Assignment 1: Connected Component Labeling

Aditya Bhargava
UGA ID: 811235738

CSCI 8820: Computer Vision and Pattern Recognition
Department of Computer Science, University of Georgia
February 12, 2026

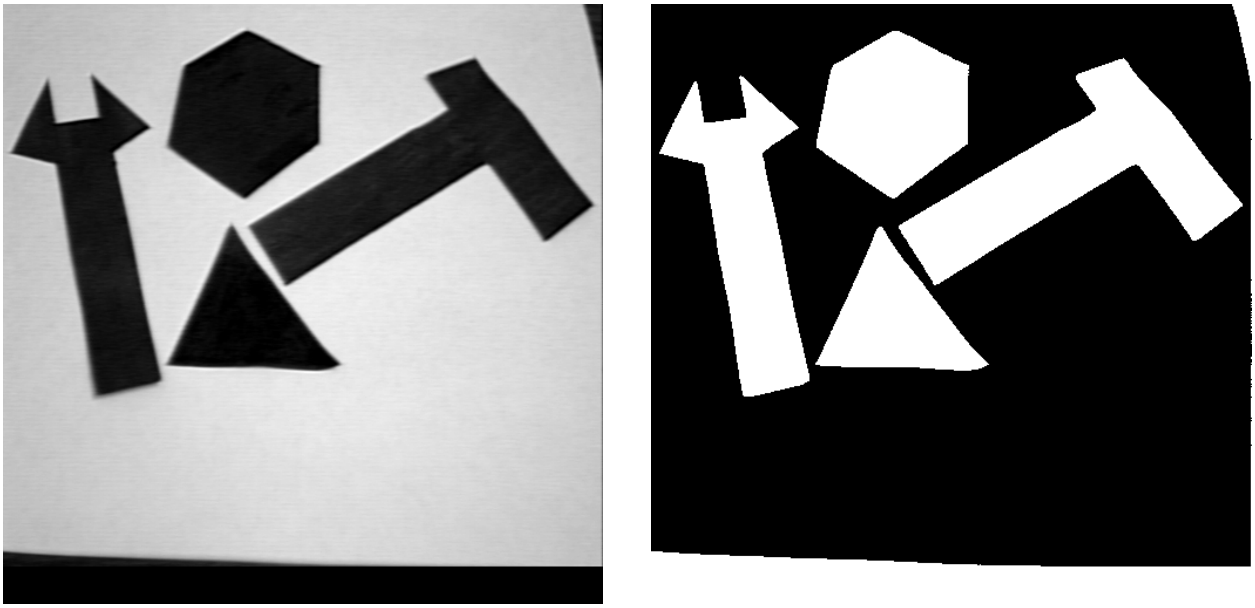
Contents

1 Introduction and Preprocessing

This report implements an iterative Connected Component Labeling (CCL) algorithm to segment and analyze objects in a grayscale image. The pipeline involves binary thresholding, labeling, and geometric feature extraction.

1.1 Thresholding and Binary Mask Generation

The input image B was converted to a binary image B_T using a fixed threshold $T = 128$. To ensure consistency, the mask was automatically inverted by $B_T = 1 - B_T$.



(a) Original Grayscale Image (B) (b) Thresholded Binary Image (with inversion) (B_T)

Figure 1: Preprocessing: Conversion from 8-bit grayscale to binary mask.

2 Methodology

2.1 Connected Component Labeling (CCL)

An iterative 4-connected CCL algorithm was implemented using a two-pass approach:

1. **First Pass:** Raster-scanning the image and assigning temporary labels based on the 4 connectivity of the top and left neighbors. Collisions were managed via a union-find data structure.

2. **Second Pass:** Resolution of equivalences to merge components and sequential re-labeling of the final IDs.

2.2 Feature Extraction and Mathematical Models

Geometric properties were derived from the zeroth-, first-, and second-order central moments.

- **Area and Centroid:** The area and centroid coordinates of a binary object $B(i, j)$ are defined as

$$A = \sum_{i=1}^{n} \sum_{j=1}^{m} B(i,j), \tag{1}$$

$$X_c = \frac{1}{A} \sum_{i=1}^{n} \sum_{j=1}^{m} jB(i,j), \tag{2}$$

$$Y_c = \frac{1}{A} \sum_{i=1}^{n} \sum_{j=1}^{m} iB(i,j), \tag{3}$$

where A denotes the object area, and (X_c, Y_c) represents the centroid location.

- **Orientation (θ):** The principal axis orientation is obtained by minimizing the second central moment:

$$\theta = \frac{1}{2} \tan^{-1}\left(\frac{b}{a-c}\right), \tag{4}$$

where a, b, and c are second-order central moments.

- **Second-Order Central Moments:** These moments are computed as

$$a = \sum_{i=1}^{n} \sum_{j=1}^{m} [X'(i,j)]^2 B(i,j), \tag{5}$$

$$b = 2 \sum_{i=1}^{n} \sum_{j=1}^{m} X'(i,j) Y'(i,j) B(i,j), \tag{6}$$

$$c = \sum_{i=1}^{n} \sum_{j=1}^{m} [Y'(i,j)]^2 B(i,j), \tag{7}$$

where $X'(i,j) = j - X_c$ and $Y'(i,j) = i - Y_c$ denote centroid-shifted coordinates.

- **Eccentricity:** The eccentricity of the object is computed from the eigenvalues of the covariance matrix:

$$\text{Eccentricity} = \sqrt{1 - \frac{I_{\min}}{I_{\max}}}, \tag{8}$$

where $I_{\max}$ and $I_{\min}$ are the major and minor principal moments, respectively.

Note: Consistency of the moment calculations was verified by checking that the invariant relation $|(a+c) - (I_{\max} + I_{\min})|$ remained close to zero. With a maximum absolute error of 6.0×10^{-8} across all tests.

- **Compactness:** Compactness, a measure of shape circularity, is defined as

$$\text{Compactness} = \frac{P^2}{A}, \tag{9}$$

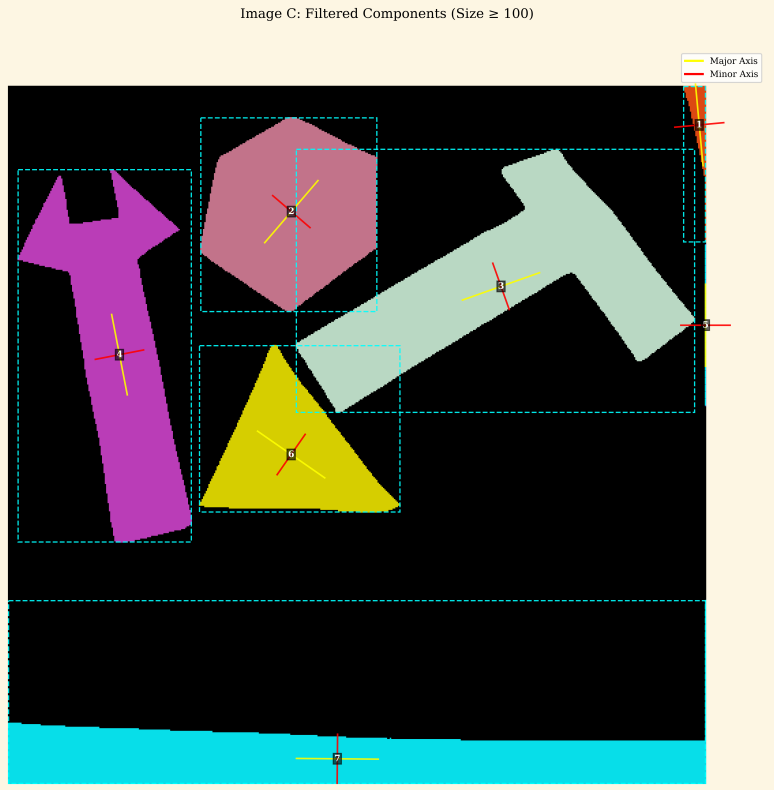
where P is the perimeter and A is the area.

3 Experimental Results

The algorithm was evaluated across three minimum size thresholds: 100, 500, and 1000 pixels.
Note: Kindly correlate the Labeled Components with their respective Feature Tables: For Case 1 refer to Figure 2, and for Case 2 (Figure 3) and Case 3 (Figure 5) refer to Figure 4 and Figure 6 respectively.

3.1 Case 1: Minimum Size Threshold = 100

At $T_{size} = 100$, the algorithm labels artifacts/noise in the image as components along with primary objects of interest as shown in Figure 2.



Component Description Table (Size ≥ 100)

ID	Area	Centroid	Bounding Box	Orient(deg)	Elongation	Eccentricity	Perimeter	Compactness
1	620	(506.4, 28.2)	[495,0,511,114]	84.7	57.10	0.991	196	61.96
2	13148	(207.3, 91.7)	[141,23,270,165]	-49.4	1.18	0.388	380	10.98
3	19762	(361.1, 146.6)	[211,46,503,239]	-19.6	5.54	0.905	693	24.30
4	15327	(81.3, 196.6)	[7,61,134,334]	79.0	13.80	0.963	732	34.96
5	119	(511.0, 175.0)	[511,116,511,234]	90.0	0.00	1.000	119	119.00
6	8913	(207.2, 269.8)	[140,190,287,312]	34.8	1.49	0.573	383	16.46
7	18472	(241.1, 493.0)	[0,377,511,511]	0.5	158.65	0.997	1200	77.96

Total Components Labelled:	7

Figure 2: Labeled Components and Corresponding Feature Table for Case 1

3.2 Case 2: Minimum Size Threshold = 500

Increasing the threshold effectively filters out noise while correctly labelling the primary objects.

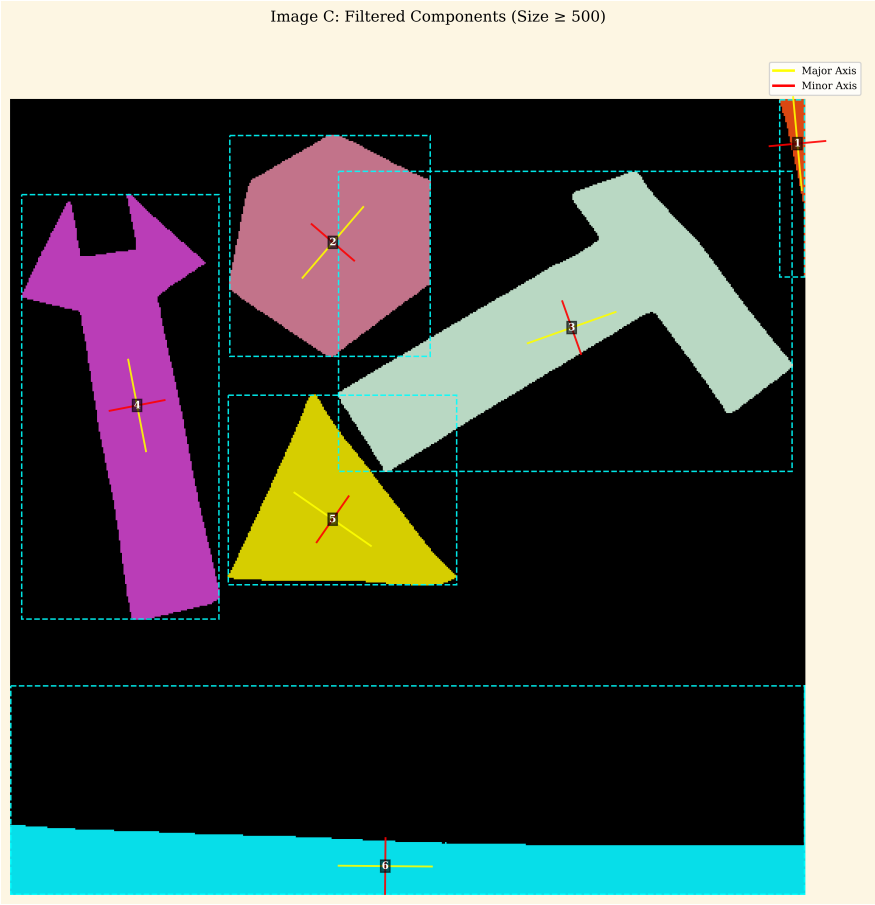


Figure 3: Labeled Components ($Size \geq 500$).

Component Description Table (Size ≥ 500)

ID	Area	Centroid	Bounding Box	Orient(deg)	Elongation	Eccentricity	Perimeter	Compactness
1	620	(506.4, 28.2)	[495,0,511,114]	84.7	57.10	0.991	196	61.96
2	13148	(207.3, 91.7)	[141,23,270,165]	-49.4	1.18	0.388	380	10.98
3	19762	(361.1, 146.6)	[211,46,503,239]	-19.6	5.54	0.905	693	24.30
4	15327	(81.3, 196.6)	[7,61,134,334]	79.0	13.80	0.963	732	34.96
5	8913	(207.2, 269.8)	[140,190,287,312]	34.8	1.49	0.573	383	16.46
6	18472	(241.1, 493.0)	[0,377,511,511]	0.5	158.65	0.997	1200	77.96

Total Components Labelled:	6

Figure 4: Extracted Features for Case 2.

3.3 Case 3: Minimum Size Threshold = 1000

At $T_{size} = 1000$, only the most significant geometric structures remain.

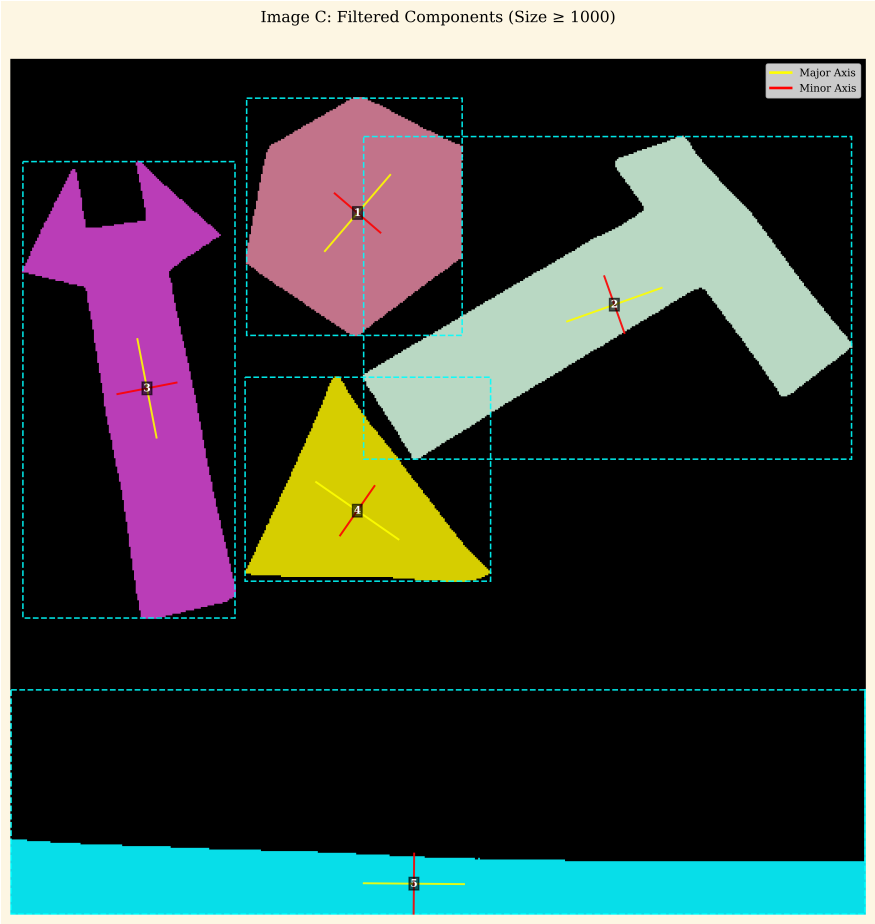


Figure 5: Labeled Components ($Size \geq 1000$).

Component Description Table (Size ≥ 1000)

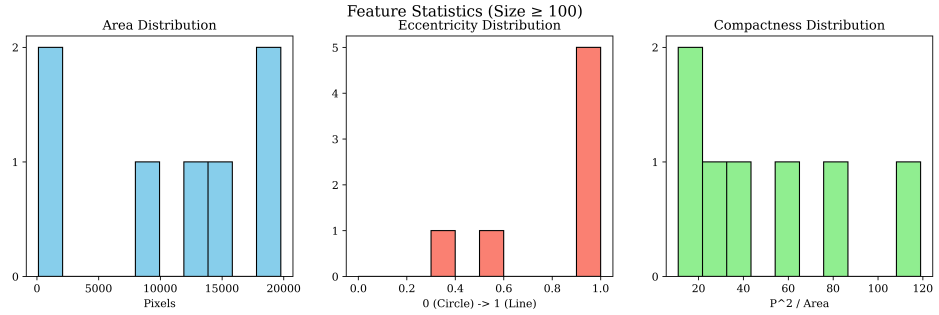
ID	Area	Centroid	Bounding Box	Orient(deg)	Elongation	Eccentricity	Perimeter	Compactness
1	13148	(207.3, 91.7)	[141,23,270,165]	-49.4	1.18	0.388	380	10.98
2	19762	(361.1, 146.6)	[211,46,503,239]	-19.6	5.54	0.905	693	24.30
3	15327	(81.3, 196.6)	[7,61,134,334]	79.0	13.80	0.963	732	34.96
4	8913	(207.2, 269.8)	[140,190,287,312]	34.8	1.49	0.573	383	16.46
5	18472	(241.1, 493.0)	[0,377,511,511]	0.5	158.65	0.997	1200	77.96

Total Components Labelled:	5

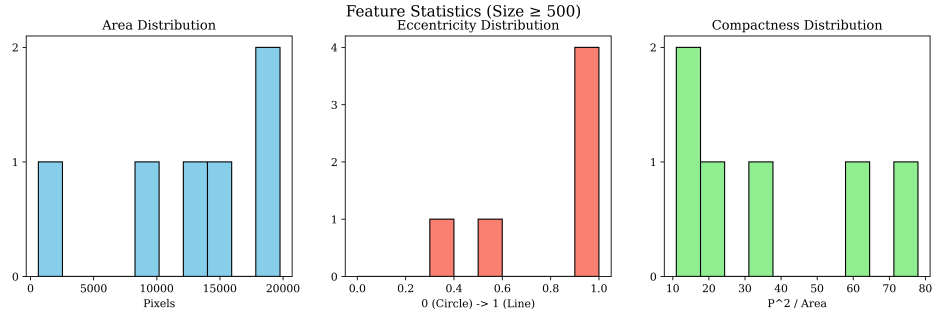
Figure 6: Extracted Features for Case 3.

4 Histogram Comparison

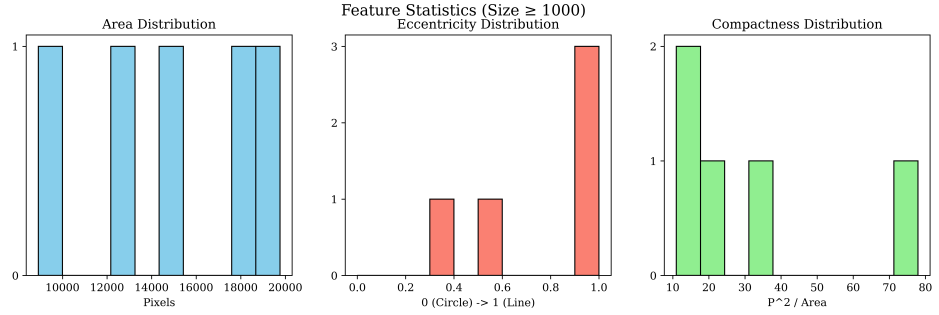
The following histograms illustrate the distribution of labels across the image at different filtering stages. As the threshold increases, the removal of noise and smaller components is reflected in the simplification of the label frequency.



(a) Frequency of Labels ($Size \geq 100$)



(b) Frequency of Labels ($Size \geq 500$)



(c) Frequency of Labels ($Size \geq 1000$)

Figure 7: Comparison of component label distributions at various size thresholds.

5 Analysis

5.1 Component Descriptions

The following components were identified and analyzed based on their geometric properties and attributes:

Hammer: The largest component in the scene with an area of 19,762 pixels. It exhibits an elongation of 5.54 and a distinct orientation of approximately -19.6°.

Wrench: A highly elongated tool-shaped object with an area of 15,327 pixels and a near-vertical orientation of 79.0°.

Hexagon: A compact geometric shape with the lowest eccentricity (0.388) among the primary objects, indicating a more equilateral structure.

Triangle: A mid-sized geometric component with an area of 8,913 pixels and a relatively low elongation of 1.49.

Cyan Base: A wide, horizontal structure spanning the bottom of the frame. It has the highest perimeter (1,200 pixels) and a very high elongation of 158.65 due to its thin, wide aspect ratio.

Orange Sliver (Noise): A thin vertical artifact located at the top-right edge with an area of 620 pixels. It is removed when the size threshold is increased to 1000.

Small Fragment (Noise): A tiny vertical sliver with an area of only 119 pixels, which is immediately filtered out at any threshold above 100.

5.2 Noise Analysis and Size Filter Trade-off

The experimental results highlight a fundamental trade-off:

- **Lower Threshold (100 & 500):** Small noise components are detected as objects.
- **High Threshold (1000):** Fine structural details may be lost if they do not meet the pixel count (T_{size}) requirement.

5.3 Geometric Validation

To ensure the accuracy of the moment-based orientation, the invariant property $I_{max} + I_{min} = a + c$ was calculated. The error was consistently negligible (6.0×10^{-8}), validating the orientation calculation and axis visualization.

6 Conclusion

The iterative CCL algorithm effectively labelled the components in the test image. The 1000-pixel threshold successfully isolates the Wrench, Hammer, Hexagon, Triangle, and Base as the primary structural components.

Note: For source code kindly refer to appendix on next page

7 Appendix: Source Code

The following Python script was used to perform all operations described in this report.

```python
# Paste your python code here.
# It will be formatted automatically with syntax highlighting.
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from collections import defaultdict

plt.rcParams.update(
    {
        "font.family": "serif",
        "font.size": 10,
        "axes.titlesize": 12,
        "axes.labelsize": 10,
        "figure.dpi": 600,
    }
)


def load_and_preprocess(path):
    try:
        with open(path, "rb") as f:
            _ = f.read(512)
            img = np.fromfile(f, dtype=np.uint8)

        original = img.reshape((512, 512))
        binary = (original > 128).astype(np.uint8)

        binary = 1 - binary

        return original, binary
    except FileNotFoundError:
        print(f"Error: File {path} not found.")
        exit()


def label_components(binary_img):
    """Iterative CCL with 4-connectivity."""
    rows, cols = binary_img.shape
    labels = np.zeros((rows, cols), dtype=int)
    current_label = 1
    equivalences = defaultdict(set)

    # Pass 1 Raster Scan
    for i in range(rows):
        for j in range(cols):
            if binary_img[i, j] == 1:
                neighbors = []
                if i > 0 and labels[i - 1, j] > 0:
                    neighbors.append(labels[i - 1, j])
                if j > 0 and labels[i, j - 1] > 0:
                    neighbors.append(labels[i, j - 1])

                if not neighbors:
                    labels[i, j] = current_label
                    current_label += 1
```

```
                elif len(neighbors) == 1:
                    labels[i, j] = neighbors[0]
                else:
                    min_lab = min(neighbors)
                    labels[i, j] = min_lab
                    for lab in neighbors:
                        if lab != min_lab:
                            equivalences[min_lab].add(lab)
                            equivalences[lab].add(min_lab)

    # Resolve Equivalences
    label_map = {}
    visited = set()

    def dfs(node, root):
        stack = [node]
        while stack:
            n = stack.pop()
            if n not in visited:
                visited.add(n)
                label_map[n] = root
                stack.extend(equivalences[n])

    for i in range(1, current_label):
        if i not in visited:
            dfs(i, i)

    # Pass 2
    for i in range(rows):
        for j in range(cols):
            if labels[i, j] > 0:
                labels[i, j] = label_map.get(labels[i, j], labels[i, j])

    return labels


def generate_color_map(labels):
    unique_labels = np.unique(labels)
    unique_labels = unique_labels[unique_labels != 0]

    np.random.seed(29)
    color_map = {}
    for lab in unique_labels:
        color = np.random.rand(3)
        while np.mean(color) < 0.4:
            color = np.random.rand(3)
        color_map[lab] = color
    return color_map


def filter_and_relabel(labels, min_size):
    unique_labels, counts = np.unique(labels, return_counts=True)
    valid_mask = (unique_labels != 0) & (counts >= min_size)
    valid_labels = unique_labels[valid_mask]

    new_labels = np.zeros_like(labels)
    mapping = {}

    for new_id, old_id in enumerate(valid_labels, 1):
```

```
        new_labels[labels == old_id] = new_id
        mapping[new_id] = old_id

    return new_labels, mapping


def extract_features(labels):
    props = {}
    indices = np.unique(labels)
    indices = indices[indices != 0]

    for lab in indices:
        coords = np.argwhere(labels == lab)
        y, x = coords[:, 0], coords[:, 1]
        area = len(coords)

        xc, yc = np.mean(x), np.mean(y)
        min_x, max_x = np.min(x), np.max(x)
        min_y, max_y = np.min(y), np.max(y)

        x_prime = x - xc
        y_prime = y - yc

        a = np.sum(x_prime**2)
        b = 2 * np.sum(x_prime * y_prime)
        c = np.sum(y_prime**2)

        theta = 0.5 * np.arctan2(b, a - c)
        theta_deg = np.degrees(theta)

        term = np.sqrt((a - c) ** 2 + b**2)
        Imax = (a + c + term) / 2
        Imin = (a + c - term) / 2

        # Elongation Ratio
        elongation = (Imax / Imin) if Imin > 0 else 0

        # Moment Error Check
        moment_error = abs((a + c) - (Imax + Imin))
        eccentricity = np.sqrt(1 - (Imin / Imax)) if Imax > 0 else 0

        perimeter = 0
        rows_img, cols_img = labels.shape
        for py, px in coords:
            is_edge = py == 0 or py == rows_img - 1 or px == 0 or px == cols_img - 1
            if not is_edge:
                if (
                    labels[py - 1, px] != lab
                    or labels[py + 1, px] != lab
                    or labels[py, px - 1] != lab
                    or labels[py, px + 1] != lab
                ):
                    is_edge = True
            if is_edge:
                perimeter += 1

        compactness = (perimeter**2) / area

        props[lab] = {
```

```
            "area": area,
            "centroid": (xc, yc),
            "bbox": (min_x, min_y, max_x, max_y),
            "orientation": theta,
            "orientation_deg": theta_deg,
            "elongation": elongation,
            "eccentricity": eccentricity,
            "perimeter": perimeter,
            "compactness": compactness,
            "moment_error": moment_error,
        }
    return props


def save_component_description_table(props, size_thresh):

    if not props:
        return

    column_labels = [
        "ID",
        "Area",
        "Centroid",
        "Bounding Box",
        "Orient(deg)",
        "Elongation",
        "Eccentricity",
        "Perimeter",
        "Compactness",
    ]
    table_data = []

    for lab in sorted(props.keys()):
        p = props[lab]
        xc, yc = p["centroid"]
        min_x, min_y, max_x, max_y = p["bbox"]

        table_data.append(
            [
                lab,
                p["area"],
                f"({xc:.1f}, {yc:.1f})",
                f"[{min_x},{min_y},{max_x},{max_y}]",
                f"{p['orientation_deg']:.1f}",
                f"{p['elongation']:.2f}",
                f"{p['eccentricity']:.3f}",
                f"{p['perimeter']}",
                f"{p['compactness']:.2f}",
            ]
        )

    total_count = len(props)
    n_rows = len(table_data) + 1

    fig_height = max(2.5, (n_rows + 1) * 0.5)

    fig, (ax_main, ax_sum) = plt.subplots(
        2, 1, figsize=(11, fig_height), gridspec_kw={"height_ratios": [n_rows, 1]}
    )
```

```
    ax_main.axis("off")
    ax_sum.axis("off")

    main_table = ax_main.table(
        cellText=table_data,
        colLabels=column_labels,
        loc="center",
        cellLoc="center",
        colWidths=[0.05, 0.08, 0.15, 0.18, 0.12, 0.10, 0.15, 0.15, 0.15],
    )

    main_table.auto_set_font_size(False)
    main_table.set_fontsize(9)
    main_table.scale(1, 1.5)

    for (row, col), cell in main_table.get_celld().items():
        cell.set_edgecolor("black")
        cell.set_linewidth(0.5)
        if row == 0:
            cell.set_text_props(weight="bold")
            cell.set_facecolor("#eaeaea")
            cell.set_linewidth(1.0)
        else:
            cell.set_facecolor("white")

    summary_data = [["Total Components Labelled:", total_count]]

    sum_table = ax_sum.table(
        cellText=summary_data, loc="top", cellLoc="center", colWidths=[0.98, 0.15]
    )

    sum_table.auto_set_font_size(False)
    sum_table.set_fontsize(10)
    sum_table.scale(1, 1.5)

    for (row, col), cell in sum_table.get_celld().items():
        cell.set_edgecolor("black")
        cell.set_linewidth(1.0)
        cell.set_text_props(weight="bold")
        if col == 0:
            cell.set_text_props(ha="center")
            cell.set_facecolor("#f9f9f9")
        else:
            cell.set_text_props(ha="center")
            cell.set_facecolor("white")

    plt.suptitle(
        f"Component Description Table (Size >= {size_thresh})"
        fontsize=12,
        fontweight="bold",
        y=0.95,
    )

    outfile = f"Component_Description_Table_Size_{size_thresh}.pdf"
    plt.savefig(outfile, format="pdf", dpi=600, bbox_inches="tight")
    plt.close()
    print(f"[Saved PDF] {outfile}")
```

```
292
293 def visualize_results(labels, props, size_thresh, global_colors, mapping):
294     fig, ax = plt.subplots(figsize=(10, 10))
295
296     colored = np.zeros((*labels.shape, 3))
297     colored[:] = [0, 0, 0]
298
299     for lab in props.keys():
300         original_id = mapping[lab]
301         colored[labels == lab] = global_colors[original_id]
302
303     ax.imshow(colored)
304     ax.set_title(
305     f"Image C: Filtered Components (Size $\\geq {size_thresh}$)\n",
306     pad=15,
307     )
308     ax.axis("off")
309
310     for lab, p in props.items():
311         xc, yc = p["centroid"]
312         min_x, min_y, max_x, max_y = p["bbox"]
313         theta = p["orientation"]
314
315         rect = patches.Rectangle(
316             (min_x, min_y),
317             max_x - min_x,
318             max_y - min_y,
319             linewidth=1.2,
320             edgecolor="cyan",
321             facecolor="none",
322             linestyle="--",
323             alpha=0.9,
324         )
325         ax.add_patch(rect)
326
327         axis_len = 30
328         x1 = xc + axis_len * np.cos(theta)
329         y1 = yc + axis_len * np.sin(theta)
330         x2 = xc - axis_len * np.cos(theta)
331         y2 = yc - axis_len * np.sin(theta)
332         ax.plot([x1, x2], [y1, y2], color="yellow", linewidth=1.5, alpha=0.9)
333
334         x3 = xc + (axis_len * 0.6) * np.cos(theta + np.pi / 2)
335         y3 = yc + (axis_len * 0.6) * np.sin(theta + np.pi / 2)
336         x4 = xc - (axis_len * 0.6) * np.cos(theta + np.pi / 2)
337         y4 = yc - (axis_len * 0.6) * np.sin(theta + np.pi / 2)
338         ax.plot([x3, x4], [y3, y4], color="red", linewidth=1.5, alpha=0.9)
339
340         ax.text(
341             xc,
342             yc,
343             str(lab),
344             color="white",
345             fontsize=8,
346             fontweight="bold",
347             ha="center",
348             va="center",
349             bbox=dict(facecolor="black", alpha=0.6, pad=1),
350         )
```

```
351
352     legend_elements = [
353         plt.Line2D([0], [0], color="yellow", lw=2, label="Major Axis"),
354         plt.Line2D([0], [0], color="red", lw=2, label="Minor Axis"),
355     ]
356     ax.legend(handles=legend_elements, loc="upper right", fontsize=8, framealpha=0.8)
357
358     outfile = f"Image_C_Size_{size_thresh}.png"
359     plt.tight_layout()
360     plt.savefig(outfile, bbox_inches="tight", dpi=600, facecolor="#fdf6e3")
361     plt.close()
362     print(f"[Saved] {outfile}")
363
364
365 if __name__ == "__main__":
366     print("Starting\n")
367
368     original_img, binary_mask = load_and_preprocess("comb.img")
369     plt.imsave("Image_B_Original.png", original_img, cmap="gray")
370     plt.imsave("Image_BT_Binary.png", binary_mask, cmap="gray")
371
372     raw_labels = label_components(binary_mask)
373     global_colors = generate_color_map(raw_labels)
374
375     size_filters = [100, 500, 1000]
376
377     for size in size_filters:
378         clean_labels, mapping = filter_and_relabel(raw_labels, size)
379         features = extract_features(clean_labels)
380
381         save_component_description_table(features, size)
382
383         visualize_results(clean_labels, features, size, global_colors, mapping)
384
385     print("Ended")
```