LAB2
Edge, Corner Detection, and Morphological Operations

Edge, corner detection, and morphological operations are fundamental techniques in the field of image processing. These methods play a crucial role in extracting meaningful features from digital images, enabling computers to understand and interpret visual data. Edge detection identifies abrupt changes in pixel intensity, such as the boundaries between objects, while corner detection pinpoint's locations where edges intersect. Morphological operations allow for the manipulation and analysis of image structures through operations like dilation and erosion. Together, these techniques form building blocks for various image analysis and computer vision applications, from object recognition to image enhancement.

1. Edge detection:

1.1 Overview

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness (intensity gradients). Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

Why Edge Detection Matters:

- Reduces the amount of data to process while preserving structural information.
- Identifies object boundaries for segmentation.
- Essential preprocessing step for feature extraction.
- Enable shape analysis and pattern recognition.

1.2 Common Edge Detection Algorithms

1.2.1 Sobel Edge Detection

The Sobel operator uses two 3×3 kernels to calculate approximations of horizontal and vertical derivatives:

Mathematical Foundation:

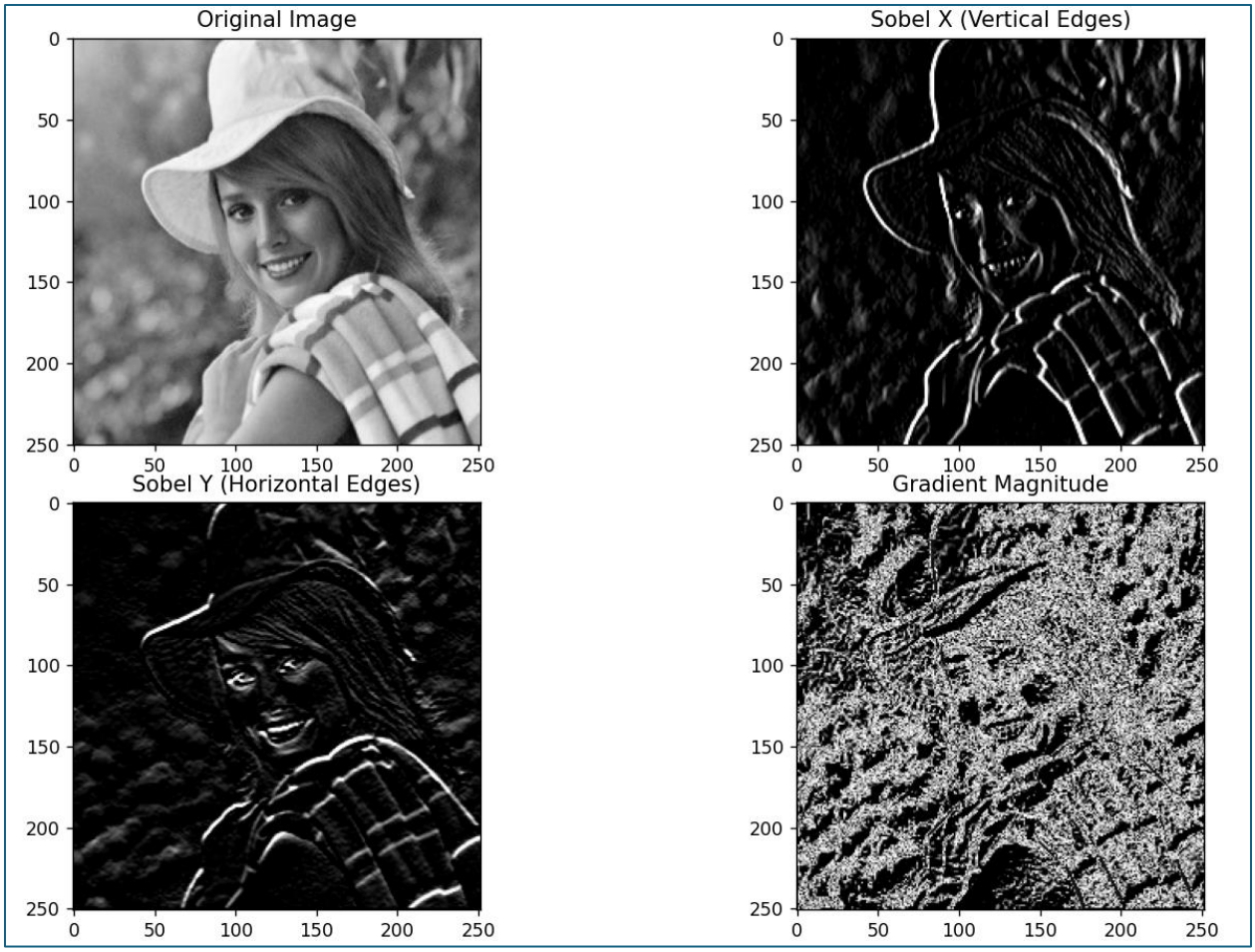
- Gx (horizontal changes): [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]].
- Gy (vertical changes): [[-1, -2, -1], [0, 0, 0], [1, 2, 1]].
- Gradient Magnitude: $G = \sqrt{(Gx^2 + Gy^2)}$.

Example 1: Sobel Filter Implementation:

- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the sample image
input_image = cv2.imread('image1.jpg', cv2.IMREAD_GRAYSCALE)

# Define the Sobel kernels
kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

# Apply the Sobel filters
sobel_x = cv2.filter2D(input_image, -1, kernel_x)
sobel_y = cv2.filter2D(input_image, -1, kernel_y)

# Calculate gradient magnitude
sobel_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
sobel_magnitude = np.uint8(sobel_magnitude)

# Combine using weighted average
sobel_combined = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)

# Display results
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes[0, 0].imshow(input_image, cmap='gray')
axes[0, 0].set_title('Original Image')
axes[0, 1].imshow(sobel_x, cmap='gray')
axes[0, 1].set_title('Sobel X (Vertical Edges)')
axes[1, 0].imshow(sobel_y, cmap='gray')
axes[1, 0].set_title('Sobel Y (Horizontal Edges)')
axes[1, 1].imshow(sobel_magnitude, cmap='gray')
axes[1, 1].set_title('Gradient Magnitude')
plt.tight_layout()
plt.show()

# Save results
cv2.imwrite('sobel_filtered_image.jpg', sobel_combined)
```

- **Output:**



1.2.2 Canny Edge Detection

Canny is a multi-stage algorithm that provides optimal edge detection:

The Five Stages of Canny:

- Noise Reduction: Apply Gaussian blur to smooth the image.
- Gradient Calculation: Find intensity gradients using Sobel.
- Non-Maximum Suppression: Thin edges to single-pixel width.
- Double Threshold: Classify edges as strong, weak, or non-edges.
- Edge Tracking by Hysteresis: Connect weak edges to strong edges.

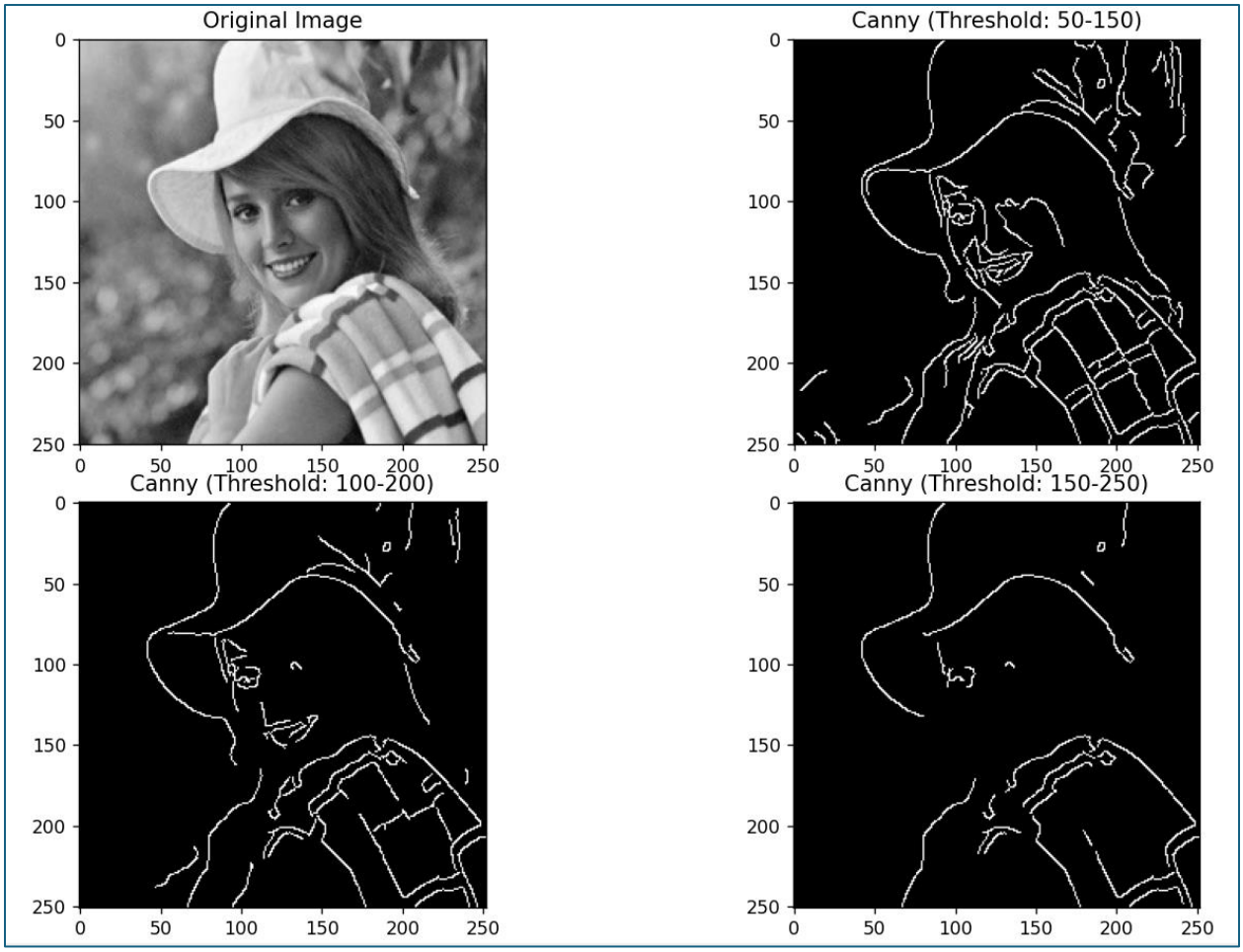
Example 2: Canny Edge Detection with Parameter Tuning:

- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the sample image
input_image = cv2.imread('image1.jpg', cv2.IMREAD_GRAYSCALE)
```

```
# Apply Gaussian blur to reduce noise
blurred = cv2.GaussianBlur(input_image, (5, 5), 1.4)

# Apply Canny edge detection with different thresholds
edges_low = cv2.Canny(blurred, 50, 150)    # Low threshold
edges_medium = cv2.Canny(blurred, 100, 200) # Medium threshold
edges_high = cv2.Canny(blurred, 150, 250)   # High threshold

# Display comparison
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes[0, 0].imshow(input_image, cmap='gray')
axes[0, 0].set_title('Original Image')
axes[0, 1].imshow(edges_low, cmap='gray')
axes[0, 1].set_title('Canny (Threshold: 50-150)')
axes[1, 0].imshow(edges_medium, cmap='gray')
axes[1, 0].set_title('Canny (Threshold: 100-200)')
axes[1, 1].imshow(edges_high, cmap='gray')
axes[1, 1].set_title('Canny (Threshold: 150-250)')
plt.tight_layout()
plt.show()

cv2.imwrite('canny_edge_detection_image.jpg', edges_medium)
```

- **Output:**



Understanding Thresholds:

- maxVal: Edges with gradient > maxVal are definite edges (strong).
- minVal: Edges with gradient < minVal are discarded (non-edges).
- Between thresholds: Edges classified based on connectivity to strong edges.

1.2.3 Prewitt Edge Detection

Similar to Sobel but with simpler kernels:

Example 3: Prewitt Edge Detection Implementation:

- **Code:**

```
import cv2
import numpy as np

# Load the image in grayscale
image = cv2.imread('image1.jpg', cv2.IMREAD_GRAYSCALE)

# Check if the image loaded correctly
if image is None:
    print("Error: Could not load image. Check the file path.")
```

```
    exit()

# Define Prewitt kernels
kernel_x = np.array([[-1, 0, 1],
                     [-1, 0, 1],
                     [-1, 0, 1]])

kernel_y = np.array([[-1, -1, -1],
                     [ 0,  0,  0],
                     [ 1,  1,  1]])

# Apply Prewitt filters
prewitt_x = cv2.filter2D(image, -1, kernel_x)
prewitt_y = cv2.filter2D(image, -1, kernel_y)

# Combine the two gradient results
prewitt_combined = cv2.addWeighted(prewitt_x, 0.5, prewitt_y, 0.5, 0)

# Stack original and output images horizontally for comparison
comparison = np.hstack((image, prewitt_combined))

# Display results
cv2.imshow('Original (Left) vs Prewitt Edge Detection (Right)',
comparison)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Output:**

1.2.4 Laplacian Edge Detection

Detects edges using second-order derivatives:

Example 4: Laplacian Edge Detection Implementation:

- **Code:**

```
import cv2
import numpy as np

# Load image in grayscale
image = cv2.imread('image1.jpg', cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded successfully
if image is None:
    print("Error: Could not load image. Check the file path.")
    exit()

# Apply Gaussian blur to reduce noise
blurred = cv2.GaussianBlur(image, (3, 3), 0)

# Apply Laplacian operator
laplacian = cv2.Laplacian(blurred, cv2.CV_64F)
laplacian = np.uint8(np.absolute(laplacian))

# Combine original and Laplacian output side by side
comparison = np.hstack((image, laplacian))

# Display both images
cv2.imshow('Original (Left) vs Laplacian (Right)', comparison)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Output:**

1.3 Applications for Edge Detection

- **Application 1: Medical Imaging - Tumor Detection:**

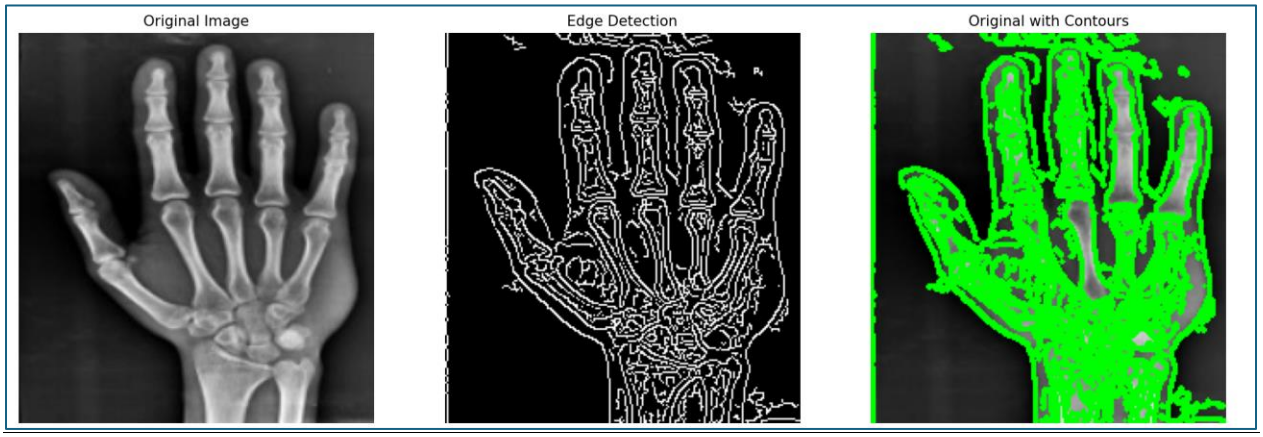
Example 5: Medical Imaging - Tumor Detection Implementation:

- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def detect_medical_boundaries(image_path):
    """
    Detect boundaries in medical images using edge detection and contour
finding.
    """
    # Load medical image
    medical_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Apply CLAHE for contrast enhancement
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    enhanced = clahe.apply(medical_image)

    # Apply Canny edge detection
    edges = cv2.Canny(enhanced, 30, 100)

    # Find contours
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Draw contours on original image
    result = cv2.cvtColor(medical_image, cv2.COLOR_GRAY2BGR)
```

```
    cv2.drawContours(result, contours, -1, (0, 255, 0), 2)

    return medical_image, result, edges

# Image path
image_path = 'example5.png'#'example5_1.png'

# Detect boundaries
original, result, edges = detect_medical_boundaries(image_path)

# ---- Show all three images in one figure ----
plt.figure(figsize=(15, 5))

# Original
plt.subplot(1, 3, 1)
plt.imshow(original, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Edge detection
plt.subplot(1, 3, 2)
plt.imshow(edges, cmap='gray')
plt.title('Edge Detection')
plt.axis('off')

# Contours on original
plt.subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.title('Original with Contours')
plt.axis('off')

plt.tight_layout()
plt.show()
# ----------------------------------------

print("Processing complete!")
```

- **Output:**



- **Application 2: Autonomous Vehicles - Lane Detection:**

Example 6: Autonomous Vehicles - Lane Detection Implementation:

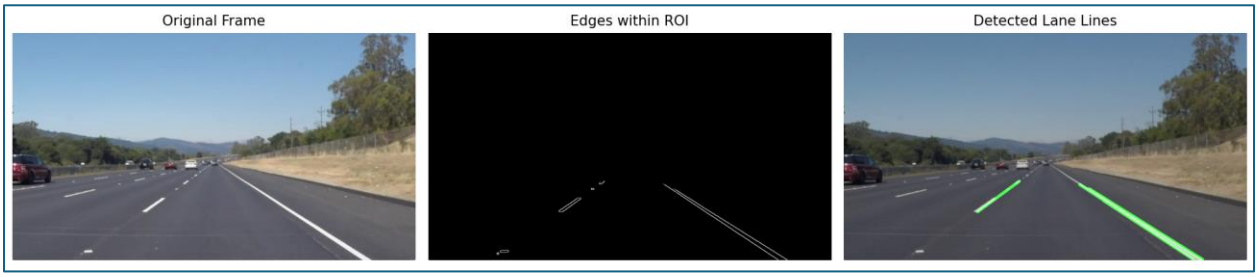
- **Code:**

```
import cv2
import numpy as np

def detect_lane_lines(video_frame):
    """
    Detect lane lines in road images for autonomous driving systems.
    Uses edge detection, masking, and Hough Line Transform.
    """
    # ---- Step 1: Preprocessing ----
    # Convert to grayscale
    gray = cv2.cvtColor(video_frame, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur to reduce noise
    blur = cv2.GaussianBlur(gray, (5, 5), 0)

    # ---- Step 2: Edge Detection ----
    # Use Canny to find edges (tune thresholds if needed)
    edges = cv2.Canny(blur, 50, 150)
```

```
    # ---- Step 3: Region of Interest (ROI) Mask ----
    height, width = edges.shape
    mask = np.zeros_like(edges)

    # Define a triangular ROI focusing on the road ahead
    polygon = np.array([[
        (0, height),
        (width // 2, int(height * 0.6)),
        (width, height)
    ]], np.int32)

    cv2.fillPoly(mask, polygon, 255)
    masked_edges = cv2.bitwise_and(edges, mask)

    # ---- Step 4: Line Detection using Hough Transform ----
    lines = cv2.HoughLinesP(
        masked_edges,
        rho=1,
        theta=np.pi / 180,
        threshold=50,
        minLineLength=100,
        maxLineGap=50
    )

    # ---- Step 5: Draw Detected Lines ----
    line_image = np.zeros_like(video_frame)
    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            cv2.line(line_image, (x1, y1), (x2, y2), (0, 255, 0), 5)

    # ---- Step 6: Combine with Original Frame ----
    result = cv2.addWeighted(video_frame, 0.8, line_image, 1, 0)

    return edges, masked_edges, result

import matplotlib.pyplot as plt

# Example: load a single frame or image
frame = cv2.imread('example6.png')
if frame is None:
    raise FileNotFoundError("Could not load the image 'example6.png'")

edges, masked_edges, result = detect_lane_lines(frame)
```

```
# ---- Display Results ----
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
plt.title('Original Frame')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(masked_edges, cmap='gray')
plt.title('Edges within ROI')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.title('Detected Lane Lines')
plt.axis('off')

plt.tight_layout()
plt.show()
print("Lane detection complete!")
```

- **Output:**



- **Application 3: Industrial Quality Control:**

Example 7: Industrial Quality Control Implementation:

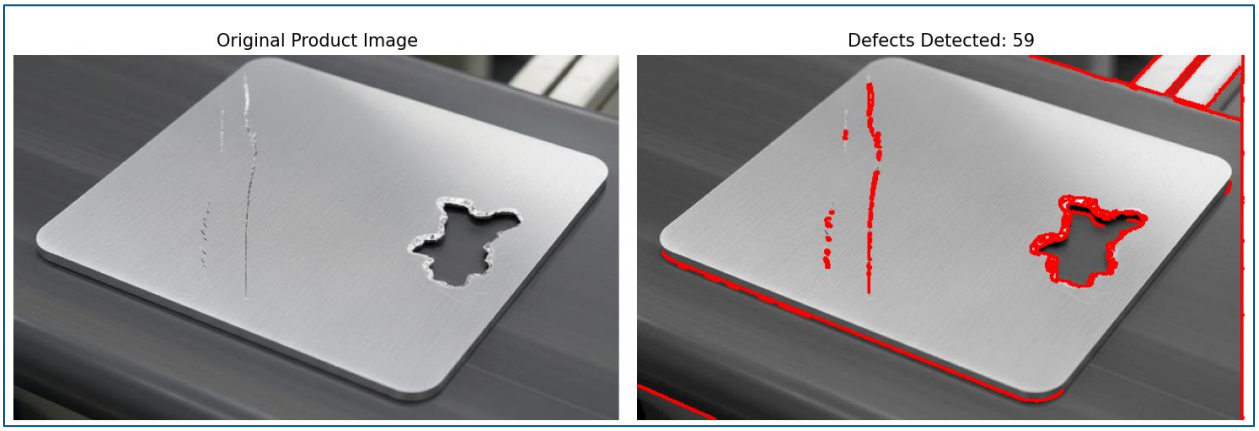
- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def inspect_product_defects(product_image_path):
    """
    Detect defects in manufactured products on assembly lines
    """
    # Load product image
```

```
    product = cv2.imread(product_image_path, cv2.IMREAD_GRAYSCALE)

    if product is None:
        raise ValueError(f"Could not load image from
{product_image_path}")

    # Apply edge detection
    edges = cv2.Canny(product, 100, 200)

    # Find contours
    contours, _ = cv2.findContours(edges, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    # Analyze contours for defects
    defects = []
    for contour in contours:
        area = cv2.contourArea(contour)
        perimeter = cv2.arcLength(contour, True)

        # Check for irregular shapes (potential defects)
        if perimeter > 0:
            circularity = 4 * np.pi * area / (perimeter ** 2)
            if circularity < 0.5:  # Irregular shape
                defects.append(contour)

    # Mark defects
    result = cv2.cvtColor(product, cv2.COLOR_GRAY2BGR)
    cv2.drawContours(result, defects, -1, (0, 0, 255), 2)

    return result, len(defects)

# Run the defect detection
result_image, defect_count = inspect_product_defects('example7.png')

# Display the results
plt.figure(figsize=(12, 6))

# Original image
plt.subplot(1, 2, 1)
original = cv2.imread('example7.png')
original_rgb = cv2.cvtColor(original, cv2.COLOR_BGR2RGB)
plt.imshow(original_rgb)
plt.title('Original Product Image')
plt.axis('off')
```

```
# Result with defects marked
plt.subplot(1, 2, 2)
result_rgb = cv2.cvtColor(result_image, cv2.COLOR_BGR2RGB)
plt.imshow(result_rgb)
plt.title(f'Defects Detected: {defect_count}')
plt.axis('off')

plt.tight_layout()
plt.show()

print(f"\nAnalysis complete!")
print(f"Number of defects detected: {defect_count}")
```

- **Output:**



- **Application 4: Document Scanning and OCR:**

Example 8: Document Scanning and OCR Implementation:

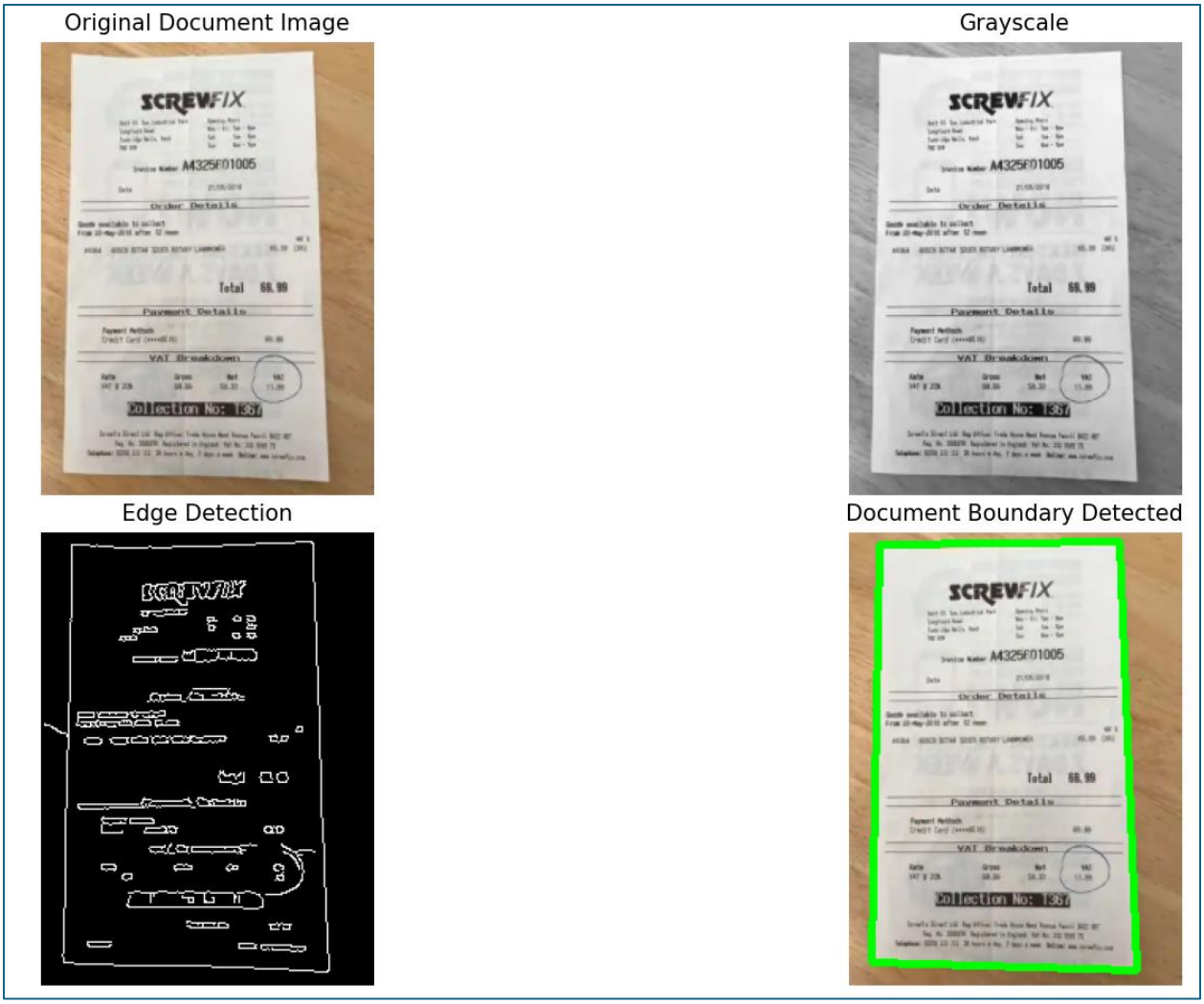
- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def preprocess_document_for_ocr(document_image_path):
    """
    Detect document edges for perspective correction and OCR
    """
    # Load image
    image = cv2.imread(document_image_path)

    if image is None:
        raise ValueError(f"Could not load image from
{document_image_path}")
```

```
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Blur and edge detection
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blur, 75, 200)

    # Find contours
    contours, _ = cv2.findContours(edges, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)

    # Find largest rectangular contour (document boundary)
    contours = sorted(contours, key=cv2.contourArea, reverse=True)[:5]

    document_contour = None
    for contour in contours:
        peri = cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, 0.02 * peri, True)

        if len(approx) == 4:  # Found rectangle
            document_contour = approx
            break

    # Draw detected document boundary
    result = image.copy()
    if document_contour is not None:
        cv2.drawContours(result, [document_contour], -1, (0, 255, 0), 3)
    else:
        print("Warning: No rectangular document boundary detected")

    return result, document_contour

# Run the document detection
result_image, document_contour =
preprocess_document_for_ocr('example8.png')

# Display the results
plt.figure(figsize=(15, 10))

# Original image
plt.subplot(2, 2, 1)
original = cv2.imread('example8.png')
original_rgb = cv2.cvtColor(original, cv2.COLOR_BGR2RGB)
plt.imshow(original_rgb)
plt.title('Original Document Image')
plt.axis('off')
```

```
# Grayscale
plt.subplot(2, 2, 2)
gray = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)
plt.imshow(gray, cmap='gray')
plt.title('Grayscale')
plt.axis('off')

# Edge detection
plt.subplot(2, 2, 3)
blur = cv2.GaussianBlur(gray, (5, 5), 0)
edges = cv2.Canny(blur, 75, 200)
plt.imshow(edges, cmap='gray')
plt.title('Edge Detection')
plt.axis('off')

# Result with document boundary detected
plt.subplot(2, 2, 4)
result_rgb = cv2.cvtColor(result_image, cv2.COLOR_BGR2RGB)
plt.imshow(result_rgb)
plt.title('Document Boundary Detected')
plt.axis('off')

plt.tight_layout()
plt.show()

print(f"\n✓ Document scanning complete!")
if document_contour is not None:
    print(f"Document boundary detected with 4 corners")
    print(f"Corner coordinates:")
    for i, point in enumerate(document_contour):
        print(f"   Corner {i+1}: {point[0]}")
else:
    print("No rectangular document boundary found")
```

- **Output:**
  ✓ Document scanning complete!
  Document boundary detected with 4 corners
  Corner coordinates:
  - Corner 1: [171 6]
  - Corner 2: [19 8]
  - Corner 3: [14 272]
  - Corner 4: [182 277]



2. Corner Detection

2.1 Overview

Corner detection identifies points in an image where the intensity changes significantly in multiple directions. Corners are robust features that remain stable under various transformations (rotation, scaling, illumination changes).

Why Corners Matter:

- More distinctive than edges alone.
- Robust to rotation and illumination changes.
- Efficient for image matching and tracking.
- Crucial for 3D reconstruction.

2.2 Harris Corner Detection

The Harris detector analyzes the autocorrelation matrix of local intensity gradients:

$$M = \Sigma(\text{window}) \begin{bmatrix} I_x^2 & I_x \cdot I_y \\ I_x \cdot I_y & I_y^2 \end{bmatrix}$$

Corner Response: $R = det(M) - k \cdot trace(M)^2$

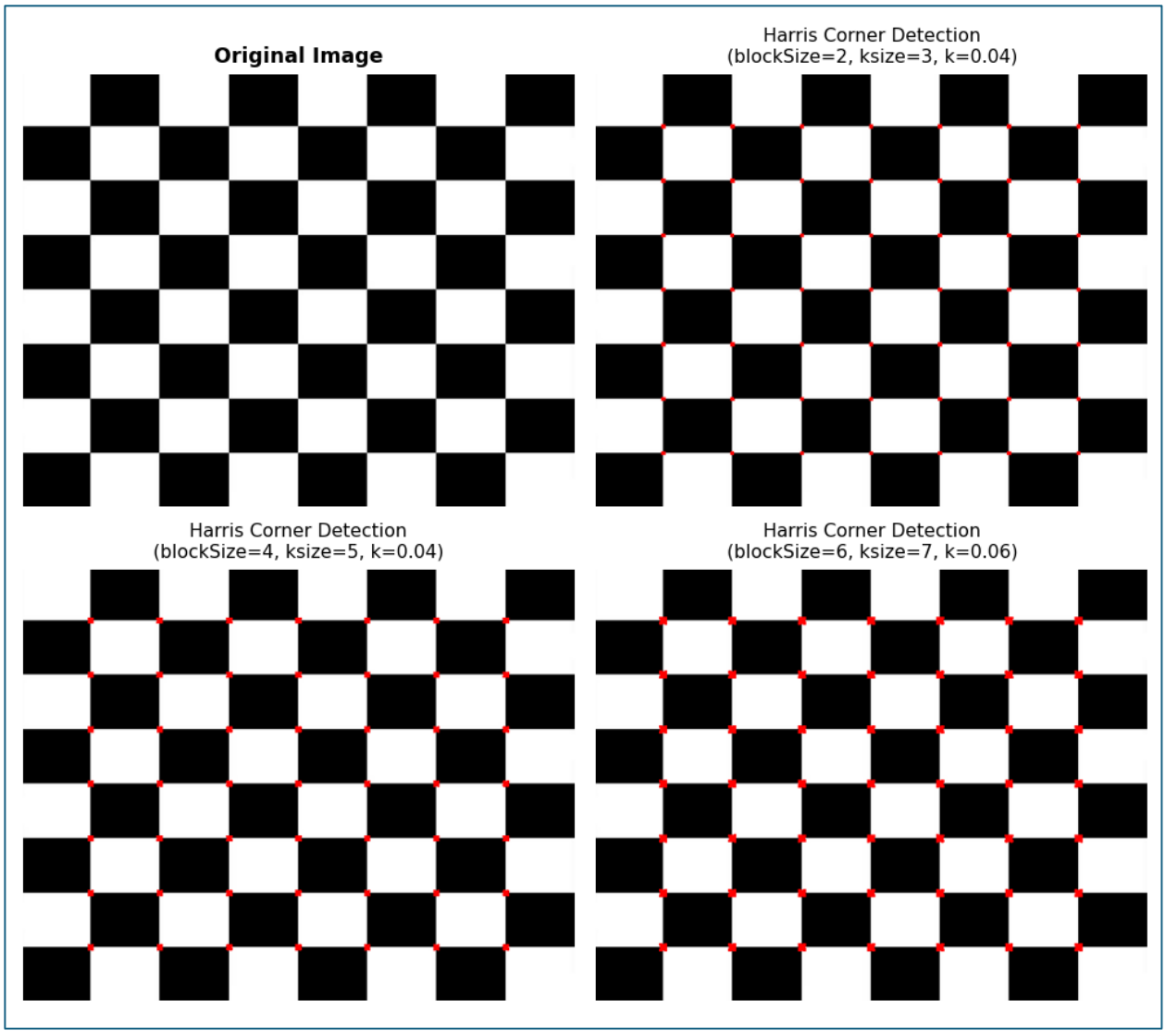
Example9: Harris Corner Detection with Parameters:

- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load an image
image = cv2.imread('image2.jpg', cv2.IMREAD_COLOR)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)

# Harris Corner Detection with different parameters
corner_detector1 = cv2.cornerHarris(gray, blockSize=2, ksize=3, k=0.04)
corner_detector2 = cv2.cornerHarris(gray, blockSize=4, ksize=5, k=0.04)
corner_detector3 = cv2.cornerHarris(gray, blockSize=6, ksize=7, k=0.06)

# Dilate corner image to enhance corner points
corner_detector1 = cv2.dilate(corner_detector1, None)
corner_detector2 = cv2.dilate(corner_detector2, None)
corner_detector3 = cv2.dilate(corner_detector3, None)

# Mark corners
image1 = image.copy()
image2 = image.copy()
image3 = image.copy()

image1[corner_detector1 > 0.01 * corner_detector1.max()] = [0, 0, 255]
image2[corner_detector2 > 0.01 * corner_detector2.max()] = [0, 0, 255]
image3[corner_detector3 > 0.01 * corner_detector3.max()] = [0, 0, 255]

# Display with reduced figure size
fig, axes = plt.subplots(2, 2, figsize=(8, 7))
axes[0, 0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axes[0, 0].set_title('Original Image', fontsize=10, fontweight='bold')
axes[0, 1].imshow(cv2.cvtColor(image1, cv2.COLOR_BGR2RGB))
axes[0, 1].set_title('Harris Corner Detection\n(blockSize=2, ksize=3,
k=0.04)', fontsize=9)
axes[1, 0].imshow(cv2.cvtColor(image2, cv2.COLOR_BGR2RGB))
axes[1, 0].set_title('Harris Corner Detection\n(blockSize=4, ksize=5,
k=0.04)', fontsize=9)
axes[1, 1].imshow(cv2.cvtColor(image3, cv2.COLOR_BGR2RGB))
axes[1, 1].set_title('Harris Corner Detection\n(blockSize=6, ksize=7,
k=0.06)', fontsize=9)
```

```
# Remove axis ticks for cleaner display
for ax in axes.flat:
    ax.axis('off')

plt.tight_layout()
plt.show()
```

- **Output:**



2.3 Shi-Tomasi Corner Detection

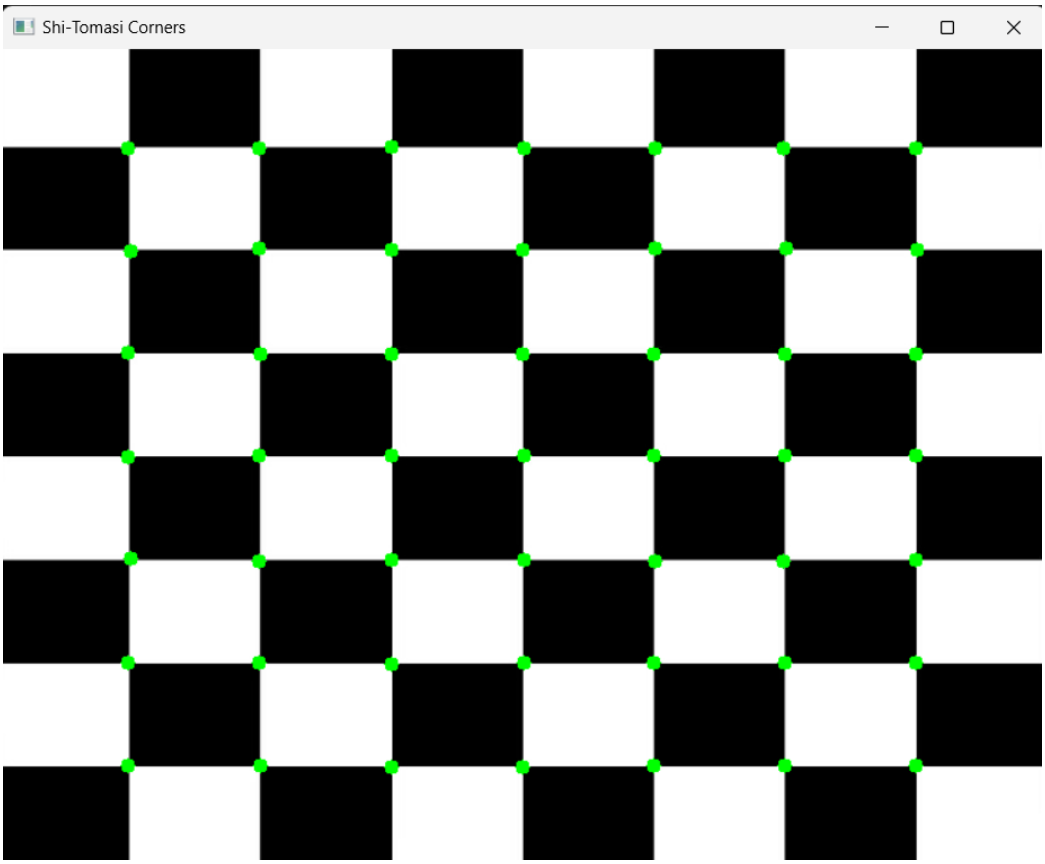
An improvement over Harris, using only the minimum eigenvalue:

Example10: Shi-Tomasi Corner Detection:

- **Code:**

```
import cv2
import numpy as np

# Load image
image = cv2.imread('image2.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Shi-Tomasi Corner Detection
corners = cv2.goodFeaturesToTrack(gray, maxCorners=100, qualityLevel=0.01,
                                  minDistance=10, blockSize=3)
# Draw corners
corners = np.int0(corners)
for corner in corners:
    x, y = corner.ravel()
    cv2.circle(image, (x, y), 5, (0, 255, 0), -1)

cv2.imshow('Shi-Tomasi Corners', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Output:**



2.4 Applications for Edge Detection

- **Application 1: Image Stitching for Panoramas:**

Example 11: Image Stitching for Panoramas Implementation:

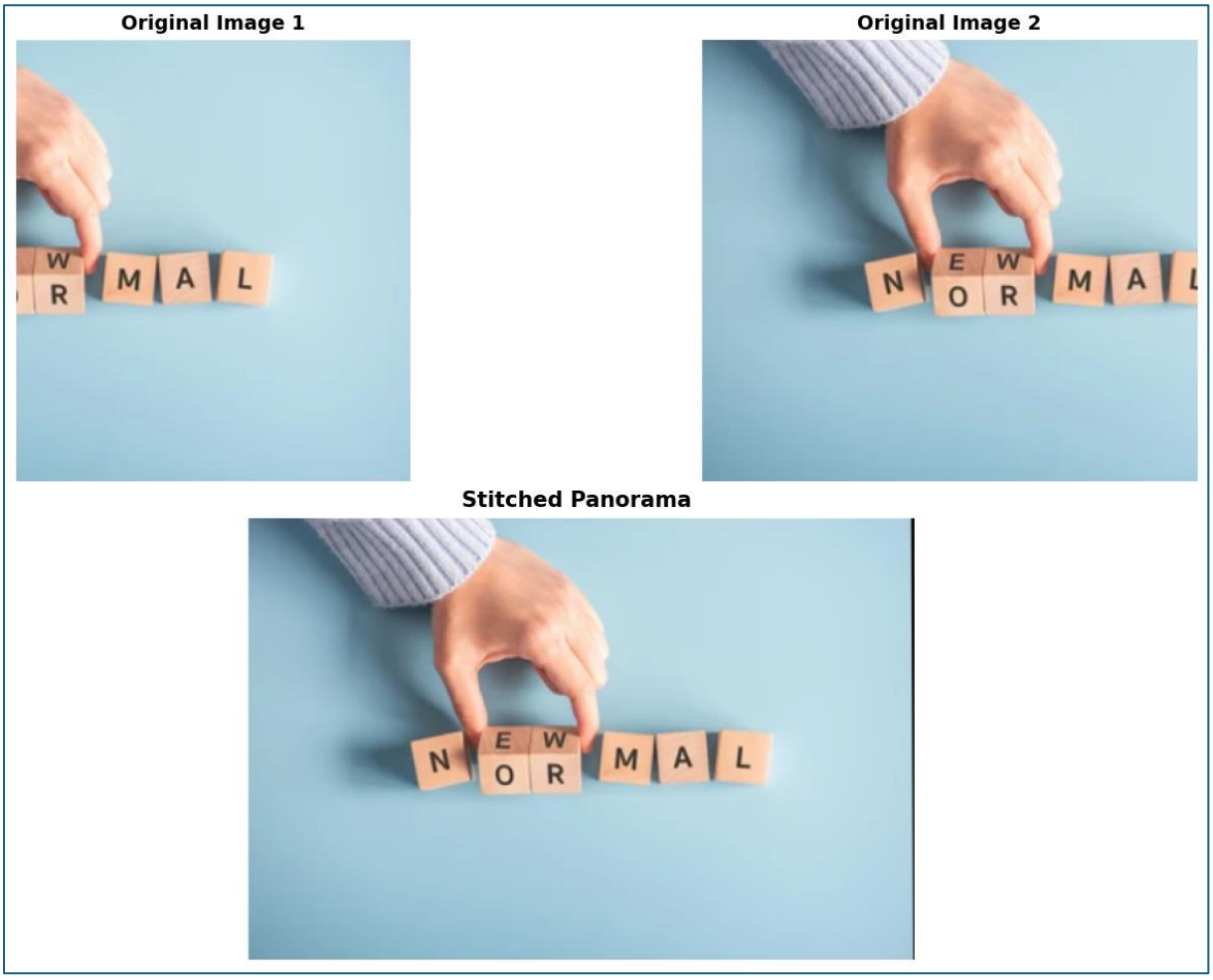
- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def create_panorama(image1_path, image2_path):
    """
    Stitch multiple images together using corner/feature detection
    """
    # Load images
    img1 = cv2.imread(image1_path)
    img2 = cv2.imread(image2_path)

    # Convert to grayscale
    gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    # Detect ORB keypoints and descriptors
    orb = cv2.ORB_create(nfeatures=2000)
    kp1, des1 = orb.detectAndCompute(gray1, None)
    kp2, des2 = orb.detectAndCompute(gray2, None)

    # Match features
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key=lambda x: x.distance)

    # Extract matched keypoints
    src_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-
1, 1, 2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-
1, 1, 2)

    # Find homography
    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

    # Warp and stitch
    height, width = img2.shape[:2]
    result = cv2.warpPerspective(img1, M, (width + img1.shape[1], height))
    result[0:height, 0:width] = img2
```

```
    # Crop black borders
    result = crop_black_borders(result)

    return result

def crop_black_borders(image):
    """
    Remove black borders from panorama
    """
    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Threshold to find non-black regions
    _, thresh = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)

    # Find contours
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Get bounding box of the largest contour
    if contours:
        x, y, w, h = cv2.boundingRect(contours[0])
        cropped = image[y:y+h, x:x+w]
        return cropped

    return image

# Call the function with your image paths
image1_path = 'R1.png'
image2_path = 'L1.png'

# Load original images for display
img1 = cv2.imread(image1_path)
img2 = cv2.imread(image2_path)

# Create panorama
panorama = create_panorama(image1_path, image2_path)

# Visualize the results
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Display original images
axes[0, 0].imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
axes[0, 0].set_title('Original Image 1', fontsize=11, fontweight='bold')
axes[0, 0].axis('off')
```

```
axes[0, 1].imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
axes[0, 1].set_title('Original Image 2', fontsize=11, fontweight='bold')
axes[0, 1].axis('off')

# Display panorama (spanning bottom row)
axes[1, 0].remove()
axes[1, 1].remove()
ax_panorama = fig.add_subplot(2, 1, 2)
ax_panorama.imshow(cv2.cvtColor(panorama, cv2.COLOR_BGR2RGB))
ax_panorama.set_title('Stitched Panorama ', fontsize=12,
fontweight='bold')
ax_panorama.axis('off')

plt.tight_layout()
plt.show()
```

- **Output:**



- **Application 2: 3D Reconstruction and Structure from Motion:**

Example 12: 3D Reconstruction and Structure from Motion Implementation:

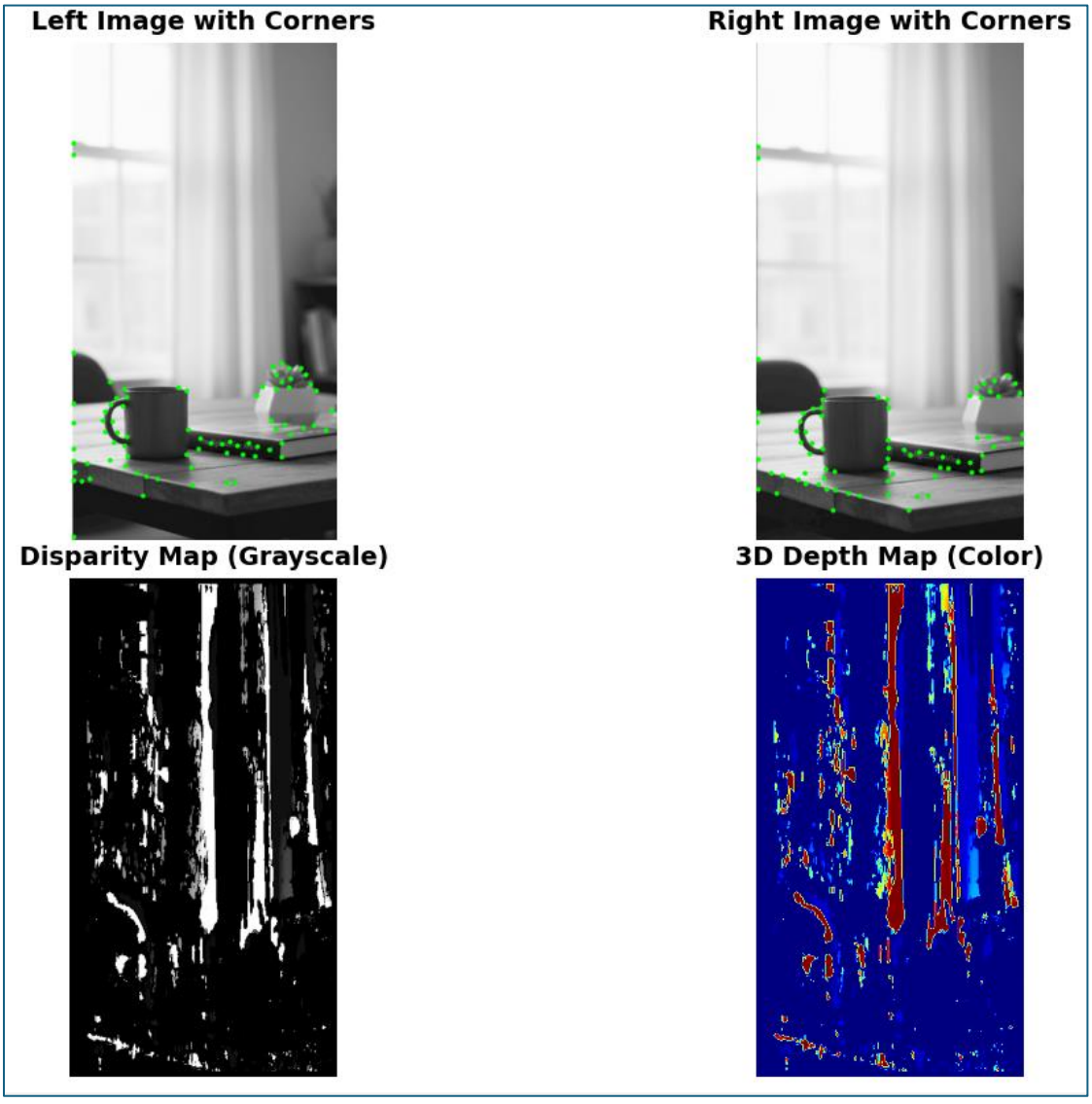
- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def reconstruct_3d_from_stereo(left_image_path, right_image_path):
    """
    Use corner matching in stereo images for 3D reconstruction
    """
    # Load stereo pair
    img_left = cv2.imread(left_image_path, cv2.IMREAD_GRAYSCALE)
    img_right = cv2.imread(right_image_path, cv2.IMREAD_GRAYSCALE)

    # Detect corners in both images
    corners_left = cv2.goodFeaturesToTrack(img_left, maxCorners=500,
                                           qualityLevel=0.01,
minDistance=10)
    corners_right = cv2.goodFeaturesToTrack(img_right, maxCorners=500,
                                            qualityLevel=0.01,
minDistance=10)

    # Match corners between images
    # Calculate disparity
    # Compute depth map
    stereo = cv2.StereoBM_create(numDisparities=16, blockSize=15)
    disparity = stereo.compute(img_left, img_right)

    return disparity, corners_left, corners_right, img_left, img_right

# Load stereo image pair
left_image_path = 'left_image.png'
right_image_path = 'right_image.png'

# Call the function
disparity, corners_left, corners_right, img_left, img_right =
reconstruct_3d_from_stereo(
    left_image_path, right_image_path
)

# Draw corners on images for visualization
img_left_corners = cv2.cvtColor(img_left.copy(), cv2.COLOR_GRAY2BGR)
img_right_corners = cv2.cvtColor(img_right.copy(), cv2.COLOR_GRAY2BGR)
```

```
if corners_left is not None:
    for corner in corners_left:
        x, y = corner.ravel()
        cv2.circle(img_left_corners, (int(x), int(y)), 3, (0, 255, 0), -1)

if corners_right is not None:
    for corner in corners_right:
        x, y = corner.ravel()
        cv2.circle(img_right_corners, (int(x), int(y)), 3, (0, 255, 0), -
1)

# Normalize disparity for better visualization
disparity_normalized = cv2.normalize(disparity, None, 0, 255,
cv2.NORM_MINMAX, cv2.CV_8U)

# Visualize results
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Original left image with corners
axes[0, 0].imshow(img_left_corners)
axes[0, 0].set_title('Left Image with Corners', fontsize=11,
fontweight='bold')
axes[0, 0].axis('off')

# Original right image with corners
axes[0, 1].imshow(img_right_corners)
axes[0, 1].set_title('Right Image with Corners', fontsize=11,
fontweight='bold')
axes[0, 1].axis('off')

# Disparity map (grayscale)
axes[1, 0].imshow(disparity_normalized, cmap='gray')
axes[1, 0].set_title('Disparity Map (Grayscale)', fontsize=11,
fontweight='bold')
axes[1, 0].axis('off')

# Disparity map (color/depth)
axes[1, 1].imshow(disparity_normalized, cmap='jet')
axes[1, 1].set_title('3D Depth Map (Color)', fontsize=11,
fontweight='bold')
axes[1, 1].axis('off')

plt.tight_layout()
plt.show()
```

```
# Optional: Save results
cv2.imwrite('disparity_map.jpg', disparity_normalized)
cv2.imwrite('left_corners.jpg', img_left_corners)
cv2.imwrite('right_corners.jpg', img_right_corners)
print("Results saved!")
print(f"Detected {len(corners_left) if corners_left is not None else 0}
corners in left image")
print(f"Detected {len(corners_right) if corners_right is not None else 0}
corners in right image")
```

- **Output:**
  Detected 78 corners in left image
  Detected 77 corners in right image



3. Morphological Operations

3.1 Overview

Morphological operations are image processing techniques that work on the shape and structure of objects in an image. They process images based on shapes using a structuring element (kernel). These operations are particularly useful for binary images but can also be applied to grayscale images.

Key Concepts:

- Structuring Element: A small binary image (kernel) that defines the neighborhood.
- Foreground: White pixels (255) representing objects.
- Background: Black pixels (0).

3.2 Morphological Operations

Morphology is a broad set of image processing operations that process images based on shapes. In a morphological operation, each pixel in the image is adjusted based on the value of other pixels in its neighborhood. By choosing the size and shape of the neighborhood, you can construct a morphological operation that is sensitive to specific shapes in the input image. It is a image processing techniques that work on the shape and structure of objects in an image. They are particularly useful for tasks like noise reduction, object extraction, and shape analysis:

3.2.1 Erosion:

Erosion shrinks or thins the object, and it is used to erode the boundaries of the foreground object in a binary image. It works by sliding a kernel over the image, if all pixel values under the kernel are 1, the center pixel is set to 1; otherwise, it is set to 0.

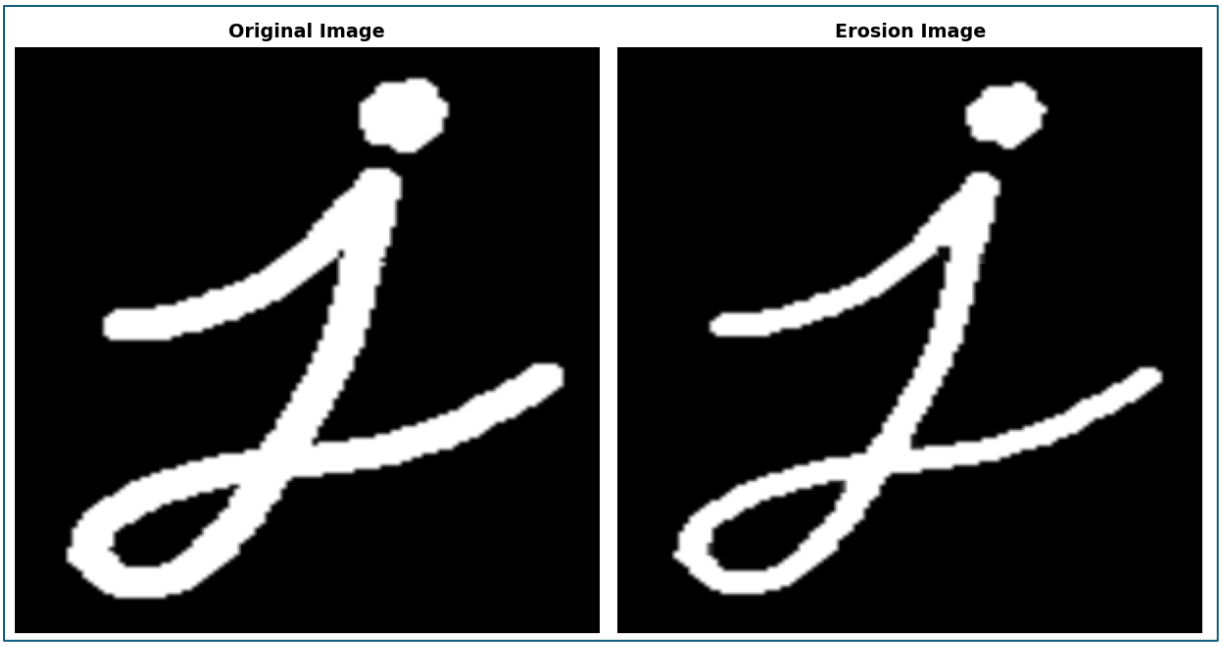
Example 13: Erosion Implementation:

- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load a binary image
image = cv2.imread('Erosion.png', 0)

# Define the new width and height
new_width = 300
new_height = 300

# Resize the image
image = cv2.resize(image, (new_width, new_height))
```

```
# Define a kernel
kernel = np.ones((5, 5), np.uint8)

# Erode the image
eroded_image = cv2.erode(image, kernel, iterations=1)

# Display both images in one plot
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image', fontsize=11, fontweight='bold')
axes[0].axis('off')

axes[1].imshow(eroded_image, cmap='gray')
axes[1].set_title('Erosion Image', fontsize=11, fontweight='bold')
axes[1].axis('off')

plt.tight_layout()
plt.show()
```

- **Output:**



3.2.2 Dilation:

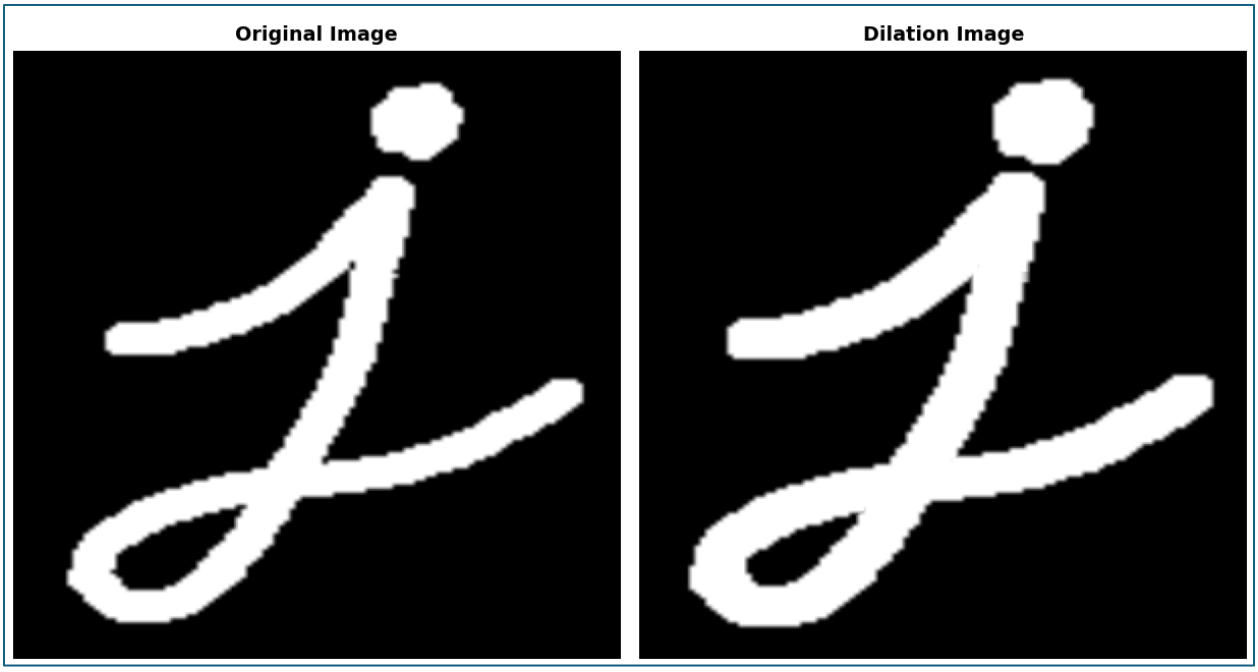
Dilation thickens the object, and it is used to expand the boundaries of the foreground object in a binary image. Like erosion, it works by sliding a kernel over the image, if any pixel under the kernel is 1, the center pixel is set to 1.

Example 14: Dilation Implementation:

- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load a binary image
image = cv2.imread('Input_Image.png', 0)

# Define the new width and height
new_width = 300
new_height = 300

# Resize the image
image = cv2.resize(image, (new_width, new_height))

# Define a kernel
kernel = np.ones((5, 5), np.uint8)

# Dilate the image
dilated_image = cv2.dilate(image, kernel, iterations=1)

# Display both images in one plot
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image', fontsize=11, fontweight='bold')
axes[0].axis('off')

axes[1].imshow(dilated_image, cmap='gray')
axes[1].set_title('Dilation Image', fontsize=11, fontweight='bold')
axes[1].axis('off')

plt.tight_layout()
plt.show()
```

- **Output:**



3.2.3 Opening:

Opening is an operation that combines erosion and dilation, and it is used to remove noise, small objects, or small details in the background. It first erodes the image and then dilates the eroded image.

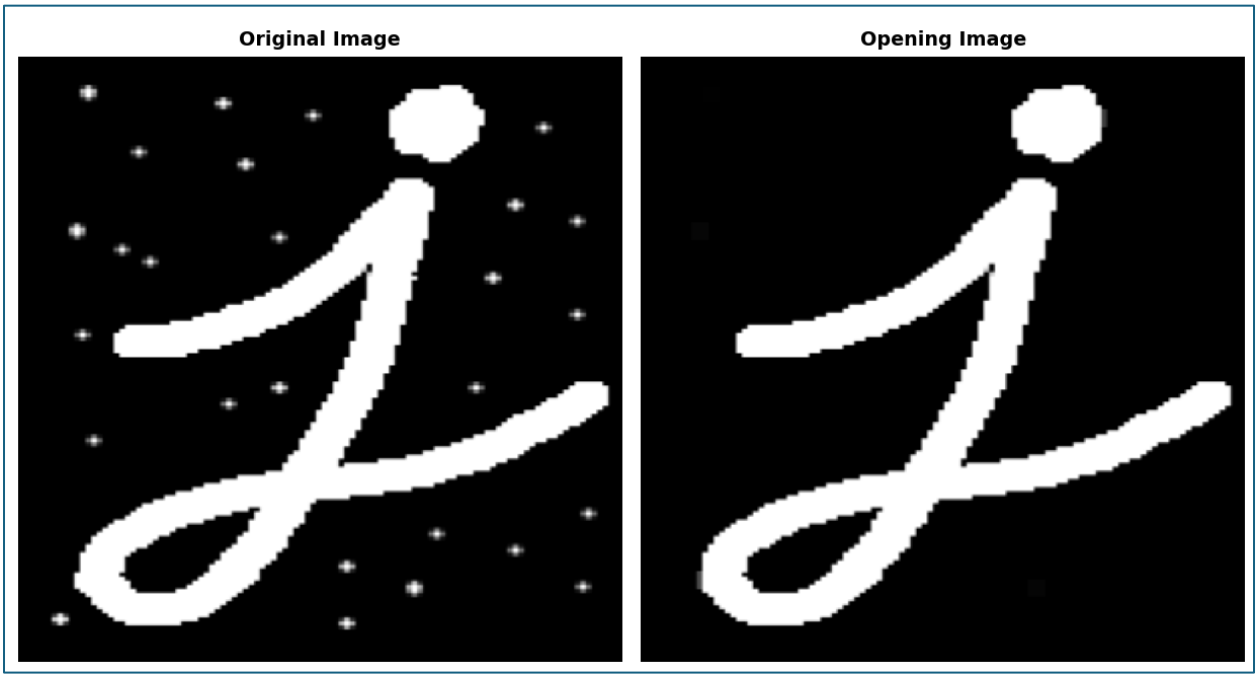
Example 15: Opening Implementation:

- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load a binary image
image = cv2.imread('opening.png', 0)

# Define the new width and height
new_width = 300
new_height = 300

# Resize the image
image = cv2.resize(image, (new_width, new_height))

# Define a kernel
kernel = np.ones((9, 9), np.uint8)

# Perform opening
```

```
opened_image = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)

# Display both images in one plot
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image', fontsize=11, fontweight='bold')
axes[0].axis('off')

axes[1].imshow(opened_image, cmap='gray')
axes[1].set_title('Opening Image', fontsize=11, fontweight='bold')
axes[1].axis('off')

plt.tight_layout()
plt.show()
```

- **Output:**



3.2.4 Closing:

Closing is an operation that combines dilation and erosion, and it is used to close small holes or gaps in the foreground object. It first dilates the image and then erodes the dilated image.

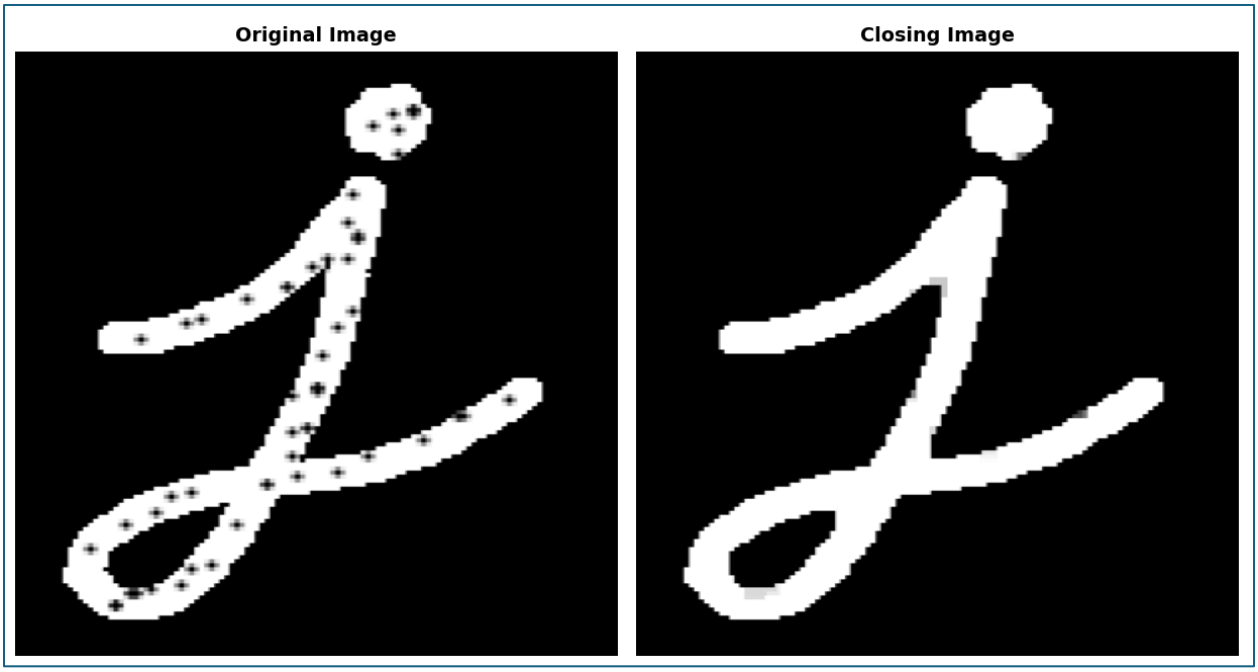
Example 16: Closing Implementation:

- **Code:**

```
import cv2
import numpy as np
```

```
import matplotlib.pyplot as plt

# Load a binary image
image = cv2.imread('closing.png', 0)

# Define the new width and height
new_width = 300
new_height = 300

# Resize the image
image = cv2.resize(image, (new_width, new_height))

# Define a kernel
kernel = np.ones((9, 9), np.uint8)

# Perform closing
closed_image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)

# Display both images in one plot
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image', fontsize=11, fontweight='bold')
axes[0].axis('off')

axes[1].imshow(closed_image, cmap='gray')
axes[1].set_title('Closing Image', fontsize=11, fontweight='bold')
axes[1].axis('off')

plt.tight_layout()
plt.show()
```

- **Output:**



3.2.5 Morphological Gradient:

The morphological gradient is the difference between the dilation and erosion of an image. It highlights the boundaries of objects in the image.

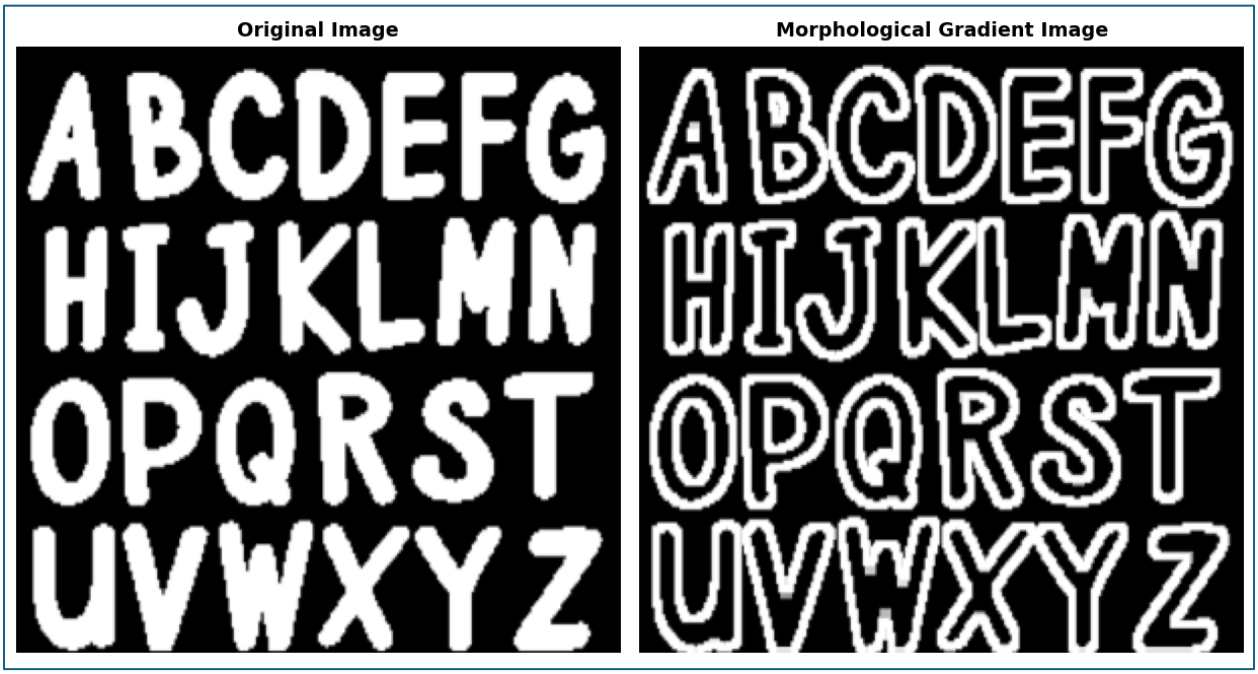
Example 17: Morphological Gradient Implementation:

- **Code:**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load a binary image
image = cv2.imread('Morphological Gradient Image.png', 0)

# Define the new width and height
new_width = 300
new_height = 300

# Resize the image
image = cv2.resize(image, (new_width, new_height))

# Define a kernel
kernel = np.ones((5, 5), np.uint8)

# Calculate the morphological gradient
gradient_image = cv2.morphologyEx(image, cv2.MORPH_GRADIENT, kernel)
```

```
# Display both images in one plot
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image', fontsize=11, fontweight='bold')
axes[0].axis('off')

axes[1].imshow(gradient_image, cmap='gray')
axes[1].set_title('Morphological Gradient Image', fontsize=11,
fontweight='bold')
axes[1].axis('off')

plt.tight_layout()
plt.show()
```

- **Output:**



4. Exercises

Task1: Edge Detection

Objective:

The objective of this task is to perform edge detection using the different filters.

Steps:

- Load a grayscale image of your choice.
- Apply a Sobel filter to the image to detect edges. You can use libraries like OpenCV or SciPy for this.
- Display the original image and the edges detected using the Sobel filter side by side.
- Try applying different edge detection filters such as the Prewitt filter or the Scharr filter and compare the results.

Task2: Corner Detection

Objective:

The objective of this task is to perform corner detection using the Harris Corner Detection method.

Steps:

- Load a grayscale image of your choice.
- Apply the Harris Corner Detection method to detect corners. You can use libraries like OpenCV or SciPy for this.
- Mark the detected corners in the image to visualize them.
- Experiment with different parameters of the Harris Corner Detection method and observe how they affect corner detection.
- You can adjust parameters such as the block size, aperture (kernel) size, and the constant k in the Harris Corner Detection method to see their impact on corner detection.

Task3: Morphological Operations for Image Enhancement

Objective:

The objective of this task is to perform image enhancement using morphological operations.

Steps:

- Load a grayscale image of your choice.
- Apply morphological operations, such as erosion and dilation, to the image to enhance specific features.
- Display the original and enhanced images side by side.
- Experiment with different structuring elements and parameters in the morphological operations to observe their impact on image enhancement.

- You can adjust parameters like the size and shape of the structuring element and the number of iterations to see how they affect the enhancement of specific features in the image.