



Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Trabajo práctico N° 02

Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento de imágenes (IA4.4)

Integrantes	Legajo	Mail
Martinez Dufour, Caterina	M-7169/2	caterinaxmd7@gmail.com

Fecha de entrega: 02/06/2025

Docentes: Gonzalo, Sad - Julián, Álvarez - Juan Manuel, Calle



ÍNDICE

1. Introducción	Página 3
2. Problema 1 - Detección y clasificación de componentes electrónicos.....	Página 4-8
3. Problema 2 - Identificación de resistencias eléctricas.....	Página 9-19
4. Conclusión.....	Página 20



1. Introducción

En este informe se explicará cómo se abordaron los problemas planteados utilizando diferentes estrategias vistas hasta el momento en la materia. Para cada solución, se comentarán los inconvenientes que surgieron durante el desarrollo, las modificaciones realizadas y la forma en que se resolvieron.

Además, se incluirán imágenes que faciliten la comprensión de los resultados obtenidos y en algunos casos, se presenta un ejemplo de su funcionamiento. El trabajo fue realizado en un repositorio de GitHub, que contenía un archivo README con instrucciones para poder ejecutar el programa paso a paso, dos archivos .py con las soluciones a los problemas dados y los archivos proporcionados por la cátedra.

2. Problema 1 - Detección y clasificación de componentes electrónicos

Descripción del problema:

El problema consiste en analizar una placa de circuito impreso (PCB) de una imagen a color que contiene una variedad de componentes electrónicos soldados. Se requiere:

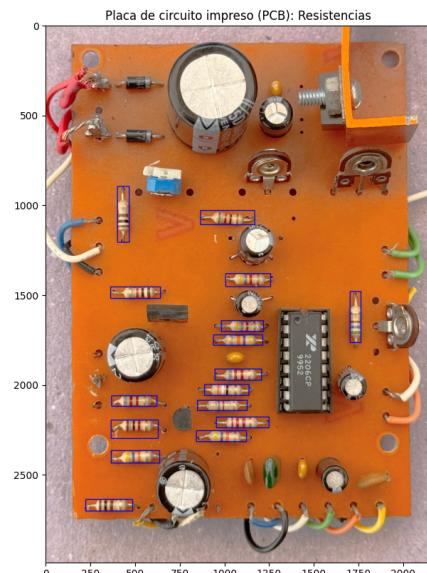
- Identificar y segmentar la imagen clasificándolo por tres tipos de componentes: resistencias, capacitores y chip. Debe devolver una imagen mostrando estas segmentaciones.
- Clasificar los capacitores electrolíticos en diferentes categorías según su tamaño y devolver una imagen mostrando estas divisiones.
- Contar la cantidad de resistencias eléctricas que se encuentran en la imagen y devolver por consola el resultado obtenido.

Resolución del problema:

Para resolver este problema se desarrollaron distintas funciones organizadas por secciones para mostrar paso a paso el funcionamiento de cada parte del proceso. Las funciones implementadas son las siguientes:

- **Función cargar_imagen:** Carga la imagen en color y la convierte en formato BGR (predeterminado por OpenCV) a RGB. Devuelve la imagen leída.
- **Función detectar_resistencias:** Esta función se encarga de detectar las resistencias de la imagen. Convierte la imagen a escala de grises, se aplica un umbral automático y luego con diferentes operaciones morfológicas mejoramos el resultado obtenido.

Utiliza componentes conectados y el contorno para filtrar según su forma, área y proporción. Devuelve la imagen marcada, las coordenadas de las resistencias,

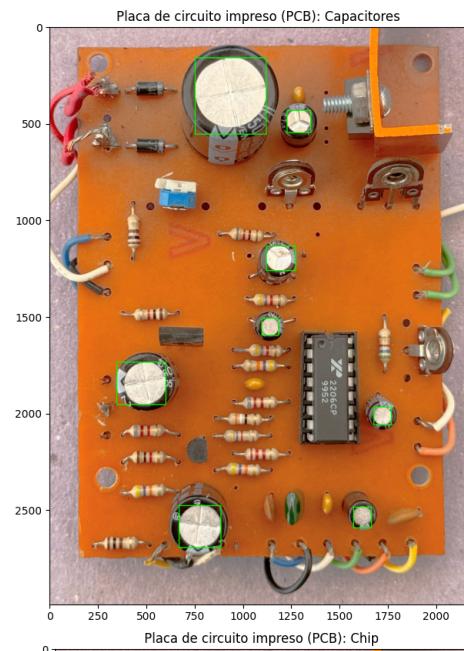


la cantidad de resistencias que se detectó y la imagen umbralizada.

- ***Función detectar_capacitores:***

Esta función detecta capacitores en la imagen binaria utilizando el mismo procedimiento que “funcion_resistencia” pero trabajando con la imagen umbralizada invertida y con las coordenadas de las resistencias, las elimina de la imagen.

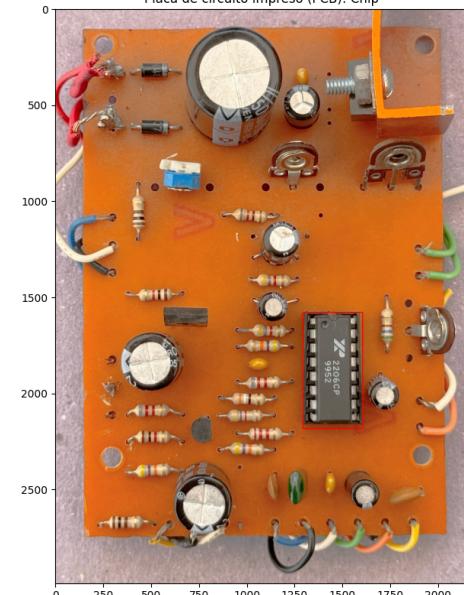
Se aplican operaciones morfológicas de clausura y dilatación para mejorar la segmentación. Filtra según la forma, área y proporción y devuelve la imagen marcada, las coordenadas de los capacitores y sus áreas.



- ***Función detectar_chip:***

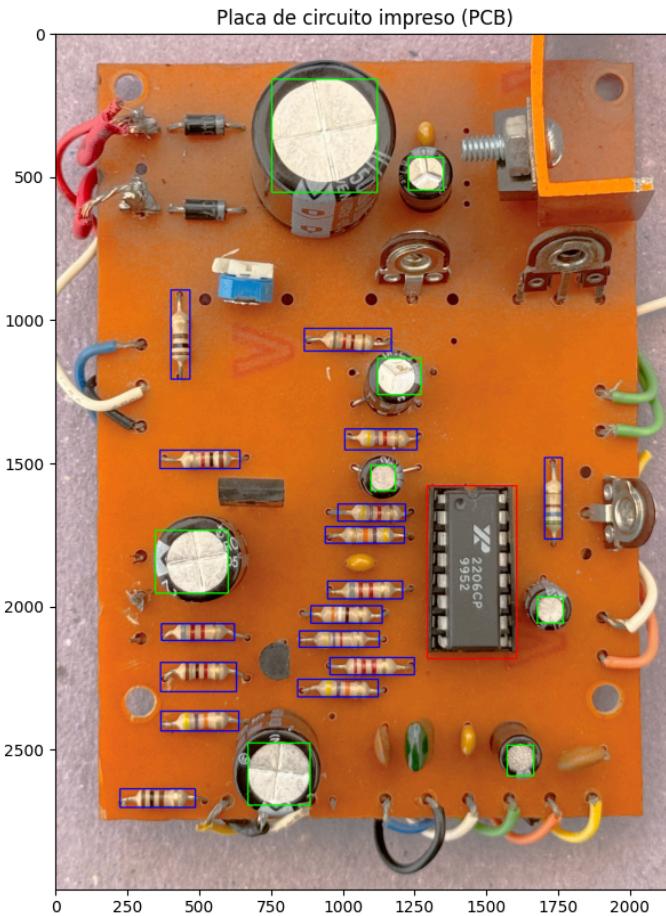
Esta función detecta el chip de la placa aplicando un umbral fijo sobre la imagen en escala de grises, a diferencia de las dos funciones anteriores que utilizaban un umbral automático.

Invertimos la imagen binaria y se eliminan las regiones donde se encuentran las resistencias y los capacitores. Se aplican operaciones morfológicas de clausura y apertura para unir partes del chip y eliminar el ruido. Devuelve la imagen con el chip marcado.



- ***Función imprimir_segmentaciones:***

Muestra por pantalla la imagen final con todos los componentes electrónicos segmentados: resistencias, capacitores y el chip.

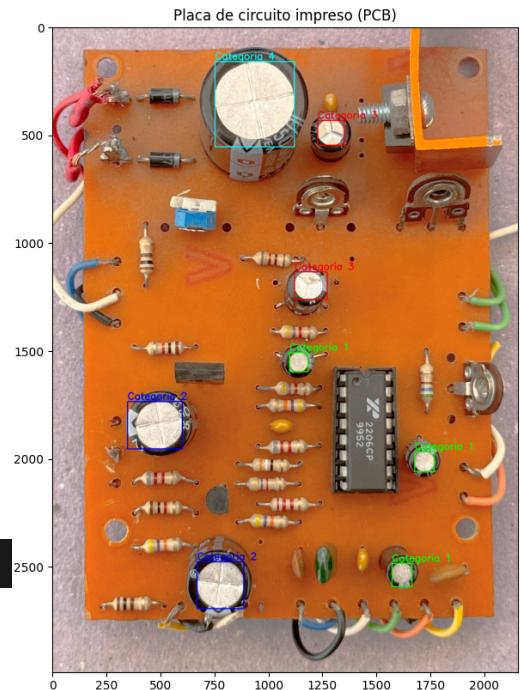


- ***Función clasificar_capacitores:***

Esta función clasifica los capacitores detectados en diferentes categorías según el valor de su área. A cada uno se le asigna una categoría y se dibuja un rectángulo de color junto con una etiqueta sobre la imagen.

- ***Función contar_resistencias:*** Imprime por consola la cantidad total de resistencias detectadas en la imagen procesada.

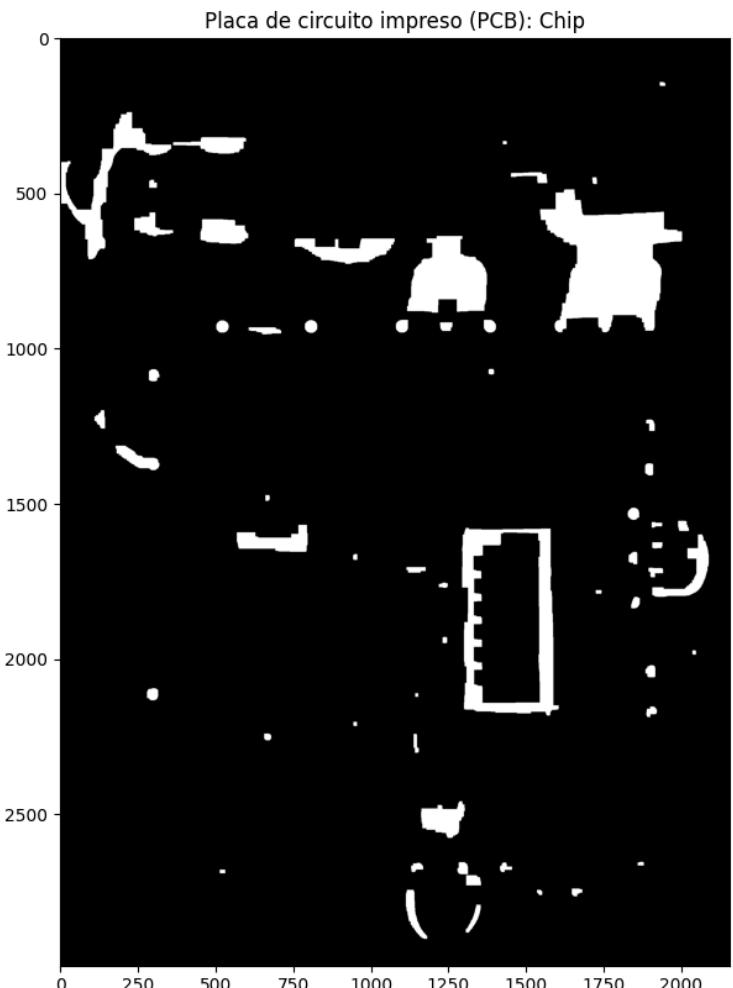
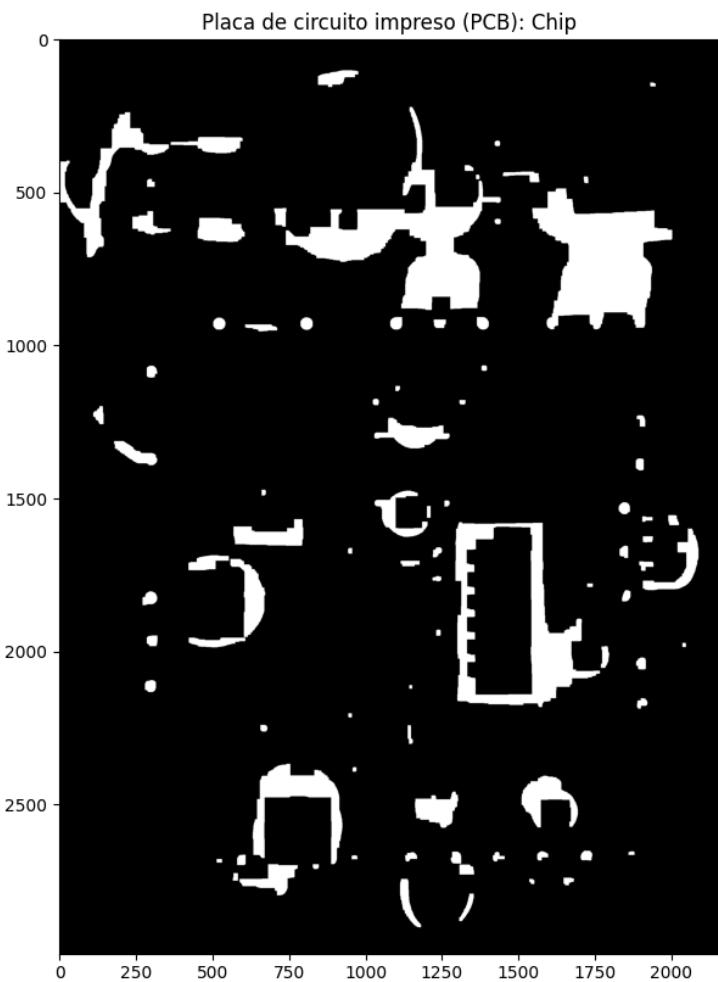
El total de resistencias detectadas en la placa son: 16



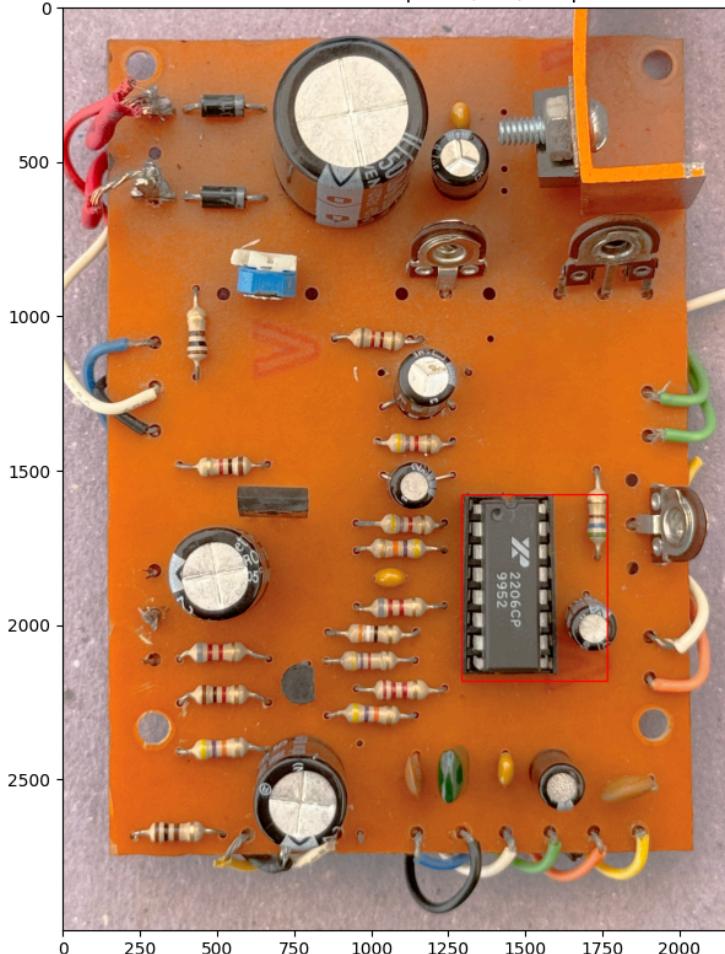
Problemas que tuve hasta encontrar la solución:

Al intentar detectar el chip tuve problemas debido a que utilizaba un umbral automático que no detectaba las zonas negras del chip. Probé trabajar con un rango de colores HSV pero no me devolvieron el resultado esperado. Es por eso que opte por aplicar un umbral fijo, ajustándolo manualmente y luego aplique operaciones morfológicas para mejorar la calidad de la segmentación.

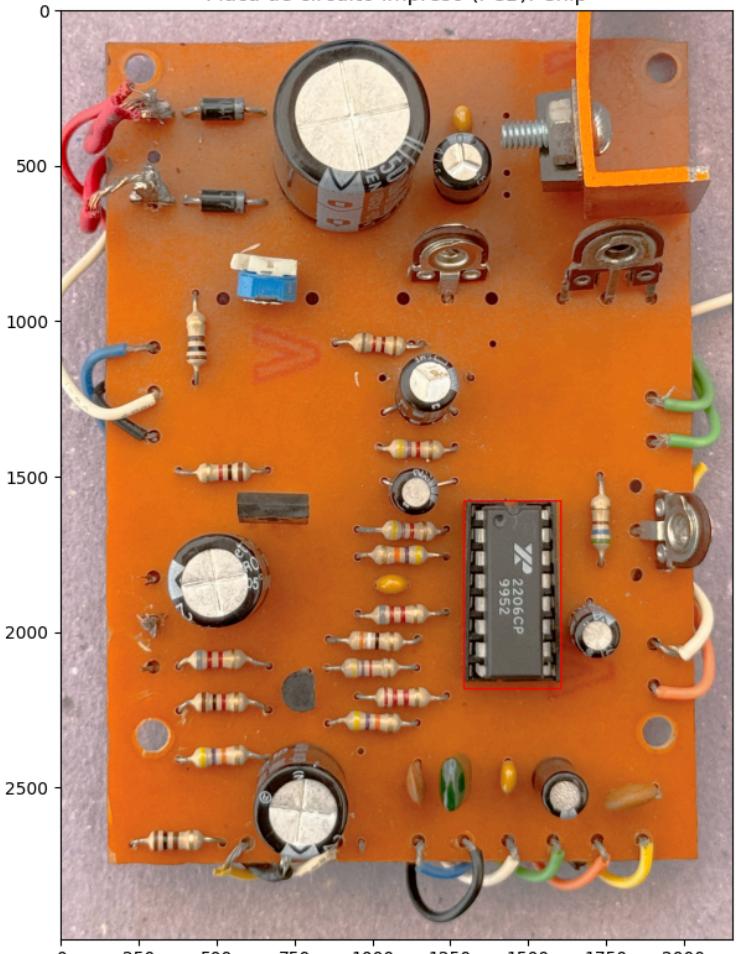
Sin embargo, al aplicar operaciones morfológicas un capacitor cercano se unía a la región del chip, lo que me provocaba que detectara una zona más grande de lo que realmente era. Es por esto que incluir un margen mayor a la región que elimina de la imagen los capacitores según su coordenada, lo que me permitió detectar finalmente de forma correcta el chip.



Placa de circuito impreso (PCB): Chip



Placa de circuito impreso (PCB): Chip





3. Problema 2 - Identificación de resistencias eléctricas

Descripción del problema:

Se dispone de un conjunto de 10 resistencias eléctricas, cada una desde 4 perspectivas distintas, lo que resulta en un total de 40 imágenes a color en formato JPG. El objetivo es procesar estas imágenes utilizando las herramientas aprendidas para resolver lo siguiente:

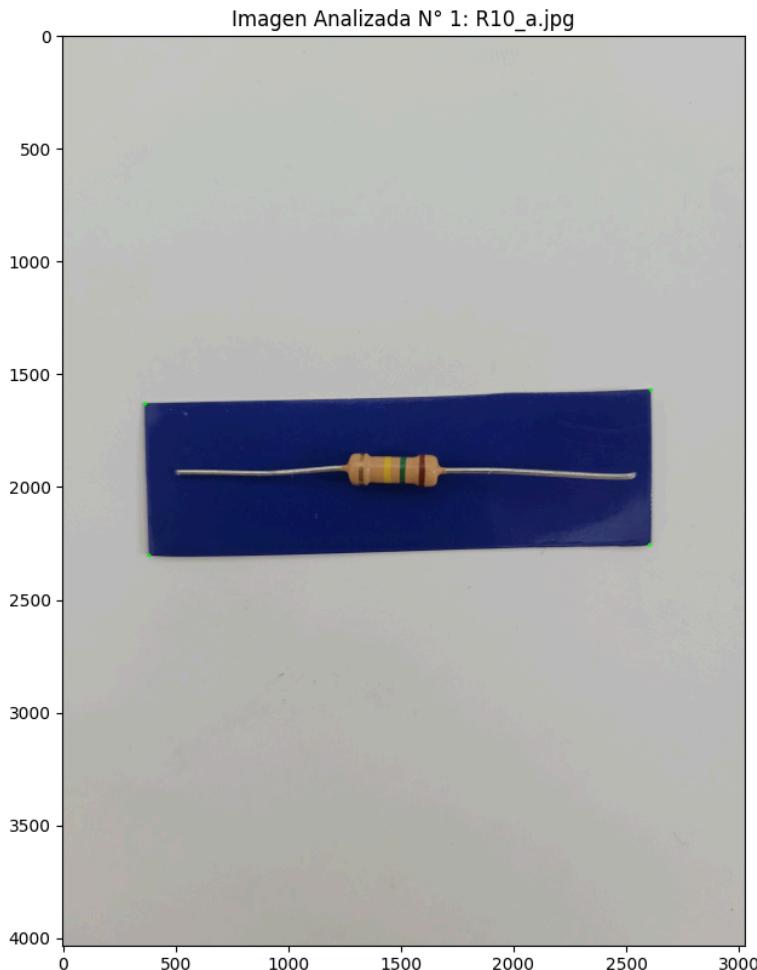
- Identificar y recortar cada imagen el rectángulo azul que contiene la resistencia de las 40 imágenes, transformándola para obtener la vista superior.
- Almacenar las imágenes resultantes del inciso anterior renombrarlas con “_out”. por ejemplo: “R1_a_out”.
- De las imágenes guardadas, se examina el código de color de solamente 10 resistencias llamadas “Rx_a_out”. Se deben considerar las 4 bandas de color, pero ignorar la banda de tolerancia (de color dorado). Las bandas deben ordenarse según su posición relativa a la banda dorada:
 - **Banda 1:** Color más alejado de la dorada.
 - **Banda 2:** Color central.
 - **Banda 3:** Color más cercano a la dorada.
- Por último, debemos calcular el valor numérico de cada resistencia y presentar el resultado en pantalla en ohmios (Ω).

Resolución del problema:

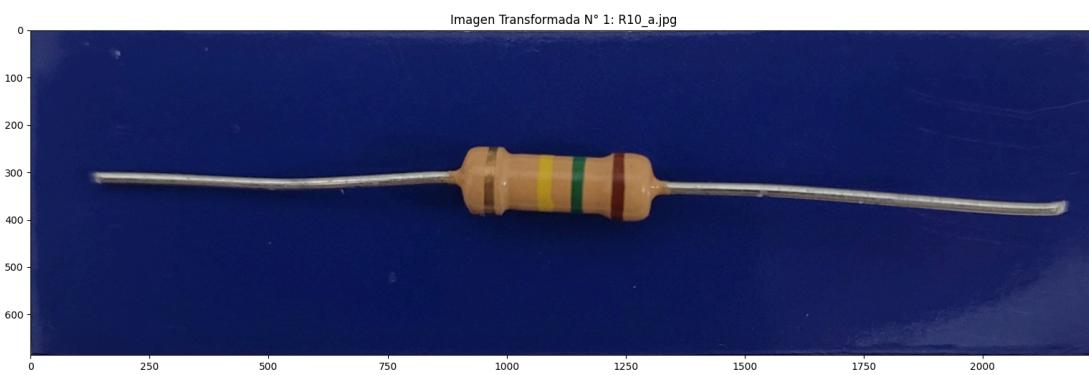
Para resolver este problema se desarrollaron distintas funciones organizadas por secciones para mostrar paso a paso el funcionamiento de cada parte del proceso. Las funciones implementadas son las siguientes:

- **función cargar_imagen:** Carga la imagen en color y la convierte en formato BGR (predeterminado por OpenCV) a RGB. Devuelve la imagen leída junto a su nombre.

- **Función analizar_imagen:** Analiza cada imagen para identificar los cuatro vértices del rectángulo azul que contiene la resistencia y las marca en la imagen. Devuelve los puntos detectados de la imagen.



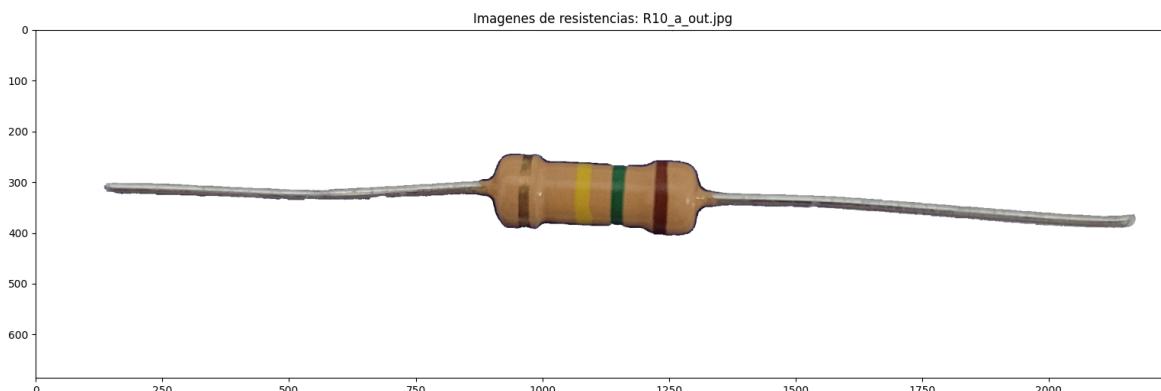
- **Función transformar_imagen:** A partir de los puntos detectados, se ordena cada uno para poder aplicar una homografía que permita obtener la vista superior de la resistencia. Devuelve la imagen transformada.



- **Función guardar_imagen:** Luego de obtener las imágenes transformadas de la función anterior, renombramos el nombre de cada una agregando el sufijo “_out” y las guardamos en una carpeta llamada “resistencias_out”. Devuelve el nombre de la imagen y la imagen.



- **Función quitar_fondo:** Esta función evalúa solamente 10 imágenes llamadas “RX_a_out” (x varía del 1 al 10). Para eliminar el fondo azul lo hacemos con una mascara de color HSV, transformandolo en blanco para evitar problemas en otras funciones. Devuelve la imagen sin fondo y sus nombres de archivo.

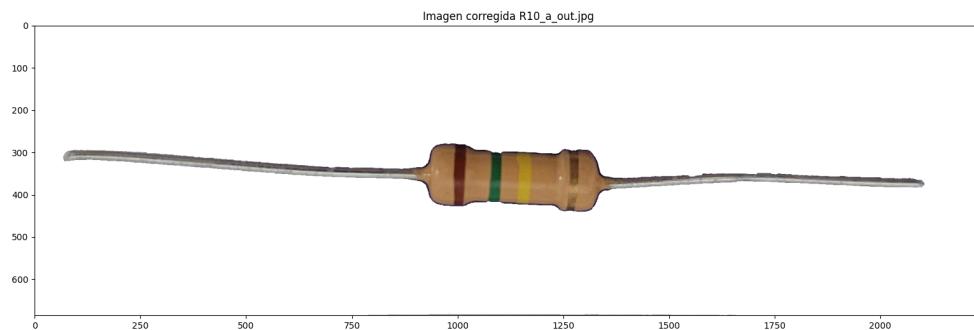


- **Función color_bandas:** Detecta los colores de las bandas utilizando máscaras de colores con HSV. Dibuja un rectángulo sobre cada banda detectada y etiqueta el color correspondiente en la imagen. Devuelve la lista de bandas detectadas y la imagen con las marcas.

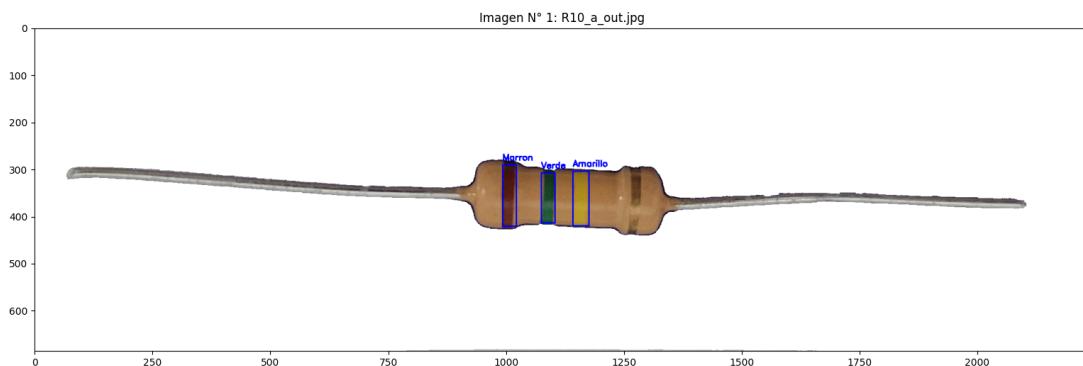
- **Función detectar_resistencia:** Detecta la resistencia con una máscara HSV y la mejora con operaciones morfológicas. Devuelve la coordenada x de la resistencia junto a su etiqueta. Ejemplo de uso:



- **Función corregir_img:** Verifica si la imagen posee la banda de tolerancia a la izquierda, si la distancia entre la primera banda y la resistencia es mayor o igual a la distancia entre la primera y tercer banda, entonces gira la imagen 180° grados. Devuelve todas las imágenes con la resistencia en la derecha. Ejemplo de uso:



- **Función detectar_bandas:** Invoca a la “función color_bandas”, “detectar_resistencia” y “corregir_img”. Con los resultados obtenidos en cada una de ellas, ordena las bandas de izquierda a derecha según la coordenada x y lo almacena en una lista. Devuelve los colores detectados ordenados y el nombre de la imagen procesada.





- **Función valor_Ohms:** Calcula el valor de la resistencia en función del color de las bandas detectadas. Muestra en pantalla el nombre de la resistencia, el valor de cada banda y el valor total en ohmios (Ω).

```
-----  
Resistencia: R10_a_out.jpg  
Banda 1: Amarillo - valor: 4  
Banda 2: Verde - valor: 5  
Banda 3: Marron - valor: 10  
Valor total de la resistencia: 450 Ohms  
-----
```

Todas las funciones se integran al final del código, lo que permite observar el funcionamiento de cada etapa del proceso de forma individual, activando la visualización con el parámetro “mostrar = True”. De esta manera podemos observar cómo actúa cada función sobre las imágenes.

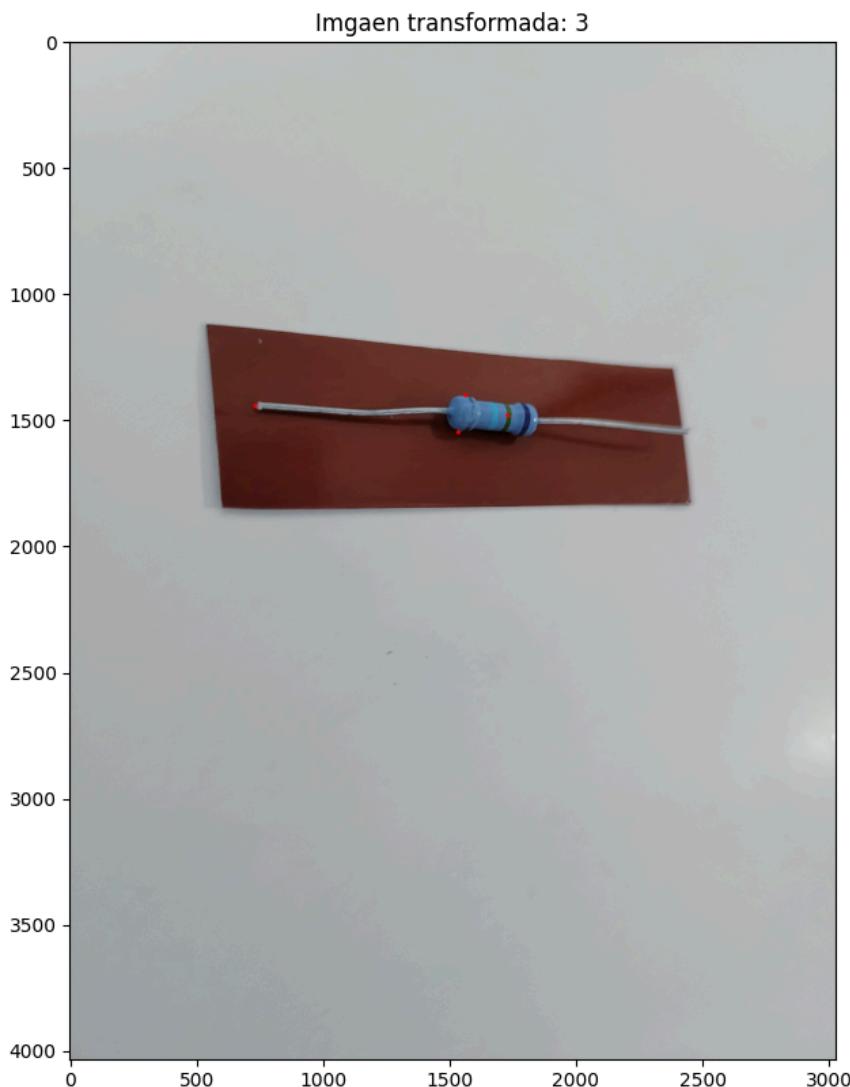
En el caso de las funciones “corregir_img” y “detectar_resistencia” (detección de la banda de tolerancia), el parámetro mostrar se modifica únicamente dentro de los parámetros de llamada, ya que estas funciones se invocan internamente dentro de “detectar_bandas” y no directamente desde el bloque principal.

Problemas que tuve hasta encontrar la solución:

Mientras resolví los diferentes puntos del problema tuve algunos inconvenientes en ciertas secciones:

Detección de esquinas del rectángulo azul:

Al principio intentaba detectar los vértices del rectángulo azul utilizando directamente la imagen en color y aplicando un filtro por canal (split) utilizando el canal azul del RGB. Además, no aplicaba ningún tipo de difuminado porque cuando realizaba esto para detectar los bordes con canny, había imágenes ya con un grado de suavizado lo que me hacía que en esas imágenes, no detecte los puntos adecuadamente. A continuación adjunto una imagen de lo que sucedía:



Para solucionar este problema, decidí transformar la imagen a escala de grises, convertirla en una imagen binaria y luego realizar una operación morfológica de clausura para llenar los huecos. Con esto ahora si luego de detectar su contorno con findcontours logre detectar los vértices adecuadamente de las 40 imágenes. A continuación muestro la imagen de como quedó la máscara y el resultado final:

Imagen Analizada N° 3: R10_c.jpg

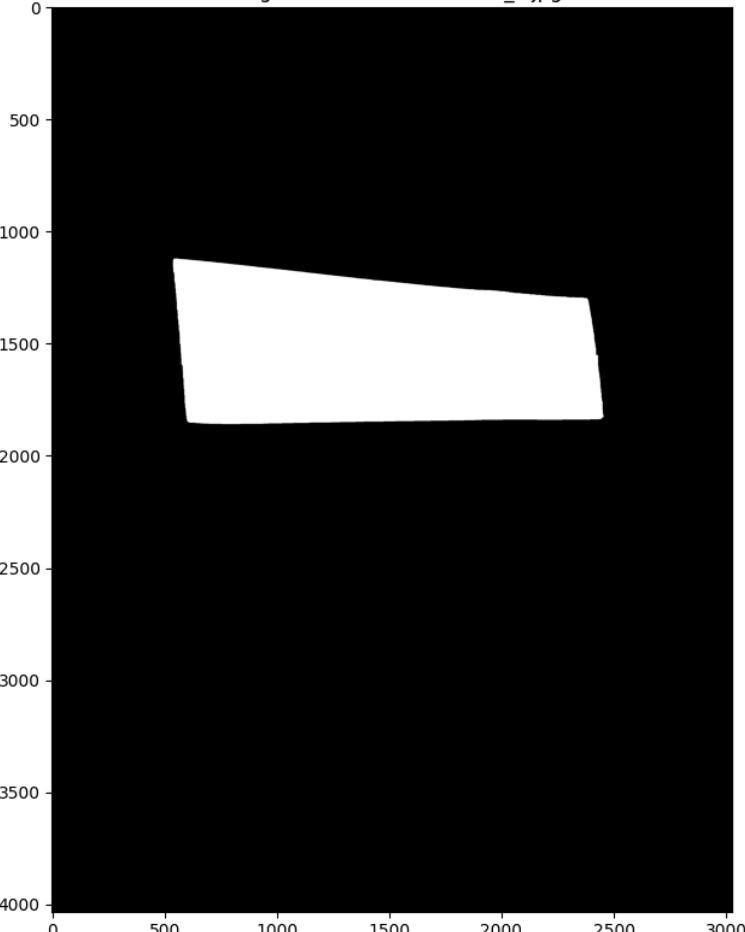
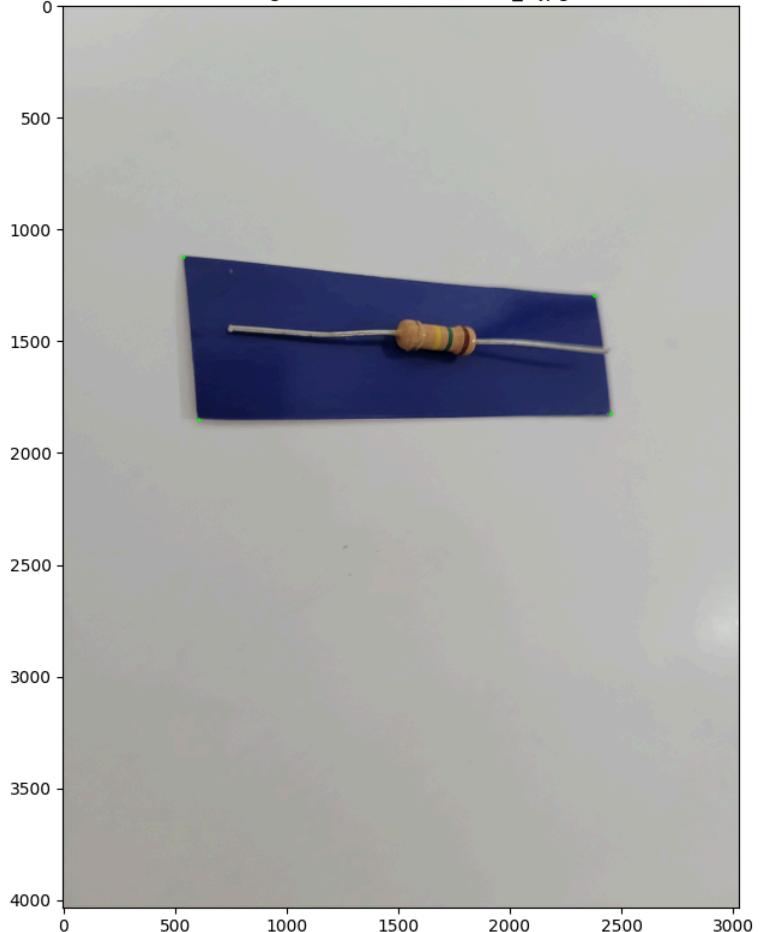


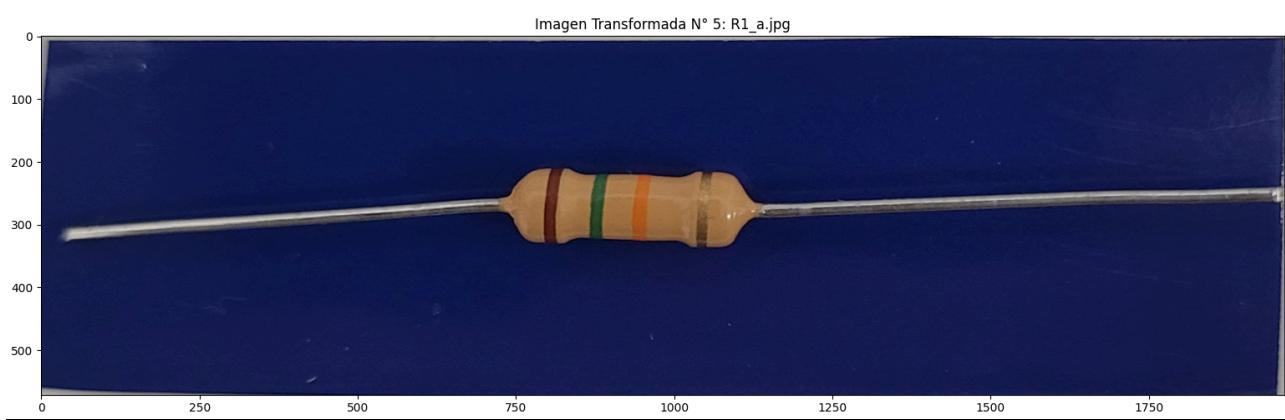
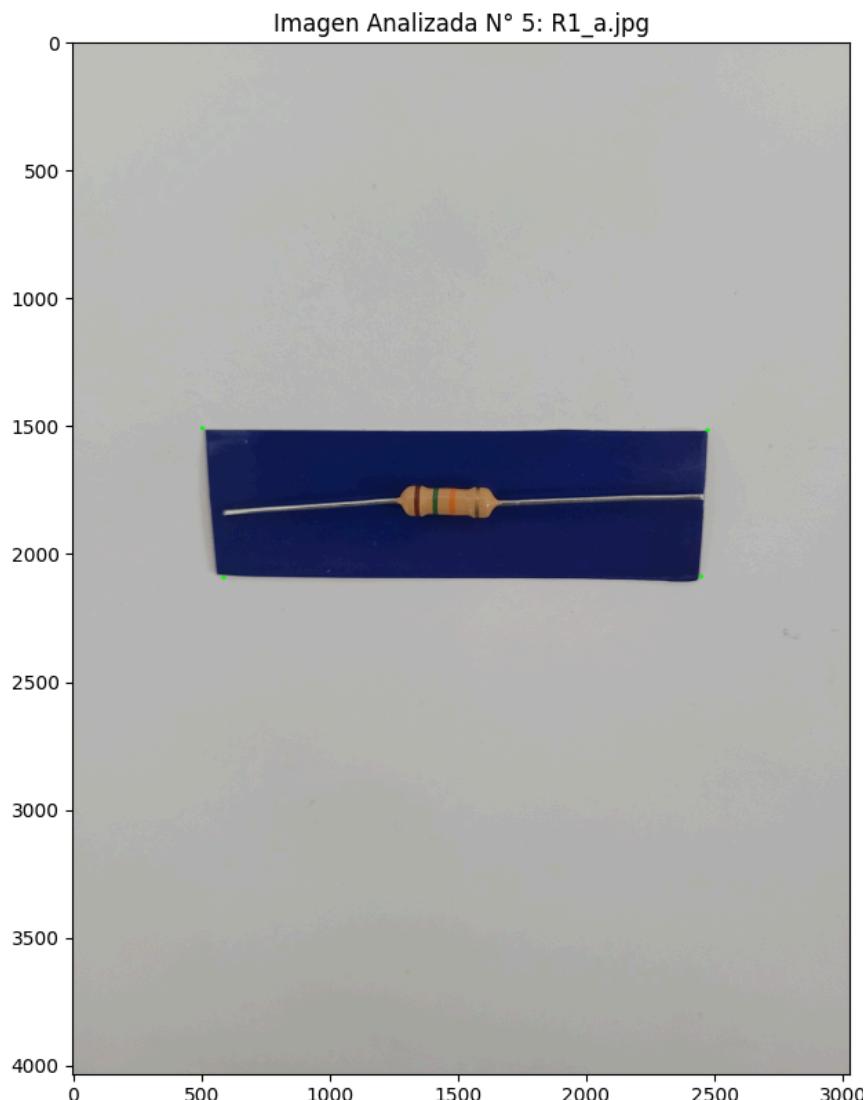
Imagen Analizada N° 3: R10_c.jpg



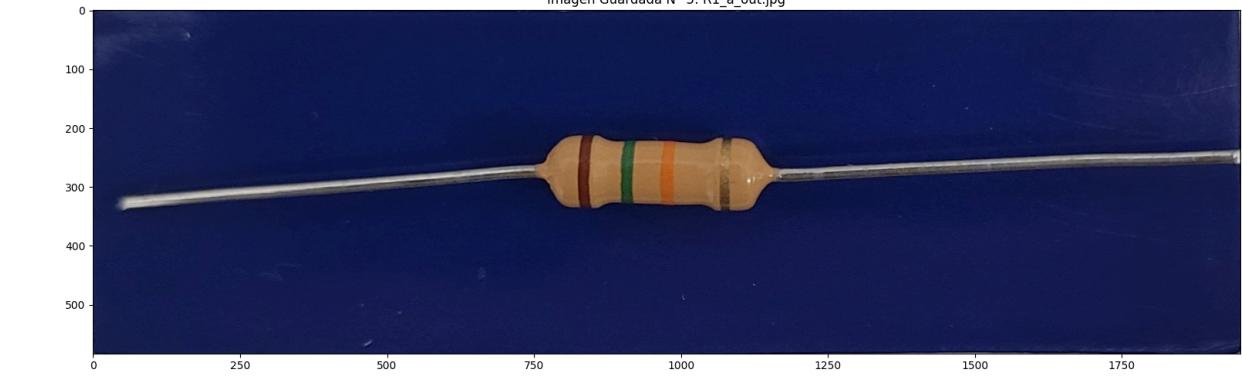
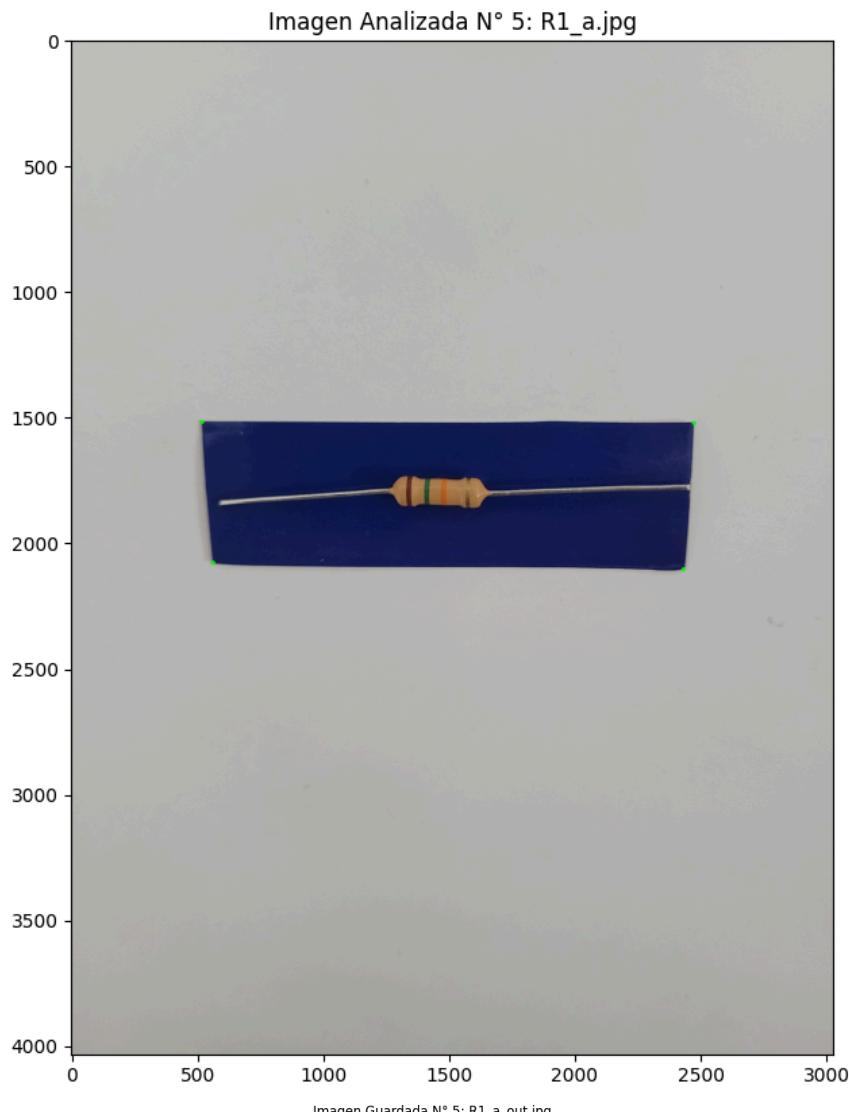
Obtener la imagen en vista superior:

En esta función si bien se logró detectar el punto anterior los vértices del rectángulo azul, al aplicar la transformación por homografía los resultados no

daban lo esperado. Los vértices estaban mal posicionados y aparecían residuos del fondo que no eran del rectángulo azul. A continuación adjunto imágenes de lo sucedido:



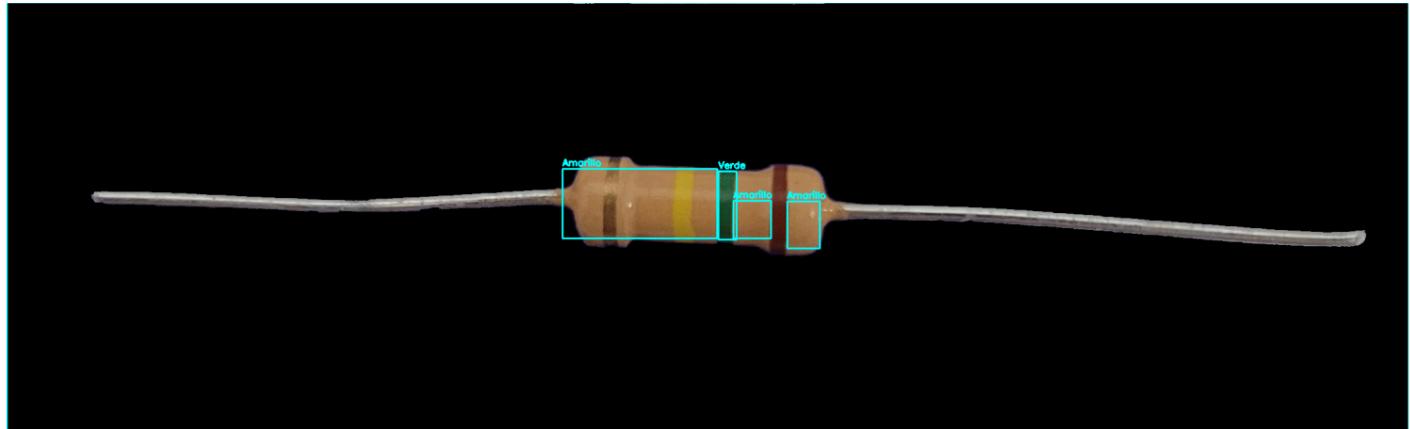
Este comportamiento se debía a que la máscara utilizada para identificar la zona de interés estaba mal formulada, como se comentó en el punto anterior. Es por eso que al aplicar la solución antes mencionada, logramos los siguientes resultados:



Detección de bandas de colores:

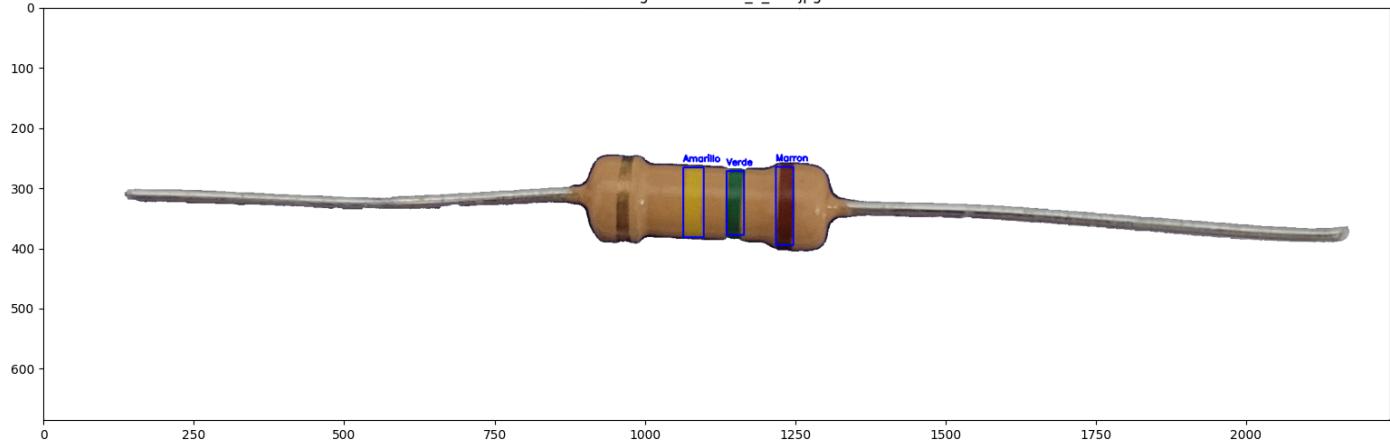
Al principio, eliminé el fondo azul de cada resistencia utilizando una máscara HSV reemplazándolo por un fondo en color negro. Luego para detectar cada color de la banda utilice diversas máscaras HSV pero a pesar de que los valores que seleccioné eran correctos, al ejecutar la función no me detectaba los colores, excepto el verde. A continuación adjunto una imagen del problema:

Imagen N° 1: R10_a_out.jpg



Luego de varios intentos, me percate que el fondo negro podria causar errores en la detección de colores asi que al reemplazar el fondo negro por uno blanco esto me permitio detectar correctamente todos los colores de las bandas utilizando rangos definidos:

Imagen N° 1: R10_a_out.jpg



**Detección de la banda de tolerancia:**

Al principio tuve problemas al detectar la banda de tolerancia debido que intentaba obtenerlo con diferentes umbrales de colores en HSV pero al ser esta banda de color dorado no podía detectarse fácilmente. Este color al ser una mezcla y no un color homogéneo como el de las bandas hacía difícil encontrarlo de esta manera.

Probé intentar obtenerlo bajando la cantidad de colores que se detectaba en la imagen pero esto causó que se perdiera la banda con el fondo de la resistencia que es de un tono similar. Al final para poder solucionarlo simplemente medir la distancia entre las bandas más lejanas y compararla con la distancia de la primera banda con la resistencia, si la distancia era mayor al de entre las bandas, eso quería decir que la banda de tolerancia se encontraba del lado izquierdo, entonces había que girar la imagen 180 grados para que queden todas las imágenes con la banda de tolerancia del lado derecho para analizar todas de manera correcta.



4. Conclusión

En este trabajo se demostró que, aplicando lo aprendido en clase, es posible tratar imágenes complejas ya sean a color, con distintas perspectivas, fondos no homogéneos o incluso con borrosidad utilizando herramientas de procesamiento de imágenes. A pesar de que a simple vista pueda parecer complicado, con las librerías de numpy, matplotlib, opencv-contrib-Python y Pillow, se pueden manipular y analizar este tipo de imágenes de manera efectiva.

En el primer ejercicio aplique umbralización, operaciones morfológicas, segmentación y criterios geométricos como área y factor de forma para detectar y clasificar componentes electrónicos en una placa PCB. Aprendí que no siempre existe un único camino para resolver un problema y que, muchas veces, conviene pensar cada parte por separado y aplicar soluciones distintas si es necesario. Como bien dice el dicho: "si el código funciona, no se toca".

En el segundo ejercicio trabaje con la detección de bandas de color en resistencias individuales. Utilice la homografía para corregir la perspectiva de las imágenes detectando las esquinas de cada rectángulo, y luego aplicamos esa transformación. También incorpore el uso de la librería os para automatizar el guardado de los resultados, y emplee rango de colores HSV para detectar las bandas. Además, calcule las distancias para ubicar correctamente la banda de tolerancia y, con esa información, pude calcular el valor de cada resistencia.

Ambos ejercicios muestran que, con las herramientas adecuadas y paciencia para ajustar los parámetros, es posible resolver problemas complejos de forma robusta y automatizada. Este trabajo práctico me permitió aplicar en la práctica los conceptos teóricos vistos en clase y resolver de forma completa los ejercicios propuestos.