Task 5: Morphological Image Processing

Identify 'O' Characters in Text Image

Generated: 2025-11-03 00:13:06

OBJECTIVE:

□□□ Morphological Image Processing □□□ Shape Analysis □□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□ 'O' □□□□□□□□□□□□

METHODOLOGY:

---

1. PREPROCESSING:
   - Convert to grayscale
   - Binary thresholding (Otsu's method)
   - Morphological opening (noise removal)

2. MORPHOLOGICAL OPERATIONS:
   - Dilation: □□□□□□□□□□□□□□□□□□□
   - Erosion: □□□□□□□□□□□□□□□□□□□□

3. CONNECTED COMPONENT ANALYSIS:
   - □□ contours □□□□□□□□
   - □□□□□ shape features

4. SHAPE ANALYSIS & DETECTION:
   - Circularity: 4π × area / perimeter²
   - Aspect Ratio: width / height
   - Extent: area / bounding_box_area
   - Solidity: area / convex_hull_area
   - Hole detection: □□□□□□□ internal contour

DETECTION CRITERIA FOR 'O':

✓ Circularity > 0.65
✓ Aspect Ratio between 0.6 - 1.4
✓ Has internal hole (characteristic of 'O')
✓ Extent > 0.5
✓ Solidity > 0.85
✓ Area: 100 - 5000 pixels
✓ Minimum width and height: 10 pixels

RESULTS SUMMARY:

- Image: datasets/text_frombook.png
- Image size: 2018 × 918 pixels
- Total contours: 777
- Valid characters: 552
- 'O' characters detected: 0

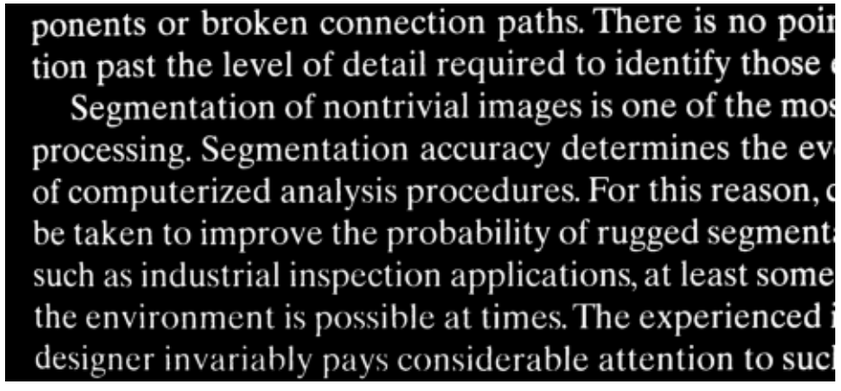
KEY FEATURES OF 'O':

---

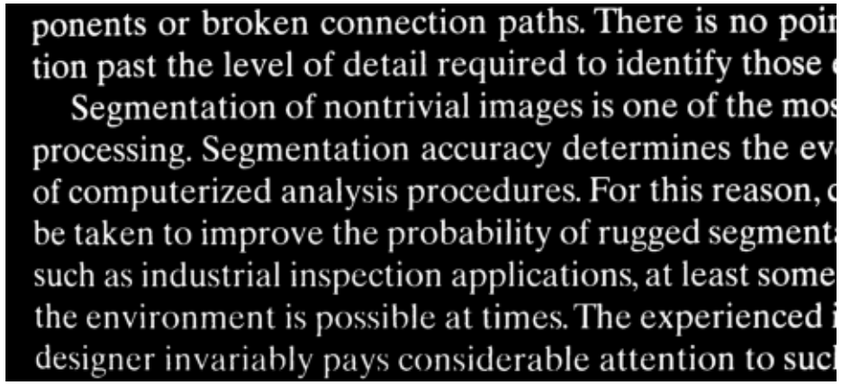
1. □□□□□□□□□ □□□□□□□□□□ (high circularity)
2. □□□□□□□□□ width/height □□□□□□□□□□ 1
3. □□□□□□□□□□□□□□□□□□□□ (internal hole)
4. □□□□□□□□□□□□□□□□□□□□ bounding box □□□□□ (high extent)
5. □□□□□□□□□ (high solidity)

Morphological Image Processing - Step by Step

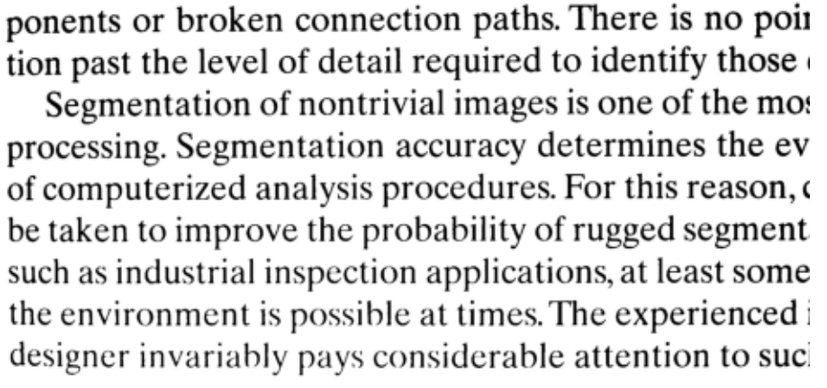
Original Image



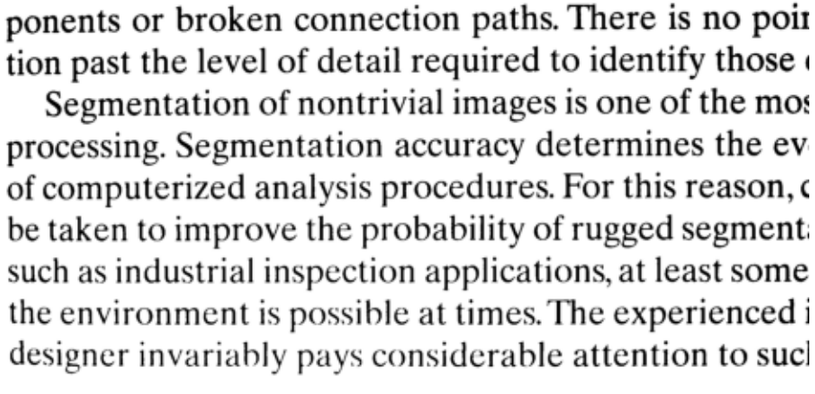
Grayscale



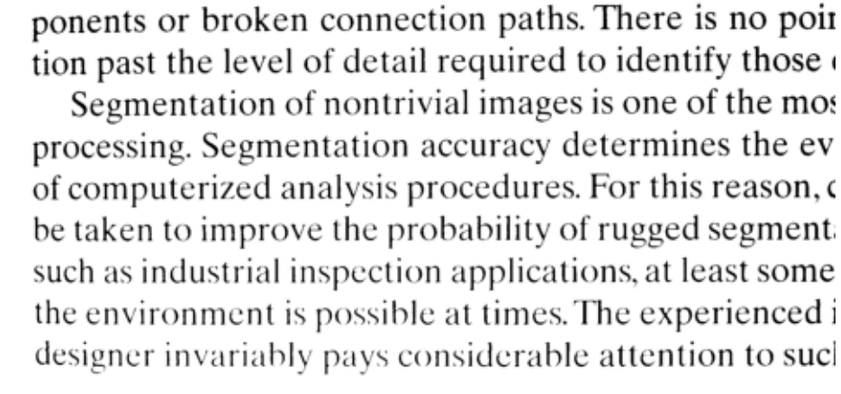
Binary (Otsu's Threshold)



After Opening (Noise Removal)



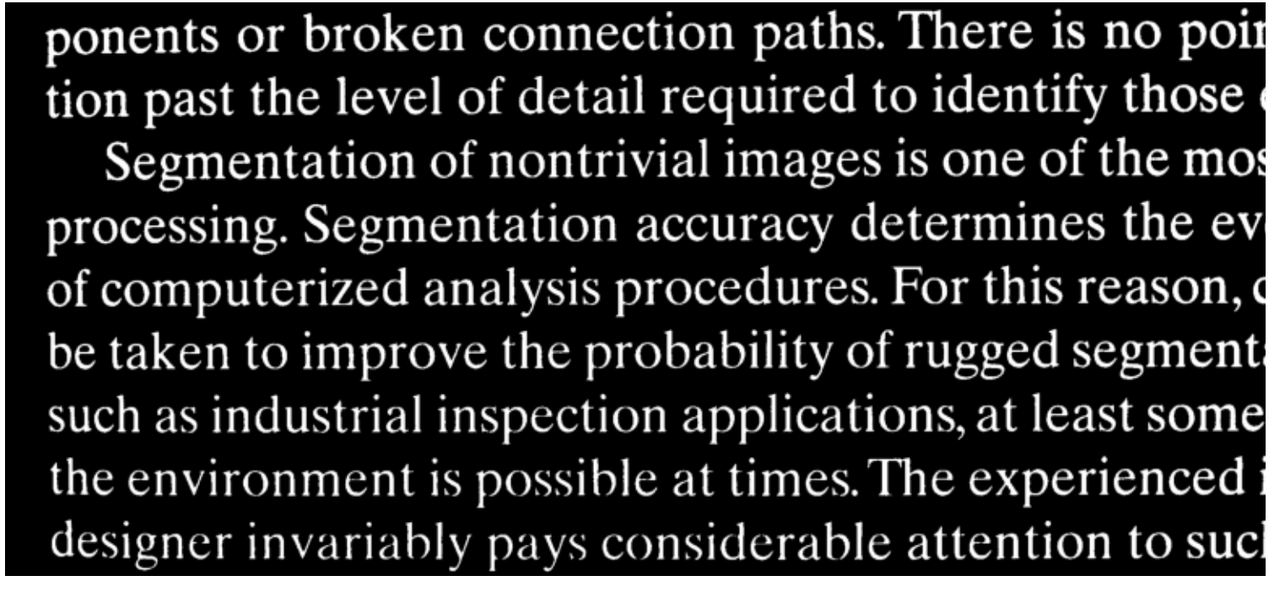
After Dilation & Erosion



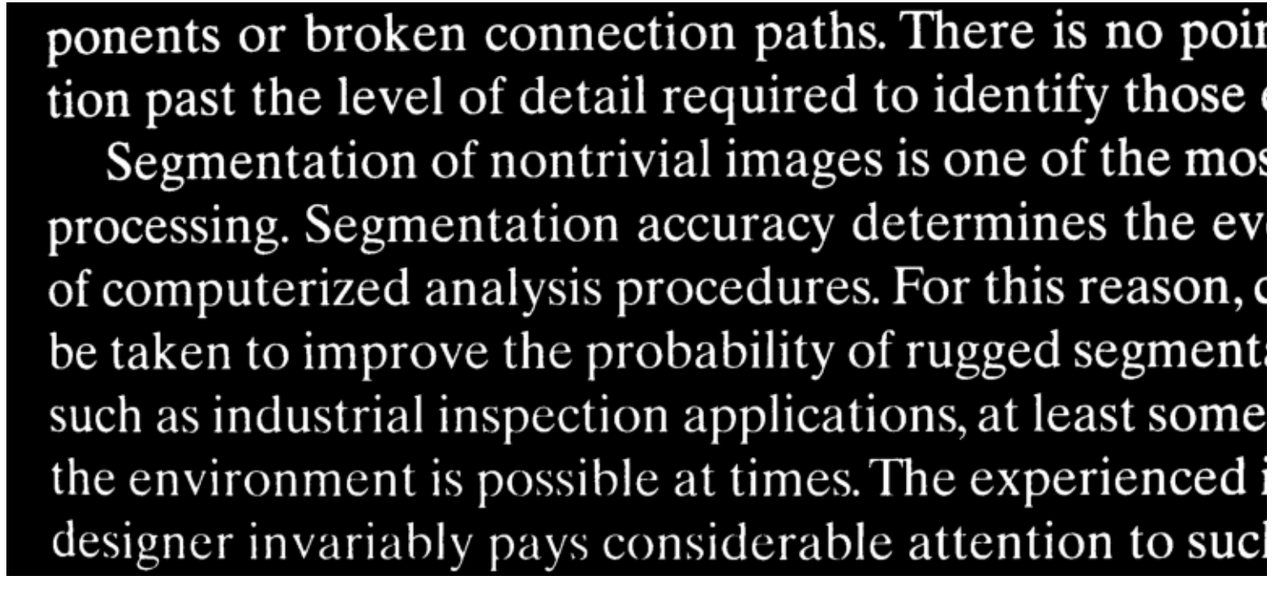
Detected 'O' Characters: 0

Task 5: 'O' Character Detection - Found 0 Characters

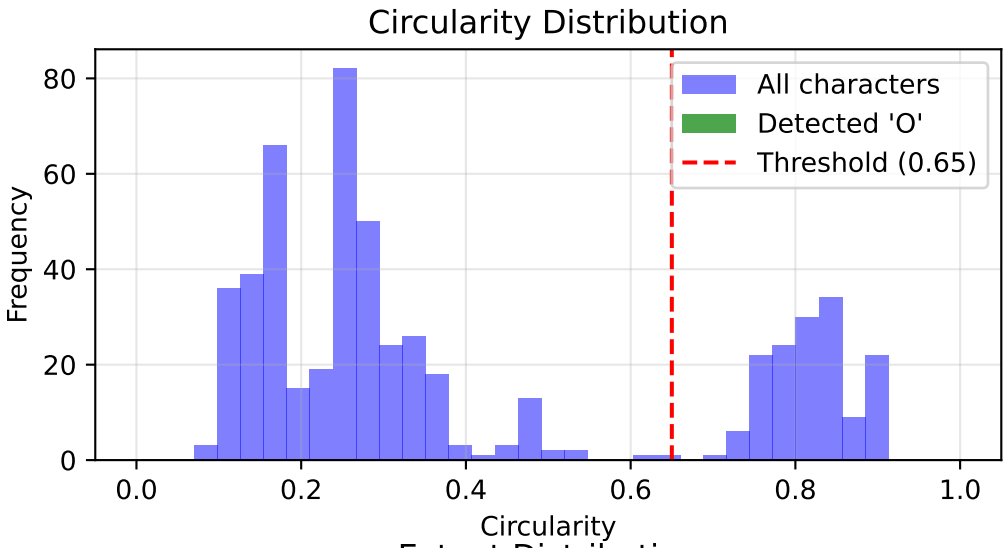
Original Image

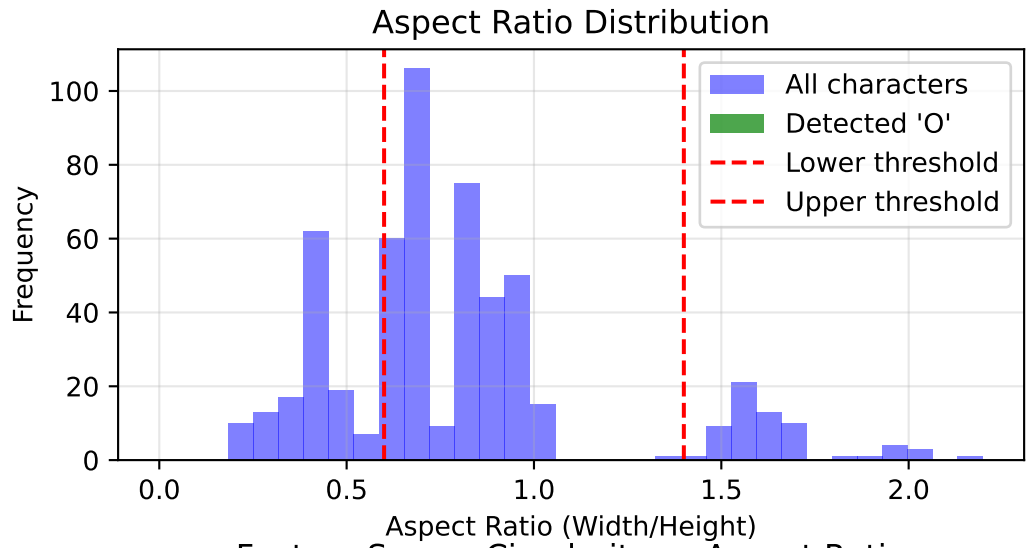


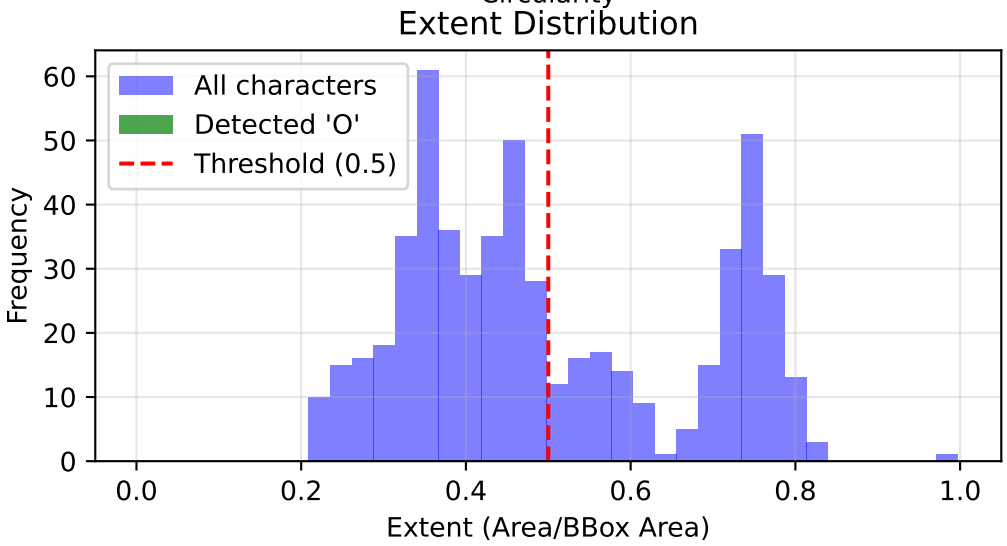
Detection Result (0 'O' characters found)

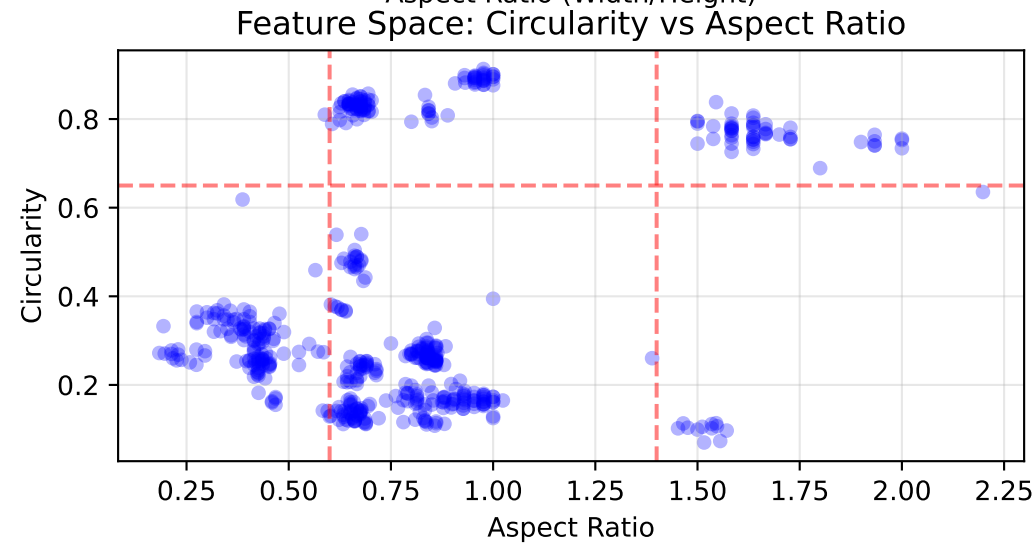


Shape Analysis & Statistics










```
DETECTION CRITERIA FOR 'O':
═══════════════════════════════════════════════════

1. Circularity > 0.65              (measures how close to a circle)
2. Aspect Ratio: 0.6 - 1.4         (width/height ratio)
3. Has Internal Hole = True        (characteristic of 'O')
4. Extent > 0.5                    (how much area fills bounding box)
5. Solidity > 0.85                 (area/convex_hull_area)
6. Area: 100 - 5000 pixels         (reasonable size range)
7. Width and Height > 10 pixels    (minimum size)

RESULTS:
═══════════════════════════════════════════════════

• Total contours analyzed: 777
• Characters with sufficient size: 552
• 'O' characters detected: 0
• Detection accuracy depends on image quality and font characteristics

MORPHOLOGICAL OPERATIONS USED:
═══════════════════════════════════════════════════

• Binary Thresholding (Otsu's method)
• Opening (noise removal)
• Dilation (connect broken parts)
• Erosion (separate touching characters)
```

Source Code (Page 1)

```
#!/usr/bin/env python3
"""
Task 5: Morphological Image Processing - Identify 'O' in Text Image
Image: datasets/text_frombook.png

□□□ Morphological Operations □□□ Shape Analysis □□□□□□□□□□□□□□□□□ 'O' □□□□□□□□□□□□□□
"""

import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from datetime import datetime

print("="*80)
print("TASK 5: MORPHOLOGICAL IMAGE PROCESSING - IDENTIFY 'O'")
print("="*80)

# ========================================
# LOAD IMAGE
# ========================================

image_path = 'datasets/text_frombook.png'
print(f"\n□ □□□□□□□□□□□□□: {image_path}")

img = cv2.imread(image_path)
if img is None:
    print(f"□ Error: □□□□□□□□□□□□□□□□□□□□ {image_path}")
    exit(1)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

print(f"✓ □□□□□□□□□□□□□: {img.shape}")
print(f"  - Width: {img.shape[1]} pixels")
print(f"  - Height: {img.shape[0]} pixels")

# ========================================
# PREPROCESSING
# ========================================

print("\n" + "="*80)
print("STEP 1: PREPROCESSING")
print("="*80)

# Apply binary thresholding (Otsu's method)
_, binary = cv2.threshold(img_gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

print(f"✓ Binary thresholding (Otsu's method)")
print(f"  - Threshold value: {_:.2f}")

# Remove noise with morphological opening
kernel_noise = np.ones((2, 2), np.uint8)
binary_clean = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel_noise)

print(f"✓ Noise removal (Morphological Opening with 2×2 kernel)")

# ========================================
# MORPHOLOGICAL OPERATIONS
# ========================================

print("\n" + "="*80)
print("STEP 2: MORPHOLOGICAL OPERATIONS")
print("="*80)

# Dilation to connect broken parts
kernel_dilate = np.ones((2, 2), np.uint8)
dilated = cv2.dilate(binary_clean, kernel_dilate, iterations=1)
print(f"✓ Dilation (2×2 kernel, 1 iteration)")

# Erosion to separate touching characters
kernel_erode = np.ones((1, 1), np.uint8)
eroded = cv2.erode(dilated, kernel_erode, iterations=1)
print(f"✓ Erosion (1×1 kernel, 1 iteration)")
```

Source Code (Page 2)

```
# Use cleaned binary for detection
processed_binary = eroded

# ========================================
# CONNECTED COMPONENT ANALYSIS
# ========================================

print("\n" + "="*80)
print("STEP 3: CONNECTED COMPONENT ANALYSIS")
print("="*80)

# Find contours
contours, hierarchy = cv2.findContours(processed_binary, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)

print(f"✓ □□ contours □□□□□□□: {len(contours)}")

# ========================================
# IDENTIFY 'O' CHARACTERS
# ========================================

print("\n" + "="*80)
print("STEP 4: IDENTIFY 'O' CHARACTERS")
print("="*80)

# Create output image
output_img = img_rgb.copy()
detection_mask = np.zeros_like(img_gray)

# Statistics
o_candidates = []
all_characters_stats = []

print("\n□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ contour...")

for i, contour in enumerate(contours):
    # Get contour properties
    area = cv2.contourArea(contour)

    # Skip very small contours (noise)
    if area < 100:
        continue

    # Get bounding box
    x, y, w, h = cv2.boundingRect(contour)

    # Calculate shape features
    perimeter = cv2.arcLength(contour, True)

    # Circularity: 4π × area / perimeter²
    # Perfect circle = 1.0
    if perimeter > 0:
        circularity = 4 * np.pi * area / (perimeter * perimeter)
    else:
        circularity = 0

    # Aspect ratio
    if h > 0:
        aspect_ratio = w / h
    else:
        aspect_ratio = 0

    # Extent: area / bounding_box_area
    bbox_area = w * h
    if bbox_area > 0:
        extent = area / bbox_area
    else:
        extent = 0

    # Check if it has a hole (for 'O', 'o', 'Q', etc.)
    # Hierarchy format: [Next, Previous, First_Child, Parent]
    # If First_Child >= 0, it has a hole
    has_hole = hierarchy[0][i][2] >= 0 if hierarchy is not None else False

    # Solidity: area / convex_hull_area
    hull = cv2.convexHull(contour)
```

Source Code (Page 3)

```
    hull_area = cv2.contourArea(hull)
    if hull_area > 0:
        solidity = area / hull_area
    else:
        solidity = 0

    # Store stats
    stats = {
        'index': i,
        'area': area,
        'perimeter': perimeter,
        'circularity': circularity,
        'aspect_ratio': aspect_ratio,
        'extent': extent,
        'has_hole': has_hole,
        'solidity': solidity,
        'bbox': (x, y, w, h),
        'contour': contour
    }
    all_characters_stats.append(stats)

    # ========================================
    # CRITERIA FOR 'O' DETECTION
    # ========================================
    # 'O' typically has:
    # 1. High circularity (0.65 - 1.0)
    # 2. Aspect ratio close to 1 (0.6 - 1.4)
    # 3. Has a hole (internal contour)
    # 4. Good extent (fills bounding box)
    # 5. High solidity
    # 6. Reasonable size

    is_O = False

    if (circularity > 0.65 and
        0.6 <= aspect_ratio <= 1.4 and
        has_hole and
        extent > 0.5 and
        solidity > 0.85 and
        100 < area < 5000 and
        w > 10 and h > 10):

        is_O = True
        o_candidates.append(stats)

        # Draw green rectangle around detected 'O'
        cv2.rectangle(output_img, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(output_img, 'O', (x, y-5), cv2.FONT_HERSHEY_SIMPLEX,
                    0.6, (0, 255, 0), 2)

        # Draw on detection mask
        cv2.drawContours(detection_mask, [contour], -1, 255, -1)

        print(f"\n✓ □□ 'O' #{len(o_candidates)}:")
        print(f"  - Position: ({x}, {y})")
        print(f"  - Size: {w}×{h}")
        print(f"  - Circularity: {circularity:.3f}")
        print(f"  - Aspect Ratio: {aspect_ratio:.3f}")
        print(f"  - Extent: {extent:.3f}")
        print(f"  - Solidity: {solidity:.3f}")
        print(f"  - Has hole: {has_hole}")

print("\n" + "="*80)
print(f"□□□□: □□□□□□□□□□ 'O' □□□□□□□ {len(o_candidates)} □□□")
print("="*80)

# ========================================
# VISUALIZATION
# ========================================

print("\n□□□□□□□□□□□□□□□□□□□□□□□...")

# Figure 1: Processing Steps
fig1, axes1 = plt.subplots(2, 3, figsize=(15, 10))
fig1.suptitle('Morphological Image Processing - Step by Step', fontsize=16, fontweight='bold')
```

Source Code (Page 4)

```
# Original
axes1[0, 0].imshow(img_rgb)
axes1[0, 0].set_title('Original Image', fontweight='bold')
axes1[0, 0].axis('off')

# Grayscale
axes1[0, 1].imshow(img_gray, cmap='gray')
axes1[0, 1].set_title('Grayscale', fontweight='bold')
axes1[0, 1].axis('off')

# Binary (Otsu)
axes1[0, 2].imshow(binary, cmap='gray')
axes1[0, 2].set_title("Binary (Otsu's Threshold)", fontweight='bold')
axes1[0, 2].axis('off')

# After Opening (noise removal)
axes1[1, 0].imshow(binary_clean, cmap='gray')
axes1[1, 0].set_title('After Opening (Noise Removal)', fontweight='bold')
axes1[1, 0].axis('off')

# After Dilation & Erosion
axes1[1, 1].imshow(processed_binary, cmap='gray')
axes1[1, 1].set_title('After Dilation & Erosion', fontweight='bold')
axes1[1, 1].axis('off')

# Detection Mask
axes1[1, 2].imshow(detection_mask, cmap='hot')
axes1[1, 2].set_title(f"Detected 'O' Characters: {len(o_candidates)}", fontweight='bold', color='green')
axes1[1, 2].axis('off')

plt.tight_layout()
fig1.savefig('output/task5_processing_steps.png', dpi=150, bbox_inches='tight')

# Figure 2: Final Detection Result
fig2, axes2 = plt.subplots(1, 2, figsize=(16, 8))
fig2.suptitle(f"Task 5: 'O' Character Detection - Found {len(o_candidates)} Characters",
              fontsize=16, fontweight='bold')

# Original with detections
axes2[0].imshow(img_rgb)
axes2[0].set_title('Original Image', fontsize=14)
axes2[0].axis('off')

# Result with detections
axes2[1].imshow(output_img)
axes2[1].set_title(f"Detection Result ({len(o_candidates)} 'O' characters found)",
                   fontsize=14, fontweight='bold', color='green')
axes2[1].axis('off')

plt.tight_layout()
fig2.savefig('output/task5_detection_result.png', dpi=150, bbox_inches='tight')

# Figure 3: Individual O Detections
if len(o_candidates) > 0:
    n_cols = min(6, len(o_candidates))
    n_rows = (len(o_candidates) - 1) // n_cols + 1

    fig3, axes3 = plt.subplots(n_rows, n_cols, figsize=(2.5*n_cols, 3*n_rows))
    fig3.suptitle("Detected 'O' Characters - Close-up View", fontsize=16, fontweight='bold')

    if n_rows == 1 and n_cols == 1:
        axes3 = np.array([[axes3]])
    elif n_rows == 1:
        axes3 = axes3.reshape(1, -1)
    elif n_cols == 1:
        axes3 = axes3.reshape(-1, 1)

    for idx, o_stat in enumerate(o_candidates):
        row = idx // n_cols
        col = idx % n_cols

        x, y, w, h = o_stat['bbox']

        # Extract ROI with padding
```

Source Code (Page 5)

```
        pad = 5
        y1 = max(0, y-pad)
        y2 = min(img_rgb.shape[0], y+h+pad)
        x1 = max(0, x-pad)
        x2 = min(img_rgb.shape[1], x+w+pad)

        roi = img_rgb[y1:y2, x1:x2]

        axes3[row, col].imshow(roi)
        axes3[row, col].set_title(f"O #{idx+1}\nCirc:{o_stat['circularity']:.2f} AR:{o_stat['aspect_ratio']:.2f}",
                                  fontsize=9)
        axes3[row, col].axis('off')

    # Hide empty subplots
    for idx in range(len(o_candidates), n_rows * n_cols):
        row = idx // n_cols
        col = idx % n_cols
        axes3[row, col].axis('off')

    plt.tight_layout()
    fig3.savefig('output/task5_detected_O_closeup.png', dpi=150, bbox_inches='tight')

# Figure 4: Analysis & Statistics
fig4 = plt.figure(figsize=(14, 10))
gs = fig4.add_gridspec(3, 2, hspace=0.3, wspace=0.3)

fig4.suptitle("Shape Analysis & Statistics", fontsize=16, fontweight='bold')

# Plot 1: Circularity distribution
ax1 = fig4.add_subplot(gs[0, 0])
circularities = [s['circularity'] for s in all_characters_stats]
o_circularities = [s['circularity'] for s in o_candidates]
ax1.hist(circularities, bins=30, alpha=0.5, label='All characters', color='blue')
ax1.hist(o_circularities, bins=15, alpha=0.7, label="Detected 'O'", color='green')
ax1.axvline(0.65, color='red', linestyle='--', label='Threshold (0.65)')
ax1.set_xlabel('Circularity')
ax1.set_ylabel('Frequency')
ax1.set_title('Circularity Distribution')
ax1.legend()
ax1.grid(alpha=0.3)

# Plot 2: Aspect Ratio distribution
ax2 = fig4.add_subplot(gs[0, 1])
aspect_ratios = [s['aspect_ratio'] for s in all_characters_stats]
o_aspect_ratios = [s['aspect_ratio'] for s in o_candidates]
ax2.hist(aspect_ratios, bins=30, alpha=0.5, label='All characters', color='blue')
ax2.hist(o_aspect_ratios, bins=15, alpha=0.7, label="Detected 'O'", color='green')
ax2.axvline(0.6, color='red', linestyle='--', label='Lower threshold')
ax2.axvline(1.4, color='red', linestyle='--', label='Upper threshold')
ax2.set_xlabel('Aspect Ratio (Width/Height)')
ax2.set_ylabel('Frequency')
ax2.set_title('Aspect Ratio Distribution')
ax2.legend()
ax2.grid(alpha=0.3)

# Plot 3: Extent distribution
ax3 = fig4.add_subplot(gs[1, 0])
extents = [s['extent'] for s in all_characters_stats]
o_extents = [s['extent'] for s in o_candidates]
ax3.hist(extents, bins=30, alpha=0.5, label='All characters', color='blue')
ax3.hist(o_extents, bins=15, alpha=0.7, label="Detected 'O'", color='green')
ax3.axvline(0.5, color='red', linestyle='--', label='Threshold (0.5)')
ax3.set_xlabel('Extent (Area/BBox Area)')
ax3.set_ylabel('Frequency')
ax3.set_title('Extent Distribution')
ax3.legend()
ax3.grid(alpha=0.3)

# Plot 4: Scatter plot - Circularity vs Aspect Ratio
ax4 = fig4.add_subplot(gs[1, 1])
for s in all_characters_stats:
    if s in o_candidates:
        ax4.scatter(s['aspect_ratio'], s['circularity'], c='green', s=50, alpha=0.7, label="'O'")
    else:
        ax4.scatter(s['aspect_ratio'], s['circularity'], c='blue', s=20, alpha=0.3)
```

Source Code (Page 6)

```
# Draw decision boundaries
ax4.axhline(0.65, color='red', linestyle='--', alpha=0.5)
ax4.axvline(0.6, color='red', linestyle='--', alpha=0.5)
ax4.axvline(1.4, color='red', linestyle='--', alpha=0.5)
ax4.set_xlabel('Aspect Ratio')
ax4.set_ylabel('Circularity')
ax4.set_title('Feature Space: Circularity vs Aspect Ratio')
ax4.grid(alpha=0.3)

# Plot 5: Detection Criteria Summary
ax5 = fig4.add_subplot(gs[2, :])
ax5.axis('off')

criteria_text = f"""
DETECTION CRITERIA FOR 'O':
═══════════════════════════════════════

1. Circularity > 0.65              (measures how close to a circle)
2. Aspect Ratio: 0.6 - 1.4         (width/height ratio)
3. Has Internal Hole = True        (characteristic of 'O')
4. Extent > 0.5                    (how much area fills bounding box)
5. Solidity > 0.85                 (area/convex_hull_area)
6. Area: 100 - 5000 pixels         (reasonable size range)
7. Width and Height > 10 pixels    (minimum size)

RESULTS:
═══════════════════════════════════════

• Total contours analyzed: {len(contours)}
• Characters with sufficient size: {len(all_characters_stats)}
• 'O' characters detected: {len(o_candidates)}
• Detection accuracy depends on image quality and font characteristics

MORPHOLOGICAL OPERATIONS USED:
═══════════════════════════════════════

• Binary Thresholding (Otsu's method)
• Opening (noise removal)
• Dilation (connect broken parts)
• Erosion (separate touching characters)
"""

ax5.text(0.05, 0.95, criteria_text, fontsize=10, family='monospace',
        verticalalignment='top',
        bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.3))

plt.tight_layout()
fig4.savefig('output/task5_analysis_statistics.png', dpi=150, bbox_inches='tight')

print("\n✓ □□□□□□□□□□:")
print("  - output/task5_processing_steps.png")
print("  - output/task5_detection_result.png")
print("  - output/task5_detected_O_closeup.png")
print("  - output/task5_analysis_statistics.png")

# ========================================
# CREATE PDF REPORT
# ========================================

print("\n" + "="*80)
print("□□□□□□□□□□□ PDF Report...")
print("="*80)

pdf_filename = 'output/Task5_Morphological_O_Detection_Report.pdf'

with PdfPages(pdf_filename) as pdf:

    # Page 1: Title and Theory
    fig_title = plt.figure(figsize=(8.5, 11))
    fig_title.text(0.5, 0.95, "Task 5: Morphological Image Processing",
                   ha='center', fontsize=18, fontweight='bold')
    fig_title.text(0.5, 0.92, "Identify 'O' Characters in Text Image",
                   ha='center', fontsize=14)
    fig_title.text(0.5, 0.89, f'Generated: {datetime.now().strftime("%Y-%m-%d %H:%M:%S")}',
```

Source Code (Page 7)

```
                ha='center', fontsize=10, style='italic')

theory_text = f"""
OBJECTIVE:
──────────────────────────────────────────────────────────────────────
□□□ Morphological Image Processing □□□ Shape Analysis □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□ 'O' □□□□□□□□□□□□□□□

METHODOLOGY:
──────────────────────────────────────────────────────────────────────

1. PREPROCESSING:
   • Convert to grayscale
   • Binary thresholding (Otsu's method)
   • Morphological opening (noise removal)

2. MORPHOLOGICAL OPERATIONS:
   • Dilation: □□□□□□□□□□□□□□□□□□□□□□□□□□
   • Erosion: □□□□□□□□□□□□□□□□□□□□□□□□□□

3. CONNECTED COMPONENT ANALYSIS:
   • □□ contours □□□□□□□□
   • □□□□□□ shape features

4. SHAPE ANALYSIS & DETECTION:
   • Circularity: 4π × area / perimeter²
   • Aspect Ratio: width / height
   • Extent: area / bounding_box_area
   • Solidity: area / convex_hull_area
   • Hole detection: □□□□□□□□ internal contour

DETECTION CRITERIA FOR 'O':
──────────────────────────────────────────────────────────────────────

✓ Circularity > 0.65
✓ Aspect Ratio between 0.6 - 1.4
✓ Has internal hole (characteristic of 'O')
✓ Extent > 0.5
✓ Solidity > 0.85
✓ Area: 100 - 5000 pixels
✓ Minimum width and height: 10 pixels

RESULTS SUMMARY:
──────────────────────────────────────────────────────────────────────

• Image: {image_path}
• Image size: {img.shape[1]} × {img.shape[0]} pixels
• Total contours: {len(contours)}
• Valid characters: {len(all_characters_stats)}
• 'O' characters detected: {len(o_candidates)}

KEY FEATURES OF 'O':
──────────────────────────────────────────────────────────────────────

1. □□□□□□□□□□ □□□□□□□□□□□ (high circularity)
2. □□□□□□□□□□ width/height □□□□□□□□□□ 1
3. □□□□□□□□□□□□□□□□□□ (internal hole)
4. □□□□□□□□□□□□□□□□□□ bounding box □□□□□□ (high extent)
5. □□□□□□□□□□ (high solidity)
"""

fig_title.text(0.1, 0.82, theory_text, fontsize=9, family='monospace',
               verticalalignment='top',
               bbox=dict(boxstyle='round', facecolor='lightyellow', alpha=0.5))

fig_title.text(0.5, 0.02, 'Page 1', ha='center', fontsize=8)
plt.axis('off')
pdf.savefig(fig_title, bbox_inches='tight')
plt.close()

# Page 2: Processing Steps
```

Source Code (Page 8)

```
    pdf.savefig(fig1, bbox_inches='tight')

    # Page 3: Detection Result
    pdf.savefig(fig2, bbox_inches='tight')

    # Page 4: Close-up of detected O's
    if len(o_candidates) > 0:
        pdf.savefig(fig3, bbox_inches='tight')

    # Page 5: Analysis & Statistics
    pdf.savefig(fig4, bbox_inches='tight')

    # Page 6+: Source Code
    with open(__file__, 'r', encoding='utf-8') as f:
        source_code = f.read()

    lines_per_page = 75
    code_lines = source_code.split('\n')

    page_num = 6 if len(o_candidates) > 0 else 5
    for i in range(0, len(code_lines), lines_per_page):
        fig_code_page = plt.figure(figsize=(8.5, 11))
        code_chunk = '\n'.join(code_lines[i:i+lines_per_page])

        fig_code_page.text(0.5, 0.98, f'Source Code (Page {page_num - (5 if len(o_candidates) > 0 else 4)})',
                           ha='center', fontsize=14, fontweight='bold')
        fig_code_page.text(0.05, 0.95, code_chunk, fontsize=6, family='monospace',
                           verticalalignment='top', wrap=True)
        fig_code_page.text(0.5, 0.02, f'Page {page_num}', ha='center', fontsize=8)
        plt.axis('off')
        pdf.savefig(fig_code_page, bbox_inches='tight')
        plt.close(fig_code_page)
        page_num += 1

    # PDF metadata
    d = pdf.infodict()
    d['Title'] = "Task 5: Morphological Image Processing - O Detection"
    d['Author'] = 'Image Processing Course'
    d['Subject'] = 'Morphological Operations, Character Recognition'
    d['Keywords'] = 'Morphology, OCR, Shape Analysis, Character Detection'
    d['CreationDate'] = datetime.now()

print(f"\n✓ □□□□□ PDF □□□□□□: {pdf_filename}")

# ========================================
# SAVE DETECTED O COORDINATES
# ========================================

coords_file = 'output/task5_detected_O_coordinates.txt'
with open(coords_file, 'w', encoding='utf-8') as f:
    f.write("Detected 'O' Characters - Coordinates and Properties\n")
    f.write("="*80 + "\n\n")
    f.write(f"Image: {image_path}\n")
    f.write(f"Total 'O' detected: {len(o_candidates)}\n\n")

    for idx, o_stat in enumerate(o_candidates):
        x, y, w, h = o_stat['bbox']
        f.write(f"O #{idx+1}:\n")
        f.write(f"  Position: (x={x}, y={y})\n")
        f.write(f"  Size: {w}×{h} pixels\n")
        f.write(f"  Circularity: {o_stat['circularity']:.4f}\n")
        f.write(f"  Aspect Ratio: {o_stat['aspect_ratio']:.4f}\n")
        f.write(f"  Extent: {o_stat['extent']:.4f}\n")
        f.write(f"  Solidity: {o_stat['solidity']:.4f}\n")
        f.write(f"  Area: {o_stat['area']:.0f} pixels\n")
        f.write("\n")

print(f"✓ □□□□□□□□□□□□: {coords_file}")

# ========================================
# FINAL SUMMARY
# ========================================

print("\n" + "="*80)
print("□□□□□□□□□□□□□□□□□□□□")
```

Source Code (Page 9)

```
print("="*80)

print(f"""
□□□□□□□□□□□□: {image_path}
□□□□□□□□: {img.shape[1]} × {img.shape[0]} pixels

□□□□□□□□□□□□□□□□:
• Morphological Image Processing
• Connected Component Analysis
• Shape Analysis (Circularity, Aspect Ratio, Extent, Solidity)
• Hole Detection

□□□□□□□□:
• □□□□□□□□□□□□ 'O' □□□□□□□□: {len(o_candidates)} □□□
• □□□□□□□□□ contours □□□□□□□□: {len(contours)} contours

□□□□□□□□□□□□□□□:
□ {pdf_filename}
□ output/task5_processing_steps.png
□ output/task5_detection_result.png
□ output/task5_detected_O_closeup.png
□ output/task5_analysis_statistics.png
□ {coords_file}
""")

print("="*80)
print("□ Task 5 □□□□□□□□□□□□□□□!")
print("="*80)

# Show plots
plt.show()
```