UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

OVERHAULING DE LA ESTACIÓN DE VISIÓN DEL MPS

Trabajo de titulación previo a la obtención del

Título de Ingeniero en Electrónica y Automatización

AUTOR: Kevin Fabricio Mosquera Pacas

TUTOR: William Paúl Oñate Amaguaña

Quito-Ecuador

2023

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, Kevin Fabricio Mosquera Pacas con documento de identificación N° 1725442923 manifiesto que:

Soy el autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 07 de agosto del año 2023

Atentamente,

Kevin Fabricio Mosquera Pacas
1725442923

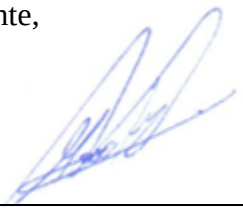
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Yo, Kevin Fabricio Mosquera Pacas con documento de identificación N° 1725442923 expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del proyecto técnico: "Overhauling de la estación de visión del MPS", el cual ha sido desarrollado para optar por el título de: Ingeniero en Electrónica y Automatización, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 07 de agosto del año 2023

Atentamente,



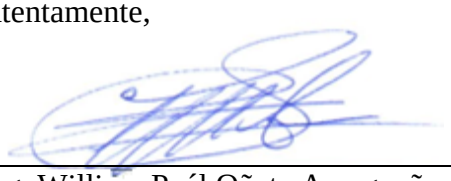
Kevin Fabricio Mosquera Pacas
1725442923

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, William Paúl Oñate Amaguaña con documento de identificación N° 1715580500 docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: OVERHAULING DE LA ESTACIÓN DE VISIÓN DEL MPS, realizado por Kevin Fabricio Mosquera Pacas con documento de identificación N° 1725442923, obteniendo como resultado final el trabajo de titulación bajo la opción de Proyecto Técnico, que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 07 de agosto del año 2023

Atentamente,



Ing. William Paúl Oñate Amaguaña, Msc.
1715580500

DEDICATORIA

La presente Tesis está dedicada principalmente a Dios, ya que gracias a él eh logrado concluir mi carrera, la dedico también a mis padres Marco y Guadalupe, porque ellos siempre estuvieron a mi lado brindándome su apoyo y sus consejos para hacer de mí una mejor persona, a mis hermanos Ruth, Johanna y Joe por sus palabras y su compañía, a mi amigo Edison que siempre ha estado en las buenas y en las malas siendo siempre ese apoyo en mi vida.

RESUMEN

Este trabajo de titulación consistió en la restauración e implementación de la estación de Visión para el laboratorio de Sistema de Producción Modular (MPS), la cual dicha estación se encontraba deshabilitado y sin ningún uso, por lo que fue necesario reintegrarlo al sistema para que junto con las otras estaciones sean utilizadas para el aprendizaje y desarrollo de nuevos temas de investigación.

Para la restauración de esta estación se utilizó como núcleo un sistema embebido específicamente una Raspberry Pi4, con una capacidad de memoria RAM de 8GB, lo que hace que su rendimiento sea el más adecuado para su uso en control de visión de artificial, y que al poseer un puerto Ethernet, pines digitales y un módulo de Wifi se facilite la comunicación con protocolos industriales.

Para que la estación sea capaz de reconocer las piezas maquinadas del laboratorio fue necesario aplicar un aprendizaje automático supervisado por el método de clasificación, en donde se utilizan como datos de entrada 900 imágenes positivas de las piezas y 500 imágenes negativas del entorno en donde iban a estar ubicadas estas piezas, y se requirió utilizar un programa en Windows capaz de generar un archivo xml llamado Cascade-Trainer-Gui en el cual se subió estas imágenes positivas y negativas y así se obtuvo el archivo para utilizar en Python.

En cuestión de machine learning el sistema se programó en Python basándose en la librería de OpenCV, que interpreta el archivo xml que contiene el aprendizaje de la pieza maquinada, que, junto con códigos de identificación, hace que su funcionamiento sea similar al reconocimiento facial de los smartphones, agregando la posibilidad de ubicar y reconocer las piezas dentro un entorno controlado.

ABSTRACT

This titling work consisted of the restoration and implementation of the Vision station for the Modular Production System (MPS) laboratory, which said station was disabled and without any use, so it was necessary to reintegrate it into the system so that together with the other stations are used for learning and developing new research topics.

For the restoration of this station, an embedded system specifically a Raspberry Pi4 was used as the core, with a RAM memory capacity of 8GB, which makes its performance the most suitable for use in artificial vision control, and at the same time Possessing an Ethernet port, digital pins, and a Wi-Fi module facilitate communication with industrial protocols.

In order for the station to be able to recognize the machined parts from the laboratory, it was necessary to apply automatic learning supervised by the classification method, where 900 positive images of the parts and 500 negative images of the environment where they were going to be used as input data. These pieces were located, and it was required to use a Windows program capable of generating an xml file called Cascade-Trainer-Gui in which these positive and negative images were uploaded and thus the file to be used in Python was obtained.

In a matter of machine learning, the system was programmed in Python based on the OpenCV library, which interprets the xml file that contains the learning of the machined part, which, together with identification codes, makes its operation similar to facial recognition of smartphones, adding the possibility of locating and recognizing the pieces within a controlled environment.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO 1

1 INTRODUCCIÓN

1.1 Hipótesis

El laboratorio de MPS tiene la finalidad de complementar la teoría abstracta en algo practico y funcional, contribuyendo en habilidades técnicas para el desarrollo profesional los estudiantes y también motivando al ingenio en posibles mejoras en producción científica. Este sistema modular está enfocado en el aprendizaje de control y automatización, basado en el control de procesos, el cual tiene disponibilidad varias estaciones para dicho aprendizaje.

De todas las estaciones que se encuentran en este laboratorio existe una que está inoperativa, esta estación es la "MPS-500 VISION", que es un sistema capaz de ver e interpretar, mediante dispositivos de captación de imágenes, este puede cerciorarse al instante si un producto cumple o no con ciertas condiciones predefinidas, al pasar un elemento por la cámara, el sistema puede obtener lecturas precisas de tamaño, inconsistencias en color, desperfecto de forma, piezas faltantes, consistencia de mezcla, desperfectos internos y superficiales y que, a diferencia del ojo humano, puede llevar a cabo esta tarea de forma ininterrumpida y tomar decisiones en base a la información que recibe, inmediatamente (Festo, Festo-Didactic, 2010). De esta manera se aplica el control de calidad por visión artificial de cualquier área industrializada, de manera eficiente, rápida y extremadamente rentable.

De lo dicho en párrafos anteriores, este trabajo de titulación pretende realizar un overhauling de la estación MPS-500 VISION, esto quiere decir que: se diseñarán y/o reemplazarán ciertas piezas en la parte de hardware, diseño de una comunicación entre estaciones del MPS con protocolos clásicos, y diseño de una arquitectura para la comunicación planta-nube.

1.2 Descripción del Problema

Las prácticas del laboratorio están dirigidas hacia cinco módulos de la estación (distribución, maquinado, almacenamiento, clasificación) de las seis que constituyen el Sistema Modular de Producción (MPS), siendo la sexta la de Visión, por otro lado, el grupo de investigación GIECA de la UPS-Quito, no ha considerado dentro de sus temas de investigación de transformación digital de Industria 4.0 esta estación de visión (Grupo de Investigación en Electrónica Control Y Automatización, 2017), debido a que, ciertas partes del hardware son difíciles de obtener, otras están en mal estado y se desconocen los drivers para el funcionamiento del software.

La estación de visión del MPS ha dejado de funcionar desde el año 2012, con lo cual las actividades planificadas por la coordinación de laboratorios de Electrónica y Automatización no se han se han completado desde entonces, adicionalmente, tanto los trabajos de titulación, los estudios de casos y artículos académicos relacionados con inteligencia para este sistema son escasos, tal es el caso que la última tesis realizada corresponde al 2012 acorde a los registros del repositorio de la UPS-Quito (Repositorio Institucional UPS, 2012).

A continuación, se deja una reseña de una revisión sistemática sobre estudios desarrollados en la estación antes mencionada, esto se lo hace para conocer el último estudio implementado en la sede Quito, y como en otras sedes de la misma institución que tienen habilitada dicha estación, han ido desarrollando temas de interés al colectivo científico. Además, de contar con otros estudios de otras instituciones del país.

- En la UPS de Quito, existe una sola Tesis con el nombre de “Diseño e implementación del manual técnico de prácticas para manejo del módulo “MPS 500 Vision” con un sistema de supervisión control y adquisición de datos-Scada, utilizando la inspección por cámara para gestionar el control de calidad vía protocolo de comunicaciones Ethernet, el cual es parte integral del “Sistema de Producción Modular (MPS)” (Auquilla & Cárdenas, 2012).

- En la Universidad de las Fuerzas Armadas, una tesis con el nombre de “Diseño e implementación de un módulo didáctico de visión artificial para la selección de objetos según colores y características morfológicas implementando un brazo robótico como complemento del sistema de producción modular (mps) existente en el laboratorio de

hidrónica y neumática de la universidad de las fuerzas armadas espe-extensión Latacunga." (Morales, 2019).

1.3 Objetivos

1.3.1 Objetivo General

Habilitar la estación de visión del laboratorio del MPS de la Universidad Politécnica Salesiana Quito-Ecuador campus Sur, para la integración al sistema de producción, mediante un overhauling.

1.3.2 Objetivos Específicos

• Investigar temas relacionados de la estación de visión del MPS para la recuperación de su funcionalidad mediante repositorios y base de datos de tesis y artículos académicos respectivamente.

• Desarrollar un algoritmo en la estación de visión del MPS para la identificación de piezas en mal estado mediante el uso de inteligencia artificial.

• Diseñar una arquitectura de comunicación entre la estación de visión y la nube para la supervisión y control de procesos mediante ecosistema IOT.

• Verificar el funcionamiento del overhauling de la estación de visión en un proceso de producción para su validación a través de pruebas experimentales.

CAPÍTULO 2

2 ESTACIÓN DE VISIÓN ARTIFICIAL

2.1 Introducción

En un proceso industrial, el producto que se pretende obtener debe cumplir ciertos parámetros, especificaciones o características que lo hagan útil para el fin propuesto, pero en un proceso automatizado no siempre se obtiene un resultado ideal con lo que es necesario tener un control de calidad dentro de este mismo proceso, por lo que, las industrias optan por contratar a personal encargado en verificar manualmente que el proceso siga la línea correcta de fabricación, pero en la actualidad con los avances tecnológicos es posible realizar esto a través de sistemas de inteligencia artificial, es por eso que, este trabajo de titulación pretende reintegrar un módulo de visión artificial mediante una cámara de alta resolución y como núcleo un sistema embebido en el laboratorio de MPS de la universidad Salesiana sede Quito, el cual es un laboratorio que simula un proceso de producción industrial con protocolos de comunicación que se aplican en la industria.

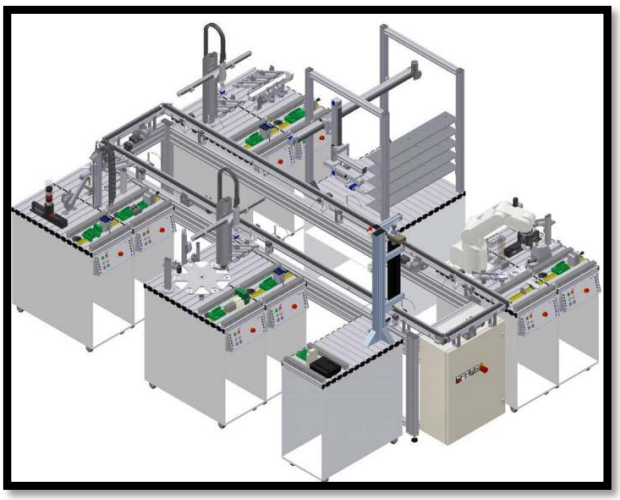
Para que esta estación pueda tener un control de calidad se utilizó visión artificial para la verificación del estado del producto, que en este caso son fichas de tres colores diferentes. La estación es capaz de comunicarse con la estación de maquinado junto con la banda transportadora y luego verificar el estado y el color de la ficha, con lo que decide si está en buen o mal estado, posteriormente ordena a la estación de clasificación que las clasifique por color o las deseche. Estas interacciones entre estaciones se logran mediante dos tipos de comunicaciones con protocolos industriales siendo la AS-I y Ethernet, además envían las cantidades de piezas verificadas a un bróker en la nube por medio del protocolo MQTT y las almacena en un historial el cual indica la hora y la ficha en la que la ficha fue verificada, y al concluir su uso antes de cerrarse envía un archivo en Excel de todo el historial a una nube privada, por último si se deseara cambiar los parámetros de caracteres que se desea analizar en la ficha la estación cuenta con una opción de calibración donde es posible obtener los nuevos valores de detalles que se encuentren en el último testeo.

2.2 El Sistema de Producción Modular (MPS 500)

El Sistema de Producción Modular (MPS) de la marca FESTO está destinado a la formación profesional y académica, siendo útiles en técnicas de enseñanza basadas en la ejemplificación del uso de electrónica en controles mecánicos y neumáticos para tecnologías de producción industriales complejos. Las plantas de maquinado y clasificación conforman parte de un sistema modular integrado que junto con la banda transportadora (denominada también como conveyor) con la posibilidad de ampliarse a futuro gracias a su arquitectura abierta tanto hardware como en software.

El MPS dispone de seis áreas, que se entre lazan por medio de una banda transportadora que esta a su vez dispone una comunicación AS-I como muestra la figura 1.

Figura 1. Estación de MPS



Fuente: (Festo, Festo didactic, 2022)

2.3 La calidad

La calidad hace referencia a qué tan bueno es algo en comparación a sus similares, es decir su grado de excelencia. Cuando es usado para describir personas, se refiere a una característica distintiva o atributo que poseen permitiendo que el término pueda ser empleado para objetos. (Prado, 2022)

En los negocios, especialmente en la fabricación, es una medida de excelencia. En este contexto, también puede referirse a un estado de estar libre de defectos.

La norma ISO 8402-1986 define la calidad como:

“La totalidad de funciones y características de un producto o servicio que tiene su capacidad para satisfacer necesidades declaradas o implícitas”. (ISO, 1986)

La Organización Internacional para la Estandarización (ISO) es un organismo internacional de establecimiento de normas formado por representantes de varias organizaciones nacionales de normalización.

En negocios, manufactura e ingeniería, el término tiene una interpretación pragmática como la superioridad o no inferioridad de algo. También se refiere a un producto como 'adecuado para su propósito', mientras que al mismo tiempo satisface las expectativas del consumidor.

La calidad es principalmente un atributo subjetivo y perceptivo esto hace que diferentes personas pueden no tener la misma comprensión del significado del término.

2.3.1 Gestión de la calidad

La gestión de calidad es el acto de supervisar todas las actividades y tareas necesarias para mantener un nivel deseado de excelencia. (Invatati afaceri, 2022) La gestión de calidad incluye la determinación de una política de calidad, la creación e implementación de la planificación y el aseguramiento de la calidad, y el control y la mejora de la calidad.

La gestión de la calidad se puede dividir en 4 fases principales:

- Inspección
- Control de Calidad
- Aseguramiento de la Calidad
- Gestión de la Calidad total

En este caso, se centrará específicamente en la inspección y el control de calidad, ya que estas dos fases están encaminadas a medir problemas de carácter operativo, y esto es lo que se requiere para la consecución de este proyecto.

2.4 Inspección Visual

Todo proceso industrial es evaluado por la calidad de su producto final. Esto hace que la etapa de control de calidad sea una fase crucial del proceso. Los mecanismos utilizados para establecer la calidad de un producto varían según los parámetros que le son relevantes.

La inspección de un objeto manufacturado pude involucrar las siguientes tareas:

- Verificar y comprobar características esperadas.
- Información detallada de esas características, como la altura, diámetro o longitud.
- Cerciorarse de la cualquier imperfección que este producto pueda tener.

Si bien la inspección visual humana es muy flexible y adaptable a múltiples situaciones, también depende de la experiencia y habilidad de las personas para examinar, así como del cansancio y la monotonía del trabajo de inspección, por lo que esta no es completamente segura a la hora de expresar resultados. (Enriquez Aguilera; Francisco Javier, 2018).

2.5 Inspección Visual Automatizada

Las técnicas de visión automatizada industrial son una mezcla de hardware y software, que facilitan capturar una imagen y emplear una serie de técnicas, que ayudan convertir y extraer características importantes, para que el sistema pueda tomar una decisión binaria. Las instrucciones de inspección y trabajo se almacenan digitalmente. Los tiempos de inspección se reducen considerablemente.

En la actualidad a este tipo de sistemas se los llama visión automatizada o artificial, por lo que se han convertido un instrumento de importante ayuda para las diferentes industrias, ya que es necesario como control de calidad que utiliza técnicas de procesamiento digital de imágenes, como es el reconocimiento de patrones o características, que determina de una manera automática si una pieza o producto está en buen o mal estado según las configuraciones preconfiguradas (Auquilla & Cárdenas, 2012).

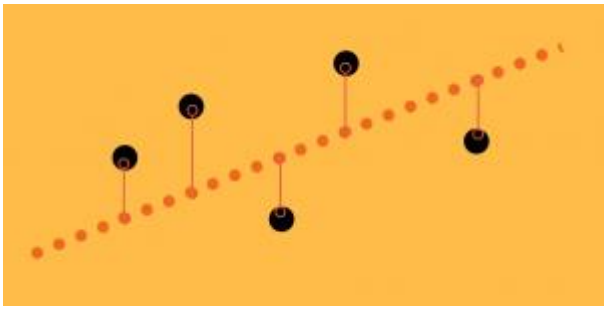
2.6 Algoritmos usados en Machine Learning (ML)

Existen distintos tipos de algoritmos para aplicar machine learning, entre los cuales destacan los detallados en los siguientes apartados.

2.6.1 Algoritmos de Regresión

Se clasifican en regresión lineal y regresión logística los cuales pretenden modelar la relación entre diferentes variables utilizando una medida de error que intentará minimizarse en un proceso iterativo para hacer predicciones donde un claro ejemplo se observa en la figura 2.

Figura 2. Algoritmos de Regresión, Logístico o Lineal

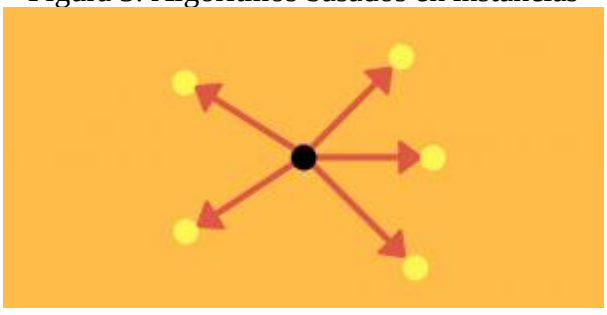


Fuente: (Aprende Machine Learning, 2017)

2.6.2 Algoritmos basados en Instancia

Este tipo de algoritmo crea un modelo a partir de una base de datos y se agregan nuevos datos comparando su similitud con las muestras existentes para encontrar la "mejor coincidencia" y hacer la predicción, la cual se representa en la figura 3.

Figura 3. Algoritmos basados en instancias



Fuente: (Aprende Machine Learning, 2017)

Se clasifican en Self- Organizing Map y en k-Nearest Neighbor (kNN) en donde el concepto de este último se utilizó para el presente proyecto.

2.6.3 K-Nearest neighbors (kNN) o vecinos más próximos

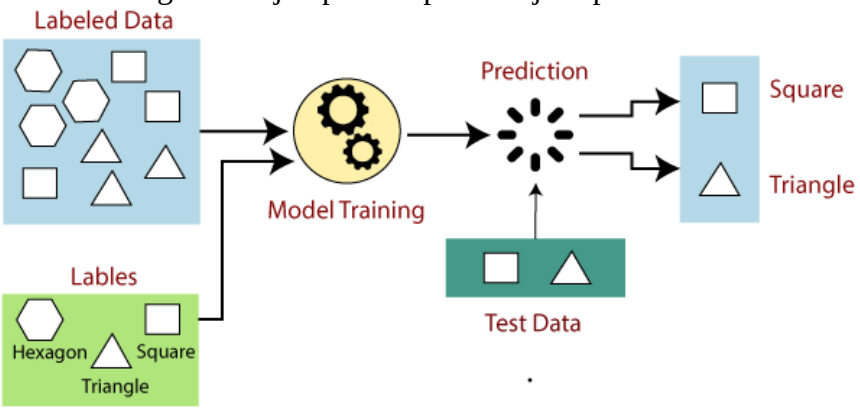
Es un algoritmo de aprendizaje supervisado en el cual se necesita un conjunto de datos para almacenarlos, para posteriormente crear un algoritmo de clasificación basado en instancias que tienen similares características (KeepCoding, 2022).

2.7 Aprendizaje supervisado

El aprendizaje supervisado es el tipo de aprendizaje automático en el que se utilizan algoritmos para realizar un entrenamiento utilizando datos previamente etiquetado y clasificado y en función de esos datos, el algoritmo predice el resultado. Los datos etiquetados significan que algunos datos de entrada ya están etiquetados con la salida correcta (Rodríguez, 2022). Es de esta manera que el algoritmo aprende a clasificar las muestras de entrada comparando el resultado del modelo y la etiqueta real de la muestra, realizando las respectivas compensaciones al modelo de acuerdo a cada error en la apreciación del resultado (javatpoint, 2022).

En la figura 4 se muestra cómo funciona el aprendizaje supervisado.

Figura 4. Ejemplo de aprendizaje supervisado



Fuente: (javatpoint, 2022)

2.7.1 Clasificadores Haar

La detección de objetos mediante clasificadores en cascada basados en funciones de Haar es un método eficaz de detección de objetos. Este es un enfoque basado en el aprendizaje automático en el que la función de cascada se entrena con muchas imágenes positivas y

negativas para posteriormente usarlas para detectar piezas u objetos en otras imágenes (Jones, 2004).

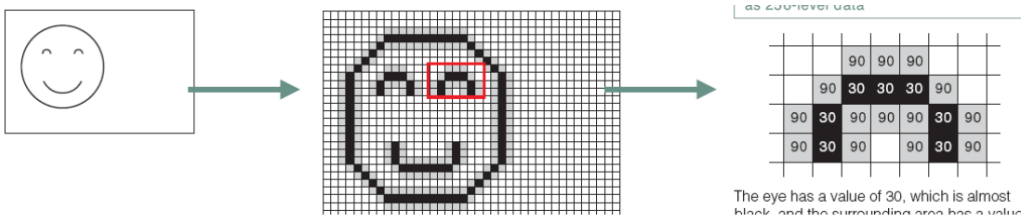
2.8 Iluminación para los sistemas de visión

Para usar los sistemas de visión artificial es necesario el uso de una cámara digital que tenga la capacidad de enfocar ya sea manual o automáticamente el área donde se requiere el uso del control de calidad, para que la imagen sea totalmente nítida y sea posible extraer la información de las características del objeto señalado.

En el área que se desea obtener la imagen del objeto propuesto debe contar con una correcta iluminación, la cual no generen sombras innecesarias alrededor de la pieza o producto a inspeccionar, por lo cual las industrias optan por realizar una cabina o un cilindro, donde la iluminación cubra casi toda la superficie, esto facilita que a la toma de imagen sea mucho más fácil identificar imperfecciones.

Una imagen digital se entiende como una larga cadena de unos y ceros que representan los puntos de color de la foto, estos se conocen como píxeles, los cuales componen la imagen (ver figura 5).

Figura 5. Pixelado de imagen digital



Fuente: (Auquilla & Cárdenas, 2012)

2.9 Filtrado del ruido en una imagen

Al momento de capturar las imágenes y procesarlas, estas presentan ruido inherente propio de los dispositivos o las perturbaciones ambientales como por ejemplo efectos de iluminación en el entorno de trabajo.

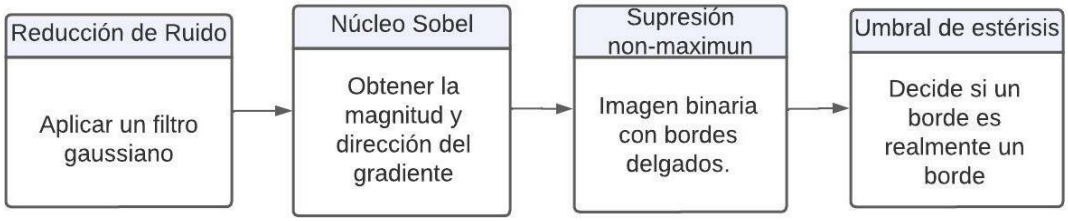
En el procesamiento de imágenes digitales existen diferentes métodos para eliminar el ruido (promedio, mediana, gaussiano o bilateral), todos estos usan la misma operación matemática de convolución, la cual consiste en recorrer píxel a píxel una imagen con una máscara o kernel de N x N. Este tamaño determina el número de píxeles con los que se va a trabajar (Hernández, 2018).

2.10 Detector de bordes Canny

El proceso para detectar bordes con el algoritmo de Canny se divide en 3 pasos, previo a ver filtrado la imagen por el método gaussiano, tal como se observa en la figura 6.

- Detección de bordes con Sobel
- Supresión de píxeles fuera del borde
- Aplicar umbral por histéresis

Figura 6. Diagrama de bloques de proceso de Canny



Fuente: Kevin Mosquera

Muchas técnicas requieren establecer parámetros o condiciones iniciales según cada situación. En muchas ocasiones esos parámetros son exclusivos para una determinada iluminación o perspectiva (Hernández, 2018), es decir algo que funciona correctamente en una condición de iluminación puede no funcionar en otra.

2.10.1 Detección de bordes con Sobel

El detector de bordes Sobel se basa en el cálculo de la primera derivada. Esta operación matemática mide las evoluciones y los cambios de una variable, básicamente se centra en detectar cambios de intensidad (Hernández, 2018).

2.10.2 Filtrado de bordes mediante la supresión non-maximun

El objetivo en esta fase es extraer aquellos bordes que cumplan cierta condición, para este caso de estudio se decidió a través del algoritmo de bordes Canny y la técnica de supresión non-maximun, aquellos patrones que tengan como grosor 1 (Hernández, 2018).

2.10.3 Aplicar umbral por histéresis

La umbralización de imágenes permite segmentar una imagen en sus diferentes objetos el cual consiste en determinar un umbral por el cual se decide si un píxel forma parte del fondo o forma parte de un objeto. El umbral por histéresis se basa en establecer dos umbrales, uno máximo y otro mínimo (Hernández, 2018).

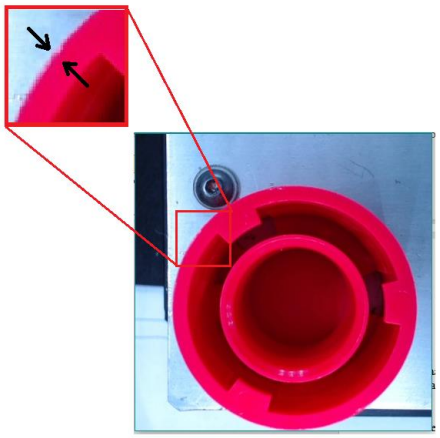
Esto ayuda a determinar si un píxel forma parte de un borde o no, pudiendo darse 3 casos:

- Si el valor del píxel es mayor que el umbral máximo, el píxel se considera parte del borde.
- Un píxel se considera que no es borde si su valor es menor que el umbral mínimo.
- Si está entre el máximo y el mínimo, será parte del borde si está conectado con un píxel que forma ya parte del borde.

2.10.4 Detector de bordes Canny con OpenCV

Por bordes de una imagen se refiere a los píxeles donde hay un cambio de intensidad que en imágenes como es la pieza dentro de un entorno, hay un fuerte contraste entre el fondo y la pieza, donde el objetivo es poder detectar este borde a través de la primera derivada (ver figura 7).

Figura 7. Borde de una pieza



Fuente: Kevin Mosquera

Los apartados 2.9.1, 2.9.2 y 2,9,3 vistos hasta el momento se encuentran desarrollados en la librería de OpenCV.

El método para calcular las aristas con el método de Canny se observa en la figura 8.

Figura 8. Método de Canny

```
8  # Convertimos a escala de grises
9  gris = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)
10
11 # Aplicar suavizado Gaussiano
12 gauss = cv2.GaussianBlur(gris, (5,5), 0)
13
14 cv2.imshow("suavizado", gauss)
15
16 # Detectamos los bordes con Canny
17 canny = cv2.Canny(gauss, 50, 150)
18
19 cv2.imshow("canny", canny)
20
```

Fuente: Kevin Mosquera

2.10.5 Buscar los contornos de una pieza

En este punto es necesario aclarar la diferencia entre un borde y un contorno donde los bordes son cambios de intensidad pronunciados y un contorno es una curva de puntos sin huecos o espacios, tiene un comienzo y el final de la curva termina en ese comienzo (Hernández, 2018).

El objetivo de esta fase es analizar todos los bordes detectados y comprobar si son contornos o no, para esto OpenCV emplea el siguiente método de la figura 9.

Figura 9. Código de detección de contornos

```
15
16 # Detectamos los bordes con Canny
17 canny = cv2.Canny(gauss, 50, 150)
18
19 cv2.imshow("canny", canny)
20
21 # Buscamos los contornos
22 (contornos,_) = cv2.findContours(canny.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
23
24 # Mostramos el número de monedas por consola
25 print("He encontrado {} objetos".format(len(contornos)))
26
27 cv2.drawContours(original,contornos,-1,(0,0,255), 2)
28 cv2.imshow("contornos", original)
29
```

Fuente: Kevin Mosquera

2.11 Elementos utilizados en la estación de visión

Para realizar una restauración de la estación de visión es necesario implementar nuevo hardware y software como es el controlador con sus respectivos programas, la cámara necesaria para dicha aplicación y otros elementos que se detallan en los siguientes subtemas.

2.11.1 Cámara con autoenfoque Logitech C922

La cámara C922 tiene un lente de cristal con enfoque automático y un campo visual diagonal de 78°. La resolución de streaming Full HD captura todos los detalles, el brillo, los colores naturales y ofrece video fluido a 1080p a30 fps o 720p a 60 fps en HD.

2.11.2 Raspberry Pi4

Es un sistema embebido con una arquitectura ARM V8, capaz de ejecutar sistemas operativos como Linux, Windows, o su propio sistema operativo (OS), ya que cuenta con todo lo necesario para dichos propósitos además de poseer pines digitales externos que ayudan a desarrollar proyectos como visión artificial. El OS que se usó fue el propio de Raspberry llamado Raspberry PI OS ejecutado en 64 bits.

2.12 Lenguaje de Programación Python

Python es un lenguaje de programación multiplataforma y de código abierto que puede utilizarse tanto para desarrollo web, creación de software y procesamiento de datos, entre muchos otros propósitos. Esta versatilidad y facilidad para aprenderlo (es ampliamente considerado el lenguaje más sencillo de aprender) lo han convertido en el lenguaje de programación más popular del mundo.

2.12.1 Liberia "Open CV"

Es una biblioteca libre de visión artificial originalmente desarrollada por Intel. OpenCV significa Open Computer Vision (Visión Artificial Abierta). En la actualidad se la sigue mencionando como la biblioteca más popular de visión artificial.1 Detección de movimiento, reconocimiento de objetos, reconstrucción 3D a partir de imágenes, son sólo algunos ejemplos de aplicaciones de OpenCV.

2.13 Protocolos de Comunicación

De forma simplificada, los protocolos son conjuntos de condiciones o reglas que habilitan el intercambio de información entre dispositivos. Estos equipos, al quedar intercomunicados o pasar bloques de datos, conforman una red de comunicación.

Como todo en los sectores de producción, los protocolos de comunicación industrial han evolucionado con el paso del tiempo. Los sistemas de producción han cambiado y es por eso que es necesario modificar la forma en que las máquinas y dispositivos intercambian información. Para este proyecto se han implementado dos tipos de protocolos, el primero es Industrial Ethernet, que es ideal para los ambientes industriales hostiles y el otro protocolo es AS-I, que es un bus de comunicación conocido por su robustez y flexibilidad, fue desarrollado especialmente para los niveles más bajos en los procesos de control. Trabaja mediante el estándar IEC 62026-2, por lo que es sencilla su conexión a diversos controladores.

CAPÍTULO 3

3 MODELADO DE LA ESTACIÓN DE VISIÓN

En este capítulo se realiza el modelado de la estación de visión para el laboratorio del MPS, con lo que es necesario contar con una cabina que aísle variantes de la luz exterior el cual se lo construyó utilizando perfiles de aluminio y acrílicos oscuros, en donde la cámara y luces led están ubicados en la parte interna de esta cabina.

Esta estación al encargarse del control de calidad por visión artificial tiene la obligación de procesar de manera automática las imágenes de las piezas maquinadas en el MPS, por lo que es necesario de un sistema embebido (Raspberry) que cumpla con las condiciones de procesamiento requeridas para este proyecto.

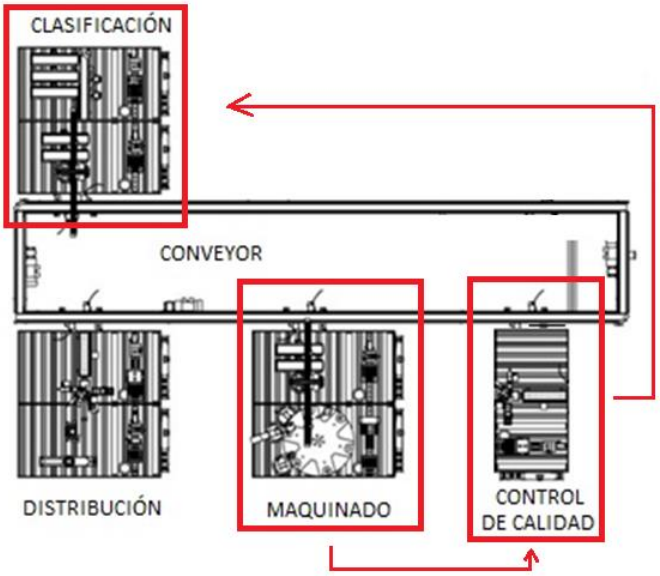
Antes de iniciar es necesario describir como es el proceso de producción del MPS, para posteriormente realizar el análisis en el ámbito del control de calidad, para poder entender en qué momento una pieza se le considera que está en buen estado y cumple con las especificaciones preestablecidas.

3.1 Descripción del proceso del MPS

Los procesos que implica el Sistema de Producción Modular (MPS) son de Distribución, Maquinado, Control de calidad (Visión), Almacenamiento, Clasificación.

A estos procesos mencionados anteriormente necesitan enlazarse por medio de la comunicación AS-I, que posee la banda transportadora, para desplazar las diferentes fichas de un proceso a otro. En la figura 10 se puede apreciar cómo es la distribución de todas estaciones.

Figura 10. Distribución de estaciones de MPS

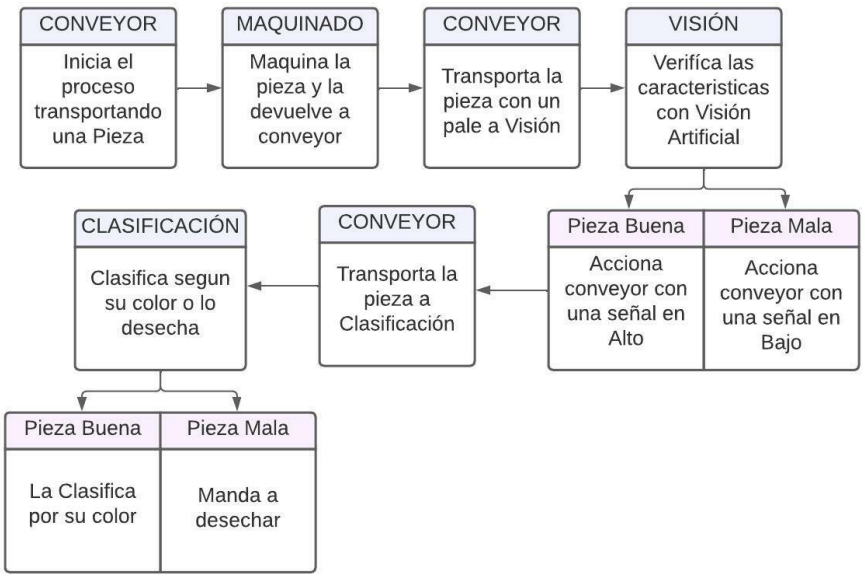


Fuente: Kevin Mosquera

3.2 Estaciones involucradas en un proceso de producción

En la figura 11 se ilustra un diagrama de bloques, en donde se generaliza el funcionamiento para la clasificación de piezas por color en el MPS, indicando una cadena industrial constituida por diferentes etapas y subprocesos.

Figura 11. Principio de funcionamiento del sistema de MPS

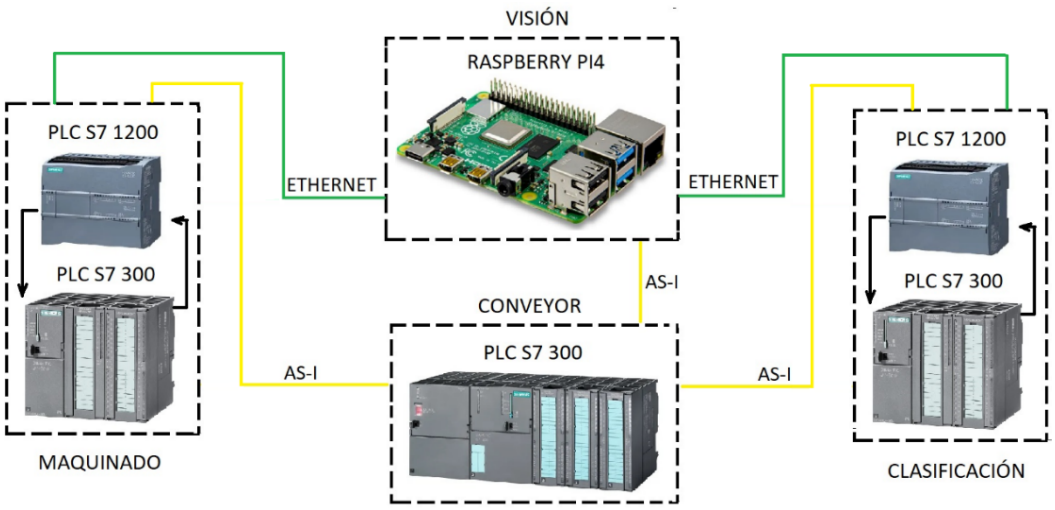


Fuente: Kevin Mosquera

3.3 Incorporación de la estación de Visión en el MPS

En la figura 12 se observa una arquitectura de comunicación a nivel de proceso, la cual permite flexibilidad para la utilización de dos tipos de protocolos (AS-I y Ethernet), mediante la heterogeneidad del sistema al incluir una tarjeta Raspberry.

Figura 12. Comunicación entre estaciones



Fuente: Kevin Mosquera

Para realizar las comunicaciones se han utilizado dos modelos de PLC en cada estación ya que los PLC's que viene por defecto son los S7 300 y no disponen de un módulo de comunicación por Ethernet, por lo que se ha decidido utilizar un PLC que ya dispone en su arquitectura un puerto Ethernet y así facilitar dicha comunicación con el resto de los dispositivos compatibles.

Para comunicar la estación de Visión por AS-I con las otras estaciones fue necesario siempre una comunicación con el conveyor porque este cuenta con módulos en los cuales reciben o proporcionan señales en alto o bajo según lo programado, en donde la Raspberry recibe y devuelve estas señales a través de los pines GPIO'S para poder iniciar el escaneo y enviar pulsos en alto o bajo según el estado de la pieza.

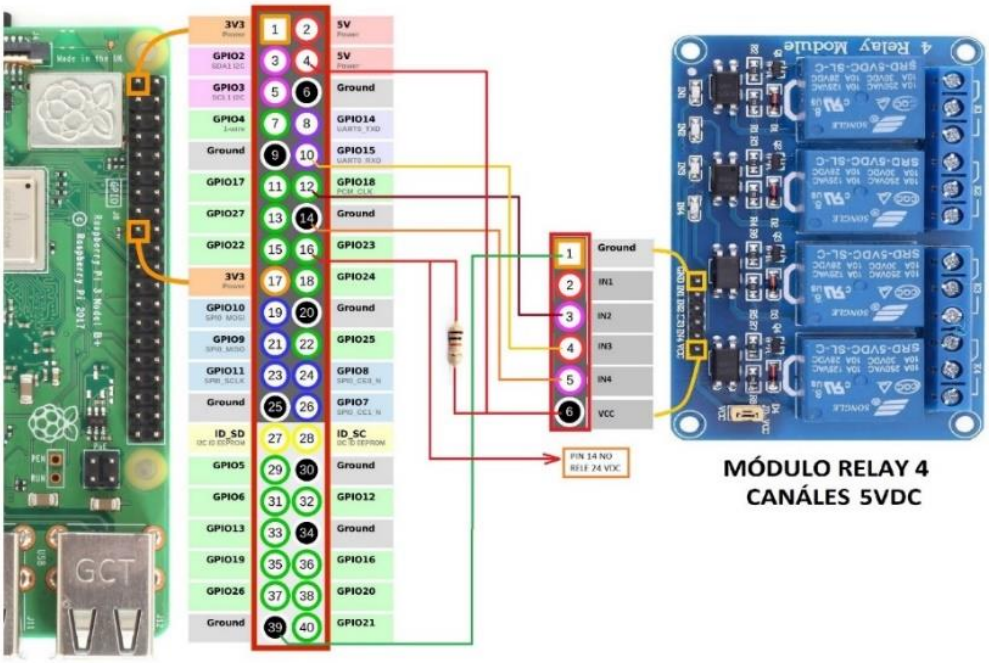
Por otro lado, para poder realizar la comunicación por Ethernet la comunicación se realiza desde la Raspberry a cada PLC S7 1200 de maquinado y de clasificación, pero estos dos últimos siguen teniendo una comunicación AS-I con el conveyor, en donde ahora la Raspberry (Visión), comienza a leer los datos de que le proporciona el PLC de maquinado el cual le indica que ya terminó de maquinar y ya envío el conveyor hacia Visión, después

de que se analice la pieza, la Raspberry envía un dato en alto o bajo según el estado de la pieza a clasificación para que esta estación entienda que debe mover el conveyor y además saber si debe clasificar o desechar cuando llegue la pieza.

3.3.1 Conexión entre Raspberry (Estación de Visión) / módulo AS-I (Conveyor)

Para poder comunicar la Raspberry con el conveyor y este a su vez tenga una comunicación por AS-I con las otras estaciones se usó los pines GPIO de este sistema embebido como entradas y salidas digitales, en el siguiente diagrama se expresa que pines fueron usados para esta comunicación además de a ver implementados relés de 5vdc y de 24vdc para manejar señales de ambos sistemas.

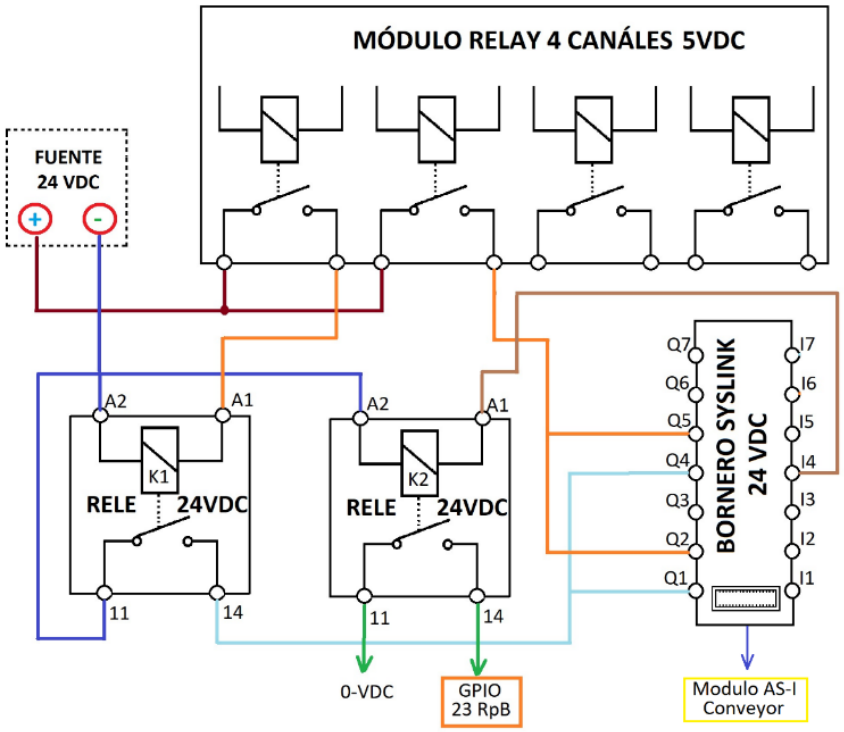
Figura 13. Diagrama de conexión de Raspberry



Fuente: Kevin Mosquera

Como se aprecia en la figura 13, se han ocupado los pines GPIO de la Raspberry para poder controlar el módulo de relés, y de esta forma poder utilizar voltajes de otro valor como es de 24 VDC.

Figura 14. Conexión entre Relevadores y bornero Syslink



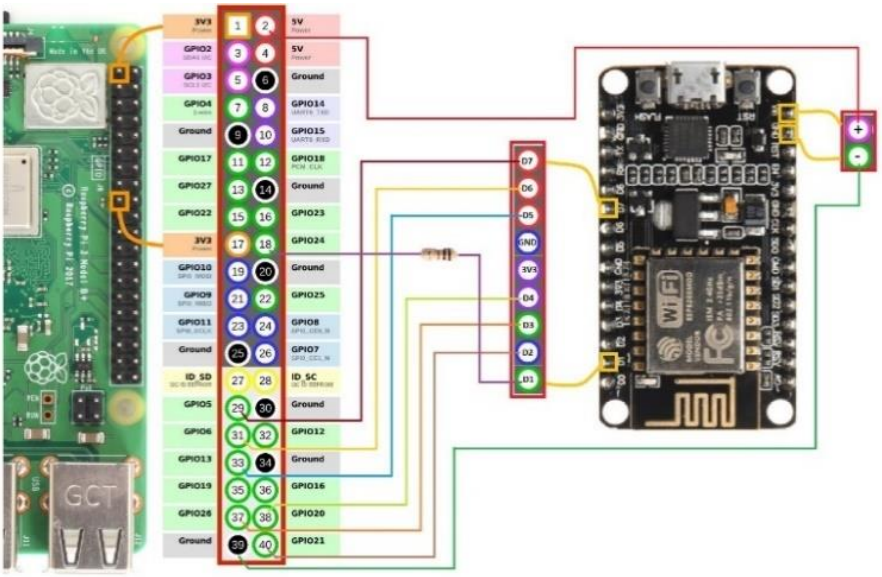
Fuente: Kevin Mosquera

En la figura 14 se aprecia el diagrama eléctrico de las conexiones para recibir y enviar señales a 24 VDC, las cuales son necesarias para la comunicación AS-I que solo maneja señales en este voltaje, haciendo uso de relevadores (relay) a 24VDC y 5 VDC.

3.3.2 Conexión entre Raspberry / NodeMCU en la estación de Visión

Para tener una comunicación con el bróker en la nube se utilizó un microcontrolador conocido como NodeMCU, que al integrar un módulo WiFi se puede tener un control y supervisión con cualquier Dashboard preconfigurada. A este NodeMCU se lo puede configurar para enviar datos a la nube por medio de una conexión inalámbrica de los datos que son necesarios enviar a la nube como es la cantidad de piezas en buen o mal estado, además del color a la que pertenece. Cabe mencionar que se utilizó esta placa ya que la propia Raspberry tuvo un conflicto al intentar enviar datos a la nube cuando estaba conectada por Ethernet.

Figura 15. Conexión con NodeMCU



Fuente: Kevin Mosquera

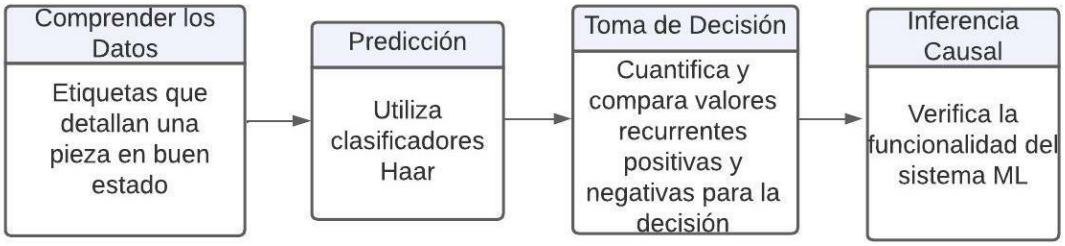
La figura 15 indica cada pin GPIO de la Raspberry que se conecta con el del NodeMCU, que corresponde a un color y a un estado, es decir, cuando el pin D2 del NodeMCU recibe una señal en alto, este entiende que debe enviar a la nube que una pieza de color rojo se encuentra en buen estado, así con los demás pines, excepto con el Pin D1, que este fue destinado para que el NodeMCU pueda controlar el accionamiento de la estación por medio pulso hacia la Rasberry.

3.4 Visión Artificial

De forma general se indica el proceso de reconocimiento de una pieza dentro de su entorno de trabajo (estación de visión) utilizando un algoritmo de aprendizaje supervisado (Supervised Machine Learning) por el método de clasificación visto en el apartado 2.6, en el cual se genera un modelo predictivo basados en datos de entrada y salida. En donde se debe tener varias imágenes de las piezas como datos de entrenamiento el cual realiza un ajuste al modelo inicial. Es así como el algoritmo aprende a clasificar las muestras de entrada comparando el resultado del modelo y la etiqueta real de la muestra para la identificación de una pieza dentro del espacio negativo en la estación de visón. Esto es posible por la estructura que presenta una ML, tal como se observa en la figura 10. Una vez que ML haya dado luz verde de la existencia de una pieza en el área de trabajo, la cámara captura una imagen y la procesa (visión por computadora), definiendo los rangos

HSV que poseen los colores en las piezas. Finalmente, a través de un algoritmo de reconocimiento de bordes explicado en la sección 3.4.3, se obtiene como resultado de salida del sistema, el estado actual de la pieza (buena o mala) (ver figura 16).

Figura 16. Pilares de aprendizaje supervisado



Fuente: Kevin Mosquera

3.4.1 Aprendizaje

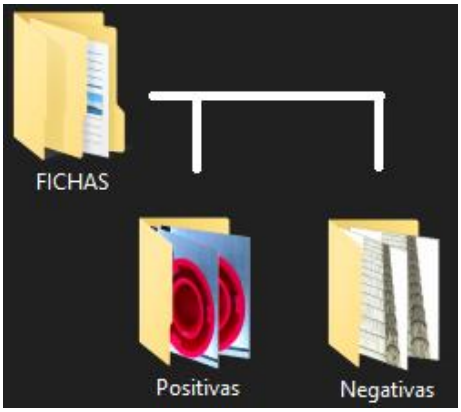
Para el aprendizaje fue necesario hacer uso del programa Cascade Trainer disponible para Windows, el cual utiliza el método de clasificadores en cascada basados en funciones de Haar en donde se necesita una gran cantidad de imágenes con tamaños máximos de 64x64 pixeles que contengan la pieza a reconocer, a estas imágenes se las denomina positivas, pero también es necesario cargar imágenes del entorno en donde no se encuentre la pieza es decir imágenes del entorno las cuales se les denomina negativas. Al terminar el aprendizaje este programa entrega un archivo con extensión XML propio para la lectura con la librería de OpenCV dentro del lenguaje de Python.

Tras varias pruebas al final en el aprendizaje se usó un total de novecientas imágenes positivas de la pieza que se deseaba que reconozca la estación y quinientas imágenes negativas del espacio donde podría estar alguna pieza, un resultado de ML aplicado en la estación de visión es el que se muestra en la figura 23(a).

3.4.2 Cargar imágenes al programa Cascade Trainer

Dentro del programa Cascade Train ir a la pestaña de Train, y seleccionar la carpeta donde se almacenan las imágenes positivas y negativas (ver figura 17).

Figura 17. Carpeta y Subcarpetas con las imágenes a entrenar



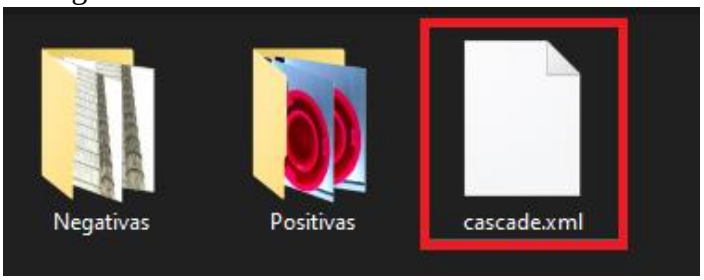
Fuente: Kevin Mosquera

3.4.3 Selección de parámetros

Seleccionar la subpestaña de Cascade y establecer el alto y ancho de las imágenes que se van a procesar, es importante que tanto las positivas como las negativas sean del mismo tamaño, y por último seleccionar en Start, esto tomará un par de minutos dependiendo de la cantidad de imágenes que se carguen en el programa.

Por último, después del entrenamiento el programa entrega un archivo con extensión XML, capaz de ser leído por Opencv, en Python (ver figura 18).

Figura 18. Archivo XML de entrenamiento

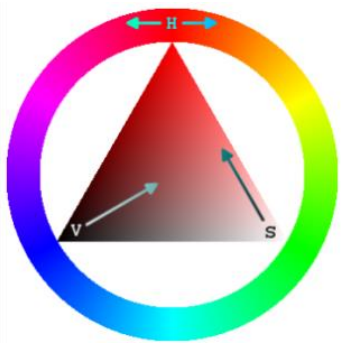


Fuente: Kevin Mosquera

3.4.4 Detección de Color con HSV (Hue, Saturation, Value)

Para la detección de colores se hizo uso del espacio de color HSV (Matiz, Saturación, Brillo) de OpenCV en Python que posee 3 componentes, similar al espacio de color RGB, por lo que resulta más fácil determinar los rangos de los colores que se desea detectar (ver figura 19).

Figura 19. Espacio de color HSV



Fuente: (Lledó, 2022)

Es necesario conocer que valores pueden tomar cada uno de los canales de HSV en OpenCV. Los valores para configurar son 3 H (0-179), S (0-255), V (0-255), hay que tener en cuenta que cada programa tiene su propio rango de valores.

3.4.5 De BGR a HSV

Por defecto OpenCV lee imágenes o fotogramas en BGR, por ello es necesario transformarlas al espacio de color HSV para lo cual se utiliza la función cv2.cvtColor; donde se transforma la imagen como primer argumento y como segundo argumento se utiliza la función cv2.COLOR_BGR2HSV, como se muestra en la figura 20.

Figura 20. Transformación de BGR a HSV en Python

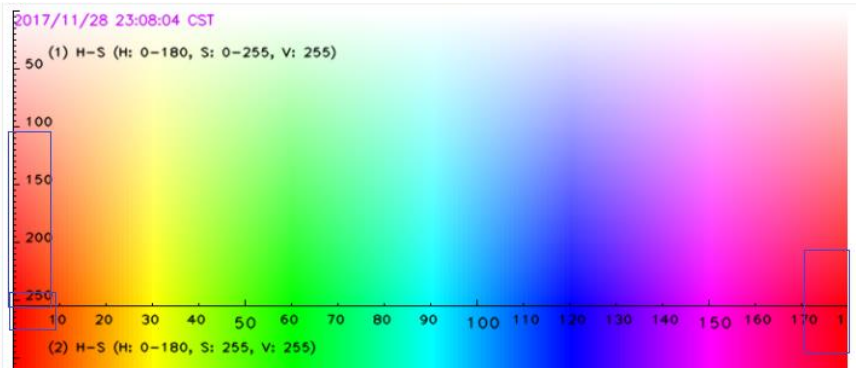
```
1.  import cv2
2.  import numpy as np
3.
4.  cap = cv2.VideoCapture(0)
5.
6.  while True:
7.    ret,frame = cap.read()
8.    if ret==True:
9.      frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
10.     cv2.imshow('frame', frame)
11.     if cv2.waitKey(1) & 0xFF == ord('s'):
12.       break
13. cap.release()
14. cv2.destroyAllWindows()
```

Fuente: Kevin Mosquera

3.4.6 Rangos de colores y su detección

En la siguiente figura de componentes HSV se observa que en H va de 0 a179 correspondientes a los colores: rojo, naranja, amarillo, verde, azul, violeta y nuevamente rojo (ver figura 21).

Figura 21. Vista de los componentes HSV



Fuente: (Solano, 2019)

Entonces se puede determinar dos rangos para el color rojo que está presente al principio y al final, el primer rango se lo determina de 0 a 8, y el segundo de 175 a 179 en H, mientras que para los componentes S va desde 100 a 255, y V va desde 20 a 255 respectivamente (ver figura 22).

Figura 22. Código para extraer el color rojo de una imagen

```
5.
6.   redBajo1 = np.array([0, 100, 20], np.uint8)
7.   redAlto1 = np.array([8, 255, 255], np.uint8)
8.
9.   redBajo2=np.array([175, 100, 20], np.uint8)
10.  redAlto2=np.array([179, 255, 255], np.uint8)
11.
12.  while True:
13.    ret,frame = cap.read()
14.    if ret==True:
15.      frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
16.      maskRed1 = cv2.inRange(frameHSV, redBajo1, redAlto1)
17.      maskRed2 = cv2.inRange(frameHSV, redBajo2, redAlto2)
18.      maskRed = cv2.add(maskRed1, maskRed2)
19.      maskRedvis = cv2.bitwise_and(frame, frame, mask= maskRed)
20.      cv2.imshow('frame', frame)
21.      cv2.imshow('maskRed', maskRed)
22.      cv2.imshow('maskRedvis', maskRedvis)
23.      if cv2.waitKey(1) & 0xFF == ord('s'):
24.        break
```

H S V

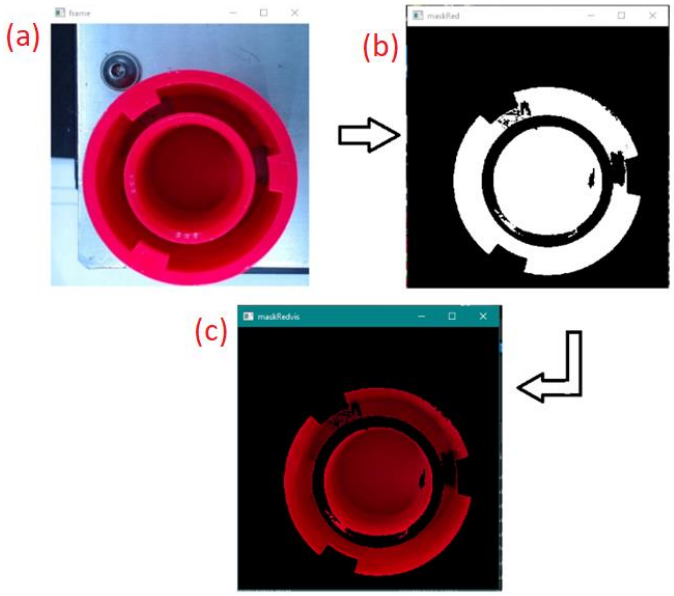
Fuente: (Solano, 2019)

A partir de aquí, se obtienen imágenes binarias maskRed1 y maskRed2 (línea 15, 16 de la figura 15), las cuales serán sumadas en maskRed, esta acción tiene como objeto interpretar que el color blanco refleje el lugar donde están presentes el primer y segundo rango, caso contrario ocurre en el color negro (ausencia de rangos).

3.4.6.1 Visualización

Una vez que se obtiene la imagen como el de la figura 23(a), esta es procesada a una imagen binaria como se explicó en el anterior apartado, consiguiendo un color blanco, que representa el color rojo según los rangos de HSV, los cuales se pueden apreciar en la siguiente figura 23(b), donde se visualiza la detección del color rojo de la pieza.

Figura 23. Detección de colores de la ficha



Fuente: Kevin Mosquera

Como se observa en la anterior figura, el algoritmo deja en color negro todo el entorno que no sea del color que se hayan especificado en este caso color rojo, ver figura 23(c), de esta manera se aplica este procedimiento para el resto de las piezas de diferentes colores.

Cabe mencionar que para aquellas piezas de color negro no existe un rango defino según HSV, es así que el medio lumínico controlado permitió manejar valores dentro del rango HSV.

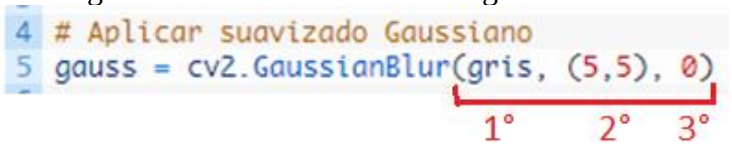
3.4.7 Obtención de caracteres de una pieza

Se procede con un algoritmo iterativo para detección de borde Canny, el cual permite la obtención de caracteres de una pieza, aislando los objetos y separándolos del fondo, esto debido a que la imagen que entrega ML, ver figura 23(a), posee características positivas y negativas, es por esta razón que se debe resaltar aquellos rangos de detalles hasta llegar a la convergencia minimizando el valor de diferencia del rango de detalle actual respecto al anterior de la pieza.

3.4.8 Conversión a escala de grises y suavizado Gaussiano

Para iniciar con la obtención de caracteres de una pieza es necesario partir de una imagen en escala de grises con el método cv2.cvtColor que convertirá la imagen en color en una imagen en escala de grises para luego suavizarla a través de la técnica gaussiana eliminando así el ruido.

Figura 24. Método de suavizado gaussiano



Fuente: Kevin Mosquera

El método cv2.GaussianBlur() de la figura 24 toma 3 parámetros. El primero es la imagen que se desea suavizar que en este caso la imagen en escala de grises, el segundo parámetro es el núcleo o máscara de convolución y el último parámetro es sigma (σ) y representa la desviación estándar en el eje X, es decir, el ancho de la campana gaussiana dando como resultado la imagen que se muestra en la figura 25.

Figura 25. Suavizado Gaussiano

Fuente: Kevin Mosquera

3.4.8.1 Calcular el detector de bordes Canny con OpenCV

Una vez que se haya suavizado la imagen, se puede eliminar los bordes no suprimidos al máximo y aplicar el umbral de histéresis.

Figura 26. Método Canny

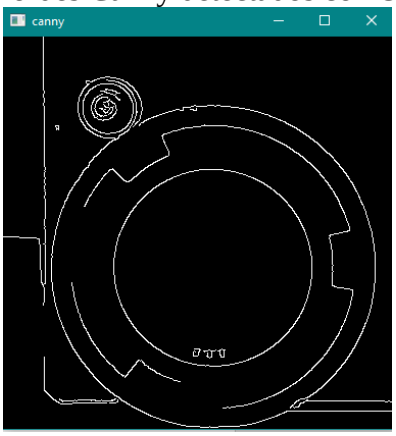
```
2 canny = cv2.Canny(gauss, 50, 150)
3
```

1° 2° 3°

Fuente: Kevin Mosquera

Estos dos procesos se realizan a través del método cv2.Canny de la figura 24 que admite 3 parámetros: el primero es la imagen donde se desea detectar los bordes. Esta imagen es la que ya se suavizado en el paso anterior, el segundo parámetro será el umbral mínimo y el tercer parámetro será el umbral máximo y finalmente se vuelve a mostrar la imagen con los bordes detectados, como se muestra en la figura 27.

Figura 27. Bordes Canny detectados con OpenCv



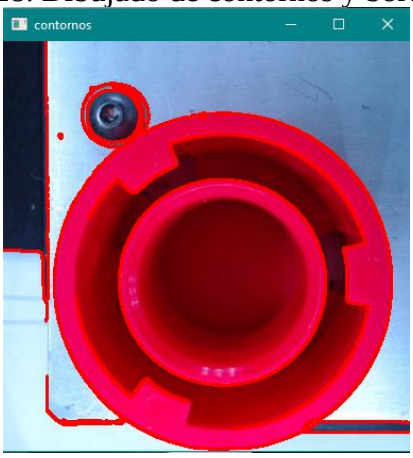
Fuente: Kevin Mosquera

3.4.8.2 Dibujar y detectar contornos

Una vez que se tiene los bordes, se debe comprobar si son contornos (caracteres) o no para desecharlos, es así que empleando el método cv2.findContours() se detectan los contornos y específicamente con la subrutina (cv2.RETR_EXTERNAL) se detecta los contornos externos, haciendo así una aproximación para eliminar los píxeles del contorno redundantes a través de (cv2.CHAIN_APPROX_SIMPLE).

Finalmente, se pinta los contornos en la imagen original con el método cv2.drawContours(), como muestra la figura 28.

Figura 28. Dibujado de contornos y bordes



Fuente: Kevin Mosquera

3.4.9 Decisión de pieza en buen o mal estado

Como se mencionó con anterioridad, gracias al aprendizaje supervisado por el método de clasificadores en cascada basados en funciones de Haar, fue posible la identificación de una pieza dentro de un entorno controlado además, utilizando visión por computadora se verificó su color y se obtuvo la cantidad de caracteres mediante la detección de contornos, es ahí donde se aplica una tolerancia de contornos en la imagen desde un mínimo a un máximo.

Por ejemplo, una imagen de una pieza en buen estado tendrá ‘N’ números de contornos, considerando que no existen perturbaciones durante el tiempo real de la captura de la imagen (cambio de posición o por rotación de la pieza), en este contexto y sin importar el número de testeos que se realicen, un operario colocará por única vez y manualmente los rangos que correspondientes a cada pieza, dando lugar a que el sistema automáticamente rechace aquellas piezas que estén fuera del rango que corresponda a cada color.

El escaneo también toma en cuenta como contornos a los objetos que estén dentro de la imagen, así que sumará estos en el resultado final, pero como este entorno será siempre una constante, esta suma de objetos no repercuta con cualquier pieza.

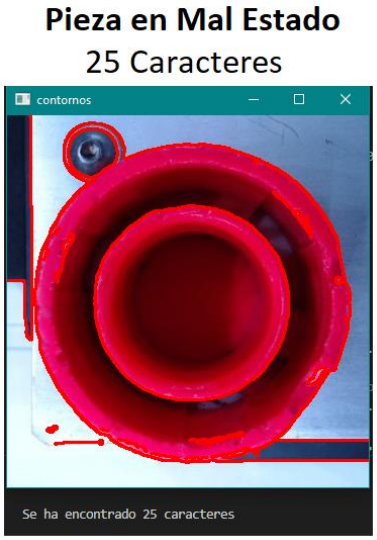
Figura 29. Rangos en los que la pieza se cataloga como en buen estado

Fuente: Kevin Mosquera

Como se observa en la figura 29, las dos piezas se encuentran en la misma posición y dentro del mismo entorno, pero en solo dos testeos ha habido una variación de tres unidades y si se lo sigue testeando podría tener otra cantidad de caracteres, pero no se alejarán de los ya indicados, a esto se le conoce como tolerancia mínimo-máximo de rango.

De los antes mencionado se considera que la tolerancia de rango de la figura 29 tendrá un valor mínimo de 12, hasta un valor máximo de 15, y esto indicaría que cualquier pieza roja que pase por la estación de visión del MPS y se encuentre en este rango seria considerado como pieza de buen estado, caso contrario será desechada en la siguiente estación (clasificación).

Figura 30. Rango en que la pieza se encuentra en mal estado



Fuente: Kevin Mosquera

Complementado lo mencionado en los párrafos anteriores, en la figura 30 se aprecia una pieza con deformaciones en sus contornos, la cual al escanearla da un resultado de 25 contornos, por ende, esta pieza se encuentra fuera del rango establecido y se la cataloga como pieza en mal estado.

3.5 Estructura de la Estación de Visión

Para la creación de la estructura se realizó un análisis del espacio disponible para determinar el tamaño y los materiales para su construcción por lo cual se tomó en consideración el ancho de la banda y la altura del pallet incluida la pieza.

Para la selección del material a usar se realizó un cuadro de decisión en el que se indica los diferentes tipos de materiales calificándolos del 1 al 5 de acuerdo con su funcionalidad.

Tabla 1. Valoración de los materiales a usar para la cabina

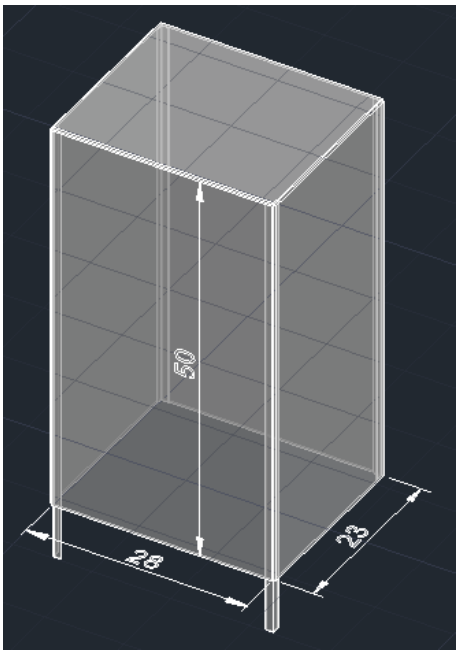
MATERIAL	FRAGILIDAD	TRANSPARENCIA	HUMEDAD	TOTAL
Vidrio	1	5	5	11
Acrílico	5	5	5	**15**
Metacrilato	3	5	2	10

Fuente: Kevin Mosquera

Como se observa en la tabla 1, el material mejor puntuado ha sido el acrílico, el cual es con el que se construyó la cabina que contiene la cámara en su interior.

A continuación, se presenta la estructura diseñada en el software Autocad antes de su creación física, su forma rectangular se debe al acoplamiento con el conveyor, el pallet y la pieza.

Figura 31. Estructura de la Estación



Fuente: Kevin Mosquera

3.6 Overhauling de la estación de visión

En esta sección se explica cómo fue la implementación de esta estación dentro del entorno del MPS, tras una caracterización de los dispositivos del propietario que fueron adquiridos desde el 2008, en donde se identifica que la parte física y los controladores de visión se encuentran defectuosos y con versiones caducas respectivamente, con lo cual se cambia por un procesador cortes ARM cuadCore, una cámara Logitech autoenfocable y una estructura que fue mencionado en la sección 3.5. Para su implementación fue necesario acoplar una nueva estructura de cabina y realizar las conexiones eléctricas que permitan la comunicación posterior y anterior de la estación de visón. A demás, se empleó un lenguaje de programación en Python, el cual permitió desarrollar ML, visión por computadora (color y contornos) y los protocolos de comunicación AS-I y Ethernet y

envío de datos hacia la nube para que un operario a través de autenticación ingrese al sistema y pueda descargar un registro del estado de las piezas.

3.7 Comunicación de Estaciones

Para la implementación del sistema de monitoreo y control, se usó una Raspberry Pi4 de 8gb de memoria RAM conjuntamente de una micros SD de 64 gb, para controlar todo el proceso; un monitor para visualizar y cargar datos, una cámara digital de marca Logitech para obtener una imagen en tiempo real de las piezas a examinar; todas estas imagen o datos son enviados a la Raspberry donde se desarrolló una aplicación con la programación en Python, que permite procesar estas imágenes y coordinar con las otras estaciones para decidir; si dicha pieza se puede almacenar o desechar y posteriormente subir esta información a una base de datos que sirva como información para ciencia de datos (análisis, procesamiento y tendencia).

Para la selección del hardware fue necesario realizar un análisis del proceso que se desea automatizar, considerando a un futuro la escalabilidad de la planta de producción.

El propietario de Raspberry provee de un sistema operativo (Raspberry Pi OS) que se lo puede instalar en una micro SD para el manejo de su hardware y revisar la compatibilidad de los elementos embebidos en la tarjeta, así también la característica de la fuente en función de la potencia absorbida por dichos dispositivos.

La tarjeta en cuestión debe manejar intercambio de información entre: PLC’s S7-1200 con protocolo Ethernet y PLC S7-300 con un módulo AS-I.

3.8 Configuración de comunicación desde Raspberry a PLC S7 1200

En este apartado se describen las configuraciones y programaciones necesarias para la comunicación entre la Raspberry (estación de visión) y el resto de las estaciones (maquinado y clasificación).

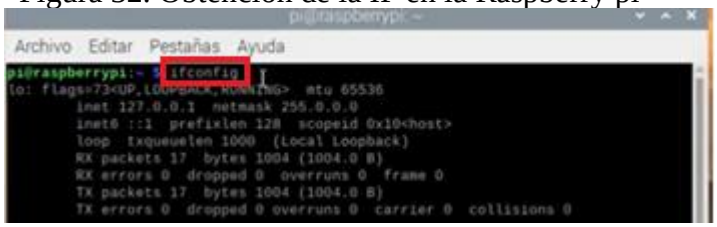
3.8.1 Configuración de Raspberry Pi OS

Cuando se inicia Raspberry Pi OS mostrará un mensaje de bienvenida, en él se puede configurar los parámetros básicos del sistema, como Región y Zona Horaria, Lenguaje, Contraseña, Conexión a Internet, parámetros recomendables para la actualización automática del sistema.

3.8.2 Cambio de IP dinámica a estática en Raspberry Pi (comunicación Ethernet)

Para realizar el cambio de la IP se debe dirigir a consola de comandos, y digitar "ifconfig" para obtener la información de la Ip Automática (ver figura 32).

Figura 32. Obtención de la IP en la Raspberry pi



Fuente: Kevin Mosquera

Ahora se debe cambiar un archivo de DHCP, por lo que se debe digitar en la consola de comandos "sudi nano /etc/dhcp.com" (ver figura 33).

Figura 33. Cambio de archivo DHCP

Fuente: Kevin Mosquera

En la siguiente ventana, se debe ir hasta el final de la ventana y colocar la siguiente información, en que indica cual será la IP estática a la que se desea que la Raspberry obtenga, para este caso será la "192.168.0.167". Y cambiar de "wlan0" a "eth0" (ver figura 34).

Figura 34. Configuración de la IP de la Raspberry

Fuente: Kevin Mosquera

Luego se debe guardar la configuración con Ctrl+O, seguido de Enter, y para salir pulsar Ctrl+X, por último, se debe reiniciar al microrordenador y los cambios habrán tenido efecto.

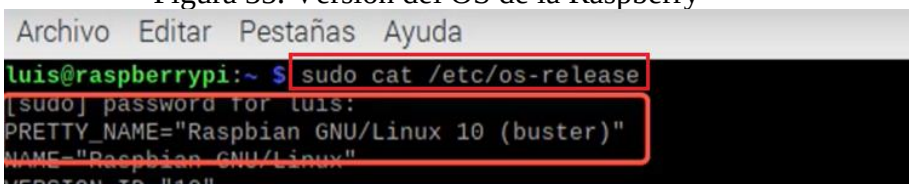
3.8.3 Instalación de librerías de visión y ML en Raspberry

Por defecto Raspberry PI OS incluye un aplicativo para programar en Python llamado Thony, sin embargo es necesario instalar otros complementos que serán de utilidad para poder realizar visión {[color (OpenCV), contornos (OpenCV)]} y ML (OpenCV), siendo la más importante la librería de Open CV en la cual se debe instalar de una manera distinta para el OS ya mencionado, explicado a detalle en el siguiente apartado pero también se han seleccionado las librerías más relevantes que se debe instalar, las cuales se explican en los apartados posteriores.

3.8.4 Librería OpenCV

Para la instalación de esta librería es necesario conocer la versión del OS, para ello en la consola de comandos digitar lo señalado en la figura 35.

Figura 35. Versión del OS de la Raspberry



Fuente: Kevin Mosquera

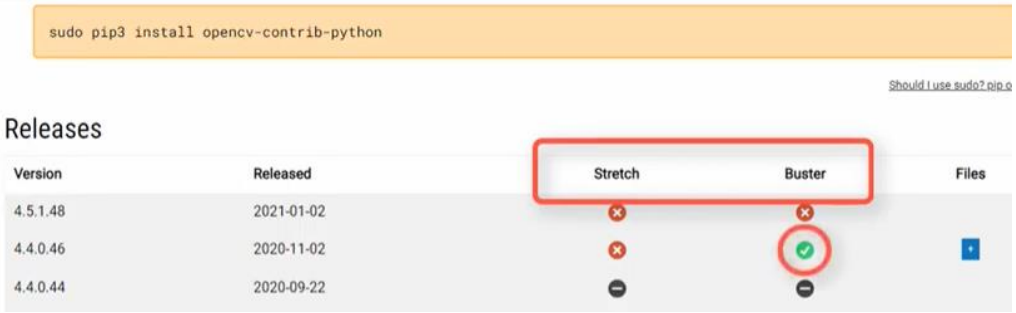
Como se puede observar en el recuadro rojo la versión es Linux 10 – búster. Ahora en la siguiente página web “https://www.piwheels.org/packages.html” se digita en la barra de búsqueda la palabra “OpenCV” y seleccionar opencv-contrib-python (ver figura 36).

Figura 36. Búsqueda de OpenCV

Fuente: Kevin Mosquera

Luego aparecerá la siguiente ventana que nos indica las versiones de OpenCV además de indicar la versión compatible con el sistema (ver figura 37).

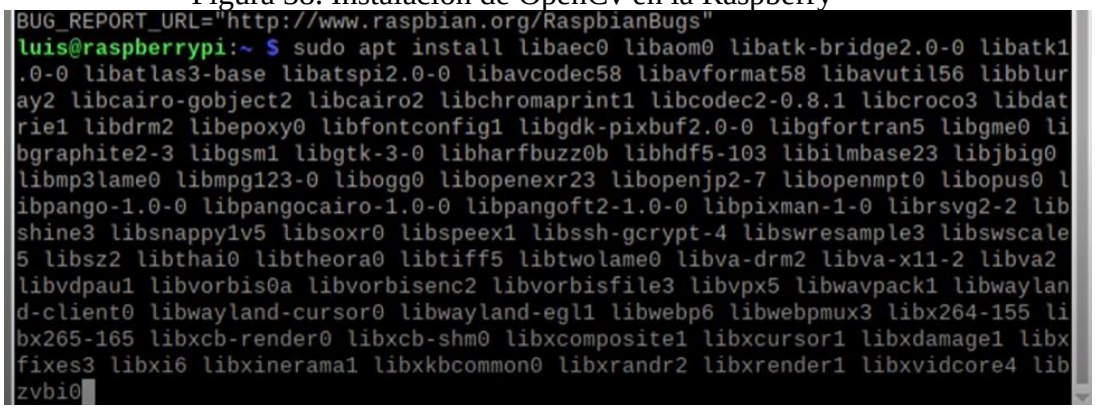
Figura 37. Selección de la versión de OpenCV



Fuente: Kevin Mosquera

Hacer clic en el icono del "+" de la versión compatible que en este caso será la 4.5.1.48 y se desplegará unas nuevas opciones que nos permitirán instalar esta librería. Con el mouse seleccionar todo el texto que está en el recuadro y copiarlo, para luego pegarlo en la consola de comandos, quedando de la siguiente manera, posteriormente presionar Enter (ver figura 38).

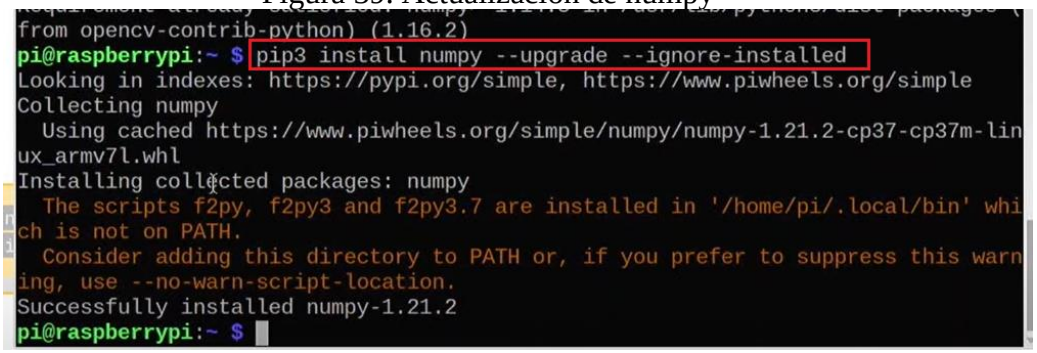
Figura 38. Instalación de OpenCv en la Raspberry



Fuente: Kevin Mosquera

Por último, digitar el siguiente comando para actualizar la librería de numpy que es otro complemento necesario para poder utilizar OpenCV correctamente (ver figura 39).

Figura 39. Actualización de numpy



Fuente: Kevin Mosquera

3.8.5 Librerías complementarias para visión y ML

Para instalar las siguientes librerías se hará desde comandos en la consola. En primer lugar, para generar una interfaz gráfica se hará uso de la librería Tkinter, y su comando de instalación es “sudo pip install tkinter”, también será necesario hacer uso de la librería de “imutils” y “os”, que sirven para crear nuevas carpetas.

Para enviar información a una nube privada (Mega) se debe instalar la librería de MegaDrive y para instalarla se debe digitar “pip install mega.py”.

Se usó las librerías “openpyxl” y “pandas” para la creación y modificación de archivos en Excel, y para su instalación a través del siguiente comando “pip install openpyxl”, y “pip install pandas” respectivamente.

Una librería importante que se usa para comunicar la Raspberry y los PLC’s S7 1200 por Ethernet, como cliente o suscriptor es la llamada “EasyModbusTCP” y para su instalación se usó el siguiente comando “pip install EasyModbus”.

3.9 Comunicación PLC S7 1200 / Raspberry

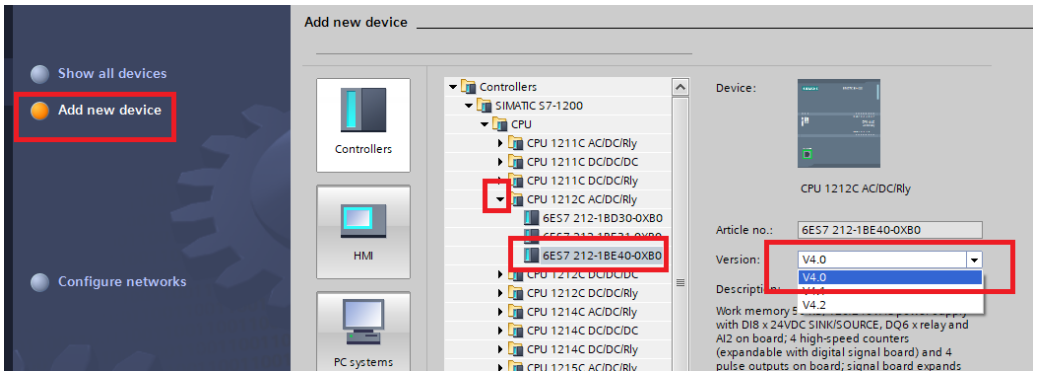
Como se ha ido recopilando en anteriores capítulos para realizar la comunicación con protocolo Ethernet es preciso que los PLC´s cuenten con un módulo Ethernet, es por esta razón que los autómatas programables S7-300 de las estaciones de maquinado y clasificación fueron reemplazados por PLC´s S7-1200.

3.9.1 Configuración de PLC S7-1200 para comunicación Ethernet

Para que dicho PLC pueda comunicarse con la Raspberry es necesario realizar una configuración con antelación, en donde se debe cambiar la IP, y asignar un bloque virtual el cual es necesario para que el PLC actúe como servidor.

En primer lugar, seleccionar el modelo de PLC junto con el CPU que se indica físicamente en el laboratorio, posteriormente seleccionar la versión V4.0, y en la parte inferior presionar en añadir (ver figura 40).

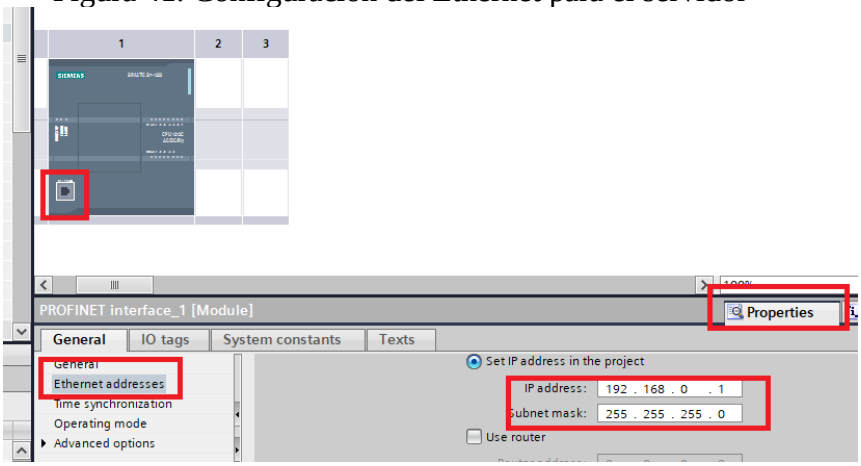
Figura 40. Configuración del PLC como servidor en tía portal



Fuente: Kevin Mosquera

Luego en el dispositivo seleccionar el puerto de ethernet del PLC, luego ir a propiedades, y configurar la IP que se desea que tenga la estación de maquinado, que en este caso seria 192.168.0.1, y en el caso de la estación de clasificación sería 192.168.0.2 y en la máscara subred digitar 255.255.255.0, que con eso estaría en el mismo entorno de Red que se encuentra la Raspberry (ver figura 41).

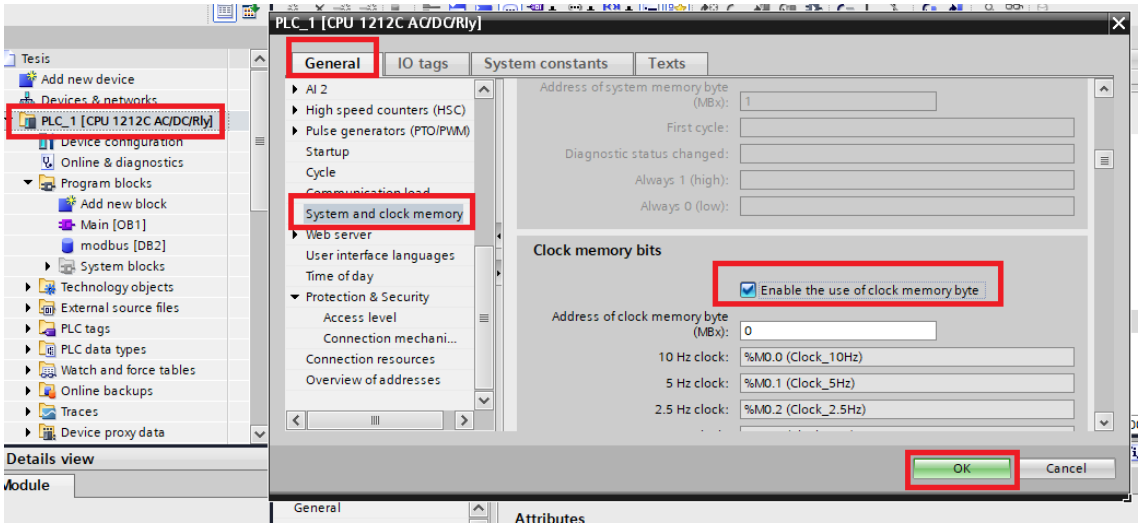
Figura 41. Configuración del Ethernet para el servidor



Fuente: Kevin Mosquera

En este punto cuando se cargue la programación al PLC este cambiará a la IP que se le ha dejado propuesta, ahora es necesario activar una memoria de reloj que dispone el PLC, para ello es necesario hacer clic derecho en el apartado del PLC y posteriormente en la opción de Propiedades y dejar configurado tal cual se muestra en la siguiente imagen (ver figura 42).

Figura 42. Configuración de la memoria de reloj del PLC

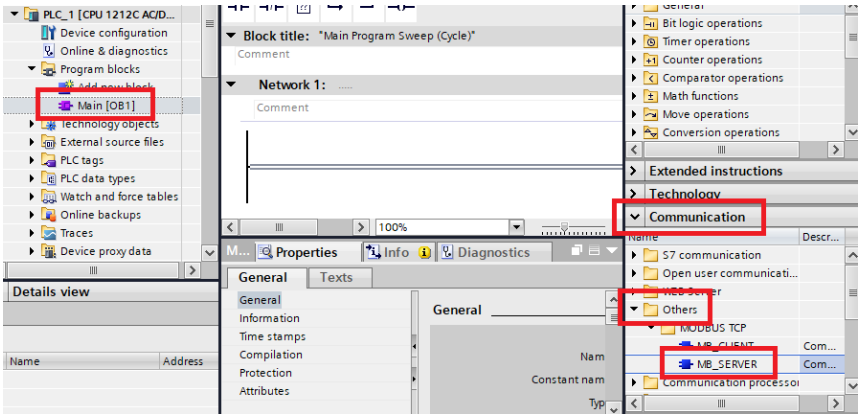


Fuente: Kevin Mosquera

3.9.1.1 Configuración de PLC S7-1200 como servidor

Para este punto se debe seleccionar el apartado de programación por bloques, y luego en la parte inferior derecha, seleccionar comunicaciones, luego otros, y por último "MB_SERVER" (ver figura 43).

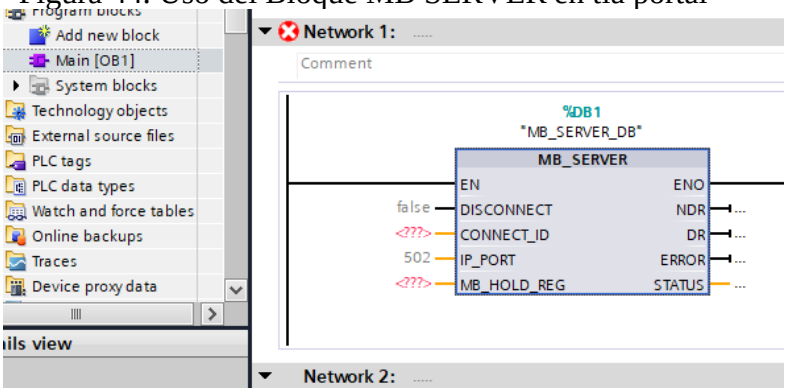
Figura 43. Configuración del servidor desde la programación



Fuente: Kevin Mosquera

Posteriormente se debe arrastrar el bloque MB_SERVER a la primera sección de programación donde debe hacer clic en OK y quedaría de la siguiente manera.

Figura 44. Uso del Bloque MB SERVER en tia portal

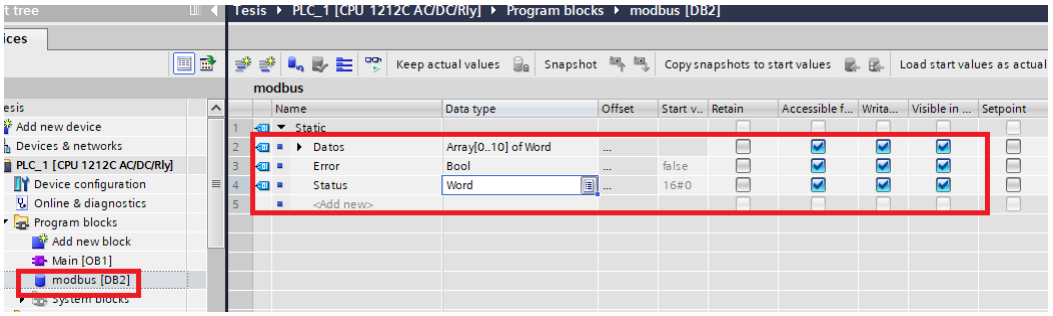


Fuente: Kevin Mosquera

Luego en el mismo aparatado de bloques de programación se debe crear un nuevo bloque, luego hacer clic en Data block, seleccionar el tipo como Global DB, renombrar a cualquier nombre que en este caso es modbus y por último presionar OK.

En este mismo bloque de modbus, crear tres nuevos datos, que el primero será de tipo Array, el siguiente Booleano, y el último como Word, (ver figura 45).

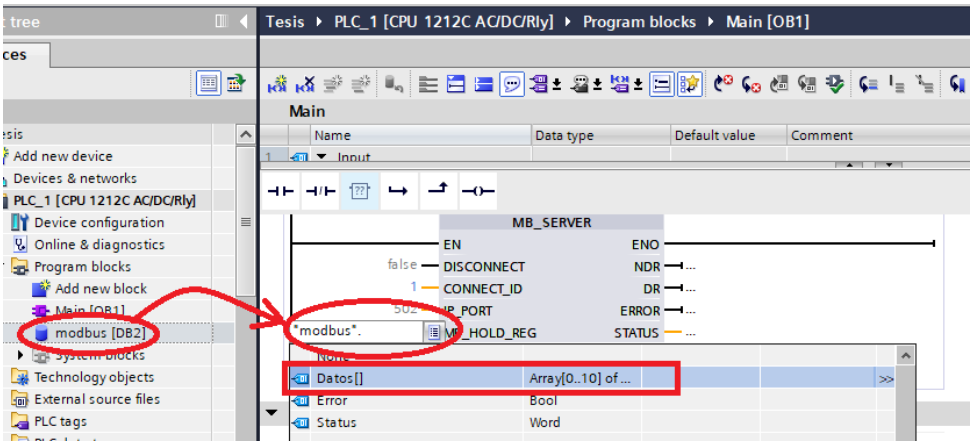
Figura 45. Creación de datos en tia portal



Fuente: Kevin Mosquera

Ahora se debe configurar el bloque que se había implementado como servidor, en el cual se debe arrastrar el bloque de dato a los pines del bloque que utilizamos como servidor (ver figura 46).

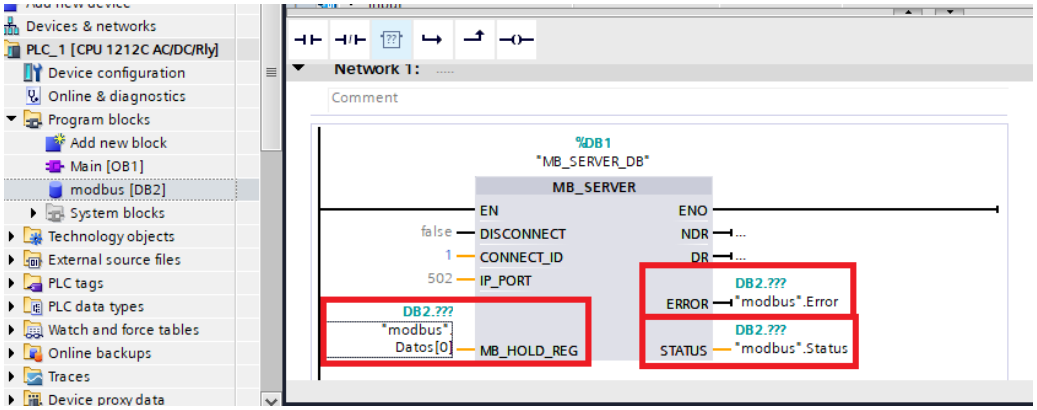
Figura 46. Enlace de los datos con el bloque de servidor



Fuente: Kevin Mosquera

El mismo procedimiento se debe realizar con el pin de Error y el de Status y debe quedar el bloque configurado como señala la figura 47.

Figura 47. Emparejamiento de datos con el servidor, parte 2



Fuente: Kevin Mosquera

Este procedimiento debe realizar independientemente para cada PLC S7-1200 que corresponde para Maquinado y Clasificación, Con esta configuración al cargarlo al PLC estará listo para tener una comunicación con cualquier cliente que se conecte a su IP, que para el proyecto será la Raspberry.

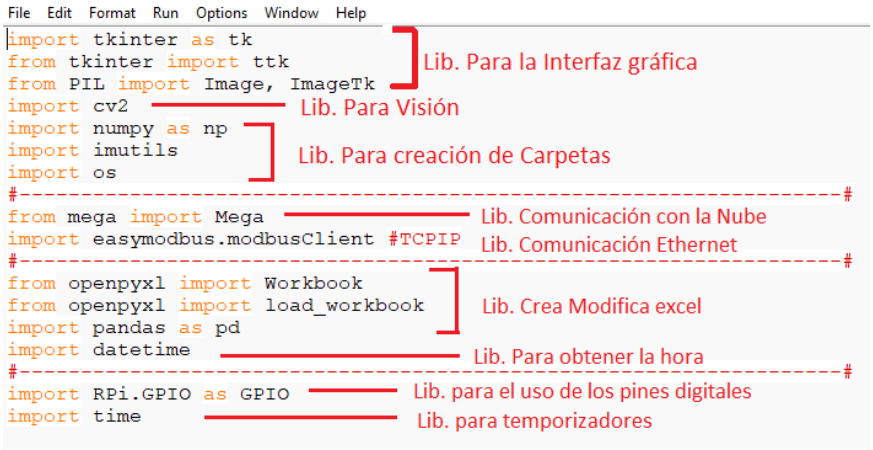
3.10 Algoritmo de Visión empleando Python (Raspberry)

En este apartado se habla de las fases que se tuvo que seguir para realizar la programación en este lenguaje y haciendo uso del software Thonny que viene preinstalado en Raspberry Pi OS el cual se tratará de aplicar diagramas de casos, para una mejor ejemplificación.

3.10.1 Importación de librerías y creación de variables

Antes de empezar la programación es necesario importar las librerías que se sean necesarias para la designación de pines, para una interfaz gráfica, para la comunicación Ethernet, para realizar visión artificial, como se muestra la figura 48.

Figura 48. Importación de librerías

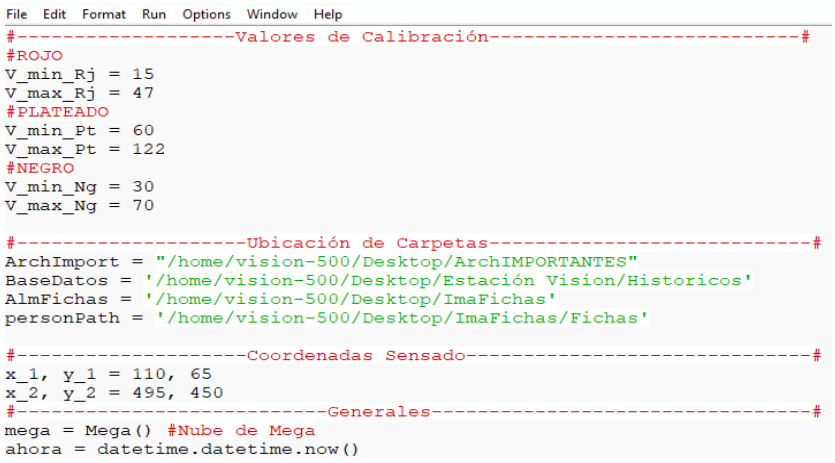


Fuente: Kevin Mosquera

3.10.2 Variables utilizadas en el algoritmo de visión

En la figura 49 se detallan algunas de las variables que se usaron para identificar una serie de operaciones o valores que se deseaba utilizar dentro del entorno de programación.

Figura 49. Creación de las variables



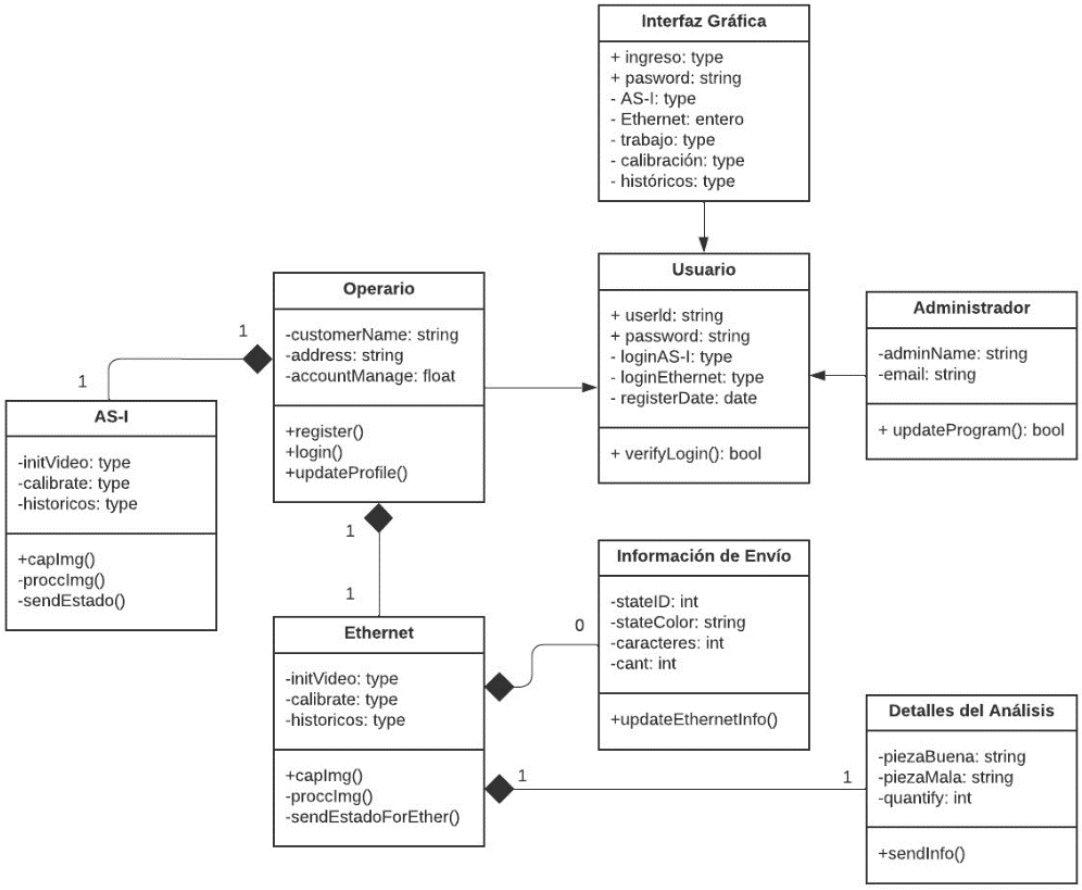
Fuente: Kevin Mosquera

Como se observa en la imagen se muestran algunas de las variables que se crearon para poder realizar la programación, como son los valores que se necesitaron para poder identificar el color de las diferentes piezas, las carpetas se fueron necesarias para almacenar la información de las imágenes que se tomaban las fotos o para la creación de la base de datos.

3.10.3 Interfaz gráfica de la estación de visión

Para realizar una interfaz gráfica se utilizó la librería llamada tkinter. Esta librería permite abrir imágenes, videos, incorporar botones de acción y admitir texto por teclado, además de poder configurar la forma de crear usuarios y contraseñas (ver figura 50).

Figura 50. Diagrama de clases de la interfaz gráfica



Fuente: Kevin Mosquera

3.10.4 Ventana de inicio (interfaz gráfica)

Para programar la ventana que se muestra al iniciar el código, se realizó una serie de comandos que dispone esta librería, como es que se muestre una imagen prediseñada con toda la información básica, también se implementó un botón que permita ingresar a la siguiente ventana (ver figura 51).

Figura 51. Programación de la ventana de inicio del programa

```
#----------------------Ventana Inicio-------------------------------------#
inicio = tk.Tk()                                    Def Variable Tk() de inicio
inicio.title("Inicio")                              Def un titulo a la vent, y esp el tamaño
inicio.geometry("1366x768+0+0")
inicio.resizable(width=False, height=False)         Selec Img a proyectar
fondo = tk.PhotoImage(file=ArchImport + "/ImagenCaratula.png")
fondo1 = tk.Label(inicio, image=fondo).place(x=0, y=0, relwidth=1, relheight=1)
#------------------------------------------------------------------------#
```

Fuente: Kevin Mosquera

Y al ejecutar dicho código lo que se observará será la siguiente imagen prediseñada pero aun faltaría programar el botón de ingreso para la siguiente ventana (ver figura 52).

Figura 52. Pantalla de Inicio del programa

Fuente: Kevin Mosquera

Ahora para programar el botón de ingreso es necesario crear nuevas líneas de código utilizando la misma librería.

Figura 53. Programación de los botones para el programa

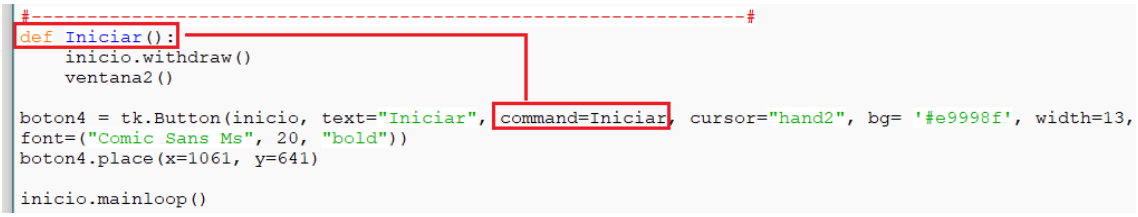
```
#------------------------------------------------------------------#
def Iniciar():
    inicio.withdraw()                    Comando para crear botones
    ventana2()
                                         Se designa el nombre, la acción, tipo de letra, etc
boton4 = tk.Button(inicio, text="Iniciar", command=Iniciar, cursor="hand2", bg= '#e9998f', width=13,
font=("Comic Sans Ms", 20, "bold"))
boton4.place(x=1061, y=641)              Indica la posición en se ubica el botón

inicio.mainloop()
```

Fuente: Kevin Mosquera

Como se puede ver en la figura 53 para programar un botón es necesario definirlo e indicar variables básicas como el nombre, una acción, tipo y tamaño de letra. Otro punto para tomar en cuenta es que las definiciones deben estar al final de cada programación de la ventana.

Ahora para asignar una acción al botón es necesario crearlo seguido de la programación de la ventana, que en este caso se le ha asignado el nombre “Iniciar”.

Figura 54. Programación de las acciones de los botones



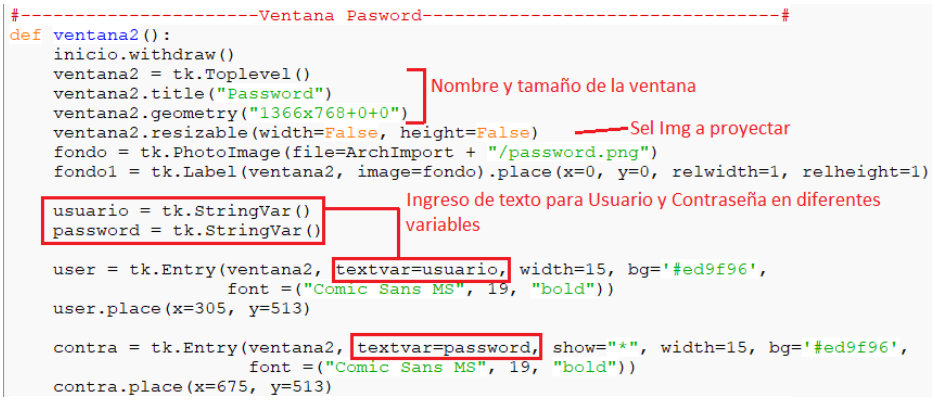
Fuente: Kevin Mosquera

En la figura 54 se ve como se declaró el botón Iniciar el cual acciona la función con el mismo nombre el cual indica que se cerrará la ventana de inicio y posteriormente abre “ventana2”.

3.10.5 Ventana para contraseña (interfaz gráfica)

En esta ventana es necesario programar un usuario y una contraseña que le permita ingresar a la venta de trabajo de AS-I o de Ethernet, el principio es el mismo que la anterior ventana solo que esta nueva es necesario dar nuevos parámetros además de botones para que se pueda insertar texto, (ver figura 55).

Figura 55. Programación del usuario y contraseña para el programa



Fuente: Kevin Mosquera

De la misma manera que la anterior ventana para crear los botones es necesario programarlos en la parte final de la ventana creada, es aquí donde ubicamos los dos botones que ayudaran a redirigir a la ventana de trabajo de AS-I, o de Ethernet (ver figura 56).

Figura 56. configuración de botones de conexión AS-I y ethernet

```
#Botones                                          Se especifica el nombre, tipo,, tamaño y posición del boton
boton13 = tk.Button(ventana2, text="Ethernet" command=login2, cursor="hand2", bg='#df6c5d', width=12,
font=("Comic Sans Ms", 19, "bold"))
boton13.place(x=603, y=625)

boton = tk.Button(ventana2, text="AS-I",command=login, cursor="hand2", bg='#df6c5d', width=12,
font=("Comic Sans Ms", 19, "bold"))
boton.place(x=377, y=625)

boton1 = tk.Button(ventana2, text="Regresar", command=regresar, cursor="hand2", bg='#df6c5d' , width=14,
font=("Comic Sans Ms", 19, "bold"))
boton1.place(x=1065, y=643)


ventana2.mainloop()
```

Fuente: Kevin Mosquera

En la figura 57 se observa cómo queda representado la ventana de ingreso, en la cual el operario podrá ingresar por teclado su nombre, contraseña y elegir el tipo de comunicación ya sea por AS-I o por Ethernet.

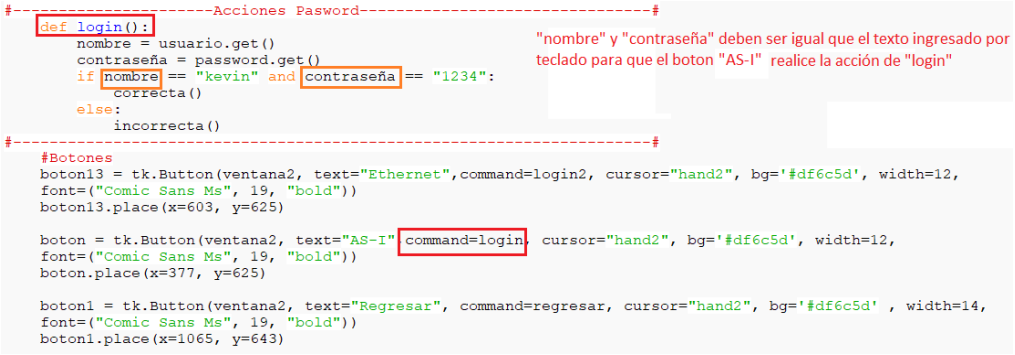
Figura 57. Ventana de Ingreso del programa

Fuente: Kevin Mosquera

Ahora para la parte de acción de cada botón se debe realizar una programación para cada uno, donde al pulsar cualquier botón de tipo de comunicación valide el usuario con la

contraseña, para configurar los parámetros se debe realizar una programación en la cual sea posible validar los datos almacenados por teclado.

Figura 58. Establecimiento de la comunicación AS-I en el programa




```
#--------------------Acciones Pasword-------------------------------#
    def login():
        nombre = usuario.get()
        contraseña = password.get()
        if nombre == "kevin" and contraseña == "1234":
            correcta()
        else:
            incorrecta()
#-------------------------------------------------------------------#
    #Botones
    boton13 = tk.Button(ventana2, text="Ethernet",command=login2, cursor="hand2", bg='#df6c5d', width=12,
    font=("Comic Sans Ms", 19, "bold"))
    boton13.place(x=603, y=625)

    boton = tk.Button(ventana2, text="AS-I" command=login, cursor="hand2", bg='#df6c5d', width=12,
    font=("Comic Sans Ms", 19, "bold"))
    boton.place(x=377, y=625)

    boton1 = tk.Button(ventana2, text="Regresar", command=regresar, cursor="hand2", bg='#df6c5d' , width=14,
    font=("Comic Sans Ms", 19, "bold"))
    boton1.place(x=1065, y=643)
```

Fuente: Kevin Mosquera.

Como se observa en la ver figura 58, para que el operario pueda ingresar a la comunicación por AS-I, debe coincidir la variable almacenada por teclado con la programada, y solo de esa manera se podrá ejecutar la acción correcta, de lo contrario dará un mensaje de Error.

3.10.6 Ventana de trabajo (interfaz gráfica) para comunicación AS-I

En esta ventana de trabajo es necesario tener una visualización de video en tiempo real, además de que tiene que mostrar las características de la pieza analizada como el tipo de color, los detalles y la valoración, y por último debe contener botones que muestren un historial, una manera de recalibrar los valores de testeo, (ver figura 59).

Figura 59. Programación de la ventana de visualización


```
#----------------------Ventana Trabajo----------------------------------#
    def correcta():
        ventana2.withdraw()
        window = tk.Toplevel()
        window.title("Vision-500")
        window.geometry("1366x768+0+0")
        window.resizable(width=False, height=False)
        fondo = tk.PhotoImage(file=ArchImport + "/ImagenInicio.png")
        fondo1 = tk.Label(window, image=fondo).place(x=0, y=0, relwidth=1, relheight=1)

        cap = None
#-----------------------------------------------------------------------#
```

Fuente: Kevin Mosquera

Ahora se debe crear los botones de acción para activar y mostrar la cámara en tiempo real, también un botón para una ventana de historial y de calibración, (ver figura 60).

Figura 60. Configuración de la cámara e historial de datos

```
# Botones----
boton3 = tk.Button(window, text="Regresar", command=atras, cursor="hand2", bg='#df6c5d', width=11,
font =("Comic Sans MS", 19, "bold"))
boton3.place(x=1112, y=647)

boton5 = tk.Button(window, text="IniciarVideo", command=video_stream, cursor="hand2", bg='#eb9b91',
font=("Comic Sans Ms", 17, "bold"))
boton5.place(x=690, y=650)

boton6 = tk.Button(window, text="QuitarVideo", command=quitar, cursor="hand2", bg='#e9988d',
font=("Comic Sans Ms", 17, "bold"))
boton6.place(x=900, y=650)

boton7 = tk.Button(window, text="Calibrar", command=calibrar, cursor="hand2", bg='#e9988d', width=1
font=("Comic Sans Ms", 17, "bold"))
boton7.place(x=85, y=650)

boton8 = tk.Button(window, text="Históricos", command=historicos, cursor="hand2", bg='#e9988d', wid
font=("Comic Sans Ms", 17, "bold"))
boton8.place(x=285, y=650)

boton8 = tk.Button(window, text="Export Cloud", command=exportar, cursor="hand2", bg='#e9988d', wid
font=("Comic Sans Ms", 17, "bold"))
boton8.place(x=1177, y=538)

#Etiqueta
etiq_de_video = tk.Label(window, bg="black")
etiq_de_image = tk.Label(window, bg="black")

window.mainloop()
```

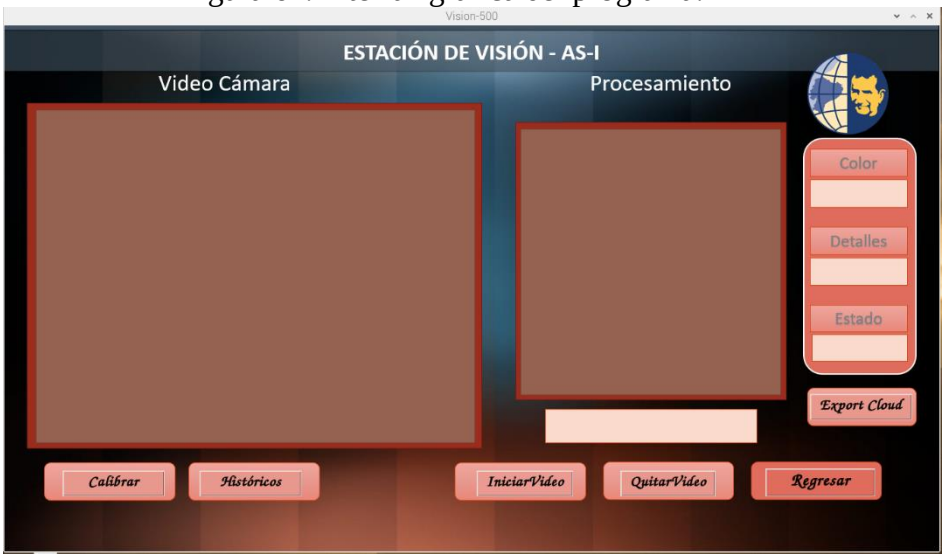
Definir el nombre de cada botón seguido de la acción y la ubicación dentro de la pantalla

Etiquetas para mostrar video e imagen en la pantalla

Fuente: Kevin Mosquera

En la ver figura 61 se muestra la interfaz de cómo queda la ventana incluyendo los botones implementados.

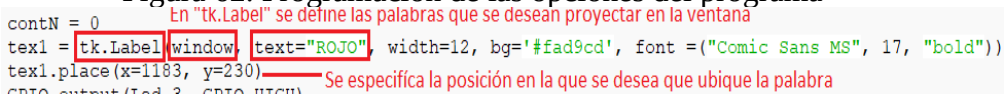
Figura 61. Interfaz gráfica del programa.



Fuente: Kevin Mosquera.

Para incluir las valoraciones dentro de la pantalla es necesario definir nuevas variables en las que se pueda visualizar los estados como son el Color, Detalles, Estado y las notificaciones, (ver figura 62).

Figura 62. Programación de las opciones del programa

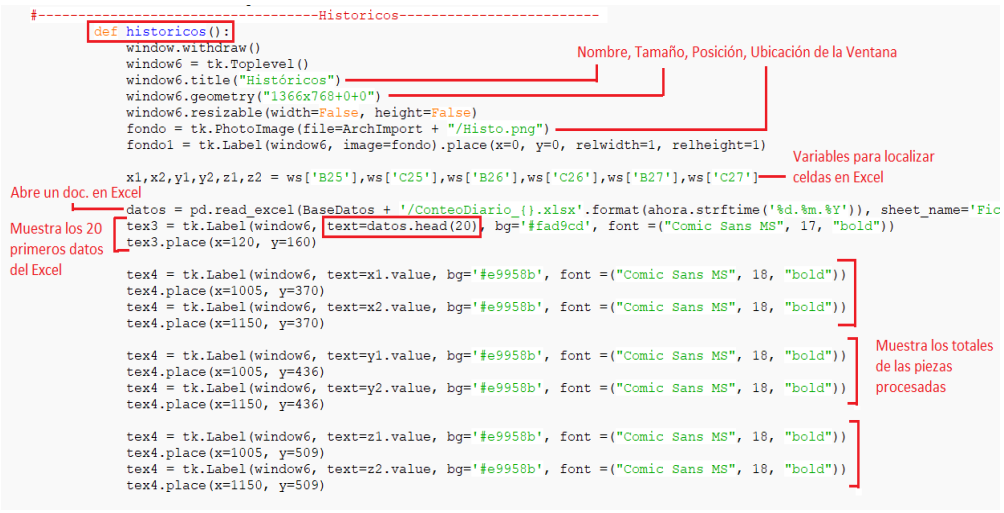


Fuente: Kevin Mosquera

3.10.7 Ventana de históricos (interfaz gráfica)

Al pulsar el botón de “Históricos” se desplegará una nueva ventana la cual desplegará la información previamente almacena en un archivo Excel en el que indicará cuantas piezas se han procesado como también el color, el estado, sus detalles, y la hora en la que fue procesado, en la figura 63 se muestra la programación realizada.

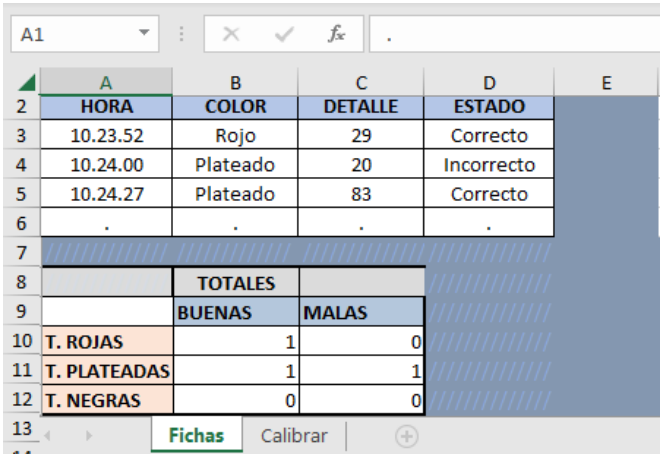
Figura 63. Programación de la interfaz de históricos



Fuente: Kevin Mosquera

En la figura 64 se muestra el archivo en Excel el cual almacena toda la información de las piezas que pasan por la estación de visión.

Figura 64. Almacenamiento de datos en Excel



	A	B	C	D	E
2	HORA	COLOR	DETALLE	ESTADO	
3	10.23.52	Rojo	29	Correcto	
4	10.24.00	Plateado	20	Incorrecto	
5	10.24.27	Plateado	83	Correcto	
6	.	.	.	.	
7					
8		TOTALES			
9		BUENAS	MALAS		
10	T. ROJAS	1	0		
11	T. PLATEADAS	1	1		
12	T. NEGRAS	0	0		
13					

Fuente: Kevin Mosquera

Para la programación se utilizó como base un archivo en Excel predefinido en donde cada vez que se procesa una pieza, esta se irá registrando tanto como su color estado y hora, como se muestra en la figura 65.

Figura 65. Interfaz de históricos

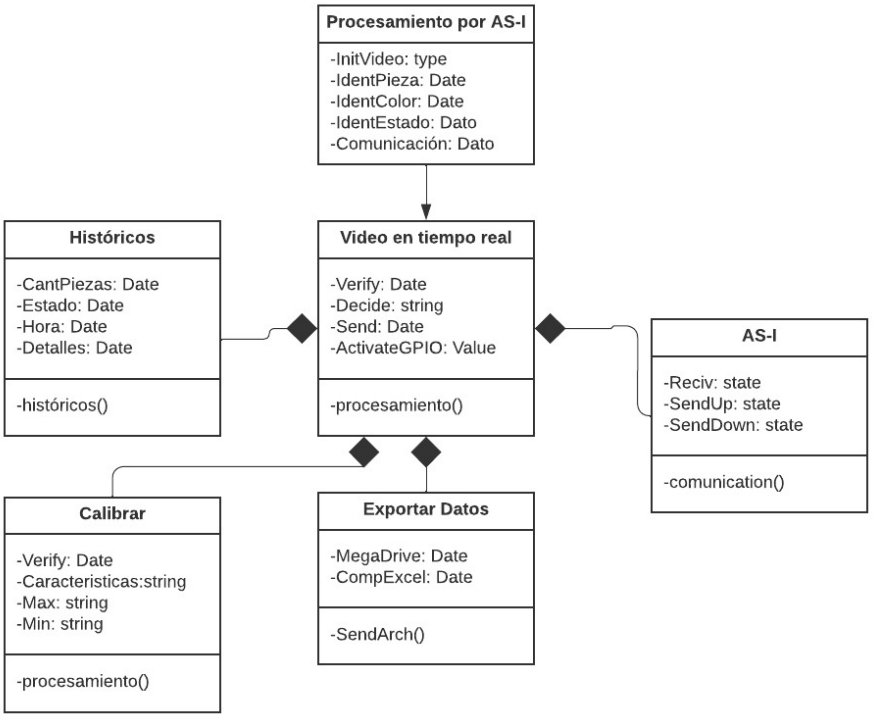
Fuente: Kevin Mosquera

Así se comprueba que la interfaz de los históricos funciona correctamente y que siempre dará la información relevante.

3.10.8 Programación de la ventana de trabajo (comunicación AS-I)

Esta programación sigue siendo parte del programa principal, en donde además de tener una interfaz gráfica se lo pueda manipular dentro del mismo entorno, a continuación, se expresa en un diagrama de clases el funcionamiento de la programación, (ver figura 66).

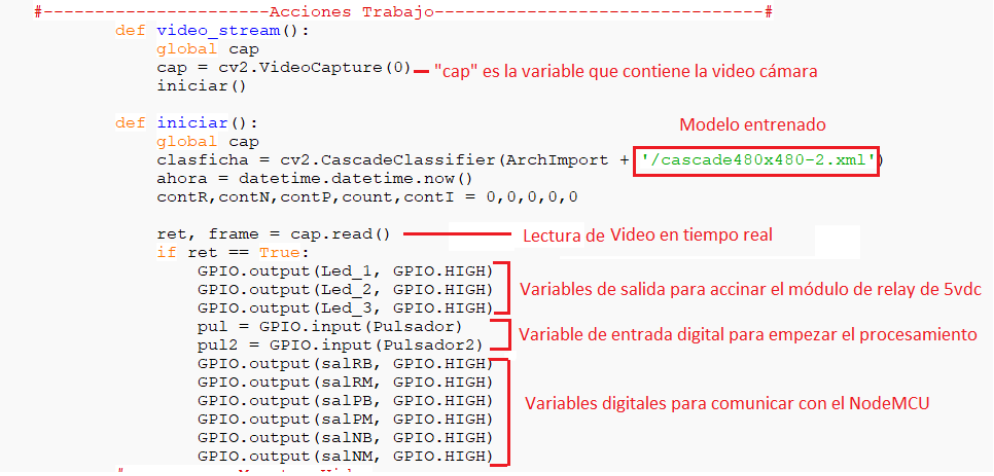
Figura 66. Diagrama de clases del procesamiento AS-I



Fuente: Kevin Mosquera

Para el botón “Iniciar video” se realizó una programación en la cual se pueda mostrar video en tiempo real, y mientras muestra video la Raspberry espera una señal por los pines digitales para que inicie el procesamiento por visión artificial, (ver figura 67).

Figura 67. Programación del video del programa

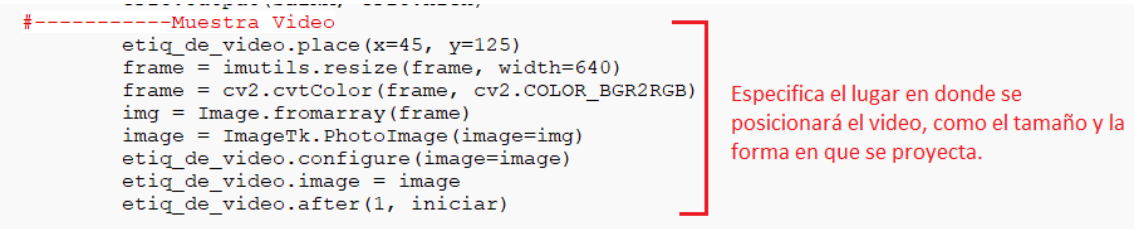



```
#----------------------Acciones Trabajo--------------------------------#
        def video_stream():
            global cap
            cap = cv2.VideoCapture(0)
            iniciar()

        def iniciar():
            global cap
            clasficha = cv2.CascadeClassifier(ArchImport + '/cascade480x480-2.xml')
            ahora = datetime.datetime.now()
            contR,contN,contP,count,contI = 0,0,0,0,0

            ret, frame = cap.read()
            if ret == True:
                GPIO.output(Led_1, GPIO.HIGH)
                GPIO.output(Led_2, GPIO.HIGH)
                GPIO.output(Led_3, GPIO.HIGH)
                pul = GPIO.input(Pulsador)
                pul2 = GPIO.input(Pulsador2)
                GPIO.output(salRB, GPIO.HIGH)
                GPIO.output(salRM, GPIO.HIGH)
                GPIO.output(salPB, GPIO.HIGH)
                GPIO.output(salPM, GPIO.HIGH)
                GPIO.output(salNB, GPIO.HIGH)
                GPIO.output(salNM, GPIO.HIGH)
```

Fuente: Kevin Mosquera

Para mostrar el video en tiempo real es necesario realizar una programación de tal manera que se pueda proyectar dentro de la ventana, por ende, es necesario usar las etiquetas que se habían creado con anterioridad, (ver figura 68).

Figura 68. Mostrar video en el programa

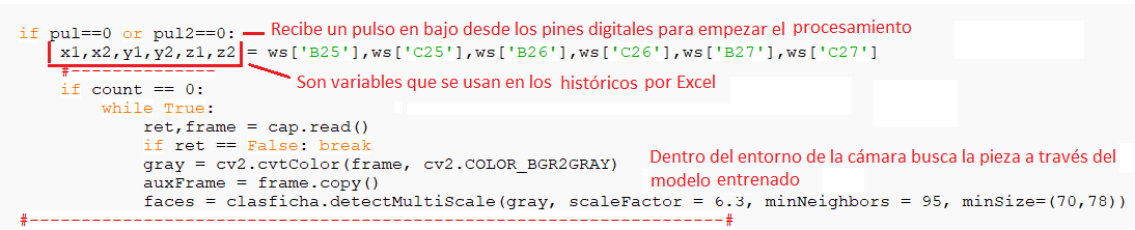



```
#-----------Muestra Video
        etiq_de_video.place(x=45, y=125)
        frame = imutils.resize(frame, width=640)
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(frame)
        image = ImageTk.PhotoImage(image=img)
        etiq_de_video.configure(image=image)
        etiq_de_video.image = image
        etiq_de_video.after(1, iniciar)
```

Fuente: Kevin Mosquera

Cuando se recibe un pulso ya sea por AS-I o por el microcontrolador NodeMCU, la Raspberry comenzará a realizar el procesamiento.

Figura 69. Reconocimiento de pulsos en el programa




```
if pul==0 or pul2==0:
    x1,x2,y1,y2,z1,z2 = ws['B25'],ws['C25'],ws['B26'],ws['C26'],ws['B27'],ws['C27']
    #-------------
    if count == 0:
        while True:
            ret,frame = cap.read()
            if ret == False: break
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            auxFrame = frame.copy()
            faces = clasficha.detectMultiScale(gray, scaleFactor = 6.3, minNeighbors = 95, minSize=(70,78))
#-------------------------------------------------------------#
```

Fuente: Kevin Mosquera

Como se observa en la figura 69 si en “pul” que corresponde a la entrada por AS-I recibe un pulso en bajo el sistema comenzará a procesar la imagen iniciando por buscar la pieza en el entorno que indique el campo de visión de la cámara.

Luego de encontrar la pieza en el entorno, el programa toma una fotografía de la misma, y continuará reconociendo el color como se había explicado en el capítulo 3.4.2, posteriormente identificará los caracteres de esta pieza explicado en el capítulo 3.4.4, y tomará la decisión del estado en la que se encuentre, dando un pulso en alto o bajo ya sea que la pieza esté en buen o mal estado correspondientemente por sus pines digitales que a su vez están conectados a la comunicación AS-I, la cual le indica a la siguiente estación de Clasificación que debe ubicar la pieza ya sea por su color o que la mande a desechar.

3.11 Ventana de trabajo (interfaz gráfica) para comunicación Ethernet

En su mayoría la programación es idéntica que por la comunicación AS-I, solo que para accionar la verificación, la estación recibe datos por Ethernet de una estación anterior que sería la de maquinado y la Raspberry envía otro dato por Ethernet del estado de la pieza analizada a la siguiente estación que es la de Clasificación, (ver figura 70).

Figura 70. Estación de trabajo por Ethernet

Fuente: Kevin Mosquera

3.11.1 Ventana para el ingreso de IP's (maquinado y clasificación)

Para que la estación de visión pueda comunicarse por Ethernet es necesario ingresar una IP por cada estación, para eso se ha creado una interfaz que sea capaz de aceptar por teclado la numeración de la IP (ver figura 71).

Figura 71. Interfaz gráfica de las direcciones IP

Fuente: Kevin Mosquera

Para llegar a este punto es necesario realizar un algoritmo en la que se pueda ingresar la IP de las estaciones (ver figura 72).

Figura 72. Algoritmo para el ingreso de la IP en el programa

```
def Ethernet():
    ventana2.withdraw()
    window7 = tk.Toplevel()
    window7.title("IpConf")                          ← Configuración de la interfaz gráfica
    window7.geometry("1366x768+0+0")
    window7.resizable(width=False, height=False)
    fondo = tk.PhotoImage(file=ArchImport + "/ipconfig.png")
    fondo1 = tk.Label(window7, image=fondo).place(x=0, y=0, relwidth=1, relheight=1)

    ipMaq = tk.StringVar()
    ipCla = tk.StringVar()                           ← Variables para obtener las dos diferentes IP's

    ipMaq1 = tk.Entry(window7, textvar=ipMaq, width=10, bg='#ed9f96', font =("Comic Sans MS", 20, "bold"))
    ipMaq1.place(x=305, y=392)
    tex1 = tk.Label(window7, text="192.168.", bg='#ed9f96', font =("Comic Sans MS", 18, "bold"))
    tex1.place(x=295, y=392)

    ipCla1 = tk.Entry(window7, textvar=ipCla, width=10, bg='#ed9f96', font =("Comic Sans MS", 20, "bold"))
    ipCla1.place(x=768, y=392)
    tex1 = tk.Label(window7, text="192.168.", bg='#ed9f96', font =("Comic Sans MS", 18, "bold"))
    tex1.place(x=758, y=392)
```

Fuente: Kevin Mosquera

Luego que el programa memoriza las dos diferentes IP, las ingresa en la librería de modbus.client, la cual lo interpreta y se conecta a ellas recibiendo la información de la primera estación, la cual indicará a la estación de visión si la pieza ya está maquinada (ver figura 73).

Figura 73. Librería modbus para la IP

```
def correcta2():
    ipMaq2 = ipMaq.get()
    ipCla2 = ipCla.get()
    modbusclient=easymodbus.modbusClient.ModbusClient('192.168.{}'.format(ipMaq2),502)
    modbusclient.connect()
    print("Se conecto a la 1ra IP")
```

Uso de la libreria TCP/IP para el ingreso de IP de las estaciones

Fuente: Kevin Mosquera

Luego de que la estación de visión realice la interpretación de la pieza, esta enviara un estado por Ethernet a la estación de Clasificación, para que clasifique por color, o que deseche la pieza.

3.12 Programación de estación de transporte (conveyor)

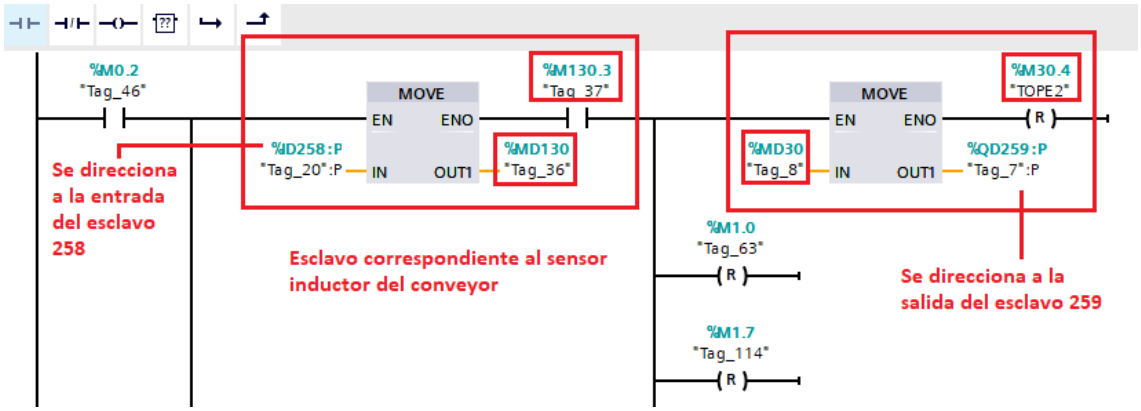
Para programar esta estación se debe considerar todos los módulos que contiene su PLC, como es el módulo de comunicación por AS-I, el cual tiene conectado en sus esclavos los diferentes sensores como son los de infrarrojo, o los inductivos, y también tiene conectado actuadores como los pistones neumáticos que detienen al pale, pero un punto importante es que dentro de sus esclavos se encuentran las entradas y salidas que se conectan a las diferentes estaciones que en este proyecto fueron la de maquinado y clasificación.

3.12.1 Direccionamiento de comunicación para esclavos AS-I

Para que la estación de conveyor pueda comunicarse con las demás estaciones es necesario que estas estén conectadas a los esclavos AS-I del conveyor de tal manera que cada vez que esta estación desee tener una comunicación con otra debe enviar pulso en alto o bajo por medio de los esclavos a las estaciones, por ende, se debe programar en su software de TIA-PORTAR, unos nuevos bloques en los que indique que esclavo se esta usando y para que estación.

A continuación, se muestra la programación realizada para una comunicación con la estación de maquinado (ver figura 74).

Figura 74. Comunicación AS-I con los esclavos



Fuente: Kevin Mosquera

Como se ve en la imagen para poder tener un direccionamiento con cada esclavo se debe crear un bloque MOVE y adicional un contacto abierto o cerrado según la lógica que se desea programar. Para el primer recuadro se esta usando para un sensor inductor que esta conectado al esclavo ID258, y tiene como variable una memoria en MD130 en el bit 3. En el segundo recuadro se usa el bloque MOVE como salida conectada al esclavo QD259, y tiene como variable una memoria en MD30 en el bit 4, que envía un pulso en alto a la estación de maquinado.

CAPÍTULO 4

4 PRUEBAS DE FUNCIONAMIENTO

En este capítulo se muestran las pruebas de funcionamiento realizadas dentro un proceso automatizado, en el cual se posicionará una pieza de cualquier color en un pallet, y mediante un pulsador de marcha de la estación de Conveyor da paso al accionamiento de una banda transportadora hasta la estación de maquinado, para que emule que maquine (posicionamiento central y perforación), luego el conveyor transporta al pallet hasta la estación de visión para identificar las piezas en buen o mal estado, de esta forma concluye el proceso transportando el pallet a la estación de clasificación.

4.1 Prueba de la detección de Pieza dentro de un entorno

Se evalúa el resultado del modelo de aprendizaje creado con clasificadores en cascada Haar, haciendo comparaciones con diferente cantidad de imágenes positivas y negativas para saber a cuál se ajusta mejor el modelado creado, en la siguiente tabla se muestra las cantidades de imágenes positivas (P) y negativas (N) que se usaron para llegar a obtener el resultado ideal.

Tabla 2. Pruebas de imágenes positivas y negativas para el modelo

		Prueba A	Prueba B	Prueba C
#	**Pieza**	**800P/800N**	**900P/500N**	**1000P/800N**
1	Rojo	**Localizada**	**Localizada**	**Localizada**
2	Rojo	No localizada	**Localizada**	**Localizada**
3	Rojo	**Localizada**	**Localizada**	No localizada
4	Negro	No localizada	**Localizada**	**Localizada**
5	Negro	**Localizada**	**Localizada**	**Localizada**
6	Negro	No localizada	**Localizada**	No localizada
7	Plateado	No localizada	**Localizada**	No localizada
8	Plateado	No localizada	**Localizada**	**Localizada**
9	Plateado	**Localizada**	No localizada	No localizada
TOTAL		4 Localizadas	8 localizadas	5 localizadas

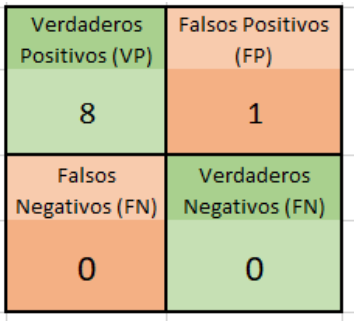
Fuente: Kevin Mosquera

En la tabla 2 se analiza la toma de decisión del modelo de clasificación respecto la realidad y a través de inferencia causal por el método de control aleatorio (sobre el pallet se colocó al azar la pieza de buen o mal estado independientemente del color), se observó que con muy pocas imágenes positivas (prueba A) el modelo tiene baja precisión, y con

demasiadas imágenes (sobreajuste) positivas y negativas (prueba C) el modelo tiende en reducir su probabilidad en un 44%, por el contrario, el menor error presenta con 900 imágenes positivas y 500 negativas (prueba B) que en las nueve piezas testeadas de diferente color se encontró un falso positivo, teniendo una precisión del 88% en este modelo, el cual fue usado para este proyecto.

En la siguiente matriz de confusión (ver figura 75) se representa la evaluación anterior, en la cual indica dos posibles valores de predicción y dos posibles valores reales.

Figura 75. Matriz de confusión



Fuente: Kevin Mosquera

Ahora se verifica la exactitud aplicando la ecuación 1 dando un resultado de 0.88, la cual se interpreta como el 88% de efectividad del ML.

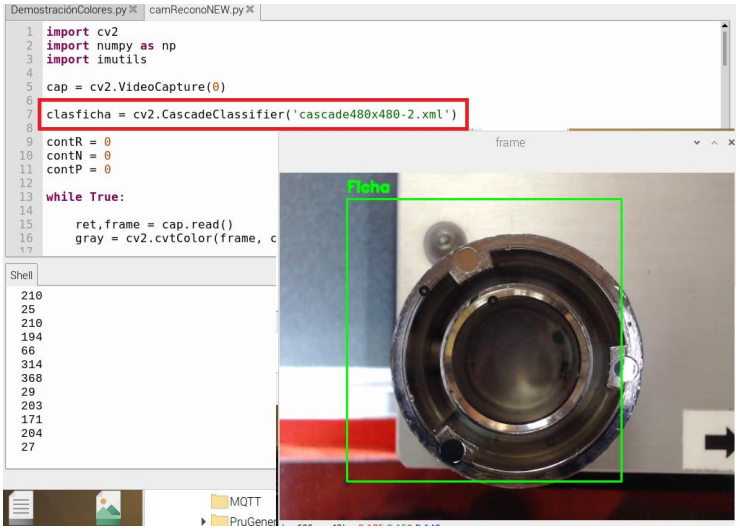
Ecuación 1. Exactitud de ML

$$Exactitud = \frac{VP + VN}{VP + FP + FN + VN} = \frac{8 + 0}{8 + 1 + 0 + 0} = 0.88$$

Fuente: Kevin Mosquera

A continuación, se muestra cómo se logra localizar las piezas sin importar su color (ver figura 76).

Figura 76. Localización de pieza dentro de un entorno



Fuente: Kevin Mosquera

4.2 Prueba de la detección de color de la Pieza

Para esta prueba se utilizaron tres colores de piezas que se dispone en el laboratorio, las cuales son de color rojo, negro y plateado, en donde el sistema debe reconocer el color de la pieza que se tiene en el campo de visión de la cámara.

En la tabla 3 se muestra las dos pruebas realizadas con piezas de diferente color, donde se observa que la primera prueba tuvo más errores de precisión que en la segunda.

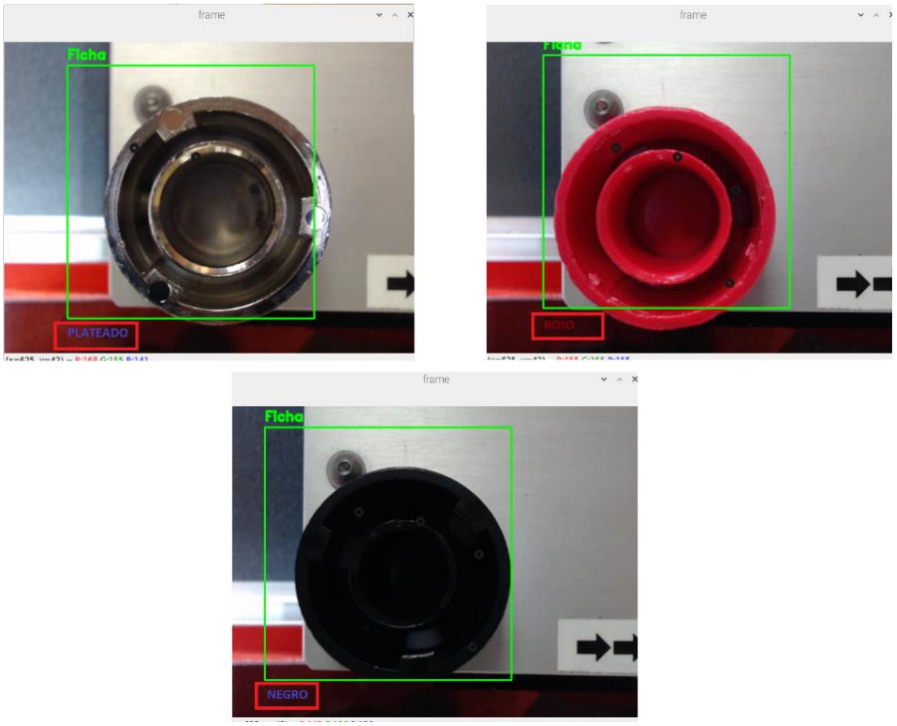
Tabla 3. Prueba de identificación de color de piezas

#	Pieza	Prueba A	Prueba B
1	Rojo	**Identificada**	**Identificada**
2	Rojo	**Identificada**	**Identificada**
3	Rojo	**Identificada**	**Identificada**
4	Negro	No Identificada	**Identificada**
5	Negro	**Identificada**	**Identificada**
6	Negro	No localizada	No localizada
7	Plateado	**Identificada**	**Identificada**
8	Plateado	No localizada	**Identificada**
9	Plateado	**Identificada**	No localizada
TOTAL		6 Identificadas	7 Identificadas

Fuente: Kevin Mosquera

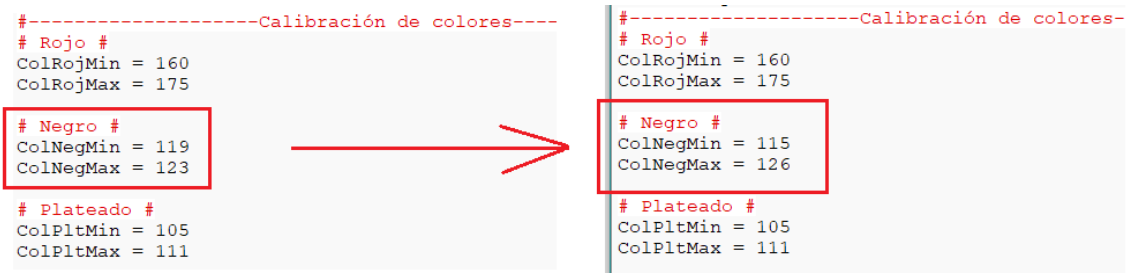
Como se observa en los resultados de detección de color de la tabla anterior, la detección de color es algo, debido a que en la primera prueba se tuvo tres errores. Sin embargo, un operario tiene la facilidad de ajustar la tolerancia para mejorar el acercamiento al color (ver figura 77), tal como indica en la segunda columna de la tabla 3, y de esa manera el proceso ya operaria de una forma más acertada. El ajuste se lo hace a las variables de colores que se muestran en la figura 78.

Figura 77. Prueba de detección de colores



Fuente: Kevin Mosquera

Figura 78. Resultados de la calibración de los colores detectados



Fuente: Kevin Mosquera

4.3 Prueba de estado de la Pieza

Para verificar el estado de la pieza en buen o mal estado, se realizó una prueba de los contornos de la pieza para obtener un mínimo y un máximo de número de contornos en los cuales se considera que se encuentra en buen estado.

En la siguiente imagen se observa cómo se realiza el testeo, los cuales serán ingresados manualmente a la programación, esta acción al igual que se explicó en la sección de detección de color se lo realiza una sola vez y luego el proceso se vuelve automático

Figura 79. Valores de la calibración de las piezas

```
#-------------------Valores de Calibración----------
#ROJO
V_min_Rj = 15
V_max_Rj = 47
#PLATEADO
V_min_Pt = 60
V_max_Pt = 122
#NEGRO
V_min_Ng = 30
V_max_Ng = 70
```

Valores tras el testeo por cada color

Fuente: Kevin Mosquera

Cabe mencionar que para realizar este testeo se realiza con una pieza de cada color como primera y única vez, esto tiene por objetivo indicar el número mayor de contornos que se puede conseguir al rotar en diferentes posiciones sobre su eje a la pieza, para de esta manera realizar la comparación, como se indica en la figura 80.

Figura 80. Obtención de valores de los bordes

Fuente: Kevin Mosquera

En la tabla 4 se muestran los resultados de una prueba de control sintético, esto quiere decir que para las piezas en buen estado; de color rojo el sistema identificó en un 100%, y para los colores negro y plateado en un 66.6% respectivamente. Para los resultados de las piezas en mal estado; el sistema identificó en un 66.6% para el color rojo y para los colores negro y plateado un 100%.

Tabla 4. Resultados de pruebas de contornos

#	Pieza	Buen Estado	Mal Estado
1	Rojo	**Correcto**	**Correcto**
2	Rojo	**Correcto**	Incorrecto
3	Rojo	**Correcto**	**Correcto**
4	Negro	**Correcto**	**Correcto**
5	Negro	**Correcto**	**Correcto**
6	Negro	Incorrecto	**Correcto**
7	Plateado	**Correcto**	**Correcto**
8	Plateado	Incorrecto	**Correcto**
9	Plateado	**Correcto**	**Correcto**

Fuente: Kevin Mosquera

4.4 Prueba de envío de datos desde la planta a la Nube

Para verificar esta prueba de envío de datos a la nube se hizo uso de un broker gratuito llamado “bróker.emqx.io”, el cual se le ha configurado para que reciba o envíe datos como la cantidad de piezas en buen o mal estado dependiendo del color. Con esto se logra la supervisión y control del proceso mediante ecosistema IOT, en base a protocolos MQTT

Para leer estos datos se utilizó una dashboard en Android llamada “MQTT-Dash”, la cual está especializada para interpretar la información enviada al bróker, (ver figura 81).

Figura 81. Interfaz dashboard

Fuente: Kevin Mosquera

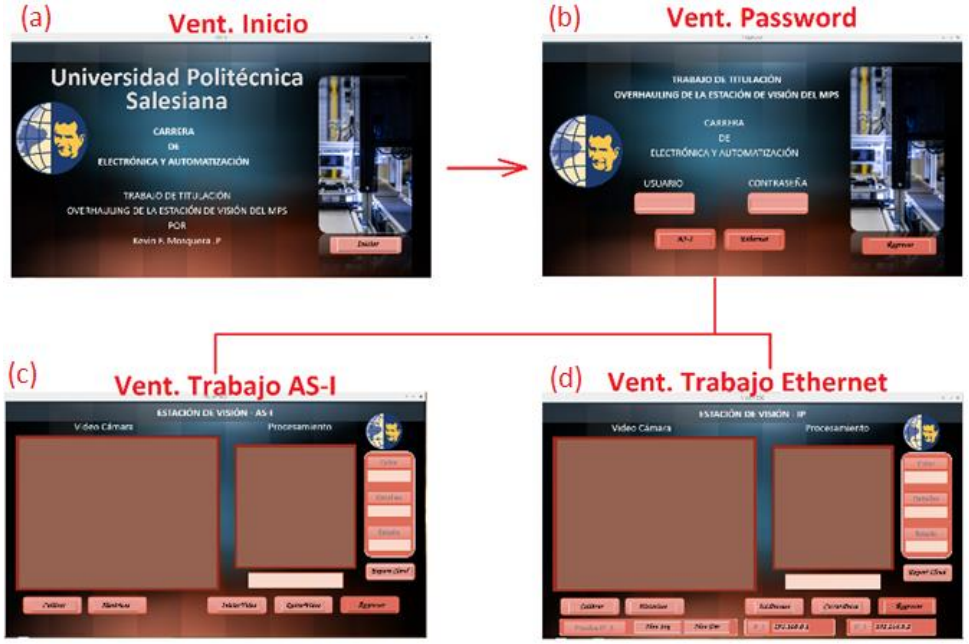
Luego de previa configuración de los "Topic" se puede observar que la información recibida en la dashboard es la misma que los totales de los históricos, esto quiere decir que la comunicación en este protocolo funciona correctamente.

4.5 Puesta en marcha de la estación de visión

En este apartado se realiza una prueba completa de la estación de visión en conjunto con las demás estaciones que conforman el MPS-500 (conveyor, maquinado, clasificación).

Como primer paso se inicia el programa de la estación de visión, dando clic en el botón de inicio de la ventana de bienvenida (figura 82-a), para luego autenticarse (figura 82-b), seguidamente el operador tiene la opción de elegir el tipo de protocolo de comunicación (AS-I o Ethernet), para el caso de la figura 82-c se eligió AS-I y posteriormente con el botón IniciarVideo.

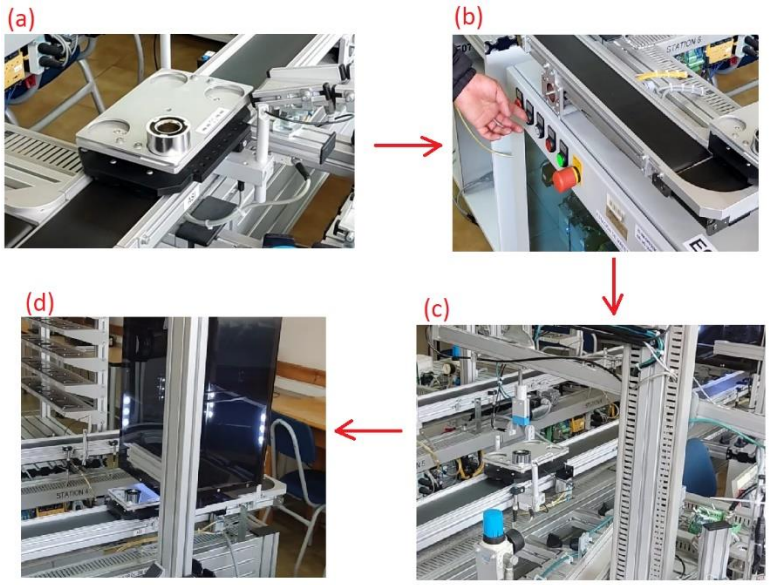
Figura 82. Pasos de ingreso a la Vent. Trabajo



Fuente: Kevin Mosquera

Como segundo paso se posiciona una pieza sobre el pallet (ver figura 83-a), y a través de un pulsador manual se dé marcha al conveyor (ver figura 83-b), llevando el pallet a la estación del maquinado ver figura 83(c) para su maquinación, una vez culminada este proceso la pieza se devuelve al pallet, arrancando nuevamente el conveyor para transportarla hacia la estación de visión ver figura 83(d), una vez que se ha llegado a esta estación el conveyor envía un pulso AS-I a la estación de visión.

Figura 83. Prueba del proceso



Fuente: Kevin Mosquera

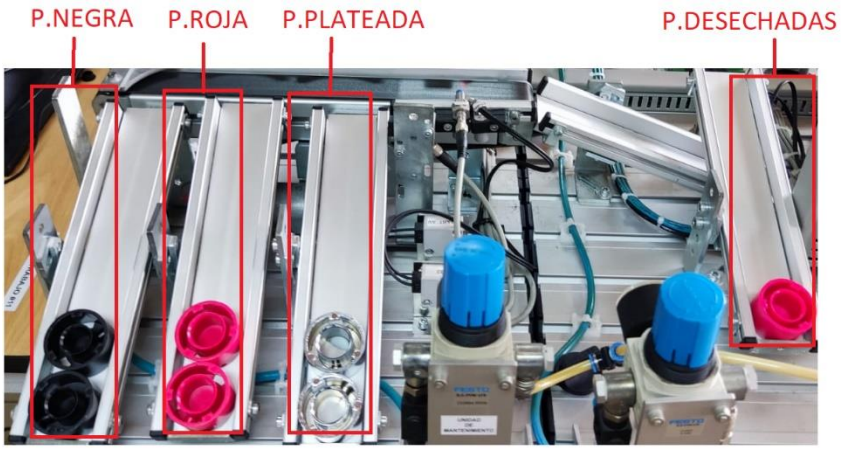
Una vez que el pallet se encuentra en el área de trabajo de la estación de visón, se acciona un proceso de verificación de estado de la pieza, y en la interfaz de trabajo nos muestra el resultado de la verificación que en este caso es el color, los contornos (detalles) ver figura 84.

Figura 84. Prueba de calidad

Fuente: Kevin Mosquera

Una vez culminado el proceso anterior la estación de visión envía al conveyor una respuesta de operador lógico (RLO por sus siglas en inglés) binaria AS-I en alto, indicando que la pieza se encuentra en buen estado, caso contrario en mal estado, es así como el conveyor entiende la orden, dando paso al traslado del pallet a la siguiente estación (clasificación), una vez que el pallet ha llegado a esta ultima estación, el conveyor transmite a la estación de clasificación el ultimo estado RLO de esta manera se separa por columnas las piezas de color en buen estado y aquellas que se encuentran en mal estado son desechadas ver figura 85.

Figura 85. Clasificación de piezas buenas



Fuente: Kevin Mosquera

Es así que de esta manera se concluye la prueba completa por comunicación AS-I, y para el caso de comunicación Ethernet el proceso es similar, con la diferencia que el operador selecciona el tipo de comunicación que requiera el proceso ver figura 82 (d).

CAPÍTULO 5

5 CRONOGRAMA Y PRESUPUESTO

5.1 Cronograma de actividades

En la tabla 5 se observa el cronograma de actividades.

Tabla 5. Cronograma de actividades

#	ACTIVIDADES	1er MES				2do MES				3er MES				4to MES			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	Indagación sobre métodos de restauración de sistemas de producción.																
2	Caracterización del MPS																
3	Identificación de módulos de sistemas embebidos compatibles para la comunicación.																
4	Identificación de cámaras de video compatibles con sistema embebido.																
5	Selección del algoritmo de control para el desarrollo de visión artificial.																
6	Diseño de una estructura para la incorporación hardware.																
7	Establecer una comunicación con el resto de las estaciones mediante protocolos clásicos.																
8	Definir protocolos de comunicación entre la estación de visión y el resto de la planta.																
9	Definir protocolos de comunicación entre la estación de visión y la cloud.																
10	Definir servicios en línea o plataformas web útiles para el desarrollo de una dashboard.																
11	Definir una arquitectura de comunicación unidireccional o bidireccional según requerimientos.																
12	Identificar puertos de salida y entrada de datos inalámbrica.																
13	Definir una infraestructura con protocolos IPV4 e IPV6, y frecuencia de wifi 4G o 5G, según asesoría de servicio técnico de la UPS.																
14	Pruebas de comunicación previo la definición de protocolos, arquitectura e infraestructura.																
15	Desarrollo de una guía de usuario en el cual indicará el procedimiento a seguir para el uso de la estación.																
16	Desarrollo del documento escrito.																
17	Resultados y avances al tutor																

Fuente: Kevin Mosquera

5.2 Presupuesto

En el presupuesto se consideró los materiales que se utilizaron para el proceso de restauración e implementación de la estación, los cuales se detallan en la tabla 6.

Tabla 6. Presupuesto para la implementación

	DETALLE	CANTIDAD	PRECIO
1	Raspberry PI4 8gb	1	$ 250
2	Switch 8 puertos	1	$ 20
3	Módulo Relay 5vdc	1	$ 6
4	NodeMCU 8266	1	$ 10
5	Breaker Schneider 5Amp	1	$ 6
6	Fuente 5vdc	1	$ 9
7	Borneras monopolares	6	$ 6
8	Acrílico 3mm	1	$ 40
9	Perfil Aluminio	1	$ 15
10	Insumos varios	1	$ 50
TOTAL			$ 412

Fuente: Kevin Mosquera

CAPÍTULO 6

6 CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

• La estación de visión permite que un operario a través de una interfaz seleccione el protocolo de comunicación (AS-I o Ethernet) dentro del proceso manufacturero. Además, el sistema inteligente de la estación de visión está constituido por inteligencia artificial de aprendizaje automático para la identificación de piezas dentro del espacio de trabajo, visión por computadora para la identificación del color y sus contornos, y una arquitectura de comunicación hacia la nube, con el objeto de que un operador a través de autentificación pueda observar y descargar los resultados del estado de las piezas en cualquier lugar y cualquier momento, de esta manera se realizó el overhauling de escalamiento de la estación de visión.

• A través de las ventajas que posee la industria flexible, se encuentra el poder comunicarse entre diferentes dispositivos de distinto propietario, dando lugar a la comunicación entre una Raspberry de la estación de visión con los PLC's de las diferentes estaciones del proceso manufacturero del MPS-500, los cuales manejan protocolos de comunicación Ethernet o AS-I. Dentro de la implementación se consiguió un 100% de efectividad en la comunicación tras nueve pruebas de proceso de producción, considerando que cada prueba requiere un tiempo de un minuto con cuarenta segundos aproximadamente.

• Tras nueve pruebas realizadas del modelo de aprendizaje creado con clasificadores en cascada Haar (ver tabla 2-b) utilizado para la detección de piezas, dio como resultado una exactitud del 88.8% visto en la ecuación 1, en donde se consideró como verdaderos positivos a las piezas detectadas correctamente y falsos negativos a la incorrecta detección de las mismas representando el 11.1% de error.

• En el proceso del sistema inteligente (detección de color), tras nueve pruebas realizadas (ver tabla 3), se encontró un ligero incremento de la exactitud de sus resultados de 11.1% cuando se ajusta el rango de HSV de los colores, pasando de 66.6% a 77.7%. Sin embargo, durante nueve pruebas de proceso de producción se tuvo una exactitud de 88.8%.

• En el proceso de detección de contornos que forma parte del sistema inteligente, se realizaron nueve pruebas con tres piezas de colores distintos (ver tabla 4), en donde se ajustó el rango de la cantidad de contornos para así pasar de un 77.7% a un 88.8% de exactitud dando un incremento del 11.1% de efectividad en el sistema al momento de verificar el estado de la pieza.

6.2 Recomendaciones

• Debido al uso de la red Ethernet para este proyecto, se recomienda hacer uso de otras estaciones como el de almacenamiento de piezas para trabajar con todos los módulos existentes en el Sistema de Producción Modular.

• Se recomienda realizar las calibraciones de las piezas antes de hacer uso de la estación para generar resultados que sean más eficientes y así mismo, al momento en que la ficha llega a la estación se recomienda no moverla de su base ya que afecta a los resultados que genere el algoritmo.

• Tener en cuenta que la instalación de todas las librerías para visión artificial en Raspberry PI OS tendrá una duración aproximada de dos horas por lo que se recomienda que el microrordenador tenga suficiente ventilación y siempre esté conectada a la red con una conexión de internet estable.

REFERENCIAS

Aprende Machine Learning. (4 de Noviembre de 2017). Obtenido de https://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/

Auquilla, B., & Cárdenas, H. (2012). *Diseño e implementación del manual técnico de prácticas para manejo del modulo "mps-500 visión" con un sistema de supervisión control y adquisición de datosscada, utilizando la inspección por cámara para gestionar el control de calidad vía protocolo de co.* Quito: UPS.

Enriquez Aguilera; Francisco Javier. (08 de 12 de 2018). *uacj*. Obtenido de http://cathi.uacj.mx/handle/20.500.11961/6326;jsessionid=D9FBC9F4CE1155722EC7101CCBCA8872

Festo. (2010). *Festo-Didactic*. Obtenido de https://www.festo-didactic.com/es-es/productos/automatizacion-de-fabricas-e-industria-4.0/modulos-de-la-fabrica-para-la-ensenanza/estaciones-mps

Festo. (2022). *Festo didactic*. Obtenido de Festo didactic: https://www.festo-didactic.com/es-es/productos/mps-sistema-de-produccion-modular/mps-el-sistema-de-produccion-modular-del-modulo-a-la-fabrica-didactica.htm?fbid=ZXMuZXMuNTQ3LjE0LjE4LjU4NS43NjMx

Grupo de Investigación en Electrónica Control Y Automatización. (2017). Obtenido de https://www.ups.edu.ec/gieca

Hernández, L. d. (Octubre de 2018). *programarfacil*. Obtenido de https://programarfacil.com/blog/vision-artificial/detector-de-bordes-canny-opencv/

Invatati afaceri. (07 de 12 de 2022). Obtenido de Invatati afaceri: https://invatatiafaceri.ro/es/diccionario-financiero/gestion-de-calidad-definicion-mas-ejemplo/

ISO. (06 de 1986). Obtenido de https://www.iso.org/standard/15570.html

javatpoint. (Diciembre de 2022). Obtenido de https://www.javatpoint.com/supervised-machine-learning

Jones, P. V. (Mayo de 2004). Robust Real-Time Face Detection. *springer*.

KeepCoding. (25 de Noviembre de 2022). *KeepCoding*. Obtenido de https://keepcoding.io/blog/algoritmo-de-vecinos-mas-proximos/

Lledó, G. (2022). *Lledo Energía.* Obtenido de https://lledoenergia.es/colorimetria-iii-espacios-de-color-hsl-hsv-y-rgb/

Morales, G. (Febrero de 2019). Diseño e implementación de un módulo didáctico de visión artificial para la selección de objetos según colores y características

morfológicas implementando un brazo robótico como complemento del mps de la universidad de las fuerzas armadas.

Pérez, V. (20 de Septiembre de 2010). *La Guía.* Obtenido de https://matematica.laguia2000.com/general/funcion-gaussiana

Prado, S. O. (11 de 10 de 2022). *BlogNews*. Obtenido de BlogNews: https://cdes.es/blog/que-es-la-calidad-definicion-y-ejemplos/

Repositorio Institucional UPS. (Noviembre de 2012). Obtenido de https://dspace.ups.edu.ec/handle/123456789/4139

Rodríguez, P. (Diciembre de 2022). *medium.* Obtenido de https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2

Solano, G. (2019). *Omes.* Obtenido de https://omes-va.com/deteccion-de-colores/

ANEXOS

Anexo 1

Guía de laboratorio para la utilización de la estación de visión (Aprendizaje)

	REVISIÓN 1/1	*Página 1 de 8*
UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
LABORATORIO	SISTEMA DE PRODUCCIÓN MODULAR (MPS)	
CARRERA	INGENIERÍA ELECTRÓNICA	
SEDE	QUITO-CAMPUS SUR	

1. **DATOS INFORMATIVOS**
   a. **PRÁCTICA No:** 1
   b. **ESTACIÓN:** VISIÓN 500
   c. **TIEMPO ESTIMADO:** 2 horas
   d. **NOMBRE DEL ESTUDIANTE:** _____

2. **DATOS DE LA PRÁCTICA**
   a. **TEMA:**
   APRENDIZAJE PARA LA DETECCIÓN DE UNA PIEZA DENTRO DE UN ENTORNO CONTROLADO

   b. **OBJETIVO GENERAL**
   - Realizar el aprendizaje de una pieza para encontrarla dentro de un entorno controlado por medio de visión artificial.

   c. **OBJETIVOS ESPECÍFICOS**
   - Tomar varias fotografías de la pieza que se desea identificar para ingresar al programa de aprendizaje.
   - Instalar el programa de aprendizaje llamado "Cascade-Trainer-GUI" para obtener un archivo entrenado con extensión xml.
   - Probar el archivo entrenado dentro de un script de Python para verificar el aprendizaje.

3. **MARCO TEÓRICO**
   a. **Aprendizaje automático combinado con la visión artificial**

Las técnicas de aprendizaje automático se pueden aplicar a una amplia gama de tareas de clasificación o procesamiento de imágenes de visión artificial. Y, gracias a la escala a la que se adoptan estas aplicaciones, se está volviendo rentable agregar capacidades de aprendizaje automático y visión artificial a una variedad cada vez mayor de productos de IoT [1].

En un entorno industrial, pueden proporcionar inspección de control de calidad de muchos artículos idénticos en una cinta transportadora sin error ni fatiga. De hecho, la visión artificial puede generar ahorros de costos y beneficios de control de calidad en casi cualquier proceso de producción en masa que requiera monitoreo visual [1].

   b. **Cascade-Trainer-GUI**

Elaborado por: Kevin Mosquera	**Revisado por:**	**Aprobado por:**
Fecha de Elaboración	**Fecha de Revisión**	**Número de Resolución Consejo de Carrera:**

Cascade Trainer GUI es un programa que se puede utilizar para entrenar, probar y mejorar los modelos de clasificación en cascada. Utiliza una interfaz gráfica para establecer los parámetros y facilitar el uso de las herramientas de OpenCV para entrenar y probar clasificadores [2].

Actualmente, la GUI de Cascade Trainer se puede usar en Windows (7 o superior). El procedimiento de instalación es bastante sencillo y solo implica presionar un par de botones "Siguiente" [2].

c. Preparando los datos de entrenamiento

Para realizar el entrenamiento será necesario contar con gran cantidad (cientos o miles) de imágenes en donde esté presente el objeto que deseemos detectar (conjunto de muestras positivas) y otro conjunto de imágenes donde NO esté presente dicho objeto (conjunto de muestras negativas). Estas imágenes no deben ser muy grandes, ya que esto podría provocar lentitud en la detección [3].

Tomar en cuenta que, para obtener mejores resultados de detección, las imágenes de entrenamiento deben tener la mayor variedad posible, tomando en cuenta el ambiente en donde va a trabajar el detector [3].

4. RECURSOS UTILIZADOS (EQUIPOS, ACCESORIOS Y MATERIAL CONSUMIBLE):

- ✓ Raspberry Pi4
- ✓ Monitor
- ✓ Teclado
- ✓ Mause
- ✓ CPU con Windows 7 o superior
- ✓ Fuente de voltaje 5vdc 2amp
- ✓ FlasMemory 1gb o más

5. MARCO PROCEDIMENTAL

PARTE1:
Se debe tomar muchas fotografías de la pieza que se desea identificar, al igual que el entorno donde van a estar. Se recomienda alrededor de mil fotografías positivas y mil negativas, siendo las positivas las piezas y las negativas el entorno en donde va estar la pieza a identificar.

Para tomar captura de la imagen con Python es necesario ejecutar el siguiente script el cual tomará varias fotografías en la carpeta asignada, presionando la tecla "s".

```
import cv2
import numpy as np
import imutils
import os
Datos = 'p'
if not os.path.exists(Datos):
   print('Carpeta creada: ',Datos)
   os.makedirs(Datos)
cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
x1, y1 = 190, 80
x2, y2 = 450, 398
count = 0
while True:
   ret, frame = cap.read()
   if ret == False: break
   imAux = frame.copy()
   cv2.rectangle(frame,(x1,y1),(x2,y2),(255,0,0),2)
   objeto = imAux[y1:y2,x1:x2]
   objeto = imutils.resize(objeto,width=38)
   #print(objeto.shape)
   k = cv2.waitKey(1)
   if k == ord('s'):
      cv2.imwrite(Datos+'/objeto_{}.jpg'.format(count),objeto)
      print('Imagen guardada:'+'/objeto_{}.jpg'.format(count))
      count = count +1
   if k == 27:
      break
   cv2.imshow('frame',frame)
   cv2.imshow('objeto',objeto)
cap.release()
cv2.destroyAllWindows()
```

Hay que recordar que se deben crear dos carpetas, una de imágenes positivas y otra de imágenes negativas, quedando de la siguiente manera.

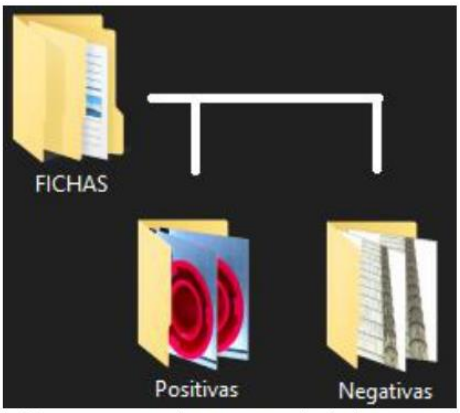


Ilustración 1. Carpeta y Subcarpetas con las imágenes a entrenar

Como se observa en la ilustración 1 las fotografías están ubicadas en dos diferentes carpetas.

PARTE2:

Para instalar el programa de entrenamiento es necesario un computador con Windows 7 o superior, en el cual nos dirigiremos a la siguiente pagina web https://amin-ahmadi.com/downloads/, y escogeremos el archivo "Interfaz gráfica de usuario de Cascade Trainer", que empezará a descargarse.
Posteriormente se debe instalar en el computador y abrirlo, con lo que nos encontraremos con la siguiente interfaz, en donde se debe seleccionar la carpera que contiene las imágenes positivas y negativas.

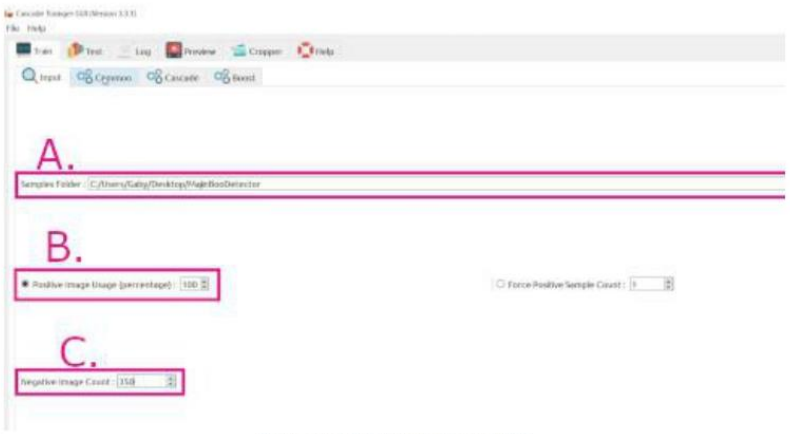


Ilustración 2. Cascade Trainer GUI

A. En esta sección debes seleccionar la carpeta en donde están contenidas las carpetas p y n.

B. Aquí necesitamos especificar el porcentaje de imágenes positivas a usar, en este caso he establecido 100.

C. Especificamos la cantidad de imágenes negativas a usar para el entrenamiento, en este caso serían 1000.

Luego pasaremos al aparatado *Common*.

Ilustración 3. Cascade Trainer GUI, apartado: Cascade.

A. Ingresamos el ancho que poseen las imágenes de entrenamiento, en este caso 38 pixeles.

B. Ingresamos el alto igualmente, de las imágenes de entrenamiento, en este caso 46 pixeles.

Finalmente damos clic en Start y esperamos a que se realice el entrenamiento. En mi caso se tardó aproximadamente 5 a 20 minutos.

Luego aparecerán nuevos archivos en el cual vamos a necesitar el que dice “cascade.xml”, y lo debemos copiar a la Raspberry.

PARTE 3:
En esta parte se va a realizar la prueba de entrenamiento para detectar la pieza, y para eso es necesario correr el siguiente script tomando en cuenta la ubicación del archivo en donde se copió el archivo cascade.xml, debe ser en la misma ubicación del script.

```
import cv2
cap = cv2.VideoCapture(0)
clasficha = cv2.CascadeClassifier('cascade.xml')
while True:
```

```
    ret,frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    toy = clasficha.detectMultiScale(gray, scaleFactor = 6.4, minNeighbors = 90, minSize=(70,78))
    for (x,y,w,h) in toy:
        cv2.rectangle(frame,(x-40,y-30),(x+w+20,y+h+40),(0,255,0),2)
        cv2.putText(frame,'Ficha',(x-40,y-40),2,0.7,(0,255,0),2,cv2.LINE_AA)
    cv2.imshow('frame',frame)

    if cv2.waitKey(1) == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

PARTE 4 (Opcional):

Adicione al script anterior un código que sea capaz de tomar una fotografía al rectángulo que ubica a la pieza en el entorno.

6. REGISTRO DE RESULTADOS:

Indique cuantas piezas y de qué color utilizó para el entrenamiento.
Roja:_____; Negra:_____; Plateada:_____

Fue suficiente mil imágenes de la pieza: SI __: NO__ , si su respuesta es NO, indique con cuantas imágenes logró el aprendizaje.

De cada diez piezas de cada color indique con cuantas tuvo algún inconveniente al momento de que el script la reconociera.
Roja:_____; Negra:_____; Plateada:_____

7. PREPARATORIO:

- ✓ Leer la guía
- ✓ Tener conceptos básicos de programación en Python
- ✓ Imprimir hoja guía con tabla de resultados

Anexo 2

Guía de laboratorio para la utilización de la estación de visión (Detección de color).

1. **DATOS INFORMATIVOS**
   a. **PRÁCTICA No:** 1
   b. **ESTACIÓN:** VISIÓN 500
   c. **TIEMPO ESTIMADO:** 2 horas
   d. **NOMBRE DEL ESTUDIANTE:** _____

2. **DATOS DE LA PRÁCTICA**
   a. **TEMA:**
   DETECCIÓN DE COLORES DE TRES DIFERENTES PIEZAS DENTRO DE UN ENTORNO CONTROLADO

   b. **OBJETIVO GENERAL**
   - Realizar un algoritmo en Python el cual reconozca los diferentes colores de las piezas (fichas) del laboratorio del MPS para su utilización en la estación de visión.

   c. **OBJETIVOS ESPECÍFICOS**
   - Utilizar el espacio de color HSV para encontrar el color rojo, plateado y negro con OpenCV
   - Dibujar contornos de acuerdo a su área, buscar su centro y visualizar sus coordenadas con OpenCV y Python
   - Detectar varios colores en OpenCV.

3. **MARCO TEÓRICO**
   a. **HSV (Hue, Saturation, Value)**

El tono, la saturación y el valor son las principales propiedades del color que nos permiten distinguir entre diferentes colores. El uso efectivo del color es uno de los elementos más esenciales en la fotografía, ya que el color puede atraer la atención del espectador hacia su composición y afectar el estado de ánimo y el impacto emocional de su foto.

   b. **Escala de color HSV**

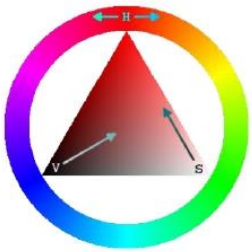
La escala HSV (que significa valor de saturación de tono) proporciona una lectura numérica de su imagen que corresponde a los nombres de color que contiene. El tono se mide en grados de 0 a 360. Por ejemplo, el cian se encuentra entre 181 y 240 grados y el magenta entre 301 y 360 grados. El

valor y la saturación de un color se analizan en una escala de 0 a 100 por ciento. La mayoría de los selectores de color digitales se basan en la escala HSV, y los modelos de color HSV son particularmente útiles para seleccionar colores precisos para arte, muestras de color y gráficos digitales.

c. HSV (Hue, Saturation, Value) en OpenCV

El espacio de color HSV (Hue, Saturation, Value / Matiz, Saturación, Brillo), posee 3 componentes, similar al espacio de color RGB que tratamos anteriormente en un post. Vamos a utilizar este espacio de color debido a que podremos determinar de forma más sencilla los rangos de los colores que deseamos detectar.

Figura 1. Espacio de color HSV



Fuente: [1]

Para determinar un color nos vamos a centrar principalmente en el componente H que corresponde al matiz. En la figura 1 podemos ver como este componente va cambiando de rojo, amarillo, verde, violeta hasta llegar nuevamente a rojo.

4. RECURSOS UTILIZADOS (EQUIPOS, ACCESORIOS Y MATERIAL CONSUMIBLE):

- ✓ Raspberry Pi4
- ✓ Monitor
- ✓ Teclado
- ✓ Mause
- ✓ Fuente de voltaje 5vdc 2amp

5. MARCO PROCEDIMENTAL

PARTE1: Imagen a procesar

Lo primero que se necesita es una imagen o fotograma para sobre este detectar los colores. Se lo puede conseguir de dos formas, leyendo una imagen o a través de un video. En esta ocasión se harás a través de un video streaming.

PARTE2: Transformar de BGR a HSV

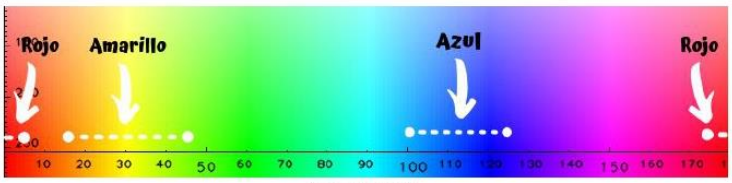
Por defecto OpenCV lee a las imágenes o fotogramas en BGR, por ello es necesario transformarlas al espacio de color HSV. Para ello nos ayudaremos de la función cv2.cvtColor, como primer argumento le daremos la imagen a transformar, y luego cv2.COLOR_BGR2HSV, ejecutar el siguiente algoritmo.

```
import cv2
import numpy as np
cap = cv2.VideoCapture(0)
while True:
  ret,frame = cap.read()
  if ret==True:
    frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('s'):
      break
cap.release()
cv2.destroyAllWindows()
```

PARTE 3: Determinar los rangos en donde se encuentra el color a detectar

Para el ejemplo se detectará el color rojo que se encuentra al principio y al final del componente H como se observa en la figura 2.

Figura 2. Vista de los componentes HSV



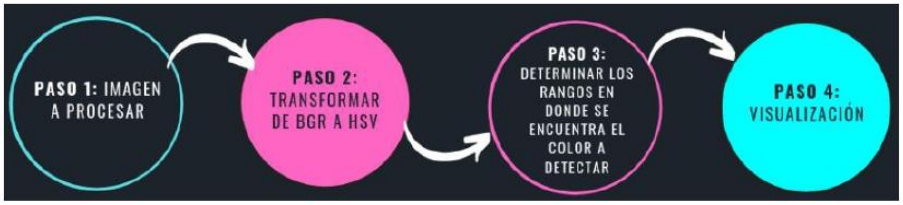
Fuente: [1]

```
import cv2
import numpy as np
cap = cv2.VideoCapture(0)
redBajo1 = np.array([0, 100, 20], np.uint8)
redAlto1 = np.array([8, 255, 255], np.uint8)
redBajo2=np.array([175, 100, 20], np.uint8)
redAlto2=np.array([179, 255, 255], np.uint8)
while True:
  ret,frame = cap.read()
  if ret==True:
    frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    maskRed1 = cv2.inRange(frameHSV, redBajo1, redAlto1)
    maskRed2 = cv2.inRange(frameHSV, redBajo2, redAlto2)
    maskRed = cv2.add(maskRed1, maskRed2)
    maskRedvis = cv2.bitwise_and(frame, frame, mask= maskRed)
    cv2.imshow('frame', frame)
    cv2.imshow('maskRed', maskRed)
    cv2.imshow('maskRedvis', maskRedvis)
    if cv2.waitKey(1) & 0xFF == ord('s'):
      break
cap.release()
cv2.destroyAllWindows()
```

PARTE 4: Detectar varios colores en OpenCV

Se debe detectar el color azul, amarillo y rojo, y te darás cuenta que va a ser un proceso similar.

Figura 3. Pasos para la detección de colores



Ejecutar el siguiente algoritmo para la detección de distintos colores.

```
import cv2
import numpy as np
def dibujar(mask,color):
  _,contornos,_ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
     cv2.CHAIN_APPROX_SIMPLE)
  for c in contornos:
    area = cv2.contourArea(c)
    if area > 3000:
      M = cv2.moments(c)
      if (M["m00"]==0): M["m00"]=1
      x = int(M["m10"]/M["m00"])
      y = int(M['m01']/M['m00'])
      nuevoContorno = cv2.convexHull(c)
      cv2.circle(frame,(x,y),7,(0,255,0),-1)
      cv2.putText(frame,'{},{}'.format(x,y),(x+10,y), font, 0.75,(0,255,0),1,cv2.LINE_AA)
      cv2.drawContours(frame, [nuevoContorno], 0, color, 3)
cap = cv2.VideoCapture(0)
azulBajo = np.array([100,100,20],np.uint8)
azulAlto = np.array([125,255,255],np.uint8)
amarilloBajo = np.array([15,100,20],np.uint8)
amarilloAlto = np.array([45,255,255],np.uint8)
redBajo1 = np.array([0,100,20],np.uint8)
redAlto1 = np.array([5,255,255],np.uint8)
redBajo2 = np.array([175,100,20],np.uint8)
redAlto2 = np.array([179,255,255],np.uint8)
font = cv2.FONT_HERSHEY_SIMPLEX
while True:
  ret,frame = cap.read()
  if ret == True:
    frameHSV = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
    maskAzul = cv2.inRange(frameHSV,azulBajo,azulAlto)
    maskAmarillo = cv2.inRange(frameHSV,amarilloBajo,amarilloAlto)
    maskRed1 = cv2.inRange(frameHSV,redBajo1,redAlto1)
    maskRed2 = cv2.inRange(frameHSV,redBajo2,redAlto2)
    maskRed = cv2.add(maskRed1,maskRed2)
    dibujar(maskAzul,(255,0,0))
```

```
    dibujar(maskAmarillo,(0,255,255))
    dibujar(maskRed,(0,0,255))
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('s'):
      break
cap.release()
cv2.destroyAllWindows()
```

PARTE 5: Identificación de colores de las piezas del laboratorio

Como último paso se debe realizar un algoritmo que sea capaz de reconocer los colores de las piezas que se encuentran en el laboratorio, las cuales son de color rojo, negro, plateado.

6. REGISTRO DE RESULTADOS:

- Indique los valores HSV que se usó para la detección de piezas de color:
  Roja: H_____; S_____; V_____ / H_____; S_____; V_____
  Negra: H_____; S_____; V_____
  Plateada: H_____; S_____; V_____

- Indique los factores que tomó en cuenta para definir los anteriores rangos.
  _____
  _____
  _____

- Porque es necesario utilizar este método de HSV y no BGR?.
  _____
  _____
  _____

7. PREPARATORIO:

- ✓ Leer la guía
- ✓ Tener conceptos básicos de programación en Python
- ✓ Imprimir hoja guía con tabla de resultados