FCEIA - UNR

Procesamiento de Imágenes

Trabajo Práctico N°. 1

1er cuatrimestre 2025

Gianfranco Frattini

Alejandro Peralta

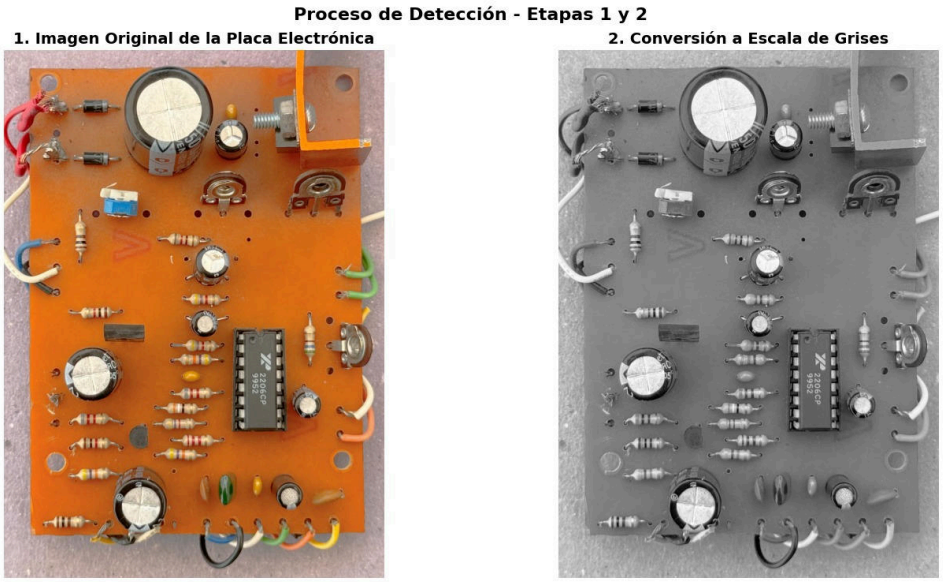
Matias Prado

Índice

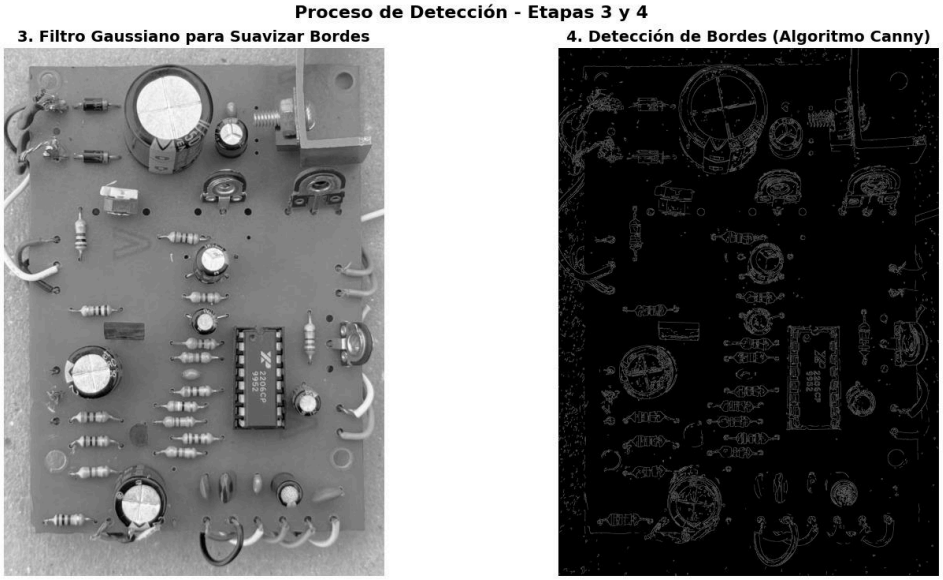
Ejercicio 1

El proceso de detección y clasificación de los diferentes componentes en la imagen se realiza a través de una serie de etapas, desde la carga de la imagen hasta el análisis final de las componentes detectadas. Aquí se detallan los pasos:

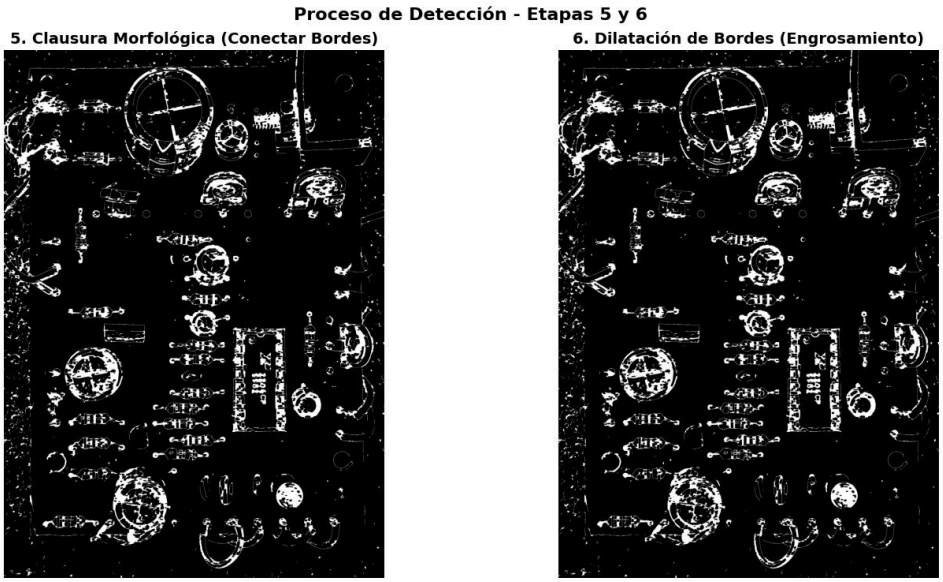
1. ETAPA 1: Cargar y convertir imagen original. El script inicia cargando la imagen desde una ruta específica y la convierte al espacio de color RGB para su visualización y procesamiento posterior.
2. ETAPA 2: Conversión a escala de grises. Se crea una versión en escala de grises de la imagen, lo cual es un paso común para muchos algoritmos de procesamiento de imágenes que operan mejor en monocromo.



3. ETAPA 3: Aplicación de filtro Gaussiano. Se aplica un filtro Gaussiano a la imagen en escala de grises para suavizar los bordes y reducir el ruido, preparando la imagen para la detección de bordes.
4. ETAPA 4: Detección de bordes (Algoritmo Canny). Utilizando el algoritmo Canny, el script identifica los bordes de los objetos en la imagen suavizada.



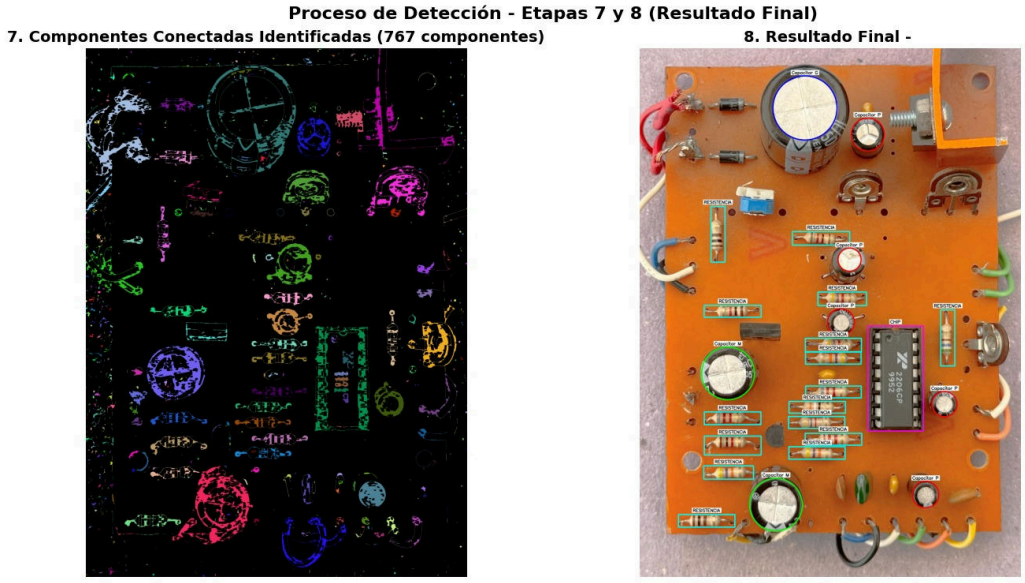
5. ETAPA 5: Operación morfológica de clausura. Se aplica una operación morfológica de clausura a los bordes detectados. Esta operación ayuda a conectar segmentos de bordes que podrían estar separados, cerrando pequeñas brechas.
6. ETAPA 6: Dilatación de bordes. Los bordes conectados se dilatan para engrosarlos, lo que puede ayudar a asegurar que las componentes conectadas sean capturadas adecuadamente en la siguiente etapa.



7. ETAPA 7: Análisis de componentes conectadas. El script utiliza la función cv2.connectedComponentsWithStats sobre la imagen con los bordes engrosados para identificar cada región contigua de píxeles blancos como una componente conectada individual. Se obtienen estadísticas para cada componente, incluyendo el área, las dimensiones

del bounding box y el centroide. Estas componentes son visualizadas inicialmente con colores aleatorios.

8. ETAPA 8: Detección y clasificación final. Se itera sobre cada componente conectada identificada (excepto el fondo, que es la etiqueta 0). La clasificación se basa en criterios de área, forma (relación de aspecto y detección de círculos) y color:



◦ CHIPS: Si el área de una componente es muy grande (mayor a 10000 píxeles), se clasifica directamente como CHIP. Se dibuja un rectángulo magenta alrededor de ellos y se cuenta el total de chips.

◦ Capacitores: Si el área es mayor a 3200 píxeles y la relación de aspecto del bounding box es cercana a 1 (menor a 1.7), el script asume que podría ser un capacitor y busca círculos usando la transformada de Hough dentro de esa región.

- Si se detectan círculos, se clasifican por su radio:

• Capacitores Pequeños: Si el radio es menor a 100 píxeles. Se dibujan en rojo y se cuentan.

• Capacitores Medianos: Si el radio es menor a 150 píxeles pero no pequeño. Se dibujan en verde y se cuentan.

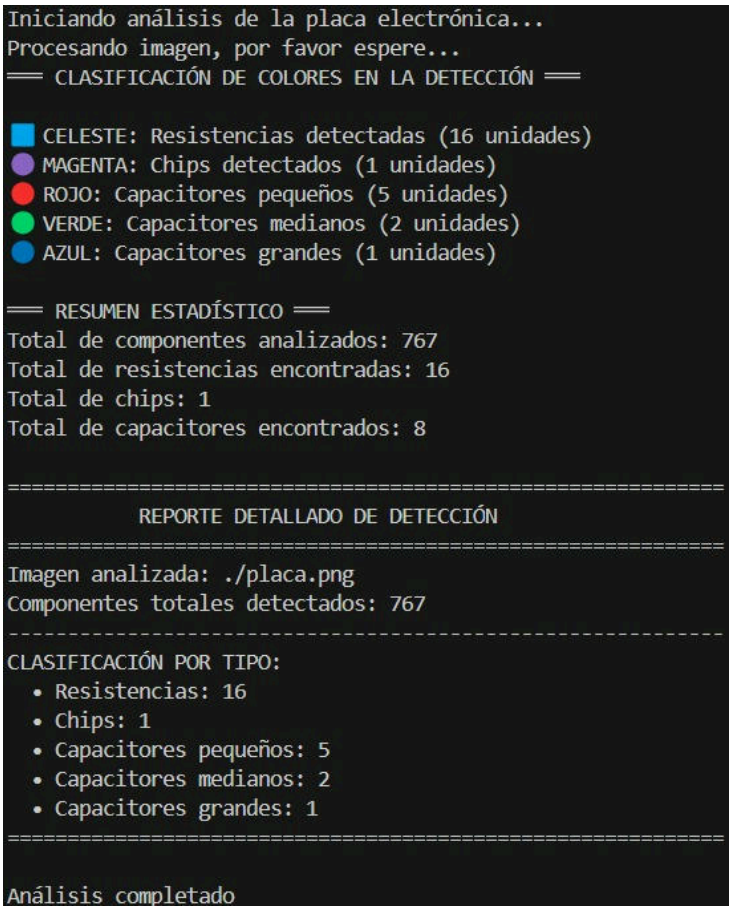
• Capacitores Grandes: Si el radio es igual o mayor a 150 píxeles. Se dibujan en azul y se cuentan.

- Además de la forma, se verifica que el interior del círculo sea suficientemente brillante (valor promedio de gris mayor a 130) para ser considerado un capacitor.

◦ Resistencias: Para las componentes que no cumplen los criterios anteriores (no son chips por área muy grande ni capacitores por forma circular y brillo), se realiza un análisis de color dentro de su región.

- Se compara el color de los píxeles con un color objetivo definido en el script (ajustable) dentro de un umbral de diferencia.
- Si el porcentaje de píxeles cuyo color está cerca del color objetivo supera el 23%, la componente se clasifica como RESISTENCIA. Se dibuja un rectángulo cian alrededor de ellas y se cuenta el total de resistencias.

Una vez clasificadas las componentes, el script procede a dibujar los rectángulos o círculos de los colores correspondientes y a añadir etiquetas de texto en la imagen final. Finalmente, imprime un resumen detallado en la consola con el conteo de cada tipo de componente detectada



Ejercicio 2 Parte I

El script sigue una serie de pasos para cumplir su objetivo de procesar imágenes de resistencias y aplicarles una transformación de perspectiva. El proceso general se orquesta principalmente en la función procesar_todas_las_imagenes:

1. Preparación de la Carpeta de Salida: El script verifica si existe una carpeta llamada “imagenes_out”. Si esta carpeta no existe, el script la crea. Se imprime un mensaje confirmando la creación de la carpeta.
2. Búsqueda de Imágenes de Entrada: El script busca todos los archivos con extensión .jpg dentro de una carpeta específica llamada “imagenes”. Utiliza el patrón de búsqueda “imagenes/*.jpg”.
3. Verificación de Imágenes Encontradas: Comprueba si se encontraron archivos .jpg en la carpeta de entrada. Si no se encuentra ninguna imagen, imprime un mensaje informándolo y el proceso termina.
4. Iteración sobre las Imágenes: Si se encontraron imágenes, el script itera a través de cada ruta de archivo encontrada. Se imprime un mensaje indicando cuántas imágenes se encontraron para procesar.
5. Procesamiento Individual de Cada Imagen: Para cada imagen encontrada, se extrae el nombre del archivo sin la extensión. Luego, se llama a la función transformar_resistencia para procesar esa imagen específica. Se imprime un mensaje indicando qué archivo se está procesando.

Dentro de la función transformar_resistencia, se realizan los siguientes pasos para procesar una sola imagen:

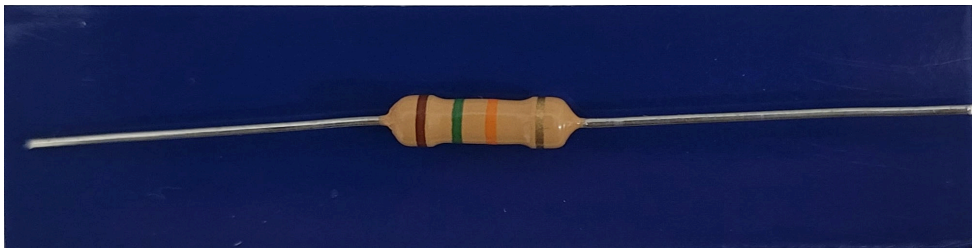
6. Carga de la Imagen: La imagen se lee desde la ruta proporcionada utilizando cv2.imread().
7. Verificación de Carga: Se verifica si la imagen se cargó correctamente. Si no se pudo cargar (por ejemplo, si la ruta es inválida o el archivo está corrupto), se imprime un mensaje de error y la función retorna None, indicando que esta imagen no se pudo procesar.
8. Conversión a Espacio de Color HSV: La imagen cargada se convierte del espacio de color BGR a HSV. Esto se hace para facilitar la detección del color azul.

9. Creación de Máscara Azul: Se define un rango para el color azul en el espacio HSV (lower_blue y upper_blue). Luego, se crea una máscara que contiene solo las regiones de la imagen que caen dentro de ese rango de color azul.
10. Búsqueda de Contornos: Se encuentran los contornos en la máscara azul generada.
11. Verificación de Contornos: Se verifica si se encontraron contornos. Si no se encuentra ningún contorno azul, se imprime un mensaje de error para esa imagen y la función retorna None.
12. Identificación del Contorno Principal: Si se encontraron contornos, se selecciona el contorno más grande asumiendo que corresponde al cuadrilátero azul que se desea transformar.
13. Aproximación del Contorno: El contorno más grande se aproxima a un polígono para identificar sus esquinas.
14. Obtención de las 4 Esquinas: Se obtienen los puntos de las esquinas del polígono aproximado, asegurándose de tener exactamente 4 puntos. Si la aproximación inicial tiene más de 4 puntos, se utiliza cv2.convexHull y cv2.approxPolyDP nuevamente para intentar obtener 4 puntos que formen el área convexa.
15. Definición de Puntos de Destino: Se definen los 4 puntos de destino para la transformación de perspectiva. Estos puntos forman un rectángulo de dimensiones específicas (2000x500 píxeles), que será el tamaño y la forma final de la imagen transformada.
16. Ordenamiento de Puntos de Origen: Los 4 puntos de las esquinas encontrados en la imagen original (src_points) se ordenan en sentido horario utilizando la función auxiliar ordenar_puntos_clockwise. Esta función calcula el centroide de los puntos, los ordena por ángulo alrededor del centroide, y ajusta el orden para que el primer punto sea el «superior izquierdo».
17. Cálculo de la Matriz de Transformación: Se calcula la matriz de transformación de perspectiva utilizando los puntos de origen ordenados y los puntos de destino definidos. Esta matriz contiene la información necesaria para «aplanar» la región del cuadrilátero en la imagen original a la forma del rectángulo de destino.
18. Aplicación de la Transformación: Se aplica la matriz de transformación de perspectiva a la imagen original para obtener la imagen resultante, que es la región del cuadrilátero azul «aplanada» al tamaño de 2000x500 píxeles.

19. Retorno del Resultado: La imagen transformada (o None si hubo un error) se retorna.

Volviendo a la función procesar_todas_las_imagenes después de la llamada a transformar_resistencia:

20. Manejo del Resultado de Procesamiento: Se verifica si la imagen transformada retornada no es None.
21. Guardado de la Imagen Transformada (Si tuvo éxito): Si la imagen transformada es válida (no None), se construye la ruta completa para guardar el archivo de salida dentro de la carpeta “imagenes_out”, usando el nombre del archivo original con el sufijo _out.jpg. La imagen transformada se guarda en esta ruta utilizando cv2.imwrite(). Se verifica si la operación de guardado fue exitosa.
22. Conteo de Resultados: Se incrementa el contador de imágenes procesadas exitosamente si la imagen se transformó y guardó correctamente. Se incrementa el contador de imágenes fallidas si transformar_resistencia retornó None (fallo al cargar la imagen, no encontró contornos, etc.) o si falló el guardado de la imagen transformada.



23. Resumen Final: Una vez que se han iterado y procesado (o intentado procesar) todas las imágenes encontradas, el script imprime un resumen que muestra cuántas imágenes fueron procesadas exitosamente, cuántas fallaron y el total de imágenes encontradas inicialmente

Ejercicio 2 Parte II

Los pasos clave en el procesamiento de una imagen de resistencia para identificar su valor son los siguientes:

1. Cargar y Preprocesar la Imagen: La imagen de entrada (en formato BGR) es cargada. Se crea una copia de la imagen original. La imagen se suaviza aplicando un filtro Gaussiano (cv2.GaussianBlur). Luego, se convierte al espacio de color HSV (cv2.cvtColor) para facilitar la segmentación de colores.

2. Crear Máscara del Fondo (Cuerpo Amarillo): Utilizando el espacio de color HSV, se define un rango específico para el color amarillo (lower_yellow, upper_yellow). Se crea una máscara binaria (mask_fondo) donde los píxeles dentro de este rango amarillo se marcan (generalmente blanco) y el resto se marcan (generalmente negro). El objetivo es aislar el cuerpo de la resistencia.
3. Crear Máscara de las Bandas: Se invierte la máscara del fondo (mask_fondo) usando una operación bit a bit (cv2.bitwise_not) para obtener una máscara (mask_bandas) que represente las bandas y cualquier otra parte de la imagen que no sea el cuerpo amarillo.
4. Limpiar la Máscara con Morfología: Se aplican operaciones morfológicas (cv2.morphologyEx) a la máscara de las bandas para limpiarla, eliminando ruido y cerrando posibles huecos dentro de las regiones de las bandas. Específicamente, se aplica un MORPH_CLOSE seguido de un MORPH_OPEN con un kernel de (17, 5). Esto produce una máscara limpia (mask_bandas_limpia) más adecuada para la detección de contornos.
5. Preparar Imagen para Contornos: Se invierte la máscara limpia (mask_bandas_limpia) nuevamente (cv2.bitwise_not) para obtener una imagen (img_invertida) donde las bandas ahora aparecen como objetos detectables por el algoritmo de contornos.
6. Detectar Contornos Iniciales: Se utiliza la función cv2.findContours en la imagen invertida (img_invertida) para encontrar los contornos externos (cv2.RETR_EXTERNAL) de las bandas. Se usa una aproximación simple (cv2.CHAIN_APPROX_SIMPLE).
7. Corregir Orientación (Rotar si es Necesario): Se evalúa si el contorno más a la izquierda (después de ordenar por coordenada X) tiene el área máxima. Si es así, se considera que la resistencia está al revés y se espeja horizontalmente (cv2.flip) la imagen original y la imagen HSV. Si se espeja, se repite todo el proceso de preprocesamiento, enmascaramiento y detección de contornos en la imagen espejada para obtener los contornos finales ordenados correctamente. Si no es necesario espejar, se usan los contornos detectados inicialmente.
8. Dibujar Contornos y Separadores: Se dibujan los contornos finales detectados sobre una copia de la imagen final (posiblemente espejada). También se dibujan líneas separadoras: una línea horizontal a la altura media del primer contorno y líneas

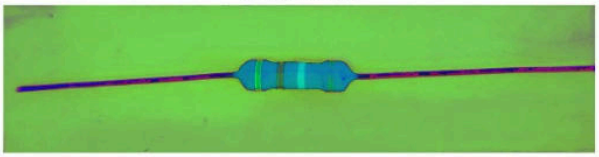
verticales a medio camino entre contornos consecutivos para visualizar las regiones de análisis de color.

9. Detectar Colores en Regiones de Intersección: Para cada par de contornos consecutivos, se calcula un punto medio entre ellos. Se toma una pequeña región de píxeles (ancho=4, alto=4) alrededor de este punto medio en la imagen HSV. Se aplica el algoritmo K-Means (con K=2) a los valores HSV de estos píxeles para encontrar el color dominante de esa región. El color dominante (en formato HSV) se clasifica en un nombre de color predefinido (como «Rojo», «Azul», etc.) utilizando la función get_color_name. Los nombres de los colores detectados se almacenan en una lista (colores_detectados) en orden. Se dibujan flechas y los nombres de los colores detectados en la imagen resultante.
10. Calcular el Valor de Resistencia: La lista de colores detectados (colores_detectados) se utiliza para calcular el valor de la resistencia. Se toman los primeros tres colores de la lista. Se buscan los valores numéricos (dígitos) de los dos primeros colores en el diccionario VALORES_COLORES y el factor multiplicador del tercer color en el diccionario MULTIPLICADORES. El valor de la resistencia se calcula usando la fórmula (digito1 × 10 + digito2) × multiplicador.
11. Formatear y Presentar Resultados: El valor numérico calculado se formatea para ser más legible, usando unidades como Ω, kΩ, o MΩ según corresponda. Se crea una imagen compuesta que muestra visualmente varias etapas del procesamiento (original, HSV, máscaras, contornos, separadores, colores detectados). Esta imagen compuesta incluye texto resumen con el número de contornos encontrados, los colores detectados en orden, y el valor de resistencia calculado. Finalmente, esta imagen compuesta se guarda como un archivo PNG, y un resumen textual de los resultados (nombre del archivo, colores detectados, valor de resistencia, número de contornos) se escribe en un archivo de texto de resumen general (resumen_resistencias.txt)

Procesamiento de R1_a_out

1. Imagen Original

2. Imagen HSV



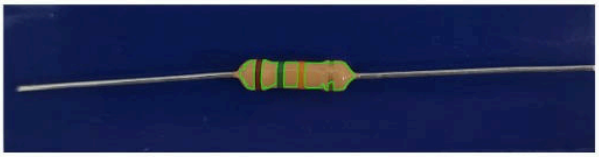
3. Máscara del Fondo (Amarillo)

4. Máscara de las Bandas

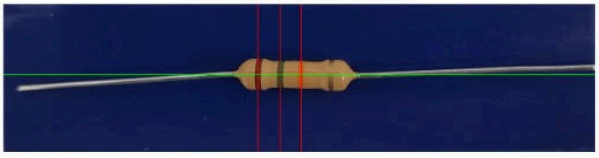
5. Máscara Limpia (Morfología)

6. Imagen Invertida

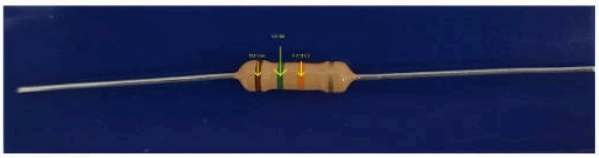
7. Contornos Detectados



8. Separadores entre Bandas



9. Colores Detectados



Contornos encontrados: 4
Colores detectados: Marron → Verde → Naranja
Valor calculado: 15.0kΩ