Universidad de La Laguna

Escuela Superior de Ingeniería Y Tecnología
Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO DE FIN DE GRADO

Integración de un sistema de visión On Board para un Robot Delta

Mayo 2023

Autora: Paula Patricia Arcano Bea
Tutor: Santiago Torres Álvarez

Agradecimientos:

En primer lugar, quiero agradecer a mi familia, por su incondicional apoyo y comprensión a lo largo de toda mi vida y durante el desarrollo de este proyecto. Gracias por estar siempre a mi lado y por motivarme a seguir adelante en los momentos más difíciles, su ayuda ha sido invaluable y siempre estaré agradecida por ella.

También quiero agradecer a todos mis profesores, en especial a Leopoldo Acosta Sánchez, Silvia Alayón Miranda, José Francisco Sigut Saavedra y a mi tutor Santiago Torres Álvarez, por compartir su conocimiento y experiencia, por ser una fuente de inspiración y motivación constante, brindándome siempre su apoyo y ayudándome a desarrollar mis habilidades y conocimientos para crecer profesionalmente.

Mil gracias a todos.

Índice:

Lista de figuras:

Resumen:

En este trabajo se presenta un sistema innovador que combina tecnologías como la impresión 3D, el *machine learning* y el *deep learning*, con el objetivo de automatizar la detección y manipulación de objetos en tiempo real en un entorno industrial pequeño; para esto se utilizó un robot delta existente al que se le incorporó una cámara y una pinza como efector final. El sistema se basa en un modelo personalizado de YOLOv8, entrenado específicamente para detectar objetos geométricos que ingresan al área de trabajo mediante una cinta transportadora.

Una vez detectados los objetos, se analizaron sus propiedades como forma, color, orientación y textura, para luego ser recogidos por la pinza y desplazados a otra cinta transportadora. Durante el desarrollo del proyecto se presentarán los detalles del diseño e implementación del sistema, así como los resultados obtenidos en las pruebas realizadas.

Los resultados obtenidos mostraron que el modelo de detección implementado fue efectivo y preciso, con tasas de detección y falsos positivos satisfactorias al realizar las tareas de detección y segmentación de objetos en tiempo real.

Debido a esto, se puede afirmar que este sistema puede ser una solución rentable para pequeñas fábricas industriales mejorando su eficiencia y competitividad en el mercado ya que cuenta con una alta capacidad de personalización, flexibilidad que se traduce en un ahorro significativo en los costes de implementación.

La implementación de estas tecnologías puede mejorar significativamente la eficiencia y productividad en la industria reduciendo la necesidad de intervención humana en entornos peligrosos. Se espera que esta tecnología siga evolucionando y mejorando en el futuro para proporcionar soluciones más eficientes y precisas en diferentes campos.

Abstract:

This work presents an innovative system that combines technologies such as 3D printing, machine learning, and deep learning, with the aim of automating real-time object detection and manipulation in a small industrial environment; for this, an existing delta robot was used and equipped with a camera and a gripper as its end effector. The system is based on a custom YOLOv8 model, specifically trained to detect geometric objects that enter the workspace via a conveyor belt.

Once the objects were detected, their properties such as shape, colour, orientation, and texture were analyzed and then picked up by the gripper and moved to another conveyor belt. This project details the system design and implementation, as well as the results obtained during the tests.

The results showed that the implemented detection model was effective and precise, with satisfactory detection and false-positive rates when performing real-time object detection and segmentation tasks.

As a result, it can be stated that this system can be a cost-effective solution for small industrial factories, improving their efficiency and competitiveness in the market as it has a high level of customization and flexibility, which translates into significant cost savings.

The implementation of these technologies can significantly improve efficiency and productivity in the industry, reducing the need for human intervention in dangerous environments. It is expected that this technology will continue to evolve and improve in the future to provide more efficient and precise solutions in different fields.

1.Introducción:

La automatización, la robótica y la visión por computador en la actualidad se han convertido en herramientas claves para mejorar la eficiencia y productividad de diferentes procesos industriales.

Estas tecnologías permiten una mayor precisión en la ejecución de tareas repetitivas y reducen la necesidad de intervención humana en entornos peligrosos, además de minimizar los errores humanos aumentando la velocidad y la calidad del trabajo realizado.

La detección y seguimiento en tiempo real de objetos gracias a la visión por computador resulta muy útil en la industria, ya que permite el control de calidad, la identificación de productos y la monitorización de procesos. Asimismo, el análisis de imágenes y vídeos aporta información valiosa para la optimización de procesos y la toma de decisiones. En la actualidad, la visión por computador con aplicación de redes neuronales convolucionales se ha vuelto más accesible y fácil permitiendo la detección y segmentación de objetos en tiempo real, en imágenes y en vídeos.

En este trabajo se utilizan distintas tecnologías, como la impresión 3D, el *machine learning* y el *deep learning* aplicadas en un robot delta ya existente en el cual se ha colocado una cámara en su extremo y una pinza como efector final, con la finalidad de automatizar la detección y manipulación de objetos en tiempo real, a través de un modelo personalizado de YOLOv8 entrenado específicamente para detectar objetos geométricos que van pasando por una cinta transportadora.

Una vez detectados, se analizarán distintas propiedades de dichos objetos, como la forma, el color, la orientación y la textura, para luego ser recogidos con la pinza y desplazados a otra cinta transportadora.

Durante el desarrollo de la memoria, presentaré los detalles del diseño e implementación del sistema, así como los resultados obtenidos en las distintas pruebas realizadas. También haré referencia a sus posibles aplicaciones y limitaciones del sistema y por último se darán unas conclusiones sobre su funcionamiento y eficiencia en la identificación y manipulación de objetos en tiempo real.

1.1.Objetivo, alcance e importancia

El principal objetivo de este proyecto es diseñar e implementar un sistema de visión y manipulación de objetos en tiempo real que permita la detección, análisis y recogida de objetos que van entrando en una cinta transportadora.Se utilizará la red neuronal convolucional YOLOv8 para crear un modelo personalizado que permita la detección y segmentación de objetos geométricos. Además se realizará el diseño e impresión 3D de una pinza que será colocada en el extremo de un robot delta para ser utilizada como efector final; de la misma manera se le integrará al extremo del robot una cámara que permita visualizar el entorno de trabajo en tiempo real.

La importancia de este trabajo radica en la necesidad de automatizar y mejorar los procesos de manipulación de objetos en pequeñas industrias que requieran la selección y desplazamiento de objetos de manera autónoma y en ambientes controlados, todo esto con un coste reducido.

Por otra parte, el uso de materiales impresos en 3D permite una mayor personalización y flexibilidad en el diseño de herramientas y dispositivos, lo que puede mejorar la adaptabilidad y versatilidad de los sistemas automatizados.

1.2.Estructura de la memoria

En el primer capítulo se presenta la introducción en la que se establece el objetivo, alcance e importancia del trabajo y se describe la estructura de la memoria. Después de la introducción, se encuentra el capítulo de fundamentos teóricos, en el cual se explican teóricamente conceptos claves que serán de utilidad para entender lo desarrollado en este trabajo. Dentro de los fundamentos teóricos tenemos la descripción de los robots delta, de las impresoras 3D, de la visión por ordenador, de las redes neuronales, del modelo YOLOv8 y de los espacios de colores HSV y LAB.

En el tercer capítulo trata del diseño e implementación, en el que se describe el proceso de creación e integración de las piezas impresas en 3D al robot delta existente. También se habla de los componentes electrónicos y se describen las herramientas de Python utilizadas para el procesamiento de imágenes y video en tiempo real. En el cuarto capítulo se presentan los resultados obtenidos en la implementación del modelo y se discuten posibles aplicaciones, mejoras y limitaciones actuales.

Finalmente, en el quinto y sexto capítulo se presentan las conclusiones generales y la bibliografía utilizada para finalizar con los anexos en el séptimo capítulo

2.Fundamentos Teóricos

2.1.Robot Delta

Un robot delta (Figura 1) es un tipo de robot que se caracteriza por tener un diseño en forma de delta, con tres brazos articulados que se unen en un efector final en el extremo; son uno de los principales tipos de robots industriales también llamados robots araña o paralelos debido a su construcción [1]. Estos robots son conocidos por su rapidez y eficiencia, por lo que son ideales para aplicaciones con alta producción y actividades de ensamblaje, como por ejemplo para el *pick and place* de cintas transportadoras a cajas para empaquetado.

Estos robots paralelos pueden llegar a disponer de hasta cinco grados de libertad y que en función del modelo tiene hasta 6 ejes y son capaces de llegar a hacer hasta 300 movimientos por minuto[2].

Figura 1: Ejemplos de Robots Delta

2.1.1.Origen y características

A partir de la década de 1980 se empezaron a proponer cientos de diseños de nuevos robots paralelos para diferentes aplicaciones entre las que destacó, a mediados de la década, el diseño del profesor Reymond Clavel de la École

Polytechnique Fédérale de Lausanne en Suiza, quien consolidó la idea de utilizar paralelogramos para construir una estructura robusta con tres grados de libertad de traslación y un movimiento rotatorio y enfocado en aplicaciones de *pick and place* a alta velocidad. A esta estructura se le dio el nombre de robot Delta[3].

Actualmente, los robots delta son una opción popular en la industria debido a su alta velocidad, precisión y capacidad para manejar cargas ligeras a medianas en entornos de producción automatizados. Estos robots se utilizan en una variedad de aplicaciones industriales, como la fabricación de productos electrónicos, alimentos y bebidas, así como en la industria farmacéutica y automotriz.

Los robots Delta se caracterizan por tener varios brazos móviles conectados a una estructura central, formando una configuración en forma de paralelogramo. Los brazos se mueven de manera sincronizada para proporcionar una amplia gama de movimientos en 3 dimensiones.

Este tipo de construcción ofrece la seguridad de realizar trabajos repetitivos y posicionados con alta precisión. Contienen tecnologías avanzadas como la Visión Artificial y software manipulado por *deep learning*. En la actualidad, los fabricantes han desarrollado softwares que permiten configuraciones precisas para que puedan interaccionar hasta 8 robots en una misma línea de producción; la gran mayoría de ellos disponen de una categoría de protección IP65 e IP67 que les permite trabajar en el sector alimentario (Figura 2) [4].

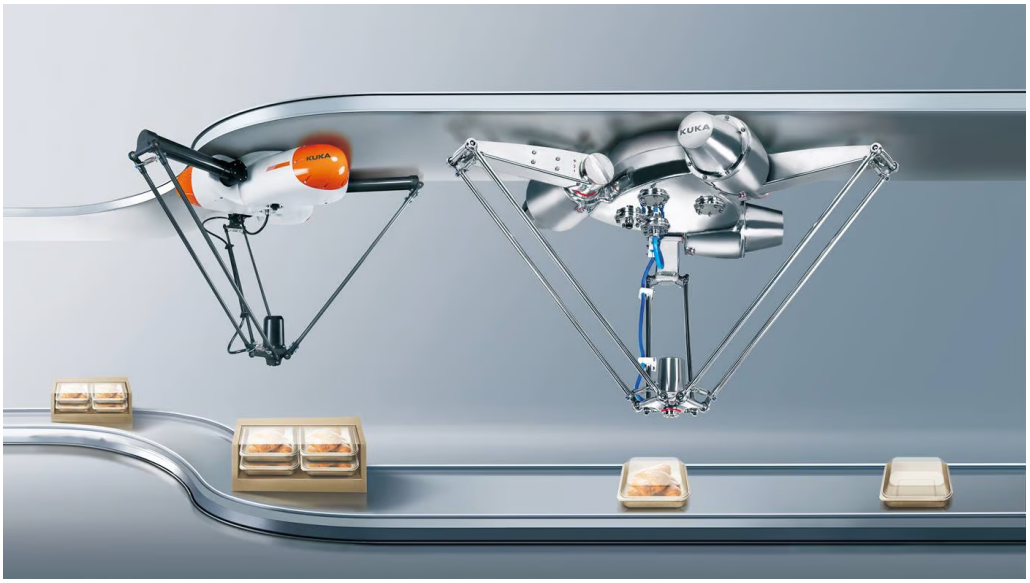


Figura 2: Ejemplos de uso de Robots Delta en la industria alimentaria

2.1.2.Componentes y funcionamiento

Los robots Delta están diseñados por una estructura compuesta por varias partes. El cuerpo principal o base que incorpora todas las articulaciones principales, los motores, los reductores planetarios, el equipo electromecánico y la electrónica.

Los brazos mecánicos que actúan como bielas son cadenas cinemáticas y se encargan de desplazar el efector final sobre la superficie [1].En la parte interna suelen ir integrados los tubos neumáticos para controlar el eje z de los desplazamientos.

Los efectores finales son los dispositivos que se acoplan al extremo del brazo del robot para realizar tareas específicas, estos pueden ser tanto actuadores rotacionales como lineales, en función del uso que se le vaya a dar . En el caso de los robots delta utilizados en industrias, estos suelen ser pinzas o herramientas para la manipulación de objetos.

2.1.3.Ventajas

Los robots Delta alcanzan tasas de productividad mucho más altas por su mayor velocidad y rendimiento garantizando la productividad en los procesos.

Pueden procesar hasta 3 objetos por segundo o 200 objetos por minuto, lo que aumenta significativamente el rendimiento e incrementa la competitividad en las empresas . Estos permiten realizar control de calidad con alta precisión además de requerir un mantenimiento mínimo.

Por otro lado, estos robots contribuyen a reducir la insatisfacción laboral porque evitan lesiones por estrés y fatiga a los operarios humanos al tener la capacidad de realizar procesos a altas velocidades y gran repetibilidad.

Se suelen colocar en el techo, lo que los hace ideales para ahorrar espacio y sumarse a una celda de trabajo robótica ya integrada y así reducir el tamaño de las líneas de producción.

Los robots Delta están diseñados, especialmente, para aplicaciones de baja carga útil tales como actividades de ensamblado o *pick and place*. Por lo mismo, son

complementos perfectos para la industria alimentaria, electrónica, cosmética y farmacéutica [3].

Además, con el paso del tiempo, las mejoras realizadas a los robots Delta hacen que estos ya no se limiten solo a tareas de embalaje, ensamblaje o desplazamiento, sino que sean empleados para aplicaciones de ultraprecisión (en escala de nanómetros), cirugía robótica remota, relojería y mecanizado de precisión; asimismo, pueden desempeñarse para simuladores de vuelo, manipuladores de grandes cargas y posicionamiento de antenas.

2.2.Impresión 3D

La impresión 3D es una tecnología de fabricación aditiva con la cual se pueden crear objetos tridimensionales partiendo de un modelo digital [5]. La impresión 3D construye un objeto capa por capa, a partir de un material base que se va depositando de manera controlada.

Para imprimir un objeto en 3D se necesita un modelo en digital del mismo, este modelo puede ser creado mediante el escaneo y digitalización de un objeto existente o diseñado utilizando herramientas de software para modelado en 3D.

2.2.1. Impresoras y materiales

En la actualidad, existen múltiples tipos de impresoras 3D como por ejemplo las impresoras SLS (*Selective Laser Sintering*) que utilizan un láser para fusionar y solidificar un polvo termoplástico, las SLA (*Stereolithography Apparatus*) que utilizan un proceso de fotopolimerización, las DLP (*Digital Light Processing*) (Figura 3) que utilizan una fuente de luz UV para solidificar resinas fotosensibles y las FDM (*Fused Deposition Modeling*) (Figura 3) siendo estas las más populares y asequibles en el mercado. Funcionan extruyendo un filamento termoplástico que se derrite y se deposita capa por capa para crear un objeto 3D [6].

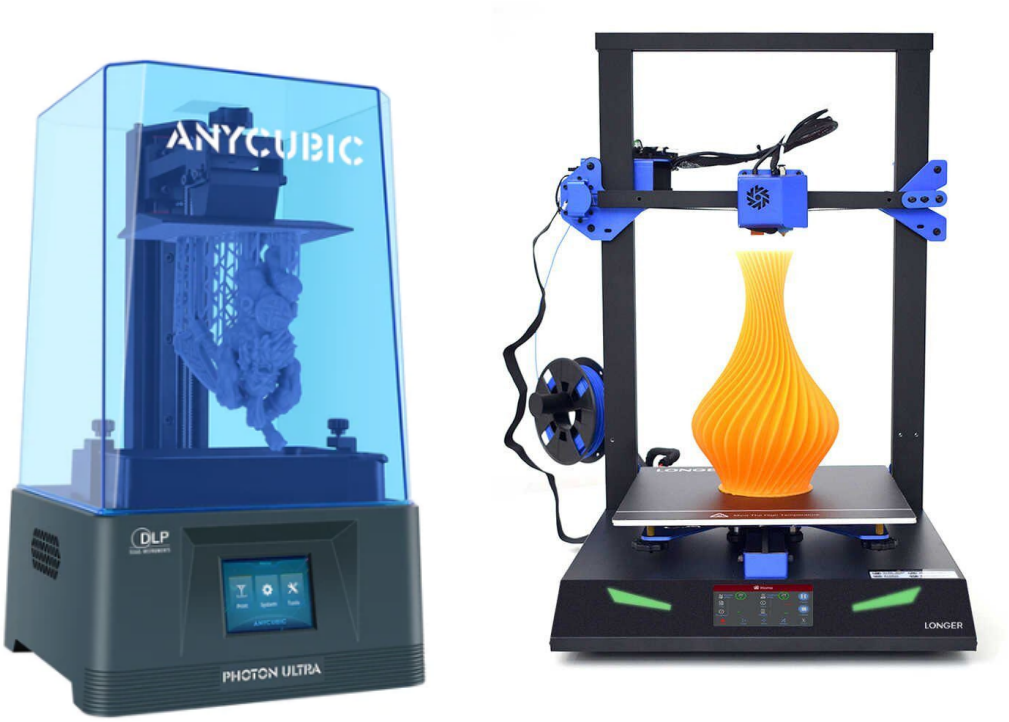


Figura 3: Impresoras 3D de tipo DLP y FDM

Existen diversos materiales que pueden ser utilizados para la impresión 3D, entre los más comunes están: el ABS (acrilonitrilo butadieno estireno) (Figura 4) que es un material fuerte y duradero adecuado para la fabricación de piezas que necesiten resistencia al impacto, el PETG (tereftalato de polietileno glicol) que es un material resistente pero no excesivamente rígido con buena resistencia térmica y mecánica, el Nylon (Figura 4) que es un material resistente y flexible y por último el PLA (ácido poliláctico) (Figura 4) que es el material más utilizado debido a la simplicidad en su uso y configuración además de ser un material de origen vegetal biodegradable que es respetuoso con el medio ambiente.

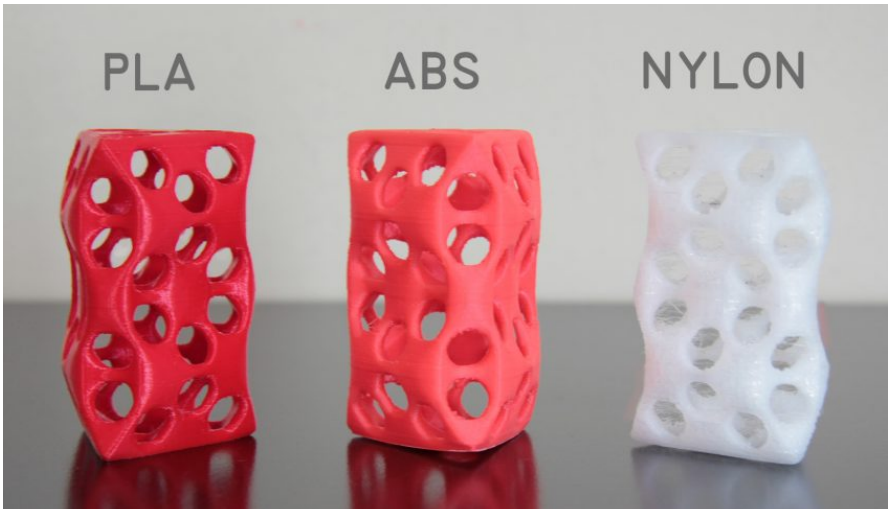


Figura 4: Ejemplo de objetos impresos en tres materiales distintos

2.2.2. Software de impresión y modelado 3D

Entre los software disponibles para modelado 3D, uno de los más comunes es el Fusion 360 de AutoDesk el cual es un es un software de diseño asistido por computadora (CAD) y fabricación asistida por computadora (CAM). Este software ofrece una amplia gama de herramientas que permiten crear modelos complejos y precisos para su posterior fabricación en una variedad de procesos, incluyendo la impresión 3D [7]. Este software también cuenta con herramientas de simulación y análisis, renderizado y colaboración en línea, lo que lo convierte en una solución completa para diseñadores, ingenieros y fabricantes. Otros softwares utilizados para el modelado 3D son TinkerCAD [18], 3D Slash [19], XYZmaker [20], Art of Illusion [21] y K-3D [22].

Uno de los softwares de código abierto más utilizados para la impresión 3D es Cura, de la empresa Ultimaker. Este software permite que los usuarios puedan importar modelos 3D, ajustar su tamaño y orientación, agregar soportes para áreas complicadas y generar código G para enviar a la impresora 3D [8]. Además, Cura también cuenta con características avanzadas, como la capacidad de personalizar la configuración de la impresora, ajustar la velocidad y la temperatura de impresión, y utilizar diferentes perfiles de material para obtener resultados óptimos. Otros software muy utilizados para el laminado en la impresión 3D son PrusaSlicer [23], IdeaMaker [24], IceSL [25] y Kiri:Moto [26].

2.3.Visión por ordenador

La visión por ordenador o computadora es una rama de la inteligencia artificial que se encarga de interpretar la información visual recibida mediante cámaras y otros dispositivos dándoles la capacidad de percibir el mundo que las rodea [16]. En la actualidad, la visión por ordenador se utiliza en numerosas aplicaciones, como en la robótica, la seguridad, la conducción autónoma de vehículos, la medicina, entre otros.

La visión por ordenador permite obtener datos relevantes a partir de imágenes y videos, y realizar diversas tareas tales como la identificación, rastreo y reconocimiento de objetos, la medición de propiedades físicas y geométricas, clasificación de elementos en diferentes categorías, análisis de patrones de movimiento y conducta, estimación de profundidad y creación de modelos 3D, entre

otros usos. La extracción de esta información se logra mediante la aplicación de algoritmos y técnicas de análisis de imágenes y visión por ordenador, que permiten identificar y analizar patrones y características específicas de las imágenes y videos.

La detección de objetos es una técnica muy importante en la visión por ordenador que consiste en identificar la presencia y la posición de objetos; uno de los principales objetivos que tiene esta técnica es la detección, identificación y seguimiento de los objetos en vídeos, imágenes y en tiempo real.

Para la detección de objetos se pueden utilizar diversas técnicas, como el procesamiento de imágenes, las redes neuronales y el aprendizaje automático [17].

2.4 Redes neuronales

Entre las técnicas más utilizadas en la detección de objetos se encuentran las redes neuronales, las cuales son un tipo de algoritmo de aprendizaje automático que se inspira en el funcionamiento del cerebro humano[9].

Consisten en una colección de nodos interconectados que se utilizan para procesar información, clasificar datos y predecir resultados.

A estos nodos interconectados se les suele llamar neuronas, en las cuales cada neurona recibe información de otras neuronas a través de conexiones (Figura 5) que tienen un peso asociado que representa la fuerza de la influencia de una neurona en la otra.

El entrenamiento de una red neuronal consiste en ajustar dichos pesos de conexión entre nodos para producir una salida deseada para un conjunto de entradas dados, permitiendo así mejorar la precisión del modelo en la tarea que se está realizando ya sea clasificación, predicción, regresión, segmentación de objetos, reconocimiento de patrones, entre otros.

Las redes neuronales pueden ser muy elaboradas, con múltiples capas de neuronas interconectadas y algoritmos de optimización para ajustar los pesos de manera eficiente, permitiéndoles aprender patrones complejos en los datos pudiendo ser utilizadas en una amplia variedad de aplicaciones de aprendizaje automático

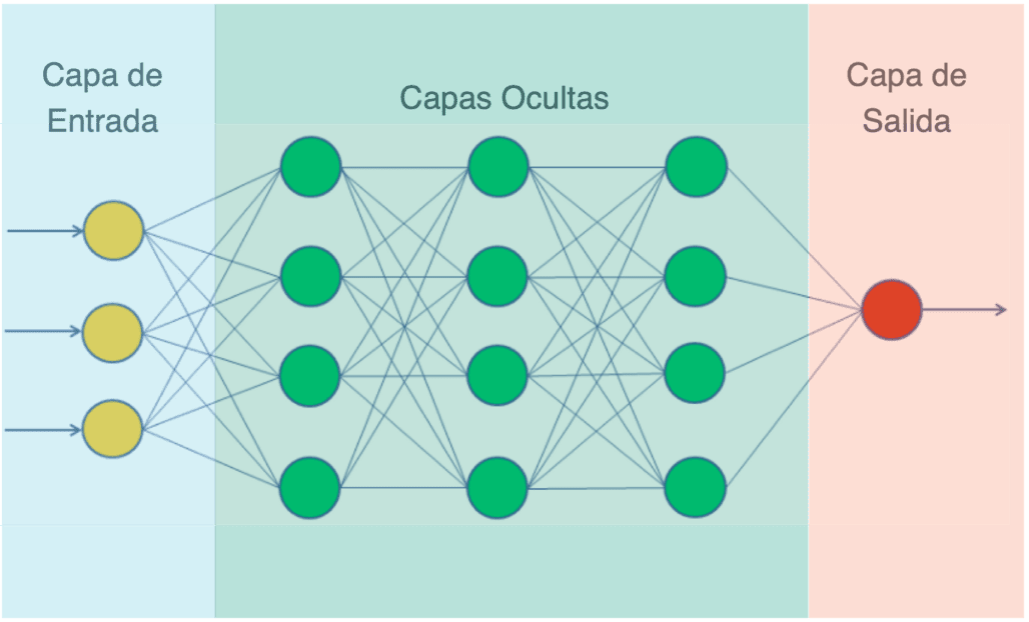


Figura 5: Ejemplo de funcionamiento de una red neuronal

2.5.Redes neuronales convolucionales

Las redes neuronales convolucionales, también conocidas como CNN por sus siglas en inglés (*Convolutional Neural Networks*) son un tipo de red neuronal profunda que se utiliza principalmente en el procesamiento de imágenes y señales de audio. Estas redes están diseñadas específicamente para procesar datos que tienen una estructura de matriz.

A diferencia de las redes neuronales generales, las CNN utilizan una operación matemática llamada convolución en la cual la información se propaga a través de capas convolucionales, en lugar de una multiplicación matricial estándar en la cual se propaga hacia adelante de información a través de múltiples capas. Esto permite que la CNN capture características específicas de las imágenes crudas (sin necesidad de pre-procesarlas), como bordes y formas, de una manera más eficiente.

Estas redes aplican filtros convolucionales (Figura 6) y capas de pooling para reducir la dimensionalidad de los datos sin perder información importante, para aprender las características específicas en las imágenes, lo que les permite ser muy efectivas en tareas de detección y clasificación de objetos.

Por otro lado, en el área de procesamiento de imágenes, las CNN se caracterizan por tener capas convolucionales que realizan la extracción de

características de las imágenes y capas completamente conectadas que realizan la clasificación de los objetos.

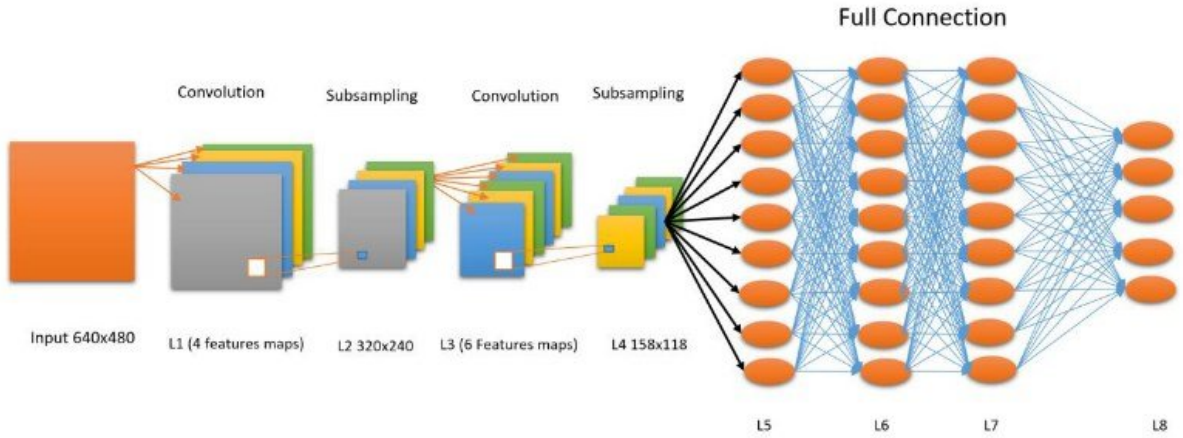


Figura 6: Ejemplo de funcionamiento de una red neuronal convolucional

2.6. YOLO

YOLO (*You Only Look Once*) es un algoritmo de detección de objetos que utiliza una red neuronal convolucional para detectar objetos y clasificarlos en diferentes categorías partiendo de imágenes o vídeos, este algoritmo fue desarrollado por Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi en 2015 en la Universidad de Washington y Facebook AI Research[13].

La primera versión de YOLO fue lanzada en 2016 y fue rápidamente adoptada por la comunidad de visión por computadora debido a su velocidad y precisión en la detección de objetos.

El algoritmo de detección de objetos se basa en una sola pasada a través de una red neuronal convolucional, en lugar de hacer múltiples pasadas y combinar resultados como en otros enfoques, lo que significa que el algoritmo puede ser capaz de detectar objetos de forma rápida, eficiente y con un alto grado de precisión, permitiendo también su aplicación en tiempo real.

2.6.1 YOLOv8

A lo largo de los años YOLO ha tenido varias actualizaciones y versiones, cada una con el objetivo de mejorar la precisión y la velocidad.

La última versión del modelo de detección de objetos e imagen en tiempo real YOLO es YOLOv8 que ha sido desarrollado y lanzado por Ultralytics el 10 de enero de 2023. Esta es una empresa que está dedicada a crear modelos de inteligencia artificial en código abierto y que ofrecen soluciones de vanguardia para una amplia gama de tareas de IA, incluyendo detección, segmentación, clasificación, seguimiento y estimación de pose [10].

Construido sobre los avances más recientes en aprendizaje profundo y visión por computadora, YOLOv8 ofrece un rendimiento excepcional en términos de velocidad y precisión [11]. Su diseño optimizado permite su adaptabilidad a diversas aplicaciones y plataformas de hardware, desde dispositivos de borde hasta APIs en la nube. Cabe destacar que, a diferencia de versiones anteriores, YOLOv8 proporciona soporte para la segmentación de instancias y mejora la velocidad de aplicación así como aumenta la complejidad del mismo, ampliando así su capacidad para identificar y separar objetos individuales en una imagen.

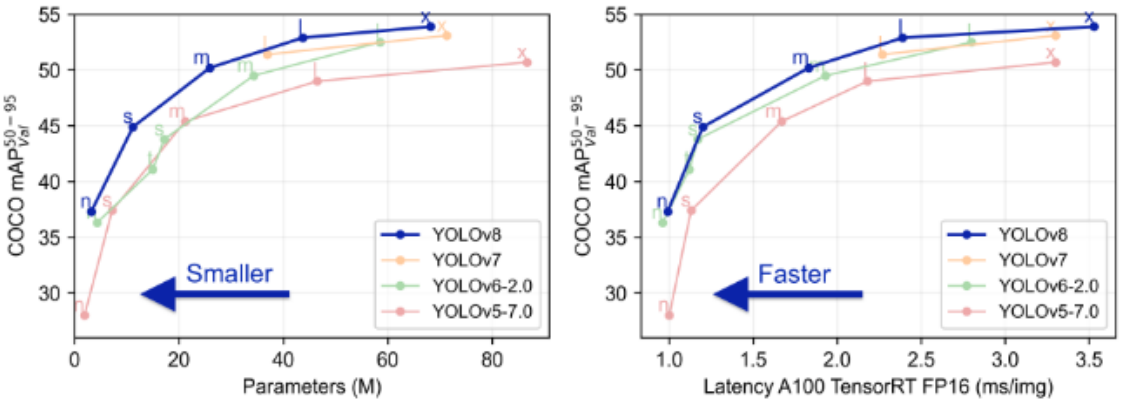


Figura 7: Gráficas de rendimiento de las diferente versiones de YOLO

Como podemos observar en la figura 7, YOLOv8 a diferencia de sus predecesores aumenta en gran magnitud su velocidad de aplicación y la complejidad del modelo disminuyendo las dimensiones del mismo.

2.6.1.1.Modelo Personalizado de YOLOv8

Un modelo personalizado en YOLOv8 se refiere a un modelo de detección de objetos entrenado específicamente con un conjunto de datos y objetos particulares, es decir, se utiliza el algoritmo YOLOv8 como base, pero se entrena con un conjunto de datos personalizados y se ajusta para detectar objetos específicos.

El uso de un modelo personalizado en la detección de objetos permite lograr una mayor precisión en comparación con un modelo genérico. Esto se debe a que el modelo personalizado está específicamente diseñado para detectar objetos en las condiciones y contextos particulares de la tarea. Además, un modelo personalizado puede ser adaptado según las necesidades y requisitos específicos de un proyecto, lo que lo hace más flexible y adaptable que los modelos genéricos.

Para generar un modelo personalizado, es necesario seguir tres pasos fundamentales (Figura 8). En primer lugar, se debe generar un conjunto de datos, también conocido como dataset, el cual consiste en almacenar un conjunto de imágenes que contengan los objetos que se desean detectar. En segundo lugar, es necesario etiquetar cada imagen, es decir, señalar de manera específica la ubicación del objeto dentro de la imagen. Este proceso de etiquetado es esencial para el tercer paso que será el entrenamiento del modelo, ya que le permite reconocer patrones y características del objeto en cuestión.

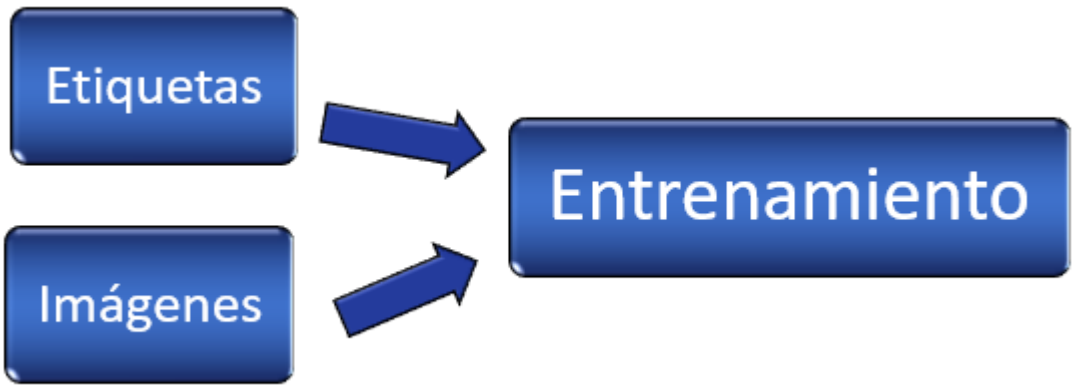


Figura 8: Esquema de elementos necesarios para el entrenamiento del modelo personalizado de YOLOv8

2.6.1.2. Dataset y entrenamiento del modelo

Actualmente existen varias plataformas que permiten el etiquetado de objetos en imágenes; una de las más utilizadas es RoboFlow, que incluso ofrece una opción gratuita. En esta plataforma, se puede cargar el conjunto de imágenes y, de forma manual, asignar etiquetas a los objetos[12]. Además, se puede expandir el tamaño del dataset ya que permite realizar sobre las imágenes múltiples variaciones como por ejemplo el ajuste de brillo, rotación o zoom. Además en ella se puede modificar los porcentajes de imágenes que se utilizaran para el entrenamiento, validación y prueba del modelo *(train, val, test*).

Por otro lado, esta plataforma también permite exportar el dataset en función del modelo que se utilizará para el entrenamiento.

En el caso de YOLOv8, RoboFlow proporciona un archivo comprimido en el cual se encuentran las carpetas de entrenamiento, validación y prueba (train, val, test) con sus respectivas imágenes y etiquetas, así como el archivo .yaml que contiene la ruta de acceso a los elementos de las carpetas y la lista de las diferentes clases de objetos a identificar.

Para realizar el entrenamiento del modelo, Ultralytics nos da la opción de realizarlo en diferentes entornos, el más utilizado de estos es Google Colab, en el cual tenemos acceso a un cuaderno que nos permite realizar el entrenamiento del modelo paso a paso.

2.6.1.3. Detección y segmentación de objetos

Una de las grandes ventajas de YOLOv8 es el permitir crear un modelo personalizado, de forma simple y sencilla, que sea capaz de detectar, hacer seguimiento y segmentar objetos desde una imagen, un vídeo o hasta en tiempo real (Figura 9).

Además de realizar todas estas tareas también nos permite extraer una gran cantidad de características de las detecciones realizadas. Por ejemplo, podemos obtener información acerca de la localización del centroide, el tamaño y localización del *bounding box*, la clase a la que pertenece el objeto, el valor de confianza que se tiene en la asignación de dicha clase y la máscara del objeto.

Todo esto nos proporciona una gran cantidad de información detallada sobre los objetos detectados, lo que a su vez nos permite realizar análisis más avanzados y precisos sobre las imágenes y los objetos en ellas presentes.

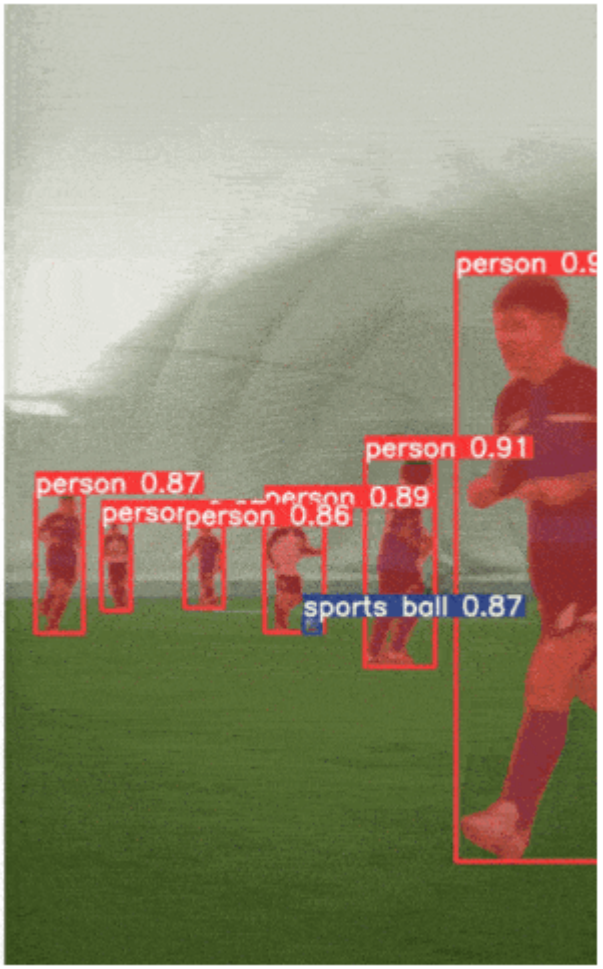


Figura 9: Ejemplo de aplicación de un modelo pre entrenado de YOLOv8 para la detección y segmentación de objetos

2.7. Color en la visión por ordenador

El color es un aspecto fundamental en la visión por computadora, ya que proporciona información valiosa para la detección, seguimiento y reconocimiento de objetos. En general, el color se define por la longitud de onda de la luz que se refleja en los objetos y es captada por los sensores de la cámara. En la visión por ordenador, se utilizan espacios o modelos de color para representar la información cromática de una imagen digital. El espacio de color mas utilizado es el RGB (rojo, verde y azul), el cual descompone el color en tres canales diferentes, uno para cada

uno de los colores primarios. Cada canal tiene una intensidad de 0 a 255, donde 0 representa la ausencia del color y 255 la máxima intensidad; la combinación de estos tres canales en diferentes proporciones permite obtener una amplia variedad de colores. En las diferentes técnicas de procesamiento de imágenes se suelen utilizar, además del espacio de color RGB, el espacio HSV, CMYK, LAB, HSL y el espacio YCbCr.

2.7.1. HSV

El espacio de color HSV se basa en tres valores: matiz o tono (*hue*), saturación (*saturation*) y valor (*value*)[14] (Figura 10). El matiz describe el color en sí mismo, la saturación se refiere a la pureza o intensidad del color, y el valor indica el brillo o la claridad del color. El valor de matiz que representa el tono del color se mide en grados en un círculo, variando de 0 a 360. La saturación se refiere a la intensidad del color se mide como un porcentaje, variando de 0 a 100%. El valor es la luminosidad del color se mide también como un porcentaje, variando de 0 a 100%. Una de las ventajas que tiene este modelo es el ser menos sensible a la variación de iluminación en la imagen en comparación con otros modelos de color como RGB o CMYK, lo que lo hace ideal para aplicaciones en condiciones de iluminación variables.

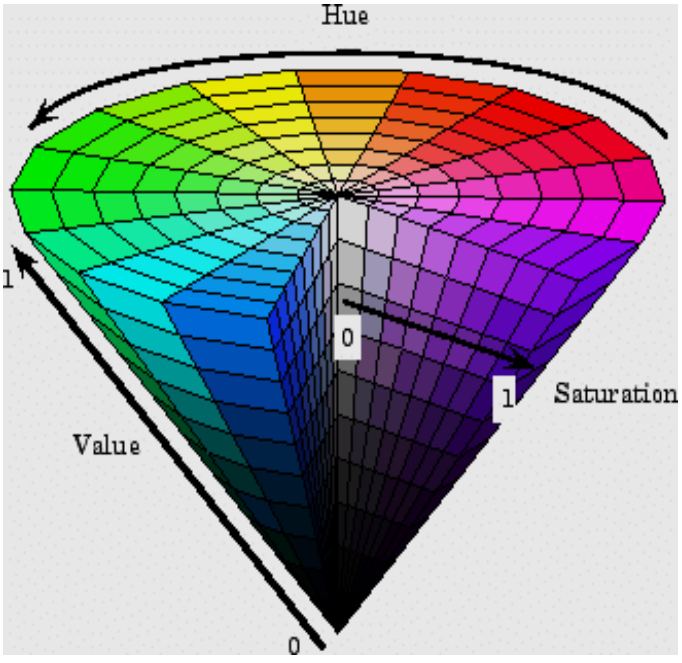


Figura 10: Espacio de color HSV

2.7.2. LAB

El espacio de color LAB se basa en la percepción humana del color en lugar de la estructura física del color como ocurre en otros modelos de color, como el RGB. Este se utiliza para separar la información de luminancia de la información de color.

En este espacio el color se divide en tres componentes: L (luminosidad), A (componente verde-rojo) y B (componente azul-amarillo)[15].

La componente L representa la la luminancia o brillo de la imagen en una escala de 0 a 100, mientras que las componentes A y B representan las variaciones de los colores rojo/verde en escala de -500 a 500 y azul/amarillo en una escala de -200 a 200 (Figura 11).

La principal ventaja de este espacio de color es que nos permite separar la información de luminosidad y color de manera independiente, lo que hace que sea más fácil trabajar con los colores de una imagen.

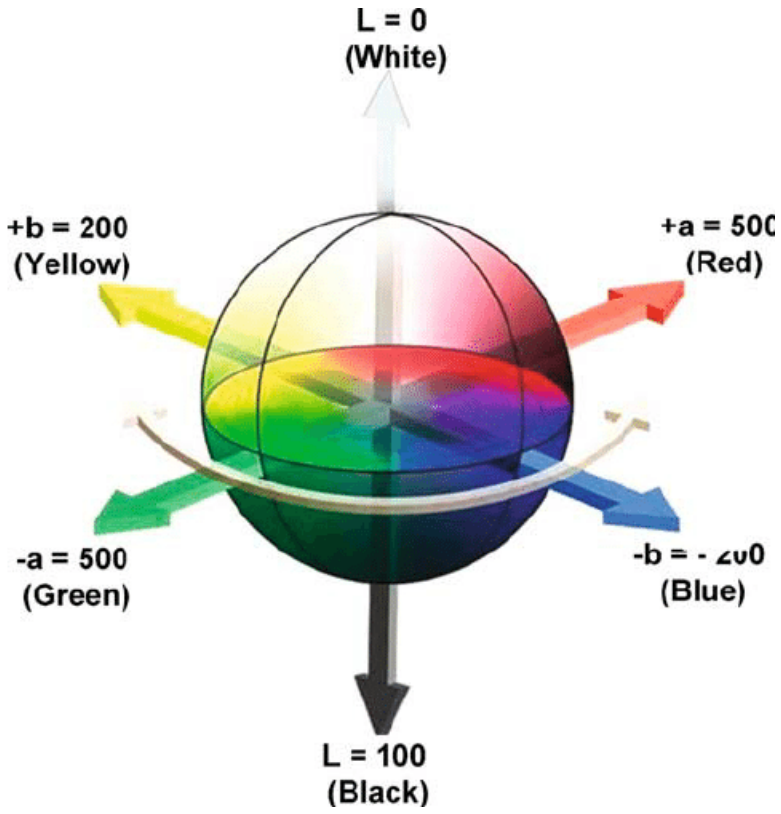


Figura 11: Espacio de color LAB

2.8.Texturas en la visión por ordenador

En la visión por ordenador, la textura se refiere a la apariencia superficial de una imagen, es decir, cómo se ven los detalles finos de la superficie de los objetos en la imagen; también puede hacer referencia a la distribución de los colores o tonalidades en la misma. Las texturas se pueden clasificar como suaves o rugosas, uniformes o irregulares, puede variar en escalas y en orientaciones y se utilizan para identificar y distinguir objetos o superficies en una imagen.

Para identificar y clasificar la textura en una imagen se pueden utilizar diversas técnicas y algoritmos, uno de los métodos más utilizados es el cálculo de la matriz de coocurrencia de niveles de gris GLCM (*gray-level co-occurrence matrix*, por sus siglas en inglés) calculada a partir de la imagen original. Esta matriz mide la frecuencia de pares de niveles de gris que ocurren en la imagen utilizando dos parámetros: la dirección y la distancia entre píxeles. A partir de la matriz, se pueden extraer varias características estadísticas, como el contraste, la disimilaridad, la homogeneidad y la energía, que se utilizan para caracterizar la textura.

La librería de Python *scikit-image* proporciona una función llamada *graycomatrix* que permite calcular la GLCM de una imagen dada. A partir de la GLCM, se pueden calcular las propiedades estadísticas de la textura utilizando otra función llamada *greycoprops*; siendo estas propiedades las que nos permitirán realizar la diferenciación entre texturas.

3.Diseño e implementación

3.1.Descripción del robot delta existente

El robot delta existente (Figura 12) cuenta con la mayoría de sus componentes impresos en 3D, para el soporte del mismo, han diseñado una estructura de madera a la cual se atornilla el robot (bases de las poleas de los bíceps y anclaje de motores). Los elementos que forman la estructura son: anclajes de motores y bíceps, base del efector final, antebrazos, partes derecha e izquierda del bíceps, poleas, guardas de polea, separadores y diversos soportes para los finales de carrera. Además de estos elementos, se han utilizado tornillos, tuercas, planchas y listones de madera, y juntas de rótula M3.

Cada uno de los tres brazos del robot está compuesto por un motor nema 17 con su respectivo anclaje a la base, el cual tiene en su eje acoplado una polea de 16 dientes donde se coloca una correa que conecta el motor con la polea que controla el bíceps; el antebrazo va conectado al bíceps y a la base del efector final mediante juntas de rótula.

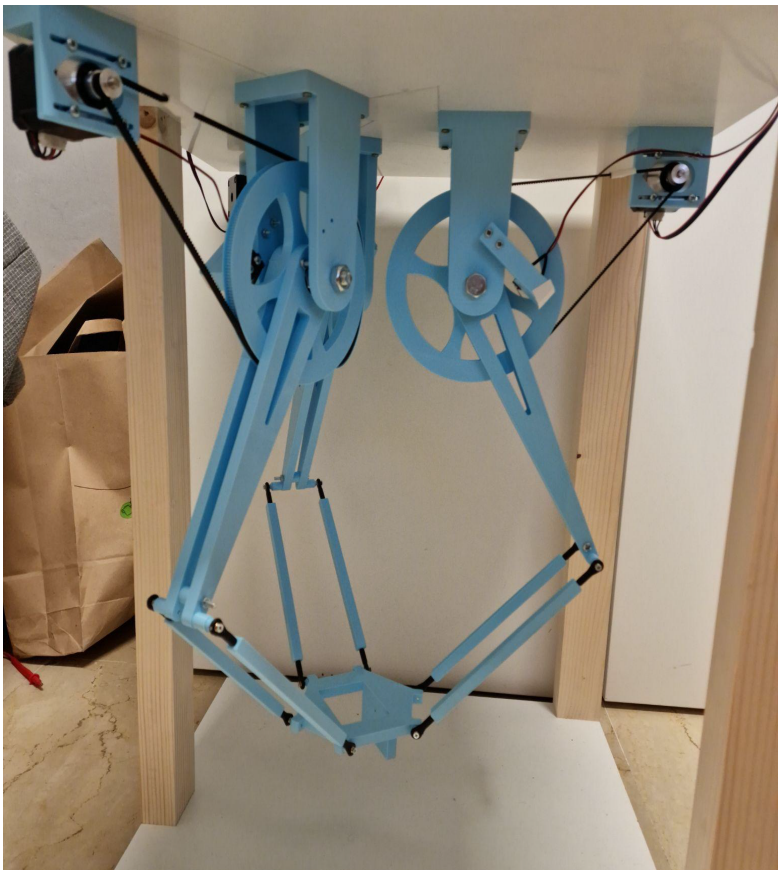


Figura 12:Robot delta existente impreso en 3D

3.2. Diseño, construcción e integración de piezas en 3D

3.2.1.Impresora, material y softwares

Para la impresión de los componentes en 3D incorporados durante la realización de este trabajo se utilizó una impresora de tipo FDM de la marca Artillery modelo Genius Pro (Figura 13)[27].

La Genius Pro cuenta con un driver de control extremadamente silencioso, lo que garantiza un ambiente de trabajo cómodo y tranquilo.

Además, tiene un doble eje Z sincronizado, lo que asegura que la altura de la impresión sea uniforme en toda la superficie ya que mejora la estabilidad y precisión en los movimientos.

El manejo de la Genius Pro es muy intuitivo gracias a su pantalla táctil. Su tamaño máximo de impresión es de 220x220x250mm y su velocidad máxima es de hasta 150mm/s.

El calentamiento de la cama de cristal cerámico es rápido y eficiente, lo que reduce el tiempo de espera antes de comenzar con la impresión; por otro lado su eficiente consumo energético de solo 600W proporciona una opción más sostenible y económica

En términos de seguridad, cuenta con medidas de protección contra cortes de luz y rotura de filamento, lo que garantiza la continuidad de la impresión en caso de imprevistos. Además, posee extrusión de tipo titan con extrusión directa la cual permite un mejor control del flujo de filamento, asegurando una alta precisión en la impresión de hasta 50 micras.

Por otro lado, cabe destacar que la impresora cuenta con una boquilla de 0.4mm de diámetro la cual logra calentar hasta 240ºC en menos de tres minutos, pudiendo utilizar en esta filamentos de hasta 1.75mm de diámetro.

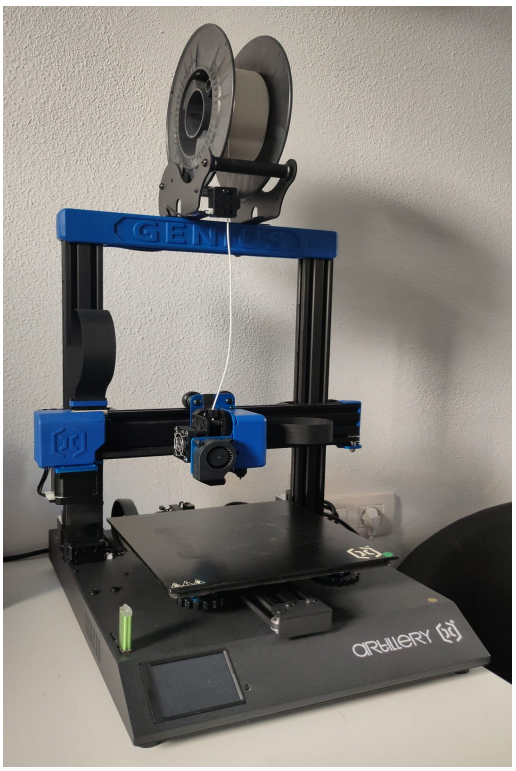


Figura 13: Impresora Genius Pro utilizada

El material elegido para realizar la impresión fue el PLA (ácido poliláctico) ya que es un material de origen vegetal biodegradable que puede ser ajustado de forma simple y sencilla a cualquier impresora 3D además de tener un bajo coste, olor ligero, buena adherencia a la cama de impresión y resistencia media. En este trabajo se utilizó específicamente el filamento PLA Go&Print blanco de la empresa Española Sakata 3D (Figura 14).

Figura 14: Filamento PLA Sakata 3D blanco utilizado

Con respecto al software para el diseño de piezas se utilizó el Fusion 360 de Autodesk en su versión 2.0.16009. Por otro lado, el software utilizado para realizar las configuraciones de la impresión fue el UltiMaker Cura 3D en su versión 5.2.2.

Para la correcta impresión de las piezas se ajustaron distintos parámetros (Figuras 15 y 16), los cuales fueron:

- Altura de capa: 0.2mm
-Ancho de muro: 0.6mm
-Ancho de capas superiores e inferiores: 0.6mm
-Relleno: 20% (Giroide)
-Temperatura de impresión: 195 °C
-Temperatura de la placa de impresión: 60 °C
-Velocidad de impresión: 50 mm/s
-Soportes de tipo normal o árbol en algunas piezas según necesidad

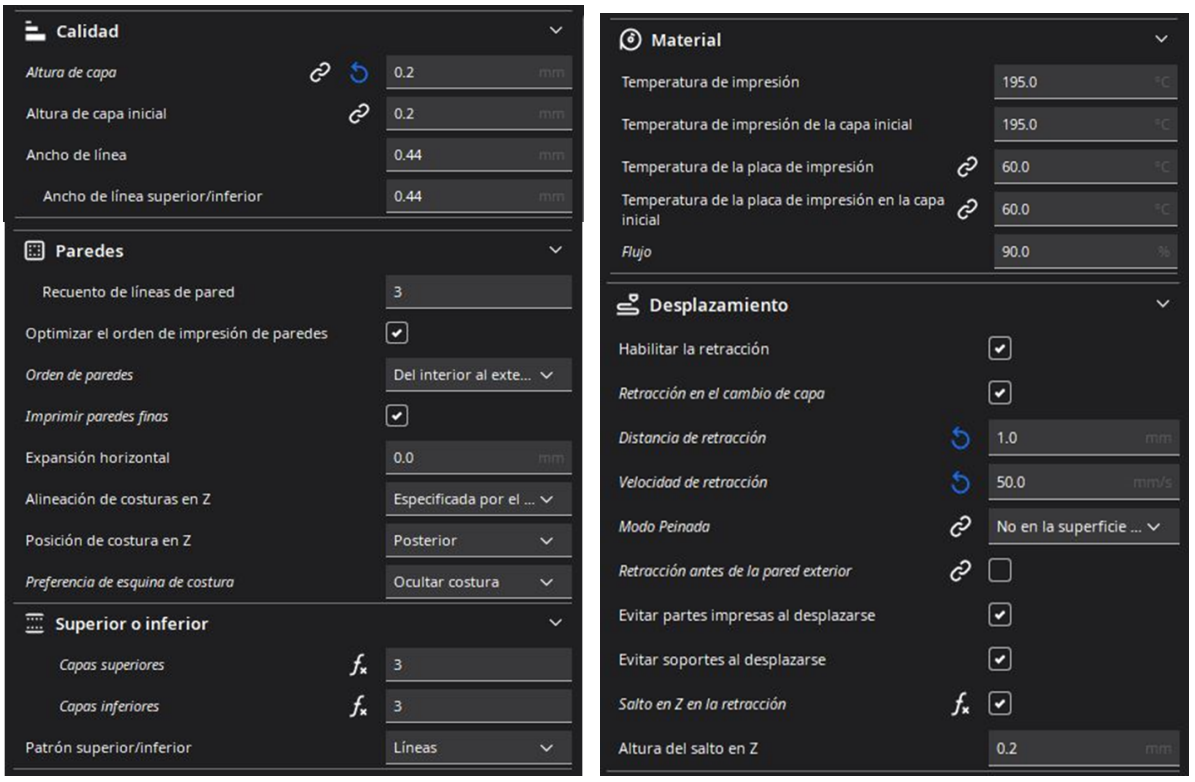


Figura 15: Configuraciones de impresión

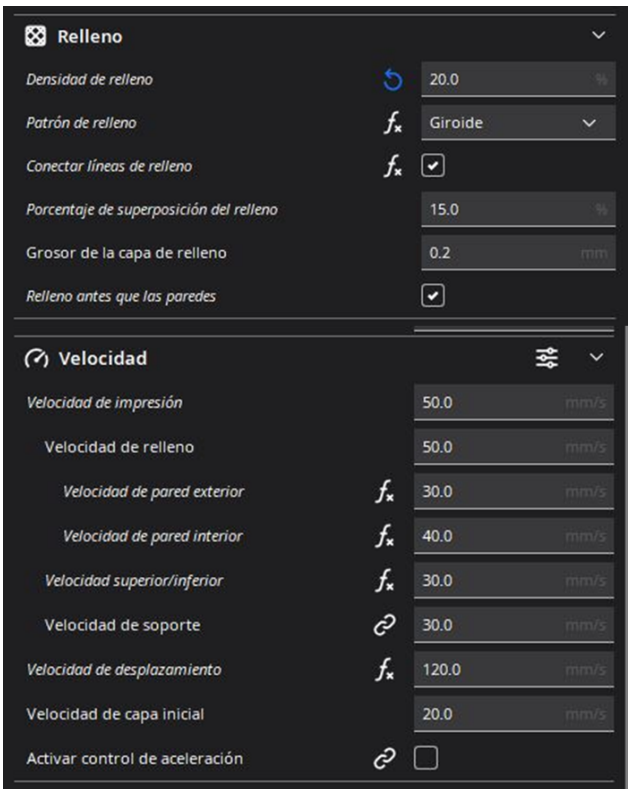


Figura 16: Configuraciones de relleno y velocidad de impresión

3.2.2. Pinza y acople de la cámara

Para incorporar un efector final al robot delta se realizó el diseño de una pinza que consta de una muñeca que permite la rotación de la pinza alrededor de su eje longitudinal (roll), pudiendo orientarla en cualquier dirección según la necesidad de la aplicación en la que se esté utilizando el robot.

La muñeca se encuentra accionada por un servomotor de 180º que le permite realizar dicho movimiento, además esta muñeca se encuentra acoplada a una base de soporte de un servomotor y de desplazamiento de las paletas.

La base del servomotor permite conectar la muñeca con la base del robot delta, a esta se le incorporó un acople para integrar la cámara, dicho acople será colocado justo por encima de la base del servomotor de 180º sobresaliendo justo por encima de la base del robot (Figuras 17 y 18).

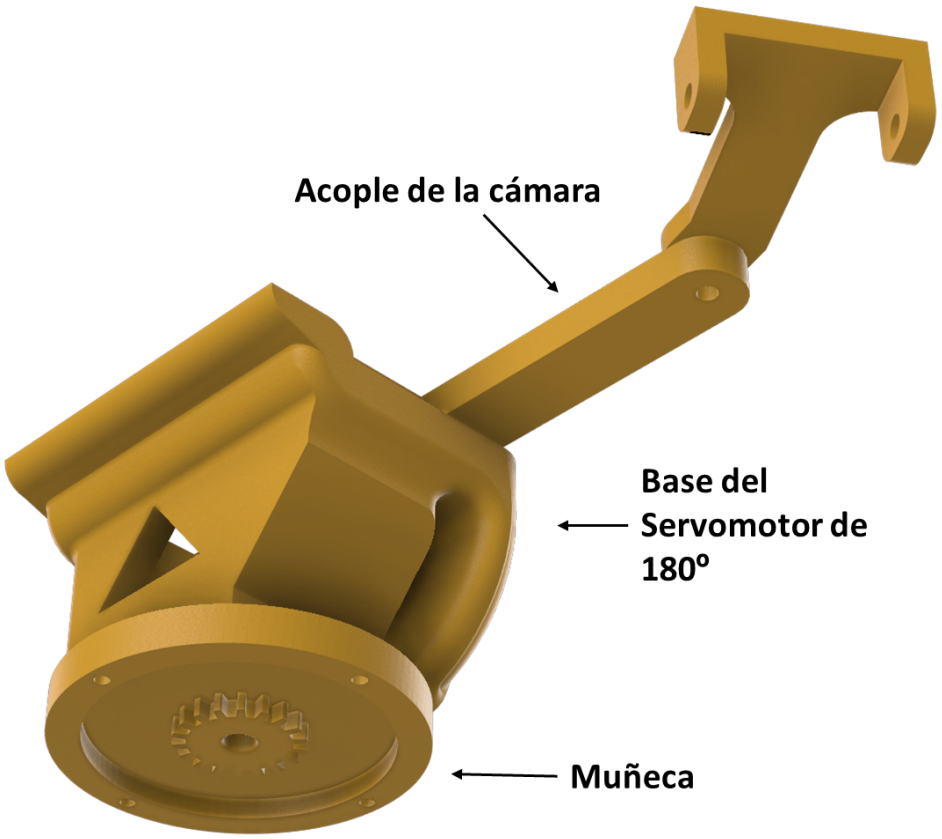


Figura 17: Esquema de partes de la muñeca (vista inferior)

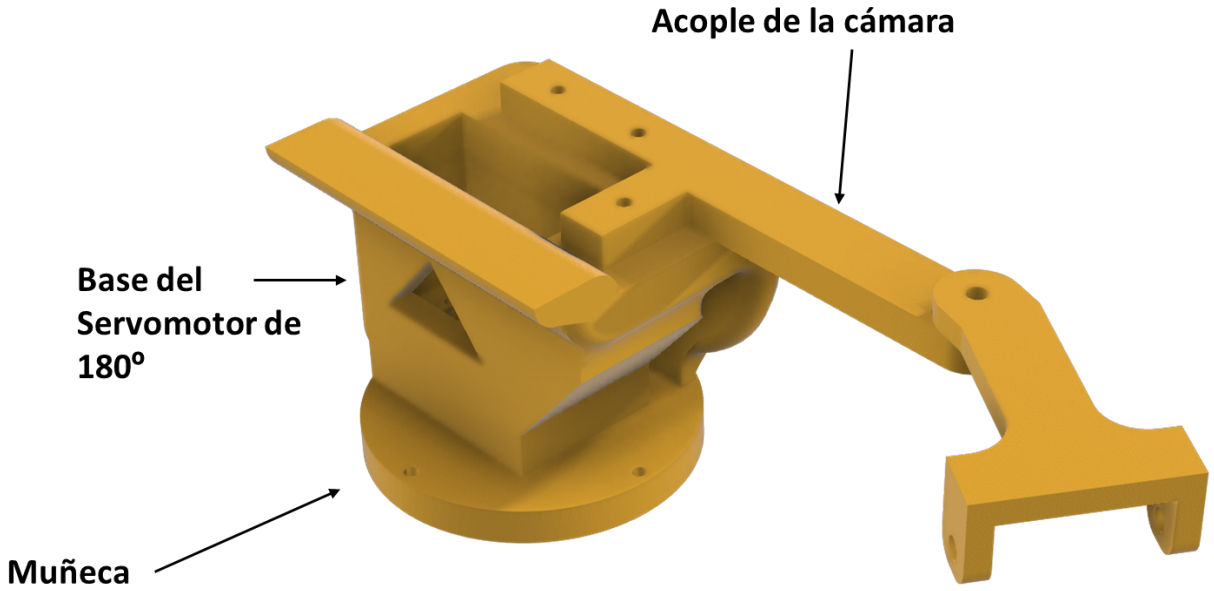


Figura 18: Esquema de partes de la muñeca (vista superior)

La base de desplazamiento de las paletas consta de cuatro rieles en los cuales se colocaran las dos paletas conectadas mediante un engranaje (Figura 19) que será acoplado a un servomotor de 270º para permitir el desplazamiento de dichas paletas en los rieles pudiendo realizar el movimiento de cierre y apertura de la pinza (Figura 20).

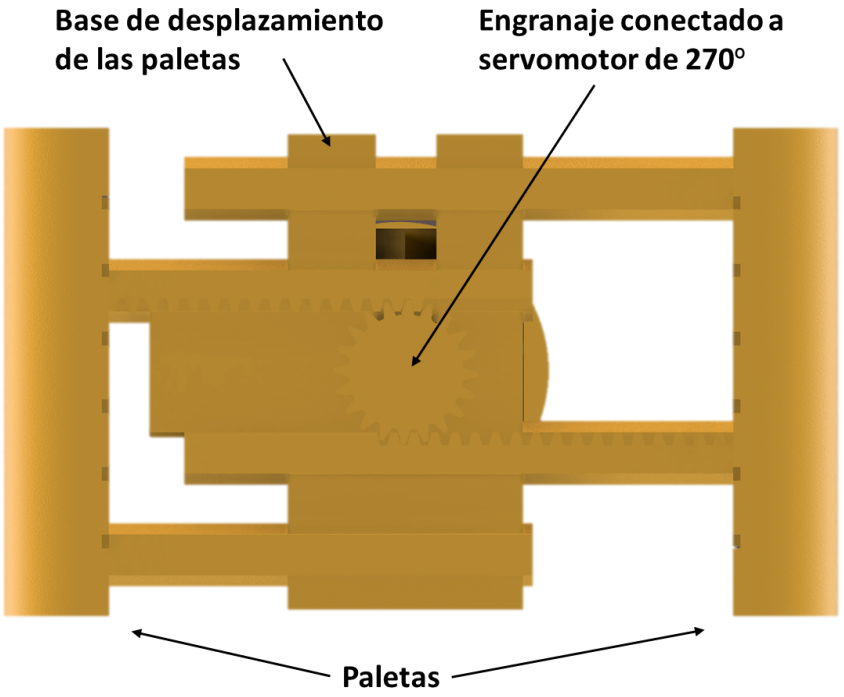


Figura 19: Esquema de partes de la pinza (vista inferior)

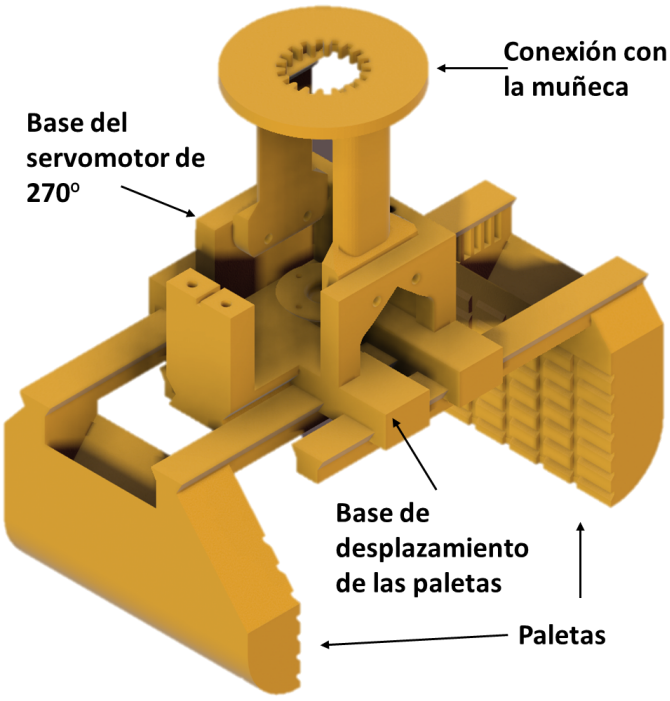


Figura 20: Esquema de partes de la pinza (vista superior)

Por otro lado, se diseñó la base del servomotor de 180º el cual permitirá conectar la muñeca con la base del robot delta. A esta base de acople entre dispositivos también se debe de integrar la cámara, es por esto que se diseñó un acople para esta que será colocado justo por encima de la base de conexión entre pinza y robot (Figura 21). Todos estos elementos fueron colocados en el software Cura para verificar su correcta impresión (Figura 22).

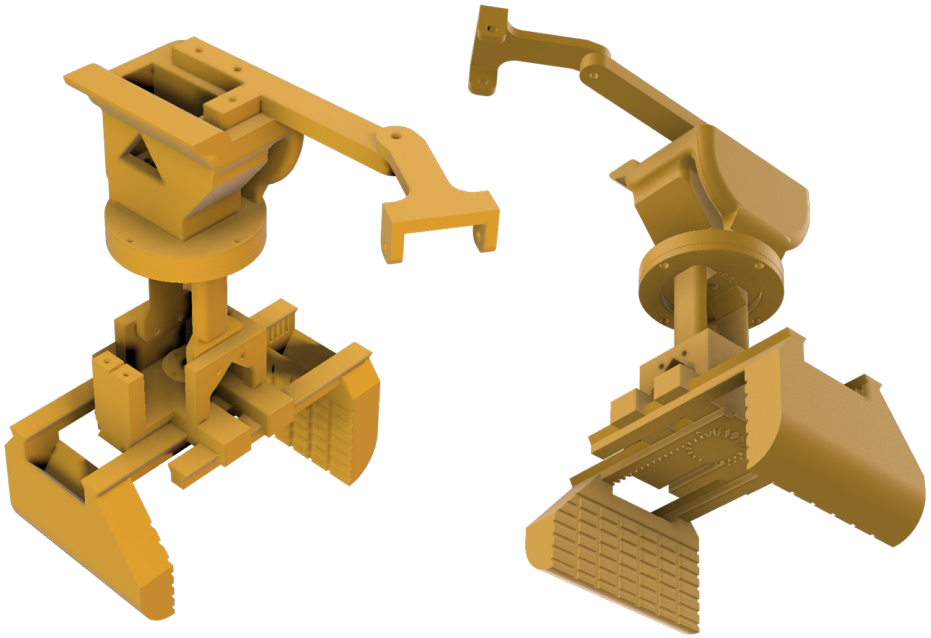


Figura 21: Pinza completa en vista superior e inferior

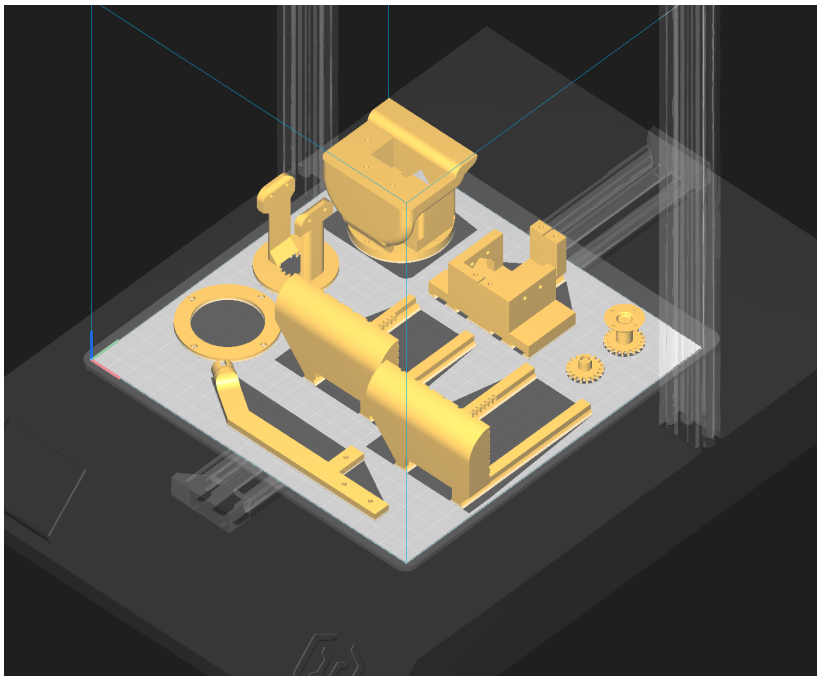


Figura 22: Disposición de los elementos de la pinza en el software Cura para la impresión

3.2.3 Cintas Transportadoras

Para permitir la entrada de objetos en el campo de visión de la cámara de forma controlada y la salida del área de trabajo de las piezas seleccionadas se fabricaron dos cintas transportadoras. Para cada una de estas cintas se diseñaron distintos elementos que fueron impresos en 3D, estos fueron: dos rodillos, un soporte para el rodillo y el motor (en este caso se utilizaron motores paso a paso Nema 17) y un soporte para otro rodillo, así como tres pasadores de forma cónica para sostener dichos rodillos (Figura 23). En las bases que sostienen los rodillos se colocaron 3 rolineras que permiten el giro libre de estos. Un extremo, de uno de los dos rodillos, está conectado al eje de un motor paso a paso con el propósito de permitir su rotación.

Para conectar las dos bases se utilizaron maderas recicladas las cuales fueron cortadas a medida para acoplarlas correctamente a los soportes impresos en 3D. Con respecto a la cinta, se utilizó tela de tapicería de techo de coche ya que esta proporciona una buena elasticidad y firmeza a un bajo coste. Todos estos elementos fueron colocados en el software Cura para verificar su correcta impresión (Figura 24).

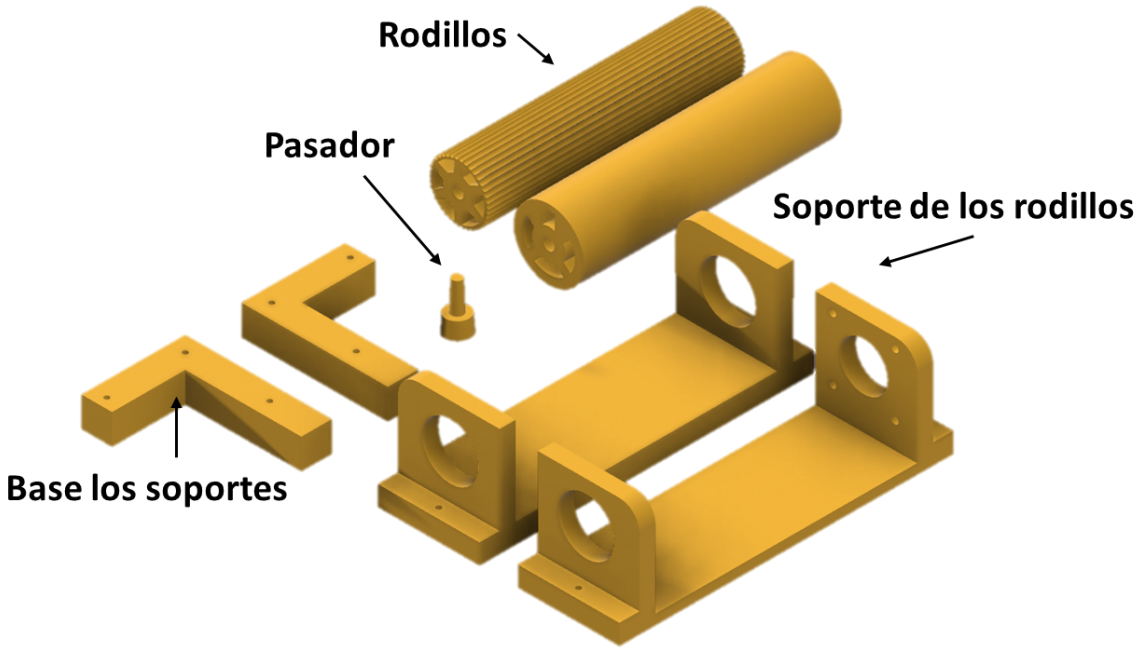


Figura 23: Esquema de partes de las cintas transportadoras

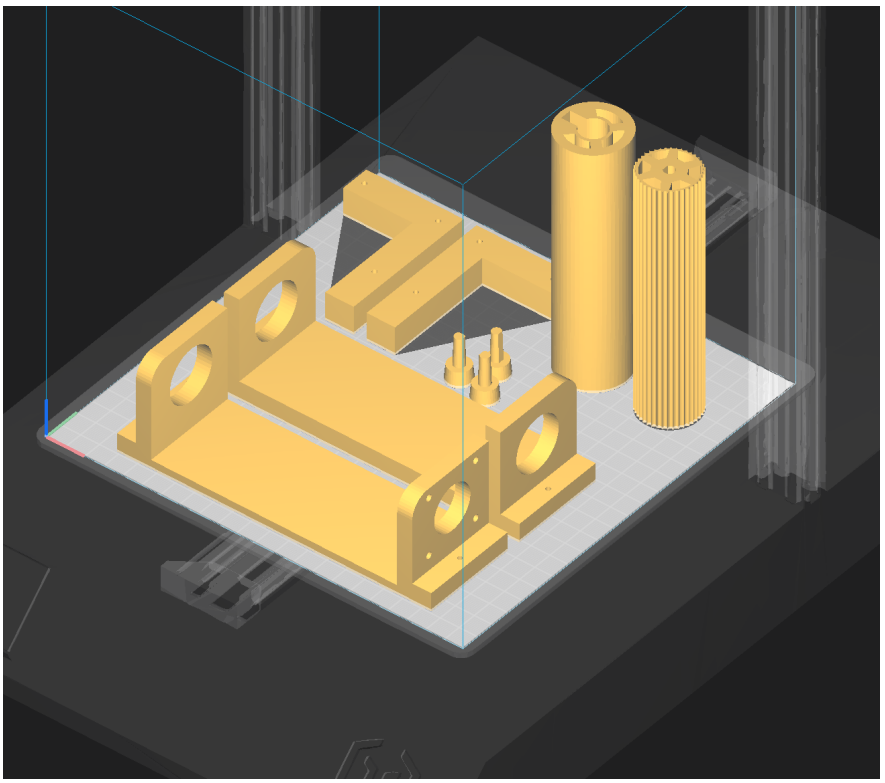


Figura 24: Disposición de los elementos de las cintas transportadoras en el software Cura para la impresión

3.2.4. Objetos a analizar

Como se ha dicho anteriormente, en este trabajo realizaremos un modelo personalizado de YOLOv8 para detectar objetos específicos; es por este motivo que se decidió realizar la impresión en 3D de dichos objetos. Se decidió que los objetos tuvieran formas geométricas variadas (Figura 25) (círculos,cuadrados,rectángulos y hexágonos) así como diferentes texturas y colores. Para simplificar la tarea de la impresión se utilizó unicamente filamento de color blanco y luego las piezas fueron pintadas a mano para tener objetos en distintos colores.

Con respecto al tamaño se decidió que los objetos tuvieran una medida estándar de diámetro o de lado de 3.5cm en el caso de los círculos, hexágonos y cuadrados, y de 7cmx3.5cm para los rectángulos.

Con respecto a las texturas se eligieron 3 patrones con superficies irregulares que fueron colocados en la parte superior de las piezas.

Todos estas piezas fueron colocadas en el software Cura para verificar su correcta impresión (Figura 26).

Figura 25: Diseño de los objetos geométricos a detectar

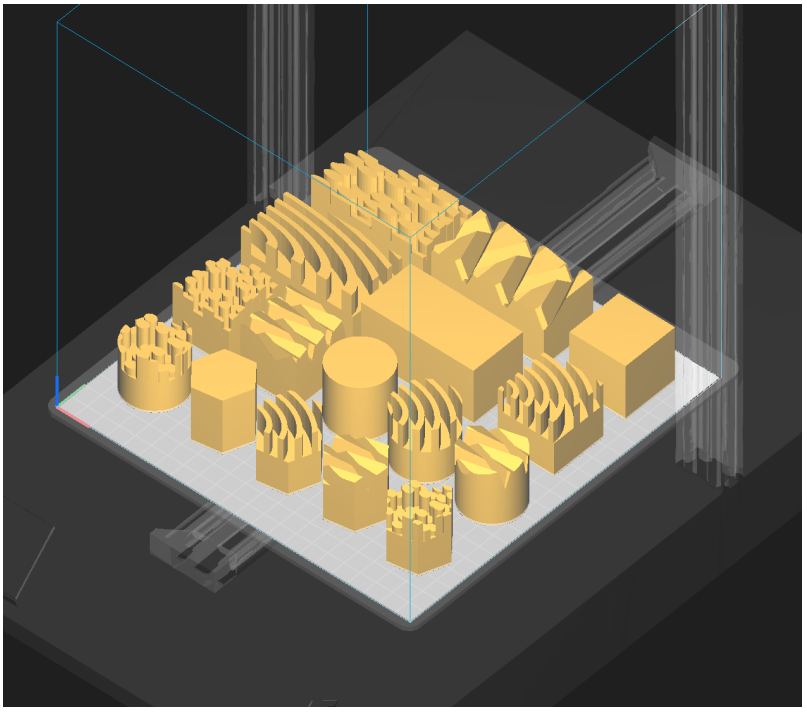


Figura 26: Disposición de los objetos geométricos en el software Cura para la impresión

3.2.5. Caja de arduino

Se diseñó una caja que permitiese fijar ambos arduinos con su correspondiente CNC shield y drivers de los motores, incluyendo agujeros para permitir la salida de los cables de conexión (Figura 27).

A esta caja se le incorporó un ventilador que permite disipar el calor que generan los drivers de los motores. También se le incorporó a esta el interruptor de emergencia y el interruptor de encender/apagar la alimentación para facilitar su pulsación en el caso de que existiese una emergencia. Además, se le agregó un conector que permite conectar la fuente de alimentación.

Por último, se diseñaron unas pestañas en sus extremos que permiten fijar con tornillos la caja al soporte superior de madera de la base del robot. Esta caja fue colocada en el software Cura para verificar su correcta impresión (Figura 28).

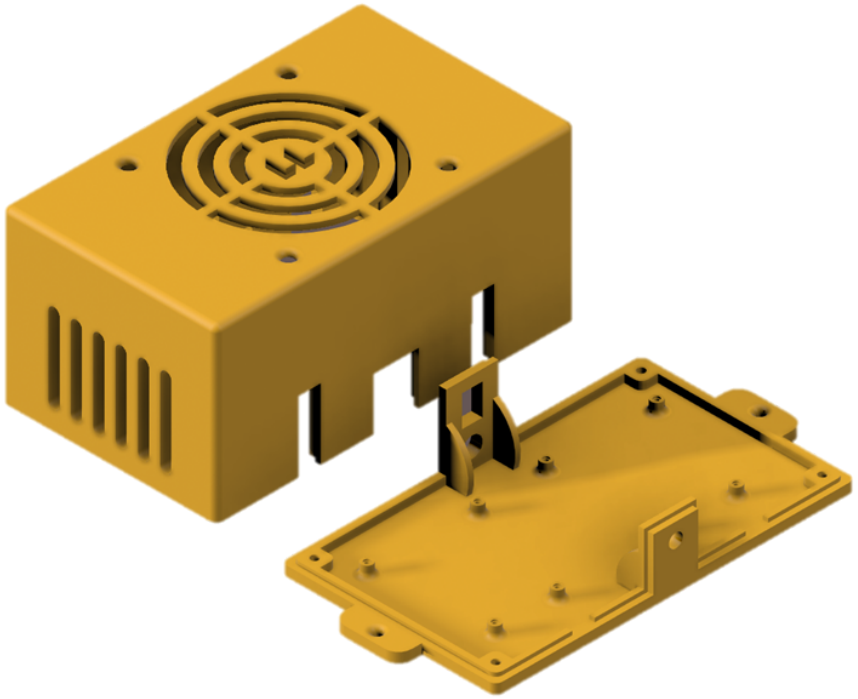


Figura 27: Diseño de la caja de soporte de los Arduino

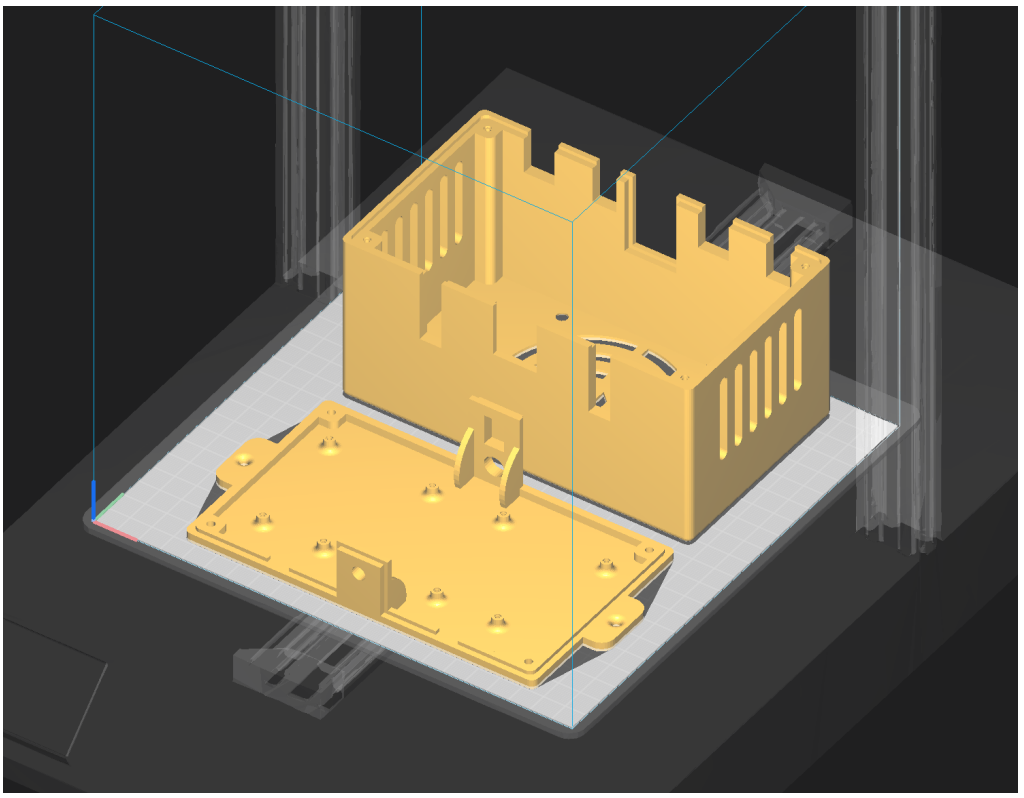


Figura 28: Disposición de la caja de soporte de los Arduino en el software Cura para la impresión

3.3. Componentes electrónicos

Los componentes electrónicos utilizados que se encontraban ya incorporados al robot delta existente son:

- 3 motores paso a paso Nema 17
- 3 drivers TMC 2208
- 3 finales de carrera
- 1 fuente de alimentación de 12V

Los componentes electrónicos que se incorporaron fueron:

- 2 placas Arduino Uno
- 2 módulos CNC Shield V3

- 2 controladores TMC 2226
- 2 motores paso a paso Nema 17
- 1 fuente de alimentación de 18V
- 1 interruptor de emergencia de tres posiciones
- 1 interruptor para cortar alimentación
- 1 mini servomotor SG90 de180º
- 1 servomotor MG946 de 270º
- 1 ventilador 12V
- Cables Dupont para conexionado de servomotores

En el proyecto se utilizaron Arduino Uno debido a que son una opción de placa controladora sencilla y económica pero que ofrece suficientes recursos para adaptarse al proyecto. A ambas placas se les acopló un módulo CNC shield V3 .

A la placa controladora del robot (arduino con módulo CNC shield, ver figura 29) se le conectaron los tres drivers TMC 2208 y los tres motores paso a paso Nema 17 de cada uno de los brazos del robot. También se le conectó el interruptor de emergencia y los tres finales de carrera del robot

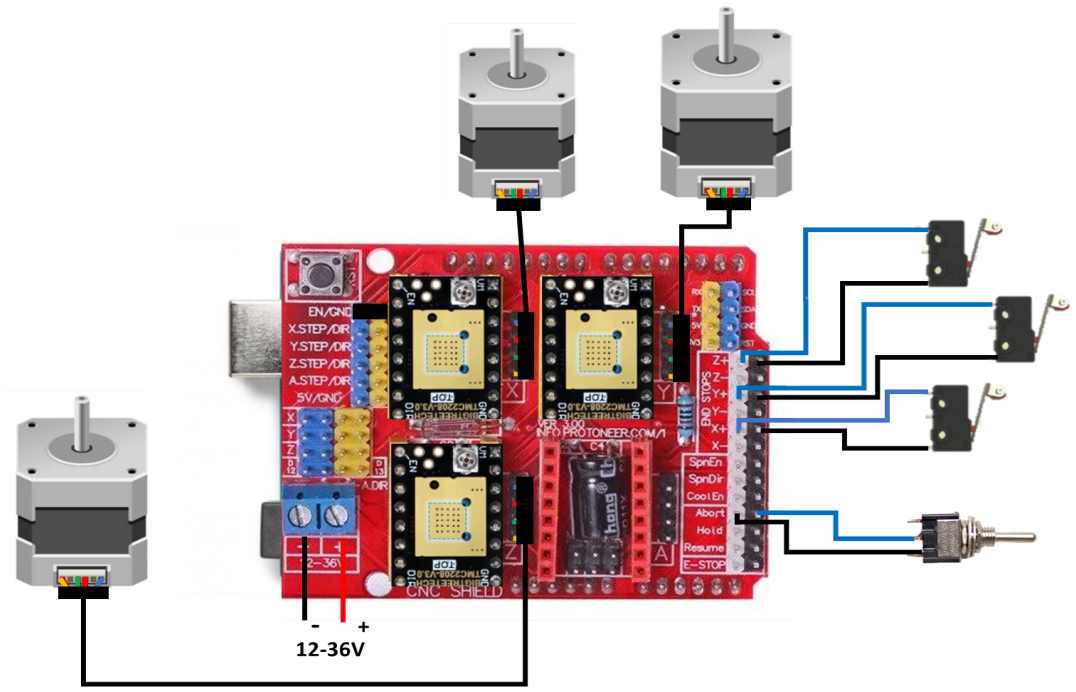


Figura 29: Esquema de conexiones para el control del robot

A la placa controladora de las cintas y de la pinza (otro arduino con módulo CNC shield) se le conectaron los dos drivers TMC 2226 y dos motores paso a paso Nema 17 de cada una de las dos cintas transportadoras, también se le conectó el interruptor de emergencia y los dos servomotores (SG90 y MG946) (Figura 30). Ambas placas fueron alimentadas por la fuente de 18V.

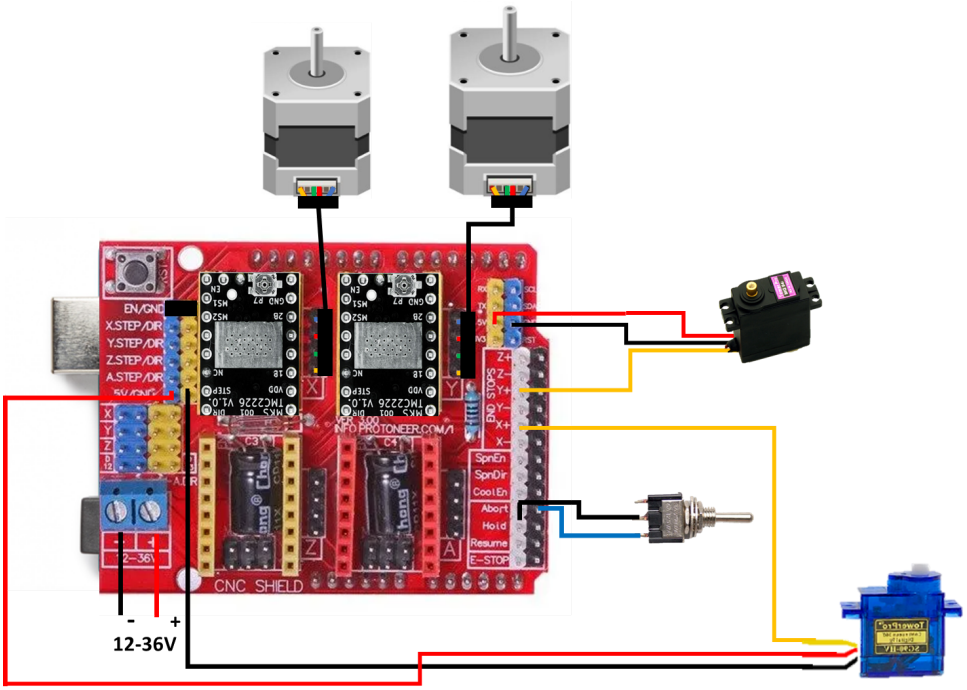


Figura 30: Esquema de conexiones para el control de la pinza y las cintas transportadoras

3.4Entornos de programación

3.4.1.Arduino IDE

Para programar los controladores se utilizó el entorno de desarrollo integrado de software libre Arduino IDE. Este entorno es una aplicación que proporciona una interfaz gráfica de usuario para escribir, compilar y cargar el código en la placa de Arduino. El software utiliza el lenguaje de programación Arduino, que está basado en C/C+ +.

Se realizaron dos programas, uno para el controlador del robot y otro para el controlador de las cintas y la pinza.

3.4.2.Python

Con respecto al ámbito de visión por ordenador y procesamiento de imágenes, se utilizó el lenguaje de programación Python en el entorno de desarrollo Visual Studio Code, debido a que este lenguaje tiene la capacidad de hacer funcionar modelos personalizados de YOLOv8, así como por la amplia variedad de librerías especializadas en el procesamiento de imágenes, tales como OpenCV, NumPy y Regionprops.

Al utilizar estas herramientas, se pueden detectar objetos y segmentarlos para obtener características como color, textura, ubicación y orientación, lo que permite realizar análisis detallados y precisos en el procesamiento de imágenes.

En los siguientes apartados se explicará paso a paso el uso que se le dio a cada una de estas herramientas para la detección de características en objetos

3.5.Procesamiento de imágenes y video en tiempo real

En este apartado se describirán todos los elementos utilizados para llevar a cabo las tareas de detección de objetos, segmentación y extracción de características.

3.5.1 Integración de la cámara

Con el objetivo de poder visualizar el área de trabajo, se ha incorporado una cámara en la base del efector final del robot. Para sostener firmemente la cámara, se ha colocado un acople que permite sujetar y asegurar la cámara en la posición deseada pudiendo obtener una vista clara y precisa del área de trabajo del robot.

La cámara utilizada es la Tyro de la marca Trust (Figura 31) la cual cuenta con una resolución de 1920 x 1080 píxeles y una velocidad máxima de cuadro de 30 pps. Además cuenta con sistema de autoenfoque y balance de blancos automático.

Figura 31:Cámara incorporada al robot

3.5.2 Puesta en marcha del modelo personalizado de YOLOv8

3.5.2.1 Construcción del Dataset

Como hemos descrito en el apartado de fundamentos teóricos, el modelo personalizado de YOLOv8 nos permite realizar los procesos de detección y segmentación de objetos específicos. Para realizar estas tareas es necesario en principio generar el modelo personalizado, para esto es necesario realizar las siguientes tareas que se describen a continuación.

En principio, se debe crear un dataset que contenga las imágenes con los objetos que se desean detectar así como las etiquetas correspondientes a los objetos en cada una de las imágenes. Para crear el dataset de este trabajo se utilizó la plataforma de Roboflow, en la cual se creó un proyecto en el que se subieron 175 imágenes de los objetos a identificar que se habían tomado previamente en entornos variados (ver figura 32).

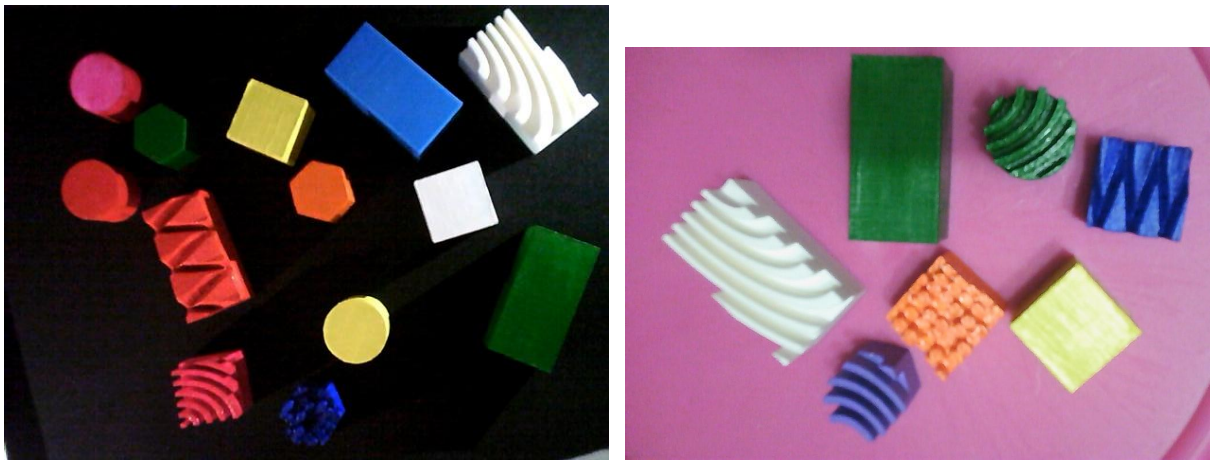


Figura 32:Ejemplo de imágenes utilizadas para entrenar el modelo

Seguidamente, se procede a realizar el etiquetado de los objetos en cada una de las imágenes, para esto se utilizó la herramienta Smart Polygon de RoboFlow, que utiliza un modelo de aprendizaje automático detrás de escena para sugerir una forma para los objetos.

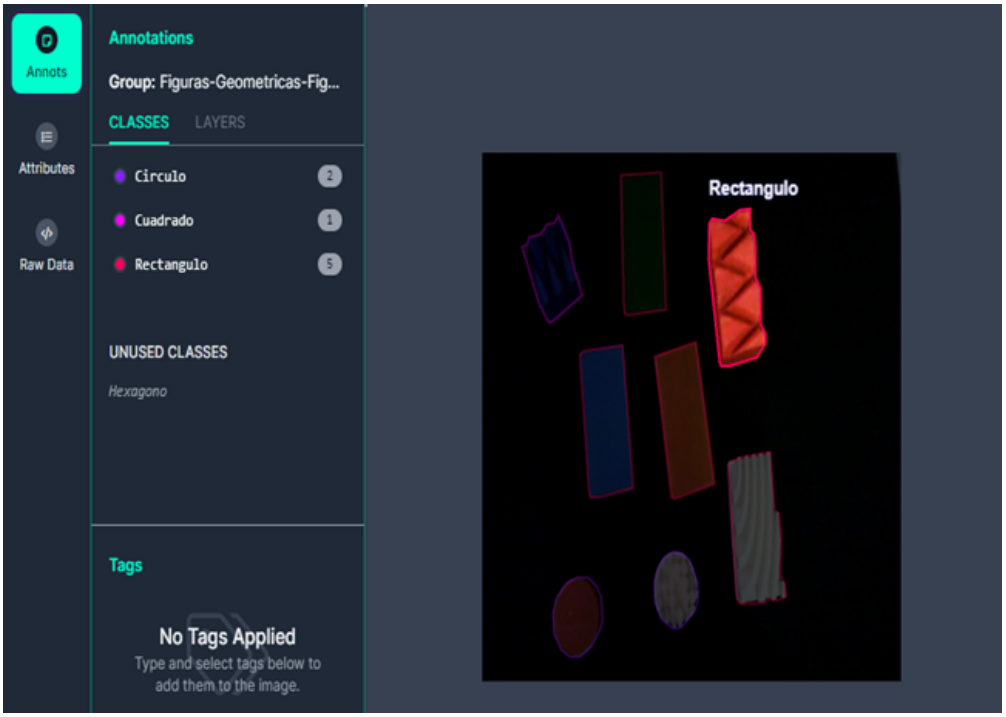


Figura 33:Ejemplo etiquetado de imágenes en RoboFlow

Una vez etiquetados todos los objetos en todas las imágenes (Figura 33), se divide el total de las imágenes entre imágenes para entrenamiento (*Train*), imágenes para validación (*Val*) e imágenes para prueba (*Test*). El porcentaje escogido en este caso fue un 80% de las imágenes para *Train*, un 15% para *Val* y un 5% para *Test* (ver figura 34).

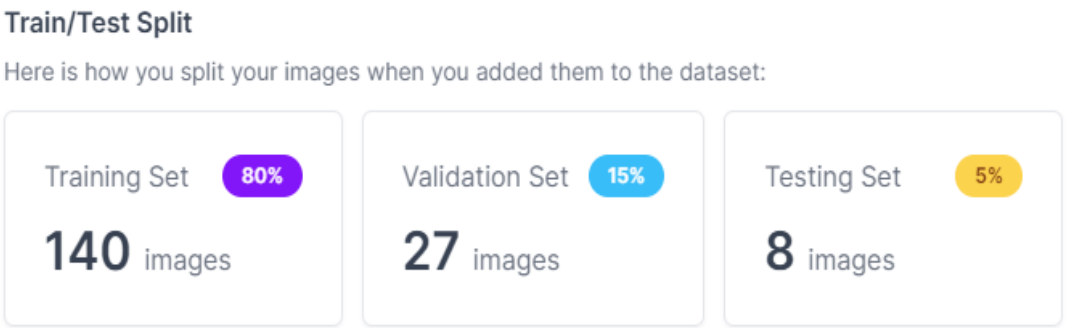


Figura 34:División del dataset (train/val/test) en RoboFlow

Seguidamente, se nos permite aplicar a las imágenes técnicas de preprocesado con la finalidad de reducir el tiempo de entrenamiento y aumentar el rendimiento aplicando transformaciones de imagen a todas las imágenes del conjunto de datos. Las técnicas de preprocesado utilizadas fueron la auto-orientación de imágenes y el redimensionamiento a un tamaño de 640x640.

Antes de exportar los datos, podemos realizar la expansión del tamaño del dataset utilizando transformaciones en las imágenes existentes para crear nuevas variaciones con la finalidad de hacer que los modelos sean más precisos.En este caso se utilizó la modificación de rotación en la imagen (± 38º) y el aumento y disminución del brillo (± 54%). Esto permitió tener un dataset con un total de 455 imágenes.

Para finalizar se realizó la exportación del modelo, en el cual se nos permitió elegir entre diferentes formatos y versiones de YOLO, con la opción de descargar un archivo comprimido o de obtener el código para descargarlo desde un entorno de programación. Se seleccionó el formato YOLOv8 y la descarga en formato comprimido.

Este archivo comprimido contiene el archivo .yaml y tres carpetas (Train, Val y Test) que a su vez cada una tendrá una carpeta llamada images dónde estarán las imágenes correspondientes y otra llamada labels donde estarán las etiquetas.

El archivo .yaml contiene la ruta de acceso a los elementos de las carpetas y la lista de las diferentes clases de objetos a identificar que en este caso son las siguientes figuras geométricas: ['Circulo', 'Cuadrado', 'Hexagono', 'Rectangulo'](Ver figura 35).

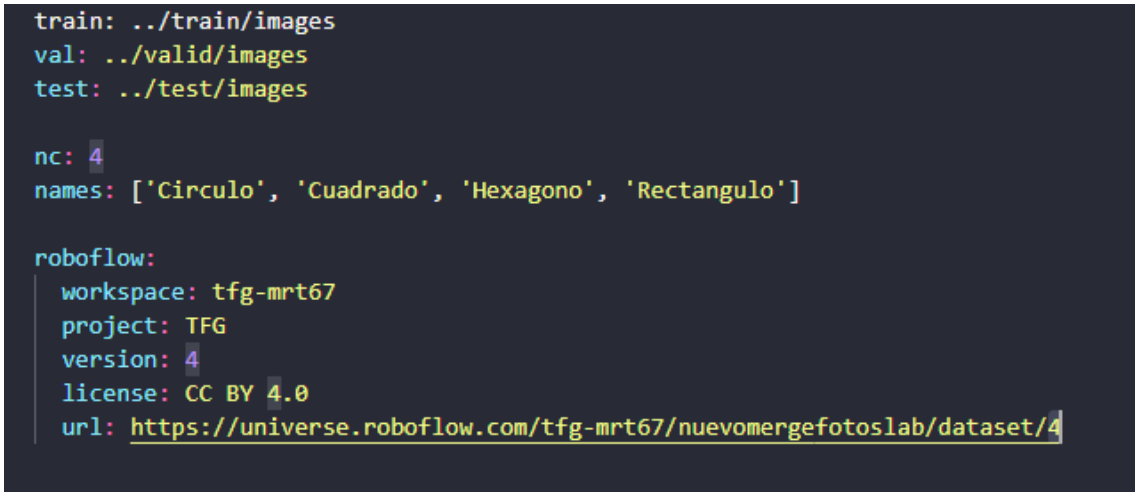


Figura 35:Contenido del archivo data.yaml

Antes de iniciar el proceso de entrenamiento del modelo, se deben descomprimir y subir todos los elementos obtenidos en el archivo .zip a Google Drive.

3.5.2.2. Entrenamiento del modelo

Para realizar el entrenamiento del modelo, se utilizó parte de la información contenida en el cuaderno de Google Colab oficial para el entrenamiento de modelos personalizados proporcionados por Ultralytics. Debido a que dicho cuaderno contenía mucha información, se decidió hacer un nuevo cuaderno de Google Colab que permitiese el entrenamiento del modelo de forma sencilla y en la menor cantidad de pasos posibles.

Este cuaderno inicialmente se instalan e importan las librerías necesarias, luego conecta con el Drive para tener acceso al dataset y al archivo .yaml

Seguidamente pasamos a editar los parámetros que se colocan al entrenar el modelo (ver figura 36).

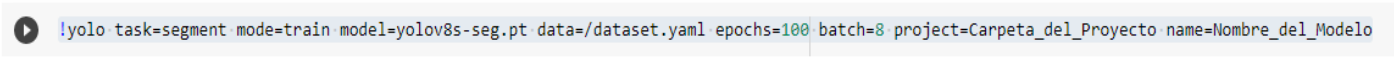


Figura 36:Parámetros de entrenamiento del modelo personalizado de YOLOv8

En principio colocaremos en “task” la tarea que queremos realizar ya sea detectar, segmentar o clasificar. En segundo lugar colocaremos el “mode” en “train” indicando que queremos realizar el entrenamiento del modelo. En tercer lugar tenemos las épocas “epochs” que hace referencia al número de veces que se entrenará el modelo en el conjunto de datos especificados; a medida que el modelo se entrene en cada época, ajustará sus pesos y parámetros para mejorar su rendimiento en la tarea especificada. Es usual colocar valores entre 50 y 150 épocas para obtener un buen resultado.

En cuarto lugar tenemos el “batch” que hace referencia a la cantidad de imágenes que se procesan simultáneamente durante el entrenamiento de un modelo, debido a que el modelo no procesa todas las imágenes de entrenamiento de una sola vez, sino que se divide en lotes más pequeños y se entrena en esos lotes. Los valores comunes para el batch suelen ser entre 6 y 64, un valor más alto aumenta la eficiencia del entrenamiento pero requiere más memoria y afecta el

rendimiento de la red neuronal, por otro lado, un valor de batch más bajo reduce el uso de memoria y aumenta la precisión de la red neuronal, pero también puede hacer que el proceso de entrenamiento sea más lento. En quinto lugar tenemos los parámetros “project” que será la ruta de Google Drive en donde queremos guardar toda la información de nuestro modelo y el parámetro “name” en el que colocaremos el nombre que se le quiere dar al proyecto.

Por último, tenemos el parámetro “model” que hace referencia al archivo en formato .pt del modelo pre-entrenado que se utilizará para el entrenamiento del modelo, este archivo contiene los pesos y la arquitectura que se utilizan como punto de partida para el entrenamiento del modelo con los datos personalizados especificados en el archivo .yaml el cual se colocará como "data=dataset.yaml" .

YOLOv8 proporciona diversos modelos pre-entrenados en función de la tarea que se requiera realizar, la velocidad de aplicación de dicho modelo y la complejidad del mismo. En función de la velocidad de aplicación del modelo tenemos versiones nano, small, medium, large y xlarge.La versión nano es la más rápida pero menos precisa, estas versiones van aumentando su precisión pero disminuyendo su velocidad hasta llegar a la versión xlarge.

En función de la tarea a realizar tenemos la detección, la segmentación y la clasificación, para cada una de estas tareas tendremos todos los diferentes modelos según su velocidad, por ejemplo, en este caso la tarea que queremos realizar es la segmentación de objetos en tiempo real, por lo que se escogió la versión de segmentación small, también para realizar es la segmentación de objetos sobre imágenes se escogió el xlarge para que fuese lo más precisa posible ya que partiendo de esa se realizará el análisis de características.

Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-seg	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s-seg	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m-seg	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l-seg	640	52.3	42.6	572.4	2.79	46.0	220.5
YOLOv8x-seg	640	53.4	43.4	712.1	4.02	71.8	344.1

Figura 37:Versiones del modelo pre-entrenado de YOLOv8 que realizan tareas de detección y segmentación de objetos

En la figura 37 se pueden observar las versiones existentes en la tarea de segmentación y sus parámetros de funcionamiento.

3.5.2.3. Aplicación de los modelos personalizados

Una vez obtenidos el modelo personalizado en formato small y xlarge se procede a la aplicación de los mismos.

El modelo en el formato small será aplicado en tiempo real debido a tener mayor rapidez de procesamiento, este irá analizando los objetos que entran en el campo de visión mediante una cinta transportadora.

Una vez detectado por el modelo small un objeto que contenga la forma seleccionada por el usuario, se detendrá la cinta por la que entran objetos y se realizará una captura. A esta imagen se le aplicará el modelo xlarge para obtener información más precisa de los elementos detectados y así poder aplicar las técnicas de procesamiento de imágenes que se describirán a continuación

3.5.3 Herramientas de Python para la implementación

3.5.3.1. Detección de objetos en tiempo real

La detección de objetos en tiempo real se realiza utilizando la función VideoFeedObjectDetection, en esta función, primero se carga el modelo personalizado (en su versión small) de YOLOv8 que se ha creado. Luego, se definen las coordenadas de la región del video en la cual se desea realizar la detección de objetos.

Seguidamente se inicia el movimiento de la cinta transportadora (controlada a través de una conexión con Arduino) para el desplazamiento de los objetos a la región de interés especificada.

Una vez encendida la cinta transportadora se crea un bucle infinito en el cual se recorta el frame para quedarse únicamente con la región de interés y se aplica el modelo en esta región. Además de aplicar el modelo, a medida que van ingresando los objetos en el área especificada se va analizando la forma de estos (clase a la

que pertenece) (Figura 38) y si coincide con la especificada se inicia un tiempo de espera para esperar a que entre completamente en el área de interés, luego se apaga la cinta transportadora y se realiza una captura del frame, esta captura será devuelta por la función para su posterior procesamiento.

Figura 38:Detección y segmentación de objetos sobre la cinta transportadora

3.5.3.2. Detección de objetos en un frame

Una vez obtenida la captura del frame descrita en el apartado anterior se realizará el procesamiento de la misma, para esto se ha creado una función llamada ImgSegmentacion la cual recibe como parámetro de entrada la captura que contiene el objeto. A esta captura se le aplicará el modelo personalizado (en su versión xlarge) de YOLOv8 que se ha creado y se irán extrayendo las características aportadas por el mismo, que son el valor de la clase a la que pertenece (la forma de la objeto), la ubicación de su centroide, las coordenadas del *bounding box* y la máscara que contiene al objeto (Figura 39). Seguidamente, se procederá a crear una nueva máscara que contenga a cada uno de los objetos reconocidos por

separado con sus colores originales utilizando la máscara extraída del modelo y la imagen original.

Esta función devolverá toda la información que se ha extraído de los objetos detectados (clases, *bounding box*) y las máscaras de cada uno de ellos.

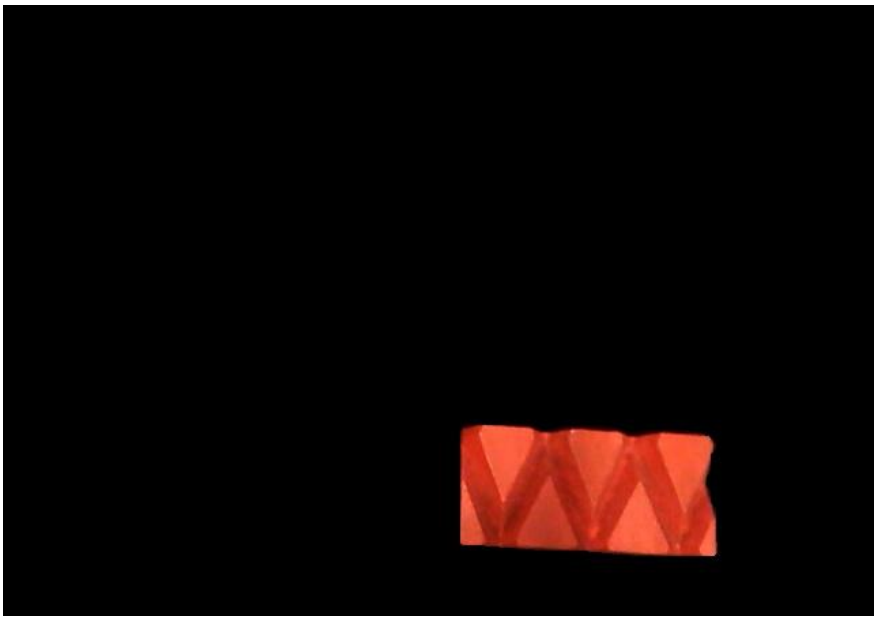


Figura 39: Ejemplo de máscara generada por la función ImgSegmentacion

3.5.3.3. Deteccion de Color

El análisis del color de los objetos se realizará utilizando los espacios de colores HSV y LAB, para realizar esta tarea se creará una función en Python que recibirá como parámetro de entrada la mascara del objeto al cual se le desea aplicar el analisis de color, esta mascara contiene unicamente el objeto a tratar con sus colores originales. En esta función, inicialmentese convierte la imagen original a los espacios de color HSV y LAB. Luego, se definen los valores de los rangos de color de cada color de interés en ambos espacios mediante la especificación de los valores de los límites inferior y superior de cada color con la finalidad de poder identificar los píxeles que corresponden a ese color en particular.

Después de definir los rangos de color, se aplican las máscaras de color a la imagen de entrada. Las máscaras se generan utilizando la función cv2.inRange, que crea una imagen binaria en la que los píxeles correspondientes al color de interés se marcan como 1 y los píxeles que no corresponden a ese color se marcan como 0

Por ultimo, se crea un diccionario en donde se asignan como claves los nombres de los colores de interés y como valores la suma de los valores de los píxeles en las máscaras correspondientes a cada color en los espacios de color HSV y LAB. Con este diccionario y utilizando la función max obtendremos la clave (es decir, el nombre del color) correspondiente al valor máximo en el diccionario.

3.5.3.4. Detección de Textura

Para determinar la textura que poseen los objetos detectados se ha creado una función que analiza las propiedades de la imagen utilizando la matriz de co-ocurrencia de nivel de gris (GLCM). Esta función recibe como parámetro de entrada la máscara que contiene únicamente el objeto a tratar con sus colores originales, convirtiéndola luego a escala de grises para poder calcular la matriz GLCM utilizando la función *graycomatrix* de la librería *skimage* con una distancia de 1 pixel y un ángulo de 0 grados.

Seguidamente, se calculan diversas propiedades de la GLCM (mediante la función *graycoprops*), como el contraste, la disimilitud, la homogeneidad, el ASM (segundo momento angular), la energía y la correlación. Estas propiedades proporcionan información sobre las características de textura presentes en la imagen.

Después de calcular las propiedades de la GLCM, se procede a determinar si la textura es lisa o áspera. Esto se realiza utilizando una serie de condiciones que comparan los valores de las propiedades de la GLCM con ciertos umbrales predefinidos.

Finalmente, se retorna la etiqueta de la textura (liso o áspero), junto con los valores de energía, homogeneidad, disimilitud y contraste, que pueden ser útiles para análisis posteriores.

3.5.3.5. Identificación de Orientaciones

Para obtener la orientación de los objetos se ha creado una función a la cual se le da como parámetro de entrada máscara de la imagen con el objeto que se desea analizar. En primer lugar, se utiliza la función de cv2 llamada connectedComponents,

que permite asignar etiquetas a las regiones en la imagen, convirtiendo previamente la imagen a escala de grises; pudiendo de esta forma identificar y distinguir cada región de la imagen.

Seguidamente se utiliza la función *regionprops* de la biblioteca *skimage*, que calcula las propiedades de la región de una imagen etiquetada para luego iterar sobre estas y encontrar la región más grande.

Una vez que se ha identificado la región más grande, se calcula su orientación utilizando la propiedad "*orientation*" de la región. Esta orientación se devuelve en radianes. Para mayor conveniencia, la orientación se convierte de radianes a grados utilizando la función np.rad2deg y este valor es el que devolverá la función ya que se trata de la orientación del objeto detectado en grados.

3.5.3.6. Definición de la clase Objeto y sus métodos

Con el propósito de optimizar la organización de los objetos y sus respectivas características y máscaras, se optó por implementar una clase en la cual se pueda reunir y gestionar eficientemente dicha información.

La clase creada se denominó “Objeto” y cuenta con los siguientes atributos (Figura 40):

- Imagen: representa la imagen original que contiene el objeto.
- Máscara: representa la máscara del objeto, que indica la región del objeto en la imagen.
- Clase: indica la clase a la que pertenece el objeto.
- Centrox y centroy: representan las coordenadas del centro del objeto.
- Xmin, xmax, ymin, ymax: definen las coordenadas del cuadro delimitador (*Bounding Box*) del objeto.
- AnchoBB y altoBB: representan las dimensiones del *Bounding Box*.
- Orientación: indica la orientación del objeto.
- Color: indica el color del objeto.
- Textura: describe la textura del objeto.
- *Energy*, *homogeneity*, *dissimilarity*, *contrast*: características relacionadas con la textura del objeto.

```python
#Clase objeto
class Objeto:
        #Atributos:
        def __init__(self, imagen, mascara, clase, centrox, centroy, xmin, xmax, ymin, ymax,
                     anchoBB, altoBB, orientacion, color, textura,energy,homogeneity,dissimilarity,contrast):
            self.imagen = imagen
            self.mascara = mascara
            self.clase = clase
            self.centrox = centrox
            self.centroy = centroy
            self.xmin = xmin
            self.xmax = xmax
            self.ymin = ymin
            self.ymax = ymax
            self.anchoBB = anchoBB
            self.altoBB = altoBB
            self.orientacion = orientacion
            self.color = color
            self.textura = textura
            self.energy=energy
            self.homogeneity=homogeneity
            self.dissimilarity=dissimilarity
            self.contrast=contrast
```

Figura 40: Atributos de la clase Objeto

La clase "Objeto" también cuenta con los siguientes métodos:

- PrintBBObj: muestra la imagen con el *Bounding Box* dibujado alrededor del objeto.
- PrintMascara: muestra la máscara del objeto.
- SaveMascara: guarda la máscara del objeto como una imagen separada.
- PrintInfo: muestra por consola la información detallada del objeto, como la clase, las coordenadas, las dimensiones, la orientación, el color y la textura.
- ShowOrientacion: muestra la orientación del objeto en la máscara, dibujando una línea que representa la orientación y una línea perpendicular a ella.
- ShowRecorteTextura: muestra la imagen original con un recorte alrededor del objeto, resaltando un área específica que se utiliza en el análisis de textura.
- ShowMascaraColorTexturaForma: muestra la máscara del objeto con información adicional, como el color, la textura y la forma del objeto.
- Forma: devuelve la forma del objeto(círculo, cuadrado, hexagonal o rectangulo)

Por otro lado, para ir creando elementos de la clase "Objeto" se realizó una función llamada “GeneradorDeObjetos”. Esta función tiene como parámetro de entrada la captura realizada por la función VideoFeedObjectDetection. Luego esta

captura se le pasa como parámetro de entrada a la función ImgSegmentacion para que nos devuelva la información y la máscara extraída mediante la aplicación del modelo de cada uno de los objetos. Con esta información(clase y características del *bounding box*) y las funciones que realizan la detección de color, textura y orientacion utilizando la máscara del objeto, vamos creando, uno a uno,elementos de la clase objeto y guardandolos en una matriz. Esta función devolverá dicha matriz, que en su interior contiene cada uno de los objetos detectados.

3.5.3.7. Calibración y transformación de sistemas de referencia

Para realizar la calibración se realizó una función (Figura 41) que permite obtener el factor de conversión para pasar de distancia en píxeles a cm. Esta función recibe una imagen en la cual se encuentra el área de trabajo y una distancia en mm(ancho del área de trabajo); esta imagen se muestra y se le pide al usuario que dibuje un rectángulo (ROI) para delimitar el área de trabajo. Una vez delimitada el área de trabajo se calcula el factor de conversión de la imagen utilizando la distancia de calibración y la longitud del rectángulo ROI. Finalmente, la función muestra la imagen original con un rectángulo que delimita el ROI y devuelve las coordenadas del ROI, el factor de conversión y la imagen cortada.

```python
def calib(img, distCal_mm=None):

    if distCal_mm is None:
        distCal_mm = 130

    cv2.imshow("Image", img)
    print("Dibuja el rectángulo ROI sobre la imagen y presiona Enter")
    print("Presiona Esc para cancelar")
    coor = cv2.selectROI("Image", img, fromCenter=False)
    cv2.destroyAllWindows()

    Xmin = int(coor[0])
    Xmax = int(coor[0] + coor[2])
    Ymin = int(coor[1])
    Ymax = int(coor[1] + coor[3])

    # Imagen cortada:
    imgCut = img[Ymin:Ymax, Xmin:Xmax]

    # Cálculo del factor de conversión
    d = np.sqrt((coor[2]) ** 2 + (coor[3]) ** 2)
    convF = distCal_mm / d

    # Imagen indicando el área recortada en la imagen original
    cv2.rectangle(img, (Xmin, Ymin), (Xmax, Ymax), (0, 255, 0), thickness=2)
    cv2.imshow("Image with ROI", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return coor, convF, imgCut
```

Figura 41: Función de calibración

Una vez obtenido el factor de conversión se procede a realizar la transformación entre sistemas de referencia; para esto se crearon diferentes funciones que permiten pasar desde el sistema de referencia de la cámara al de la pinza y al de la base del robot. La función principal se llama puntosXYObjetos, ésta recibe como parámetros de entrada las coordenadas del objeto en píxeles, la ubicación actual de la base y el factor de conversión. Seguidamente, se transforman las coordenadas del objeto de píxeles a milímetros utilizando el factor de conversión. Luego se utiliza la función llamada matricesTransformacion, en la cual se crean las matrices de transformación para moverse entre sistemas de referencia colocando los datos de las ubicaciones de cada uno de los sistemas de referencia con respecto a uno de ellos. Seguidamente se llama a la función deCamaraAPinza la cual recibe las coordenadas del objeto en milímetros y una de las matrices de transformación creadas en la función anterior. Esta devolverá las coordenadas del objeto en el sistema de referencia de la pinza.

Por último, se utiliza la función ptoFinalXY a la cual se le pasarán como parámetros de entrada las coordenadas obtenidas en la función anterior y las coordenadas de la ubicación actual de la base; en esta función se calculará las coordenadas a la cual debe de moverse el robot (utilizando los respectivos cálculos de cinemática inversa y dichas coordenadas) para recoger el objeto.Estas coordenadas serán devueltas por la función principal (puntosXYObjetos).

3.5.3.8. Recogida y desplazamiento de piezas

Para realizar la conexión entre la electrónica y el código de Python referente a la detección de objetos, se implementaron varias funciones en Python denominada conexionArduino, conexionArduinoCintasYPinza y conexionArduinoCNC las cuales establecen la comunicación serial con los Arduinos y envían los valores pertinentes para su procesamiento.

- Función conexionArduinoCintasYPinza:

Con respecto a la función de Python llamada conexionArduinoCintasYPinza (Figura 42), se puede decir que esta se encarga de conectarse con el arduino que controla las cintas transportadoras y los servomotores de la pinza para enviarle los valores necesarios para que realicen movimientos específicos.

Los valores que se le dan como parámetros de entrada a la función son los grados correspondientes a la orientación que se le desea colocar a la muñeca, el objeto de clase serial para realizar la comunicación con el arduino y valores de 0 o 1 en función de si se desea abrir pinza e iniciar el movimiento de las cintas(el valor será 1) o se desea cerrar la pinza y detener las cintas (el valor será 0).

Además de transmitirle esta información al arduino, la función esperará dos líneas de respuesta del arduino que confirmen la recepción y ejecución de las instrucciones correspondientes.

```
def conexionArduinoCintasYPinza(mun,pinza,cintaObjetos,cintaDesechos,ser1):
    if pinza == 1:
        pinza = 270 #Si se recibe un 1 en pinza indica que se quiere
                    #abrir la pinza, por lo que le pasamos 270 grados
    if pinza == 0:
        pinza = 0 #Si se recibe un 0 en pinza indica que se quiere
                  #cerrar la pinza, por lo que le pasamos 0 grados


    input1 = "{},{},{},{}".format(int(mun), int(pinza), int(cintaObjetos),cintaDesechos)
    ser1.write(input1.encode('ascii'))

   #leer datos com6
    num_datos_com6 = 2  # cantidad de lineas
    datos_recibidos_com6 = []

    for i in range(num_datos_com6):
        linea = ser1.readline().decode('cp1252').rstrip()
        datos_recibidos_com6.append(linea)

    #imprimimos los datos
    for i, dato in enumerate(datos_recibidos_com6):
        print(dato)
```

Figura 42: Función conexionArduinoCintasYPinza

- Función conexionArduinoCNC:

La función conexionArduinoCNC (Figura 43) se comporta de manera similar a la anterior, con la variación en los parámetros de entrada que serán, además de el objeto de clase serial para realizar la comunicación con el arduino correspondiente, las coordenadas x,y y z del objeto a recoger así como valores de 1 o 0 en caso de que se desee ir a la posición de home o realizar la calibración o no respectivamente.

```python
def conexionArduinoCNC(posicionX,posicionY,posicionZ,home,calibracion,ser2):

    #MOVER A HOME
    if home == 1 and calibracion == 0:
        input2 = "{},{},{},{},{}".format(0, 0, 0, 1, 0)
        ser2.write(input2.encode('ascii'))

    #CALIBRAR
    if home == 0 and calibracion == 1:
        input2 = "{},{},{},{},{}".format(0, 0, 0, 0, 1)
        ser2.write(input2.encode('ascii'))

    #MOVER
    if home == 0 and calibracion == 0 and (posicionX !=0 or posicionY != 0 or posicionZ != 0):
        input2 = "{},{},{},{},{}".format(posicionX, posicionY, posicionZ, 0, 0)
        ser2.write(input2.encode('ascii'))

    #leer datos com4
    num_datos_com4 = 2  # cantidad de lineas
    datos_recibidos_com4 = []

    for i in range(num_datos_com4):
        linea = ser2.readline().decode('cp1252').rstrip()
        datos_recibidos_com4.append(linea)

    #imprimimos los datos
    for i, dato in enumerate(datos_recibidos_com4):
        print(dato)
```

Figura 43: Función conexionArduinoCNC

- Función conexionArduino:

Por último, tenemos la función conexionArduino la cual incorpora las dos funciones anteriores y las funciones de transformación de sistemas de referencia (puntosXYObjetos) para realizar las tareas de recogida, entrega y desecho de piezas en la cinta transportadora, a esta función le daremos como parámetros de entrada las coordenadas de la pieza en pixeles, la orientación a la cual se debe girar la muñeca, los objetos de clase serial para realizar la comunicación con los arduinos, y los valores de 1 o 0 en caso de que se desee realizar calibración o llevar a home o no respectivamente.

Debido a los problemas que serán descritos en el apartado de limitaciones, se dividió el área de recogida en dos cuadrantes de dimensiones 75x80mm y se utilizó la función puntosXYObjetos para determinar en cuál de los dos cuadrantes se encuentra el centroide del objeto. Una vez obtenido el cuadrante donde se encuentra el centroide del objeto se procede a realizar una serie de pasos para su recogida utilizando las funciones conexionArduinoCNC y conexionArduinoCintasYPinza para mover el robot, recoger y soltar el objeto e iniciar la cinta que desechará el objeto

Asimismo, en el entorno de desarrollo de Arduino (Arduino IDE), se crearon dos códigos que permitieran controlar los dispositivos electrónicos conectados. Logrando de esta manera la integración entre la electrónica y el software de detección de objetos.

- Control del Robot (CNCRobot)

El primero de estos tiene como objetivo el control de los tres motores paso a paso que mueven los brazos del robot, este recibe el nombre de CNCRobot y utiliza las librerías AccelStepper.h y MultiStepper.h para controlar los motores. En este código inicialmente se crean tres objetos (uno por cada motor indicando los pines de control de los mismos) de la clase AccelStepper, luego se crea un objeto de la clase MultiStepper que contendrá los tres objetos de la clase AccelStepper creados anteriormente (Figura 44). Seguidamente se realizan algunas configuraciones iniciales, como establecer la velocidad máxima de los motores, configurar pines de entrada y salida, y establecer la posición actual de los motores.

```
#include <AccelStepper.h>
#include <MultiStepper.h>

// Definimos las constantes para los pines de control de cada motor
#define MOTOR1_STEP 2
#define MOTOR1_DIR 5
#define MOTOR2_STEP 3
#define MOTOR2_DIR 6
#define MOTOR3_STEP 4
#define MOTOR3_DIR 7
#define switchPinX 9
#define switchPinY 10
#define switchPinZ 11
const int emergencyPin = A0;

// Creamos objetos para cada motor con sus respectivos pines
AccelStepper stepper1(AccelStepper::DRIVER, MOTOR1_STEP, MOTOR1_DIR);
AccelStepper stepper2(AccelStepper::DRIVER, MOTOR2_STEP, MOTOR2_DIR);
AccelStepper stepper3(AccelStepper::DRIVER, MOTOR3_STEP, MOTOR3_DIR);

//Creamos un objeto de la clase multistepper
MultiStepper steppers;
```

Figura 44: Uso de las librerías de control de motores paso a paso y creación de objetos de clase AccelStepper y MultiStepper

En el bucle principal (loop), se espera recibir los comandos a través del puerto serial.Estos comandos deberán de ser en formato (posición del motor X,posición del motor Y, posición del motor Z, home, calibración).

Inicialmente se debe de realizar la calibración del robot para poderlo mover a una posición determinada al home.

Para realizar la calibración se debe enviar el comando (0,0,0,0,1) colocando cero en todos los parámetros a excepción del parámetro de calibración; una vez recibido este comando por el puerto serial se procederá a mover el brazo conectado al motor X del robot hacia atrás hasta que sea pulsado el final de carrera, justo en ese instante se marcará como cero dicha posición y luego se moverá hacia adelante hasta alcanzar el valor de posición definido como home; este proceso se repetirá para el motor Y y luego para el motor Z.

Una vez finalizado el proceso de calibración tendremos el robot colocado en la posición determinada como home.

Por otro lado, si se recibe por el puerto serial valores de posición en los motores y ceros en los parámetros de home y calibración, se procederá a desplazar el robot, siempre y cuando ya se haya realizado la calibración. El robot se moverá a la posición indicada en el comando, calculando las velocidades necesarias de cada motor para que los tres lleguen a la posición indicada al mismo tiempo.

Por último, si se recibe por el puerto serial el valor de 1 en el parámetro home y el valor de 0 en el de calibración indica que se desea mover al robot a la posición de home establecida, por lo que se mueven los tres motores para alcanzar dicha posición.

- Control de las cintas transportadoras y la pinza (CNCCintasYPinza):

El segundo código realizado tiene como objetivo el control de los dos motores paso a paso que mueven las cintas transportadoras y los dos servomotores.

En este código se utiliza la librería servo.h para facilitar el control de los servomotores. Inicialmente se definen los pines a los que están conectados los elementos, se inicializa la comunicación serial y se crean dos elementos de la clase Servo (Figura 45).

```
//Icluimos la libreria para mover los sevomotores
#include <Servo.h>

//Indicamos los pines conectados
const int StepX = 2;
const int DirX = 5;
const int StepY = 3;
const int DirY = 6;

const int emergencyPin = A0;

#define Pin_Mu 9 //limit x+
#define Pin_Pinza 10 //limit y+

//Creamos dos objeto de la clase Servo
Servo mu;
Servo pinz;
```

Figura 45: Uso de las librerías de control de servomotores y creación de objetos de clase Servo

En este, al igual que el anterior se espera recibir los comandos a través del puerto serial .Estos comandos deberán de ser en formato (grados a girar la muñeca, apertura(valor 270) o cierre de la pinza(valor 0), inicio o parada de cinta de entrada de piezas (1 ó 0 respectivamente),inicio o parada de cinta de desecho de piezas (1 ó 0 respectivamente)).

La cinta transportadora de entrada de objetos al área de trabajo, una vez inicializada, girará constantemente por un largo tiempo determinado en caso de que se desee que se detenga y se reciba dicho comando por el puerto serial. Por otro lado, la cinta transportadora de objetos desechados está configurada para que gire un número de vueltas específico para que permita la salida de la pieza del área de trabajo, sin embargo al igual que la otra cinta, esta puede detenerse si recibe el comando por el puerto serial.

Por otro lado, el control de los servos se hace de forma sencilla, para este inicialmente se crean dos objetos de la clase Servo y con el método .write() se le indica la posición en grados a la que debe girar. Para la pinza el valor de 0 indica que esta estará cerrada y 270 que estará abierta, y para la muñeca el valor en grados será la orientación que se desee para facilitar la recogida de los objetos.

Durante la realización de ambos códigos se contempló el caso de que sucediese una emergencia en uno de los dos arduinos, por lo que se incorporó un botón de tres posiciones que detiene todas las acciones que se encuentran ejecutándose si se mueve hacia el lado en el que se encuentra conectado el arduino en el cual está sucediendo la emergencia. Por otro lado, también se contempló el caso en que la emergencia sucediese en ambos arduinos por lo que se le incorporó un interruptor para cortar la corriente de ambos arduinos.

3.5.3.9. Menú de selección de características

Con la finalidad de facilitar al usuario la selección de características que deben de tener los objetos que se desean recoger, se realizó una interfaz gráfica utilizando la librería tkinter de python; con esta se crea la clase VentanaMenu que representa la interfaz gráfica de la aplicación.

Esta ventana (Figura 46) contiene opciones desplegables para seleccionar el color, la forma, la textura y el modo de funcionamiento, también hay etiquetas y botones que se utilizan para mostrar y confirmar las selecciones realizadas por el usuario.

Figura 46: Menú de Selección de características (VentanaMenu)

Dicha ventana tiene varios métodos; unos para confirmar la pulsación de los botones de enter (Figura 48), calibrado y home, otro para la verificación de la selección de forma (Figura 47) antes de la inicialización de la detección de objetos, otro de calibrado que utiliza la función conexionArduino descrita anteriormente para calibrar el robot, otra de home que también utiliza la función conexionArduino para mover el robot a la posición de home. Luego tenemos los métodos de ejecución, selección de elementos a mover, recogida de piezas, modo manual y modo automático.

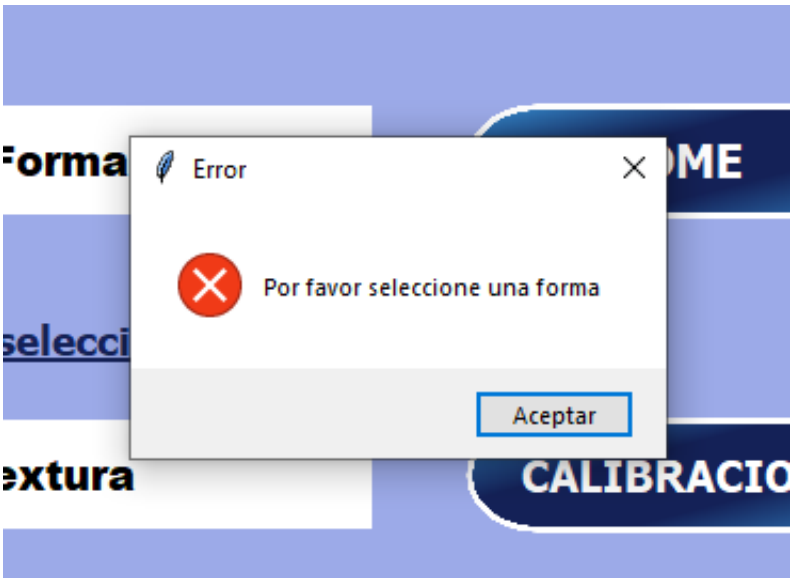


Figura 47: Ventana de error que indica que se debe de seleccionar una opción de forma para continuar

El método de ejecución se utiliza para procesar las selecciones introducidas por el usuario de características de los objetos que se desean recoger.

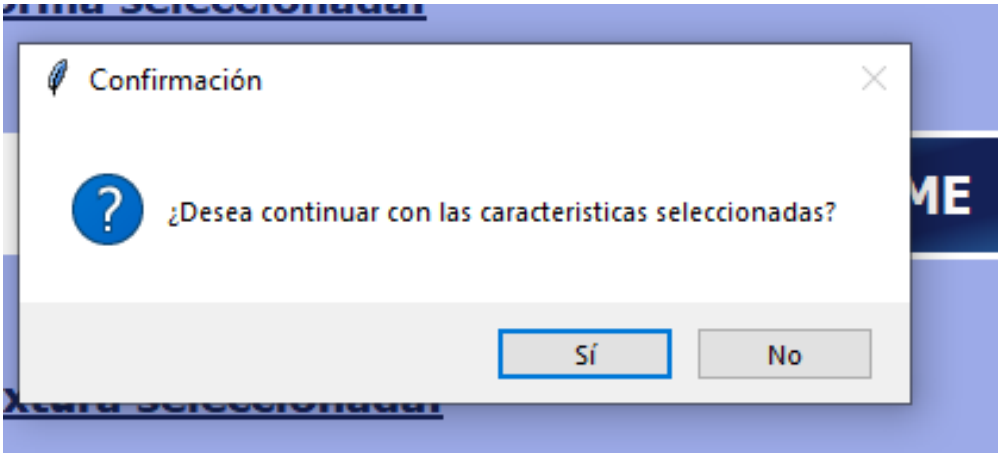


Figura 48: Ventana de confirmación de características seleccionadas

En este se obtienen las selecciones de forma y modo de funcionamiento introducidas, y luego se inicia la detección de objetos utilizando la función descrita anteriormente VideoFeedObjectDetection y la función GeneradorDeObjetos para detectar y extraer las características de los objetos que ingresan en el área especificada, luego se utiliza el método elementosAMover que evalúa si los objetos dentro del campo de vision cumplen con las características seleccionadas de color y textura, en caso negativo, indica por la terminal que no existen objetos con dichas características, y en caso positivo, devuelve un array que contiene los objetos que cumplen con todas las características seleccionadas. Luego de esto, siempre y cuando existan objetos de las características seleccionadas, se evalúa la selección del modo de funcionamiento, que puede ser manual o automático.

Si se seleccionó el modo manual se muestra un mensaje de verificación para saber si se desea o no recoger el objeto, en caso afirmativo llama al método mover_objetos el cual utiliza la función conexionArduino para realizar la recogida del objeto. Una vez finalizado el desplazamiento del objeto, se volverá a la ventana que contiene el menú de selección.

En el modo automático, a diferencia del modo manual, no se mostrará el mensaje de confirmación, sino directamente se utilizará la función mover_objetos para desplazar los objetos que cumplan con las características; una vez finalizado el desplazamiento automáticamente se volverá a iniciar la detección de objetos con las características que se introdujeron en un inicio y así sucesivamente hasta que se presione el botón ESC cuando se indique en la terminal luego de recoger los objetos.

4.Análisis y discusión de los resultados

4.1.Resultados obtenidos

4.1.1. Entrenamiento y rendimiento del modelo

Para analizar los resultados obtenidos en el entrenamiento del modelo personalizado podemos utilizar diferentes gráficas que fueron generadas en la creación del mismo y que nos aportan información acerca del rendimiento y estabilidad del modelo.

Una de las gráficas que podemos utilizar para medir el rendimiento es la que muestra el valor de F1 promedio para diferentes umbrales de confianza. El valor de F1 promedio se utiliza para evaluar la precisión y el recall (sensibilidad o tasa de verdaderos positivos) de un modelo de detección de objetos, lo que significa que tiene en cuenta tanto la cantidad de verdaderos positivos como de falsos positivos y falsos negativos. Mientras más alto sea el valor de F1 mayor precisión tendrá el modelo en la detección de objetos.

En la figura 49 podemos ver la gráfica de F1 en función de la confianza del modelo, como podemos observar, se tienen altos valores de F1 para valores de confianza mayores a 0.8 y se establece el punto que tenga mejor relación confianza-F1 para todas las clases a un 78% de confianza con un 0.92 de F1

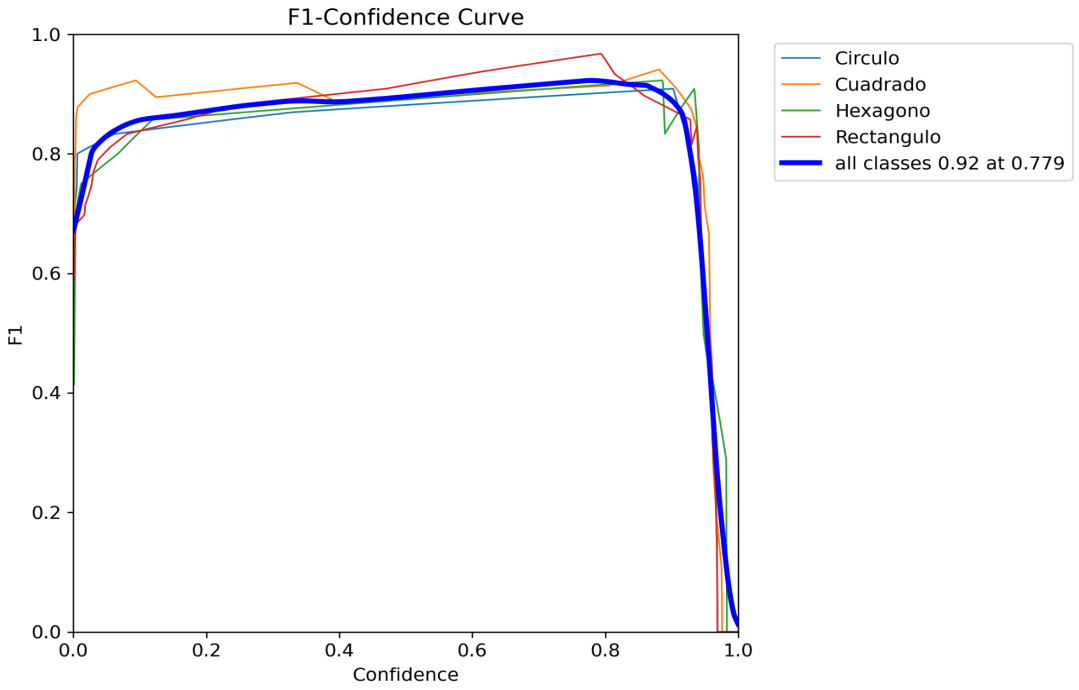


Figura 49: Gráfica F1 vs Confianza del modelo de YOLOv8 creado

Por otro lado tenemos la gráfica de Recall vs confianza; esta gráfica muestra el recall promedio (en términos de cuántas cajas de detección son verdaderos positivos) para diferentes umbrales de confianza. Esta información se puede utilizar para determinar el umbral de confianza óptimo para el modelo.El recall se expresa como un número entre 0 y 1, donde 1 indica un modelo con un alto recall que puede detectar todos los casos positivos en la imagen, y 0 indica un modelo con un recall nulo que no detecta ningún caso positivo.

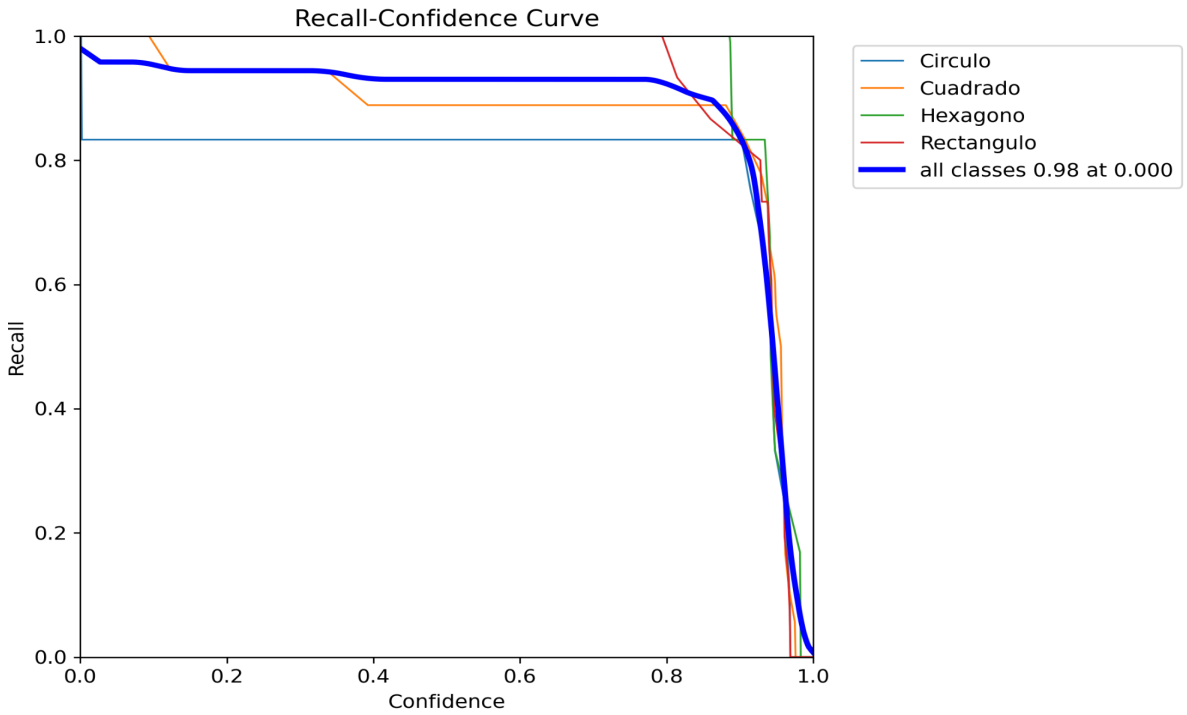


Figura 50: Gráfica de Recall vs Confianza del modelo de YOLOv8 creado

Como podemos observar en la figura 50 se obtuvo que para todas las clases el valor de recall promedio es de 0.98 lo que indica que el modelo puede detectar correctamente un gran porcentaje de los objetos relevantes en la imagen.

Además tenemos la gráfica de precisión vs confianza, en esta se representa la precisión del modelo en función de la confianza en la detección realizada. Como podemos observar en la figura 51, se obtuvieron altos valores de precisión para valores de confianza mayores al 80%, además de tener una precisión absoluta(valor de uno) a partir de un 96,4% de confianza

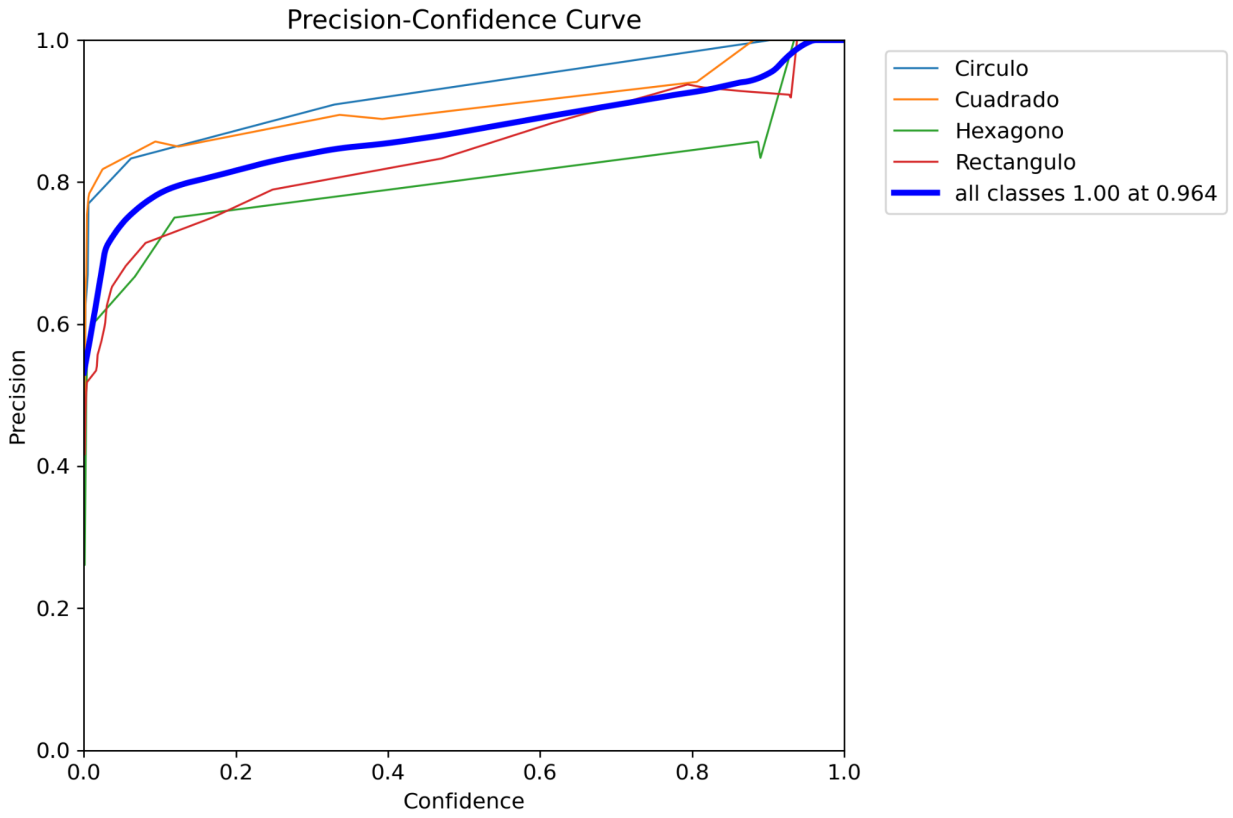


Figura 51: Gráfica Precisión vs Confianza del modelo de YOLOv8 creado

Por último, podemos utilizar la matriz de confusión normalizada obtenida para evaluar el modelo realizado; en esta podemos observar en el eje de las ordenadas las clases detectadas por el modelo y en el de las abscisas las clases reales de los objetos.

Cada cuadro según su color y el número que contiene en su interior nos indica qué porcentaje de objetos fue detectado de cierta clase (según lo indique el eje de las ordenadas) y la clase real a la que pertenece (según lo indique el eje de las abscisas).

Es decir, como vemos en la figura 52 para objetos de la clase círculo un 93% se detectó correctamente como círculo y un 7% tuvo una detección errónea como hexágono.

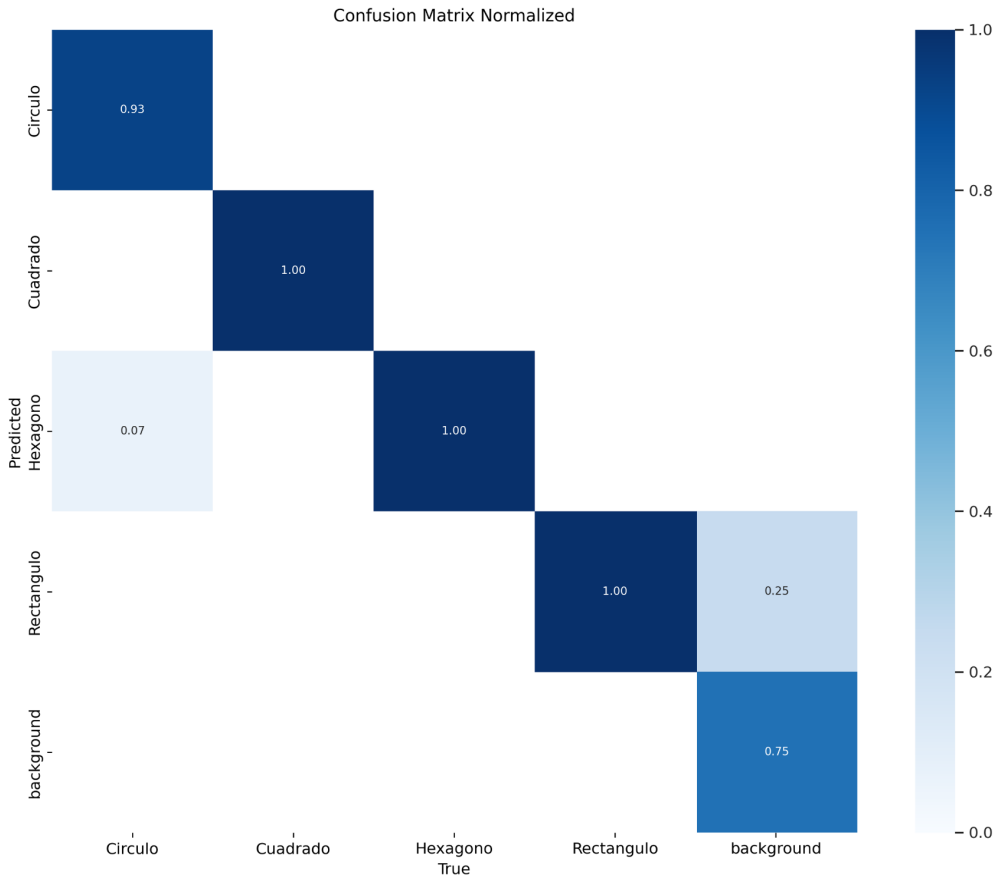


Figura 52: Matriz de confusión normalizada del modelo de YOLOv8 creado

A pesar de aun existir detecciones erróneas, podemos ver que el modelo es capaz de detectar, clasificar y segmentar objetos correctamente ya que la tasa de error en las detecciones es igual o inferior al 6,4%

Basándonos en los resultados obtenidos durante el entrenamiento del modelo YOLOv8, se puede concluir que este modelo ha logrado un rendimiento significativamente alto en la tarea de detección y segmentación de objetos. En particular, el modelo alcanzó un valor de F1 de 0.92 para un umbral de confianza de 0.799, lo que sugiere una capacidad bastante alta para detectar tanto los verdaderos positivos como los verdaderos negativos.

Además, se observó un valor de recall promedio de 0.98, lo que indica que el modelo es muy bueno para detectar todos los objetos de la imagen. Asimismo, el valor de precisión de 1 para una confianza de 0.946 muestra que el modelo es altamente preciso en la detección y segmentación de objetos.

Con una tasa de error del 6,4%, el modelo personalizado de YOLOv8 muestra una notable capacidad para minimizar los errores de clasificación. Esto implica que

el modelo tiene una alta precisión en la asignación correcta de clases a los objetos detectados.

En general, estos resultados sugieren que el modelo YOLOv8 puede ser altamente efectivo en la tarea de detección y segmentación de objetos y puede ser utilizado en una amplia gama de aplicaciones que requieren detección precisa y confiable de objetos en imágenes.

4.1.2. Elementos impresos en 3D

Con respecto a los elementos impresos en 3D se pudo observar que estos presentaron una calidad satisfactoria, evidenciando una adecuada precisión en la impresión y un acabado superficial adecuado.

Asimismo, se pudo constatar que la pinza desarrollada (Figuras 53 y 54) presentó una construcción adecuada, permitiendo un correcto ensamblaje de los servomotores en los soportes diseñados específicamente para estos dentro de la pinza, además de evidenciarse el correcto funcionamiento de los movimientos que debe de realizar. Por otro lado, la cinta transportadora (Figuras 55 y 56) demostró un adecuado desempeño en la tarea de desplazar los objetos, evidenciando una correcta sincronización en el movimiento de los mismos. Por último, la caja de soporte de los Arduinos tuvo una construcción adecuada permitiendo incorporar los componentes electrónicos en este (Figura 57).

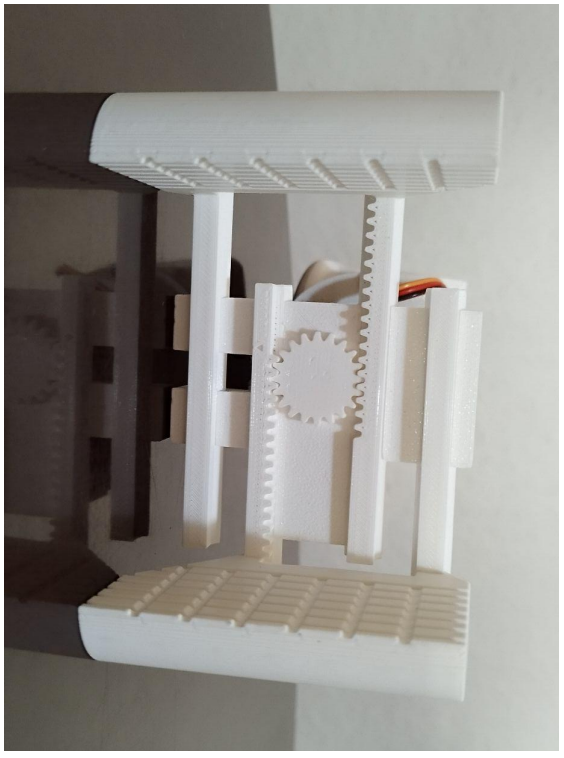


Figura 53: Pinza impresa en 3D (Vista inferior)

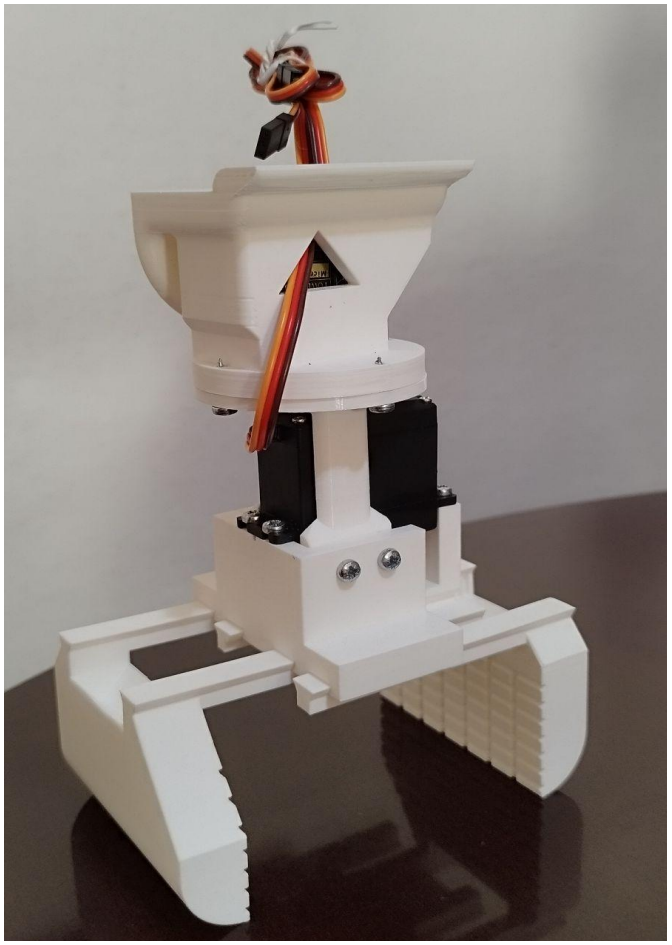


Figura 54: Pinza impresa en 3D (Vista lateral)

Figura 55: Cintas transportadoras impresas en 3D (vista lateral)

Figura 56: Cintas transportadoras impresas en 3D (vista superior)

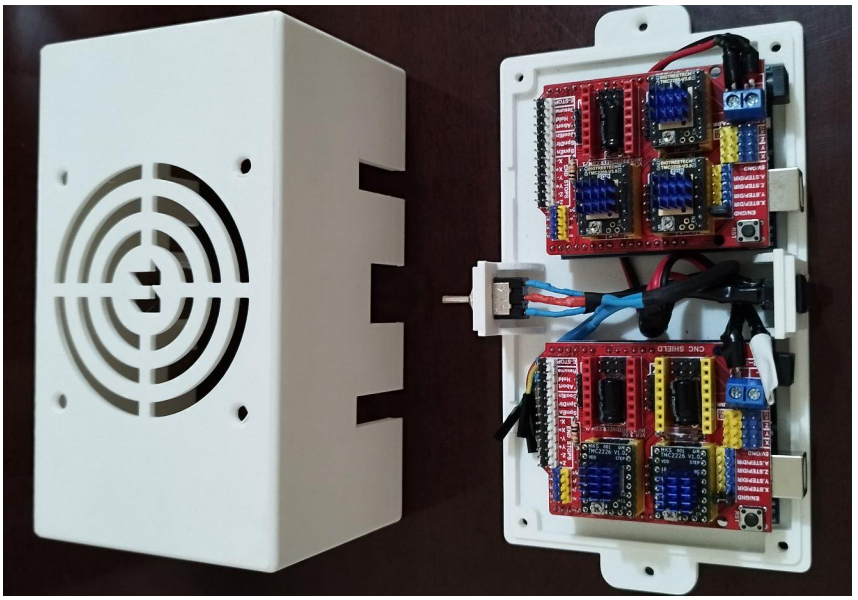


Figura 57: Caja de los Arduino impresa en 3D

4.2. Posibles aplicaciones y mejoras

El trabajo desarrollado ofrece múltiples aplicaciones prácticas y beneficios significativos para las fábricas industriales pequeñas, debido a su bajo costo y flexibilidad.

Una de las contribuciones de este trabajo es la fabricación de la pinza mediante impresión 3D, lo que permite que sea modificada fácilmente según las necesidades específicas de la aplicación. Esto brinda a las fábricas industriales

pequeñas la capacidad de adaptar la pinza a diferentes tamaños y formas de objetos, optimizando así sus procesos de manipulación y ensamblaje.

Otro aspecto importante es la utilización de una cámara de bajo costo incorporada en el sistema. Esta cámara proporciona capacidades de visión asequibles y eficientes, lo que resulta especialmente beneficioso para las fábricas industriales pequeñas que pueden tener recursos limitados. La capacidad de monitorear y detectar objetos con precisión a través de la cámara permite al robot delta realizar tareas de detección y manipulación de manera más efectiva y precisa.

Además, el uso de controladores Arduino Uno para el control de toda la electrónica del sistema es una elección estratégica en términos de costo y versatilidad. Los controladores Arduino Uno son reconocidos por ser económicos y ampliamente disponibles en el mercado. Esto implica que las fábricas industriales pequeñas pueden implementar y mantener el sistema a un costo razonable, sin comprometer la calidad o la funcionalidad.

En cuanto a las posibles mejoras, una significativa sería el poder incorporar un sistema de control del eje Z que permita al robot delta realizar movimientos en todos los planos de Z. Esto posibilitará la expansión del campo de visión de la cámara lo cual permitiría recoger piezas de un área más extensa.

Otra mejora o aplicación posible es el desarrollo de un nuevo modelo de detección de objetos que contenga una mayor cantidad de datos para realizar el entrenamiento. Aunque el modelo utilizado en este trabajo ha demostrado ser eficiente en la tarea de detección y recogida de piezas, existe la oportunidad de entrenar un modelo específico para detectar objetos particulares según las necesidades de la aplicación del usuario.

Por último, podemos destacar que otra posible mejora sería el cálculo de la cinemática inversa, teniendo en cuenta la relación entre los motores y las poleas que mueven los brazos del robot, lo que permitiría la realización de aplicaciones más complejas que involucren movimientos en múltiples planos y recogida de objetos de diferentes alturas.

4.3.Limitaciones

Durante el desarrollo del trabajo se identificaron dos limitaciones importantes que afectan el desempeño del robot delta en la tarea de recoger piezas de una cinta transportadora.

La primera de ella es la falta de un sistema de control del eje Z lo que limita significativamente las capacidades del robot delta en cuanto a su rango de movimientos. Esto significa que el robot sólo puede realizar movimientos dentro de un plano que se asemeja a una superficie de bol (ver figura 58).

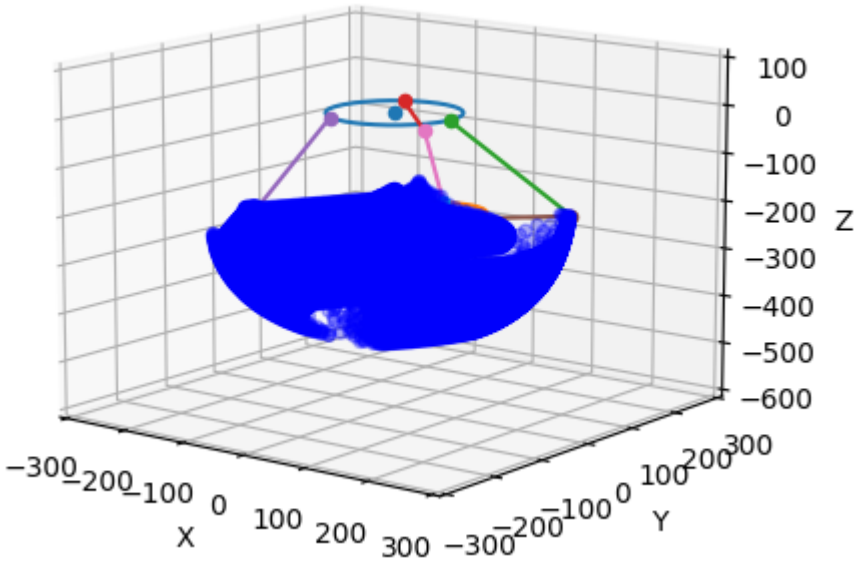


Figura 58: Representación del robot delta y del plano de movimientos del efector final (plano en color azul)

Debido a esta limitación, se debe restringir el área de visión del robot a una pequeña sección de la cinta transportadora para que este sea capaz de recoger los objetos de la cinta.

Para superar esta limitación, se definió una pequeña área de visión que abarca únicamente el ancho de la cinta transportadora(13 centímetros) y una longitud de 8 centímetros.

Esta restricción permite al robot delta recoger piezas que se encuentren dentro del área de la visión definida, sin embargo se observó que existía una inclinación de 12.07 grados de la pinza cuando se deseaba recoger objetos en el extremo de la

cinta transportadora, ya que en este punto la pinza empieza a describir el plano que se asemeja a un bol.

La segunda limitación encontrada se relaciona con la falta de un cálculo específico de cinemática inversa que tenga en cuenta la relación entre los motores y las poleas que mueven los brazos del robot. Sin embargo, gracias a la delimitación del área de visión y recolección de piezas, que se explicó anteriormente, se pudo calcular de forma manual los movimientos que debe realizar el robot para recoger las piezas en función de sus coordenadas X e Y (en mm).

Dicho cálculo manual implicó analizar la relación entre los movimientos de los motores y la posición de los brazos del robot, así como la ubicación de la base y de la pinza. A partir de esta información, se pudo determinar las posiciones que debe de tener el robot para realizar la recogida de objetos.

Aunque este proceso de cálculo manual permitió que el robot delta pudiera recolectar objetos con éxito, es importante destacar que es un enfoque limitado y que no es escalable para aplicaciones más complejas.

5.Presupuesto

CONCEPTO	PRECIO	UNIDADES	SUBTOTAL	TOTAL
COSTES MATERIALES				**198.80 €**
COMPONENTES ELECTRONICOS				**147.05 €**
CNC SHIELD V3	4.95 €	2	9.90 €	9.90 €
ARDUINO UNO R3	14.95 €	2	29.90 €	29.90 €
SERVO MG946 DE 270º	15.90 €	1	15.90 €	15.90 €
MINI SERVO DE 180º SG90	4.95 €	1	4.95 €	4.95 €
CABLES DUPONT 30cm	2.80 €	3	8.40 €	8.40 €
VENTILADOR 12V	5.95 €	1	5.95 €	5.95 €
CABLES DE CONEXIÓN ARDUINO	2.95 €	2	5.90 €	5.90 €
MOTOR PASO A PASO NEMA 17 CON DRIVER TMC 2226	20.98 €	2	41.96 €	41.96 €
FUENTE DE ALIMENTACIÓN 14-24V	21.99 €	1	21.99 €	21.99 €
INTERRUPTORES BIPOLARES Y UNIPOLARES	1.10 €	2	2.20 €	2.20 €
IMPRESIÓN 3D				**18.90 €**
FILAMENTO SAKATA PLA GO&PRINT 1.75mm 1kg	18.90 €	1	18.90 €	18.90 €
VISIÓN				**17.90 €**
WEBACAM TRUST TYRO	17.90 €	1	17.90 €	17.90 €
OTROS ELEMENTOS				**14.95 €**
TORNILLERIA	5.00 €	1	5.00 €	5.00 €
TELA ELÁSTICA (por metro)	14.00 €	0.5	7.00 €	7.00 €
CINTA ANTIDESLIZANTE	2.95 €	1	2.95 €	2.95 €
MADERA	5.90 €	3	17.70 €	17.70 €
COSTES DE EJECUCIÓN				**6,358.54 €**
IMPRESIÓN 3D				**358.54 €**
HORAS DE IMPRESIÓN 3D	3.94 €	91	358.54 €	358.54 €
TRABAJO DE INGENIERÍA				**6,000.00 €**
HORAS DE TRABAJO INGENIERO TECNICO INDUSTRIAL	20.00 €	300	6,000.00 €	6,000.00 €
			TOTAL	**6,557.34 €**

CONCEPTO		TOTAL
COSTES MATERIALES		198.80 €
COSTES DE EJECUCIÓN		6,358.54 €
TOTAL DE COSTES		**6,557.34 €**
GASTOS GENERALES	**13%**	852.45 €
BENEFICIO INDUSTRIAL	**6%**	393.44 €
SUMA DE G.G. Y B.I.		1,245.89 €
COSTE ESTIMADO		**7,803.23 €**
I.G.I.C.	**7%**	546.23 €
COSTE TOTAL DEL PROYECTO		**8,349.46 €**

EL COSTE TOTAL DEL PROYECTO ES DE OCHO MIL TRESCIENTOS NOVENTA Y CUATRO EUROS CON SESENTA Y UN CENTIMOS

6.Conclusiones

En este trabajo, se ha desarrollado un sistema que combina distintas tecnologías, como la impresión 3D, el *machine learning* y el *deep learning*, para automatizar la detección y segmentación de objetos en tiempo real utilizando un robot delta equipado con una cámara y una pinza como efector final. Para llevar a cabo dichas tareas se ha entrenado un modelo personalizado de YOLOv8 que permite la detección y segmentación de objetos geométricos desplazados en una cinta transportadora, y se han analizado propiedades como la forma, el color, la orientación y la textura de los objetos detectados.

En cuanto al análisis de la efectividad y precisión del modelo, se ha evaluado la tasa de detección y la tasa de falsos positivos, obteniendo resultados satisfactorios que demuestran la efectividad y precisión del modelo de detección implementado.

Por otro lado, se han propuesto posibles mejoras para el sistema, destacando la implementación de un sistema de control del eje Z para ampliar el campo de visión de la cámara lo que permitirá mejorar la aplicabilidad del sistema.

El sistema desarrollado ha demostrado que el uso de robots delta para la recolección de piezas en un entorno industrial es factible y puede ser una solución rentable para pequeñas fábricas industriales permitiéndoles aprovechar tecnologías avanzadas a un costo razonable, mejorando así su eficiencia y competitividad en el mercado.

La capacidad de personalización y el bajo costo de los componentes hacen que este sistema sea una opción atractiva para pequeñas empresas. La utilización de una pinza fabricada mediante impresión 3D permite su modificación según las necesidades específicas de la aplicación, y la incorporación de una cámara de bajo costo brinda capacidades de visión eficientes y asequibles. El control de toda la electrónica se realiza con controladores Arduino Uno, que son económicos y versátiles.

En general, podemos afirmar que la implementación de estas tecnologías puede mejorar significativamente la eficiencia y productividad en la industria, ya que permite la detección y segmentación de objetos en tiempo real, lo que facilita la monitorización de procesos y el control de calidad.

Este trabajo demuestra el potencial de la visión por computador y la robótica en la industria, y se espera que esta tecnología siga evolucionando y mejorando en el futuro, para proporcionar soluciones más eficientes y precisas en diferentes campos.

7.Conclusion

In this project, a system has been developed that combines different technologies, such as 3D printing, machine learning, and deep learning, to automate real-time detection and segmentation of objects using a delta robot equipped with a camera and a gripper as the end effector. To carry out these tasks, a custom YOLOv8 model has been trained to detect and segment geometric objects on a conveyor belt, and properties such as shape, color, orientation, and texture of the detected objects have been analyzed.

Regarding the effectiveness and accuracy of the model, the detection rate and false positive rate have been evaluated, obtaining satisfactory results that demonstrate the effectiveness and accuracy of the implemented detection model.

On the other hand, possible improvements for the system have also been proposed, highlighting the implementation of a Z-axis control system to expand the camera's field of view, which will improve the system's applicability.

The developed system has demonstrated that the use of delta robots for collecting pieces in an industrial environment is feasible and can be a cost-effective solution for small industrial factories, allowing them to take advantage of advanced technologies at a reasonable cost, thus improving their efficiency and competitiveness in the market.

The customization capability and low cost of the components make this system an attractive option for small businesses. The use of a gripper made through 3D printing allows modification according to specific application needs, and the incorporation of a low-cost camera provides efficient and affordable vision capabilities. The control of all electronics is carried out with Arduino Uno controllers, which are economical and versatile.

In general, we can say that the implementation of these technologies can significantly improve efficiency and productivity in the industry, allowing real-time object detection and segmentation, which facilitates process monitoring and quality control.

This work demonstrates the potential of computer vision and robotics in the industry, and it is expected that this technology will continue to evolve and improve in the future, providing more efficient and accurate solutions in different fields.

8.Bibliografía

1. https://www.innovacion-tecnologia.com/robotica/que-son-los-robots-delta-aplicaciones-y-precios/
2. https://www.edsrobotics.com/robots-industriales/
3. https://urany.net/blog/robots-delta-qu%C3%A9-c%C3%B3mo-cu%C3%A1ndo-y-por-qu%C3%A9#:~:text=En%201940%2C%20Willard%20Pollard%20present%C3%B3,estaban%20unidos%20mediante%20articulaciones%20universales
4. https://revistaderobots.com/robots-y-robotica/robot-delta-aplicaciones-y-precios
5. https://www.euroinnova.edu.es/blog/como-hacer-modelos-3d-para-imprimir
6. https://www.3dnatives.com/es/modelado-por-deposicion-fundida29072015/
7. https://formlabs.com/es/blog/fusion-360-tutorial-y-consejos-para-impresion-3d/
8. https://of3lia.com/ultimaker-cura/
9. https://aws.amazon.com/es/what-is/deep-learning/
10. https://blog.roboflow.com/whats-new-in-yolov8/
11. https://github.com/ultralytics/ultralytics
12. https://docs.roboflow.com/
13. https://docs.ultralytics.com
14. https://www.comunicacion-multimedia.info/2010/05/modos-o-modelos-de-color-hsb-o-hsv-y.html
15. https://www.comunicacion-multimedia.info/2010/05/modos-o-modelos-de-color-lab-e-indexado.html
16. Arturo de la Escalera, Visión por Computador: Fundamentos y Métodos, Ed.Pearson Education, 2001
17. Rafael C. González. Richard E. Woods. Digital Image Processing (second edition). Ed. Prentice Hall, 2002.
18. https://www.tinkercad.com/3d-design
19. https://www.3dslash.net/features_product.php
20. https://www.xyzprinting.com/es-ES/software-series/DESIGN/xyzmaker-suite
21. http://www.artofillusion.org/
22. http://www.k-3d.org/
23. https://www.prusa3d.com/es/pagina/prusaslicer_424/
24. https://www.raise3d.com/ideamaker/
25. https://icesl.loria.fr/features/
26. https://grid.space/kiri/
27. https://artillery3d.es/artillery-genius-pro/
28. https://www.handsontec.com/dataspecs/cnc-3axis-shield.pdf
29. https://c-3d.niceshops.com/upload/file/TMC2226_V1.0_Instruction_Manual.pdf

30. https://grobotronics.com/images/companies/1/datasheets/TMC2208-V3.0%20specification%20.pdf?1560521092408
31. http://www.airspayce.com/mikem/arduino/AccelStepper/classMultiStepper.html
32. http://www.airspayce.com/mikem/arduino/AccelStepper/classAccelStepper.html
33. https://docs.opencv.org/4.x/index.html
34. https://scikit-image.org/docs/stable/api/api.html
35. https://docs.python.org/3/library/tk.html

9.Anexos

Funciones de Python:

Datasheets:

Funciones de Arduino IDE:

menu.py

```
import tkinter as tk
import cv2
import numpy as np
from tkinter import messagebox, ttk
import time
import keyboard
import serial

#importamos los archivos
import GeneradorDeObjetos
import VideoFeedObjectDetection
import conexionArduino

class VentanaMenu:
    def __init__(self, calibrado,cap,ser1,ser2):
        self.calibrado = calibrado
        self.cap=cap
        self.ser1=ser1
        self.ser2=ser2
        colorOpciones="#98c3ed"
        colorFondo= "#9CAAE8"
        colorSelec="#2d2c55"
        colorLetrasSelec ="#142157"
        colorFondoSelec="white"
        #creamos la ventana
        self.ventana = tk.Tk()
        #ponemos titulo a la ventana
        self.ventana.title("Menu de deteccion/recogida de piezas")
        #definimos el tamaño de la ventana
        self.ventana.geometry('950x500')
        #definimos el color del fondo de la ventana
        self.ventana.configure(background=colorFondo)

        #creamos una lista variables con opciones de colores
        self.var = tk.StringVar(self.ventana)
        #valor inicial de la variable de tipo string
        self.var.set('Color')
        #lista de opciones disponibles
        self.opciones = ['blanco', 'naranja', 'amarillo', 'verde','violeta','rosa'
                         ,'azul','rojo']

        #creamos otra lista variables 1 con opciones para las formas
        self.var1 = tk.StringVar(self.ventana)
        #valor inicial de la variable de tipo string
        self.var1.set('*Forma')
        #lista de opciones disponibles
        self.opciones1 = ['Rectangulo', 'Cuadrado', 'Circulo', 'Hexagono']

        #creamos otra lista variables 2 con opciones para las texturas
        self.var2 = tk.StringVar(self.ventana)
        #valor inicial de la variable de tipo string
        self.var2.set('Textura')
        #lista de opciones disponibles
        self.opciones2 = ['liso', 'aspero']

        #creamos otra lista variables 3 con opciones para las texturas
```

```
        self.var3 = tk.StringVar(self.ventana)
        #valor inicial de la variable de tipo string
        self.var3.set('Manual')
        #lista de opciones disponibles
        self.opciones3 = ['Auto', 'Manual']

        #Creamos el menu de Modo de funcionamiento y lo configuramos
        img_inputDisplay = tk.PhotoImage(file="inputDisplay.png")
        img_inputDisplay = img_inputDisplay.subsample(12) # reducir tamaño imagen


        self.opcion3 = tk.OptionMenu(self.ventana, self.var3, *self.opciones3)
        self.opcion3.config(font=('Tahoma', 16,"bold"),image=img_inputDisplay,
                            compound=tk.CENTER, fg="white",bg=colorFondo,
                            borderwidth=0,activebackground=colorFondo,
                            highlightthickness=0,indicatoron=0,
                            activeforeground="white" )
        self.opcion3['menu'].config(font=('Tahoma', 14, "bold"))
        self.opcion3.grid(row=2, column=3, padx=0,pady=10,sticky="nsew")



        #Creamos el menu de Color y lo configuramos

        img_Display = tk.PhotoImage(file="display.png")
        img_Display = img_Display.subsample(11) # reducir tamaño de la imagen

        self.opcion = tk.OptionMenu(self.ventana, self.var, *self.opciones)
        self.opcion.config(font=('Tahoma', 17, "bold"),image=img_Display,
                           compound=tk.CENTER, highlightthickness=0,indicatoron=0,
                           fg="white",bg=colorFondo,borderwidth=0,
                           activebackground=colorFondo)
        self.opcion['menu'].config(bg=colorOpciones, font=('Tahoma', 12, "bold"),
                                   activebackground=colorFondoSelec)
        self.opcion.grid(row=2, column=1, padx=20,pady=0,sticky="nsew")

        #Creamos el menu de forma y lo configuramos
        self.opcion1 = tk.OptionMenu(self.ventana, self.var1, *self.opciones1)
        self.opcion1.config(font=('Tahoma', 17, "bold"),image=img_Display,
                            compound=tk.CENTER, highlightthickness=0,indicatoron=0,
                            fg="white",bg=colorFondo,borderwidth=0,
                            activebackground=colorFondo)
        self.opcion1['menu'].config(bg=colorOpciones, font=('Tahoma', 12, "bold"),
                                    activebackground=colorFondoSelec)
        self.opcion1.grid(row=4, column=1, padx=20,pady=0, sticky="nsew")
        self.opcion1.bind('<Return>', self.validar_forma)

        #Creamos el menu de texturas y lo configuramos
        self.opcion2 = tk.OptionMenu(self.ventana, self.var2, *self.opciones2)
        self.opcion2.config(font=('Tahoma', 17, "bold"),image=img_Display,
                            compound=tk.CENTER, highlightthickness=0,indicatoron=0,
                            fg="white",bg=colorFondo,borderwidth=0,
                            activebackground=colorFondo)
        self.opcion2['menu'].config(bg=colorOpciones, font=('Tahoma', 12, "bold"),
                                    activebackground=colorFondoSelec)
        self.opcion2.grid(row=6, column=1, padx=20,pady=0, sticky="nsew")

        #tag de caracteristicas
        self.el = tk.Label(self.ventana, text="Caracteristicas:",bg=colorFondo,
                           fg=colorLetrasSelec,width=20,
```

```
                    font=("Tahoma", 14, "bold",'underline'))
        self.el.grid(row=1, column=1, padx=0,pady=0)


        img_seleccion = tk.PhotoImage(file="seleccion.png")
        img_seleccion = img_seleccion.subsample(13) # reducir tamaño de la imagen

        #tag del color que hemos seleccionado
        self.el = tk.Label(self.ventana, text="Color seleccionado:",bg=colorFondo,
                           fg=colorLetrasSelec,width=20,
                           font=("Tahoma", 14, "bold",'underline'))
        self.el.grid(row=1, column=2, padx=20,pady=0)

        #recuadro que muestra la variable seleccionada
        self.color = tk.Label(self.ventana,textvariable=self.var, padx=20, pady=10,
                              width=20,font=("Arial Black", 14, "bold"),
                              background=colorFondoSelec)
        self.color.grid(row=2, column=2, padx=20,pady=20)

        #tag de la forma que hemos seleccionado
        self.el1 = tk.Label(self.ventana, text="Forma seleccionada:", bg=colorFondo,
                            fg=colorLetrasSelec,
                            font=("Tahoma", 14, "bold",'underline'))
        self.el1.grid(row=3, column=2, padx=20,pady=0)

        #recuadro que muestra la variable forma seleccionada
        self.forma = tk.Label(self.ventana,textvariable=self.var1, padx=20, pady=10,
                              width=20,font=("Arial Black", 14, "bold"),
                              background=colorFondoSelec )
        self.forma.grid(row=4, column=2, padx=20,pady=0)

        #tag de la textura que hemos seleccionado
        self.el2 = tk.Label(self.ventana, text="Textura seleccionada:",bg=colorFondo,
                            fg=colorLetrasSelec,width=20,
                            font=("Tahoma", 14, "bold",'underline'))
        self.el2.grid(row=5, column=2, padx=20,pady=0)

        #recuadro que muestra la variable forma seleccionada
        self.tex = tk.Label(self.ventana,textvariable=self.var2, padx=20, pady=10,
                            width=20,font=("Arial Black", 14, "bold"),
                            background=colorFondoSelec)
        self.tex.grid(row=6, column=2, padx=20,pady=0)

        self.mod = tk.Label(self.ventana, text="Modo de Funcionamiento:",
                            bg=colorFondo, fg=colorLetrasSelec,width=20,
                            font=("Tahoma", 14, "bold",'underline'))
        self.mod.grid(row=1, column=3,padx=0,pady=0)

        img_boton = tk.PhotoImage(file="enter.png")
        img_boton = img_boton.subsample(10) # reducir tamaño de la imagen

        #boton enter
        self.enter = tk.Button(self.ventana, text="OK", command=self.validar_forma,
                               font=("Tahoma", 18, "bold"),fg="white",
                               image=img_boton,bg=colorFondo,compound=tk.CENTER,
                               borderwidth=0,activebackground=colorFondo)
        self.enter.grid(row=8, column=4, padx=0,pady=0)

        #boton home
        self.home = tk.Button(self.ventana, text="HOME", command=self.confirmar_home,
```

```
                            font=("Tahoma", 16,"bold"),fg="white",bg=colorFondo,
                            image=img_seleccion,compound=tk.CENTER,borderwidth=0,
                            activebackground=colorFondo)
        self.home.grid(row=4, column=3, padx=0,pady=40,sticky="nsew")

        #boton Calibracion
        self.calib = tk.Button(self.ventana, text="CALIBRACION",
                            command=self.confirmar_calibrado,
                            font=("Tahoma", 15,"bold"),fg="white",bg=colorFondo,
                            image=img_seleccion,compound=tk.CENTER,borderwidth=0,
                            activebackground=colorFondo)
        self.calib.grid(row=6, column=3, padx=0,pady=20,sticky="nsew")

        self.ventana.mainloop()



    def ejecucion(self):

        #obtenemos la seleccion de forma y de modo
        formaSeleccionada=self.var1.get()
        modo=self.var3.get()

        #asignamos el numero de la clase correspondiente a la forma a detectar
        if formaSeleccionada =="Rectangulo":
            objADetectar=3
            print('entro a Rectangulo ')
        if formaSeleccionada =="Cuadrado":
            objADetectar=1
            print('entro a Cuadrado ')
        if formaSeleccionada =="Circulo":
            objADetectar=0
            print('entro a Circulo ')
        if formaSeleccionada =="Hexagono":
            objADetectar=2
            print('entro a Hexagono ')


        #iniciamos el video para detectar objetos
        captura=VideoFeedObjectDetection.VideoFeedObjectDetection(objADetectar,
                                                                  self.cap,self.ser1)

        #procesamos un frame para obtener la informacion de los objetos detectados
        detecciones = GeneradorDeObjetos.GeneradorDeObjetos(captura)

        #mientras exista problemas con la captura
        while detecciones==None:
            print('Hubo un error en la captura del frame, se volvera a hacer una captura')
            capturaError=VideoFeedObjectDetection.reintentarCaptura(self.cap)
            detecciones = GeneradorDeObjetos.GeneradorDeObjetos(capturaError)

        #cuando existan objetos detectados en la captura
        if detecciones != None and not keyboard.is_pressed('esc'):
            print("Numero de Objetos detectados:")
            print(len(detecciones))

            cv2.imshow('Captura realizada',captura)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
```

```
            #evaluamos cuantos y cuales son los elementos a mover
            piezasARecoger=self.elementosAMover(objADetectar, detecciones)

            #verificamos que existen piezas a recoger
            if len(piezasARecoger) == 0:
                print("No hay elementos con las caracteristicas de color y textura
                      seleccionadas")
            else:
                if modo=="Manual":
                    self.modo_manual(piezasARecoger)
                if modo == "Auto":
                    self.modo_auto(piezasARecoger)


    #genera un array con los objetos de las caracteristicas especificadas
    def elementosAMover(self,objADetectar, detecciones):

        #obtenemos lo seleccionado para color y textura
        colorSeleccionado=self.var.get()
        texturaSeleccionada=self.var2.get()
        piezasARecoger =[]
        piezasFormas=[]
        piezasColor=[]

        for pieza in detecciones:
            pieza.showMascaraColorTexturaForma()
            print("Orientacion Pieza: ", pieza.orientacion)
            if objADetectar == pieza.clase:
                piezasFormas.append(pieza)

        if colorSeleccionado != "Color":
            for pieza in piezasFormas:
                if colorSeleccionado == pieza.color:
                    piezasColor.append(pieza)
        else:
            piezasColor=piezasFormas

        if texturaSeleccionada != "Textura":
            for pieza in piezasColor:
                if texturaSeleccionada == pieza.textura:
                    piezasARecoger.append(pieza)
        else:
            piezasARecoger=piezasColor

        return piezasARecoger


    #mueve los objetos especificados
    def mover_objetos(self,piezasARecoger):
        for pieza in piezasARecoger:
            #Recogemos el ojeto, lo llevamos a la cinta desechos y regresamos a home
            conexionArduino.conexionArduino(pieza.centrox,pieza.centroy,0,0,
                                            self.ser2,pieza.orientacion,self.ser1)
            print('se ha movido el objeto escogido')


    #confirmacion de movimiento manual
    def modo_manual(self,piezasARecoger):
        for elementos in piezasARecoger:
           elementos.printBBObj()
```

```
        confir=messagebox.askyesno("Confirmación de recogida", "¿Desea desplazar los
                                   objetos con las cacacteristicas seleccionadas?")
        if confir:
            self.mover_objetos(piezasARecoger)
            messagebox.showinfo('Info','Moviendo el objeto')

    def modo_auto(self,piezasARecoger):
        for elementos in piezasARecoger:
           elementos.printBBObj()
        #self.mover_objetos(piezasARecoger)
        print("Para salir del modo auto presione ESC")
        time.sleep(2)
        while not keyboard.is_pressed('esc'):
            time.sleep(3) #esperar 3 segundos para una nueva ejecucion
            self.ejecucion()


    #validar que se ha seleccionado la caracteristica de forma
    def validar_forma(self):
        #Verifica si se ha seleccionado una forma
        if self.var1.get() == '*Forma':
            messagebox.showerror('Error', 'Por favor seleccione una forma')
            self.ventana.focus() # vuelve a enfocar la ventana principal
            return "break"
        # si se ha seleccionado una forma, continúa con la siguiente función
        self.confirmar_enter()


    #confirmar las caracteristicas seleccionadas
    def confirmar_enter(self):
        confirmacion=messagebox.askyesno("Confirmación", "¿Desea continuar con las
                                         caracteristicas seleccionadas?")
        if confirmacion:
            if self.calibrado == 0:
                self.calibrado = self.calibrar()
        self.ventana.destroy()
        self.ejecucion()
        VentanaMenu(self.calibrado,self.cap,self.ser1,self.ser2)

    def confirmar_calibrado(self):
        confirmacion=messagebox.askyesno("Confirmación", "¿Desea Calibrar?")
        if confirmacion:
            calibrado=self.calibrar()
            print("calibrado es :", calibrado)
            return self.calibrado

    def confirmar_home(self):
        confirmacion=messagebox.askyesno("Confirmación", "¿Desea mover a Home?")
        if confirmacion:
            self.irHome()


    def calibrar(self):
        print("Calibrando...")
        #Calibramos el robot para dejarlo en home
        conexionArduino.conexionArduino(0,0,0,1,self.ser2,0,self.ser1)
        print("Calibrado correcto")
        self.calibrado = 1
        return self.calibrado
```

```
    def irHome(self):
        print("Moviendo a home...")
        conexionArduino.conexionArduino(0,0,1,0,self.ser2,0,self.ser1)
        print("Se ha posicionado en home")

if __name__ == '__main__':
    calibrado=0
    print("abriendo captura...")
    #iniciamos captura en la webcam
    cap = cv2.VideoCapture(1) #0 webcam incorporada, 1 webcam externa
    print("captura abierta...")
    print("abriendo comunicacion serial...")

    #Abrir conexion con las cintas y pinzas al COM6(malo) en ser1
    ser1 =  serial.Serial("COM6",9600)
    time.sleep(2)

    #Abrir conexion con el Robot (CNC) conectado al COM4(bueno) en ser2
    ser2 = serial.Serial("COM4", 9600)
    time.sleep(2)

    print("comunicacion serial abierta...")


    ventana_menu = VentanaMenu(calibrado,cap,ser1,ser2)

    #cerramos conexiones
    ser1.close()
    time.sleep(2)
    ser2.close()
    print("Conexiones cerradas")
```

VideoFeedObjectDetection.py

```python
from ultralytics import YOLO
from ultralytics.yolo.v8.detect.predict import DetectionPredictor
import torch
import cv2, time
import numpy as np
import conexionArduino

def VideoFeedObjectDetection(objADetectar,cap,ser1):

    print("cargando el modelo...")
    #Cargamos el modelo
    model = YOLO("C:/Users/paula/Desktop/Python Finales/TFGSmall/weights/best.pt")

    print("modelo cargado....")

    # Definir las coordenadas de la región a recortar
    x1, y1 = 22, 36
    x2, y2 = 541, 419
    x11, y11 = 34, 157
    x22, y22 = 543, 427

    tiempo_espera = 2 # tiempo de espera en segundos
    tiempo_inicio = time.time()
    print("moviendo cinta")
    conexionArduino.conexionArduinoCintasYPinza(0,1,1,0,ser1)
    print("cinta se mueve")

    while True:
        ret, frame = cap.read()
        frame = frame[y1:y2, x1:x2]
        #verificamos si se abrio la camara correctamente
        if not ret:
            print("Error al iniciar la cámara")
            break
        #aplicamos el modelo
        results = model.predict(source=[frame], conf=0.45, save=False, show=True)
        #creamos un tensor vacio para almacenar las clases
        boxescls = torch.Tensor()
        #vamos llenando el tensor con las clases
        for result in results:
            boxescls = result.boxes.cls
        if objADetectar in boxescls:
            print("_Se ha detectado un objeto")

            if objADetectar != 2:
                tiempo_espera -= time.time() - tiempo_inicio
                tiempo_inicio = time.time()

                while tiempo_espera > 0:
                    ret, frame = cap.read()
                    frame = frame[y11:y22, x11:x22]
                    tiempo_espera -= time.time() - tiempo_inicio
                    tiempo_inicio = time.time()

            #Detenemos la cinta luego de detectar el objeto y pasado el tiempo de espera
            conexionArduino.conexionArduinoCintasYPinza(0,1,0,0,ser1)
```

```
            ret, frame = cap.read()
            frame = frame[y11:y22, x11:x22]
            print('captura post time')
            captura = np.array(frame)
            break

        #cerrar si se presiona escape
        t = cv2.waitKey(5)
        if t == 27:
            break




    return captura

def reintentarCaptura(cap):
    x11, y11 = 34, 157
    x22, y22 = 543, 427
    ret, frame = cap.read()
    frame = frame[y11:y22, x11:x22]
    captura = np.array(frame)
    print('Se realizo una nueva captura')
    return captura
```

GeneradorDeObjetos.py

```
#Aqui creamos la clase Objeto y definimos la función que genera el array con los
#objetos tipo objetos detectados

import cv2
import numpy as np
from skimage.measure import regionprops

#importamos los archivos
import obtenerColor
import ImgSegmentacion
import recortarImgparaTexturas
import obtenerTexturas
import obtenerOrientaciones

#Clase objeto
class Objeto:
        #Atributos:
        def __init__(self, imagen, mascara, clase, centrox, centroy, xmin, xmax, ymin,
                     ymax,anchoBB, altoBB, orientacion, color, textura,energy,
                     homogeneity,dissimilarity,contrast):
            self.imagen = imagen
            self.mascara = mascara
            self.clase = clase
            self.centrox = centrox
            self.centroy = centroy
            self.xmin = xmin
            self.xmax = xmax
            self.ymin = ymin
            self.ymax = ymax
            self.anchoBB = anchoBB
            self.altoBB = altoBB
            self.orientacion = orientacion
            self.color = color
            self.textura = textura
            self.energy=energy
            self.homogeneity=homogeneity
            self.dissimilarity=dissimilarity
            self.contrast=contrast

        #metodos:
        def printBBObj(self):
            color = (0, 0, 255)
            thickness = 2
            cv2.rectangle(self.imagen, (int(self.xmin), int(self.ymin)),
                          (int(self.xmax), int(self.ymax)), color, thickness)
            cv2.imshow('Imagen con Bounding Box', self.imagen)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

        def printMascara(self):
             cv2.imshow('Mascara',self.mascara)
             cv2.waitKey(0)
             cv2.destroyAllWindows()

        def saveMascara(self):
             cv2.imwrite("Mascara.jpg",self.mascara)
```

```
    def printInfo(self):
        print('Nro de clase:')
        print(int(self.clase))
        print('centro en x:')
        print(int(self.centrox))
        print('centro en y:')
        print(int(self.centroy))
        print('xmin:')
        print(int(self.xmin))
        print('xmax:')
        print(int(self.xmax))
        print('ymin:')
        print(int(self.ymin))
        print('ymax:')
        print(int(self.ymax))
        print('ancho Bounding Box:')
        print(int(self.anchoBB))
        print('alto Bounding Box:')
        print(int(self.altoBB))
        print('Orientacion')
        print(self.orientacion)
        print('color:')
        print(self.color)
        print('textura:')
        print(self.textura)

    def showOrientacion(self):
        centro_x, centro_y = self.centrox, self.centroy
        orientacion = self.orientacion
        orientacion = np.radians(orientacion)
        # Dibujar la línea que representa la orientación del objeto
        punto_final = (int(centro_x + 50*np.sin(orientacion)),
                       int(centro_y + 50*np.cos(orientacion)))
        cv2.line(self.mascara, (int(centro_x), int(centro_y)),
                 punto_final, (0, 0, 255), 2)

        # Dibujar la línea perpendicular en color azul a la línea ya dibujada
        vector_perpendicular = np.array([np.cos(orientacion), -
                                         np.sin(orientacion)]) * 50
        punto_final_perpendicular = (int(centro_x + vector_perpendicular[0]),
                                     int(centro_y + vector_perpendicular[1]))
        cv2.line(self.mascara, (int(centro_x), int(centro_y)),
                 punto_final_perpendicular, (255, 0, 0), 2)

        cv2.imshow("Orientaciones", self.mascara)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    def showRecorteTextura(self):
        color = (0, 0, 255)
        thickness = 2
        xminima=int(self.centrox -40)
        xmaxima=int(self.centrox+ 40)
        yminima=int(self.centroy-40)
        ymaxima=int(self.centroy+ 40)
        cv2.rectangle(self.imagen, (xminima, yminima), (xmaxima, ymaxima),
                      color, thickness)
        cv2.imshow('Imagen con recorte',self.imagen)
        cv2.waitKey(0)
```

```
            cv2.destroyAllWindows()

        def showMascaraColorTexturaForma (self):
            cv2.putText(self.mascara, "Textura: " + self.textura, (10, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.putText(self.mascara, "Color: " + self.color, (10, 60),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            forma=self.forma()
            cv2.putText(self.mascara, "Forma: " + forma, (10, 90),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.imshow('Mascara',self.mascara)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

        def forma(self):
            if(self.clase == 0):
                shape = 'Circulo'
            if(self.clase == 1):
                shape = 'Cuadrado'
            if(self.clase == 2):
                shape = 'Hexagono'
            if(self.clase == 3):
                shape = 'Rectangulo'
            return shape


    def GeneradorDeObjetos(img):

        #aplicamos el modelo en la imagen y guardamos los resultados
        boxescls,boxesxyxy,boxesxywh,resultados,clases=ImgSegmentacion.ImgSegmentacion(img)

        if boxescls != None:
            #resize de la img para guardarla en el objeto de clase objeto
            resImg = cv2.resize(img, dsize=(640,448),interpolation=cv2.INTER_CUBIC)

            #creamos dos matrices vacias para guardar colores y texturas
            colores=[]
            texturas=[]
            orientaciones=[]
            energy=[]
            homogeneity=[]
            dissimilarity=[]
            contrast=[]


            #buscamos las características de los objs segmentados y vamos llenando
            # las matrices colores y texturas
            for elementos, box, boxx in zip(resultados, boxesxywh, boxesxyxy):
                orienta = obtenerOrientaciones.obtenerOrientaciones(elementos)
                color = obtenerColor.obtenerColor(elementos)
                rec=recortarImgparaTexturas.recortarImgparaTexturas(elementos,boxx,box)
                textura,energ,homogeneit,dissimilarit,contras=obtenerTexturas.obtenerTexturas(rec)
                if(orienta > 0):
                    orienta = 180-orienta
                else:
                    orienta = -orienta

                colores.append(color)
                texturas.append(textura)
                orientaciones.append(orienta)
```

```
            energy.append(energ)
            homogeneity.append(homogeneit)
            dissimilarity.append(dissimilarit)
            contrast.append(contras)


        #creamos una matriz para guardar todos los objetos de tipo objeto
        detecciones=[]

        #vamos creando objetos y añadiendolos a la matriz detecciones
        for i in range(boxescls.numel()):
            objeto = Objeto(resImg,resultados[i],boxescls[i],  boxesxywh[i][0],
                            boxesxywh[i][1], boxesxyxy[i][0], boxesxyxy[i][2],
                            boxesxyxy[i][1], boxesxyxy[i][3], boxesxywh[i][2],
                            boxesxywh[i][3], orientaciones[i], colores[i],
                            texturas[i],energy[i], homogeneity[i], dissimilarity[i],
                            contrast[i])
            detecciones.append(objeto)

        #detecciones es un array de elementos de clase objeto
    else:
        detecciones=None
    return detecciones
```

ImgSegmentacion.py

```
#Esta función aplica el modelo en la imagen y devuelve información de la clase, ubicacion
espacial
# y máscaras de cada uno de los objetos detectados

from ultralytics import YOLO
import cv2
import numpy as np




def ImgSegmentacion(img):

    #cargamos el modelo
    model = YOLO("C:/Users/paula/Desktop/Python Finales/TFGFXlarge/weights/best.pt")


    #le cambiamos las dimensiones a la imagen
    resImg = cv2.resize(img, dsize=(640,448),interpolation=cv2.INTER_CUBIC)

    #aplicamos el modelo
    results = model.predict(source=resImg, save=False,conf=0.7)
    clases = model.model.names

    #vamos colocando los datos obtenidos en tensores
    for result in results:
        boxescls = result.boxes.cls     #clase del objeto reconocido
        boxesxyxy = result.boxes.xyxy #xmin, ymin, xmax, ymax (una fila por cada objeto)
        boxesxywh = result.boxes.xywh #centro x, centro Y, ancho, alto
        if boxescls.numel() != 0:
            mask = result.masks.data       # mascaras (0 no obj, 1 obj)



    if boxescls.numel() != 0:
        #creo un array de numpy vacío para asilar los objetos segun su máscara
        objetos_aislados = mask.numpy()

        #crear un array y una matriz vacios
        objetocolor = np.zeros_like(resImg)
        resultados = []


        for o in range(len(objetos_aislados)):
            for i in range(len(objetos_aislados[o])):
                for j in range(len(objetos_aislados[o][i])):
                    if(objetos_aislados[o][i][j] == 1):
                        objetocolor[i][j] = resImg[i][j]
            resultados.append(objetocolor)
            objetocolor = np.zeros_like(resImg)

    if boxescls.numel() == 0:
        boxescls=None
        boxesxyxy=None
        boxesxywh=None
        resultados=None
```

```
        return boxescls, boxesxyxy, boxesxywh, resultados, clases
```

obtenerColor.py

```python
import cv2
import numpy as np

def obtenerColor(img):

    #CONVERTIR LA IMAGEN A HSV Y LAB
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    # Definir los rangos de color para cada color de interés en cada espacio de color
    blanco_hsv_bajo = np.array([0, 0, 200])
    blanco_hsv_alto = np.array([100, 90, 250])
    blanco_lab_bajo = np.array([215, 110, 135])
    blanco_lab_alto = np.array([249, 150, 154])
    naranja_hsv_bajo = np.array([5, 150,200])
    naranja_hsv_alto = np.array([30, 255, 254])
    naranja_lab_bajo = np.array([155, 140, 145])
    naranja_lab_alto = np.array([200, 175, 195])
    amarillo_hsv_bajo = np.array([20, 80, 100])
    amarillo_hsv_alto = np.array([40, 255, 255])
    amarillo_lab_bajo = np.array([190, 105, 155])
    amarillo_lab_alto = np.array([215, 155, 195])
    verde_hsv_bajo = np.array([50, 50, 50])
    verde_hsv_alto = np.array([80, 255, 255])
    verde_lab_bajo = np.array([100, 135, 105])
    verde_lab_alto = np.array([140, 155, 135])
    violeta_hsv_bajo = np.array([100, 30, 50])
    violeta_hsv_alto = np.array([150, 255, 255])
    violeta_lab_bajo = np.array([80, 130, 87])
    violeta_lab_alto = np.array([210, 155, 190])
    rosa_hsv_bajo = np.array([130, 50, 50])
    rosa_hsv_alto = np.array([175, 255, 255])
    rosa_lab_bajo = np.array([80, 135, 130])
    rosa_lab_alto = np.array([215, 165, 175])
    azul_hsv_bajo = np.array([85, 50, 50])
    azul_hsv_alto = np.array([140, 255, 255])
    azul_lab_bajo = np.array([85, 100, 70])
    azul_lab_alto = np.array([170, 155, 190])
    rojo_hsv_bajo1 = np.array([0, 90, 20])
    rojo_hsv_alto1 = np.array([50, 255, 255])
    rojo_hsv_bajo2 = np.array([175, 100, 20])
    rojo_hsv_alto2 = np.array([180, 255, 250])
    rojo_lab_bajo = np.array([0, 135, 135])
    rojo_lab_alto = np.array([215, 155, 175])

    # Aplicar las máscaras de color a la imagen
    mask_blanco_hsv = cv2.inRange(hsv, blanco_hsv_bajo, blanco_hsv_alto)
    mask_blanco_lab = cv2.inRange(lab, blanco_lab_bajo, blanco_lab_alto)
    mask_naranja_hsv = cv2.inRange(hsv, naranja_hsv_bajo, naranja_hsv_alto)
    mask_naranja_lab = cv2.inRange(lab, naranja_lab_bajo, naranja_lab_alto)
    mask_amarillo_hsv = cv2.inRange(hsv, amarillo_hsv_bajo, amarillo_hsv_alto)
    mask_amarillo_lab = cv2.inRange(lab, amarillo_lab_bajo, amarillo_lab_alto)
    mask_verde_hsv = cv2.inRange(hsv, verde_hsv_bajo, verde_hsv_alto)
    mask_verde_lab = cv2.inRange(lab, verde_lab_bajo, verde_lab_alto)
    mask_violeta_hsv = cv2.inRange(hsv, violeta_hsv_bajo, violeta_hsv_alto)
    mask_violeta_lab = cv2.inRange(lab, violeta_lab_bajo, violeta_lab_alto)
```

```
        mask_rosa_hsv = cv2.inRange(hsv, rosa_hsv_bajo, rosa_hsv_alto)
        mask_rosa_lab = cv2.inRange(lab, rosa_lab_bajo, rosa_lab_alto)
        mask_azul_hsv = cv2.inRange(hsv, azul_hsv_bajo, azul_hsv_alto)
        mask_azul_lab = cv2.inRange(lab, azul_lab_bajo, azul_lab_alto)
        mask_rojo_hsv1 = cv2.inRange(hsv, rojo_hsv_bajo1, rojo_hsv_alto1)
        mask_rojo_hsv2 = cv2.inRange(hsv, rojo_hsv_bajo2, rojo_hsv_alto2)
        mask_rojo_hsv = cv2.bitwise_or(mask_rojo_hsv1, mask_rojo_hsv2)
        mask_rojo_lab = cv2.inRange(lab, rojo_lab_bajo, rojo_lab_alto)

        # Obtener el color predominante en la imagen(model)
        colors = {
            "blanco": np.sum(mask_blanco_hsv) + np.sum(mask_blanco_lab),
            "naranja": np.sum(mask_naranja_hsv) + np.sum(mask_naranja_lab),
            "amarillo": np.sum(mask_amarillo_hsv) + np.sum(mask_amarillo_lab),
            "verde": np.sum(mask_verde_hsv) + np.sum(mask_verde_lab),
            "violeta": np.sum(mask_violeta_hsv) + np.sum(mask_violeta_lab),
            "rosa": np.sum(mask_rosa_hsv) + np.sum(mask_rosa_lab),
            "azul": np.sum(mask_azul_hsv) + np.sum(mask_azul_lab),
            "rojo": np.sum(mask_rojo_hsv) + np.sum(mask_rojo_lab),
        }

        predominantColor = max(colors, key=colors.get)

        #devuelve el color del objeto
        return predominantColor
```

recortarImgparaTexturas.py

```python
#Esta funcion recorta una parte de la mascara para poder aplicar el detector de texturas

import cv2

def recortarImgparaTexturas(img,boxesxyxy,boxesxywh):


    img = cv2.resize(img, dsize=(640,448),interpolation=cv2.INTER_CUBIC)

    centro = boxesxywh
    xmin=boxesxyxy[0]
    xmax=boxesxyxy[2]
    ymin=boxesxyxy[1]
    ymax=boxesxyxy[3]
    xmin = int(xmin)
    ymin = int(ymin)
    xmax = int(xmax)
    ymax = int(ymax)

    #recortamos la imagen en funcion del bounding box
    imagen_recortada = img[ ymin:ymax,xmin:xmax]

    #resize de la imagen recortada
    imagen_recortada = cv2.resize(imagen_recortada, dsize=(640,448),
                                  interpolation=cv2.INTER_CUBIC)

    #devuelve la imagen recortada
    return imagen_recortada
```

obtenerTexturas.py

```
#obtener textura utilizando la matriz de co-ocurrencia de nivel de gris (GLCM) y sus
propiedades

from skimage.feature import graycomatrix, graycoprops
import cv2


def obtenerTexturas(img):
    # Convertir imagen a un array de nivel de gris
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Calcular matriz de co-ocurrencia de nivel de gris (GLCM)
    # usando una distancia de 1 pixel y un ángulo de 0 grados
    glcm = graycomatrix(img_gray, distances=[1], angles=[0], levels=256,
                        symmetric=True, normed=True)

    # Calcular propiedades de GLCM
    contrast = graycoprops(glcm, 'contrast')
    dissimilarity = graycoprops(glcm, 'dissimilarity')
    homogeneity = graycoprops(glcm, 'homogeneity')
    ASM = graycoprops(glcm, 'ASM')
    energy = graycoprops(glcm, 'energy')
    correlation = graycoprops(glcm, 'correlation')


    textura =  'Hay un error en el calculo'
    # Determinar si la textura es lisa o aspera basándose en las propiedades de GLCM

    if ((energy < 0.41 and homogeneity > 0.82 and energy>0.19 and homogeneity<0.95)
        or (energy > 0.45 and energy < 0.5 and dissimilarity > 0.83 and
            contrast > 4 and contrast<37)
        or (energy > 0.30 and energy < 0.44 and dissimilarity > 0.43 and
            contrast > 3.4 and contrast<30 and
            (homogeneity > 0.69 or homogeneity < 0.68))
        or (energy<0.26  and energy > 0.12 and dissimilarity > 0.87 and
            dissimilarity < 1.1 and contrast<8.8 and contrast>5.5)
        or (energy<0.2 and energy > 0.19 and dissimilarity > 2 and
            dissimilarity < 2.3 and contrast <35 and contrast > 26)
        or (energy<0.14 and energy > 0.12 and dissimilarity > 0.8 and
            dissimilarity < 0.9 and contrast <14 and contrast > 13 )
        or (energy<0.18 and energy > 0.12 and dissimilarity > 0.2 and
            dissimilarity < 0.6 )
        or (energy < 0.3 and homogeneity > 0.53 and energy>0.19 and homogeneity<0.57)
        or (energy > 0.07 and energy < 0.11 and  contrast >8.5 and contrast<30 and
            dissimilarity<2.4 and dissimilarity > 1.4)
        or (energy<0.14 and energy > 0.12 and dissimilarity > 0.5 and
            dissimilarity < 0.6 and contrast <3.5)):
        textura = 'liso'
    else:
        textura = 'aspero'



    return textura,energy,homogeneity,dissimilarity,contrast
```

obtenerOrientaciones.py

```python
#obtener las orientaciones de los objetos utilizando la mascara de los mismos

from skimage.measure import regionprops
import numpy as np
import cv2


def obtenerOrientaciones(resultados):
    etiquetas = cv2.connectedComponents(cv2.cvtColor(resultados, cv2.COLOR_BGR2GRAY))[1]
    sO = regionprops(etiquetas, intensity_image=cv2.cvtColor(resultados, cv2.COLOR_BGR2GRAY))

    # Encuentra la región más grande
    mayor_region = None
    mayor_tamano = 0
    for prop in sO:
        tamano = prop.area
        if tamano > mayor_tamano:
            mayor_tamano = tamano
            mayor_region = prop

    # Calcula la orientación solo para la región más grande
    orientacion = mayor_region.orientation

    # Convierte la orientación de radianes a grados
    orientacionGrados = np.rad2deg(orientacion)
    return orientacionGrados
```

calibra.py

```python
import numpy as np
import cv2

def calib(img, distCal_mm=None):

    if distCal_mm is None:
        distCal_mm = 130

    cv2.imshow("Image", img)
    print("Dibuja el rectángulo ROI sobre la imagen y presiona Enter")
    print("Presiona Esc para cancelar")
    coor = cv2.selectROI("Image", img, fromCenter=False)
    cv2.destroyAllWindows()

    Xmin = int(coor[0])
    Xmax = int(coor[0] + coor[2])
    Ymin = int(coor[1])
    Ymax = int(coor[1] + coor[3])

    # Imagen cortada:
    imgCut = img[Ymin:Ymax, Xmin:Xmax]

    # Cálculo del factor de conversión
    d = np.sqrt((coor[2]) ** 2 + (coor[3]) ** 2)
    convF = distCal_mm / d

    # Imagen indicando el área recortada en la imagen original
    cv2.rectangle(img, (Xmin, Ymin), (Xmax, Ymax), (0, 255, 0), thickness=2)
    cv2.imshow("Image with ROI", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return coor, convF, imgCut

img = cv2.imread("calibrado.jpg")#captura del area de trabajo
coor,convF,imgcut=calib(img)
print("coor")
print(coor)
print("convF")
print(convF)
```

transformacionSistemas.py

```python
import numpy as np


def matricesTransformacion():
    #Valores fijos mm
    # Coordenadas del sistema de referencia 1 de la base del robot
    p1 = np.array([0, 0, 0])
    # Coordenadas del sistema de referencia 2 de la pinza
    p2 = np.array([9.067, -1.763, -163.319])
    # Coordenadas del sistema de referencia 3 de la camara
    p3 = np.array([22.243, 125.091, -4.309]) #REVISAR Z DE LA CAMARA,

    #Calculo de las matrices de transformacion
    # Vector de traslación de s1 a s2
    t12 = p2 - p1
    # Matriz de transformación de s2 a s1
    t12 = np.vstack([np.hstack([np.eye(3), t12.reshape(3, 1)]), [0, 0, 0, 1]])
    # Matriz de transformación de s2 a s1
    t21 = np.linalg.inv(t12)
    # Vector de traslación de s3 a s2
    t23 = p3 - p2
    # Matriz de transformación de s3 a s2
    t23 = np.vstack([np.hstack([np.eye(3), t23.reshape(3, 1)]), [0, 0, 0, 1]])
    # Matriz de transformación de s2 a s3
    t32 = np.linalg.inv(t23)

    return t12,t21,t23,t32

def deCamaraAPinza(xCamara,yCamara,t23):
    pto3 = [xCamara,yCamara,0]
    pto2 = t23.dot(np.hstack([pto3, 1]))[:3]
    return pto2

def ptoFinalXY(xpinza,ypinza,pBase):

    pto = [xpinza+pBase[0],ypinza+pBase[1]]
    print("Coordenadas a mover el robot :")
    print(pto)
    return pto

def puntosXYObjetos(objCent, pBase, convF=None):

    if convF is None:
        convF = 0.31089981918126963 #Colocar el valor obtenido en la funcion calib


    #Convertimos la X e Y del objeto a mm
    x = convF*objCent[0]
    y = convF*objCent[1]

    #Crea las matrices de transformacion
    t12,t21,t23,t32=matricesTransformacion()
    #Calcula la ubicacion del punto requerido(x,y) en funcion a la pinza
    ptoPinzadeCamara = deCamaraAPinza(x,y,t23)
    #Calcula el punto [X,Y] al cual se debe mover la base del robot
    ptoFinal=ptoFinalXY(ptoPinzadeCamara[0],ptoPinzadeCamara[1],pBase)
```

```
        return ptoFinal
```

conexionArduino.py

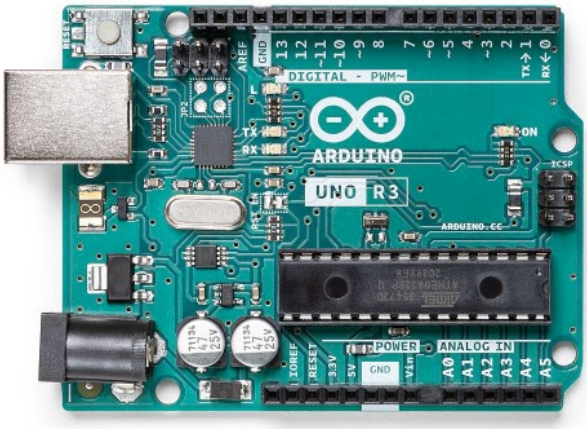
```
import serial
import transformacionSistemas
import numpy as np
import cv2



def conexionArduino(centrox,centroy,home,calibracion,ser2,mun,ser1):

    #MOVER A HOME
    if home == 1 and calibracion == 0:
        conexionArduinoCNC(0,0,0,1,0,ser2)

        #Posicionar la pinza correctamente
        conexionArduinoCintasYPinza(0,1,0,0,ser1)



    #CALIBRAR
    if home == 0 and calibracion == 1:
        conexionArduinoCNC(0,0,0,0,1,ser2)

        #Posicionar la pinza correctamente
        conexionArduinoCintasYPinza(0,1,0,0,ser1)



    #RECOGER PIEZA
    if home == 0 and calibracion == 0:

        pBase = np.array([0, 0])    #Punto [X,Y] del home
        ptoFinal= transformacionSistemas.puntosXYObjetos([centrox,centroy], pBase)
        print("pto final")
        print(ptoFinal[0])

        if ptoFinal[0] < 75 :
            cuadrante = 1
        else:
            cuadrante=2
        print("Pieza esta en el cuadrante: ", cuadrante)
        cv2.waitKey(0)

        if cuadrante == 1:

            #Punto intermedio para recogida
            conexionArduinoCNC(50,-1250,2200,0,0,ser2)


            #Girar muñeca
            conexionArduinoCintasYPinza(mun,1,0,0,ser1)


            #Punto de recogida
            conexionArduinoCNC(450,-1650,2600,0,0,ser2)
```

```
        #Cerrar pinza
        conexionArduinoCintasYPinza(mun,0,0,0,ser1)


        #Punto intermedio para recogida
        conexionArduinoCNC(50,-1250,2200,0,0,ser2)


        #Punto intermedio para entrega
        conexionArduinoCNC(2000,-250,1200,0,0,ser2)


        #Punto de entrega
        conexionArduinoCNC(2200,-550,1500,0,0,ser2)



        #Abrir pinza
        conexionArduinoCintasYPinza(mun,1,0,0,ser1)



        #Punto intermedio para entrega
        conexionArduinoCNC(2000,-250,1200,0,0,ser2)


        #Home y muevo cintaDesechos enderezando pinza
        conexionArduinoCNC(0,0,0,1,0,ser2)
        conexionArduinoCintasYPinza(0,1,0,1,ser1)



    if cuadrante == 2:
        #Punto intermedio para recogida
        conexionArduinoCNC(400,-1050,2000,0,0,ser2)


        #Girar muñeca
        conexionArduinoCintasYPinza(mun,1,0,0,ser1)



        #Punto de recogida
        conexionArduinoCNC(750,-1400,2350,0,0,ser2)



        #Cerrar pinza
        conexionArduinoCintasYPinza(mun,0,0,0,ser1)



        #Punto intermedio para recogida
        conexionArduinoCNC(400,-1050,2000,0,0,ser2)



        #Punto intermedio para entrega
```

```
            conexionArduinoCNC(2000,-250,1200,0,0,ser2)



            #Punto de entrega
            conexionArduinoCNC(2200,-550,1500,0,0,ser2)



            #Abrir pinza
            conexionArduinoCintasYPinza(mun,1,0,0,ser1)



            #Punto intermedio para entrega
            conexionArduinoCNC(2000,-250,1200,0,0,ser2)



            #Home y muevo cintaDesechos enderezando pinza
            conexionArduinoCNC(0,0,0,1,0,ser2)
            conexionArduinoCintasYPinza(0,1,0,1,ser1)



def conexionArduinoCNC(posicionX,posicionY,posicionZ,home,calibracion,ser2):

    #MOVER A HOME
    if home == 1 and calibracion == 0:
        input2 = "{},{},{},{},{}".format(0, 0, 0, 1, 0)
        ser2.write(input2.encode('ascii'))

    #CALIBRAR
    if home == 0 and calibracion == 1:
        input2 = "{},{},{},{},{}".format(0, 0, 0, 0, 1)
        ser2.write(input2.encode('ascii'))

    #MOVER
    if home == 0 and calibracion == 0 and (posicionX !=0 or posicionY != 0 or posicionZ != 0):
        input2 = "{},{},{},{},{}".format(posicionX, posicionY, posicionZ, 0, 0)
        ser2.write(input2.encode('ascii'))

    #leer datos com4
    num_datos_com4 = 2  # cantidad de lineas
    datos_recibidos_com4 = []

    for i in range(num_datos_com4):
        linea = ser2.readline().decode('cp1252').rstrip()
        datos_recibidos_com4.append(linea)

    #imprimimos los datos
    for i, dato in enumerate(datos_recibidos_com4):
        print(dato)



def conexionArduinoCintasYPinza(mun,pinza,cintaObjetos,cintaDesechos,ser1):
    if pinza == 1:
        pinza = 270 #Si se recibe un 1 en pinza indica que se quiere
```

```
                    #abrir la pinza, por lo que le pasamos 270 grados
    if pinza == 0:
        pinza = 0 #Si se recibe un 0 en pinza indica que se quiere
                  #cerrar la pinza, por lo que le pasamos 0 grados


    input1 = "{},{},{},{}".format(int(mun), int(pinza), int(cintaObjetos),cintaDesechos)
    ser1.write(input1.encode('ascii'))

   #leer datos com6
    num_datos_com6 = 2  # cantidad de lineas
    datos_recibidos_com6 = []

    for i in range(num_datos_com6):
        linea = ser1.readline().decode('cp1252').rstrip()
        datos_recibidos_com6.append(linea)

    #imprimimos los datos
    for i, dato in enumerate(datos_recibidos_com6):
        print(dato)
```

Product Reference Manual
SKU: A000066



Description

The Arduino UNO R3 is the perfect board to get familiar with electronics and coding. This versatile microcontroller is equipped with the well-known ATmega328P and the ATMega 16U2 Processor.
This board will give you a great first experience within the world of Arduino.

Target areas:

Maker, introduction, industries

Features

- **ATMega328P** Processor
  - **Memory**
    - AVR CPU at up to 16 MHz
    - 32KB Flash
    - 2KB SRAM
    - 1KB EEPROM
  - **Security**
    - Power On Reset (POR)
    - Brown Out Detection (BOD)
  - **Peripherals**
    - 2x 8-bit Timer/Counter with a dedicated period register and compare channels
    - 1x 16-bit Timer/Counter with a dedicated period register, input capture and compare channels
    - 1x USART with fractional baud rate generator and start-of-frame detection
    - 1x controller/peripheral Serial Peripheral Interface (SPI)
    - 1x Dual mode controller/peripheral I2C
    - 1x Analog Comparator (AC) with a scalable reference input
    - Watchdog Timer with separate on-chip oscillator
    - Six PWM channels
    - Interrupt and wake-up on pin change
  - **ATMega16U2 Processor**
    - 8-bit AVR® RISC-based microcontroller
  - **Memory**
    - 16 KB ISP Flash
    - 512B EEPROM
    - 512B SRAM
    - debugWIRE interface for on-chip debugging and programming
  - **Power**
    - 2.7-5.5 volts

CONTENTS

1 The Board

1.1 Application Examples

The UNO board is the flagship product of Arduino. Regardless if you are new to the world of electronics or will use the UNO as a tool for education purposes or industry-related tasks.

First entry to electronics: If this is your first project within coding and electronics, get started with our most used and documented board; Arduino UNO. It is equipped with the well-known ATmega328P processor, 14 digital input/output pins, 6 analog inputs, USB connections, ICSP header and reset button. This board includes everything you will need for a great first experience with Arduino.

Industry-standard development board: Using the Arduino UNO board in industries, there are a range of companies using the UNO board as the brain for their PLC's.

Education purposes: Although the UNO board has been with us for about ten years, it is still widely used for various education purposes and scientific projects. The board's high standard and top quality performance makes it a great resource to capture real time from sensors and to trigger complex laboratory equipment to mention a few examples.

1.2 Related Products

- Starter Kit
- Tinkerkit Braccio Robot
- Example

2 Ratings

2.1 Recommended Operating Conditions

Symbol	Description	Min	Max
	Conservative thermal limits for the whole board:	-40 °C (-40°F)	85 °C (185°F)

NOTE: In extreme temperatures, EEPROM, voltage regulator, and the crystal oscillator, might not work as expected due to the extreme temperature conditions

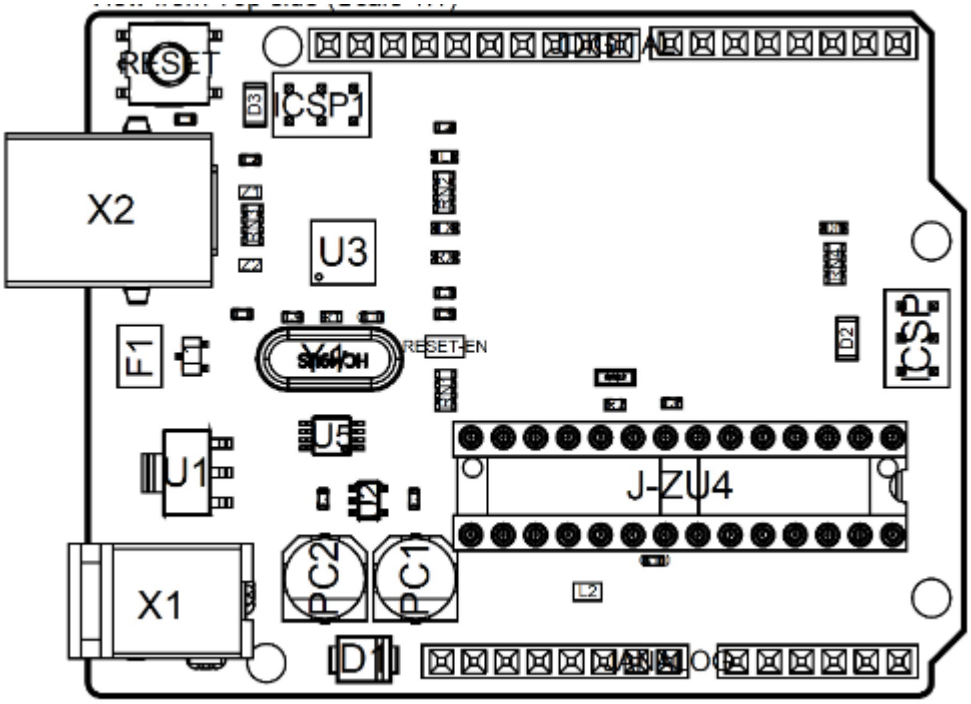
2.2 Power Consumption

Symbol	Description	Min	Typ	Max	Unit
VINMax	Maximum input voltage from VIN pad	6	-	20	V
VUSBMax	Maximum input voltage from USB connector		-	5.5	V
PMax	Maximum Power Consumption	-	-	xx	mA

3 Functional Overview

3.1 Board Topology

Top view



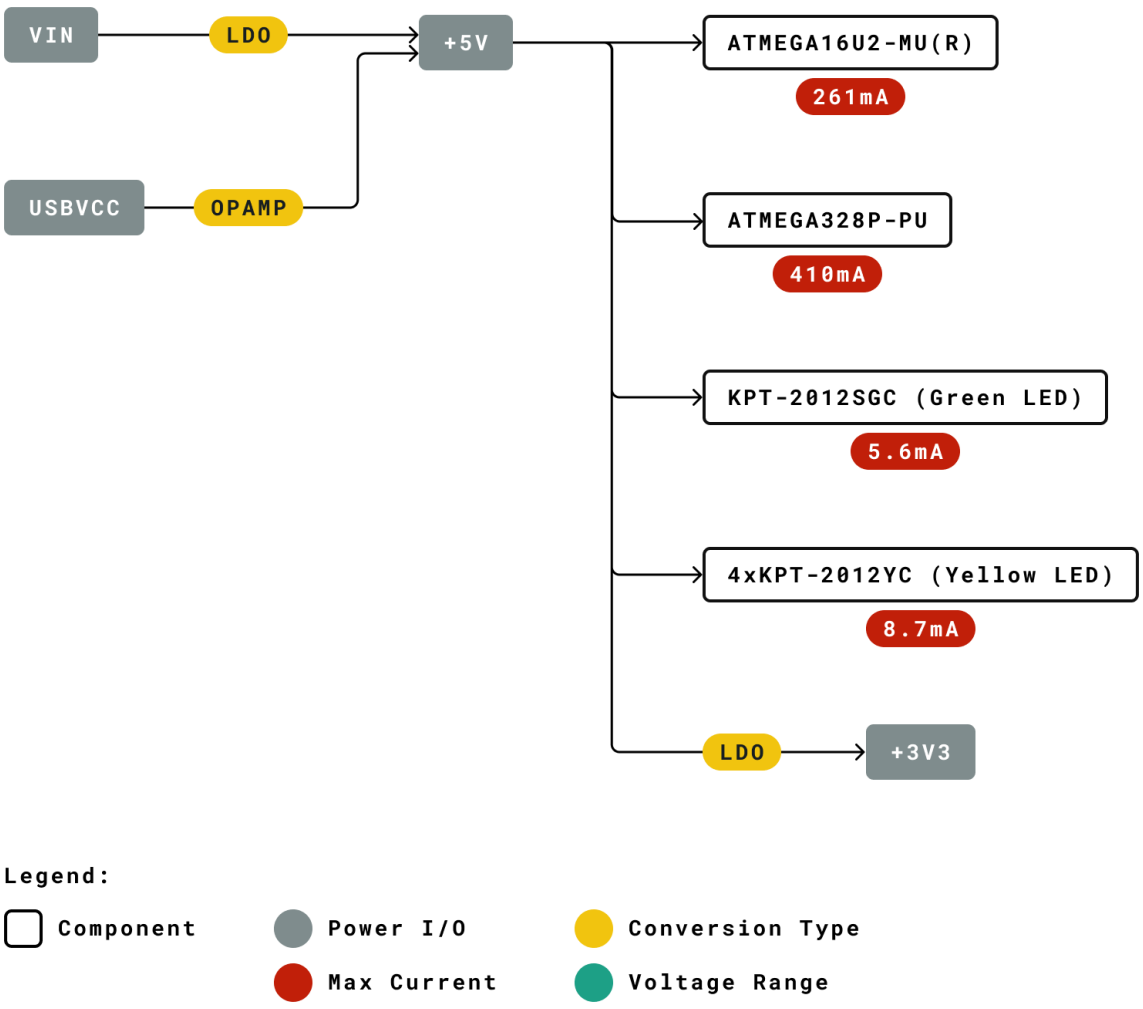
Board topology

Ref.	Description	Ref.	Description
X1	Power jack 2.1x5.5mm	U1	SPX1117M3-L-5 Regulator
X2	USB B Connector	U3	ATMEGA16U2 Module
PC1	EEE-1EA470WP 25V SMD Capacitor	U5	LMV358LIST-A.9 IC
PC2	EEE-1EA470WP 25V SMD Capacitor	F1	Chip Capacitor, High Density
D1	CGRA4007-G Rectifier	ICSP	Pin header connector (through hole 6)
J-ZU4	ATMEGA328P Module	ICSP1	Pin header connector (through hole 6)
Y1	ECS-160-20-4X-DU Oscillator		

3.2 Processor

The Main Processor is a ATmega328P running at up tp 20 MHz. Most of its pins are connected to the external headers, however some are reserved for internal communication with the USB Bridge coprocessor.

3.3 Power Tree



Power tree

4 Board Operation

4.1 Getting Started - IDE

If you want to program your Arduino UNO while offline you need to install the Arduino Desktop IDE [1] To connect the Arduino UNO to your computer, you'll need a Micro-B USB cable. This also provides power to the board, as indicated by the LED.

4.2 Getting Started - Arduino Web Editor

All Arduino boards, including this one, work out-of-the-box on the Arduino Web Editor [2], by just installing a simple plugin.

The Arduino Web Editor is hosted online, therefore it will always be up-to-date with the latest features and support for all boards. Follow **[3]** to start coding on the browser and upload your sketches onto your board.

4.3 Getting Started - Arduino IoT Cloud

All Arduino IoT enabled products are supported on Arduino IoT Cloud which allows you to Log, graph and analyze sensor data, trigger events, and automate your home or business.

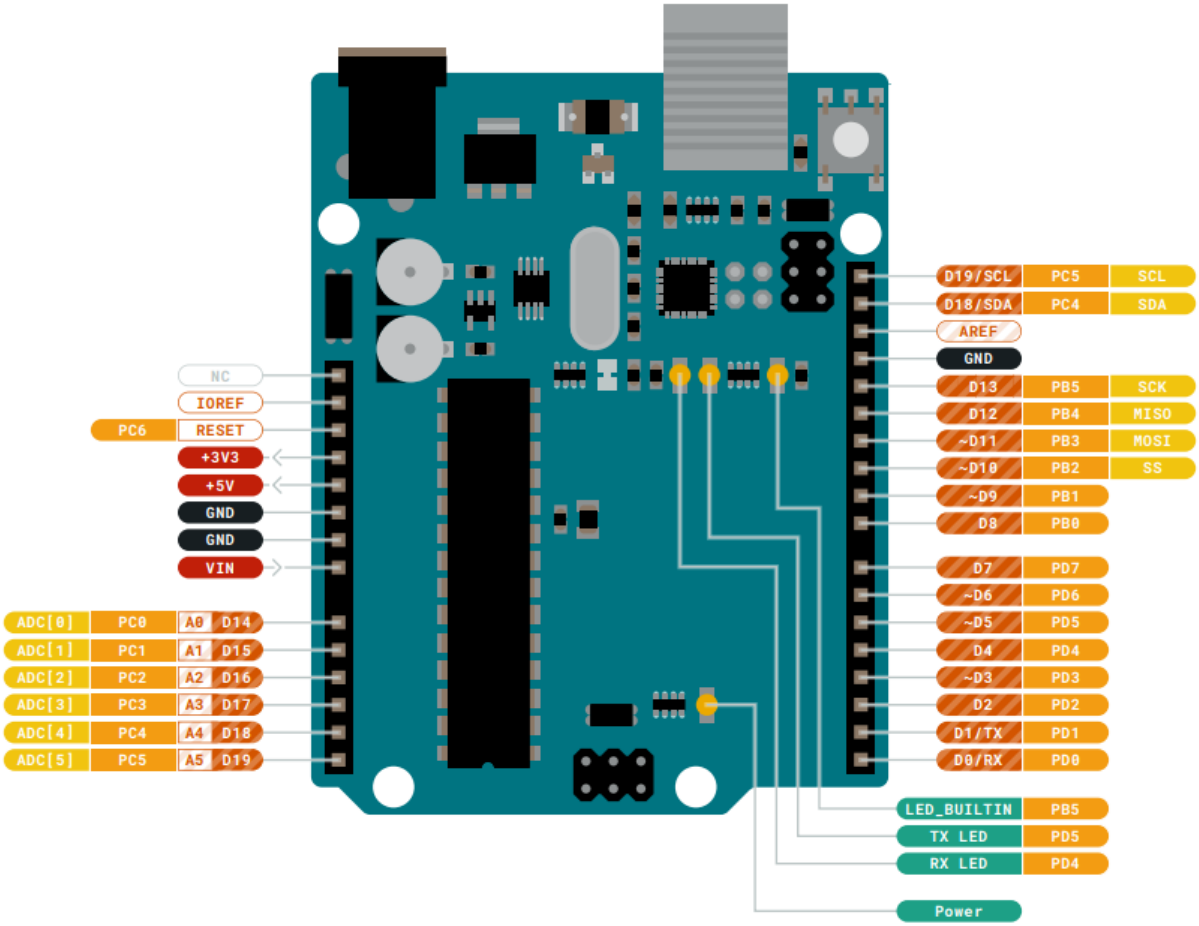
4.4 Sample Sketches

Sample sketches for the Arduino XXX can be found either in the "Examples" menu in the Arduino IDE or in the "Documentation" section of the Arduino Pro website [4]

4.5 Online Resources

Now that you have gone through the basics of what you can do with the board you can explore the endless possibilities it provides by checking exciting projects on ProjectHub **[5]**, the Arduino Library Reference **[6]** and the online store **[7]** where you will be able to complement your board with sensors, actuators and more

5 Connector Pinouts



Pinout

5.1 JANALOG

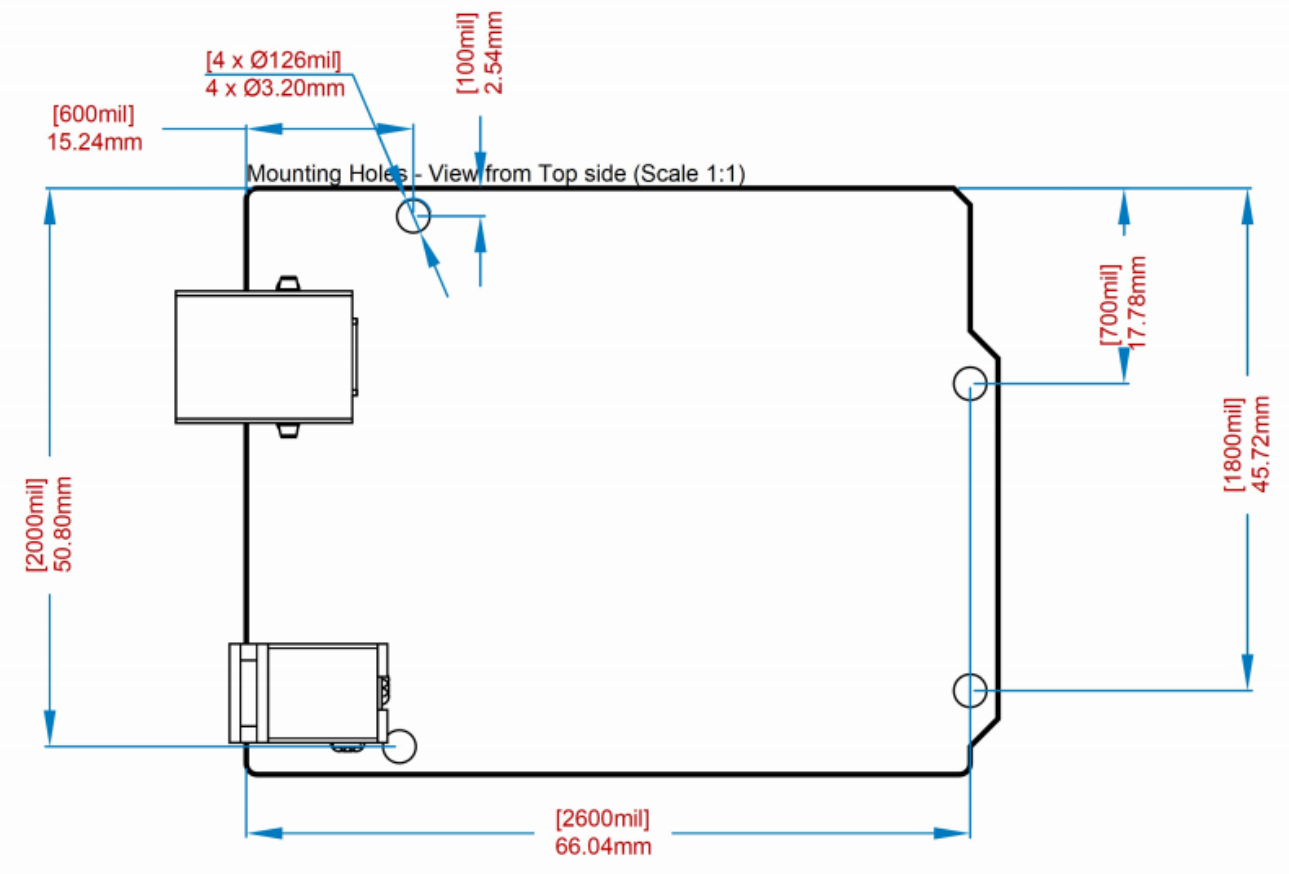
Pin	Function	Type	Description
1	NC	NC	Not connected
2	IOREF	IOREF	Reference for digital logic V - connected to 5V
3	Reset	Reset	Reset
4	+3V3	Power	+3V3 Power Rail
5	+5V	Power	+5V Power Rail
6	GND	Power	Ground
7	GND	Power	Ground
8	VIN	Power	Voltage Input
9	A0	Analog/GPIO	Analog input 0 /GPIO
10	A1	Analog/GPIO	Analog input 1 /GPIO
11	A2	Analog/GPIO	Analog input 2 /GPIO
12	A3	Analog/GPIO	Analog input 3 /GPIO
13	A4/SDA	Analog input/I2C	Analog input 4/I2C Data line
14	A5/SCL	Analog input/I2C	Analog input 5/I2C Clock line

5.2 JDIGITAL

Pin	Function	Type	Description
1	D0	Digital/GPIO	Digital pin 0/GPIO
2	D1	Digital/GPIO	Digital pin 1/GPIO
3	D2	Digital/GPIO	Digital pin 2/GPIO
4	D3	Digital/GPIO	Digital pin 3/GPIO
5	D4	Digital/GPIO	Digital pin 4/GPIO
6	D5	Digital/GPIO	Digital pin 5/GPIO
7	D6	Digital/GPIO	Digital pin 6/GPIO
8	D7	Digital/GPIO	Digital pin 7/GPIO
9	D8	Digital/GPIO	Digital pin 8/GPIO
10	D9	Digital/GPIO	Digital pin 9/GPIO
11	SS	Digital	SPI Chip Select
12	MOSI	Digital	SPI1 Main Out Secondary In
13	MISO	Digital	SPI Main In Secondary Out
14	SCK	Digital	SPI serial clock output
15	GND	Power	Ground
16	AREF	Digital	Analog reference voltage
17	A4/SD4	Digital	Analog input 4/I2C Data line (duplicated)
18	A5/SD5	Digital	Analog input 5/I2C Clock line (duplicated)

5.3 Mechanical Information

5.4 Board Outline & Mounting Holes



Board outline

6 Certifications

6.1 Declaration of Conformity CE DoC (EU)

We declare under our sole responsibility that the products above are in conformity with the essential requirements of the following EU Directives and therefore qualify for free movement within markets comprising the European Union (EU) and European Economic Area (EEA).

ROHS 2 Directive 2011/65/EU	
Conforms to:	EN50581:2012
Directive 2014/35/EU. (LVD)	
Conforms to:	EN 60950-1:2006/A11:2009/A1:2010/A12:2011/AC:2011
Directive 2004/40/EC & 2008/46/EC & 2013/35/EU, EMF	
Conforms to:	EN 62311:2008

6.2 Declaration of Conformity to EU RoHS & REACH 211 01/19/2021

Arduino boards are in compliance with RoHS 2 Directive 2011/65/EU of the European Parliament and RoHS 3 Directive 2015/863/EU of the Council of 4 June 2015 on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

Substance	Maximum limit (ppm)
Lead (Pb)	1000
Cadmium (Cd)	100
Mercury (Hg)	1000
Hexavalent Chromium (Cr6+)	1000
Poly Brominated Biphenyls (PBB)	1000
Poly Brominated Diphenyl ethers (PBDE)	1000
Bis(2-Ethylhexyl} phthalate (DEHP)	1000
Benzyl butyl phthalate (BBP)	1000
Dibutyl phthalate (DBP)	1000
Diisobutyl phthalate (DIBP)	1000

Exemptions: No exemptions are claimed.

Arduino Boards are fully compliant with the related requirements of European Union Regulation (EC) 1907 /2006 concerning the Registration, Evaluation, Authorization and Restriction of Chemicals (REACH). We declare none of the SVHCs (https://echa.europa.eu/web/guest/candidate-list-table), the Candidate List of Substances of Very High Concern for authorization currently released by ECHA, is present in all products (and also package) in quantities totaling in a concentration equal or above 0.1%. To the best of our knowledge, we also declare that our products do not contain any of the substances listed on the "Authorization List" (Annex XIV of the REACH regulations) and Substances of Very High Concern (SVHC) in any significant amounts as specified by the Annex XVII of Candidate list published by ECHA (European Chemical Agency) 1907 /2006/EC.

6.3 Conflict Minerals Declaration

As a global supplier of electronic and electrical components, Arduino is aware of our obligations with regards to laws and regulations regarding Conflict Minerals, specifically the Dodd-Frank Wall Street Reform and Consumer Protection Act, Section 1502. Arduino does not directly source or process conflict minerals such as Tin, Tantalum, Tungsten, or Gold. Conflict minerals are contained in our products in the form of solder, or as a component in metal alloys. As part of our reasonable due diligence Arduino has contacted component suppliers within our supply chain to verify their continued compliance with the regulations. Based on the information received thus far we declare that our products contain Conflict Minerals sourced from conflict-free areas.

7 FCC Caution

Any Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference

(2) this device must accept any interference received, including interference that may cause undesired operation.

FCC RF Radiation Exposure Statement:

1. This Transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.
2. This equipment complies with RF radiation exposure limits set forth for an uncontrolled environment.
3. This equipment should be installed and operated with minimum distance 20cm between the radiator & your body.

English: User manuals for license-exempt radio apparatus shall contain the following or equivalent notice in a conspicuous location in the user manual or alternatively on the device or both. This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions:

(1) this device may not cause interference

(2) this device must accept any interference, including interference that may cause undesired operation of the device.

French: Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes :

(1) l' appareil nedoit pas produire de brouillage

(2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

IC SAR Warning:

English This equipment should be installed and operated with minimum distance 20 cm between the radiator and your body.

French: Lors de l' installation et de l' exploitation de ce dispositif, la distance entre le radiateur et le corps est d 'au moins 20 cm.

Important: The operating temperature of the EUT can't exceed 85°C and shouldn't be lower than -40°C.

Hereby, Arduino S.r.l. declares that this product is in compliance with essential requirements and other relevant provisions of Directive 2014/53/EU. This product is allowed to be used in all EU member states.

8 Company Information

Company name	Arduino S.r.l
Company Address	Via Andrea Appiani 25 20900 MONZA Italy

9 Reference Documentation

Reference	Link
Arduino IDE (Desktop)	https://www.arduino.cc/en/Main/Software
Arduino IDE (Cloud)	https://create.arduino.cc/editor
Cloud IDE Getting Started	https://create.arduino.cc/projecthub/Arduino_Genuino/getting-started-with-arduino-web-editor-4b3e4a
Arduino Pro Website	https://www.arduino.cc/pro
Project Hub	https://create.arduino.cc/projecthub?by=part&part_id=11332&sort=trending
Library Reference	https://www.arduino.cc/reference/en/
Online Store	https://store.arduino.cc/

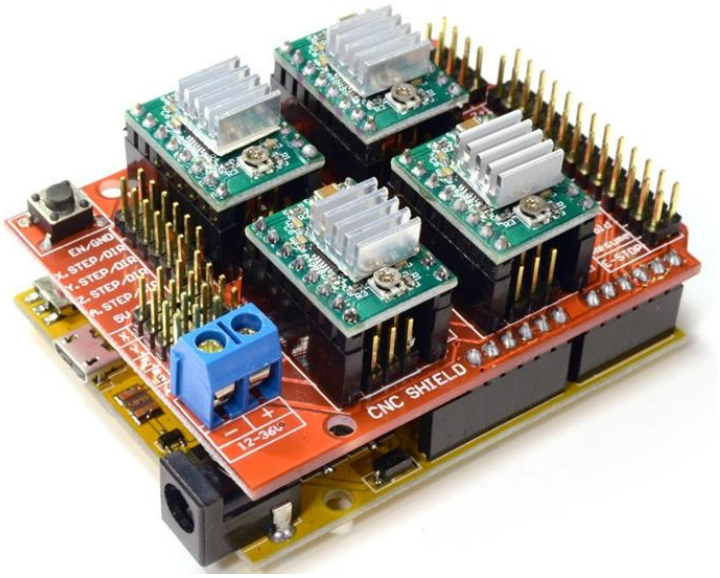
10 Revision History

Date	Revision	Changes
xx/06/2021	1	Datasheet release

User Manual

3-Axis CNC/Stepper Motor Shield for Arduino

The Arduino CNC Shield makes it easy to get your CNC projects up and running in a few hours. It uses opensource firmware on Arduino to control 4 stepper motors using 4 pieces of A4988 Stepper Motor driver breakout board, with this shield and ArduinoUno/Mega, you can build all kinds of robotics, linear motion project or projects including CNC routers, laser cutters and even pick&place machines.



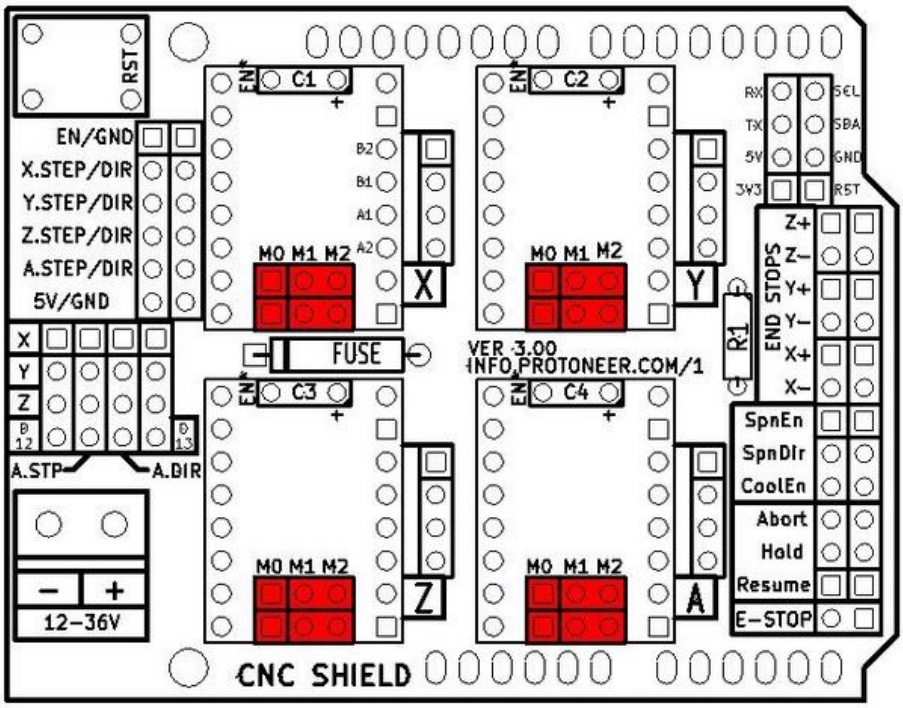
SKU: DRV1001

Brief Data:

- GRBL 0.9 compatible. (Open source firmware that runs on an Arduino UNO that turns G-code commands into stepper signals)
- 4-Axis support (X, Y, Z , A-Can duplicate X,Y,Z or do a full 4th axis with custom firmware using pins D12 and D13)
- 2 x End stops for each axis (6 in total)
- Coolant enable
- Uses removable A4988 compatible stepper drivers. (A4988, DRV8825 and others)(Not Included)
- Jumpers to set the Micro-Stepping for the stepper drivers. (Some drivers like the DRV8825 can do up to 1/32 micro-stepping)
- Compact design.
- Stepper Motors can be connected with 4-pin Molex connectors or soldered in place.
- Runs on 12-36VDC. (At the moment only the DRV8825 drivers can handle up to 36V so please consider the operation voltage when powering the board.)

1. Configuring Micro Stepping for Each Axis

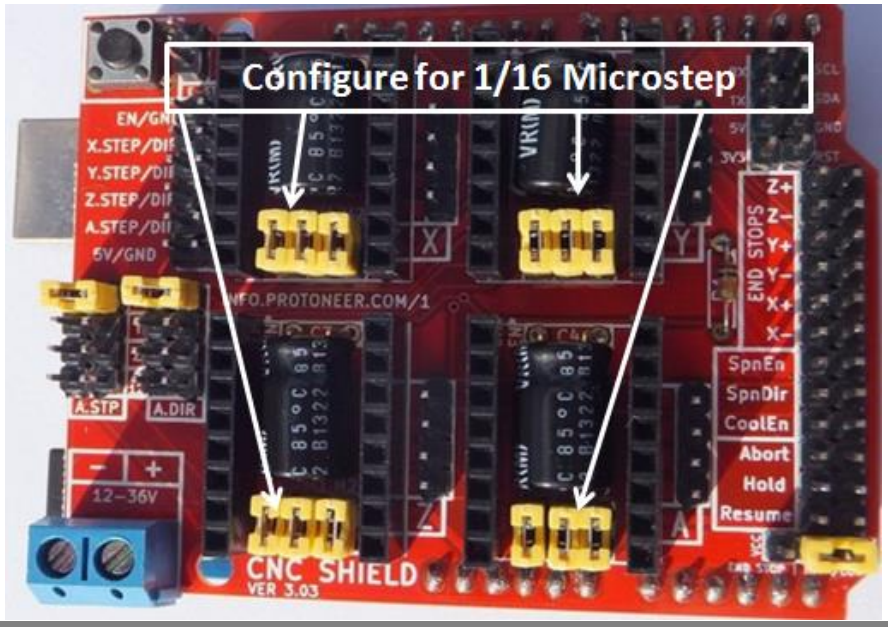
Each axis has 3 jumpers that can be set to configure the micro stepping for the A4988 plug-in driver board.



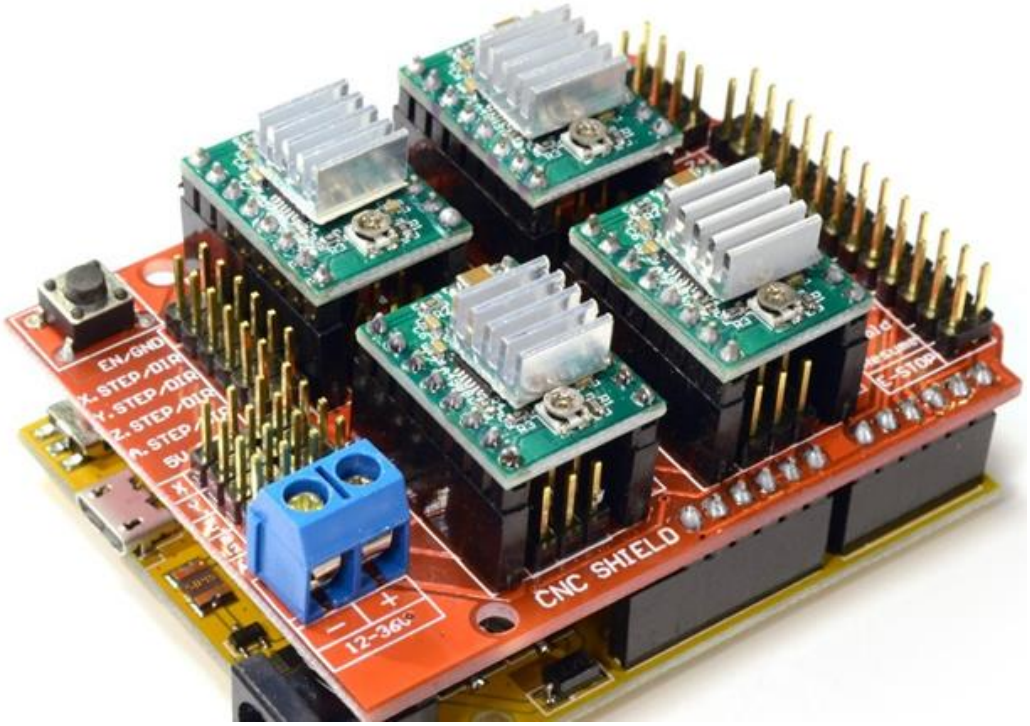
Micro-stepping jumper location, before inserting A4988.

In the tables below 'High' indicates that a jumper is insert and 'Low' indicates that no jumper is inserted.

MS0	MS1	MS2	Microstep Resolution
Low	Low	Low	Full Step
High	Low	Low	½ Step
Low	High	Low	¼ Step
High	High	Low	$^1/_8$ Step
High	High	High	$^1/_{16}$ Step



After setting the microstep jumper, you can plug-in A4988 driver boards as shown in the photo below. The photo also shown this CNC sit nicely on top of Arduino Uno board, without any external jumper wires.



!!! Beware of the orientation of the A4988 driver boards! You will destroy the A4988 driver board if plug-in with wrong orientation.

2. GRBL Control Software/Firmware for Arduino

Before you can use this CNC shield with Arduino, a control firmware need to be downloaded into Arduino board. We are going to use 'GRBL' to accomplish our job. GRBL is open-source software that runs on an Arduino Uno that takes G-Code commands via Serial and turns the commands into motor signals. Grbl is a no-compromise, high performance, low cost alternative to parallel-port-based motion control for CNC machine. It accepts standards-compliant g-code and has been tested with the output of several CAM tools with no problems. Arcs, circles and helical motion are fully supported, as well as, all other primary g-code commands. Macro functions, variables, and most canned cycles are not supported, but we think GUIs can do a much better job at translating them into straight g-code anyhow.

A copy of this open-source firmware can be downloaded from the below link:

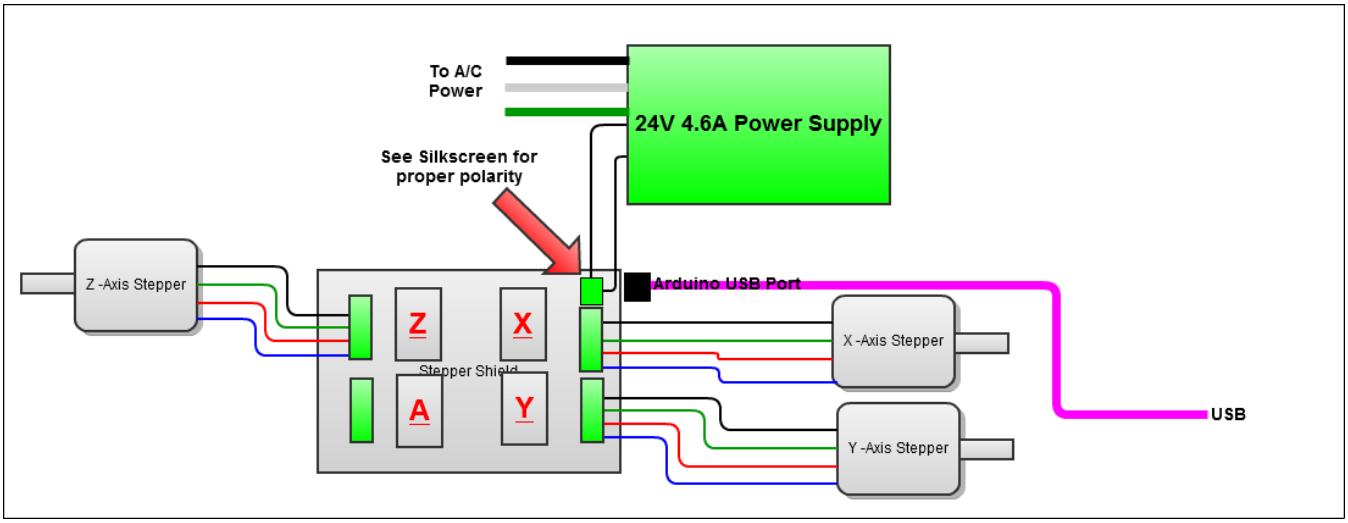
Following the below steps to prepare this CNC Shield board to function properly:

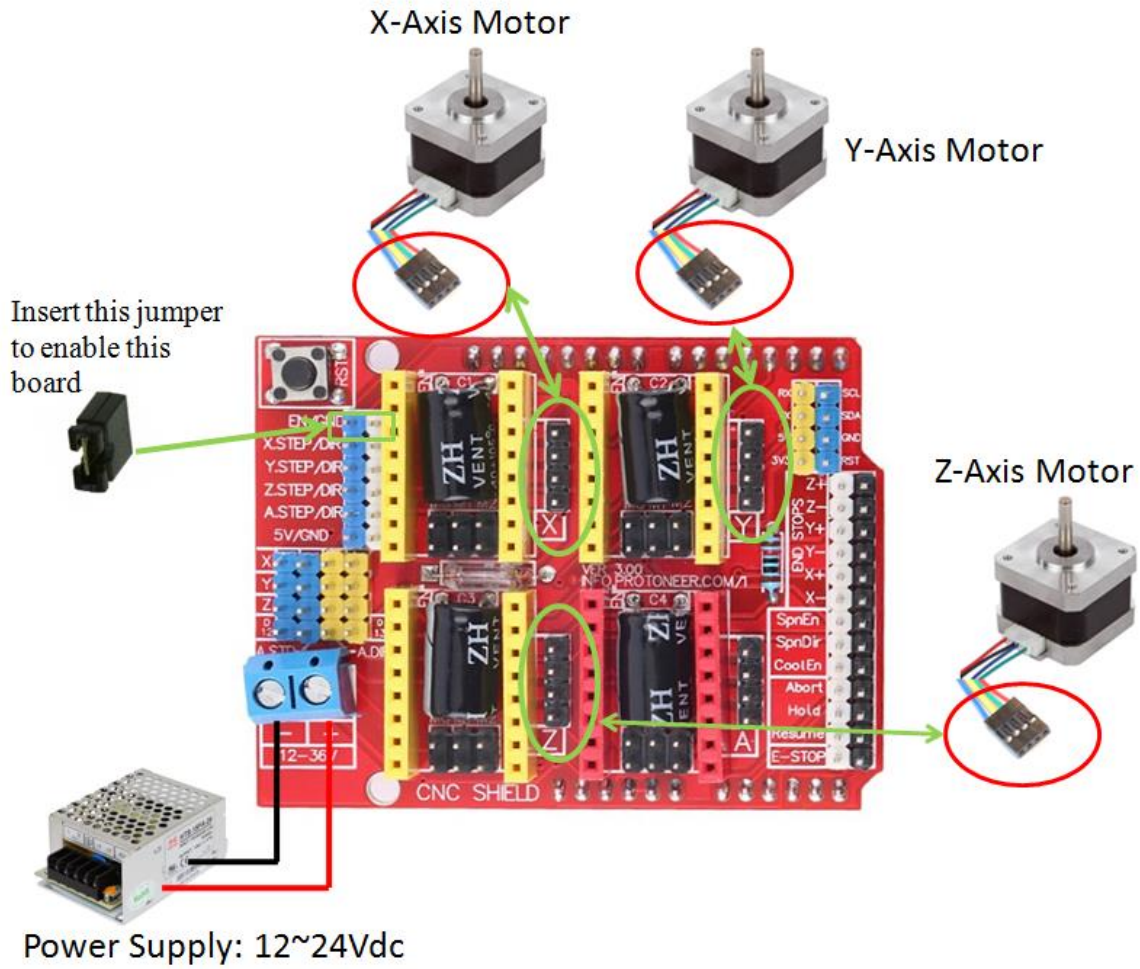
1. Download a copy of GRBL from: https://github.com/grbl/grbl

If you can receive response message " ***Grbl 0.9j ['$' for help]*** " from your Serial Monitor, congratulation! You have successful uploaded the 'GRBL' firmware into your Arduino board.

3. Hooking Up the Stepper Motor to CNC Shield

Connect steppers motor to CNC Shield board as the below block diagram. of the CNC Shield connected to 3-stepper motor:

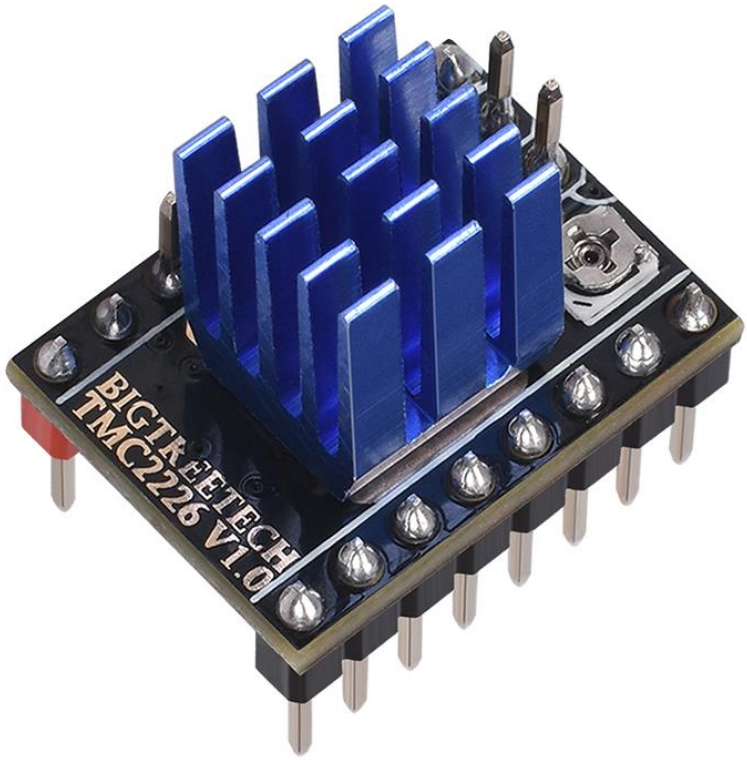




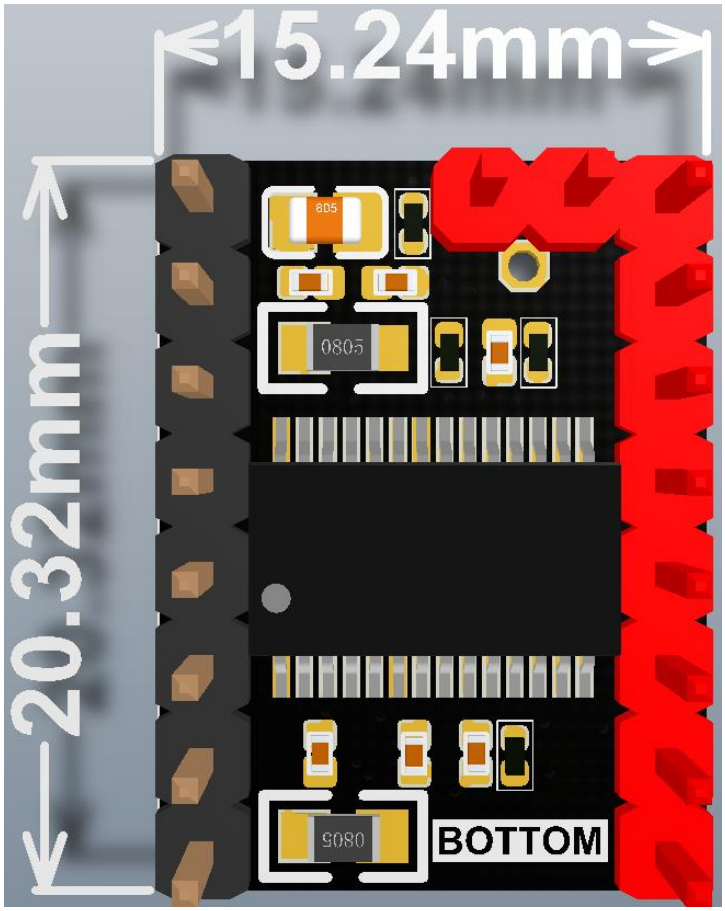
Your CNC Shield board is now ready to go for a test run, let's try to turn the motor as to our instruction !!

BIGTREETECH TMC2226-V1.0

Stepper motor driver module



I.Size parameters

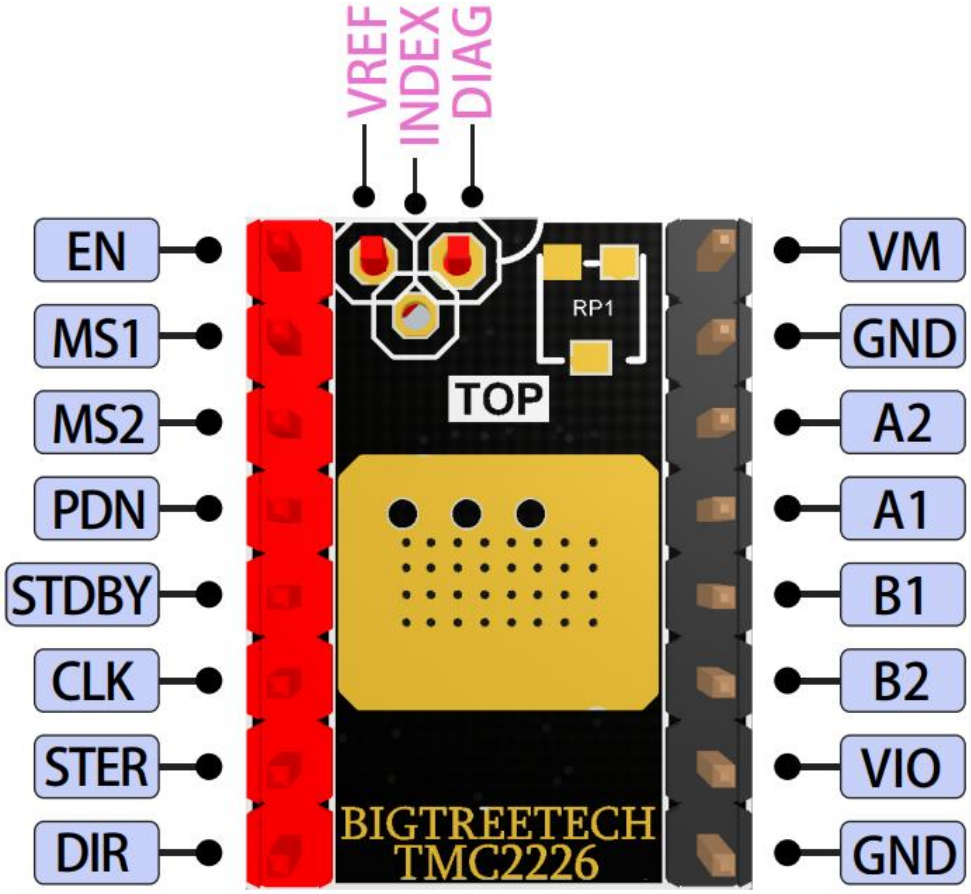


Parameter description:

2-phase stepper motors up to 2.8A coil current (peak), 2A RMS
STEP/DIR Interface with 8, 16, 32 or 64 microstep pin setting
Smooth Running 256 microsteps by **MicroPlyer™** interpolation
StealthChop2 silent motor operation
SpreadCycle highly dynamic motor control chopper
StallGuard4 load and stall detection for StealthChop
CoolStep current control for energy savings up to 75%
Low RDSon, Low Heat-Up LS 170mΩ & HS 170mΩ (typ. at 25°C)
Voltage Range 4.75… 29V DC
Low Power Standby to fit standby energy regulations
Internal Sense Resistor option (no sense resistors required)
Passive Braking, Freewheeling, and automatic power down
Single Wire UART & OTP for advanced configuration options
Integrated Pulse Generator for standalone motion
Full Protection & Diagnostics
Thermally optimized HTSSOP package for optical inspection

II. Working mode description

1. STEP/DIR mode：



Subdivision selection:MS1、MS2：

MS1/MS2: CONFIGURATION OF MICROSTEP RESOLUTION FOR STEP INPUT		
MS2	**MS1**	**Microstep Setting**
GND	GND	8 microsteps
GND	VCC_IO	32 microsteps (different to TMC2208!)
VCC_IO	GND	64 microsteps (different to TMC2208!)
VCC_IO	VCC_IO	16 microsteps

Standby mode:

STDBY	23	DI (pd)	STANDBY input. Pull up to disable driver internal supply regulator. This will bring the driver into a low power dissipation state. 100kOhm pulldown. (may be left unconnected) *Hint: Also shut down VREF voltage and ENN to 0V during standby.*

2.UART Working mode：

Benefits of UART mode:

Motor parameters can be set arbitrarily through firmware or screen；

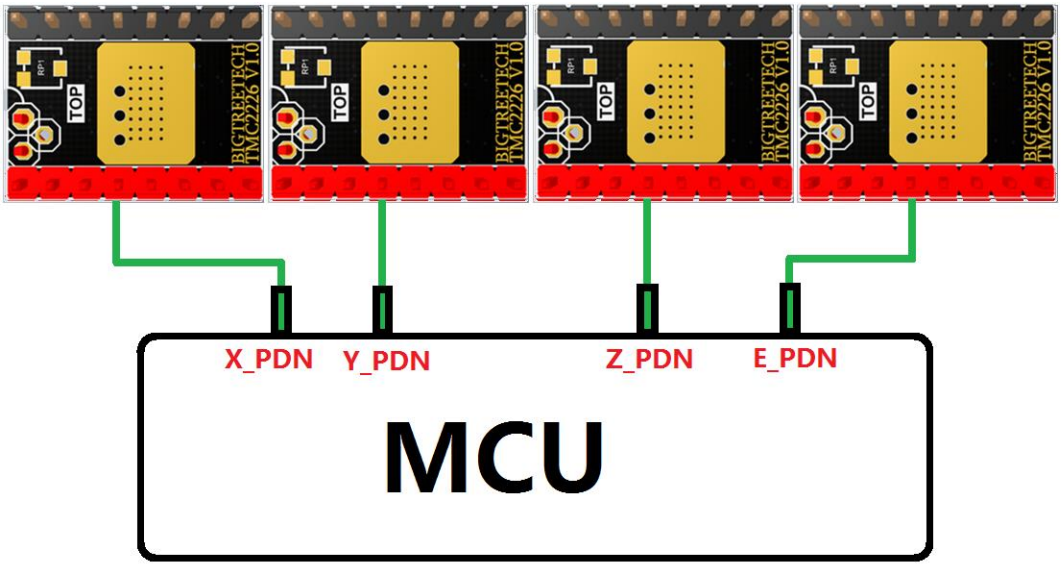
The actual interpolated microsteps can be combined to achieve maximum torque；

Firmware can dynamically switch the stepper motor between stealthChop2 and spreadCycle mode via UART；

Sensorless homing function can be used.

DIAG	15	DO	Diagnostic and StallGuard output. Hi level upon stall detection or driver error. Reset error condition by ENN=high.

The wiring diagram is as follows:



III.Firmware description

Firmware(Marlin-BUGFIX-2.0):Use TMC2209 firmware.

The TMC2226 UART mode can be used by directly replacing TMC2226 on the motherboard using the TMC2209UART mode.

IV.Notes

1. When selecting the working mode of hardware, use the

soldering iron carefully to prevent scalding of hands. After treatment, carefully observe whether there is residual tin slag in the module, and clean it to prevent it from causing the module short circuit burning;

2. Pay attention to the line sequence and IO port when wiring. Connecting the wrong line will directly cause the drive to not work, corresponding to the above connection carefully;

3. When inserting the driver into the motherboard, pay attention to the direction of the driver. Do not insert it backwards to prevent the driver from being burned;

4. Be sure to do the heat dissipation work before the driving work (heat sink and cooling fan)；

If you encounter other problems, please contact us. We will definitely answer your questions patiently. If you have any good suggestions on our products, please give us feedback. We will consider them seriously. Thank you for choosing BIGTREETECH products!

~\Desktop\ControlRobot.cpp

```cpp
//CONTROL DEL ROBOT
#include <AccelStepper.h>
#include <MultiStepper.h>

// Definimos las constantes para los pines de control de cada motor
#define MOTOR1_STEP 2
#define MOTOR1_DIR 5
#define MOTOR2_STEP 3
#define MOTOR2_DIR 6
#define MOTOR3_STEP 4
#define MOTOR3_DIR 7
#define switchPinX 9
#define switchPinY 10
#define switchPinZ 11
const int emergencyPin = A0;

// Creamos objetos para cada motor con sus respectivos pines
AccelStepper stepper1(AccelStepper::DRIVER, MOTOR1_STEP, MOTOR1_DIR);
AccelStepper stepper2(AccelStepper::DRIVER, MOTOR2_STEP, MOTOR2_DIR);
AccelStepper stepper3(AccelStepper::DRIVER, MOTOR3_STEP, MOTOR3_DIR);

MultiStepper steppers;


long tope = 5555;
int homeX = 0;
int homeY = 0;
int homeZ = 0;
int fcX = 0;
int fcY = 0;
int fcZ = 0;
int homeEjesX = 600; // posicion de cada eje en el home
int homeEjesY =-1700;
int homeEjesZ=700;

long homeRetroceso[3];
long movimientoHome[3];
long movimiento[3];
int values[5];
int velocidad=800;


void setup() {

  // Configuramos los parámetros de los motores
  stepper1.setMaxSpeed(velocidad);
  stepper2.setMaxSpeed(velocidad);
  stepper3.setMaxSpeed(velocidad);
  pinMode(emergencyPin, INPUT_PULLUP);
  Serial.begin(9600);
    delay(30);

  steppers.addStepper(stepper1);
  steppers.addStepper(stepper2);
  steppers.addStepper(stepper3);
  pinMode(switchPinX, INPUT_PULLUP);
```

```
    pinMode(switchPinY, INPUT_PULLUP);
    pinMode(switchPinZ, INPUT_PULLUP);
    stepper1.setCurrentPosition(0);
    stepper2.setCurrentPosition(0);
    stepper3.setCurrentPosition(0);


}




void loop() {
    // Loop para esperar los valores de los grados

  while (values[0] == 0 && values[1] == 0 && values[2] == 0 && values[3] == 0 &&
  values[4] == 0){

    if (Serial.available() > 0) { // Espera el input
      String input = Serial.readStringUntil('\n');
      int index = 0;
      char *ptr = strtok(const_cast<char *>(input.c_str()), "(),");
      while (ptr != NULL && index < 5) {
        values[index++] = atoi(ptr);
        ptr = strtok(NULL, "(),");
      }
    }
    // CALIBRACION
    if (values[4] == 1 && digitalRead(emergencyPin) == HIGH && values[3] == 0 &&
    values[0] == 0 && values[1] == 0 && values[2] == 0) {
      homeRetroceso[0] = -tope;
      homeRetroceso[1] = 0;
      homeRetroceso[2] = 0;
      steppers.moveTo(homeRetroceso);

      while(digitalRead(switchPinX) == LOW) {
        if (digitalRead(emergencyPin) == HIGH) {
          stepper1.runSpeed();
          fcX =1;
        }
        else{
          stepper1.stop();
          fcX=0;
          break;
        }
      }
      stepper1.stop();

      if(fcX==1){
        stepper1.setCurrentPosition(0);

        homeRetroceso[0] = tope;
        homeRetroceso[1] = tope ;
        homeRetroceso[2] = 0 ;
        steppers.moveTo(homeRetroceso);
        while (stepper1.currentPosition() != homeEjesX) {
          if (digitalRead(emergencyPin) == HIGH) {
            stepper1.runSpeed();
            homeX=1;
```

```
    }
    else{
      stepper1.stop();

      homeX=0;
      break;
    }
  }
  if(homeX==1){
    while(digitalRead(switchPinY) == LOW ) {
      if (digitalRead(emergencyPin) == HIGH) {
        stepper2.runSpeed();
        fcY=1;
      }
      else{
        stepper2.stop();

        fcY=0;
        break;
      }
    }
    stepper2.stop();
    if(fcY==1){
      stepper2.setCurrentPosition(0);
      homeRetroceso[0] = homeEjesX;
      homeRetroceso[1] = -tope ;
      homeRetroceso[2] = -tope ;
      steppers.moveTo(homeRetroceso);

      while (stepper2.currentPosition() != homeEjesY ) {
        if (digitalRead(emergencyPin) == HIGH) {
          stepper2.runSpeed();
          homeY=1;
        }
        else{
          stepper2.stop();
          homeY=0;
          break;
        }
      }

      if (homeY==1){
        while(digitalRead(switchPinZ) == LOW ) {
          if (digitalRead(emergencyPin) == HIGH) {
            stepper3.runSpeed();
            fcZ=1;
          }
          else{
            stepper3.stop();
            fcZ=0;
            break;
          }
        }
        stepper3.stop();
        if(fcZ==1){
          stepper3.setCurrentPosition(0);
          homeRetroceso[0] = homeEjesX;
          homeRetroceso[1] = homeEjesY;
          homeRetroceso[2] = tope ;
          steppers.moveTo(homeRetroceso);
```

```
            while (stepper3.currentPosition() != homeEjesZ ) {
              if (digitalRead(emergencyPin) == HIGH) {
                stepper3.runSpeed();
                homeZ=1;
              }
              else{
                stepper3.stop();
                homeZ=0;
                break;
              }
            }


          }
        }
      }
    }
  }
  Serial.print("X: ");
  Serial.print(stepper1.currentPosition());
  Serial.print("  Y: ");
  Serial.print(stepper2.currentPosition());
  Serial.print("  Z: ");
  Serial.println(stepper3.currentPosition());
}


// HOME
if (values[4] == 0 && digitalRead(emergencyPin) == HIGH && values[3] == 1 &&
values[0] == 0 && values[1] == 0 && values[2] == 0){
if (stepper1.currentPosition() != 0 && stepper2.currentPosition() != 0 &&
stepper3.currentPosition() != 0){  //Verificamos que se ha calibrado

    movimientoHome[0] = homeEjesX;
    movimientoHome[1] = homeEjesY;
    movimientoHome[2] = homeEjesZ;

    steppers.moveTo(movimientoHome);

    while (stepper1.currentPosition() != movimientoHome[0]||
    stepper2.currentPosition() != movimientoHome[1] ||
    stepper3.currentPosition() != movimientoHome[2]) {
      if (digitalRead(emergencyPin) == HIGH) {
        steppers.run();
      }
      else{
        stepper1.stop();
        stepper2.stop();
        stepper2.stop();
        Serial.println(" Se detuvieron los motores haciendo el homing por Emergencia ");
        break;
      }

    }
  Serial.print("X: ");
  Serial.print(stepper1.currentPosition());
  Serial.print("  Y: ");
  Serial.print(stepper2.currentPosition());
  Serial.print("  Z: ");
```

```
        Serial.println(stepper3.currentPosition());
        }
        else{
          Serial.println("Realice primero la calibracion antes de mover a home los motores");
        }

      }



      // MOVER
      if (values[4] == 0 && digitalRead(emergencyPin) == HIGH && values[3] == 0 &&
      (values[0] !=0 || values[1] !=0 || values[2] !=0)) {
        if (stepper1.currentPosition() != 0 && stepper2.currentPosition() != 0 &&
        stepper3.currentPosition() != 0){  //Verificamos que se ha calibrado

          movimiento[0] = values[0];
          movimiento[1] = values[1];
          movimiento[2] = values[2];

          steppers.moveTo(movimiento);

          while (stepper1.currentPosition() != movimiento[0]||
          stepper2.currentPosition() != movimiento[1] ||
          stepper3.currentPosition() != movimiento[2]) {
            if (digitalRead(emergencyPin) == HIGH) {
              steppers.run();
            }
            else{
              stepper1.stop();
              stepper2.stop();
              stepper2.stop();
              Serial.println(" Se detuvieron los motores por Emergencia cuando se estaba
moviendo");
              break;
            }

          }
        Serial.print("X: ");
        Serial.print(stepper1.currentPosition());
        Serial.print("  Y: ");
        Serial.print(stepper2.currentPosition());
        Serial.print("  Z: ");
        Serial.println(stepper3.currentPosition());
        }
        else{
          Serial.println("Realice primero la calibracion antes de mover los motores");
        }

      }


    }

      values[0] = 0;
      values[1] = 0;
      values[2] = 0;
      values[3] = 0;
      values[4] = 0;
```

```
  Serial.println("Finalizo el codigo del Robot");
  }
```

~\Desktop\ControlCintasyPinza.cpp

```
//CONTROL DE LAS CINTAS Y LA PINZA
//Icluimos la libreria para mover los sevomotores
#include <Servo.h>

//Indicamos los pines conectados
const int StepX = 2;
const int DirX = 5;
const int StepY = 3;
const int DirY = 6;

const int emergencyPin = A0;

#define Pin_Mu 9 //limit x+
#define Pin_Pinza 10 //limit y+

//Creamos dos objeto de la clase Servo
Servo mu;
Servo pinz;

//Setup con la comunicación serial y los pines como output
void setup() {

  pinMode(StepX,OUTPUT);
  pinMode(DirX,OUTPUT);
  pinMode(StepY,OUTPUT);
  pinMode(DirY,OUTPUT);

  pinMode(emergencyPin, INPUT_PULLUP);

  mu.attach(Pin_Mu);
  mu.write(0);
  pinz.attach(Pin_Pinza);
  pinz.write(0);


  Serial.begin(9600);
  delay(30);
}

//Valores fijos a utilizar
int MovX = 0;
int MovY = 0;
long pasosInfinitos = 500000;//
int pasosDesechos= 15000;
int velocidad=1000;
int EM=0;

float transformar_a_pasos_nema = 200 / 360.00;
int values[4];
int gradser1, gradser2, valor;
bool direccionX = LOW;
bool direccionY = LOW;



void loop() {
```

```
//Para saber que no se ha ingresado nada, reiniciar y esperar nuevamente el input
values[0]=5000;
values[1]=5000;
values[2]=5000;
values[3]=5000;
gradser1=5000;
gradser2=5000;


while (values[0] == 5000 && values[1]== 5000 && values[2]== 5000 && values[3]== 5000){

  if (Serial.available() > 0) {
    String input = Serial.readStringUntil('\n');
    int index = 0;
    char *ptr = strtok(const_cast<char*>(input.c_str()), "(),");
      while (ptr != NULL && index < 4) {
        values[index++] = atoi(ptr);
        ptr = strtok(NULL, "(),");
      }
  }

  if(digitalRead(emergencyPin) == LOW){
    Serial.println("Emergencia esta pulsado");
    Serial.println("Por favor quite la emergencia");
    break;
  }

  //Para el servo de la pinza
  if (values[1] == gradser2) {
    //si el valor input es = a lo que estaba anteriormente no se mueve
  }
  if (digitalRead(emergencyPin) == LOW ) {
    //si esta en EM se abre la pinza
    pinz.write(0);
    EM=1;
    break;
  }

  if (values[1] != gradser2 && digitalRead(emergencyPin) == HIGH) {
    //Si no esta en EM y el valor input es != a lo que estaba anteriormente
    gradser2 = values[1];
    valor=gradser2/1.5;
    pinz.write(valor);
      if(EM == 0){
      Serial.print("Pinza en ");
      Serial.print(gradser2);
      Serial.print(" grados");
      }
    }


 //Para el servo de la muñeca
  if (digitalRead(emergencyPin) == LOW || values[0] == gradser1) {
    //si esta en EM o el valor input es = a lo que estaba anteriormente no se mueve

  }
  if (values[0] != gradser1 && digitalRead(emergencyPin) == HIGH) {
    //Si no esta en EM y el valor input es != a lo que estaba anteriormente
    gradser1 = values[0];
```

```
    mu.write(gradser1);
    if (EM == 0){
    Serial.print(", Muñeca en ");
    Serial.print(gradser1);
    Serial.println(" grados ");
    }
  }



  //Control del Motor X(CintaObjetos)
  if(values[2] == 1 || values[2]==-1){
    int valorAnterior=values[2];
        if (values[2]==-1){
          direccionX = HIGH;
        }
        else{
          direccionX=LOW;
        }
        if (digitalRead(emergencyPin) == LOW) {
          break;
         }
        else{
        digitalWrite(DirX, direccionX);
          long pasos_a_moverX = pasosInfinitos * transformar_a_pasos_nema;
          Serial.println(" Se enciende la cinta de entrada de objetos ");
          for (float x = 0.00; x < pasos_a_moverX; x++)
          { // loop para mover los pasos

            if (Serial.available() > 0) {        //Reviso si se ingreso otro valor
              String input = Serial.readStringUntil('\n');
              int index = 0;
              char *ptr = strtok(const_cast<char*>(input.c_str()), "(),");
              while (ptr != NULL && index < 4) {
                values[index++] = atoi(ptr);
                ptr = strtok(NULL, "(),");
              }
              if (values[2] != valorAnterior ){
                Serial.println("Se ha detenido la cinta de entrada de objetos");
                valorAnterior = values[2];
              }
            }



              if (digitalRead(emergencyPin) == LOW || values[2]==0)
              { // si esta en EM o se ingresa un cero no se mueve
                  MovX = 0;
                  EM =1;
                  Serial.println(" El Motor X se detuvo ");
                  break;
              }
              else{

                      digitalWrite(StepX, HIGH);
                      delayMicroseconds(velocidad);
                      digitalWrite(StepX, LOW);
                      delayMicroseconds(velocidad);
                       MovX = 1;
              }
```

```
        }


    }
  }

  //Control del Motor Y(CintaDesechos)
  if(values[3] == 1 || values[3]==-1){
  int valorAnteriorY=values[3];
    if (values[3]==-1){
          direccionY = HIGH;
        }
        else{
          direccionY=LOW;
        }
    if (digitalRead(emergencyPin) == LOW) {
      break;
    }
    else{
      digitalWrite(DirY, direccionY);
          float pasos_a_moverY = pasosDesechos * transformar_a_pasos_nema;
          Serial.println(" Se enciende la cinta de desechos ");
          for (float x = 0.00; x < pasos_a_moverY; x++)
          { // loop para mover los pasos

            if (Serial.available() > 0) {       //Reviso si se ingreso otro valor
              String input = Serial.readStringUntil('\n');
              int index = 0;
              char *ptr = strtok(const_cast<char*>(input.c_str()), "(),");
              while (ptr != NULL && index < 4) {
                values[index++] = atoi(ptr);
                ptr = strtok(NULL, "(),");
              }
              if (values[3] != valorAnteriorY ){
                Serial.println("Se ha detenido la cinta de desechos");
                valorAnteriorY = values[3];
              }
            }



              if (digitalRead(emergencyPin) == LOW || values[3]==0)
              { // si esta en EM o se ingresa un cero no se mueve
                  MovY = 0;
                  EM=1;
                  Serial.println(" El Motor Y se detuvo ");
                  break;
              }
              else{

                      digitalWrite(StepY, HIGH);
                      delayMicroseconds(velocidad);
                      digitalWrite(StepY, LOW);
                      delayMicroseconds(velocidad);
                       MovY = 1;
              }
          }

    }
```

```
      }
    }
    if(EM==0){
      Serial.println("Finalizo el codigo Cintas y Pinzas");
    }
    EM=0;
  }
```