y la distancia en el eje Y entre el centro de la imagen y el centro de la pieza, esto se debe a que en el punto *4.4.2* se muestra que para obtener la coordenada X del robot, debe realizarse la suma de la posición inicial del eje X del robot más la coordenada negada del eje Y de la cámara debido al sistema de referencia tanto de la cámara como del robot. Debe tenerse en cuenta, que se realiza una suma en la ecuación (*5*) dado que previamente la distancia desde el centro de la imagen hasta la pieza ha sido negada, puesto que el eje X del robot y el eje Y de la cámara están alineados, pero en sentidos contrarios.

$$Y_{final} = Y_{inicial} + dx \qquad (7)$$

El cálculo de la coordenada Y del robot, se realiza por medio de la ecuación (*7*) en la que se exhiben los siguientes términos:

- **Y_{final}:** valor de retorno de la función *calculo_coordenadas()*. Consiste en la coordenada Y calculada a la que debe moverse el robot.
- **$Y_{inicial}$:** parámetro de entrada de la función *calculo_coordenadas()*. Consiste en la coordenada Y en la que se encuentra el robot.
- **dx:** parámetro de entrada de la función *calculo_coordenadas()*. Consiste en la distancia existente en el eje X entre el centro de la imagen y el centro de la pieza. Esta distancia se mide en milímetros por ser calculada previamente la relación píxeles-milímetros en base a dicha unidad métrica. Empleando la relación entre los píxeles de la cámara y los milímetros de la realidad, se obtiene la distancia que se emplea en la ecuación anterior:

$$dx = -dx \cdot relación_{pym} \qquad (8)$$

Como puede observarse en la ecuación (*7*) se realiza una suma de la coordenada Y inicial del robot y la distancia en el eje X entre el centro de la imagen y el centro de la pieza, esto se debe a que en el punto *4.4.2* se muestra que para obtener la coordenada Y del robot, debe realizarse la suma de la posición inicial del eje Y del robot más la coordenada negada del eje X de la cámara debido al sistema de referencia tanto de la cámara como del robot. Debe tenerse en cuenta, que se realiza una suma en la ecuación (*7*) dado que previamente la distancia desde el centro de la imagen hasta la pieza ha sido negada, ya que el eje Y del robot y el eje X de la cámara están alineados, pero en sentidos contrarios.

4.5.3. Orientación de las piezas

El proceso de orientación de las piezas resulta imprescindible para el correcto almacenamiento en el blíster correspondiente de los distintos tipos de piezas empleadas, puesto que se encuentran ubicadas de manera totalmente aleatoria sobre el plano de trabajo.

Para llevar a cabo dicho proceso, una vez ha sido detectada la pieza, se acerca el brazo robótico hasta la posición de la pieza a una altura determinada, para el caso que se aborda a 255,00 mm en la coordenada Z del brazo robótico, de modo que la cámara solo capte la pieza deseada con objeto de poder calcular su ángulo de orientación, evitando que algún agente externo pueda influir en el proceso.

Previamente a realizar el cálculo del ángulo necesario para orientar adecuadamente la pieza, existe una situación anterior llamada 'R_PIEZA' en la que, ya estando el brazo robótico con su coordenada Z a 255,00 mm, identifica en el caso de la pieza 3DC11LP la cara de la pieza detectada, es decir, si se trata de su cara superior o inferior. En caso de identificar la cara superior de la pieza 3DC11LP o haber identificado en la situación 'BUSQUEDA' una pieza de la clase 3DC14EMR-ULP, el programa continúa hacia la siguiente situación llamada 'ROTACION'. Sin embargo, si la red neuronal encargada de distinguir entre la cara superior e inferior de la pieza 3DC11LP, identifica que se trata de su cara inferior, el brazo robótico recoge la pieza sin realizar la rotación y la desecha en un contenedor, saltando el programa tras desechar la pieza a la situación 'INICIAL' con el fin de seguir almacenando piezas.

Por tanto, para el caso de la pieza 3DC14EMR-ULP y la cara superior de la pieza 3DC11LP, se llega a la situación 'ROTACION' con el fin de obtener el ángulo de rotación necesario para orientar adecuadamente la pieza.

En la situación 'ROTACION', en primera instancia, se procesa la imagen de la pieza de modo que se convierte a escala de grises con la finalidad de a continuación binarizar dicha imagen, donde lo que se pretende es que los valores negros de la imagen aparezcan en blanco y el resto en negro. Para ello, estando la imagen en escala de grises, es necesario establecer un umbral a fin de comparar los píxeles de la imagen en escala de grises con dicho valor umbral. Para el caso que se aborda, donde el objetivo es convertir los valores negros de la imagen a blancos, los niveles de grises que sean superiores al umbral especificado se convertirán a negro, mientras que los valores inferiores a dicho umbral se convertirán a blanco [55].

Figura 66. *Binarización de las piezas 3DC11LP y 3DC14EMR-ULP.*

Una vez detectado el contorno deseado en la imagen binaria, se establece el área mínima y el área máxima. Estos valores determinan el rango de áreas aceptables para los contornos, despreciando los contornos que se encuentren fuera de dicho rango. A continuación, si el contorno detectado se encuentra dentro del rango de áreas, se le proporciona un rectángulo delimitador por medio de la función *cv2.boundingRect()*. Seguidamente, se calcula el rectángulo delimitador mínimo que abarca el contorno de la pieza empleando *cv2.minAreaRect()* del que se almacenan sus vértices, ordenados en sentido horario, en una variable llamada "box" una vez obtenidos por medio de *cv2.boxPoints()* y se calcula el punto central del contorno.

Llegados a este punto, se muestran por pantalla tanto el rectángulo delimitador como el inclinado. Posteriormente, se calcula el centro de la pieza y se llama a la función *clasificarPuntos()* previamente definida, con la intención de calcular el ángulo de orientación de la pieza y almacenar los vértices obtenidos a fin de facilitar su manipulación.

Con objeto de poseer un criterio de orientación, se calculan las distancias desde el cuadro delimitador hasta el contorno, y posteriormente se establece una "cabeza" de la pieza (HEAD). Finalmente, se ajusta el ángulo de orientación en grados en función de la cabeza de la pieza.

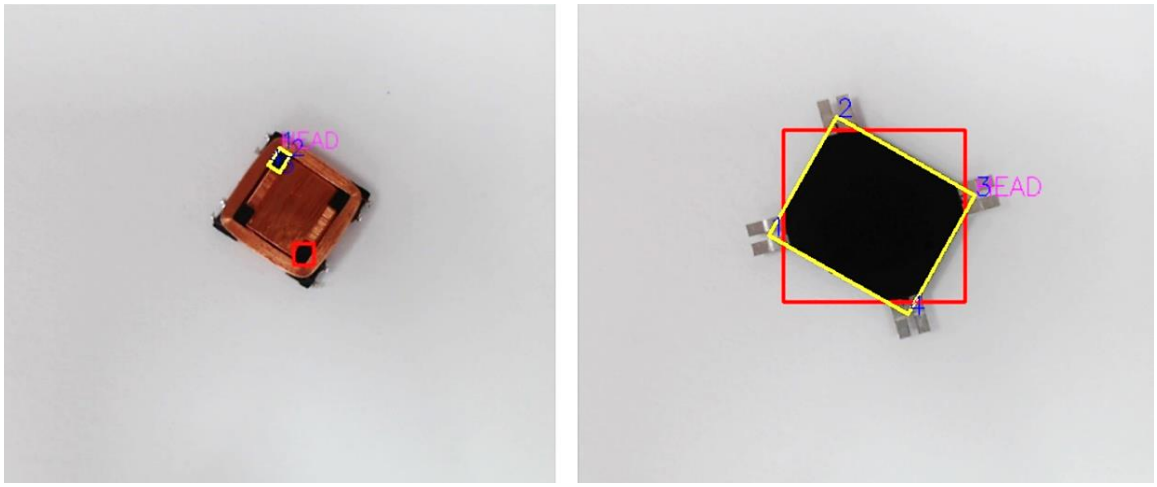

Figura 67. *Orientación de las piezas 3DC11LP y 3DC14EMR-ULP.*

Cabe mencionar que, a la hora de realizar la rotación de las piezas para su correcto almacenamiento, existe una diferencia en la ejecución del proceso entre la primera implementación (5.2. Clasificador de piezas) y la tercera (5.4. Clasificador y rastreador de piezas).

Dicha diferencia radica en que, en el proceso de rotación de la primera implementación, es decir en el Clasificador de piezas, el brazo robótico se acerca a la posición de la pieza, obtiene el ángulo de orientación como se ha desarrollado anteriormente, rota su articulación seis y a continuación recoge la pieza. Sin embargo, en la tercera implementación, es decir en el Clasificador y rastreador de piezas, el brazo robótico se acerca a la posición de la pieza, obtiene el ángulo de orientación como se ha desarrollado anteriormente, recoge la pieza y es en este instante en el corrige el ángulo de rotación.

Esta modificación se llevó a cabo debido a que, en la primera implementación, al realizar la rotación de la articulación seis del brazo robótico, al no estar completamente la ventosa perpendicular al plano de trabajo, se percibió que si la inclinación que debía corregir el brazo robótico era grande (aproximadamente superior a 30º) se apreciaba un error en la recogida de la pieza, siendo este casi imperceptible. Aunque dicho error no es de gran magnitud, se decidió corregirlo debido a la recogida de la pieza 3DC11LP, ya que, si esta no es cogida justamente por el centro de la pieza puede caerse debido a su textura y tamaño.

Es importante mencionar, que los valores de las coordenadas del brazo robótico mencionados en el presente apartado hacen referencia a la implementación del Clasificador de piezas, aunque las coordenadas del Clasificador y rastreador de piezas se encuentran modificadas de modo que se mantienen las relaciones debido a que el objetivo de la cámara se encuentra ubicado a la misma altura respecto al plano de trabajo, que en este caso es una cinta transportadora.

4.5.4. Posicionamiento en el blíster

Con miras a alcanzar un correcto posicionamiento de las piezas 3DC11LP y 3DC14EMR-ULP en su blíster correspondiente, es necesario realizar previamente a este paso el almacenamiento de la posición exacta del blíster, ya que se prevé que este, aunque se encuentre en una ubicación fija, pueda ser mínimamente movido debido a agentes externos.

Para llevar a cabo el almacenamiento de la ubicación de los blísteres empleados, asegurando así un correcto almacenamiento de las piezas, se utilizan las situaciones definidas como ‘GUARDAR_POS1’ y ‘GUARDAR_POS2’.

La situación ‘GUARDAR_POS1’ presenta como objetivo almacenar la posición inicial del blíster que contendrá las piezas 3DC11LP. Para llevar a cabo dicha labor, una pieza del tipo mencionado se encuentra ubicada en la posición que se tomará como inicial del blíster. La cámara ubicada en el extremo del brazo robótico detectará la pieza ubicada en el blíster por medio de la red neuronal encargada de realizar la detección de las piezas, calculando y almacenando su ubicación por medio de la función *calculo_coordenadas_blister()* del código de programación, la cual posee como parámetros de entrada las distancias en el eje X e Y desde el centro de la imagen hasta el centro de la pieza y las coordenadas actuales X e Y del robot presentando como valores de retorno las nuevas coordenadas X e Y del robot, en la posición en la que se encuentra la pieza depositada en el blíster, es decir, la posición tomada como inicial para el almacenamiento de la pieza 3DC11LP en el blíster.

De forma análoga a la anterior, se almacena la posición tomada como inicial en el blíster empleado para el almacenamiento de la pieza 3DC14EMR-ULP.

Además, es necesario conocer la relación entre los píxeles de la imagen y los milímetros de la realidad calculada en el punto *4.4.3* con objeto de pasar las medidas en píxeles capturadas por la cámara a distancia real en milímetros. Una vez que se tienen los parámetros de entrada indicados y la relación entre píxeles y milímetros, se procede a realizar el cálculo de las nuevas coordenadas del robot que indican las posiciones iniciales de los blísteres por medio de las ecuaciones que se exponen a continuación.

$$X_{ini\_blister} = X_{actual\_robot} + dy \qquad (9)$$

El cálculo de la coordenada X del robot que define la posición inicial del robot en dicho eje, se realiza por medio de la ecuación (*9*) en la que se exhiben los siguientes términos:

- **$X_{ini\_blister}$:** valor de retorno de la función *calculo_coordenadas_blister()*. Consiste en la coordenada X calculada a la que debe moverse el robot para alcanzar la posición inicial del blíster.
- **X_{actual_robot}:** parámetro de entrada de la función *calculo_coordenadas_blister()*. Consiste en la coordenada X en la que se encuentra el robot.
- **dy:** parámetro de entrada de la función *calculo_coordenadas_blister()*. Consiste en la distancia existente en el eje Y entre el centro de la imagen y el centro de la pieza. Esta distancia se mide en milímetros por ser calculada previamente la relación píxeles-milímetros en base a dicha unidad métrica. Empleando la relación entre los píxeles de la cámara y los milímetros de la realidad, se obtiene la distancia que se emplea en la ecuación anterior:

$$dy = -dy \cdot relación_{pym} \qquad (10)$$

Como puede observarse en la ecuación (*9*) se realiza una suma de la coordenada X inicial del robot y la distancia en el eje Y entre el centro de la imagen y el centro de la pieza, esto se debe a que en el punto *4.4.2.* se muestra que para obtener la coordenada X del robot debe realizarse la suma de la posición inicial del eje X del robot más la coordenada negada del eje Y de la cámara debido al sistema de referencia tanto de la cámara como del robot. Debe tenerse en cuenta que se realiza una suma en la ecuación (*9*) dado a que previamente la distancia desde el centro de la imagen hasta la pieza ha sido negada, ya que el eje X del robot y el eje Y de la cámara están alineados, pero en sentidos contrarios.

$$Y_{ini\_blister} = Y_{actual\_robot} + dx \qquad (11)$$

El cálculo de la coordenada Y del robot que define la posición inicial del robot en dicho eje, se realiza por medio de la ecuación (*11*) en la que se exhiben los siguientes términos:

- **$Y_{ini\_blister}$:** valor de retorno de la función *calculo_coordenadas_blister()*. Consiste en la coordenada Y calculada a la que debe moverse el robot para alcanzar la posición inicial del blíster.
- **Y_{actual_robot}:** parámetro de entrada de la función *calculo_coordenadas_blister()*. Consiste en la coordenada Y en la que se encuentra el robot.
- **dx:** parámetro de entrada de la función *calculo_coordenadas_blister()*. Consiste en la distancia existente en el eje X entre el centro de la imagen y el centro de la pieza. Esta distancia se mide en milímetros por ser calculada previamente la relación píxeles-milímetros en base a dicha unidad métrica. Empleando la relación entre los píxeles de la cámara y los milímetros de la realidad, se obtiene la distancia que se emplea en la ecuación anterior:

$$dx = -dx \cdot relación_{pym} \qquad (12)$$

Como puede observarse en la ecuación (*11*) se realiza una suma de la coordenada Y inicial del robot y la distancia en el eje X entre el centro de la imagen y el centro de la pieza, esto se debe a que en el punto *4.4.2.* se muestra que para obtener la coordenada Y del robot debe realizarse la suma de la posición inicial del eje Y del robot más la coordenada negada del eje X de la cámara debido al sistema de referencia tanto de la cámara como del robot. Debe tenerse en cuenta que se realiza una suma en la ecuación (*11*) dado a que previamente la distancia desde el centro de la imagen hasta la pieza ha sido negada, ya que el eje Y del robot y el eje X de la cámara están alineados, pero en sentidos contrarios.

Una vez conocida la posición inicial del blíster, su número de columnas y de filas, así como la distancia que separa el centro de una columna de otra además de la distancia existente que divide el centro de una fila de otra, se crea una matriz de coordenadas por medio de un bucle anidado que recorre cada columna y cada fila de la matriz. Para cada posición de la matriz, se calculan las coordenadas X e Y centrales de la posición y se agregan a una la lista llamada "Matriz" que inicialmente se inicializó como vacía. Esto es posible llamando desde el programa principal al subprograma 'Matriz_p1' para el caso de la pieza 3DC11LP o 'Matriz_p2' para el caso de la pieza 3DC14EMR-ULP, que previamente han debido ser importados para que dichos subprogramas puedan ser empleados en el programa principal:

```
from Matriz_p1 import *
from Matriz_p2 import *
```

Posteriormente, una vez alcanzada la situación 'FINAL', se extraen las coordenadas X e Y de la matriz que contiene las coordenadas del blíster de almacenamiento con objeto de seguidamente, en la situación de 'ALMACENAMIENTO' mover el robot a la posición en la que debe depositar la pieza que previamente ha sido recogida. Debe tenerse en cuenta que el criterio de almacenamiento escogido es rellenar las filas de una en una de modo que cuando la fila esté completa salte a la fila siguiente, es decir, en primer lugar, se rellenan todas las columnas de una fila y a continuación se pasa a la siguiente fila.

Debe tenerse en cuenta además que, aunque se emplea el mismo tipo de blíster para ambas piezas, el número de columnas y filas para el almacenamiento de la pieza 3DC11LP se declara como inferior al usado para almacenar las piezas 3DC14EMR-ULP debido a la limitación del rango de trabajo del robot. De modo que, desde la perspectiva del programa, para realizar el almacenamiento de la pieza 3DC11LP se emplea un blíster de 8 filas por 10 columnas, de las que se utilizan 6 filas y 6 columnas.

4.5.5. Rastreo de las piezas

Con el propósito de comprobar que el factor limitante es el movimiento del robot y no el tiempo requerido por la red neuronal para detectar las piezas, se opta por realizar un sistema rastreador de piezas que posteriormente se integra al clasificador complementándolo por medio del uso de una cinta transportadora.

Con objeto de llevar a cabo la función del rastreador de piezas, se emplean las situaciones 'POS_ACTUAL' y 'PERSEGUIR' del código de programación. A continuación, se explican dichas situaciones.

En el instante en el que en el código de programación se llega a la situación 'POS_ACTUAL', se emplea la función *xarm.get_position()* con el fin de obtener la posición actual del robot y almacenarla en una variable llamada "*pose_ini*". Posteriormente, se almacenan por separado las coordenadas actuales del brazo robótico en los ejes X, Y y Z, con el fin de facilitar su manipulación posterior a lo largo del programa.

Una vez almacenada la posición actual del brazo robótico, se llega a la situación 'PERSEGUIR' donde su función es obtener las nuevas coordenadas de la pieza con el propósito de mover el brazo robótico hasta su nueva ubicación. Dicho escenario, en el caso de la implementación del Rastreador de piezas, se repite durante un bucle infinito; no obstante, en la implementación del Clasificador y rastreador de piezas, el bucle se repite hasta que el brazo robótico alcanza los 220,00 mm en su coordenada X pasando de este modo a la posición de 'BUSQUEDA'.

Con el fin de lograr la nueva ubicación de la pieza, se emplea la función *calculo_coordenadas()* vista anteriormente con las coordenadas actuales del robot y las distancias en el eje X e Y desde el centro de la imagen hasta el centro de la pieza como parámetros de entrada, obteniendo de este modo las nuevas coordenadas X e Y de la ubicación de la pieza.

Una vez obtenidas las coordenadas X e Y de la pieza, ya que la coordenada Z sigue siendo la misma, se está en disposición de utilizar la siguiente función donde se emplea la cinemática inversa, con objeto de calcular los ángulos de las articulaciones del brazo robótico a partir de las coordenadas tridimensionales (X, Y, Z) aportadas:

```
angles=xarm.get_inverse_kinematics([x_final, y_ini, z_final,179.3,-0.5,3.9])
```

Posteriormente, obtenidos ya los valores de los ángulos de las articulaciones del brazo robótico para alcanzar la posición destino, se está en disposición de mover el brazo robótico a dicha posición por medio un movimiento articular siguiendo la siguiente función:

```
xarm.set_servo_angle(angle=angles,speed=params2['angle_speed'],
mvacc=params2['angle_acc'], wait=False, radius=-1.0)
```

Se realiza un movimiento articular y no uno cartesiano, puesto que se pretende mostrar la rapidez de detección de las piezas por medio de la red neuronal y como se ha comentado en el apartado 4.4.1, el movimiento articular presenta mayor velocidad que el cartesiano por lo que el movimiento articular presenta mayor interés en el presente contexto.

Por otro lado, cabe destacar que el movimiento articular sería conveniente emplearlo en todos los movimientos que debe realizar el brazo robótico, ya que lo que se pretende es realizar el proceso en el menor tiempo posible aprovechando la rapidez en la detección de las piezas gracias a la red neuronal aunque esto no es posible debido a que como indica Daniel Wang, miembro del equipo UFACTORY, en [56] si se desea calcular los ángulos de las articulaciones del brazo robótico en un punto, este punto no debe estar a más de 10 mm de la posición actual del brazo robótico, ya que en caso de estar a una distancia superior a la mencionada, la

cinemática inversa por medio de la función mostrada anteriormente, no llegaría a ningún resultado válido. Dicho de otro modo, para obtener buenos resultados por medio de la función *xarm.get_inverse_kinematics()*, el punto que desea calcularse debe estar a una distancia inferior a 10 mm del punto actual en el que se encuentra el brazo robótico.

De este modo, la función *xarm.get_inverse_kinematics()* no puede ser empleada en la recogida de las piezas, puesto que no se sabe con certeza si la posición de la pieza que debe ser recogida es superior a 10 mm, siendo esto lo más probable, ya que el espacio que abarca el objetivo de la cámara es muy superior a dicha medida.

5. Implementaciones y resultados obtenidos

En la presente sección, se exponen los resultados más destacados obtenidos a lo largo de la ejecución del proyecto, así como la explicación de las implementaciones desarrolladas con el fin de cumplir los objetivos previamente planteados.

5.1. Resultados destacables de las redes neuronales

En el actual apartado, se exponen los resultados obtenidos en relación con la red neuronal "detectpiezasp_alt2f" con el fin de evitar la redundancia de información, dado que la otra red neuronal empleada en la ejecución del proyecto "detectcara_1" presenta resultados muy similares en cuanto a tiempos de detección.

En primer lugar, en la siguiente figura se muestra el código QR de un video en el que se puede observar el comportamiento de la red neuronal encargada de realizar la detección de las piezas 3DC11LP y 3DC14EMR-ULP que se encuentran sobre el plano de trabajo, en el campo visual de la cámara.

Figura 68. *Código QR del funcionamiento de la red neuronal "detectpiezasp_alt2f" de detección de piezas PREMO.*

En dicho video se puede contemplar el comportamiento de la red neuronal en tiempo real, donde las piezas se encuentran aleatoriamente sobre el plano de trabajo y la red neuronal identifica los dos tipos de componentes, indicando la clase a la que pertenecen. Además, una mano cubre y descubre las piezas con el fin de comprobar la rapidez de la red a la hora de realizar la detección.

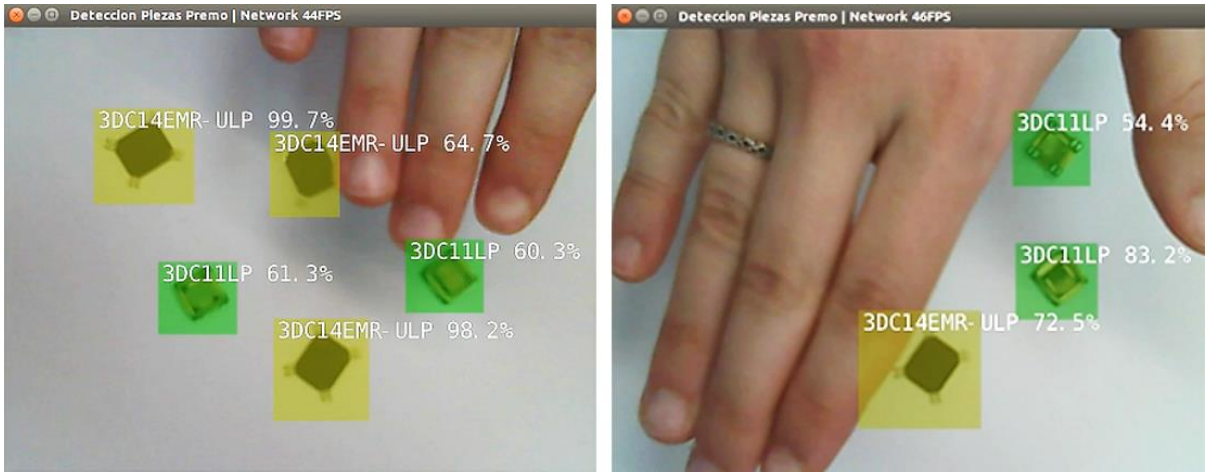

Figura 69. *Resultados red neuronal detección de piezas PREMO.*

Posteriormente, el código empleado únicamente para realizar la detección de las piezas se modifica a fin de emplear la biblioteca "cv2" de OpenCV en lugar de la biblioteca "jetson", lo cual aporta mayor compatibilidad, más documentación a la hora de realizar una búsqueda de información sobre alguna cuestión, etc.

Al realizar dicha modificación, se añade la función de identificación de la pieza más cercana al centro de la imagen, mostrando por pantalla una circunferencia sobre dicha pieza. Asimismo, en la esquina superior izquierda de la imagen se exhibe el tiempo en milisegundos (ms) que tarda en realizar la red neuronal la detección de las piezas.

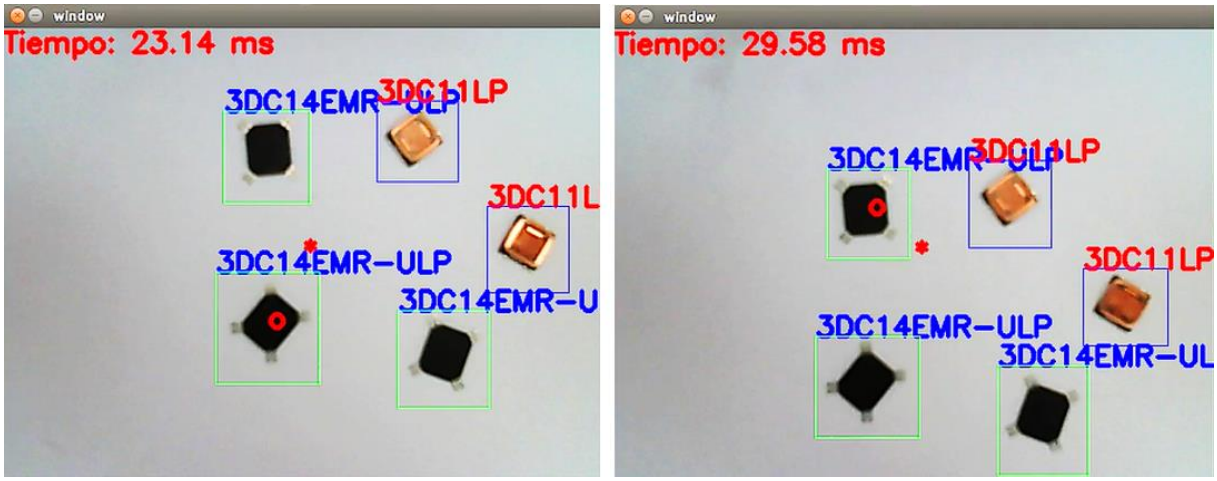

Figura 70. *Resultados de tiempos (ms) de detección de la red neuronal "detectpiezasp_alt2f".*

En el caso de la red "detectpiezasp_alt2f", tras la realización de numerosas pruebas en diferentes días y con distintos fondos, se comprueba que el tiempo medio de inferencia de la red neuronal es de 26 ms, no siendo en ningún caso superior a 30 ms, y siendo siempre la clasificación aportada por la red correcta.

También fue comprobado que el grueso del tiempo de procesamiento, el cual incluye inferencia más búsqueda de referencias más el cálculo del ángulo de orientación, se lo lleva la inferencia con valores del orden de 26 ms mientras que el resto de los cálculos se realizan, en total, en tiempos inferiores a 1 ms.

A continuación, se expone el código QR de un video en el que se muestran los tiempos de inferencia de la red neuronal *"detectpiezasp_alt2f"* en diferentes casos:

Figura 71. *Código QR de los resultados de tiempos (ms) de detección de la red neuronal "detectpiezasp_alt2f".*

5.2. Clasificador de piezas

La implementación denominada *Clasificador de piezas* presenta el objetivo de, estando un conjunto de piezas colocadas aleatoriamente sobre el plano de trabajo, detectarlas, orientarlas y depositarlas en su blíster correspondiente con el fin de almacenarlas en la posición y orientación adecuada.

Para llevar a cabo dicha implementación, es necesario emplear el código de programación reflejado en el apartado B del Anexo IV. A continuación, se detalla la ejecución de la aplicación que se aborda en el presente apartado, sin entrar en detalle de las funciones y situaciones empleadas, ya explicadas previamente:

En primer lugar, se almacena la posición exacta de cada uno de los blísteres empleados para el almacenamiento de las piezas 3DC11LP y 3DC14EMR-ULP con objeto de cada vez que se ejecuta la aplicación, conocer la ubicación de los blísteres, ya que se prevé que estos puedan ser movidos debido a agentes externos al control del usuario.

Posteriormente, el brazo robótico se posiciona en una ubicación previamente fijada para realizar la detección de las piezas, presentando la coordenada Z del brazo en dicha posición un valor de 380,50 mm. Una vez alcanzada la posición, la red neuronal encargada de realizar la detección

de las piezas 3DC11LP y 3DC14EMR-ULP, identifica todas las que es capaz de visualizar la cámara para seguidamente realizar la recogida de la pieza detectada que se encuentra más cerca del centro de la imagen.

Previamente a realizar el cálculo del ángulo necesario para orientar adecuadamente la pieza, existe una situación llamada 'R_PIEZA' en la que, ya estando el brazo robótico con su coordenada Z a 255,00 mm, identifica en el caso de la pieza 3DC11LP la cara de la pieza detectada, es decir, si se trata de su cara superior o inferior. En caso de identificar la cara superior de la pieza 3DC11LP o haber identificado en la situación 'BUSQUEDA' una pieza de la clase 3DC14EMR-ULP, el programa continúa hacia la siguiente situación llamada 'ROTACION'. Sin embargo, si la red neuronal encargada de distinguir entre la cara superior e inferior de la pieza 3DC11LP, identifica que se trata de su cara inferior, el brazo robótico recoge la pieza sin realizar la rotación necesaria y la desecha en un contenedor, saltando el programa tras desechar la pieza a la situación 'INICIAL' con el fin de seguir almacenando piezas.

A continuación, habiendo identificado la red neuronal la pieza 3DC14EMR-ULP o la cara superior de la pieza 3DC11LP, a la hora de realizar la recogida de la pieza en la presente propuesta, el brazo robótico ya ubicado en la posición de la pieza identificada como la más cercana al centro de la imagen, obtiene el ángulo de orientación como se ha desarrollado detalladamente en el punto 4.5.3, rota su articulación seis los grados que sean necesarios y a continuación recoge la pieza.

Llegados a este punto, en el que el brazo robótico posee la pieza adecuadamente orientada, la deposita en el blíster correspondiente, en la posición adecuada. De modo que la pieza queda almacenada de manera apropiada en su respectivo blíster, en la posición y orientación adecuada.

A continuación, se muestra el código QR del vídeo donde es posible comprobar la puesta en marcha de la implementación desarrollada:

Figura 72. *Código QR del funcionamiento de la implementación Clasificador de Piezas.*

El funcionamiento del clasificador de piezas fue validado con una cámara de baja resolución, empleando 24 piezas de cada tipo, lo cual rellena 3 filas completas del blíster en el que se almacenan las piezas 3DC14EMR-ULP y 4 filas completas del blíster en el que se almacenan las piezas 3DC11LP, no intentando rellenar ambos blísteres completamente debido a la limitación del rango de movimiento del brazo robótico. La validación se llevó a cabo realizando el mismo procedimiento mencionado, en tres días diferentes, en unas condiciones de iluminación ambientales, obteniendo en todos los casos una clasificación de las piezas correcta.

5.3. Rastreador de piezas

La implementación denominada *Rastreador de piezas* presenta el objetivo de que el brazo robótico realice el seguimiento de la pieza, siendo esta movida aleatoriamente por el plano de trabajo con el fin de observar la rapidez en la detección de las piezas proporcionada por el uso de la computación acelerada. Al aplicar la cinemática inversa en los movimientos del brazo robótico con objeto de efectuar movimientos articulares, dejan estos de ser prácticamente el factor limitante en términos de tiempo en dicha aplicación y puede apreciarse adecuadamente la velocidad a la que trabaja el sistema.

Para ejecutar la presente implementación, es necesario emplear el código de programación reflejado en el apartado C del Anexo IV.

A continuación, se detalla la propuesta mencionada, sin entrar en detalle de las funciones y situaciones empleadas, ya explicadas previamente:

En primer lugar, el brazo robótico se posiciona en una ubicación previamente fijada, presentando la coordenada Z del brazo en dicha posición un valor de 380,50 mm, a fin de mantener la relación entre los píxeles de la imagen y los milímetros de la realidad previamente calculados para dicha altura.

Seguidamente, la red neuronal encargada de detectar piezas del tipo 3DC11LP y 3DC14EMR-ULP, detecta la pieza y almacena sus datos con miras a calcular la posición de la pieza detectada por medio de la función *calculo_coordenadas()* con objeto de obtener la nueva posición a la que debe moverse el robot para alcanzar la pieza.

Una vez obtenidas las coordenadas X e Y de la pieza, puesto que la coordenada Z sigue siendo la misma, se está en disposición de utilizar la siguiente función *get_inverse_kinematics()* donde se emplea la cinemática inversa, con objeto de calcular los ángulos de las articulaciones del brazo robótico a partir de las coordenadas tridimensionales (X, Y, Z) aportadas.

Posteriormente, obtenidos ya los valores de los ángulos de las articulaciones del brazo robótico para alcanzar la posición destino, se está en disposición de mover el brazo robótico a dicha posición por medio un movimiento articular. Dado que el movimiento articular presenta mayor velocidad que el cartesiano como se comenta en el apartado 4.4.1, el movimiento articular presenta mayor interés en el presente contexto.

Cabe recordar que como se mencionó en el apartado 4.5.5, para obtener buenos resultados por medio de la función *xarm.get_inverse_kinematics()*, el punto que desea calcularse debe estar a una distancia inferior a 10 mm del punto actual en el que se encuentra el brazo robótico [56], no siendo esto un problema en este caso debido a que como se ha visto anteriormente, la detección de las piezas se realiza en un tiempo aproximado de 26 milisegundos (ms), entendiendo que las piezas no van a ser movidas a una velocidad superior.

A continuación, se muestra el código QR del vídeo donde es posible comprobar el funcionamiento de la implementación desarrollada:

Figura 73. *Código QR del funcionamiento de la implementación Rastreador de Piezas.*

En el video en el que se expone el funcionamiento del rastreador de piezas es posible observar que es el robot el que limita el seguimiento fluido de la pieza.

El funcionamiento del rastreador de piezas fue validado en diferentes días con una cámara de baja resolución, empleando los dos tipos de piezas que es capaz de detectar la red neuronal "detectpiezasp_alt2f", en unas condiciones de iluminación ambientales, obteniendo en todos los casos un rastreo adecuado de la pieza.

5.4. Clasificador y rastreador de piezas

La implementación denominada *Clasificador y rastreador de piezas* presenta el objetivo de, estando un conjunto de piezas colocadas aleatoriamente sobre una cinta transportadora, detectarlas, seguirlas a lo largo de la cinta, orientarlas y depositarlas en su blíster correspondiente con el fin de almacenarlas correctamente.

Para ejecutar dicha implementación, es necesario emplear el código de programación reflejado en el apartado D del Anexo IV.

A continuación, se detalla la ejecución de la aplicación mencionada, sin entrar en detalle de las funciones y situaciones empleadas, ya explicadas previamente:

En primer lugar, al igual que en la implementación del Clasificador de piezas, se almacena la posición exacta de cada uno de los blísteres empleados para el almacenamiento de las piezas 3DC11LP y 3DC14EMR-ULP con objeto de cada vez que se ejecuta la aplicación, conocer la ubicación de los blísteres, ya que se prevé que estos puedan ser movidos debido a agentes externos al control del usuario.

Posteriormente, el brazo robótico se posiciona en una ubicación previamente fijada para realizar la detección de las piezas, presentando la coordenada Z del brazo en dicha posición un valor de 492,40 mm. Una vez alcanzada la posición, la red neuronal encargada de realizar la detección de las piezas 3DC11LP y 3DC14EMR-ULP, identifica todas las que es capaz de visualizar la cámara para seguidamente realizar la recogida de la pieza detectada que se encuentra más cerca del centro de la imagen.

Previamente a realizar el cálculo del ángulo necesario para orientar adecuadamente la pieza, existe una situación llamada 'R_PIEZA' en la que, ya estando el brazo robótico con su coordenada Z a 370,00 mm, identifica en el caso de la pieza 3DC11LP la cara de la pieza detectada, es decir, si se trata de su cara superior o inferior. En caso de identificar la cara superior de la pieza 3DC11LP o haber identificado en la situación 'BUSQUEDA' una pieza de la clase 3DC14EMR-ULP, el programa continúa hacia la siguiente situación llamada 'ROTACION'. Sin embargo, si la red neuronal encargada de distinguir entre la cara superior e inferior de la pieza 3DC11LP, identifica que se trata de su cara inferior, el brazo robótico recoge la pieza sin realizar la rotación necesaria y la desecha en un contenedor, saltando el programa tras desechar la pieza a la situación 'INICIAL' con el fin de seguir almacenando piezas.

A continuación, habiendo identificado la red neuronal la pieza 3DC14EMR-ULP o la cara superior de la pieza 3DC11LP, a la hora de realizar la recogida de la pieza en la presente propuesta, el brazo robótico ya ubicado en la posición de la pieza identificada como la más cercana al centro de la imagen, obtiene el ángulo de orientación como se ha desarrollado detalladamente en el punto 4.5.3, recoge la pieza y es entonces cuando corrige el ángulo de orientación.

Llegados a este punto, en el que el brazo robótico posee la pieza correctamente orientada, la deposita en el blíster correspondiente, en la posición adecuada. De modo que la pieza queda almacenada de manera apropiada en su respectivo blíster, en la posición y orientación adecuada.

Cabe mencionar que, aunque las coordenadas del eje Z del brazo robótico presenten valores distintos a los utilizados a la hora de realizar la aplicación del Clasificador de piezas, las alturas desde el plano de trabajo hasta el objetivo de la cámara son las mismas por lo que la relación entre los píxeles de la imagen y los milímetros en la realidad se mantienen en la presente propuesta.

En la siguiente figura aparece el código QR del vídeo donde es posible comprobar la puesta en marcha de la aplicación desarrollada:

Figura 74. *Código QR del funcionamiento de la implementación Clasificador y Rastreador de Piezas.*

El funcionamiento del clasificador y rastreador de piezas fue validado con una cámara de baja resolución, empleando 24 piezas de cada tipo, lo cual rellena 3 filas completas del blíster en el que se almacenan las piezas 3DC14EMR-ULP y 4 filas completas del blíster en el que se almacenan las piezas 3DC11LP, no intentando rellenar ambos blísteres completamente debido a la limitación del rango de movimiento del brazo robótico. La validación se llevó a cabo realizando el mismo procedimiento mencionado, en dos días diferentes, en condiciones de iluminación ambientales, obteniendo en todos los casos una clasificación de las piezas correcta. Con lo cual se concluye que al integrar el sistema rastreador al clasificador de piezas cumple los objetivos requeridos.

6. Presupuesto

En el presente apartado, se expone el presupuesto requerido para llevar a cabo el proyecto *"Implementación de un sistema robótico de Pick and Place a través de visión y computación acelerada"*, en el cual se realiza un desglose del coste de los materiales empleados en la ejecución del proyecto, sin incluir la mano de obra.

Se trata de una primera factura orientativa y realizada para un primer prototipo, incluyendo precios unitarios de los recursos empleados, sin tener en cuenta los posibles descuentos al comprar los materiales al por mayor.

El precio del sistema desarrollado a lo largo de presente documento es de **11110,35 €**. El presupuesto obtenido refleja una circunstancia favorable desde el punto de vista de los precios del mercado actual. Esto es debido a que el coste de una aplicación con robot colaborativo puede oscilar entorno a los 25.000 € y los 80.000 € [57], dependiendo de las funcionalidades y características demandadas por la aplicación. Este hecho da margen de presupuesto para realizar pequeñas mejoras en el sistema propuesto con el fin de hacerlo más competitivo y preciso.

Datos de contacto	
Nombre:	María del Carmen Salas Casas
Titulación:	Ingeniería Electrónica Industrial
Teléfono:	672 42 74 00
Correo:	marisc197@hotmail.com

Cliente	
Nombre:	Escuela de Ingenierías Industriales de Málaga
Correo:	secretaria.eii@uma.es

N.º Presupuesto:	#2023-08-30
Fecha Presupuesto:	30/08/2023
Válido hasta:	30/11/2023
Total:	**11110,35 €**

Proyecto: Implementación de un sistema robótico de Pick and Place a través de visión y computación acelerada.

Materiales:

N.º	Descripción	UDS	P. S/ IVA	IVA	P. C/ IVA	PT. C/ IVA
01	Cámara Web Logitech C210	1	9,88 €	21 %	11,95 €	11,95 €
02	Brazo robótico colaborativo de 6 grados de libertad (pArm6)	1	8330,58 €	21 %	10.080,00 €	10.080,00 €
03	Efector de vacío pArm6 Vacuum Gripper	1	643,80 €	21 %	779,00 €	779,00 €
04	NVIDIA Jetson Nano Developer Kit (4GB)	1	186,69 €	21 %	225,90 €	225,90 €
05	Racor recto – Tubo 6mm	2	0,90 €	21 %	1,10 €	2,20 €
06	Tubo neumático 6x4 mm 10 m	1 m	1,08 €/m	21%	1,31 €/m	1,31 €
07	Soporte de ventosa y ventosa neumática de vacío	1	8,26 €	21 %	9,99 €	9,99 €
					Subtotal material (con IVA) ……	11110,35 €

En Málaga, a 04 de septiembre de 2023.

Fdo.: María del Carmen Salas Casas.

7. Conclusiones y Líneas Futuras

7.1.Conclusiones

El presente proyecto se ha centrado en la creación de redes neuronales diseñadas para la identificación de dos tipos de piezas y la distinción entre la cara superior e inferior de una pieza determinada, integradas adecuadamente con los movimientos de un brazo robótico empleado para la recogida y almacenamiento de las piezas.

A lo largo de la ejecución del proyecto, se ha podido comprobar que las redes neuronales y el brazo robótico logran trabajar en conjunto de manera efectiva cumpliendo con los objetivos previamente planteados, aunque se observa que la velocidad a la que el brazo robótico realiza los movimientos cartesianos limita la eficacia de la red en términos de tiempo, quedando como factor limitante en tiempo los movimientos cartesianos del brazo robótico, siendo conveniente emplear en todo momento los movimientos articulares, aunque esto no es posible en todos los casos calculando los ángulos de las articulaciones del brazo robótico mediante la función *xarm.get_inverse_kinematics()* debido a que, para obtener buenos resultados, por medio de dicha función el punto que desea calcularse debe estar a una distancia inferior a 10 mm del punto actual en el que se encuentra el brazo robótico.

En cuanto a la red neuronal creada y entrenada con el fin de identificar dos tipos de piezas diferentes, es posible apreciar que a la hora de detectar la pieza 3DC14EMR-ULP la red posee una mayor precisión que al detectar la pieza 3DC11LP debido a que se trata de una pieza de color negro, sin reflejos, la cual presenta mayor facilidad de percepción sobre cualquier fondo con un contraste perceptible. Además, se obtuvieron mejores resultados en cuanto a la precisión en la detección de las piezas sobre un fondo blanco que sobre un fondo de color rojo, especialmente con la pieza 3DC11LP, ya que al tratarse de una pieza de color bronce con presencia de reflejos debido a las luces, sobre un fondo blanco logra resaltarse en mayor medida gracias al contraste generado por dichos colores.

Por otro lado, la red encargada de realizar la distinción entre la cara superior e inferior de la pieza 3DC11LP, aunque lograba llevar a cabo su labor exitosamente al emplear una cámara de baja resolución presenta una menor precisión que la anteriormente comentada debido a la similitud entre ambas caras, no fácilmente distinguibles por medio de una cámara de baja resolución.

Otros inconvenientes encontrados a lo largo del transcurso del proyecto debidos a la utilización de una cámara de baja resolución, comprobados que con una cámara de mejores prestaciones no sucedían, son:

- Menor precisión a la hora de localizar el centro exacto de las piezas.
- A la hora de la rotación, a pesar de realizarla adecuadamente, en ciertas ocasiones en las que las condiciones de iluminación no eran favorables, podía haber un pequeño error en términos de grados al rotar la pieza 3DC11LP.

Finalmente se concluye que el sistema diseñado para la detección, clasificación y rastreo de las piezas **cumple con los objetivos planteados**, siendo capaces las redes neuronales de realizar las detecciones pertinentes en un tiempo aproximado de 26 ms, aunque se observa que los

movimientos articulares del brazo robótico son más rápidos que los cartesianos, de modo que si fuera posible emplear la cinemática inversa en todos los movimientos realizados por el brazo se acelerarían considerablemente los resultados obtenidos y en caso de emplear una cámara de mayor resolución, aumentaría la precisión en cuanto a la recogida y orientación de las piezas.

Con la ejecución de este proyecto, además de aumentar la autonomía de los robots colaborativos al dotarlos de visión, se demuestra la viabilidad de automatizar un proceso industrial basado en el almacenamiento de piezas gracias a un brazo robótico colaborativo y una cámara web empleando la visión y computación acelerada. Esto mejora la eficiencia del sistema y los tiempos de detección de las piezas, lo que repercute en un menor tiempo empleado en el almacenamiento de estas.

Adicionalmente, como se ha visto en el apartado de presupuesto, el precio obtenido para el desarrollo del presente proyecto en comparación con el coste de una aplicación con robot colaborativo del mercado actual proporciona cierto margen de presupuesto que permite realizar futuras mejoras en el sistema propuesto con el fin de hacerlo más competitivo y preciso.

7.2. Líneas futuras

Como líneas futuras se contempla la posibilidad de incorporar la cinemática inversa en la totalidad de los movimientos del brazo robótico, permitiendo mejorar considerablemente el tiempo de almacenamiento de las piezas. Al ser el factor limitante en términos de tiempo del presente proyecto la velocidad de los movimientos cartesianos, al implementar dicha mejora los movimientos del brazo se adecuarían al tiempo de detección de las piezas, evitando los cuellos de botella en dicho proceso.

Además, el campo de la visión y de la computación acelerada se encuentra en constante expansión, habiendo hoy en día multitud de tecnologías con mayores prestaciones que la empleada y otras que están aún por llegar, por lo que cabe esperar que los resultados obtenidos en el presente proyecto sean mejorados mediante el uso de una mejor plataforma de hardware para la aceleración computacional y las versiones actualizadas de los paquetes empleados.

También podría ser conveniente explorar la implementación de la red neuronal empleando una arquitectura diferente a la SSD, con el fin de evaluar, observando el equilibrio entre precisión y velocidad, cuál sería la opción más conveniente para alcanzar el objetivo de este proyecto. Adicionalmente, sería conveniente incorporar el proceso de auto calibración de la cámara evitando realizar el cálculo manual entre la relación de los píxeles de la imagen y los milímetros en la realidad para diferentes alturas.

Por otro lado, además de incorporar una cámara de mayor resolución que permita una mayor precisión y la posibilidad de distinguir también entre la cara superior e inferior de la pieza 3DC14EMR-ULP, podría incorporarse un sistema que al detectar que una pieza ha sido identificada por su cara inferior, este le diera la vuelta para poder almacenarla en lugar de depositarla en un contenedor.

Finalmente, podría ser de gran ayuda desarrollar una interfaz de usuario que permita a los operadores interactuar con el sistema de forma intuitiva y eficiente, incorporando esta lo siguiente:

- Un recuento del número de piezas de cada tipo que han sido almacenadas.
- El número total de blísteres que han sido completados para cada tipo de pieza.
- La cantidad de piezas que han sido identificadas por su cara inferior, con el fin de que si el número es elevado realizar mejoras que eviten dicho problema en la línea de producción.
- Un botón de pausa y reanudación por donde iba ejecutándose el proceso.

Incorporando adecuadamente las mejoras mencionadas, una propuesta futura con vistas a optimizar una cadena de producción sería implementar una estrategia logística denominada ***kitting o preparación de kits*** [58]. Dicha estrategia consiste en agrupar los materiales que conforman un producto y almacenarlos en un paquete para posteriormente ensamblar el producto directamente en las líneas de ensamblaje, es decir, el objetivo sería diseñar un sistema capaz de identificar los diferentes materiales que conforman un producto y almacenarlos en la posición adecuada dentro de un paquete de modo que cuando el paquete haya sido completado, este continúe el proceso hasta ser embalado.

Efectuando dicho proceso se optimiza el tiempo de trabajo y se reducen los movimientos ya que, al tener los materiales preparados, los operarios tienen a mano todo lo que necesitan y no tienen que buscar los artículos que precisan. Además, garantiza el abastecimiento ininterrumpido de productos en las líneas de ensamblaje.

8. Referencias

[1] International Federation of Robotics (IFR). Building Resilience for Europe through Automation [Online]. IFR Executive Roundtable. 28 de junio de 2023. Disponible en: https://ifr.org/ifr-press-releases/news/eu-industries-invest-heavily-in-robotics [Consultado: 30/06/2023].

[2] Nexon Automation. Sistemas de visión VRG [Online]. 11 de julio de 2019. Disponible en: https://www.nexonautomation.com/es/sistemas-de-vision-vgr/ [Consultado: 30/06/2023].

[3] Basogain Olabe, X. Redes Neuronales Artificiales y sus aplicaciones. Bilbao: Dpto. Ingeniería de Sistemas y Automática, Escuela Superior de Ingeniería Bilbao; 2008. Disponible en: https://ocw.ehu.eus/pluginfile.php/40137/mod_resource/content/1/redes_neuro/contenidos/pdf/libro-del-curso.pdf [Consultado: 25/07/2023].

[4] Ali A. Introduction of Deep Learning. The Art of Data Scicne [Online]. Medium; 31 de marzo de 2019. Disponible en: Introduction of Deep Learning. In this Chapter, we will discuss Deep… | by Amir Ali | The Art of Data Scicne | Medium [Consultado: 20/03/2023].

[5] Valecun. Evolución de las redes neuronales artificiales [Online]. Timetoast. Disponible en: https://www.timetoast.com/timelines/evolucion-de-las-redes-neuronales-artificiales-c20eb783-ed94-4675-ad4f-dde7e4b8470c[Consultado: 25/07/2023].

[6] Javier. Redes Neuronales y Deep Learning: Avances en Inteligencia Artificial [Online]. Jacar; 18 de marzo de 2023. Disponible en: https://jacar.es/redes-neuronales-y-deep-learning-avances-en-inteligencia-artificial/[Consultado: 25/07/2023].

[7] Barrios J. Redes Neuronales Convolucionales [Online]. Health Big Data; 2022. Disponible en: Redes neuronales convolucionales son un tipo de redes neuronales (juanbarrios.com) [Consultado: 01/08/2023].

[8] Bootcamp AI. Intro a las redes neuronales convolucionales [Online]. Medium; 23 de noviembre de 2019. Disponible en: Intro a las redes neuronales convolucionales | by Bootcamp AI | Medium[Consultado: 01/08/2023].

[9] Microsoft**.** Tutorial 2 [Online]. Microsoft Cognitive Toolkit; 4 de enero de 2023. Disponible en: https://learn.microsoft.com/es-es/cognitive-toolkit/tutorial2/tutorial2 [Consultado: 01/08/2023].

[10] Ali A. Convolutional Neural Network (CNN) with Practical Implementation [Online]. The Art of Data Scicne, Medium; 22 de mayo de 2019. Disponible en:

https://medium.com/machine-learning-researcher/convlutional-neural-network-cnn-2fc4faa7bb63 [Consultado: 01/08/2023]

[11] Rguibi Z, Hajami A, Zitouni D, Elqaraoui A, Bedraoui A. CXAI: Explaining Convolutional Neural Networks for Medical Imaging Diagnostic. Electronics; 2022. Disponible en: https://www.mdpi.com/2079-9292/11/11/1775[Consultado: 02/08/2023]

[12] NVIDIA. ¿Qué es la computación acelerada por GPU? [Online]. Santa Clara, California, Estados Unidos: NVIDIA; 2020. Disponible en: https://www.nvidia.com/es-la/drivers/what-is-gpu-computing/ [Consultado: 03/08/2023]

[13] Merritt R. ¿Qué es la computación acelerada? [Online]. NVIDIA; 19 de noviembre de 2019. Disponible en: https://la.blogs.nvidia.com/2021/11/19/que-es-la-computacion-acelerada/[Consultado: 03/08/2023]

[14] Universidad de Sevilla. Segunda Parte: Tecnología CUDA. Sevilla. Disponible en:https://biblus.us.es/bibing/proyectos/abreproy/11926/fichero/Segunda+parte+TECNOLOGIA+CUDA%252Fi_cuda.pdf[Consultado: 03/08/2023]

[15] Represa Pérez C, Cámara Nebreda JM, Sánchez Ortega PL. Introducción a la programación en CUDA. Área de Tecnología Electrónica, Departamento de Ingeniería Electromecánica, Universidad de Burgos; 2016. Disponible en: https://riubu.ubu.es/bitstream/handle/10259/3933/Programacion_en_CUDA.pdf;jsessionid=7EAC2C3E6B34457ADFFE29AF4AA6CCC4?sequence=1 [Consultado: 03/08/2023]

[16] Basco AI, Beliz G, Coatz D, Garnero P. Industria 4.0: Fabricando el Futuro. Inter-American Development Bank; 2018. Disponible en: file:///C:/Users/Usuario/Downloads/Industria-40-Fabricando-el-Futuro.pdf [Consultado: 04/08/2023]

[17] Blanco R, Fontrodona J, Poveda C. La industria 4.0: El estado de la cuestión. Economía industrial; 2017. p. 151-164. Disponible en: https://www.mincotur.gob.es/Publicaciones/Publicacionesperiodicas/EconomiaIndustrial/RevistaEconomiaIndustrial/406/BLANCO,%20FONTRODONA%20Y%20POVEDA.pdf [Consultado: 04/08/2023]

[18] Faccio M, Granata I, Menini A, Milanese M, Rossato C, Bottin M, Minto R, Pluchino P, Gamberini L, Boschetti G, Rosati G. Human factors in cobot era: a review of modern production systems features. J Intell Manuf; 2023. Vol. 34; p. 85-106. Disponible en: https://doi.org/10.1007/s10845-022-01953-w[Consultado: 04/08/2023]

[19] PREMO Robotics. PARM6 COBOT ARM+CONTROLLER 6DoF 5kg-700mm [Online]. Disponible en: https://premorobotics.com/product/parm-6/ [Consultado: 06/08/2023]

[20] PREMO [Online]. Disponible en: https://www.grupopremo.com/en/.

[21] PREMO Robotics. PARM6 VAC-GRIPPER [Online]. Disponible en: https://premorobotics.com/product/parm-vacuum-gripper/https://premorobotics.com/product/parm-6/ [Consultado: 06/08/2023]

[22] Logitech. Webcam C210. 2010. Disponible en: https://docs.rs-online.com/8a51/0900766b80f0a4b0.pdf [Consultado: 20/03/2023]

[23] Dassault Systèmes. SolidWorks [Software de diseño]. Versión 2022. Waltham, MA: Dassault Systèmes; 2021.

[24] CARGADOR ESP (Amazon). Cargador Corriente 5V 4A 4000mA Clavija/Conector: 5.5mm 2.1mm 20W Salida reemplazo para 5V 0.5A / 1A / 1.5A / 2A / 2.5A / 3A / 3.5A / 4A Cable Alimentacion Adaptador [Online]. Disponible en:https://www.amazon.es/Cargador-Corriente-4000mA-5-5mm-2-1mm/dp/B01M0XSI6L

[25] PREMO. 3DC11LP, SMD 3D Coil Low profile [Online]. PREMO General Catalogue; 2018. Disponible en: https://www.grupopremo.com/upload/cat/3DC11LP%20SMD%203D%20Coil%20Low%20profle.pdf [Consultado: 15/04/2023]

[26] PREMO. 3DC14EMR-ULP, SMD 3D Coil Ultra-Low-Profile [Online]. PREMO General Catalogue; 2021. Disponible en: https://www.grupopremo.com/upload/cat/3DC14EMR-ULP.pdf [Consultado: 15/04/2023]

[27] Arévalo VM, González J, Ambrosio G. La librería de visión artificial OpenCV Aplicación a la docencia e investigación. Dpto. De Ingeniería de Sistemas y Automática, Universidad de Málaga, España; 2004. Disponible en: http://mapir.isa.uma.es/varevalo/drafts/arevalo2004lva1.pdf [Consultado: 06/08/2023]

[28] NVIDIA. JetPack SDK [Online]. NVIDIA Developer. Disponible en: https://developer.nvidia.com/embedded/jetpack [Consultado: 25/03/2023]

[29] Franklin D (NVIDIA Jetson Developer Collecting your own Detection Datasets [Repositorio de software]. GitHub; modificado el 27 de octubre de 2020. Disponible en: https://github.com/dusty-nv/jetson-inference/blob/master/docs/building-repo.md [Consultado: 02/04/2023]

[30] Quiza J. Deep Learning con PyTorch [Online]. Medium; 12 de junio de 2018. Disponible en: https://medium.com/datos-y-ciencia/deep-learning-con-pytorch-672469c637f4 [Consultado: 06/08/2023]

[31] NVIDIA. NVIDIA TensorRT [Online]. NVIDIA Developer. Disponible en: https://developer.nvidia.com/tensorrt [Consultado: 06/08/2023]

[32] Vallim R, Jenks A, Sharkey K, Radich Q, Cowley E. Modelos de ONNX [Online]. Microsoft; 11 de julio de 2023. Disponible en: https://learn.microsoft.com/es-es/windows/ai/windows-ml/get-onnx-model [Consultado: 06/08/2023]

[33] Sánchez Alberca A. La librería Numpy [Online]. Aprende con Alf; 12 de mayo de 2022.
Disponible en: https://aprendeconalf.es/docencia/python/manual/numpy/ [Consultado: 07/08/2023]

[34] Sánchez Alberca A. La librería Matplotlib [Online]. Aprende con Alf; 4 de octubre de 2020.
Disponible en: https://aprendeconalf.es/docencia/python/manual/matplotlib/ [Consultado: 07/08/2023]

[35] Python Software Fundation. math-Funciones matemáticas [Online]. Disponible en: https://docs.python.org/es/3/library/math.html [Consultado: 07/08/2023]

[36] Pyhton Software Fundation. time- Acceso a tiempo y conversiones [Online]. Disponible en: https://docs.python.org/es/3/library/time.html [Consultado: 07/08/2023]

[37] Franklin D (NVIDIA Jetson Developer). Jetson-inference. Deploying Deep Learning [Repositorio de software]. GitHub; 28 de septiembre de 2020. Disponible en: https://github.com/dusty-nv/jetson-inference [Consultado: 30/03/2023]

[38] Franklin D (NVIDIA Jetson Developer). Jetson-utils [Repositorio de software]. GitHub; 1 de febrero de 2018. Disponible en: https://github.com/dusty-nv/jetson-utils [Consultado: 08/08/2023]

[39] Vargas Carvajal M. Aplicación de Técnicas de Aprendizaje Profundo para la Detección de Defectos en la Producción de Componentes Electromagnéticos [Trabajo Final de Grado]. Universidad de Málaga; 5 de septiembre de 2022.

[40] NVIDIA. Kits de desarrollo y módulos para sistemas integrados [Online]. Disponible en: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/ [Consultado: 20/03/2023]

[41] NVIDIA. NVIDIA Jetson Comparison [Online]. Disponible en: https://www.seeedstudio.com/blog/nvidia-jetson-comparison-nano-tx2-nx-xavier-nx-agx-orin/ [Consultado: 18/03/2023]

[42] Serzhenko F. Benchmark comparison for Jetson Nano, TX2, Xavier NX and AGX [Online]. Linkedin; 6 de abril de 2021. Disponible en:

https://www.linkedin.com/pulse/benchmark-comparison-jetson-nano-tx2-xavier-nx-agx-fyodor-serzhenko [Consultado: 18/03/2023]

[43] NVIDIA. Jetson Nano Developer Kit User Guide [Online]. 15 de enero de 2020. Disponible en: https://developer.nvidia.com/embedded/downloads#?search=Jetson%20Nano%20Developer%20Kit%20User%20Guide [Consultado: 21/03/2023]

[44] Balena. Downland Etcher [Online]. Disponible en: https://etcher.balena.io/#download-etcher [Consultado: 22/03/2023]

[45] NVIDIA. JetPack SDK 4.6 Release Page [Online]. NVIDIA Developer. Disponible en: https://developer.nvidia.com/embedded/jetpack-sdk-46 [Consultado: 23/03/2023]

[46] NVIDIA Developer. Jetson AI Fundamentals - S1E1 - First Time Setup with JetPack [Youtube]. 13 de octubre de 2020. Disponible en: https://www.youtube.com/watch?v=uvU8AXY1170 [Consultado: 27/03/2023]

[47] Pyrestone. Jetson-fan-ctl [Repositorio de software]. GitHub. 8 de agosto de 2021. Disponible en: https://github.com/Pyrestone/jetson-fan-ctl [Consultado: 28/03/2023]

[48] Microsoft. Visual Studio Code [Online]. Redmond, Washington: Microsoft. Disponible en: https://code.visualstudio.com/

[49] Franklin D (NVIDIA Jetson Developer). Building the Project from Source [Repositorio de software]. GitHub; modificado el 25 de marzo de 2023. Disponible en: https://github.com/dusty-nv/jetson-inference/blob/master/docs/building-repo.md [Consultado: 07/04/2023]

[50] Araujo Santos L. SSD Introduction [Online]. Artificial Inteligence; 2020. Disponible en: https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/single-shot-detectors/ssd

[51] Franklin D (NVIDIA Jetson Developer). Re-training SSD-Mobilenet [Repositorio de software]. GitHub; modificado el 21 de mayo de 2021. Disponible en: https://github.com/dusty-nv/jetson-inference/commits/e4ebc40967604945fd501b8d35ed0b9e86ca8b2d/docs/pytorch-ssd.md [Consultado: 09/04/2023]

[52] Bareeva J. How to Run Inference Using TensorRT C++ API [Online]. Learn OpenCV; 24 de agosto de 2020. Disponible en: https://learnopencv.com/how-to-run-inference-using-tensorrt-c-api/

[53] UFactory. xArm Collaborative Robot. [Online]. Disponible en: https://www.ufactory.cc/xarm-collaborative-

robot?utm_source=google+search&utm_medium=cpc&utm_campaign=additional+link&utm_id=Branding&gclid=CjwKCAjww7KmBhAyEiwA5-PUSnwqgFpXs9hz17GsEm6jJ0AHa5xZXaOkto7TudgNJle364h8bWGYWhoC__YQAvD_BwE [Consultado: 20/04/2023]

[54] GIMP. GIMP 2.10.12 - The GNU Image Manipulation Program [Online]. 12 de junio de 2019. Disponible en: https://www.gimp.org/news/index4.html

[55] Progreva. Binarización inversa de una imagen – Python [Online]. Blogger; 14 de febrero de 2021. Disponibles en: http://programeva.blogspot.com/2021/02/binarizaion-inversa-de-una-imagen-python.html [Consultado: 03/05/2023]

[56] Wang D. Why I can not get the right joint agnles of the position by "get_inverse_kinematics"? [Online]. UFactory; 24 de marzo de 2020. Disponible en: https://forum.ufactory.cc/t/why-i-can-not-get-the-right-joint-agnles-of-the-position-by-get-inverse-kinematics/2492 [Consultado: 10/08/2023]

[57] Pelegrí J. ¿Cuánto cuesta un robot colaborativo? ¡Veámoslo! [Online]. Universal Robots; 9 de febrero de 2022. Disponible en: https://www.universal-robots.com/es/blog/coste-robot-colaborativo/ [Consultado: 30/08/2023]

[58] Mecalux. Kitting: estrategia logística para optimizar el ensamblaje en el almacén. 12 de mayo de 2020. Disponible en: https://www.mecalux.es/blog/kitting

[59] Santiago Sánchez. EJECUTA TUS PRIMERAS REDES NEURONALES EN TIEMPO REAL | Jetson Nano | Instalacion Jetson inference [Online]. Aprende e Ingenia, Youtube; 4 de marzo de 2022. Disponible en: https://www.youtube.com/watch?v=akMlbh0W9cY&list=PLnKxR99sdlEg3JYlvjTy-p77UjKiWCj0V&index=3 [Consultado: 24/03/2023]

[60] Sergio Canu. Install Opencv 4.1 on Nvidia Jetson Nano [Online]. Pysource; 26 de agosto de 2019. Disponible en: https://pysource.com/2019/08/26/install-opencv-4-1-on-nvidia-jetson-nano/ [Consultado: 24/03/2023]

[61] xArm-Developer. xArm-Python-SDK [Repositorio de software]. GitHub. 2019. Disponible en: https://github.com/xArm-Developer/xArm-Python-SDK [Consultado: 28/03/2023]

[62] Aytar Y, Eslami A, Sorokin A. The PASCAL Visual Object Classes Homepage [Online]. Disponible en: http://host.robots.ox.ac.uk/pascal/VOC/ [Consultado: 07/04/2023]

[63] Dustin Franklin (NVIDIA Jetson Developer). SyntaxError: future feature annotations is not defined #1534 [Online]. GitHub. 14 de diciembre de 2022. Disponible en: https://github.com/dusty-nv/jetson-inference/issues/1534 [Consultado: 07/04/2023]

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Anexos

Anexo I. Códigos MATLAB®

En el presente Anexo se recogen los comandos necesarios para realizar la cargar de los datos requeridos para hacer la comparativa del rendimiento en la detección de anomalías con y sin el uso de la GPU, almacenar los tiempos, y realizar las gráficas e histogramas con los tiempos obtenidos.

En primer lugar, se cargan los datos de la cara BOTTOM en este caso:

```
%CARGAR DATOS CARA BOTTOM 1.

variables_pad_1=load('pad_1');
variables_pin_1=load('pin_1');
variables_winding_1=load('winding_1');
mask_winding_1=load('mask_winding_1');
ref_1=load('vbles_ref_1');
ROIs_1=load('ROIs_1');
yolo_1=load('yolo_pad_1');
```

Es necesario tener en cuenta que a la hora de cargar los datos y realizar las pruebas sin la GPU, debe ejecutarse el siguiente comando:

```
%Desconectar GPU:
setenv CUDA_VISIBLE_DEVICES -1
```

A continuación, se almacenan los tiempos que tarda en detectarse las anomalías de las imágenes de la pieza 3DC11LP:

```
%CARA BOTTOM
imdsBOTTOM=imageDatastore("C:\Users\Usuario\OneDrive\Escritorio\SEXTO\TFG
ELECTRÓNICA\Linea2\1\ORIGINALES"); % ruta de la carpeta de imágenes BOTTOM

n=500;
times = zeros(n,1);
for i = 1:n
    tic
    imgIn = read(imdsBOTTOM);
    figure;
    imshow(imgIn)
    title('Imagen de entrada','fontname','Times New Roman','fontsize',16);
    [imgOut,flag] =
Bot1_Processing_YOLO(imgIn,variables_pad_1,variables_pin_1,variables_winding_
1,mask_winding_1,ref_1,1,ROIs_1,yolo_1);
    figure;
    imshow(imgOut)
    title('Imagen de salida con las anomalías detectadas','fontname','Times
New Roman','fontsize',16);
    times(i)=toc;
end
```

Una vez almacenados los tiempos se procede a realizar la gráfica correspondiente comparando los tiempos obtenidos en la detección de las anomalías con y sin la GPU por medio de los siguientes comandos:

```
% GRÁFICA VISTA BOTTOM

figure;set(gcf,'PaperPositionMode','manual','PaperUnits','centimeters',
'PaperType', 'A4','PaperOrientation','landscape','PaperPosition',[0.63 0.63
28.41 19.72]);
lw = 1; % Line width
ms = 2; % Marker size
hold on
plot(timesBOTGPU,'-o', 'LineWidth', lw, 'MarkerSize', ms);
hold on
plot(timesBOTSINGPU,'-o', 'LineWidth', lw, 'MarkerSize', ms);

xlabel('Pieza (uds)','fontname','Times New Roman');
ylabel('Tiempo (s)','fontname','Times New Roman');
title('Análisis comparativo del rendimiento en la detección de anomalías de
la cara BOTTOM en piezas electromagnéticas,',' con y sin el uso de una unidad
de procesamiento gráfico (GPU)','fontname','Times New Roman');
legend('Con GPU','Sin GPU','fontname','Times New Roman');
set(gca,'fontsize',12); % Axes font size
grid on
hold off
```

Finalmente, con objeto de realizar una comparación visualmente más llamativa, se realiza la comparación por medio de un histograma:

```
%%HISTOGRAMA VISTA BOTTOM
figure;set(gcf,'PaperPositionMode','manual','PaperUnits','centimeters',
'PaperType', 'A4','PaperOrientation','landscape','PaperPosition',[0.63 0.63
28.41 19.72]);
lw = 1; % Line width
ms = 2; % Marker size
hold on
histogram(timesBOTGPU(2:end),40) %AQUÍ INDICAMOS QUE NO COGEMOS EL PRIMER
VALOR

xlabel('Tiempo (s)','fontname','Times New Roman');
ylabel('Piezas (uds)','fontname','Times New Roman');
title('Histograma vista BOTTOM con GPU','fontname','Times New Roman');
legend('Con GPU','fontname','Times New Roman');
set(gca,'fontsize',12); % Axes font size
grid on
hold off

%Sin GPU: #D95319

% HISTOGRAMA VISTA BOTTOM
figure;set(gcf,'PaperPositionMode','manual','PaperUnits','centimeters',
'PaperType', 'A4','PaperOrientation','landscape','PaperPosition',[0.63 0.63
28.41 19.72]);
lw = 1; % Line width
ms = 2; % Marker size
```

```
hold on
histogram(timesBOTSINGPU(2:end),40, 'FaceColor','#D95319') %AQUÍ INDICAMOS
QUE NO COGEMOS EL PRIMER VALOR

xlabel('Tiempo (s)','fontname','Times New Roman');
ylabel('Piezas (uds)','fontname','Times New Roman');
title('Histograma vista BOTTOM sin GPU','fontname','Times New Roman');
legend('Sin GPU','fontname','Times New Roman');
set(gca,'fontsize',12); % Axes font size
grid on
hold off

%Sin GPU: #D95319

% HISTOGRAMA VISTA BOT EN CONJUNTO:
figure;set(gcf,'PaperPositionMode','manual','PaperUnits','centimeters',
'PaperType', 'A4','PaperOrientation','landscape','PaperPosition',[0.63 0.63
28.41 19.72]);
lw = 1; % Line width
ms = 2; % Marker size
hold on
histogram(timesBOTGPU(2:end),40) %aQUÍ INDICAMOS QUE NO COGEMOS EL PRIMER
VALOR
hold on
histogram(timesBOTSINGPU(2:end),40, 'FaceColor','#D95319') %aQUÍ INDICAMOS
QUE NO COGEMOS EL PRIMER VALOR

xlabel('Tiempo (s)','fontname','Times New Roman');
ylabel('Piezas (uds)','fontname','Times New Roman');
title('Histograma comparativo de la detección de anomalías en la cara BOTTOM
de piezas electromagnéticas,','con y sin el uso de una unidad de
procesamiento gráfico (GPU)','fontname','Times New Roman');
legend('Con GPU','Sin GPU','fontname','Times New Roman');
set(gca,'fontsize',12); % Axes font size
grid on
hold off
```

Anexo II. Configuración y preparación del entorno de trabajo

A. Desinstalación de LibreOffice y actualización del sistema

En el presente apartado del Anexo II se recogen los comandos necesarios para la desinstalación de LibreOffice:

```
$ sudo apt-get remove --purge libreoffice*
$ sudo apt clean
$ sudo apt-get autoremove
```

Además, también se incluyen los comandos necesarios para actualizar el sistema, proceso que puede demorarse varios minutos:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

B. Aumento de la memoria Swap

A continuación, se recogen los comandos empleados en el aumento de la memoria Swap de la Jetson Nano:

```
$ free -m  (Para observar el espacio de memoria)
```

Tras analizar el espacio de memoria para posteriormente corroborar su aumento, se procede a ampliar el tamaño a 4 GB escribiendo los siguientes comandos en la terminal:

```
$ sudo systemctl disable nvzramconfig (Deshabilitamos zram)
$ sudo fallocate -l 4G /mnt/4GB.swap
$ sudo chmod 600 /mnt/4GB.swap
$ sudo mkswap /mnt/4GB.swap
$ sudo vi /etc/fstab
```

Una vez ingresados los comandos anteriores, aparece una ventana en la que se debe agregar, tal y como aparece en la ***Figura 75***, al final de toda la información la siguiente línea adicional:

```
/mnt/4GB.swap swap swap defaults 0 0
```

Finalmente escriba ":wq" y presione "enter" para cerrar el archivo.

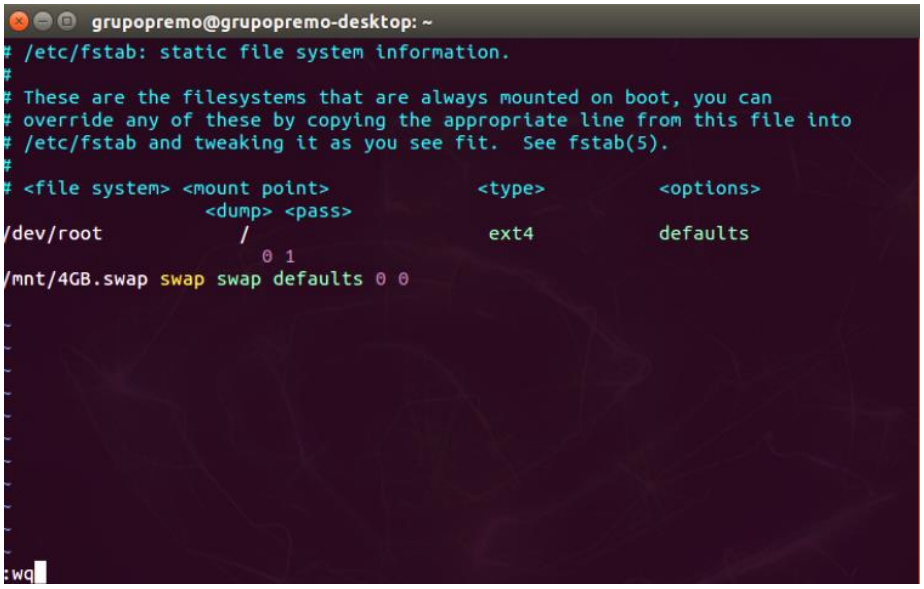

Figura 75. *Archivo fstab a editar para aumentar la memoria Swap.*

C. Activación del ventilador

Siguiendo los siguientes pasos se pone en marcha el ventilador instalado en la Jetson en función de la temperatura de la placa.

En primer lugar, Python 3 debe estar preinstalado en la Jetson Nano, en caso contrario es posible instalar las dependencias mínimas necesarias con el siguiente comando:

```
$ sudo apt install python3-dev
```

A continuación, se accede a la carpeta de descargas y se clona el repositorio [47] en dicha carpeta:

```
$ cd Descargas
$ git clone https://github.com/Pyrestone/jetson-fan-ctl.git
```

Una vez clonado el repositorio, se accede a la carpeta y se ejecuta la aplicación que pone en funcionamiento el ventilador:

```
$ cd jetson-fan-ctl
$ ./install.sh
```

D. Configuración del entorno para Jetson Stats e instalación

Como se ha comentado en el apartado *4.2.7. Configuración del entorno para Jetson Stats*, el propósito del apartado es realizar la creación de un entorno para *Jetson Stats* y su correspondiente instalación con el apoyo del vídeo [59]. Para ello en primer lugar, se descargan un conjunto de herramientas útiles con el fin de crear el entorno de soporte:

```
$ sudo apt-get update
$ sudo apt-get install git cmake
$ sudo apt-get install python3-dev
$ sudo apt-get install libatlas-base-dev gfortran
$ sudo apt-get install libhdf5-serial-dev hdf5-tools
```

A continuación, se instala la herramienta "*pip"* para Python 3:

```
$ sudo apt-get install python3-pip
```

En este punto, la instalación de *Jetson Stats* ya puede ser llevada a cabo por medio del siguiente comando:

```
$ sudo -H pip3 install -U jetson-stats
```

Finalmente, se reinicia la plataforma de desarrollo y se ejecuta el siguiente comando:

```
$ jtop
```

E. Instalación OpenCV con CUDA

En el presente apartado del Anexo II, con el apoyo del repositorio [60], se detallan los pasos a seguir en la instalación de OpenCV 4.1 con soporte CUDA en Jetson Nano. Este se trata de un proceso largo, el cual dura alrededor de dos horas.

En primer lugar, se actualizan e instalan los paquetes necesarios para compilar OpenCV desde el código fuente:

```
$ sudo apt update
$ sudo apt install -y build-essential cmake git libgtk2.0-dev pkg-config
libswscale-dev libtbb2 libtbb-dev
$ sudo apt install -y python-dev python3-dev python-numpy python3-numpy
$ sudo apt install -y curl
```

Posteriormente, se procede a la instalación de códecs de video e imagen, que son un conjunto de herramientas empleadas para comprimir y descomprimir datos de videos e imágenes:

```
$ sudo apt install -y libjpeg-dev libpng-dev libtiff-dev libjasper-dev
$ sudo apt install -y libavcodec-dev libavformat-dev
$ sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
$ sudo apt install -y libv4l-dev v4l-utils qv4l2 v4l2ucp libdc1394-22-dev
```

A continuación, por medio de los dos siguientes comandos se descarga la versión 4.1 de OpenCV y la biblioteca abierta de contribuciones de OpenCV, la cual contiene bibliotecas y dependencias adicionales:

```
$ curl -L https://github.com/opencv/opencv/archive/4.1.0.zip -o opencv-
4.1.0.zip
$ curl -L https://github.com/opencv/opencv_contrib/archive/4.1.0.zip -o
opencv_contrib-4.1.0.zip
```

Una completada la descarga, se descomprimen los paquetes y se accede a la carpeta de OpenCV 4.1:

```
$ unzip opencv-4.1.0.zip
$ unzip opencv_contrib-4.1.0.zip
$ cd opencv-4.1.0/
```

Antes de comenzar con la creación real de la biblioteca es necesario realizar un paso previo. Se debe crear un directorio donde sea posible ubicar los archivos de compilación:

```
$ mkdir release
$ cd release/
```

Una vez se accede al directorio creado llamado "*release*" se procede a la creación de OpenCV usando CMake, la cual es una herramienta de código abierto empleada para facilitar el proceso de construcción y configuración de proyectos de software. Además, es necesario agregar comentarios necesarios para la correcta creación de OpenCV, como: la habilitación del soporte CUDA, del soporte GStreamer y de la biblioteca libv4l, el establecimiento de la ruta de las contribuciones adicionales, la compilación y creación de la biblioteca OpenCV con soporte tanto para Python 2 como para Python 3, la desactivación de la compilación de pruebas unitarias, de rendimiento y de ejemplos y finalmente el establecimiento del tipo de compilación como "*RELEASE*" y del directorio de instalación de OpenCV:

```
$ cmake -D WITH_CUDA=ON \
-D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib-4.1.0/modules \
-D WITH_GSTREAMER=ON \
-D WITH_LIBV4L=ON \
-D BUILD_opencv_python2=ON \
-D BUILD_opencv_python3=ON \
-D BUILD_TESTS=OFF \
-D BUILD_PERF_TESTS=OFF \
```

```
-D BUILD_EXAMPLES=OFF \
-D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local ..
```

En último lugar, se compila OpenCV y tras una compilación exitosa se instalan los paquetes recién generados en base a los datos del sistema:

```
$ make -j4
$ sudo make install
```

F. Instalación de librerías necesarias

En el presente apartado se recogen los comandos necesarios para realizar la instalación de las librerías requeridas para la ejecución del proyecto y que hasta el momento no se han instalado:

```
$ sudo pip3 install python-time
$ sudo pip3 install DateTime
$ sudo pip3 install traceback2
$ sudo pip3 install numpy
$ sudo pip3 install pandas
$ sudo pip3 install opencv-python==4.6.0.66
```

Finalmente, como se ha comentado en el apartado 3.1. Hardware, es necesario instalar el SDK del robot. Para llevar a cabo dicha instalación es necesario clonar el repositorio [61], acceder a la carpeta clonada y ejecutar el archivo de instalación con los siguientes comandos:

```
$ git clone https://github.com/xArm-Developer/xArm-Python-SDK.git
$ cd xArm-Python-SDK/
$ sudo python3 setup.py install
```

Además, en el presente apartado se realiza la conexión de la cámara a la Jetson Nano por medio de uno de los puertos USB y se verifica su conexión:

```
$ sudo apt install v4l-utils
$ v4l2-ctl --list-devices
```

En la siguiente figura se verifica la conexión de la cámara empleada "*/dev/video0*":

```
grupopremo@grupopremo-desktop:~$ v4l2-ctl --list-devices
Integrated Camera (usb-70090000.xusb-2.3):
        /dev/video0
```

Figura 76. *Verificación de la conexión de la cámara.*

Anexo III. Creación de redes neuronales

En el presente Anexo se describen los pasos a seguir para realizar la instalación del entorno de desarrollo de *Jetson Inference* en Jetson Nano, así como la creación de redes neuronales personalizadas.

A. Instalación del entorno de desarrollo de Jetson Inference

Con el apoyo del repositorio [49] se instala el entorno de desarrollo de *Jetson Inference* en su Jetson Nano y se ejecutan las primeras redes neuronales preentrenadas siguiendo los siguientes pasos. En primer lugar, es necesario actualizar nuevamente el sistema:

```
$ sudo apt-get update
```

Y asegurarse de que Git y CMake estén instalados correctamente y si no es así, pueden instalarse mediante el siguiente comando:

```
$ sudo apt-get install git cmake
```

A continuación, se procede a crear una carpeta con el nombre deseado a fin de tener el repositorio organizado y fácilmente localizable. En el presente caso, la carpeta ha sido creada con el nombre *"detección-objetos1"* y una vez creada, se accede a ella:

```
$ mkdir deteccion-objetos1
$ cd deteccion-objetos1
```

Una vez dentro de la carpeta, se clona el repositorio *jetson-inference* y se accede a la carpeta de *jetson-inference* descargada:

```
$ git clone https://github.com/dusty-nv/jetson-inference
$ cd jetson-inference
```

Posteriormente, se ejecuta el submódulo "*init*" para actualizar y descargar submódulos necesarios del repositorio Git empleado como, por ejemplo, *Jetson Utils*:

```
$ git submodule update --init
```

La operatividad de Python se implementa por medio de módulos de extensión de Python que enlazan el código nativo de C++ empleando la interfaz de programación de aplicaciones (API) de Python C. De manera predeterminada, el sistema de Ubuntu posee los paquetes "*libpython-dev"* y "*python-numpy*" para la versión de Python 2.7. Por medio del siguiente comando, se instalan para Python 3.6 (intérprete de Python preinstalado en Ubuntu) los archivos de desarrollo para la compilación de extensiones de Python y la biblioteca Numpy, la cual permite trabajar en Python con matrices y operaciones matemáticas complejas:

```
$ sudo apt-get install libpython3-dev python3-numpy
```

A continuación, se crea un directorio de compilación dentro del propio proyecto y se ejecuta mediante CMake para así configurar la compilación. En el instante en el que CMake sea ejecutado, se inicia un script llamado "*CMakePreBuild.sh*", el cual se encarga de instalar las dependencias necesarias y descargar los modelos de redes neuronales profundas (DNN dada su traducción en inglés, Deep Neural Network).

```
$ mkdir build
$ cd build
$ cmake ../
```

Una vez terminado el proceso de ejecución de CMake, es necesario descargar los modelos preentrenados que se deseen. El proyecto incluye diferentes redes preentrenadas que pueden ser descargadas e instaladas por el usuario mediante la herramienta de descarga de modelos "*download-models.sh*". Por defecto no todos los modelos se seleccionan inicialmente con el fin de ahorrar espacio en el disco.

Al configurar el proyecto inicialmente, CMake ejecuta automáticamente la herramienta de descarga de modelos, lo que permite seleccionar los modelos. Aparece entonces la siguiente ventana, donde deben seleccionarse los modelos deseados:

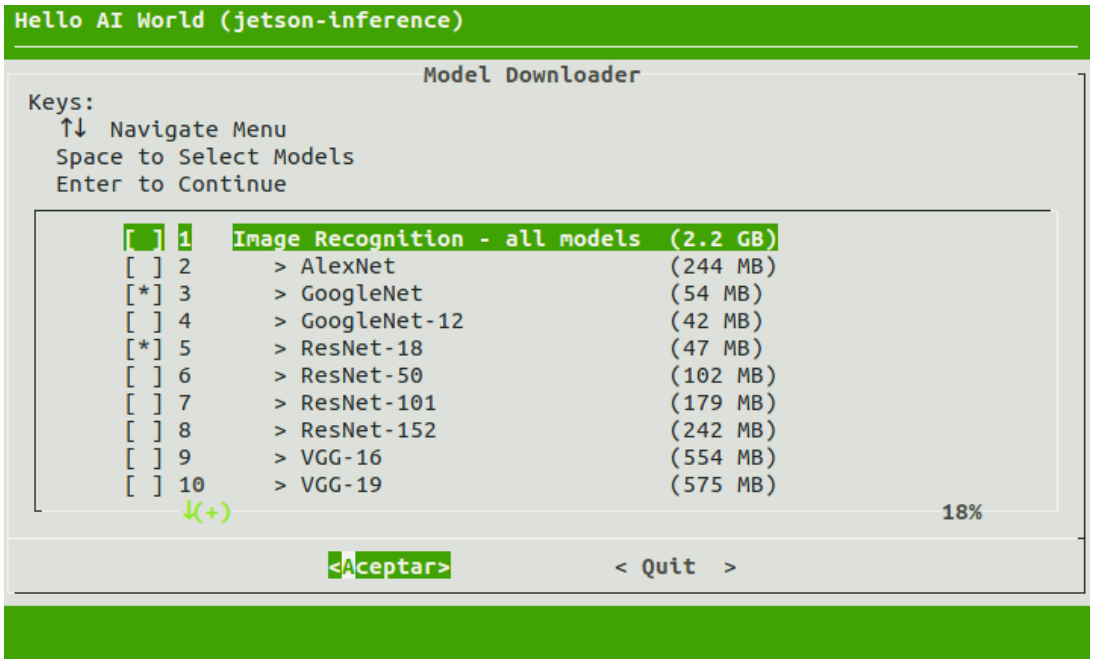

Figura 77. *Selección de modelos de redes preentrenadas para su descarga.*

Una vez señalados los modelos, se selecciona la opción "Aceptar" y se presiona la tecla "*Enter*" para continuar.

En caso de necesitar ejecutar nuevamente la herramienta de descarga de modelos es necesario ejecutar los siguientes comandos:

```
$ cd jetson-inference/tools
$ ./download-models.sh
```

Una vez descargados todos los modelos, aparece una ventana para realizar la instalación de PyTorch 1.6.0 para Python 3.6. Si desea instalarlo es necesario seleccionar la opción "Aceptar" y presionar la tecla "*Enter*" para continuar.

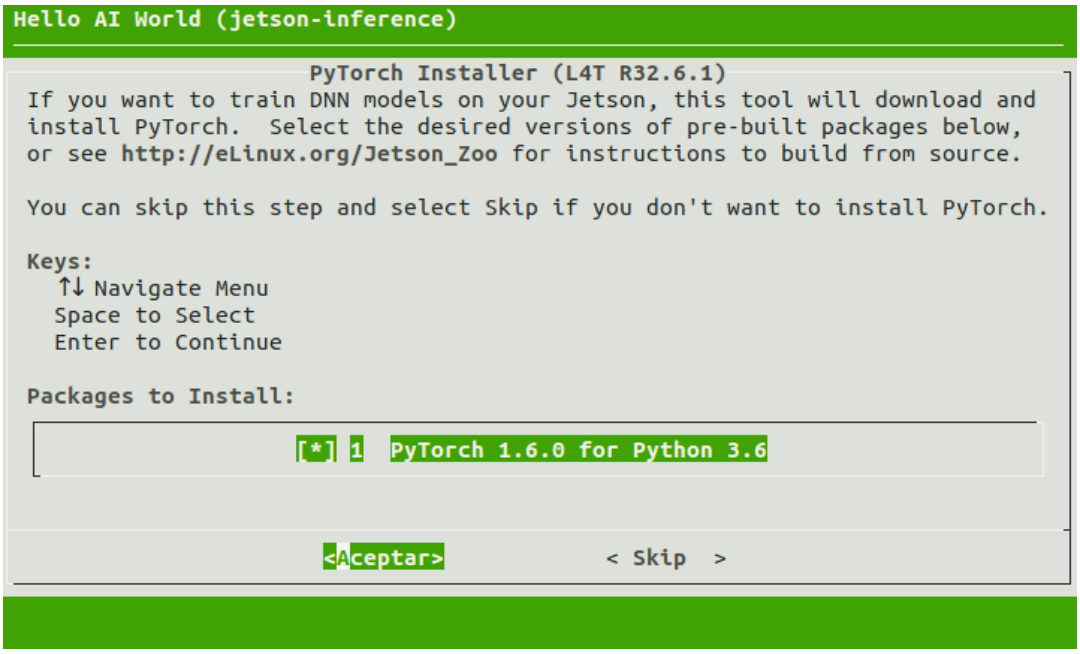

Figura 78. *Ventana de instalación de PyTorch 1.6.0 para Python 3.6.*

Nos aseguramos de que nos encontramos en el directorio "*jetson-inference/build*" creado previamente y se ejecuta "*make"* seguido de *sudo make install* con el objetivo de compilar los enlaces de extensión de Python, las bibliotecas y los ejemplos de código:

```
$ make -j8
$ sudo make install
$ sudo ldconfig
```

Tras la ejecución del comando "*sudo make install*" los módulos "*jetson_inference*" y "*jetson_utils*" deben de poder ser importados en Python.

Llegados a este punto, es posible compilar los modelos anteriormente descargados a partir de la carpeta "*bin*" situada en "*build/aarch64*". Para acceder a dicha carpeta desde la carpeta "*build*" es necesario ejecutar el siguiente comando:

```
$ cd aarch64/bin/
```

Una vez dentro de la carpeta deseada es posible visualizar su contenido por medio del siguiente comando:

```
$ ls
```

En estas circunstancias, es posible ejecutar las diversas redes neuronales preentrenadas previamente descargadas.

B. Red neuronal personalizada para la clasificación de piezas

Con el apoyo del repositorio [29] se procede a la recopilación del conjunto de datos de detección de las piezas 3DC11LP y 3DC14EMR-ULP fabricadas por la empresa PREMO con el fin de entrenar un modelo que realice dicha labor. Por tanto, esta sección se divide en dos partes bien diferenciadas, una donde se muestran los pasos a seguir para la recopilación de los datos de detección y otra para realizar el entrenamiento del modelo.

- **Recopilación de datos de detección.**

En primer lugar, se accede a la carpeta "*ssd*" dentro de las carpetas "*training/detection*" del entorno de *Jetson Inference* mediante el siguiente comando, el cual nos indica el directorio en el que se sitúa la carpeta:

```
$ cd deteccion-objetos1/jetson-inference/python/training/detection/ssd/
```

Posteriormente, se comprueba si dicha carpeta se encuentra o no vacía. En caso de no contener ningún documento, es necesario clonar el repositorio *SSD-based Object Detection in PyTorch*:

```
$ git clone https://github.com/dusty-nv/pytorch-ssd.git
```

A continuación, se accede a la carpeta clonada:

```
$ cd pytorch-ssd
```

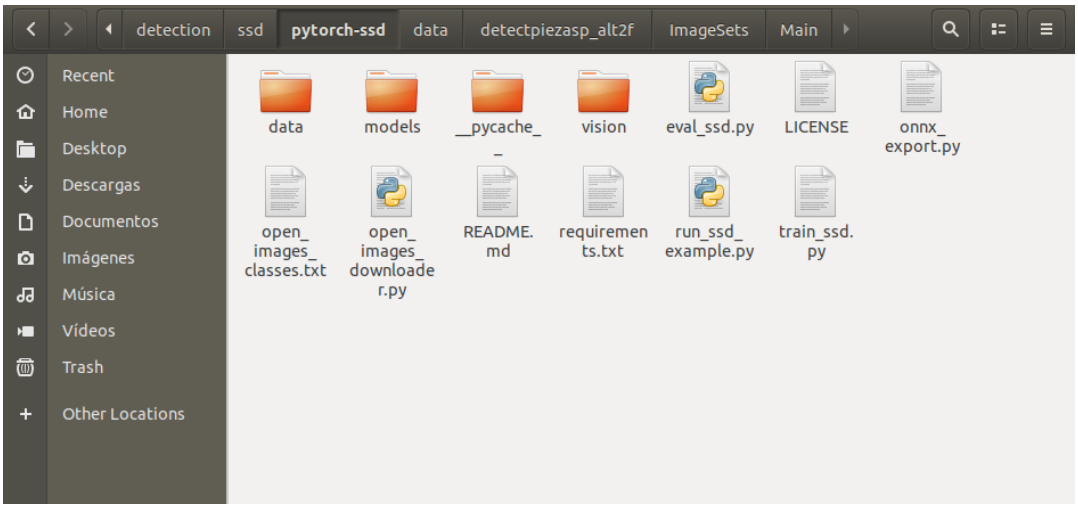

Figura 79. *Carpeta pytorch-ssd clonada.*

Se instalan los requerimientos necesarios para la ejecución de la red neuronal y "*boto3*", requisito que no se instala por defecto:

```
$ pip3 install -v -r requirements.txt
$ sudo pip3 install boto3
```

Probablemente se aprecie un mensaje de error a la hora de ejecutar el comando para la instalación de "*boto3*". Dicho mensaje indica al usuario que no posee los permisos adecuados para la escritura de los directorios de caché utilizados por *pip*. Esto puede ocurrir cuando se utiliza sudo para instalar paquetes de Python en un entorno virtual de Python que se creó con un usuario diferente.

El mensaje sugiere que se verifiquen los permisos y el propietario de los directorios *'/home/grupopremo/.cache/pip/http'* y *'/home/grupopremo/.cache/pip'* y debe asegurarse de que pertenecen al usuario actual.

Una posible solución al problema expuesto puede ser ejecutar el comando sudo con la opción -H. Esto establecerá la variable de entorno HOME en el directorio de inicio del usuario actual, lo cual puede servir de ayuda para garantizar que los archivos de caché se guarden en los directorios correctos.

Por tanto, se emplea el siguiente comando para la instalación de "*boto3"*:

```
$ sudo -H pip3 install boto3
```

Una vez ejecutado dicho comando se realiza la instalación de "*boto3"* correctamente.

Llegados a este punto, se encuentra en disposición de recopilar un conjunto de datos de detección propio. Para ello desde la carpeta "*data*" situada dentro de la carpeta "*pytorch-ssd*" se crea un nuevo directorio donde se recopilarán los datos del conjunto de detección que se desea crear:

```
$ cd deteccion-objetos1/jetson-
inference/python/training/detection/ssd/pytorch-ssd/data
$ mkdir detectpiezasp_alt2f
```

Dentro de la carpeta creada (detectpiezasp_alt2f), se crea un archivo de texto con el nombre "*labels.txt*" donde debe escribirse el nombre de los diferentes objetos o piezas que se desean detectar. En el presente caso lo que se pretende es distinguir entre dos tipos de piezas fabricadas por la empresa PREMO, por lo que se escribe el nombre de dichas piezas en el archivo de texto mencionado:

- 3DC11LP.
- 3DC14EMR-ULP.

Tras la creación del directorio para el almacenamiento de la información del *dataset* y del archivo de texto con las piezas a detectar, se emplea la herramienta "control de captura de datos" desde la carpeta "*pytorch-ssd*":

```
$ cd deteccion-objetos1/jetson-
inference/python/training/detection/ssd/pytorch-ssd
$ camera-capture /dev/video0
```

Tras el último comando ejecutado, aparece la ventana que se muestra a continuación para el control de captura de datos:

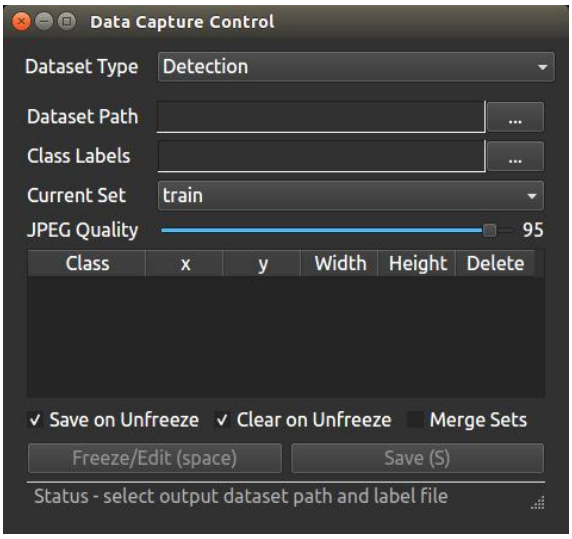

Figura 80. *Control de captura de datos inicial.*

En ella debe completarse la siguiente información antes de proceder a la recopilación de imágenes:

- **Tipo de datos** (Dataset Type). Cuando el menú desplegable del tipo de datos se encuentra en modo detección, como es el caso, la herramienta crea conjuntos de datos en formato Pascal VOC (*Visual Object Classes*) [62]. Pascal VOC consiste en un conjunto de datos para evaluar y crear algoritmos de detección de objetos, clasificación de imágenes y segmentación semántica.
- **Ruta del conjunto de datos** (Dataset Path). Indicar la ruta en la que se desea almacenar la información del conjunto de datos. En este caso: ***deteccion-objetos1/jetson-inference/python/training/detection/ssd/pytorch-ssd/data/detectpiezasp_alt2f.***
- **Etiquetas de clases** (Class Labels). Indicar la ruta en la que se encuentra el archivo de texto con las diferentes clases a detectar. En este caso: ***deteccion-objetos1/jetson-inference/python/training/detection/ssd/pytorch-ssd/data/detectpiezasp_alt2f/labels.txt.***
- **Conjunto actual** (Current Set). Seleccionar entre los conjuntos "*train*", "*val*" y "*test*" aunque si se selecciona la opción Marge sets, como se menciona a continuación, los datos de replican en los tres conjuntos.
- **Calidad JPEG** (JPEG Quality). Control de calidad de la codificación y el tamaño del disco de las imágenes guardadas.
- **Guardar al descongelar** (Save on Unfreeze). Su selección almacena automáticamente los datos al descongelar la imagen.

- **Borrar al descongelar** (Clear on Unfreeze). Su selección elimina los recuadros delimitadores al descongelar la imagen.
- **Fusionar conjuntos** (Merge sets). Su selección permite duplicar los datos en los conjuntos "*train*", "*val*" y "*test*".

- **Entrenamiento del modelo.**

Una vez recopilada una cantidad suficiente de datos, se realiza el entrenamiento del modelo por medio del script "*train_ssd.py*" contenido en la carpeta "*pytorch-ssd*". Para ello es necesario ejecutar un comando desde la terminal de comandos que se compone de:

- El **intérprete de Python** que se empleará para ejecutar el script, Python 3 en este caso.
- El **nombre del script** que desea ejecutar (*train_ssd.py*).
- **Tipo del conjunto de datos** empleados en el entrenamiento (--dataset-type=voc).
- La **ubicación del conjunto de datos** (--data=data/detectpiezasp_alt2f).
- El **directorio para crear los puntos de control del modelo entrenado** (--model-dir=models/detectpiezasp_alt2f).
- El **tamaño del lote empleado** durante el entrenamiento, es decir, el número de imágenes que se procesan en cada paso de entrenamiento (--batch-size=4).
- Cantidad de **subprocesos del cargador de datos** (--workers=1).
- **Número de épocas** que se emplearán para el entrenamiento (--epochs=30).

```
$ python3 train_ssd.py --dataset-type=voc --data=data/detectpiezasp_alt2f --model-dir=models/detectpiezasp_alt2f --batch-size=4 --workers=1 --epochs=30
```

Tras la ejecución del comandando anterior, es posible la aparición de un mensaje de error. Este puede deberse a que la última versión de "*pillow*" solo es compatible con versiones de Python superiores a la 3.7 y como se ha comentado con anterioridad, la versión empleada es la 3.6. Para solucionar este problema se ejecuta el siguiente comando tal y como recomienda uno de los desarrolladores de NVIDIA [63], el cual instalará "*pillow<9*" en el script *"install-pytorch.sh":*

```
$ sudo pip3 install 'pillow<9'
```

Se ejecuta nuevamente el comando de entrenamiento de la red y una vez transcurrido el tiempo necesario para el entrenamiento, debe convertirse el modelo PyTorch a ONNX:

```
$ python3 onnx_export.py --model-dir=models/detectpiezasp_alt2f
```

Al ejecutar dicho comando, se genera un archivo "*ssd-mobilenet.onnx*" en la carpeta "*deteccion-objetos1/jetson-inference/python/training/detection/ssd/pytorch-ssd/models/detectpiezasp_alt2f*" donde se encuentran los modelos entrenados en PyTorch con extensión .pth. Con el fin de facilitar el siguiente proceso, se accede manualmente a dicha carpeta y se cambia el nombre del archivo por uno que presente relación con el tema, "*detectpiezasp_alt2f.onnx*" en este caso.

A continuación, se accede a la carpeta "*deteccion-objetos1/jetson-inference/build/aarch64/bin"*, y se crea una llamada "*detectpiezasp_alt2f*" con el objeto de almacenar de manera organizada lo necesario para la ejecución de la red. En la carpeta mencionada que es desde donde se va a ejecutar el programa, se copian los archivos "*detectpiezasp_alt2f.*onnx" y "*labels.txt*".

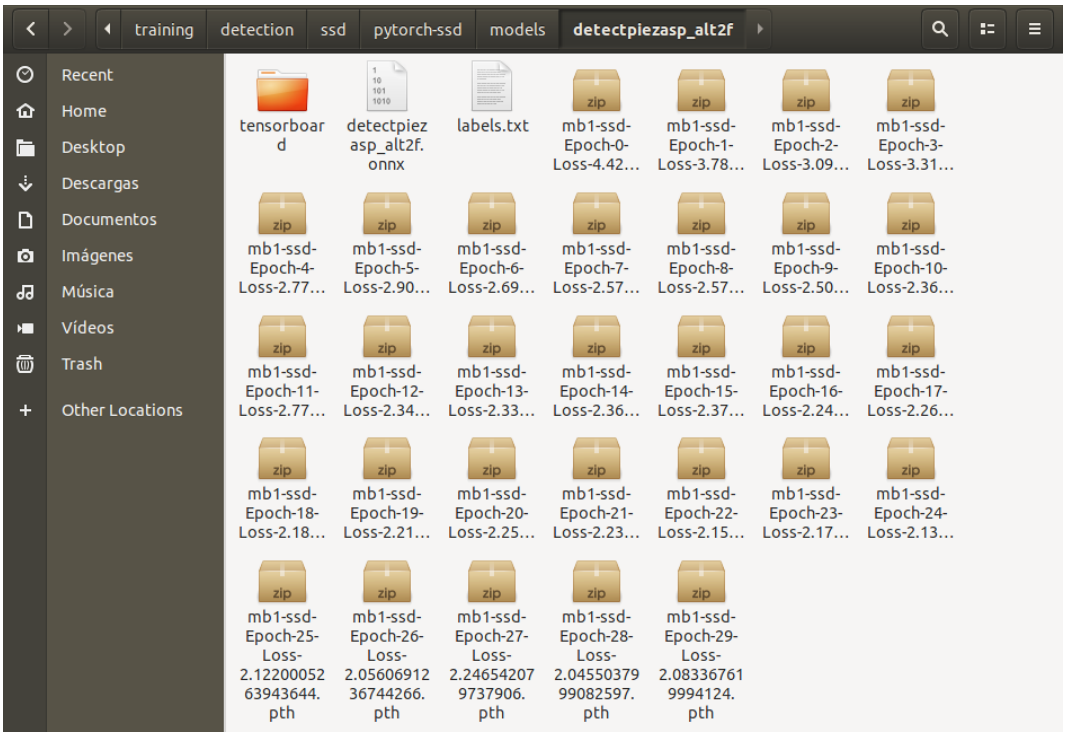

Figura 81. *Almacenamiento de los modelos entrenados en PyTorch para la red de clasificación de piezas.*

C. Red neuronal personalizada para la distinción entre la cara superior e inferior de la pieza 3DCoil.

Al igual que en el apartado anterior, con el apoyo del repositorio [29], se procede a la recopilación del conjunto de datos para la identificación de la cara superior e inferior de la pieza 3DC11LP con el fin de entrenar un modelo que realice dicha distinción. Por tanto, esta sección se divide en dos partes bien diferenciadas, una donde se muestran los pasos a seguir para la recopilación de los datos de detección y otra para realizar el entrenamiento del modelo. En el presente apartado, no se profundizará en los pasos a seguir debido a que ha sido explicado detalladamente en el punto anterior. De este modo, se procederá a exponer únicamente los comandos necesarios para la creación de la red neuronal:

- **Recopilación de datos de detección.**

En primer lugar, se accede a la carpeta "*ssd*" dentro de las carpetas "*training/detection*" del entorno de *Jetson Inference* mediante el siguiente comando, el cual nos indica el directorio en el que se sitúa la carpeta:

```
$ cd deteccion-objetos1/jetson-inference/python/training/detection/ssd/
```

Debido a que previamente ha sido creada la red neuronal encargada de detectar las piezas 3DC11LP y 3DC14EMR-ULP no es necesario realizar ninguna instalación, puesto que se realizó en el apartado anterior. Llegados a este punto, se encuentra en disposición de recopilar un conjunto de datos de detección propio. Para ello desde la carpeta "*data*" situada dentro de la carpeta "*pytorch-ssd*" se crea un nuevo directorio donde se recopilarán los datos del conjunto de detección que se desea crear:

```
$ cd deteccion-objetos1/jetson-
inference/python/training/detection/ssd/pytorch-ssd/data
$ mkdir detectcara_1
```

Dentro de la carpeta creada (detectcara_1), se crea un archivo de texto con el nombre "*labels_c.txt*" donde debe escribirse el nombre de los diferentes objetos o piezas que se desean detectar. En el presente caso lo que se pretende es distinguir entre la cara superior e inferior de la pieza 3DC11LP fabricada por la empresa PREMO, por lo que se escribe dicha información en el archivo de texto mencionado:

- TOP.
- BOTTOM.

Tras la creación del directorio para el almacenamiento de la información del *dataset* y del archivo de texto con las piezas a detectar, se emplea la herramienta "control de captura de datos" desde la carpeta "*pytorch-ssd*":

```
$ cd deteccion-objetos1/jetson-
inference/python/training/detection/ssd/pytorch-ssd
$ camera-capture /dev/video0
```

Tras el último comando ejecutado, aparece la ventana que se muestra en la ***Figura 80***, en la que debe completarse la siguiente información antes de proceder a la recopilación de imágenes:

- **Tipo de datos** (Dataset Type). Cuando el menú desplegable del tipo de datos se encuentra en modo detección, como es el caso, la herramienta crea conjunto de datos en formato Pascal VOC (*Visual Object Classes*).
- **Ruta del conjunto de datos** (Dataset Path). Indicar la ruta en la que se desea almacenar la información del conjunto de datos. En este caso: ***deteccion-objetos1/jetson-inference/python/training/detection/ssd/pytorch-ssd/data/detectcara_1.***
- **Etiquetas de clases** (Class Labels). Indicar la ruta en la que se encuentra el archivo de texto con las diferentes clases a detectar. En este caso: ***deteccion-objetos1/jetson-inference/python/training/detection/ssd/pytorch-ssd/data/detectcara_1 /labels_c.txt.***

- **Conjunto actual** (Current Set). Seleccionar entre los conjuntos "*train*", "*val*" y "*test*" aunque si se selecciona la opción Marge sets, como se menciona a continuación, los datos de replican en los tres conjuntos.
- **Calidad JPEG** (JPEG Quality). Control de calidad de la codificación y el tamaño del disco de las imágenes guardadas.
- **Guardar al descongelar** (Save on Unfreeze). Su selección almacena automáticamente los datos al descongelar la imagen.
- **Borrar al descongelar** (Clear on Unfreeze). Su selección elimina los recuadros delimitadores al descongelar la imagen.
- **Fusionar conjuntos** (Merge sets). Su selección permite duplicar los datos en los conjuntos "*train*", "*val*" y "*test*".

- **Entrenamiento del modelo.**

Una vez recopilada una cantidad suficiente de datos, se realiza el entrenamiento del modelo por medio del script "*train_ssd.py*" contenido en la carpeta "*pytorch-ssd*". Para ello es necesario ejecutar un comando desde la terminal de comandos que se compone de:

- El **intérprete de Python** que se empleará para ejecutar el script, Python 3 en este caso.
- El **nombre del script** que desea ejecutar (*train_ssd.py*).
- **Tipo del conjunto de datos** empleados en el entrenamiento (--dataset-type=voc).
- La **ubicación del conjunto de datos** (*--data=data/detectcara_1*).
- El **directorio para crear los puntos de control del modelo entrenado** (*--model-dir=models/detectcara_1*).
- El **tamaño del lote empleado** durante el entrenamiento, es decir, el número de imágenes que se procesan en cada paso de entrenamiento (*--batch-size=4*).
- Cantidad de **subprocesos del cargador de datos** (*--workers=1*).
- **Número de épocas** que se emplearán para el entrenamiento (*--epochs=30*).

```
$ python3 train_ssd.py --dataset-type=voc --data=data/detectcara_1 --model-
dir=models/detectpiezasp_alt2f --batch-size=4 --workers=1 --epochs=30
```

Tras la ejecución del comandando expuesto, si se ha realizado previamente la red neuronal anterior no es necesario ejecutar el siguiente comando, ya que el error ha sido solucionado previamente y la red neuronal comenzará a entrenarse sin mayor problema:

```
$ sudo pip3 install 'pillow<9'
```

Una vez transcurrido el tiempo necesario para el entrenamiento, debe convertirse el modelo PyTorch a ONNX:

```
$ python3 onnx_export.py --model-dir=models/detectcara_1
```

Al ejecutar dicho comando, se genera un archivo "*ssd-mobilenet.onnx*" en la carpeta "*deteccion-objetos1/jetson-inference/python/training/detection/ssd/pytorch-ssd/models/detectcara_1*" donde se encuentran los modelos entrenados en PyTorch con extensión .pth. Con el fin de facilitar el siguiente proceso, se accede manualmente a dicha carpeta y se cambia el

nombre del archivo por uno que presente relación con el tema, "*detectcara_1.onnx*" en este caso.

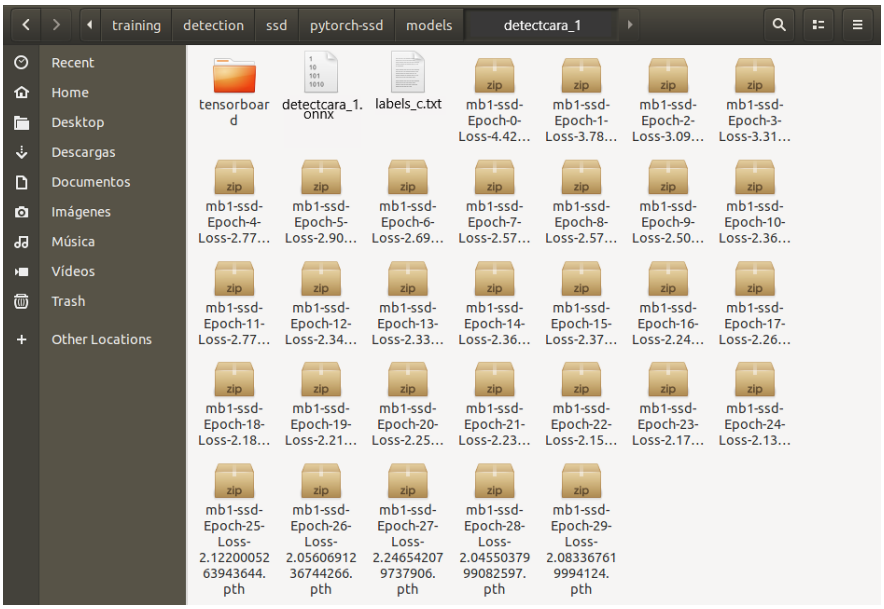

Figura 82. *Almacenamiento de los modelos entrenados en PyTorch para la distinción entre la cara superior e inferior de la pieza 3DC11LP.*

A continuación, se accede a la carpeta "*deteccion-objetos1/jetson-inference/build/aarch64/bin"*, y se crea una llamada "*detectcara_1*" con el objeto de almacenar de manera organizada lo necesario para la ejecución de la red. En la carpeta mencionada que es desde donde se va a ejecutar el programa, se copian los archivos "*detectcara_1*.onnx" y "*labels_c.txt*".

Anexo IV. Códigos

A continuación, se exponen los códigos necesarios para la ejecución de cada una de las implementaciones. En primer lugar, se presentan los módulos que contienen el código generado en archivos separados y que posteriormente son llamados desde el código principal en los casos necesarios.

A. Módulos de código importados en el programa principal

- **Parámetros.**

```
import cv2
from tkinter import filedialog

#   Tipo de Fuente
font= cv2.FONT_HERSHEY_SIMPLEX
fontScale=0.5
crSize=15
iniText=(10,10)

#Distancia minima para la consideracion de una misma pieza
distancia_minima=100
#Distancia desde posicion origen hasta posicion destino (mm)
xrecogida=440
#Posicion inicial (mm)
xorigen=0
```

- **Seguidor de piezas.**

```
import math
from parametros import *

class rastreador:
    def __init__(self):
        #Inicializamos las variables:
        #Posiciones centrales de las piezas:
        self.centros_piezas = {}
        #Contador piezas identificadas:
        self.id_count = 0

    def rastreo(self, objeto):
        #Piezas identificadas
        piezas_ident = []

        #Punto central de la nueva pieza:
        for rect in objeto:
```

```
            x, y, w, h = rect
            cx = (x + x + w) // 2
            cy = (y + y + h) // 2

            #Fue ya detectado?
            p_Ident = False
            for id, pt in self.centros_piezas.items():
                #Calculamos la distancia entre los centros identificados
y los nuevos.
                distancia = math.hypot(cx - pt[0], cy - pt[1])

                #En caso de que la distancia sea menor de la minima
establecida (distancia_minima) es probable que se trate de la misma
pieza:
                if distancia < distancia_minima:
                    self.centros_piezas[id] = (cx, cy)

                    piezas_ident.append([x, y, w, h, id])
                    p_Ident = True
                    break

            #Si se trata de una nueva pieza, le asignamos el id.
            if p_Ident is False:
                self.centros_piezas[self.id_count] = (cx, cy)
                piezas_ident.append([x, y, w, h, self.id_count])
                self.id_count += 1

        #Limpieza lista puntos
        new_centros_piezas = {}

        for obj_id in piezas_ident:
            _, _, _, _, objecto_id = obj_id
            centro = self.centros_piezas[objecto_id]
            new_centros_piezas [objecto_id] = centro

        #Actualizacion lista
        self.centros_piezas = new_centros_piezas.copy()
        return piezas_ident
```

- **Matriz_p1.**

```
import cv2
import math
import numpy as np
from matplotlib import pyplot as plt

def Matriz1 (x_ini,y_ini):
    num_columnas = 6
    num_filas = 6
    contador = 0
    dist_centroFila = 30.09
    dist_centroColumna = 31.37
    num_piezas = num_columnas*num_filas
    r = 2
    x_ini_aux = x_ini
    num_columnas_aux = num_columnas
    Matriz=[]
    num_columnas = num_columnas-1
    x_ini = x_ini + dist_centroColumna

    for i in range (num_piezas):
        Matriz.append([])

    for f in range (num_filas):
        y_cent = y_ini + (f*dist_centroFila)
        #print (y_cent)
        for c in range (num_columnas):
            x_cent = x_ini + (c*dist_centroColumna)
            #print(x_cent)
            Matriz[contador].append(round(x_cent,r))
            Matriz[contador].append(round(y_cent,r))
            contador = contador+1
        x_ini=x_ini_aux
        num_columnas = num_columnas_aux

    return Matriz
```

- **Matriz_p2.**

```
import cv2
import math
import numpy as np
from matplotlib import pyplot as plt

def Matriz2 (x_ini,y_ini):
    num_columnas = 8
    num_filas = 10
    contador = 0
    dist_centroFila = 31.37
    dist_centroColumna = 30.09
    num_piezas = num_columnas*num_filas
    r = 2
    x_ini_aux = x_ini
    num_columnas_aux = num_columnas
    Matriz=[]
    num_columnas = num_columnas-1
    x_ini = x_ini + dist_centroColumna

    for i in range (num_piezas):
        Matriz.append([])

    for f in range (num_filas):
        y_cent = y_ini + (f*dist_centroFila)
        #print (y_cent)
        for c in range (num_columnas):
            x_cent = x_ini + (c*dist_centroColumna)
            #print(x_cent)
            Matriz[contador].append(round(x_cent,r))
            Matriz[contador].append(round(y_cent,r))
            contador = contador+1
        x_ini=x_ini_aux
        num_columnas = num_columnas_aux

    return Matriz
```

B. Clasificador de piezas

A continuación, se presenta el código con el que se ejecuta la implementación del clasificador de piezas.

```
import cv2
import jetson.inference
import jetson.utils
import time
import traceback
import numpy as np
import math
import matplotlib.pyplot as plt
import os

from rastreador import *
from xarm import version
from xarm.wrapper import XArmAPI
from Matriz_p2 import *
from Matriz_p1 import *

#Declaración del detector de piezas
net = jetson.inference.detectNet(argv=['--
model=detectpiezasp_alt2f.onnx','--labels=labels.txt','--input-
blob=input_0','--output-cvg=scores','--output-bbox=boxes'],
threshold=0.5)

#Declaración del detector de caras de la 3DC11LP
net2 = jetson.inference.detectNet(argv=['--model=detectcara_1.onnx','--
labels=labels_c.txt','--input-blob=input_0','--output-cvg=scores','--
output-bbox=boxes'],threshold=0.5)

#Declaracion de la camara
camera = cv2.VideoCapture("/dev/video0")
camera.set(cv2.CAP_PROP_FRAME_HEIGHT,480)
camera.set(cv2.CAP_PROP_FRAME_WIDTH,640)
camera.set(cv2.CAP_PROP_BUFFERSIZE,1)

#camara = jetson.utils.videoSource("/dev/video0")
#display = jetson.utils.videoOutput()


print('xArm-Python-SDK Version:{}'.format(version.__version__))

# Conexión al robot xArm
xarm = XArmAPI('192.168.1.195')
xarm.clean_warn()
```

```
xarm.clean_error()
xarm.motion_enable(True)
xarm.set_mode(0)
xarm.set_state(0)
xarm.set_self_collision_detection(True)
time.sleep(1)

situacion='GUARDAR_POS2'

variables = {}
params = {'speed': 1000, 'acc': 50000, 'angle_speed': 180, 'angle_acc':
1146, 'events': {}, 'variables': variables, 'callback_in_thread': True,
'quit': False}

# Declaramos algunas constantes:

pos_inicial=[8.5, -18.1, -46.4, -0.8, 64.2, 6.5]
pos_in_1=[39,6.3,-64,-0.3,57.8,40.7]
pos_in_2=[69.8,-26,-31.3,-0.2,57.5,71.5]
pos_final_b=[56.8,-12.7,-47.2,-0.9,58.8,3.6]
pos_final_m=[-46,-19.9,-37,-0.8,57.1,6.5]

# Posicion inicial del robot:
x_ini = 377.513306
y_ini = 55.591763
z_ini = 380.5

#Alt finales:
alt_f1_c=240.2
alt_f2_c=239.1

#Parametros para el almacenamiento de la pieza 3DC14EMR-ULP
contPiezas2=0
contPiezasA2=0
num_colum=10

#Parametros para el almacenamiento de la pieza 3DC11LP
contPiezas1=0
contPiezasA1=0

def calculo_centro_lados (b,h):

    h_centro=(h/2)
    b_centro=(b/2)

    return b_centro,h_centro
```

```
def calculo_centro_esq (x1,y1,x2,y2):

    x_centro=((x1+x2)/2)
    y_centro=((y1+y2)/2)

    return x_centro,y_centro

def distancia_centro_pieza (xcp,ycp,bc,hc):

    dx=(xcp-bc)
    dy=(ycp-hc)

    return dx,dy

def calculo_distancia (dx, dy):

    # Calculo de la distancia en unidades de medida. (Teorema de
Pitágoras)
    dist = math.sqrt((dx ** 2) + (dy ** 2))

    return dist

def calculo_coordenadas (dx,dy,x_ini, y_ini):

    dx=-dx*0.236
    dy=-dy*0.236

    # Calculo de las nuevas coordenadas del robot
    xf = x_ini + dy
    yf = y_ini + dx

    return xf, yf

def calculo_coordenadas_blister (dx,dy,x_ini, y_ini):

    dx=-dx*0.167
    dy=-dy*0.167

    # Calculo de las nuevas coordenadas del robot
    xf = x_ini + dy
    yf = y_ini + dx

    return xf, yf

def clasificadorPuntos(puntos):
```

```
    numPuntos = puntos

    y_Clas_puntos = sorted(numPuntos, key=lambda numPuntos: numPuntos[1])
    x1_Clas_puntos = y_Clas_puntos[:2]

    x1_Clas_puntos = sorted(x1_Clas_puntos, key=lambda x1_Clas_puntos:
x1_Clas_puntos[0])
    x2_Clas_puntos = y_Clas_puntos[2:4]

    x2_Clas_puntos = sorted(x2_Clas_puntos, key=lambda x2_Clas_puntos:
x2_Clas_puntos[0])

    pClasificados=[x1_Clas_puntos[0],
x1_Clas_puntos[1],x2_Clas_puntos[1], x2_Clas_puntos[0]]

    if pClasificados[0][1]>pClasificados[1][1]:

        #Calculo distancia de punto a punto (base):
        Dif_AX=pClasificados[1][0]-pClasificados[0][0]
        Dif_AY=pClasificados[1][1]-pClasificados[0][1]

        #Calculo de la base:
        BASEpieza=math.sqrt((Dif_AX**2)+(Dif_AY**2))

        #Angulo en radianes y grados de la base:
        ang_rad=math.atan(Dif_AY/Dif_AX)
        ang_deg=math.degrees(ang_rad)
        ang_deg=360+ang_deg-180

        #Calculo distancia de punto a punto (alto):
        Dif_AXh=pClasificados[1][0]-pClasificados[2][0]
        Dif_AYh=pClasificados[2][1]-pClasificados[1][1]

        #Calculo altura pieza:
        ALTOpieza=math.sqrt((Dif_AXh**2)+(Dif_AYh**2))

        #Angulo en radianes y grados de la altura:
        ang_radh=math.atan(Dif_AXh/Dif_AYh)
        ang_degh=math.degrees(ang_radh)
        ang_degh=(-ang_degh-90)

        if BASEpieza>ALTOpieza:
            pClasificados=[pClasificados[1],pClasificados[2],pClasificado
s[3], pClasificados[0]]
            ang_deg=-ang_degh

    else:
```

```
            Dif_AX=pClasificados[1][0]-pClasificados[0][0]
            Dif_AY=pClasificados[0][1]-pClasificados[1][1]

            BASEpieza=math.sqrt((Dif_AX**2)+(Dif_AY**2))
            ang_rad=math.atan(Dif_AY/Dif_AX)
            ang_deg=math.degrees(ang_rad)
            ang_deg=-ang_deg

            Dif_AXh=pClasificados[2][0]-pClasificados[1][0]
            Dif_AYh=pClasificados[2][1]-pClasificados[1][1]

            ALTOpieza=math.sqrt((Dif_AXh**2)+(Dif_AYh**2))
            ang_radh=math.atan(Dif_AYh/Dif_AXh)
            ang_degh=math.degrees(ang_radh)
            ang_degh=-ang_degh-90-90

            if BASEpieza>ALTOpieza:
                pClasificados=[pClasificados[3],pClasificados[0],pClasificado
    s[1], pClasificados[2]]

                ang_deg=-ang_degh

        return pClasificados, ang_deg, BASEpieza, ALTOpieza

    while (True):

        #Leemos los fotogramas através de la cámara
        #img = camara.Capture()

        #Obtencion de la posicion del blister de la pieza 3DC14EMR-ULP
        if situacion == 'GUARDAR_POS2':
            xarm.set_servo_angle(angle=pos_in_2, speed=params['angle_speed'],
    mvacc=params['angle_acc'], wait=True, radius=-1.0)
            pose_ini2 = xarm.get_position()

            pose_ini2=pose_ini2[1]
            #Me quedo con los 5 primeros elementos
            pose_ini2=pose_ini2[:2]
            print (pose_ini2)

            #Guardamos las posiciones
            posex_robot2_ini=pose_ini2[0]
            posey_robot2_ini=pose_ini2[1]

            camera.set(cv2.CAP_PROP_BUFFERSIZE,1)
            r,frame = camera.read()
```

```
        r,frame = camera.read()
        r,frame = camera.read()
        img =
jetson.utils.cudaFromNumpy(cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA).astype(
np.float))

        #Sacamos el ancho y el alto de la imagen para calcular el centro
        alto_img=img.shape[0] #Filas
        ancho_img=img.shape[1] #Columnas

        anchoc_img,altoc_img=calculo_centro_lados(ancho_img,alto_img)
        print('Centro imagen: ancho={},
alt={}'.format(anchoc_img,altoc_img))

        detect= net.Detect(img)

        if detect and situacion=='GUARDAR_POS2':

            for det in detect:
                #Determinamos que objeto es (si es 3DC11LP o 3DC14EMR-
ULP):
                clase = det.ClassID

                #Si se detecta 3DC11LP
                if clase == 1:
                    print('3DC11LP')
                    color =(0,0,225)
                    cv2.putText(frame,'3DC11LP',(int(det.Left),int(det.To
p)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(255,0,0),1)


                #Si se detecta 3DC14EMR-ULP
                elif clase == 2:
                    print('3DC14EMR-ULP')
                    color=(255,0,0)
                    cv2.putText(frame,'3DC14EMR-
ULP',(int(det.Left),int(det.Top)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(0,255,0),1)

                    # Calcular coordenadas del centro de la pieza
```

```
                    x1,y1,x2,y2=det.Left, det.Top, det.Right,
det.Bottom
                    xc_INI2,yc_INI2=calculo_centro_esq(x1,y1,x2,y2)
                    print('Centro pieza: ancho={},
alt={}'.format(xc_INI2, yc_INI2))

                    #Calculo distancia
                    dx,dy =
distancia_centro_pieza(xc_INI2,yc_INI2,anchoc_img,altoc_img)

                    x_robotf_2, y_robotf_2 =
calculo_coordenadas_blister(dx,dy,posex_robot2_ini,posey_robot2_ini)
                    z_robotf_2=300
                    pos_matriz_p2=NuevaMatrizV2(x_robotf_2, y_robotf_2)
                    print(pos_matriz_p2)

                if xc_INI2>anchoc_img and yc_INI2<altoc_img: #Primer
cuadrante
                    #Incremento
                    Ix_2=52
                    Iy_2=-5
                    print('C1')

                elif xc_INI2<anchoc_img and yc_INI2<altoc_img: #Segundo
cuadrante
                    #Incremento
                    Ix_2=50
                    Iy_2=-7
                    print('C2')

                elif xc_INI2<anchoc_img and yc_INI2>altoc_img: #Tercero
cuadrante
                    #Incremento
                    Ix_2=56
                    Iy_2=-4.5
                    print('C3')

                elif xc_INI2>anchoc_img and yc_INI2>altoc_img: #Cuarto
cuadrante
                    #Incremento
                    Ix_2=54
                    Iy_2=-3.5
                    print('C4')

        situacion='GUARDAR_POS1'
```

```
        #Obtencion de la posicion del blister de la pieza 3DC11LP

        if situacion == 'GUARDAR_POS1':
            xarm.set_servo_angle(angle=pos_in_1, speed=params['angle_speed'],
    mvacc=params['angle_acc'], wait=True, radius=-1.0)
            time.sleep(0.5)
            pose_ini1 = xarm.get_position()
            print(pose_ini1)

            pose_ini1=pose_ini1[1]
            #Me quedo con los 5 primeros elementos
            pose_ini1=pose_ini1[:2]
            print (pose_ini1)

            #Guardamos las posiciones
            posex_robot1_ini=pose_ini1[0]
            posey_robot1_ini=pose_ini1[1]

            camera.set(cv2.CAP_PROP_BUFFERSIZE,0)
            r,frame = camera.read()
            r,frame = camera.read()

            img =
    jetson.utils.cudaFromNumpy(cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA).astype(
    np.float))

            #Sacamos el ancho y el alto de la imagen para calcular el centro
            alto_img=img.shape[0] #Filas
            ancho_img=img.shape[1] #Columnas

            anchoc_img,altoc_img=calculo_centro_lados(ancho_img,alto_img)
            print('Centro imagen: ancho={},
    alt={}'.format(anchoc_img,altoc_img))

            detect= net.Detect(img)

            if detect and situacion=='GUARDAR_POS1':

                for det in detect:
                    #Determinamos que objeto es (si es 3DC11LP o 3DC14EMR-
    ULP):
                    clase = det.ClassID

                    #Si se detecta 3DC11LP
                    if clase == 1:
                        print('3DC11LP')
                        color =(0,0,225)
```

```
                    cv2.putText(frame,'3DC11LP',(int(det.Left),int(det.To
p)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(255,0,0),1)
                    # Calcular coordenadas del centro de la pieza
                    x1,y1,x2,y2=det.Left, det.Top, det.Right,
det.Bottom
                    xc_INI1,yc_INI1=calculo_centro_esq(x1,y1,x2,y2)
                    print('Centro pieza: ancho={},
alt={}'.format(xc_INI1, yc_INI1))

                    #Calculo distancia
                    dx,dy =
distancia_centro_pieza(xc_INI1,yc_INI1,anchoc_img,altoc_img)

                    x_robotf_1, y_robotf_1 =
calculo_coordenadas_blister(dx,dy,posex_robot1_ini,posey_robot1_ini)
                    z_robotf_1=300
                    pos_matriz_p1=NuevaMatrizV1(x_robotf_1, y_robotf_1)
                    print(pos_matriz_p1)

                #Si se detecta 3DC14EMR-ULP
                elif clase == 2:
                    print('3DC14EMR-ULP')
                    color=(255,0,0)
                    cv2.putText(frame,'3DC14EMR-
ULP',(int(det.Left),int(det.Top)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(0,255,0),1)


                if xc_INI1>anchoc_img and yc_INI1<altoc_img: #Primer
cuadrante
                    #Incremento
                    Ix_1=52
                    Iy_1=-6.5
                    print('C1')

                elif xc_INI1<anchoc_img and yc_INI1<altoc_img: #Segundo
cuadrante
                    #Incremento
                    Ix_1=50
```

```
                    Iy_1=-11
                    print('C2')

                elif xc_INI1<anchoc_img and yc_INI1>altoc_img: #Tercero
cuadrante
                    #Incremento
                    Ix_1=56
                    Iy_1=-8
                    print('C3')

                elif xc_INI1>anchoc_img and yc_INI1>altoc_img: #Cuarto
cuadrante
                    #Incremento
                    Ix_1=54
                    Iy_1=-6
                    print('C4')

        situacion='INICIAL'

    if situacion == 'INICIAL':
        xarm.set_servo_angle(angle=pos_inicial,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
        situacion='BUSQUEDA'

    camera.set(cv2.CAP_PROP_BUFFERSIZE,0)
    r,frame = camera.read()
    r,frame = camera.read()

    img =
jetson.utils.cudaFromNumpy(cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA).astype(
np.float))

    #Sacamos el ancho y el alto de la imagen para calcular el centro
    alto_img=img.shape[0] #Filas
    ancho_img=img.shape[1] #Columnas

    anchoc_img,altoc_img=calculo_centro_lados(ancho_img,alto_img)
    print('Centro imagen: ancho={}, alt={}'.format(anchoc_img,altoc_img))
    t_inid=time.time()
    # Deteccion
    detect= net.Detect(img)
    t_finald=round(((time.time()-t_inid)*1000),2)

    #Creamos un objeto de seguimiento
    seguimiento = rastreador()
```

```
    #.....................................................
    robot=False
    #declaramos un array vacio para almacenar los vertices
    puntos = []


    # Busqueda de piezas 3DC11LP y 3DC14EMR-ULP
    if situacion == 'BUSQUEDA':

        if detect and situacion=='BUSQUEDA':
            piezas=[]

            for det in detect:
                #Determinamos que objeto es (si es 3DC11LP o 3DC14EMR-
ULP):
                clase = det.ClassID

                #Si se detecta 3DC11LP
                if clase == 1:
                    print('3DC11LP')
                    color =(0,0,225)
                    cv2.putText(frame,'3DC11LP',(int(det.Left),int(det.To
p)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(255,0,0),1)
                    PIEZA='1'

                #Si se detecta 3DC14EMR-ULP
                elif clase == 2:
                    print('3DC14EMR-ULP')
                    color=(255,0,0)
                    cv2.putText(frame,'3DC14EMR-
ULP',(int(det.Left),int(det.Top)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(0,255,0),1)
                    PIEZA='2'

                # Calcular coordenadas del centro de la pieza
                x1,y1,x2,y2=det.Left, det.Top, det.Right,
det.Bottom
                xc_pieza,yc_pieza=calculo_centro_esq(x1,y1,x2,y2)
```

```
                #print('Centro pieza: ancho={},
alt={}'.format(xc_pieza,yc_pieza))
                # Calcular distancia entre el centro de la pieza y el
centro de la imagen
                dx,dy =
distancia_centro_pieza(xc_pieza,yc_pieza,anchoc_img,altoc_img)
                distancia=calculo_distancia (dx,dy)
                # Agregar información de la pieza y su distancia a la
lista
                piezas.append({'pieza': PIEZA, 'xc': xc_pieza, 'yc':
yc_pieza, 'distancia': distancia,
'dx':dx,'dy':dy,'x1':x1,'x2':x2,'y1':y1,'y2':y2})
                #print(piezas)

            #COGER
            # Ordenar la lista por distancia en orden ascendente
            piezas = sorted(piezas, key=lambda k: k['distancia'])
            print(piezas)
            # Seleccionar la primera pieza en la lista (la mas cercana al
centro de la imagen)
            pieza_seleccionada = piezas[0]
            print(pieza_seleccionada)

            # Calcular las coordenadas de la pieza seleccionada
            xc_pieza, yc_pieza = pieza_seleccionada['xc'],
pieza_seleccionada['yc']
            print('Centro pieza: ancho={}, alt={}'.format(xc_pieza,
yc_pieza))
            pie_select=pieza_seleccionada['pieza']
            PIEZA=pie_select
            print('La pieza seleccionada es de la
clase={}'.format(pie_select))

            # Calcular distancia entre el centro de la pieza y el centro
de la imagen
            dx,dy = pieza_seleccionada['dx'], pieza_seleccionada['dy']
            distancia=pieza_seleccionada['distancia']
            print('La pieza seleccionada esta a la siguiente distancia
dx={}, dy={}, distancia={}'.format(dx,dy,distancia))

            x1,x2,y1,y2 = pieza_seleccionada['x1'],
pieza_seleccionada['x2'],pieza_seleccionada['y1'],
pieza_seleccionada['y2']
            print('La pieza seleccionada: x1={}, x2={}, y1={},
y2={},'.format(x1,x2,y1,y2))
            print('La pieza seleccionadas es: {}'.format(PIEZA))
```

```
            x_final, y_final = calculo_coordenadas(dx,dy,x_ini,y_ini)
            z_final=255
            print('Recogiendo pieza.')
            #cv2.putText(frame,'(C1)',(10,50),cv2.FONT_HERSHEY_SIMPLEX,1,
color,3)

            print('Coordenadas finales: x={},
y={}'.format(x_final,y_final))
            print('Coordenadas INICIALES: x={},
y={}'.format(x_ini,y_ini))
            #cv2.putText(frame,'(pF)',(int(x_final),int(y_final)),cv2.FON
T_HERSHEY_SIMPLEX,1,color,3)

            cv2.putText(frame,'*',(int(anchoc_img),int(altoc_img)),cv2.FO
NT_HERSHEY_SIMPLEX,1,color,3)
            cv2.putText(frame,'o',(int(xc_pieza),int(yc_pieza)),cv2.FONT_
HERSHEY_SIMPLEX,1,color,3)
            plt.plot
((int(anchoc_img),int(altoc_img)),(int(xc_pieza),int(yc_pieza)))
            #cv2.putText(frame,'(0,0)',(0,0),cv2.FONT_HERSHEY_SIMPLEX,1,c
olor,3)
            cv2.putText(frame,f'Tiempo: {t_finald}
ms',(0,25),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)

            if PIEZA=='1':

                if xc_pieza>anchoc_img and yc_pieza<altoc_img: #Primer
cuadrante
                    #Incremento
                    Ix=53.5
                    Iy=0.5
                    print('C1')

                elif xc_pieza<anchoc_img and yc_pieza<altoc_img: #Segundo
cuadrante
                    #Incremento
                    Ix=51.5
                    Iy=3.50
                    print('C2')

                elif xc_pieza<anchoc_img and yc_pieza>altoc_img: #Tercero
cuadrante
                    #Incremento
                    Ix=49
                    Iy=1.5
                    print('C3')
```

```
                elif xc_pieza>anchoc_img and yc_pieza>altoc_img: #Cuarto
cuadrante
                    #Incremento
                    Ix=52
                    Iy=-1.5
                    print('C4')

            elif PIEZA=='2':

                if xc_pieza>anchoc_img and yc_pieza<altoc_img: #Primer
cuadrante
                    #Incremento
                    Ix=54
                    Iy=3
                    print('C1')

                elif xc_pieza<anchoc_img and yc_pieza<altoc_img: #Segundo
cuadrante
                    #Incremento
                    Ix=50
                    Iy=7
                    print('C2')

                elif xc_pieza<anchoc_img and yc_pieza>altoc_img: #Tercero
cuadrante
                    #Incremento
                    Ix=50
                    Iy=4
                    print('C3')

                elif xc_pieza>anchoc_img and yc_pieza>altoc_img: #Cuarto
cuadrante
                    #Incremento
                    Ix=52
                    Iy=0
                    print('C4')


            #angle=xarm.get_inverse_kinematics([x_ini,y_ini,z_ini,179.3,-
0.5,3.9])
            #print(angle)

            #posicion_coger=[x_final, y_final, z_final,180,0,0]
            xarm.set_position(x_final, y_final, z_final,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
```

```
            #angle=xarm.get_inverse_kinematics([x_final,y_final,z_final,1
79.3,-0.5,3.9])
            #print(angle)
            #pose = arm.get_position()
            #print(pose)

            situacion='R_PIEZA'


    if situacion=='R_PIEZA':

        if PIEZA=='1':

            #Leemos los fotogramas através de la cámara
            #camera.set(cv2.CAP_PROP_BUFFERSIZE,1)
            r,frame = camera.read()
            r,frame = camera.read()
            r,frame = camera.read()
            img2 =
jetson.utils.cudaFromNumpy(cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA).astype(
np.float))

            # Deteccion de la CARA de la PIEZA 1
            detect= net2.Detect(img2)
            CARA='NO'

            # Si hay una deteccion de TOP o BOTTOM
            if detect:

                for det in detect:
                    CARA='NO'
                    #Determinamos que objeto es (si es 3DC11LP o
3DC14EMR-ULP):
                    clase = det.ClassID

                    #Si se detecta la cara TOP
                    if clase == 1:
                        CARA='TOP'
                        print('TOP')
                        color =(0,0,225)
                        cv2.putText(frame,'TOP',(int(det.Left),int(det.To
p)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                        #cv2.putText(frame,'Esq2',(int(det.Right),int(det
.Bottom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                        cv2.rectangle(frame,(int(det.Left),int(det.Top)),
(int(det.Right),int(det.Bottom)),(255,0,0),1)
```

```
                        #xarm.set_servo_angle(angle=pos_final_b,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
                        situacion='ROTACION'
                        break

                    #Si se detecta la cara BOTTOM
                    elif clase == 2:
                        CARA='BOTTOM'
                        print('BOTTOM')
                        color =(0,0,225)
                        cv2.putText(frame,'BOTTOM',(int(det.Left),int(det
.Top)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                        #cv2.putText(frame,'Esq2',(int(det.Right),int(det
.Bottom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                        cv2.rectangle(frame,(int(det.Left),int(det.Top)),
(int(det.Right),int(det.Bottom)),(255,0,0),1)

                        #Movemos a la pos de la pieza
                        xarm.set_position(x_final+Ix, y_final+Iy,
z_final, speed=params['speed'], mvacc=params['angle_speed'], wait=True,
radius=-1.0)
                        xarm.set_position(x_final+Ix, y_final+Iy,
alt_f1_c, speed=params['speed'], mvacc=params['angle_speed'], wait=True,
radius=-1.0)
                        xarm.set_suction_cup(True,
wait=False,delay_sec=0)
                        time.sleep(0.2)

                        xarm.set_servo_angle(angle=pos_final_m,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
                        time.sleep(0.2)
                        xarm.set_suction_cup(False,
wait=True,delay_sec=0)
                        situacion='INICIAL'
                        break

            if CARA != 'TOP' and CARA!='BOTTOM':
                print ('NO se ha detectado correctamente la pieza.')

                #Movemos a la pos de la pieza
                xarm.set_position(x_final+Ix, y_final+Iy, z_final,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
```

```
                xarm.set_position(x_final+Ix, y_final+Iy, alt_f1_c,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
                xarm.set_suction_cup(True, wait=False,delay_sec=0)
                time.sleep(0.2)

                xarm.set_servo_angle(angle=pos_final_m,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
                time.sleep(0.2)
                xarm.set_suction_cup(False, wait=True,delay_sec=0)
                situacion='INICIAL'


        elif PIEZA=='2':
            #xarm.set_servo_angle(angle=pos_final_b,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
            situacion='ROTACION'

    if situacion=='ROTACION':

            r,frame = camera.read()
            r,frame = camera.read()
            if r == True:          # si hay imagen
                ancho = frame.shape[1] #columnas
                alto = frame.shape[0] # filas

                #----   Procesamiento imagen  ----
                frame_gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                _, binarizada = cv2.threshold(frame_gris, 25, 255,
cv2.THRESH_BINARY_INV)

                #----   Eliminacion de sombras mediante suavizado  ----
                filtro=cv2.GaussianBlur(binarizada, (11,11), 0) ##(11,11)

                #----  Eliminacion de grises mediante umbral de
binarizacion  ----
                _, umbral=cv2.threshold(filtro, 150, 255,
cv2.THRESH_BINARY)  ##50

                #----  Eliminacion del ruido  ----
                dila= cv2.dilate(umbral, np.ones((1,1))) #(3,3)

                #----  Creacion de un kernel (mascara)  ----
```

```
                kernel=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
(5,5))  #(3,3)

                #----  Aplicacion del kerner creado para unir los pixeles
dispersos  ----
                cerrar=cv2.morphologyEx(dila, cv2.MORPH_CLOSE,kernel)

                #Se establece el area minima y maxima que desea
identificarse:
                Area_min=100
                Area_max=100000

                #----  Contornos   ----
                contornos,_=cv2.findContours(cerrar, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

                info_deteccion=[] #Lista de la informacion detecciones
                cuenta=0
                found=0

                for cont in contornos:
                    if ( (cv2.contourArea(cont)>Area_min) &
(cv2.contourArea(cont)<Area_max))==False:
                        cuenta=cuenta+1
                    else:
                        #Rectangulo:
                        x,y,w,h=cv2.boundingRect(cont)

                        #Almacenamiento info:
                        info_deteccion.append([x,y,w,h])

                        #Rectangulo con inclinacion:
                        rect = cv2.minAreaRect(cont)
                        box = cv2.boxPoints(rect)
                        box = np.int0(box)

                        #Centro del contorno para areas irregulares
                        M = cv2.moments(cont)
                        if (M["m_pos00"]==0): M["m_pos00"]=1
                        x = int(M["m_pos10"]/M["m_pos00"])
                        y = int(M['m_pos01']/M['m_pos00'])

                        #Nuevo contorno (mejoria)
                        nuevoContorno = cv2.convexHull(cont)
                        print(box)

                #SEGUIDOR:
```

```
                    info_id=seguimiento.rastreo(info_deteccion)


                    for inf in info_id:
                        #Coordenadas:
                        x, y, w, h, id= inf

                        #Se muestra el contorno por pantalla por medio de un
    rectangulo:
                        cv2.rectangle(frame, (x,y),(x+w,y+h),(0,0,255),2)

                        #Se muestra el rectangulo con el angulo de
    inclinacion:
                        cv2.drawContours(frame,[box],0,(0,255,255),2)

                        #Centro:
                        cx=int(x + w / 2)
                        cy=int(y + h / 2)

                        #Se llama a la funcion clasificadorPuntos
                        box_salida, ang_deg, BASEpieza, ALTOpieza =
    clasificadorPuntos(box)

                        #Se almacenan los puntos obtenidos:
                        punto_1=(int(box_salida[0][0]),
    int(box_salida[0][1]))
                        punto_2=(int(box_salida[1][0]),
    int(box_salida[1][1]))
                        punto_3= (int(box_salida[2][0]),
    int(box_salida[2][1]))
                        punto_4=(int(box_salida[3][0]),
    int(box_salida[3][1]))

                        #Distancia hasta el contorno:
                        Dif_1=cv2.pointPolygonTest(cont, punto_1, True)
                        Dif_2=cv2.pointPolygonTest(cont, punto_2, True)
                        Dif_3=cv2.pointPolygonTest(cont, punto_3, True)
                        Dif_4=cv2.pointPolygonTest(cont, punto_4, True)

                        #Obtencion lados pieza:
                        baseMayor=int(abs(Dif_1))+int(abs(Dif_2))
                        baseInferior=int(abs(Dif_3))+int(abs(Dif_4))

                        if baseMayor > baseInferior:
```

```
                        #Se escribe 'HEAD' en la "cabeza" del rectangulo
y asi tener una referencia:
                        cv2.putText(frame,'HEAD',punto_3, font, 0.7,
(250,0,250))
                        if ( (ang_deg>90) & (ang_deg<180)):
                            a=ang_deg
                        else:
                            a=ang_deg+180

                    if baseMayor < baseInferior:

                        #Se escribe 'HEAD' en la "cabeza" del rectangulo
y asi tener una referencia:
                        cv2.putText(frame,'HEAD',punto_1, font, 0.7,
(250,0,250))
                        if ( (ang_deg>90) & (ang_deg<180)):
                            a=ang_deg+180
                        else:
                            a=ang_deg


                    #Vertices rectangulo:
                    cv2.putText(frame,'1',(punto_1[0],punto_1[1]), font,
0.7, (255, 0, 0))
                    cv2.putText(frame,'2',(punto_2[0],punto_2[1]), font,
0.7, (255, 0, 0))
                    cv2.putText(frame,'3',(punto_3[0],punto_3[1]), font,
0.7, (255, 0, 0))
                    cv2.putText(frame,'4',(punto_4[0],punto_4[1]), font,
0.7, (255, 0, 0))

                    #Se muestra el angulo calculado en grados:
                    print(ang_deg)

                    #Guardamos la posicion en la que esta el robot
                    pose=xarm.get_servo_angle()
                    print(pose)
                    #Me quedo con el segundo elemento de la tupla
                    pose=pose[1]
                    #Me quedo con los 5 primeros elementos
                    pose=pose[:5]
                    print (pose)

                    #Guardamos las posiciones
                    pose0=pose[0]
                    pose1=pose[1]
                    pose2=pose[2]
```

```
                        pose3=pose[3]
                        pose4=pose[4]

                        if PIEZA=='1':

                            if ang_deg>90 and ang_deg<180:
                                ang_deg=ang_deg-90

                            elif ang_deg>=180 and ang_deg<270:
                                ang_deg=ang_deg-180

                            elif ang_deg>=270:
                                ang_deg=ang_deg-270

                            angulo=(ang_deg)+6.5

                            pose_f=[pose0,pose1,pose2,pose3,pose4,angulo]
                            print(pose_f)
                            xarm.set_servo_angle(angle=[pose0,pose1,pose2,pos
e3,pose4,angulo], speed=params['angle_speed'], mvacc=params['angle_acc'],
wait=True, radius=-1.0)
                            situacion='MOVE'


                        elif PIEZA=='2':

                            if ang_deg>180:
                                ang_deg=ang_deg-180

                            angulo=(ang_deg-90)+6.5

                            pose_f=[pose0,pose1,pose2,pose3,pose4,angulo]
                            print(pose_f)
                            xarm.set_servo_angle(angle=[pose0,pose1,pose2,pos
e3,pose4,angulo], speed=params['angle_speed'], mvacc=params['angle_acc'],
wait=True, radius=-1.0)
                            situacion='MOVE'

                    cv2.imshow('camara',frame)
                    cv2.imshow('Mascara ',cerrar)

        if situacion=='MOVE':
            if PIEZA=='1':
                #Movemos a la pos de la pieza
```

```
            xarm.set_position(x_final+Ix, y_final+Iy, z_final,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
            xarm.set_position(x_final+Ix, y_final+Iy, alt_f1_c,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
            xarm.set_suction_cup(True, wait=False,delay_sec=0)
            xarm.set_position(x_final+Ix, y_final+Iy, 260,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)

            time.sleep(0.2)
            situacion='FINAL'

        elif PIEZA=='2':
            xarm.set_position(x_final+Ix, y_final+Iy, z_final,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
            xarm.set_position(x_final+Ix, y_final+Iy, alt_f2_c,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
            xarm.set_suction_cup(True, wait=False,delay_sec=0)
            time.sleep(0.2)
            xarm.set_position(x_final+Ix, y_final+Iy, 260,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)

            situacion='FINAL'


    if situacion =='FINAL':

        #Posicion de almacenamiento para la pieza 3DC11LP
        if PIEZA=='1':

            x_pos_almacenar = pos_matriz_p1[contPiezas1][0]
            y_pos_almacenar = pos_matriz_p1[contPiezas1][1]

            #Se incrementa el contador de piezas 3DC11LP almacenadas en
una unidad
            contPiezas1 = contPiezas1+1

            posicionDepositar =
xarm.set_position(x_pos_almacenar+Ix_1,y_pos_almacenar+Iy_1,
280,speed=params['speed'],mvacc=params['angle_speed'], radius=50,
wait=True)
```

```
                angulos_Pre_Almacenamiento = xarm.get_servo_angle()

                pos_Pre_Almacenamiento = angulos_Pre_Almacenamiento[1]

                if(contPiezasA1==0):

                    angulosPos = xarm.get_servo_angle()
                    pos_Alm_1 = angulosPos[1]

                if(contPiezasA1==num_colum):

                    pos_Pre_Almacenamiento=pos_Alm_1
                    contPiezasA1=-1

                contPiezasA1 = contPiezasA1+1

                situacion='ALMACENAMIENTO'

            #Posicion de almacenamiento para la pieza 3DC14EMR-ULP
            elif PIEZA=='2':

                x_pos_almacenar = pos_matriz_p2[contPiezas2][0]
                y_pos_almacenar = pos_matriz_p2[contPiezas2][1]

                #Se incrementa el contador de piezas 3DC14EMR-ULP almacenadas
    en una unidad
                contPiezas2 = contPiezas2+1

                posicionDepositar =
    xarm.set_position(x_pos_almacenar+Ix_2,y_pos_almacenar+Iy_2,
    280,speed=params['speed'],mvacc=params['angle_speed'], radius=50,
    wait=True)

                #Angulos robot
                angulos_Pre_Almacenamiento = xarm.get_servo_angle()
                pos_Pre_Almacenamiento = angulos_Pre_Almacenamiento[1]

                if(contPiezasA2==0):

                    angulosPos = xarm.get_servo_angle()
                    pos_Alm_1 = angulosPos[1]

                if(contPiezasA2==num_colum):

                    pos_Pre_Almacenamiento=pos_Alm_1
                    contPiezasA2=-1
```

```
            contPiezasA2 = contPiezasA2+1

            situacion='ALMACENAMIENTO'

    if situacion=='ALMACENAMIENTO':

        #Almacenamiento de la pieza 3DC11LP
        if PIEZA =='1':

            posicionAlmacenaje=[x_pos_almacenar+Ix_1,y_pos_almacenar+Iy_1
,252,180,0,0]
            xarm.set_position(*posicionAlmacenaje,speed=params['speed'],m
vacc=params['angle_speed'], radius=50, wait=True)
            xarm.set_suction_cup(False, wait=True,delay_sec=0)

            situacion='INICIAL'

        #Almacenamiento de la pieza 3DC14EMR-ULP
        elif PIEZA=='2':

            posicionAlmacenaje=[x_pos_almacenar+Ix_2,y_pos_almacenar+Iy_2
,245,180,0,0]
            xarm.set_position(*posicionAlmacenaje,speed=params['speed'],m
vacc=params['angle_speed'], radius=50, wait=True)
            xarm.set_suction_cup(False, wait=True,delay_sec=0)

            situacion='INICIAL'

    # Renderizamos la imagen
    cv2.imshow("window",frame)
    if cv2.waitKey(1) == ord("q"):
        break


camera.release()
cv2.destroyAllWindows()
```

C. Rastreador de piezas

En el presente apartado del Anexo IV se presenta el código con el que se ejecuta la aplicación del rastreador de piezas.

```
import cv2
import jetson.inference
import jetson.utils
import time
import traceback
import numpy as np
import math
import matplotlib.pyplot as plt
import os


from xarm import version
from xarm.wrapper import XArmAPI

#Declaración del detector
net = jetson.inference.detectNet(argv=['--
model=detectpiezasp_alt2f.onnx','--labels=labels.txt','--input-
blob=input_0','--output-cvg=scores','--output-bbox=boxes'],
threshold=0.5)

#Declaracion de la camara
camara = jetson.utils.videoSource("/dev/video0")
display = jetson.utils.videoOutput()

#camera = cv2.VideoCapture("/dev/video0")
#camera.set(cv2.CAP_PROP_FRAME_HEIGHT,480)
#camera.set(cv2.CAP_PROP_FRAME_WIDTH,640)
#camera.set(cv2.CAP_PROP_BUFFERSIZE,1)

print('xArm-Python-SDK Version:{}'.format(version.__version__))

# Conexión al robot xArm
xarm = XArmAPI('192.168.1.195')
xarm.clean_warn()
xarm.clean_error()
xarm.motion_enable(True)
xarm.set_mode(0)
xarm.set_state(0)
time.sleep(1)


pos_inicial=[6.3, 7.3, -91.1, -0.7, 83.5, 6.5]
```

```
variables = {}
params = {'speed': 800, 'acc': 1000, 'angle_speed': 180*0.17,
'angle_acc': 170, 'events': {}, 'variables': variables,
'callback_in_thread': True, 'quit': False}


def calculo_centro_lados (b,h):
    h_centro=(h/2)
    b_centro=(b/2)
    return b_centro,h_centro

def calculo_centro_esq (x1,y1,x2,y2):

    x_centro=((x1+x2)/2)
    y_centro=((y1+y2)/2)
    return x_centro,y_centro

def distancia_centro_pieza (xcp,ycp,bc,hc):
    dx=(xcp-bc)
    dy=(ycp-hc)
    return dx,dy


def calculo_distancia (dx, dy):
    # Calculo de la distancia en unidades de medida. Teorema de
Pitágoras.
    dist = math.sqrt((dx ** 2) + (dy ** 2))
    return dist

def calculo_coordenadas (dx,dy,x_ini, y_ini):
    dx=-dx*0.236
    dy=-dy*0.236

    # Calculo de las nuevas coordenadas del robot
    xf = x_ini + dy
    yf = y_ini + dx

    return xf, yf

xarm.set_servo_angle(angle=pos_inicial, speed=params['angle_speed'],
mvacc=params['angle_acc'], wait=True, radius=-1.0)
situacion='INICIAL'

while True:
```

```
        tini=time.time()


        if situacion == 'INICIAL':

            pose_ini = xarm.get_position()
            #print(pose_ini)

            pose_ini=pose_ini[1]
            #Me quedo con los 5 primeros elementos
            pose_ini=pose_ini[:3]
            #print (pose_ini)

            #Guardamos las posiciones
            x_ini=pose_ini[0]
            y_ini=pose_ini[1]
            z_ini=pose_ini[2]

            situacion='BUSQUEDA'

        #Leemos los fotogramas através de la cámara
        img = camara.Capture()


        #Sacamos el ancho y el alto de la imagen para calcular el centro
        alto_img=img.shape[0] #Filas
        ancho_img=img.shape[1] #Columnas

        anchoc_img,altoc_img=calculo_centro_lados(ancho_img,alto_img)

        time.sleep(0.25)
        # Deteccion
        detect= net.Detect(img)


        # Si hay una deteccion de 3DC11LP o 3DC14EMR-ULP
        if situacion=='BUSQUEDA':

            if detect and situacion=='BUSQUEDA':

                for det in detect:
                    #Determinamos que objeto es (si es 3DC11LP o 3DC14EMR-
    ULP):
                    clase = det.ClassID

                    #Si se detecta 3DC11LP
```

```
                if clase == 1:
                    print('3DC11LP')
                    # Calcular coordenadas del centro de la pieza
                    x1,y1,x2,y2=det.Left, det.Top, det.Right,
det.Bottom
                    xc_pieza,yc_pieza=calculo_centro_esq(x1,y1,x2,y2)
                    #print('Centro pieza: ancho={},
alt={}'.format(xc_pieza,yc_pieza))
                    # Calcular distancia entre el centro de la pieza y el
centro de la imagen
                    dx,dy =
distancia_centro_pieza(xc_pieza,yc_pieza,anchoc_img,altoc_img)
                    distancia=calculo_distancia (dx,dy)
                    x_final, y_final =
calculo_coordenadas(dx,dy,x_ini,y_ini)
                    z_final=z_ini
                    angles=xarm.get_inverse_kinematics([x_final, y_final,
z_final,179.3,-0.5,3.9])
                    #print(angle)
                    angles=angles[1]
                    angles=angles[:6]
                    #print(angles)

                    xarm.set_servo_angle(angle=angles,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=False,
radius=-1.0)
                    print(time.time()-tini)
                    #time.sleep(0.08)
                    situacion='INICIAL'



                #Si se detecta 3DC14EMR-ULP
                elif clase == 2:
                    print('3DC14EMR-ULP')
                    # Calcular coordenadas del centro de la pieza
                    x1,y1,x2,y2=det.Left, det.Top, det.Right,
det.Bottom
                    xc_pieza,yc_pieza=calculo_centro_esq(x1,y1,x2,y2)
                    #print('Centro pieza: ancho={},
alt={}'.format(xc_pieza,yc_pieza))
                    # Calcular distancia entre el centro de la pieza y el
centro de la imagen
                    dx,dy =
distancia_centro_pieza(xc_pieza,yc_pieza,anchoc_img,altoc_img)
                    distancia=calculo_distancia (dx,dy)
```

```
                    x_final, y_final =
calculo_coordenadas(dx,dy,x_ini,y_ini)
                    z_final=z_ini
                    angles=xarm.get_inverse_kinematics([x_final, y_final,
z_final,179.3,-0.5,3.9])
                    #print(angle)
                    angles=angles[1]
                    angles=angles[:6]
                    #print(angles)

                    xarm.set_servo_angle(angle=angles,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=False,
radius=-1.0)
                    print(time.time()-tini)
                    #time.sleep(0.08)
                    situacion='INICIAL'



    # Renderizamos la imagen
    display.Render(img)
    display.SetStatus("Deteccion Piezas Premo | Network
{:.0f}FPS".format(net.GetNetworkFPS()))

    #Condicion para cerrar el programa
    if not camara.IsStreaming() or not display.IsStreaming():
        break
```

D. Clasificador de piezas con cinta transportadora

Seguidamente, se expone el código con el que se ejecuta la implementación del clasificador y rastreador de piezas:

```
import cv2
import jetson.inference
import jetson.utils
import time
import traceback
import numpy as np
import math
import matplotlib.pyplot as plt
import os

from rastreador import *
from xarm import version
from xarm.wrapper import XArmAPI
from Matriz_p2 import *
from Matriz_p1 import *

#Declaración del detector de piezas
net = jetson.inference.detectNet(argv=['--
model=detectpiezasp_alt2f.onnx','--labels=labels.txt','--input-
blob=input_0','--output-cvg=scores','--output-bbox=boxes'],
threshold=0.5)

#Declaración del detector de caras de la pieza 3DC11LP
net2 = jetson.inference.detectNet(argv=['--model=detectcara_1.onnx','--
labels=labels_c.txt','--input-blob=input_0','--output-cvg=scores','--
output-bbox=boxes'],threshold=0.5)

#Declaracion de la camara

camera = cv2.VideoCapture("/dev/video0")
camera.set(cv2.CAP_PROP_FRAME_HEIGHT,480)
camera.set(cv2.CAP_PROP_FRAME_WIDTH,640)
camera.set(cv2.CAP_PROP_BUFFERSIZE,1)

#camara = jetson.utils.videoSource("/dev/video0")
#display = jetson.utils.videoOutput()

print('xArm-Python-SDK Version:{}'.format(version.__version__))

# Conexión al robot xArm
xarm = XArmAPI('192.168.1.195')
xarm.clean_warn()
```

```
xarm.clean_error()
xarm.motion_enable(True)
xarm.set_mode(0)
xarm.set_state(0)
xarm.set_self_collision_detection(True)
time.sleep(1)

situacion='GUARDAR_POS2'

variables = {}
params = {'speed': 1000, 'acc': 50000, 'angle_speed': 180, 'angle_acc':
1146, 'events': {}, 'variables': variables, 'callback_in_thread': True,
'quit': False}
#params = {'speed': 100, 'acc': 500, 'angle_speed': 18, 'angle_acc': 114,
'events': {}, 'variables': variables, 'callback_in_thread': True, 'quit':
False}
params2 = {'speed': 800, 'acc': 1000, 'angle_speed': 180*0.17,
'angle_acc': 170, 'events': {}, 'variables': variables,
'callback_in_thread': True, 'quit': False}

# Declaramos algunas constantes

#pos_inicial=[6.3, 7.3, -91.1, -0.7, 83.5, 6.5]
pos_inicial=[6.3, 7.5, -92.4, -0.7, 84.6, 6.5]
pos_in_1=[39,6.3,-64,-0.3,57.8,40.7]
pos_in_2=[69.8,-26,-31.3,-0.2,57.5,71.5]
pos_final_b=[56.8,-12.7,-47.2,-0.9,58.8,3.6]
pos_final_m=[-45.6,-27.1,-43.6,-0.1,69.1,6.5]

# Posicion inicial del robot
x_ini = 511.6
y_ini = 55.4
z_ini = 492.4

#Alt finales
alt_f1_c=354.6
alt_f2_c=353.6

#Parametros para el almacenamiento de la pieza 3DC14EMR-ULP
contPiezas2=0
contPiezasA2=0
num_colum=10

#Parametros para el almacenamiento de la pieza 3DC11LP
contPiezas1=0
contPiezasA1=0
```

```
#Parametros del contador que cuenta las veces que realiza la busqueda de
una pieza
contPBusqueda=0
contP=0
contVecesBusqueda=0

def calculo_centro_lados (b,h):

    h_centro=(h/2)
    b_centro=(b/2)

    return b_centro,h_centro

def calculo_centro_esq (x1,y1,x2,y2):

    x_centro=((x1+x2)/2)
    y_centro=((y1+y2)/2)

    return x_centro,y_centro

def distancia_centro_pieza (xcp,ycp,bc,hc):

    dx=(xcp-bc)
    dy=(ycp-hc)

    return dx,dy

def calculo_distancia (dx, dy):

    # Calculo de la distancia en unidades de medida. Teorema de
Pitágoras.
    dist = math.sqrt((dx ** 2) + (dy ** 2))

    return dist

def calculo_coordenadas (dx,dy,x_ini, y_ini):
    dx=-dx*0.236
    dy=-dy*0.236

    # Calculo de las nuevas coordenadas del robot
    xf = x_ini + dy
    yf = y_ini + dx

    return xf, yf

def calculo_coordenadas_blister (dx,dy,x_ini, y_ini):
```

```
        dx=-dx*0.167
        dy=-dy*0.167

        # Calculo de las nuevas coordenadas del robot
        xf = x_ini + dy
        yf = y_ini + dx

        return xf, yf

    def clasificadorPuntos(puntos):

        numPuntos = puntos

        y_Clas_puntos = sorted(numPuntos, key=lambda numPuntos: numPuntos[1])
        x1_Clas_puntos = y_Clas_puntos[:2]

        x1_Clas_puntos = sorted(x1_Clas_puntos, key=lambda x1_Clas_puntos:
    x1_Clas_puntos[0])
        x2_Clas_puntos = y_Clas_puntos[2:4]

        x2_Clas_puntos = sorted(x2_Clas_puntos, key=lambda x2_Clas_puntos:
    x2_Clas_puntos[0])
        pClasificados=[x1_Clas_puntos[0],
    x1_Clas_puntos[1],x2_Clas_puntos[1], x2_Clas_puntos[0]]

        if pClasificados[0][1]>pClasificados[1][1]:

            #Calculo distancia de punto a punto (base):
            Dif_AX=pClasificados[1][0]-pClasificados[0][0]
            Dif_AY=pClasificados[1][1]-pClasificados[0][1]

            #Calculo del lado:
            BASEpieza=math.sqrt((Dif_AX**2)+(Dif_AY**2))

            #Angulo en radianes y grados:
            ang_rad=math.atan(Dif_AY/Dif_AX)
            ang_deg=math.degrees(ang_rad)
            ang_deg=360+ang_deg-180

            #Calculo distancia de punto a punto (alto):
            Dif_AXh=pClasificados[1][0]-pClasificados[2][0]
            Dif_AYh=pClasificados[2][1]-pClasificados[1][1]

            #Calculo altura pieza:
            ALTOpieza=math.sqrt((Dif_AXh**2)+(Dif_AYh**2))
```

```
            #Angulo en radianes y grados de la altura:
            ang_radh=math.atan(Dif_AXh/Dif_AYh)
            ang_degh=math.degrees(ang_radh)
            ang_degh=(-ang_degh-90)

            if BASEpieza>ALTOpieza:
                pClasificados=[pClasificados[1],pClasificados[2],pClasificado
    s[3], pClasificados[0]]
                ang_deg=-ang_degh


        else:
            Dif_AX=pClasificados[1][0]-pClasificados[0][0]
            Dif_AY=pClasificados[0][1]-pClasificados[1][1]

            BASEpieza=math.sqrt((Dif_AX**2)+(Dif_AY**2))
            ang_rad=math.atan(Dif_AY/Dif_AX)
            ang_deg=math.degrees(ang_rad)
            ang_deg=-ang_deg

            Dif_AXh=pClasificados[2][0]-pClasificados[1][0]
            Dif_AYh=pClasificados[2][1]-pClasificados[1][1]

            ALTOpieza=math.sqrt((Dif_AXh**2)+(Dif_AYh**2))
            ang_radh=math.atan(Dif_AYh/Dif_AXh)
            ang_degh=math.degrees(ang_radh)
            ang_degh=-ang_degh-90-90

            if BASEpieza>ALTOpieza:

                pClasificados=[pClasificados[3],pClasificados[0],pClasificado
    s[1], pClasificados[2]]

                ang_deg=-ang_degh

        return pClasificados, ang_deg, BASEpieza, ALTOpieza


    #llamada a la funcion para seleccionar la region de interes y ajustar el
    ubral para la deteccion de la pieza
    #THmin, THmax =get_threshvals()

    while (True):

        #Leemos los fotogramas através de la cámara
        #img = camara.Capture()
```

```
        #Obtencion de la posicion del blister de la pieza 3DC14EMR-ULP
        if situacion == 'GUARDAR_POS2':
            xarm.set_servo_angle(angle=pos_in_2, speed=params['angle_speed'],
    mvacc=params['angle_acc'], wait=True, radius=-1.0)
            pose_ini2 = xarm.get_position()

            pose_ini2=pose_ini2[1]
            #Me quedo con los 5 primeros elementos
            pose_ini2=pose_ini2[:2]
            print (pose_ini2)

            #Guardamos las posiciones
            posex_robot2_ini=pose_ini2[0]
            posey_robot2_ini=pose_ini2[1]


            camera.set(cv2.CAP_PROP_BUFFERSIZE,1)
            r,frame = camera.read()
            r,frame = camera.read()
            r,frame = camera.read()
            img =
    jetson.utils.cudaFromNumpy(cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA).astype(
    np.float))

            #Sacamos el ancho y el alto de la imagen para calcular el centro
            alto_img=img.shape[0] #Filas
            ancho_img=img.shape[1] #Columnas

            anchoc_img,altoc_img=calculo_centro_lados(ancho_img,alto_img)
            print('Centro imagen: ancho={},
    alt={}'.format(anchoc_img,altoc_img))

            detect= net.Detect(img)

            if detect and situacion=='GUARDAR_POS2':

                for det in detect:
                    #Determinamos que objeto es (si es 3DC11LP o 3DC14EMR-
    ULP):
                    clase = det.ClassID

                    #Si se detecta 3DC11LP
                    if clase == 1:
                        print('3DC11LP')
                        color =(0,0,225)
```

```
                    cv2.putText(frame,'3DC11LP',(int(det.Left),int(det.To
p)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(255,0,0),1)

                #Si se detecta 3DC14EMR-ULP
                elif clase == 2:
                    print('3DC14EMR-ULP')
                    color=(255,0,0)
                    cv2.putText(frame,'3DC14EMR-
ULP',(int(det.Left),int(det.Top)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(0,255,0),1)

                    # Calcular coordenadas del centro de la pieza
                    x1,y1,x2,y2=det.Left, det.Top, det.Right,
det.Bottom
                    xc_INI2,yc_INI2=calculo_centro_esq(x1,y1,x2,y2)
                    print('Centro pieza: ancho={},
alt={}'.format(xc_INI2, yc_INI2))

                    #Calculo distancia
                    dx,dy =
distancia_centro_pieza(xc_INI2,yc_INI2,anchoc_img,altoc_img)

                    x_robotf_2, y_robotf_2 =
calculo_coordenadas_blister(dx,dy,posex_robot2_ini,posey_robot2_ini)
                    z_robotf_2=300
                    pos_matriz_p2=NuevaMatrizV2(x_robotf_2, y_robotf_2)
                    print(pos_matriz_p2)

                if xc_INI2>anchoc_img and yc_INI2<altoc_img: #Primer
cuadrante
                    #Incremento
                    Ix_2=52
                    Iy_2=-5
                    print('C1')

                elif xc_INI2<anchoc_img and yc_INI2<altoc_img: #Segundo
cuadrante
                    #Incremento
                    Ix_2=50
                    Iy_2=-7
```

```
                    print('C2')

                elif xc_INI2<anchoc_img and yc_INI2>altoc_img: #Tercero
cuadrante
                    #Incremento
                    Ix_2=56
                    Iy_2=-4.5
                    print('C3')

                elif xc_INI2>anchoc_img and yc_INI2>altoc_img: #Cuarto
cuadrante
                    #Incremento
                    Ix_2=54
                    Iy_2=-3.5
                    print('C4')

        situacion='GUARDAR_POS1'

    #Obtencion de la posicion del blister de la pieza 3DC11LP
    if situacion == 'GUARDAR_POS1':
        xarm.set_servo_angle(angle=pos_in_1, speed=params['angle_speed'],
mvacc=params['angle_acc'], wait=True, radius=-1.0)
        time.sleep(0.5)
        pose_ini1 = xarm.get_position()
        print(pose_ini1)

        pose_ini1=pose_ini1[1]
        #Me quedo con los 5 primeros elementos
        pose_ini1=pose_ini1[:2]
        print (pose_ini1)

        #Guardamos las posiciones
        posex_robot1_ini=pose_ini1[0]
        posey_robot1_ini=pose_ini1[1]

        camera.set(cv2.CAP_PROP_BUFFERSIZE,0)
        r,frame = camera.read()
        r,frame = camera.read()

        img =
jetson.utils.cudaFromNumpy(cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA).astype(
np.float))

        #Sacamos el ancho y el alto de la imagen para calcular el centro
        alto_img=img.shape[0] #Filas
        ancho_img=img.shape[1] #Columnas
```

```
        anchoc_img,altoc_img=calculo_centro_lados(ancho_img,alto_img)
        print('Centro imagen: ancho={},
alt={}'.format(anchoc_img,altoc_img))

        detect= net.Detect(img)

        if detect and situacion=='GUARDAR_POS1':

            for det in detect:
                #Determinamos que objeto es (si es 3DC11LP o 3DC14EMR-
ULP):
                clase = det.ClassID

                #Si se detecta 3DC11LP
                if clase == 1:
                    print('3DC11LP')
                    color =(0,0,225)
                    cv2.putText(frame,'3DC11LP',(int(det.Left),int(det.To
p)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(255,0,0),1)
                    # Calcular coordenadas del centro de la pieza
                    x1,y1,x2,y2=det.Left, det.Top, det.Right,
det.Bottom
                    xc_INI1,yc_INI1=calculo_centro_esq(x1,y1,x2,y2)
                    print('Centro pieza: ancho={},
alt={}'.format(xc_INI1, yc_INI1))

                    #Calculo distancia
                    dx,dy =
distancia_centro_pieza(xc_INI1,yc_INI1,anchoc_img,altoc_img)

                    x_robotf_1, y_robotf_1 =
calculo_coordenadas_blister(dx,dy,posex_robot1_ini,posey_robot1_ini)
                    z_robotf_1=300
                    pos_matriz_p1=NuevaMatrizV1(x_robotf_1, y_robotf_1)
                    print(pos_matriz_p1)


                #Si se detecta 3DC14EMR-ULP
                elif clase == 2:
                    print('3DC14EMR-ULP')
                    color=(255,0,0)
```

```
                    cv2.putText(frame,'3DC14EMR-
ULP',(int(det.Left),int(det.Top)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(0,255,0),1)


                if xc_INI1>anchoc_img and yc_INI1<altoc_img: #Primer
cuadrante
                    #Incremento
                    Ix_1=52
                    Iy_1=-6.5
                    print('C1')

                elif xc_INI1<anchoc_img and yc_INI1<altoc_img: #Segundo
cuadrante
                    #Incremento
                    Ix_1=50
                    Iy_1=-11
                    print('C2')

                elif xc_INI1<anchoc_img and yc_INI1>altoc_img: #Tercero
cuadrante
                    #Incremento
                    Ix_1=56
                    Iy_1=-8
                    print('C3')

                elif xc_INI1>anchoc_img and yc_INI1>altoc_img: #Cuarto
cuadrante
                    #Incremento
                    Ix_1=54
                    Iy_1=-6
                    print('C4')

        situacion='INICIAL'

    if situacion == 'INICIAL':
        xarm.set_servo_angle(angle=pos_inicial,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
        time.sleep(0.5)
        #Activación cinta
        xarm.set_cgpio_digital(3,1, delay_sec=0)
        xarm.set_cgpio_digital(2,1, delay_sec=0)
```

```
            contVecesBusqueda=0
            situacion='POS_ACTUAL'

        camera.set(cv2.CAP_PROP_BUFFERSIZE,0)
        r,frame = camera.read()
        r,frame = camera.read()

        img =
    jetson.utils.cudaFromNumpy(cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA).astype(
    np.float))

        #Sacamos el ancho y el alto de la imagen para calcular el centro
        alto_img=img.shape[0] #Filas
        ancho_img=img.shape[1] #Columnas

        anchoc_img,altoc_img=calculo_centro_lados(ancho_img,alto_img)
        print('Centro imagen: ancho={}, alt={}'.format(anchoc_img,altoc_img))

        time.sleep(0.25)
        # Deteccion
        detect= net.Detect(img)

        #Creamos un objeto de seguimiento
        seguimiento = rastreador()

        #.............................................................
        robot=False
        #declaramos un array vacio para almacenar los vertices
        puntos = []

        if situacion == 'POS_ACTUAL':

            pose_ini = xarm.get_position()
            #print(pose_ini)

            pose_ini=pose_ini[1]
            #Me quedo con los 5 primeros elementos
            pose_ini=pose_ini[:3]
            #print (pose_ini)

            #Guardamos las posiciones
            x_ini=pose_ini[0]
            y_ini=pose_ini[1]
            z_ini=pose_ini[2]

            situacion='PERSEGUIR'
```

```
        #Rastreo de piezas en la cinta
        if situacion =='PERSEGUIR':

            if detect and situacion=='PERSEGUIR':
                piezasc=[]


                for det in detect:
                    #Determinamos que objeto es (si es 3DC11LP o 3DC14EMR-
    ULP):
                    clase = det.ClassID

                    #Si se detecta 3DC11LP
                    if clase == 1:
                        print('3DC11LP')
                        PIEZAC='1'


                    #Si se detecta 3DC14EMR-ULP
                    elif clase == 2:
                        print('3DC14EMR-ULP')
                        PIEZAC='2'


                    # Calcular coordenadas del centro de la pieza
                    x1,y1,x2,y2=det.Left, det.Top, det.Right,
    det.Bottom
                    xc_pieza,yc_pieza=calculo_centro_esq(x1,y1,x2,y2)
                    #print('Centro pieza: ancho={},
    alt={}'.format(xc_pieza,yc_pieza))
                    # Calcular distancia entre el centro de la pieza y el
    centro de la imagen
                    dx,dy =
    distancia_centro_pieza(xc_pieza,yc_pieza,anchoc_img,altoc_img)
                    distancia=calculo_distancia (dx,dy)
                    # Agregar información de la pieza y su distancia a la
    lista
                    piezasc.append({'pieza': PIEZAC, 'xc': xc_pieza, 'yc':
    yc_pieza, 'distancia': distancia,
    'dx':dx,'dy':dy,'x1':x1,'x2':x2,'y1':y1,'y2':y2})
                    #print(piezas)

                #COGER
                # Ordenar la lista por distancia en orden ascendente
                piezasc = sorted(piezasc, key=lambda k: k['distancia'])
                print(piezasc)
```

```
            # Seleccionar la primera pieza en la lista (la mas cercana al
centro de la imagen)
            pieza_seleccionada = piezasc[0]
            print(pieza_seleccionada)

            # Calcular las coordenadas de la pieza seleccionada
            xc_pieza, yc_pieza = pieza_seleccionada['xc'],
pieza_seleccionada['yc']
            print('Centro pieza: ancho={}, alt={}'.format(xc_pieza,
yc_pieza))
            pie_select=pieza_seleccionada['pieza']
            PIEZAC=pie_select
            print('La pieza seleccionada es de la
clase={}'.format(pie_select))

            # Calcular distancia entre el centro de la pieza y el centro
de la imagen
            dx,dy = pieza_seleccionada['dx'], pieza_seleccionada['dy']
            distancia=pieza_seleccionada['distancia']
            print('La pieza seleccionada esta a la siguiente distancia
dx={}, dy={}, distancia={}'.format(dx,dy,distancia))

            x1,x2,y1,y2 = pieza_seleccionada['x1'],
pieza_seleccionada['x2'],pieza_seleccionada['y1'],
pieza_seleccionada['y2']
            print('La pieza seleccionada: x1={}, x2={}, y1={},
y2={},'.format(x1,x2,y1,y2))
            print('La pieza seleccionadas es: {}'.format(PIEZAC))

            x_final, y_final = calculo_coordenadas(dx,dy,x_ini,y_ini)
            z_final=z_ini
            angles=xarm.get_inverse_kinematics([x_final, y_ini,
z_final,179.3,-0.5,3.9])
            #print(angle)
            angles=angles[1]
            angles=angles[:6]
            #print(angles)

            #Solo persigue la pieza en caso de que la pieza se encuentre
en el tercer o cuarto cuadrante
            if (xc_pieza<anchoc_img and yc_pieza>altoc_img) or
(xc_pieza>anchoc_img and yc_pieza>altoc_img):
                xarm.set_servo_angle(angle=angles,
speed=params2['angle_speed'], mvacc=params2['angle_acc'], wait=False,
radius=-1.0)

            if x_final > 220:
```

```
                    situacion='POS_ACTUAL'

                elif x_final <= 220:
                    #Desactivamos cinta
                    pose_fin=xarm.get_servo_angle()

                    #Me quedo con el segundo elemento de la tupla
                    posef=pose_fin[1]
                    #Me quedo con los 5 primeros elementos
                    pose=posef[:6]

                    #Guardamos las posiciones
                    posef0=pose[0]
                    posef1=pose[1]
                    posef2=pose[2]
                    posef3=pose[3]
                    posef4=pose[4]
                    posef5=pose[5]

                    pose_final=[posef0,posef1,posef2,posef3,posef4,posef5]

                    xarm.set_cgpio_digital(2,0, delay_sec=0)
                    time.sleep(0.5)
                    situacion='BUSQUEDA'


        # Busqueda de piezas 3DC11LP y 3DC14EMR-ULP
        if situacion == 'BUSQUEDA':
            #time.sleep(0.25)
            contPBusqueda=0
            contVecesBusqueda +=1

            if detect and situacion=='BUSQUEDA':
                piezas=[]

                for det in detect:

                    #Determinamos que objeto es (si es 3DC11LP o 3DC14EMR-
    ULP):
                    clase = det.ClassID

                    #Si se detecta 3DC11LP
                    if clase == 1:
                        print('3DC11LP')
                        color =(0,0,225)
```

```
                    cv2.putText(frame,'3DC11LP',(int(det.Left),int(det.To
p)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(255,0,0),1)
                    PIEZA='1'
                    contPBusqueda += 1


                #Si se detecta 3DC14EMR-ULP
                elif clase == 2:
                    print('3DC14EMR-ULP')
                    color=(255,0,0)
                    cv2.putText(frame,'3DC14EMR-
ULP',(int(det.Left),int(det.Top)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    #cv2.putText(frame,'Esq2',(int(det.Right),int(det.Bot
tom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                    cv2.rectangle(frame,(int(det.Left),int(det.Top)),(int
(det.Right),int(det.Bottom)),(0,255,0),1)
                    PIEZA='2'
                    contPBusqueda += 1


                # Calcular coordenadas del centro de la pieza
                x1,y1,x2,y2=det.Left, det.Top, det.Right,
det.Bottom
                xc_pieza,yc_pieza=calculo_centro_esq(x1,y1,x2,y2)
                #print('Centro pieza: ancho={},
alt={}'.format(xc_pieza,yc_pieza))
                # Calcular distancia entre el centro de la pieza y el
centro de la imagen
                dx,dy =
distancia_centro_pieza(xc_pieza,yc_pieza,anchoc_img,altoc_img)
                distancia=calculo_distancia (dx,dy)
                # Agregar información de la pieza y su distancia a la
lista
                piezas.append({'pieza': PIEZA, 'xc': xc_pieza, 'yc':
yc_pieza, 'distancia': distancia,
'dx':dx,'dy':dy,'x1':x1,'x2':x2,'y1':y1,'y2':y2})
                #print(piezas)

            #COGER
            # Ordenar la lista por distancia en orden ascendente
            piezas = sorted(piezas, key=lambda k: k['distancia'])
            print(piezas)
```

```
            # Seleccionar la primera pieza en la lista (la mas cercana al
centro de la imagen)
            pieza_seleccionada = piezas[0]
            print(pieza_seleccionada)

            # Calcular las coordenadas de la pieza seleccionada
            xc_pieza, yc_pieza = pieza_seleccionada['xc'],
pieza_seleccionada['yc']
            print('Centro pieza: ancho={}, alt={}'.format(xc_pieza,
yc_pieza))
            pie_select=pieza_seleccionada['pieza']
            PIEZA=pie_select
            print('La pieza seleccionada es de la
clase={}'.format(pie_select))

            # Calcular distancia entre el centro de la pieza y el centro
de la imagen
            dx,dy = pieza_seleccionada['dx'], pieza_seleccionada['dy']
            distancia=pieza_seleccionada['distancia']
            print('La pieza seleccionada esta a la siguiente distancia
dx={}, dy={}, distancia={}'.format(dx,dy,distancia))

            x1,x2,y1,y2 = pieza_seleccionada['x1'],
pieza_seleccionada['x2'],pieza_seleccionada['y1'],
pieza_seleccionada['y2']
            print('La pieza seleccionada: x1={}, x2={}, y1={},
y2={},'.format(x1,x2,y1,y2))
            print('La pieza seleccionadas es: {}'.format(PIEZA))

            x_final, y_final = calculo_coordenadas(dx,dy,x_ini,y_ini)
            z_final=370
            print('Recogiendo pieza.')
            #cv2.putText(frame,'(C1)',(10,50),cv2.FONT_HERSHEY_SIMPLEX,1,
color,3)

            print('Coordenadas finales: x={},
y={}'.format(x_final,y_final))
            print('Coordenadas INICIALES: x={},
y={}'.format(x_ini,y_ini))
            #cv2.putText(frame,'(pF)',(int(x_final),int(y_final)),cv2.FON
T_HERSHEY_SIMPLEX,1,color,3)

            cv2.putText(frame,'*',(int(anchoc_img),int(altoc_img)),cv2.FO
NT_HERSHEY_SIMPLEX,1,color,3)
            cv2.putText(frame,'o',(int(xc_pieza),int(yc_pieza)),cv2.FONT_
HERSHEY_SIMPLEX,1,color,3)
```

```
            plt.plot
((int(anchoc_img),int(altoc_img)),(int(xc_pieza),int(yc_pieza)))
            #cv2.putText(frame,'(0,0)',(0,0),cv2.FONT_HERSHEY_SIMPLEX,1,c
olor,3)
            #cv2.putText(frame,f'Tiempo: {t_finald}
ms',(0,25),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)


            if PIEZA=='1':

                if xc_pieza>anchoc_img and yc_pieza<altoc_img: #Primer
cuadrante
                    #Incremento
                    Ix=53.5
                    Iy=0.5
                    print('C1')

                elif xc_pieza<anchoc_img and yc_pieza<altoc_img: #Segundo
cuadrante
                    #Incremento
                    Ix=51.5
                    Iy=3.50
                    print('C2')

                elif xc_pieza<anchoc_img and yc_pieza>altoc_img: #Tercero
cuadrante
                    #Incremento
                    Ix=49
                    Iy=1.5
                    print('C3')

                elif xc_pieza>anchoc_img and yc_pieza>altoc_img: #Cuarto
cuadrante
                    #Incremento
                    Ix=52
                    Iy=-1.5
                    print('C4')


            elif PIEZA=='2':

                if xc_pieza>anchoc_img and yc_pieza<altoc_img: #Primer
cuadrante
                    #Incremento
                    Ix=54
                    Iy=3
                    print('C1')
```

```
                elif xc_pieza<anchoc_img and yc_pieza<altoc_img: #Segundo
cuadrante
                    #Incremento
                    Ix=50
                    Iy=7
                    print('C2')

                elif xc_pieza<anchoc_img and yc_pieza>altoc_img: #Tercero
cuadrante
                    #Incremento
                    Ix=50
                    Iy=4
                    print('C3')

                elif xc_pieza>anchoc_img and yc_pieza>altoc_img: #Cuarto
cuadrante
                    #Incremento
                    Ix=52
                    Iy=0
                    print('C4')


        if contVecesBusqueda == 1:
            contP=contPBusqueda+1
            Ix=Ix-4.5
        print('Este es contP antes de entrar:')
        print(contP)

        if contVecesBusqueda != contP:
            xarm.set_position(x_final, y_final, z_final,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
            print('Si se ha detectado pieza.')
            print('Veces que se realiza la BUSQUEDA de
piezas:{}'.format(contVecesBusqueda))

            situacion='R_PIEZA'

        else:
            print('No se han detectado piezas.')
            situacion='INICIAL'



    if situacion=='R_PIEZA':
```

```
        if PIEZA=='1':

            #Leemos los fotogramas através de la cámara
            #camera.set(cv2.CAP_PROP_BUFFERSIZE,0)
            r,frame = camera.read()
            r,frame = camera.read()
            r,frame = camera.read()

            img2 =
jetson.utils.cudaFromNumpy(cv2.cvtColor(frame,cv2.COLOR_BGR2RGBA).astype(
np.float))

            # Deteccion de la CARA de la PIEZA 1
            detect= net2.Detect(img2)
            CARA='NO'


            # Si hay una deteccion de TOP o BOTTOM
            if detect:

                for det in detect:
                    CARA='NO'
                    #Determinamos que objeto es (si es 3DC11LP o
3DC14EMR-ULP):
                    clase = det.ClassID

                    #Si se detecta la cara TOP
                    if clase == 1:
                        CARA='TOP'
                        print('TOP')
                        color =(0,0,225)
                        cv2.putText(frame,'TOP',(int(det.Left),int(det.To
p)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                        #cv2.putText(frame,'Esq2',(int(det.Right),int(det
.Bottom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                        cv2.rectangle(frame,(int(det.Left),int(det.Top)),
(int(det.Right),int(det.Bottom)),(255,0,0),1)
                        #xarm.set_servo_angle(angle=pos_final_b,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
                        situacion='ROTACION'
                        break


                    #Si se detecta la cara BOTTOM
                    elif clase == 2:
```

```
                        CARA='BOTTOM'
                        print('BOTTOM')
                        color =(0,0,225)
                        cv2.putText(frame,'BOTTOM',(int(det.Left),int(det
.Top)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                        #cv2.putText(frame,'Esq2',(int(det.Right),int(det
.Bottom)),cv2.FONT_HERSHEY_SIMPLEX,1,color,3)
                        cv2.rectangle(frame,(int(det.Left),int(det.Top)),
(int(det.Right),int(det.Bottom)),(255,0,0),1)

                        #Movemos a la pos de la pieza
                        xarm.set_position(x_final+Ix, y_final+Iy,
z_final, speed=params['speed'], mvacc=params['angle_speed'], wait=True,
radius=-1.0)
                        xarm.set_position(x_final+Ix, y_final+Iy,
alt_f1_c, speed=params['speed'], mvacc=params['angle_speed'], wait=True,
radius=-1.0)
                        xarm.set_suction_cup(True,
wait=False,delay_sec=0)
                        time.sleep(0.25)
                        xarm.set_position(x_final+Ix, y_final+Iy,
alt_f1_c+25, speed=params['speed'], mvacc=params['angle_speed'],
wait=True, radius=-1.0)

                        xarm.set_servo_angle(angle=pos_final_m,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
                        time.sleep(0.2)
                        xarm.set_suction_cup(False,
wait=True,delay_sec=0)
                        situacion='POS_FINAL'
                        break

            if CARA != 'TOP' and CARA!='BOTTOM':
                print ('NO se ha detectado correctamente la pieza.')

                #Movemos a la pos de la pieza
                xarm.set_position(x_final+Ix, y_final+Iy, z_final,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
                xarm.set_position(x_final+Ix, y_final+Iy, alt_f1_c,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
                xarm.set_suction_cup(True, wait=False,delay_sec=0)
                time.sleep(0.25)
```

```
                xarm.set_position(x_final+Ix, y_final+Iy, alt_f1_c+25,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)

                xarm.set_servo_angle(angle=pos_final_m,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
                time.sleep(0.2)
                xarm.set_suction_cup(False, wait=True,delay_sec=0)
                situacion='POS_FINAL'


        elif PIEZA=='2':
            #xarm.set_servo_angle(angle=pos_final_b,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
            situacion='ROTACION'


    if situacion=='ROTACION':

            r,frame = camera.read()
            r,frame = camera.read()
            if r == True:          # si hay imagen
                ancho = frame.shape[1] #columnas
                alto = frame.shape[0] # filas


                #----   Procesamiento imagen  ----
                frame_gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                _, binarizada = cv2.threshold(frame_gris, 25, 255,
cv2.THRESH_BINARY_INV)

                #----  Eliminacion de sombras mediante suavizado  ----
                filtro=cv2.GaussianBlur(binarizada, (11,11), 0) ##(11,11)

                #----  Eliminacion de grises mediante umbral de
binarizacion  ----
                _, umbral=cv2.threshold(filtro, 150, 255,
cv2.THRESH_BINARY)  ##50

                #----  Eliminacion del ruido  ----
                dila= cv2.dilate(umbral, np.ones((1,1))) #(3,3)

                #----  Creacion de un kernel (mascara)  ----
                kernel=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
(5,5))  #(3,3)
```

```
                #----  Aplicacion del kerner creado para unir los pixeles
dispersos  ----
                cerrar=cv2.morphologyEx(dila, cv2.MORPH_CLOSE,kernel)

                #Se establece el area minima y maxima que desea
identificarse:
                Area_min=100
                Area_max=100000
                #-----  Contornos   ----
                contornos,_=cv2.findContours(cerrar, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

                info_deteccion=[] #Lista de la informacion detecciones
                cuenta=0
                found=0

                for cont in contornos:
                    if ( (cv2.contourArea(cont)>Area_min) &
(cv2.contourArea(cont)<Area_max))==False:
                        cuenta=cuenta+1

                    else:
                        #Rectangulo:
                        x,y,w,h=cv2.boundingRect(cont)

                        #Almacenamiento info:
                        info_deteccion.append([x,y,w,h])

                        #Rectangulo con inclinacion
                        rect = cv2.minAreaRect(cont)
                        box = cv2.boxPoints(rect)
                        box = np.int0(box)

                        #Centro del contorno para areas irregulares
                        M = cv2.moments(cont)
                        if (M["m_pos00"]==0): M["m_pos00"]=1
                        x = int(M["m_pos10"]/M["m_pos00"])
                        y = int(M['m_pos01']/M['m_pos00'])

                        #Nuevo contorno (mejoria)
                        nuevoContorno = cv2.convexHull(cont)
                        print(box)

                #SEGUIDOR:
                info_id=seguimiento.rastreo(info_deteccion)
```

```
                for inf in info_id:
                    #Coordenadas
                    x, y, w, h, id= inf

                    #Se muestra el contorno por pantalla por medio de un
rectangulo:
                    cv2.rectangle(frame, (x,y),(x+w,y+h),(0,0,255),2)

                    #Se muestra el rectangulo con el angulo de
inclinacion:
                    cv2.drawContours(frame,[box],0,(0,255,255),2)

                    #Centro:
                    cx=int(x + w / 2)
                    cy=int(y + h / 2)


                    #Se llama a la funcion clasificadorPuntos:
                    box_salida, ang_deg, BASEpieza, ALTOpieza =
clasificadorPuntos(box)

                    #Se almacenan los puntos obtenidos:
                    punto_1=(int(box_salida[0][0]),
int(box_salida[0][1]))
                    punto_2=(int(box_salida[1][0]),
int(box_salida[1][1]))
                    punto_3= (int(box_salida[2][0]),
int(box_salida[2][1]))
                    punto_4=(int(box_salida[3][0]),
int(box_salida[3][1]))

                    #Distancia hasta el contorno:
                    Dif_1=cv2.pointPolygonTest(cont, punto_1, True)
                    Dif_2=cv2.pointPolygonTest(cont, punto_2, True)
                    Dif_3=cv2.pointPolygonTest(cont, punto_3, True)
                    Dif_4=cv2.pointPolygonTest(cont, punto_4, True)

                    #Obtencion lados pieza:
                    baseMayor=int(abs(Dif_1))+int(abs(Dif_2))
                    baseInferior=int(abs(Dif_3))+int(abs(Dif_4))

                    if baseMayor > baseInferior:

                        #Se escribe 'HEAD' en la "cabeza" del rectangulo
y asi tener una referencia:
```

```
                        cv2.putText(frame,'HEAD',punto_3, font, 0.7,
(250,0,250))

                        if ( (ang_deg>90) & (ang_deg<180)):
                            a=ang_deg
                        else:
                            a=ang_deg+180

                    if baseMayor < baseInferior:

                        #Se escribe 'HEAD' en la "cabeza" del rectangulo
y asi tener una referencia:
                        cv2.putText(frame,'HEAD',punto_1, font, 0.7,
(250,0,250))
                        if ( (ang_deg>90) & (ang_deg<180)):
                            a=ang_deg+180
                        else:
                            a=ang_deg


                    #Vertices rectangulo:
                    cv2.putText(frame,'1',(punto_1[0],punto_1[1]), font,
0.7, (255, 0, 0))
                    cv2.putText(frame,'2',(punto_2[0],punto_2[1]), font,
0.7, (255, 0, 0))
                    cv2.putText(frame,'3',(punto_3[0],punto_3[1]), font,
0.7, (255, 0, 0))
                    cv2.putText(frame,'4',(punto_4[0],punto_4[1]), font,
0.7, (255, 0, 0))

                    #Se muestra el angulo calculado en grados:
                    print(ang_deg)

                    situacion='MOVE'


    if situacion=='MOVE':
        if PIEZA=='1':
            #Movemos a la pos de la pieza
            xarm.set_position(x_final+Ix, y_final+Iy, z_final,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
            xarm.set_position(x_final+Ix, y_final+Iy, alt_f1_c,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
            xarm.set_suction_cup(True, wait=False,delay_sec=0)
```

```
            time.sleep(0.25)
            xarm.set_position(x_final+Ix, y_final+Iy, alt_f1_c+25,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)

            situacion='ROTACION_2'

        elif PIEZA=='2':
            xarm.set_position(x_final+Ix, y_final+Iy, z_final,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
            xarm.set_position(x_final+Ix, y_final+Iy, alt_f2_c,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)
            xarm.set_suction_cup(True, wait=False,delay_sec=0)
            time.sleep(0.25)
            xarm.set_position(x_final+Ix, y_final+Iy, alt_f2_c+25,
speed=params['speed'], mvacc=params['angle_speed'], wait=True, radius=-
1.0)

            situacion='ROTACION_2'

    if situacion == 'ROTACION_2':
        #Guardamos la posicion en la que esta el robot
        pose=xarm.get_servo_angle()
        print(pose)
        #Me quedo con el segundo elemento de la tupla
        pose=pose[1]
        #Me quedo con los 5 primeros elementos
        pose=pose[:6]
        print (pose)

        #Guardamos las posiciones
        pose0=pose[0]
        pose1=pose[1]
        pose2=pose[2]
        pose3=pose[3]
        pose4=pose[4]
        #Angulo de J6
        pose5=pose[5]
        print('Articulacion J6: {}'.format(pose5))

        if PIEZA=='1':

            j6=pose5

            if ang_deg>180:
```

```
                ang_deg=ang_deg-180

            angulo=ang_deg-90

            if angulo < 90:
                ang_f=j6-angulo

            elif angulo>90:
                ang_f=j6+angulo

            pose_f=[pose0,pose1,pose2,pose3,pose4,ang_f]
            print(pose_f)
            xarm.set_servo_angle(angle=[pose0,pose1,pose2,pose3,pose4,ang
_f], speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)

            situacion='FINAL'

        elif PIEZA=='2':
            j6=pose5

            if ang_deg>180:
                ang_deg=ang_deg-180

            angulo=(ang_deg-90)

            if angulo < 90:
                ang_f=j6-angulo

            elif angulo>90:
                ang_f=j6+angulo

            pose_f=[pose0,pose1,pose2,pose3,pose4,ang_f]
            print(pose_f)
            xarm.set_servo_angle(angle=[pose0,pose1,pose2,pose3,pose4,ang
_f], speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)

            situacion='FINAL'


    if situacion =='FINAL':

        #Posicion de almacenamiento para la pieza 3DC11LP
        if PIEZA=='1':
            x_pos_almacenar = pos_matriz_p1[contPiezas1][0]
```

```
            y_pos_almacenar = pos_matriz_p1[contPiezas1][1]

            #Se incrementa el contador de piezas 3DC11LP almacenadas en
una unidad
            contPiezas1 = contPiezas1+1

            posicionDepositar =
xarm.set_position(x_pos_almacenar+Ix_1,y_pos_almacenar+Iy_1,
alt_f1_c+40,speed=params['speed'],mvacc=params['angle_speed'], radius=50,
wait=True)

            angulos_Pre_Almacenamiento = xarm.get_servo_angle()

            pos_Pre_Almacenamiento = angulos_Pre_Almacenamiento[1]

            if(contPiezasA1==0):

                angulosPos = xarm.get_servo_angle()
                pos_Alm_1 = angulosPos[1]

            if(contPiezasA1==num_colum):

                pos_Pre_Almacenamiento=pos_Alm_1
                contPiezasA1=-1

            contPiezasA1 = contPiezasA1+1

            situacion='ALMACENAMIENTO'

        #Posicion de almacenamiento para la pieza 3DC14EMR-ULP
        elif PIEZA=='2':

            x_pos_almacenar = pos_matriz_p2[contPiezas2][0]
            y_pos_almacenar = pos_matriz_p2[contPiezas2][1]

            #Se incrementa el contador de piezas 3DC14EMR-ULP almacenadas
en una unidad
            contPiezas2 = contPiezas2+1

            posicionDepositar =
xarm.set_position(x_pos_almacenar+Ix_2,y_pos_almacenar+Iy_2,
alt_f2_c+40,speed=params['speed'],mvacc=params['angle_speed'], radius=50,
wait=True)

            #Angulos robot
            angulos_Pre_Almacenamiento = xarm.get_servo_angle()
            pos_Pre_Almacenamiento = angulos_Pre_Almacenamiento[1]
```

```
            if(contPiezasA2==0):

                angulosPos = xarm.get_servo_angle()
                pos_Alm_1 = angulosPos[1]

            if(contPiezasA2==num_colum):

                pos_Pre_Almacenamiento=pos_Alm_1
                contPiezasA2=-1

            contPiezasA2 = contPiezasA2+1

            situacion='ALMACENAMIENTO'

    if situacion=='ALMACENAMIENTO':

        #Almacenamiento de la pieza 3DC11LP
        if PIEZA =='1':

            posicionAlmacenaje=[x_pos_almacenar+Ix_1,y_pos_almacenar+Iy_1
,252]
            xarm.set_position(*posicionAlmacenaje,speed=params['speed'],m
vacc=params['angle_speed'], radius=50, wait=True)
            xarm.set_suction_cup(False, wait=True,delay_sec=0)

            situacion='POS_FINAL'

        #Almacenamiento de la pieza 3DC14EMR-ULP
        elif PIEZA=='2':

            posicionAlmacenaje=[x_pos_almacenar+Ix_2,y_pos_almacenar+Iy_2
,245]
            xarm.set_position(*posicionAlmacenaje,speed=params['speed'],m
vacc=params['angle_speed'], radius=50, wait=True)
            xarm.set_suction_cup(False, wait=True,delay_sec=0)

            situacion='POS_FINAL'

    if situacion =='POS_FINAL':
        xarm.set_servo_angle(angle=pose_final,
speed=params['angle_speed'], mvacc=params['angle_acc'], wait=True,
radius=-1.0)
        situacion='BUSQUEDA'


    # Renderizamos la imagen
```

```
        cv2.imshow("window",frame)
        if cv2.waitKey(1) == ord("q"):
            break

camera.release()
cv2.destroyAllWindows()
```