

EJERCICIO 1

Procesamiento Digital de Imágenes - UNR TUIA

1. Introducción

Este informe presenta la implementación de un sistema automatizado para la detección y clasificación de componentes electrónicos en placas de circuito impreso (PCB) mediante técnicas de procesamiento digital de imágenes. El sistema desarrollado en Python utiliza las librerías OpenCV, NumPy y Matplotlib para identificar y clasificar tres tipos principales de componentes: resistencias eléctricas, capacitores electrolíticos y chips.

2. Metodología

2.1 Arquitectura del Sistema

El sistema se implementó mediante la clase `PCBComponentAnalyzer`, que encapsula todas las funcionalidades necesarias para el análisis completo de la placa PCB. La arquitectura modular permite un procesamiento secuencial y organizado de las diferentes etapas del análisis.

2.2 Pipeline de Procesamiento

El procesamiento de la imagen sigue una metodología estructurada en cuatro etapas principales:

1. Preprocesamiento de imagen
2. Detección de bordes y segmentación
3. Clasificación de componentes
4. Visualización de resultados

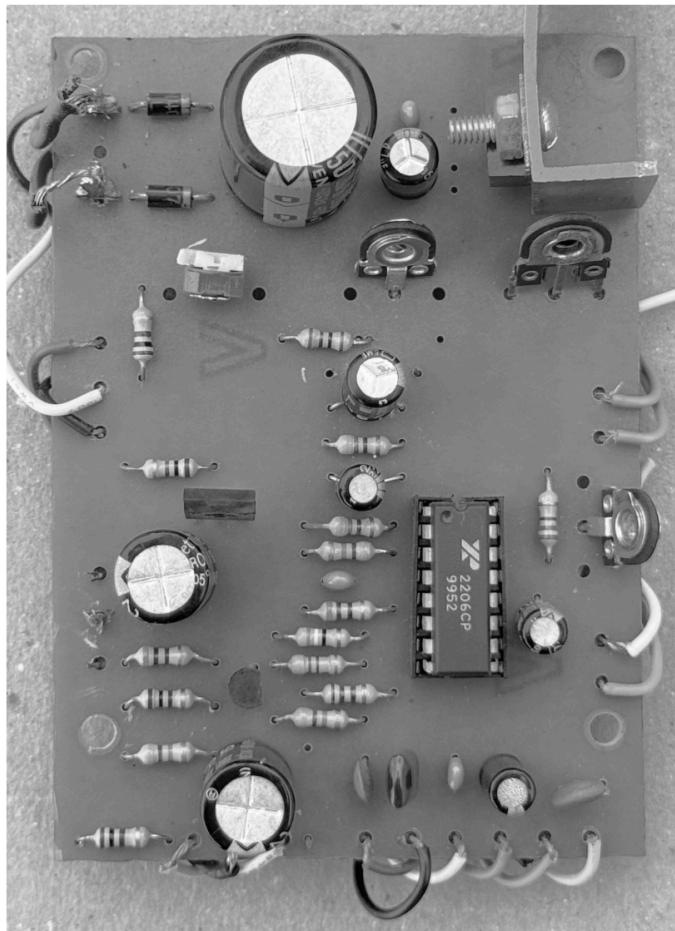
3. Implementación Técnica

3.1 Preprocesamiento de Imagen

Conversión a Escala de Grises

```
self.gray_image = cv2.cvtColor(self.original_image, cv2.COLOR_BGR2GRAY)
```

Imagen en Escala de Grises

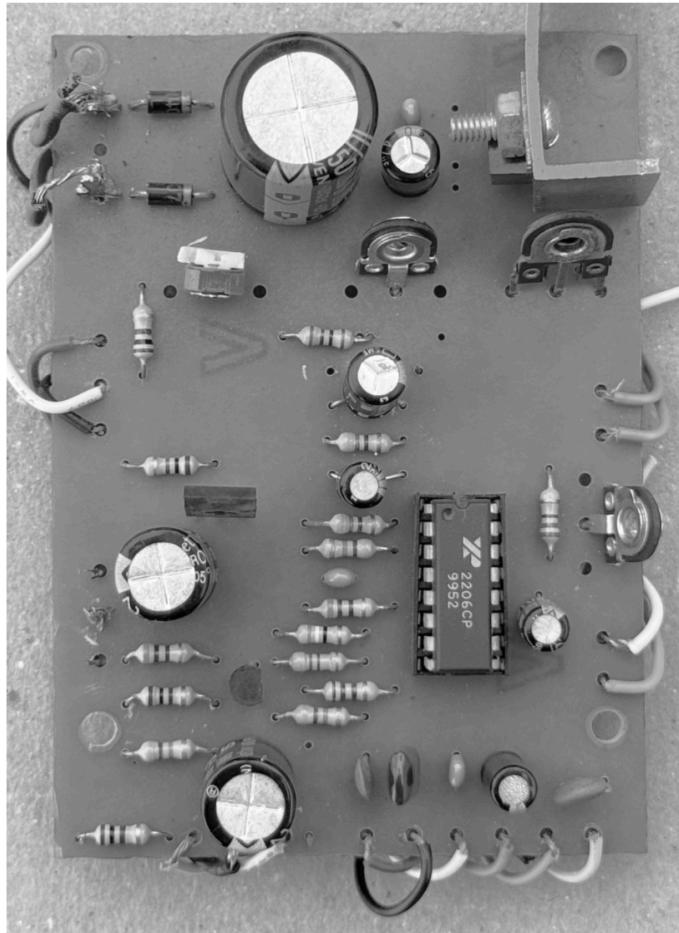


La imagen original se convierte a escala de grises para simplificar el procesamiento posterior y reducir la complejidad computacional.

Filtrado Gaussiano

```
self.blurred_image = cv2.GaussianBlur(self.gray_image, (11, 11), 1)
```

Imagen Suavizada (Filtro Gaussiano)



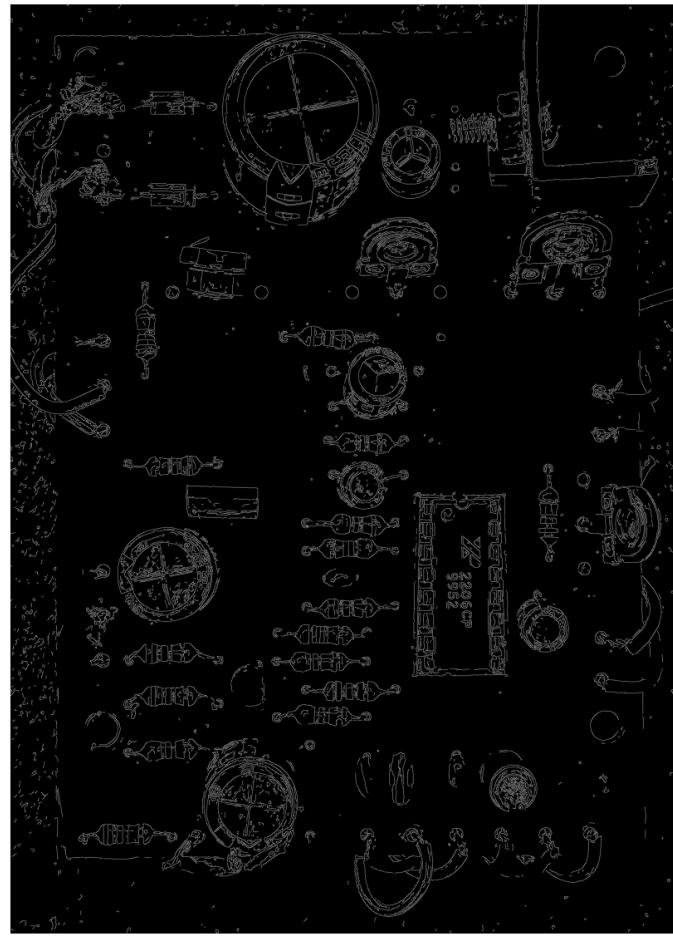
Se aplica un filtro Gaussiano con kernel de 11x11 píxeles y desviación estándar $\sigma=1$ para reducir el ruido y suavizar la imagen, preparándola para la detección de bordes.

3.2 Detección de Bordes

Algoritmo Canny

```
self.edge_image = cv2.Canny(self.blurred_image, low_thresh=50,  
high_thresh=80)
```

Detección de Bordes (Canny)

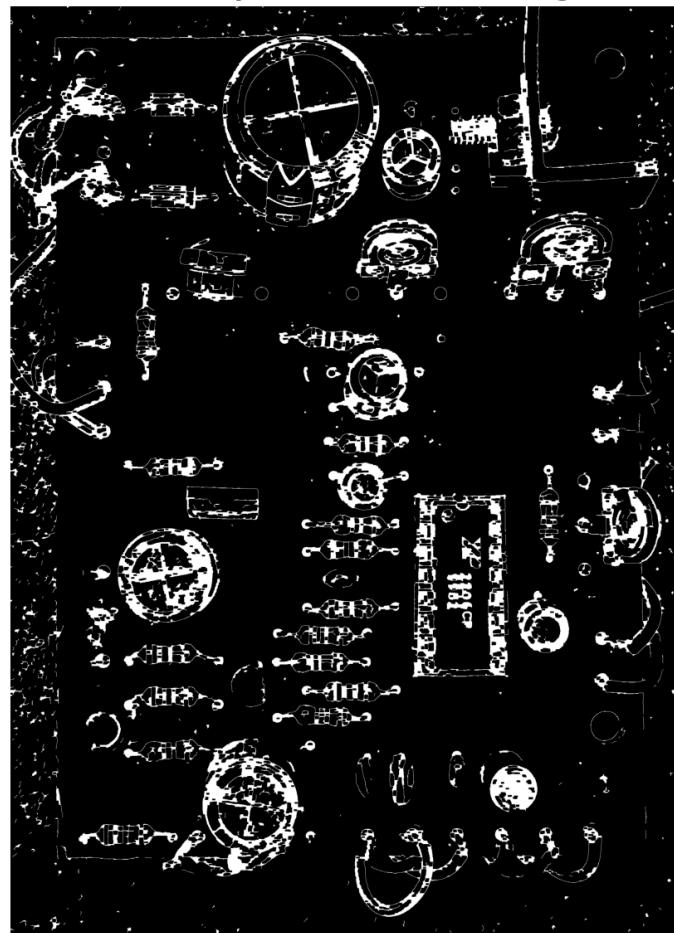


Se implementa el detector de bordes Canny con umbrales optimizados (50-80) para componentes electrónicos, permitiendo la identificación precisa de contornos.

Mejora de Bordes Mediante Operaciones Morfológicas

```
closing_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (10, 5))
closed_edges = cv2.morphologyEx(self.edge_image, cv2.MORPH_CLOSE,
closing_kernel)
```

Bordes Después del Cierre Morfológico



Se aplican operaciones de cierre morfológico para conectar bordes fragmentados y dilatación para engrosar líneas finas, mejorando la conectividad de los contornos detectados.

3.3 Segmentación de Componentes

Análisis de Componentes Conectados

```
num_components, labels, stats, centroids =  
cv2.connectedComponentsWithStats(  
self.processed_edges, connectivity=8  
)
```

Componentes Conectados Detectados (767)



Se utiliza conectividad-8 para identificar regiones conectadas en la imagen binaria. Los componentes se filtran por área mínima (3200 píxeles) para eliminar ruido y elementos irrelevantes.

3.4 Clasificación de Componentes

3.4.1 Detección de Resistencias

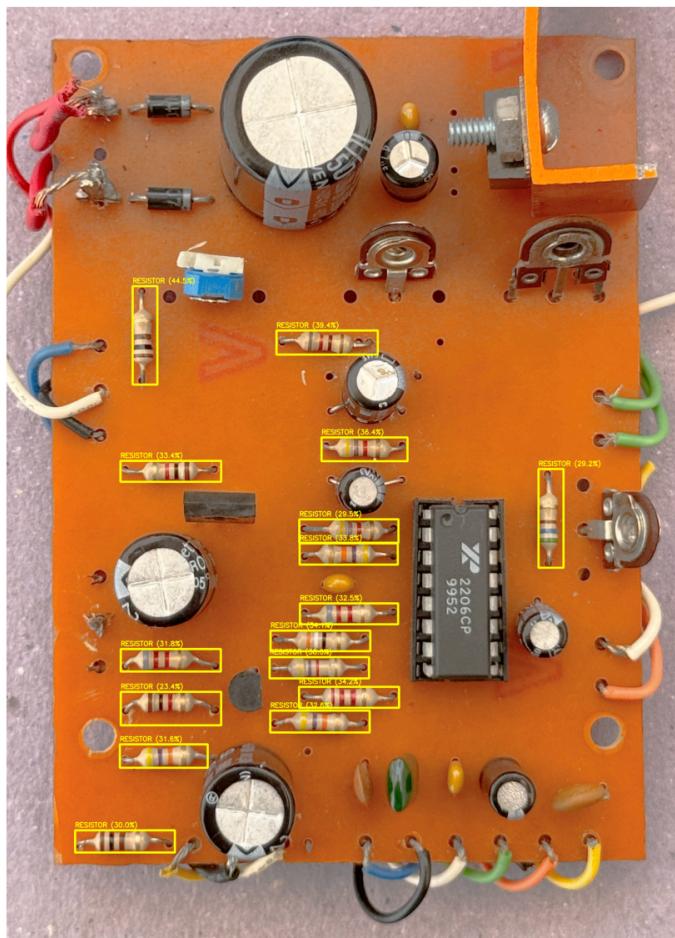
Las resistencias se identifican mediante criterios geométricos y de color:

- **Criterio geométrico:** Relación de aspecto ≥ 1.7 (forma alargada)
- **Criterio de área:** Área $\leq 10,000$ píxeles
- **Criterio de color:** Detección de tonos marrones característicos

```
if aspect_ratio >= 1.7 and area <= 10000:  
    target_color = np.array([200, 160, 100], dtype=np.uint8)
```

tolerance = 50

Resistencias Detectadas - Total: 16



El algoritmo analiza el porcentaje de píxeles que coinciden con el rango de color marrón típico de las resistencias (umbral: 23%).

3.4.2 Detección de Capacitores

Los capacitores se detectan mediante la Transformada de Hough para círculos:

```
circles = cv2.HoughCircles(  
    blurred_roi,  
    cv2.HOUGH_GRADIENT,  
    dp=1,  
    minDist=250,  
    param1=70,
```

```
param2=50,  
minRadius=25,  
maxRadius=200
```

)

Parámetros optimizados:

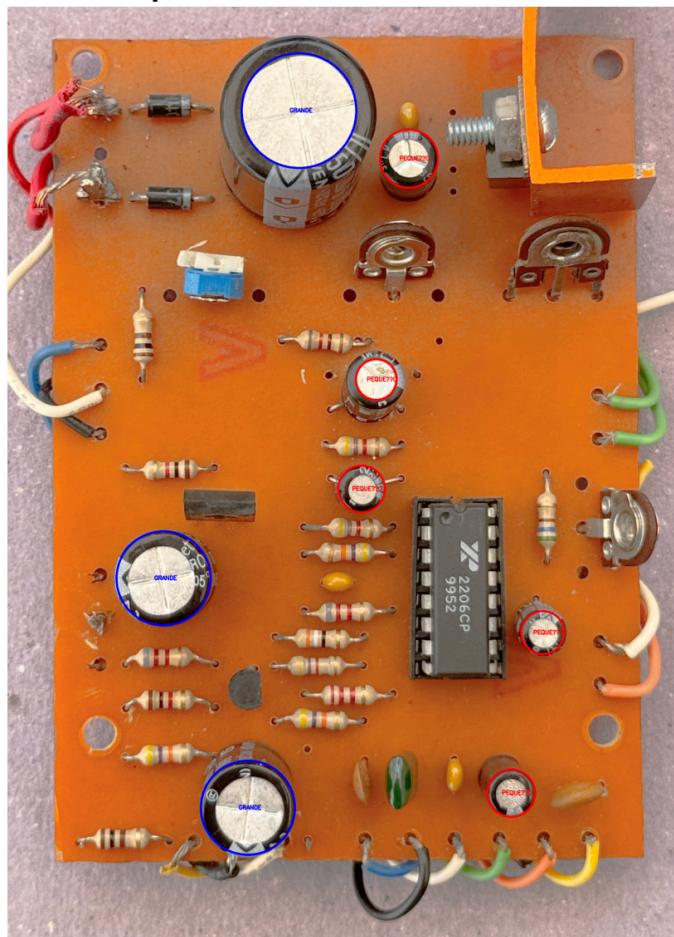
- Radio mínimo: 25 píxeles
- Radio máximo: 200 píxeles
- Distancia mínima entre centros: 250 píxeles
- Análisis de brillo interior (umbral: 130) para validar detecciones

3.4.3 Clasificación de Capacitores por Tamaño

Los capacitores se clasifican en tres categorías según su radio:

- **Pequeños**: radio < 80 píxeles (marcados en rojo)
- **Medianos**: $80 \leq \text{radio} < 180$ píxeles (marcados en verde)
- **Grandes**: radio ≥ 180 píxeles (marcados en azul)

Capacitores Detectados - Total: 8

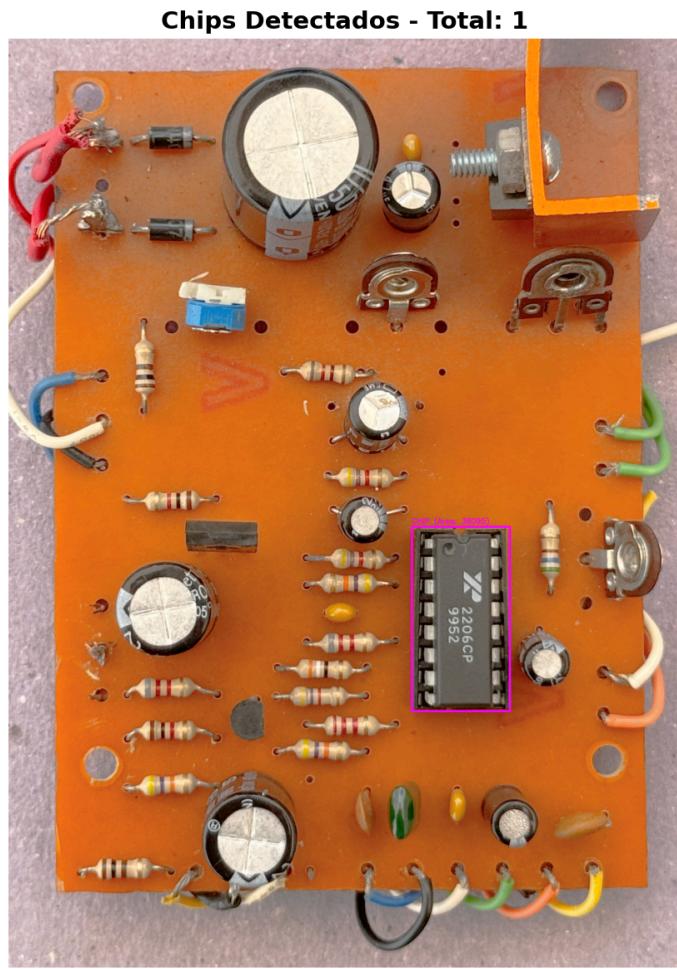


3.4.4 Detección de Chips

Los chips se identifican únicamente por criterio de área:

- **Criterio:** Área > 38,000 píxeles

```
if area > 38000:  
    chips.append(component)
```



4. Resultados

4.1 Segmentación General

El método `analyze_and_segment()` produce una imagen de salida que muestra todos los componentes detectados con marcadores de colores distintivos:

- **Resistencias:** Rectángulos amarillos con etiqueta "RESISTOR"
- **Capacitores:** Círculos de colores según tamaño
- **Chips:** Rectángulos magenta con etiqueta "CHIP"

[INSERTAR FIGURA: Imagen final de segmentación completa con todos los componentes marcados]

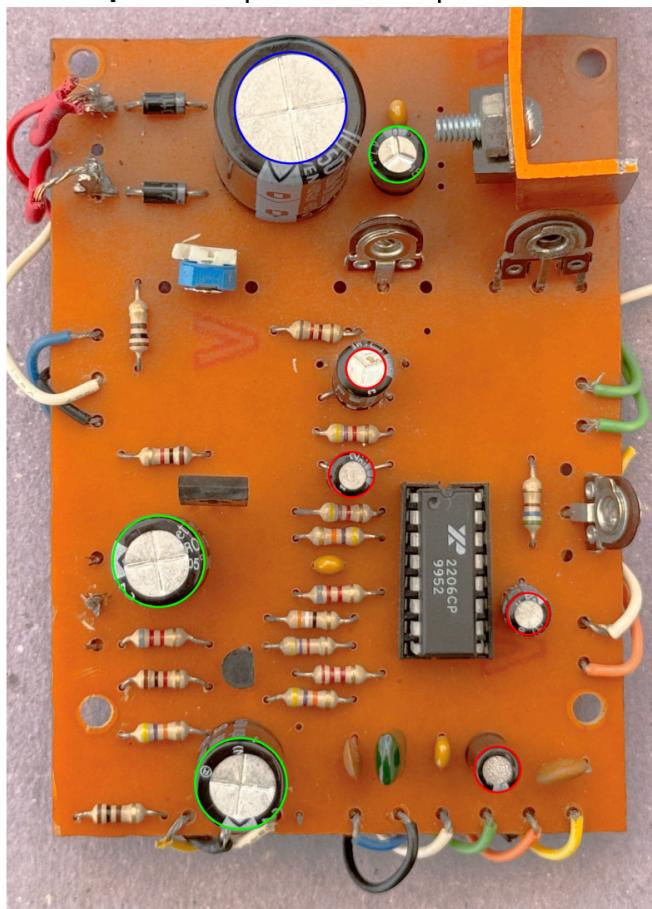
4.2 Clasificación de Capacitores

El sistema genera estadísticas detalladas de la distribución de capacitores:

```
size_distribution = {"PEQUEÑO": 0, "MEDIANO": 0, "GRANDE": 0}
```

CLASIFICACIÓN DE CAPACITORES POR TAMAÑO

Pequeños: 4 | Medianos: 3 | Grandes: 1



5. Validación y Robustez

5.1 Filtrado por Área Mínima

El umbral de 3200 píxeles elimina efectivamente el ruido y pequeños artefactos, manteniendo solo componentes significativos.

5.2 Análisis Multi-criterio

La combinación de criterios geométricos, de área y de color proporciona mayor precisión en la clasificación, reduciendo falsos positivos.

5.3 Operaciones Morfológicas

Las operaciones de cierre y dilatación mejoran la conectividad de los contornos, especialmente importante para componentes con bordes fragmentados.

6. Limitaciones y Consideraciones

6.1 Dependencia de Iluminación

El sistema puede verse afectado por variaciones significativas en las condiciones de iluminación, especialmente en la detección de colores de resistencias.

6.2 Resolución de Imagen

Los umbrales de área están calibrados para una resolución específica y pueden requerir ajustes para imágenes de diferentes escalas.

6.3 Solapamiento de Componentes

El algoritmo actual no maneja casos de solapamiento parcial entre componentes.

7. Conclusiones

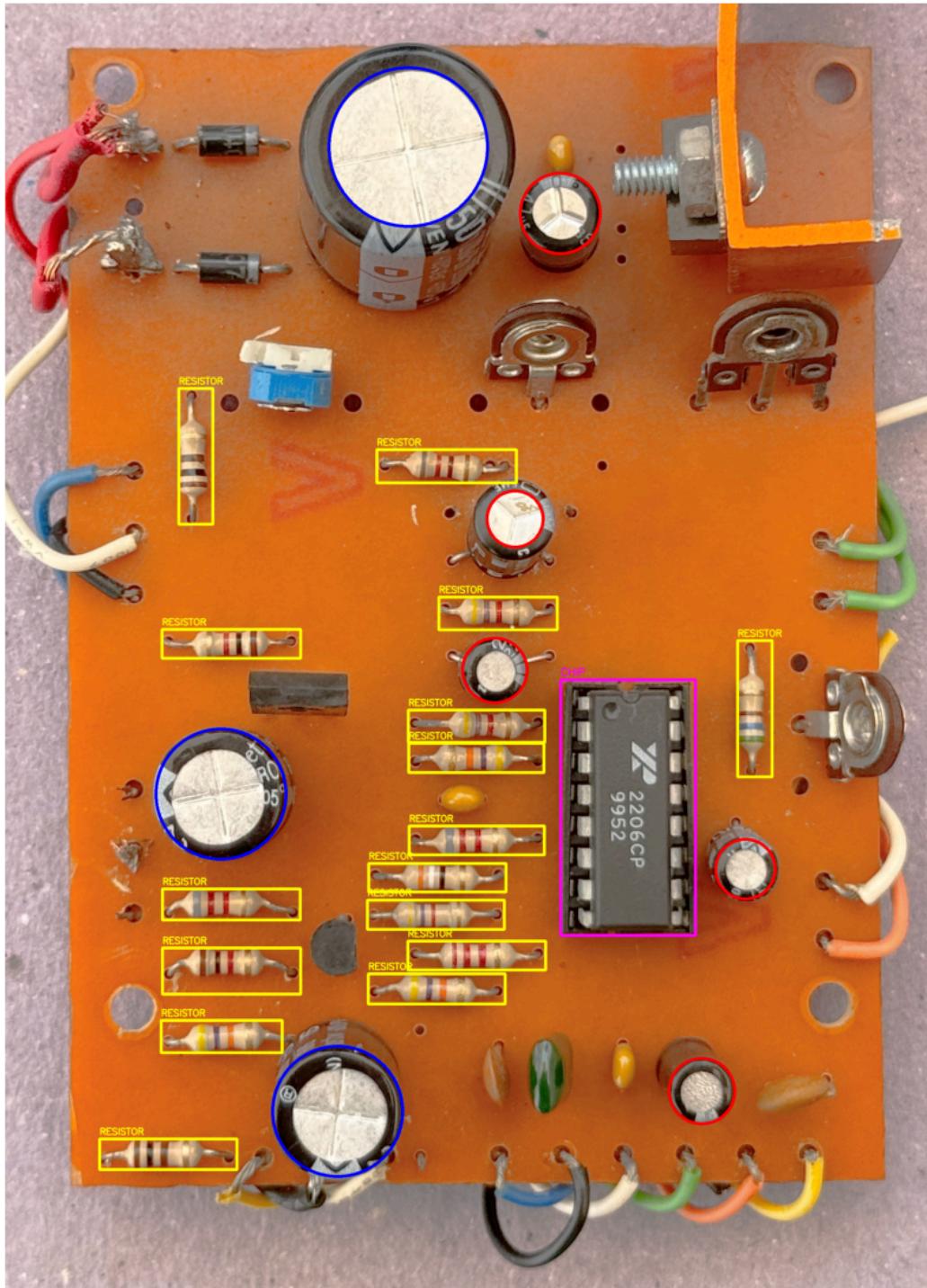
El sistema desarrollado demuestra efectividad en la detección y clasificación automatizada de componentes electrónicos en placas PCB. La metodología implementada combina técnicas clásicas de procesamiento de imágenes con criterios específicos del dominio, logrando:

1. **Segmentación precisa** de tres tipos de componentes electrónicos
2. **Clasificación automática** de capacitores por tamaño
3. **Conteo exacto** de resistencias eléctricas
4. **Visualización clara** de resultados con marcadores distintivos

La arquitectura modular facilita futuras extensiones del sistema para incluir otros tipos de componentes o mejorar la precisión de detección mediante técnicas más avanzadas.

SEGMENTACIÓN DE COMPONENTES

Resistores: 16 | Capacitores: 8 | Chips: 1



Problema 2: Identificación de Resistencias Eléctricas

Descripción del Problema

El objetivo fue desarrollar un sistema automático para identificar y calcular el valor de resistencias eléctricas a partir de sus códigos de colores. El proceso se dividió en tres etapas principales:

1. **Corrección de perspectiva:** Transformar imágenes de resistencias en diferentes ángulos a una vista superior normalizada
2. **Detección de bandas:** Localizar automáticamente las bandas de colores en la resistencia
3. **Identificación de colores:** Clasificar cada banda según su color y calcular el valor de resistencia

Metodología Implementada

A) Preprocesamiento y Corrección de Perspectiva

Objetivo: Convertir imágenes de resistencias tomadas desde diferentes ángulos a una vista superior estandarizada.

Proceso implementado:

1. **Segmentación del rectángulo azul:** Se utilizó segmentación por color en espacio HSV para detectar el fondo azul que contiene la resistencia.

```
# Rango HSV calibrado para fondo azul
azul_bajo = np.array([100, 50, 50])
azul_alto = np.array([130, 255, 255])
mascara_azul = cv2.inRange(img_hsv, azul_bajo, azul_alto)
```

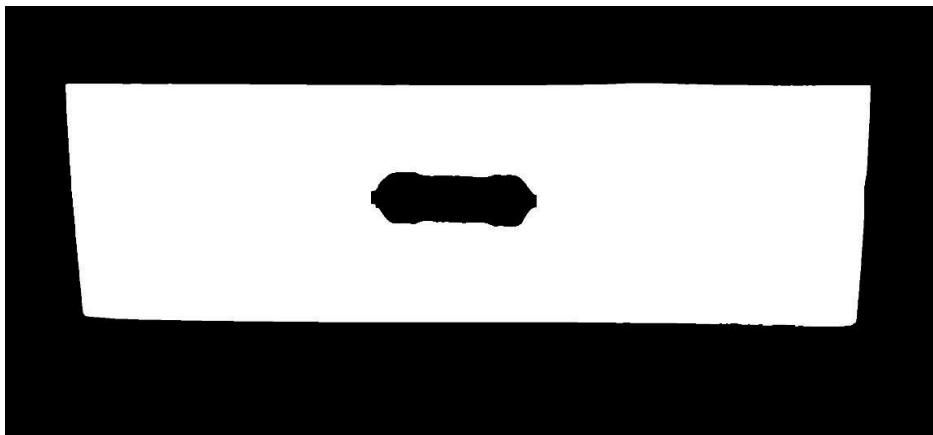


Figura 1: Máscara de detección del rectángulo azul

2. **Operaciones morfológicas:** Se aplicaron operaciones de apertura y cierre para limpiar la máscara y conectar componentes fragmentados.
3. **Detección de contornos y aproximación:** Se encontró el contorno principal y se approximó a un polígono de 4 vértices usando `cv2.approxPolyDP()`.

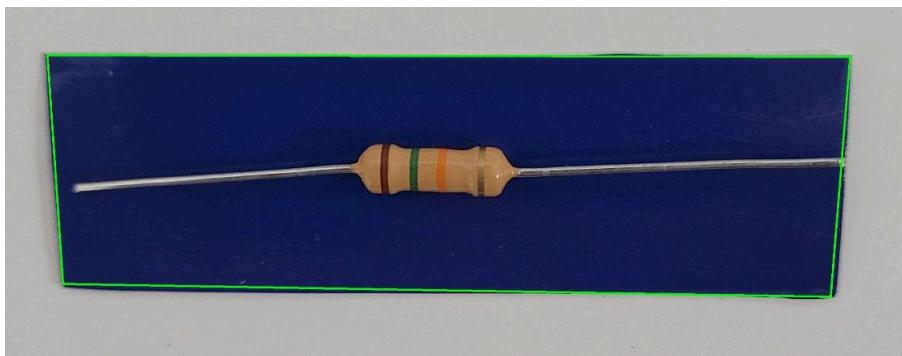


Figura 2: Contorno del rectángulo detectado con los 4 puntos de referencia

4. **Transformación de perspectiva:** Se aplicó `cv2.getPerspectiveTransform` para obtener la vista superior normalizada.



Figura 3: Resultado de la transformación a vista superior

B) Detección de Bandas de Colores

Técnica principal: Análisis de gradiente horizontal mediante filtro Sobel-X para detectar transiciones verticales (bordes de bandas).

Proceso detallado:

1. **Eliminación del fondo azul:** Se segmentó la resistencia del fondo azul y se reemplazó el fondo con blanco para maximizar el contraste.
2. **Definición de región de interés:** Se calculó una región central excluyendo los terminales metálicos mediante márgenes proporcionales (14% horizontal, 15% vertical).
3. **Aplicación de filtro Sobel-X:** Se detectaron bordes verticales que corresponden a las transiciones entre bandas de colores.

```
# Filtro Sobel horizontal para detectar bordes verticales
sobel_x = cv2.Sobel(region_bandas, cv2.CV_64F, 1, 0, ksize=1)
```

4. **Proyección vertical y suavizado:** Se sumaron las columnas para obtener un perfil 1D de intensidad de gradiente, luego se suavizó con convolución gaussiana.
5. **Detección de picos:** Se identificaron máximos locales que superaran un umbral de intensidad calibrado.

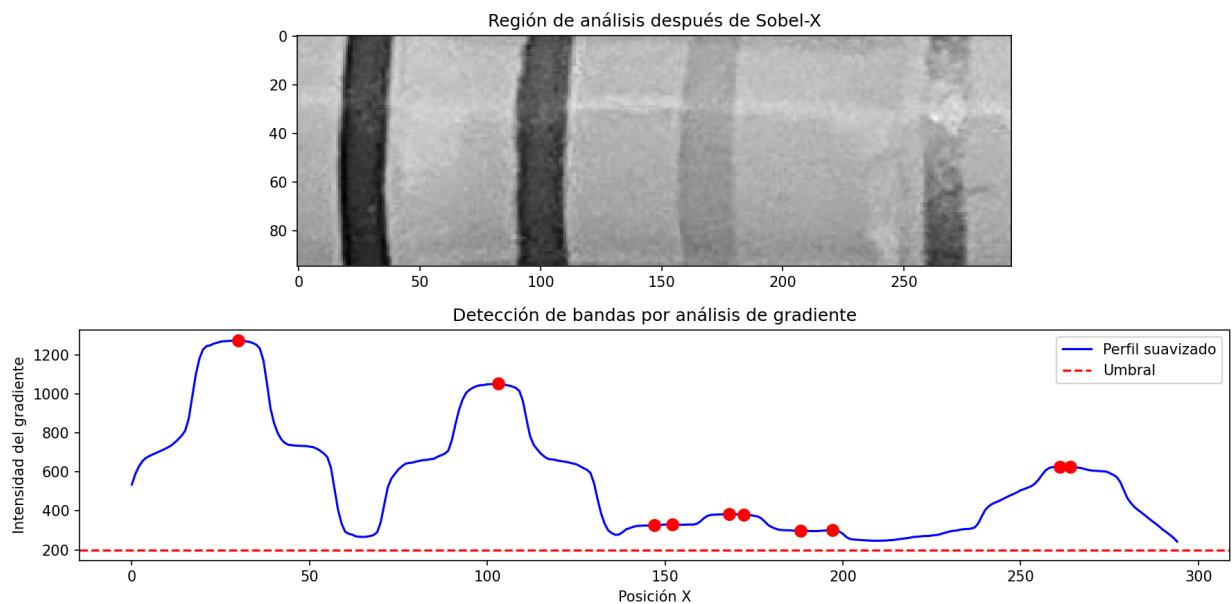


Figura 4: Análisis de gradiente mostrando la detección de bandas

6. **Supresión no-máxima:** Se eliminaron picos demasiado cercanos usando distancia mínima empírica.

C) Separación de Bandas de Valor y Tolerancia

Algoritmo de separación: Se analizaron las distancias entre bandas consecutivas para identificar gaps significativos que indican la separación de la banda de tolerancia.

```
# Detección de gap significativo

if distancias[-1] > distancia_promedio * (1 + self.ratio_distancia_tolerancia):

    posiciones_tolerancia = [posiciones_bandas[-1]]

    posiciones_valor = posiciones_bandas[:-1]
```

D) Identificación de Colores

Método: Clasificación por máxima verosimilitud usando rangos HSV calibrados para cada color estándar.

Proceso:

1. **Definición de rangos HSV:** Se establecieron rangos específicos para cada color, manejando casos especiales como el rojo que requiere dos rangos debido a la discontinuidad en 0°/360°.

```
'Rojo': {
    'hsv_range': [
        (np.array([0, 150, 100]), np.array([5, 255, 255])),
        (np.array([175, 150, 100]), np.array([180, 255, 255]))
    ]
}
```

2. **Análisis por ventanas:** Para cada banda detectada, se extrajo una ventana vertical de análisis y se contaron los píxeles que coincidían con cada rango de color.
3. **Clasificación:** Se asignó el color con mayor densidad de píxeles coincidentes.

E) Cálculo del Valor de Resistencia

Se implementó la fórmula estándar: **Valor = (D1 × 10 + D2) × M**

Donde:

- D1, D2: Primeros dos dígitos (bandas 1 y 2)
- M: Multiplicador (banda 3)

```
valor_base = (primer_digito * 10 + segundo_digito) * multiplicador
```

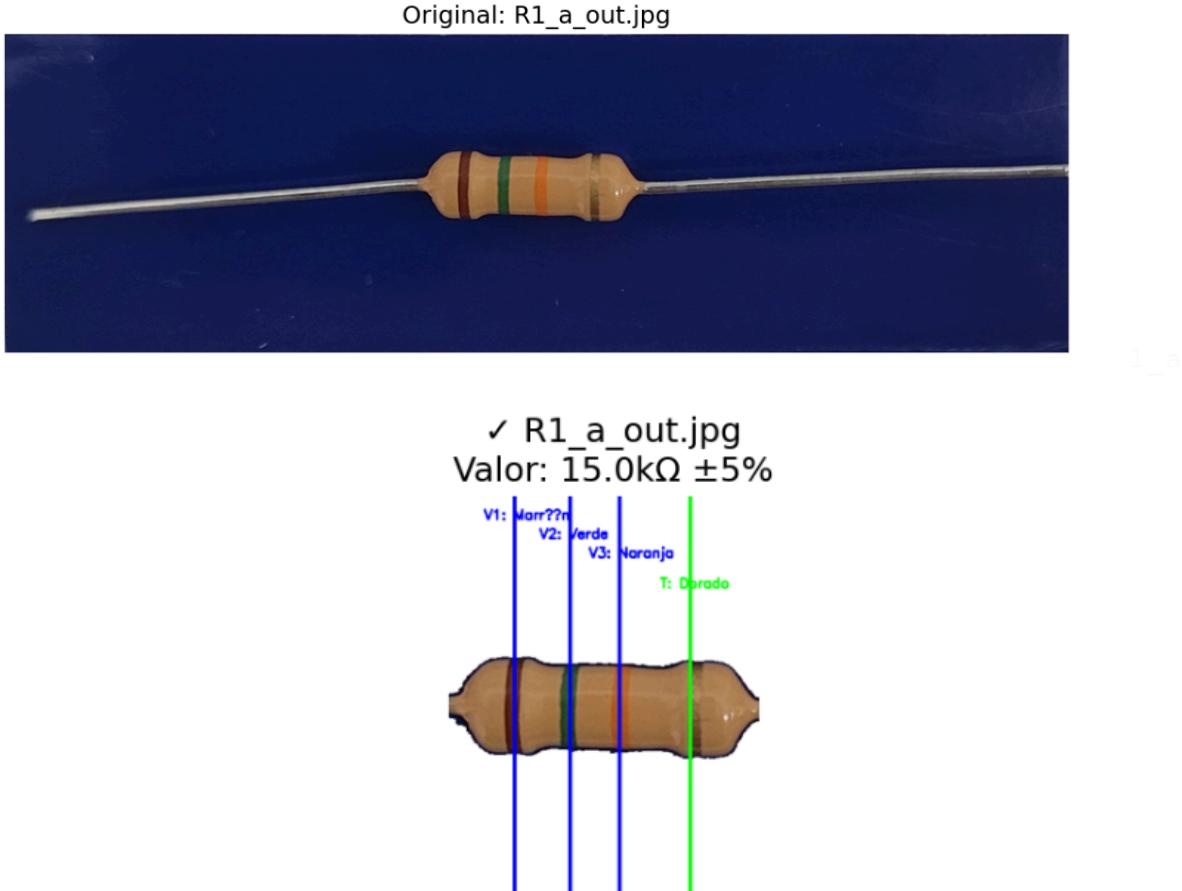


Figura 5: Resultado final mostrando bandas detectadas y valor calculado

Resultados Obtenidos

El sistema procesó exitosamente las 10 resistencias del dataset, detectando automáticamente:

- **Bandas de valor:** 3 bandas principales por resistencia
- **Banda de tolerancia:** Separada automáticamente del grupo principal
- **Valores calculados:** Conversión automática a unidades apropiadas (Ω , $k\Omega$, $M\Omega$)

Parámetros de Calibración

- **Umbral de intensidad:** 200 (para detección de picos en gradiente)
- **Ratio de distancia para tolerancia:** 0.15 (15% adicional para detectar separación)
- **Ventana de análisis por banda:** 5 píxeles de ancho
- **Márgenes de ROI:** 14% horizontal, 15% vertical

Inconvenientes Encontrados y Soluciones

1. **Variaciones de iluminación:** Solucionado usando espacio HSV en lugar de RGB
2. **Ruido en detección de bordes:** Mitigado con suavizado gaussiano del perfil de intensidad
3. **Falsos positivos en detección:** Eliminados con supresión no-máxima y umbralización adaptativa
4. **Separación banda de tolerancia:** Resuelto con análisis estadístico de distancias inter-banda

Conclusiones

Se desarrolló exitosamente un sistema completo de visión por computadora capaz de:

- Normalizar imágenes de resistencias desde cualquier perspectiva
- Detectar automáticamente bandas de colores mediante análisis de gradientes
- Clasificar colores con alta precisión usando segmentación HSV
- Calcular valores de resistencia según estándares industriales

Técnicas clave utilizadas:

- Transformación de perspectiva con OpenCV
- Filtros Sobel para detección de bordes
- Segmentación por color en espacio HSV
- Análisis estadístico para separación de componentes
- Operaciones morfológicas para limpieza de máscaras