深 圳 大 学 实 验 报 告

课程名称：数字图像处理

实验项目名称：**Exp 5: Integrated Experiment**

学院：电子与信息工程学院

专业：电子信息工程

指导教师：李斌

报告人：贾苏健　学号：**2022280485**　班级：文华班

实验时间：**2024** 年 **5** 月 **28** 日

6 月 **4** 日、**11** 日 、**18** 日、**25** 日

实验报告提交时间：**2024** 年 **6** 月 **27** 日

教务部制

实验目的（Aim of Experiment）：

(1) Be familiar with the operations of morphological image processing.

(2) Be familiar with the operations of image segmentation.

(3) Try to design GUI in Python.

实验内容与要求（Experiment Steps and Requirements）：

(1) **Erosion, Dilation, Opening and Closing.**

(a) Load the RGB image clock2.jpg. Convert it to grayscale and then convert it into a binary image using a threshold of 0.5.

(b) Use an SE as [1 1 1; 1 1 1; 1 1 1] to obtain the binary morphological erosion, dilation, opening and closing of the binary image. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(c) Use an SE as [1 0 0; 0 1 0; 0 0 1] to obtain the binary morphological erosion, dilation, opening and closing of the binary image. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(d) Load the RGB image zebras.jpg. Convert it to grayscale.

(e) Use a 3x3 SE to obtain the grayscale morphological erosion, dilation, opening and closing of grayscale image. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(f) Use a 7x7 SE to obtain the grayscale morphological erosion, dilation, opening and closing of grayscale image. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure. (Hint: skimage.morphology.binary_erosion, skimage.morphology.binary_dilation, skimage.morphology.binary_opening, skimage.morphology.binary_closing, skimage.morphology.erosion, skimage.morphology.dilation, skimage.morphology.opening and skimage.morphology.closing)

(2) **Boundary Extraction and Hole Filling.**

(a) Load the image boundary.tif. Convert it into a binary image.

(b) Use a 5x5 SE of 1s to obtain the binary morphological erosion of the binary image.

(c) Obtian the boundary of the binary image via performing the difference between the binary image and its erosion image.

(d) Load the image holefiling.tif. Convert it into a binary image.

(e) Fill the holes in binary objects. (Hine:scipy.ndimage.binary_fill_holes)

(f) Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(3) **Thining and Convex Hull.**

(a) Load the image CT.tif. Convert it into a binary image.

(b) Perform morphological thinning of a binary image. (Hint:skimage.morpholog y.thin, set the parameter of max_iter to 5, 50, and none, respectively.)

(c) Compute the convex hull image of a binary image. (Hint:skimage.morphology.convex_hull_image)

(d) Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(4) **Top-hat (white top-hat) and Bottom-hat (black top-hat) Transformation.**

(a) Load the grayscale image rice.tif. Convert it into a binary image with a threshold of 0.5.

(b) Perform top-hat transformation to the grayscale image. (Hint:skimage.morphology.white_tophat, set the parameter of footprint to skimage.morphology.square(81))

(c) Obtain thresholded top-hat image and convert it into a binary image.

(d) Perform bottom-hat transformation to the grayscale image. (Hint:skimage.morphology.black_tophat, set the parameter of footprint to skimage.morphology.square(81))

(e) Obtain thresholded bottom-hat image and convert it into a binary image.

(f) Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(5) **Edge Detection.**

(a) Load the image building.tif.

(b) Find edges in an image using the Roberts' cross operators, Sobel operators, Prewitt operators, and the Canny algorithm. (Hint:skimage.filters.roberts, skimage.filters.sobel, skimage.filters.prewitt, skimage.feature.canny) (c) Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(6) **Edge-Based Segmentation.**

(a) Load the coins image in skimage. (Hint:skimage.data.coins())

(b) Use Canny algorithm to obtain the edge image. (Hint:skimage.feature.canny, set the parameter of sigma to 3)

(c) Fill the holes to obtain the segmented image. (Hint:scipy.ndimage.binary_fill_holes)

(d) Display the images in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(7) **Hough transform.**

(a) Load the image triangle_circle.png. Convert it to grayscale.

(b) Perform a straight-line Hough transform to the grayscale image. (Hint:skimage.transform.hough_line)

(c) Obtain the peaks in a straight-line Hough transform. (Hint:skimage.transform.hough_line_peaks)

(d) Highlight the detected lines by red color. (Hint:matplotlib.pyplot.axline)

(e) Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

(8) **Thresholding-Based Segmentation.**

(a) Load the image shade_text1.tif.

(b) Perform automatic image thresholding by Otsu's method to get the threshol d value. (Hint:skimage.filters.threshold_otsu)

(c) Transform the grayscale image into a binary image using the threshold valu e obtained by Otsu's method.

(d) Compute a moving average threshold. (Hint:cv2.blur)

(e) Obtain the result of local thresholding using moving averages. (f) Display t hem in the same figure with sub-figures. Add the corresponding title to the sub-fig ure.

(9) **Image Processing Software with GUI.**

(a) Use QT or other tools to design an image processing toolbox with GUI. T here is no limitation on implementations.

(b) The GUI contain at least one basic image processing function, can it can d isplay the input image and the processed image.

(c) Submit a video recording to demonstrate the function of your GUI design i n your report.

实验代码及数据结果（Experiment Codes and Results）：

(1) **Erosion, Dilation, Opening and Closing.**

(a) Load the RGB image clock2.jpg. Convert it to grayscale and then convert i t into a binary image using a threshold of 0.5.

```
# (a) Load the RGB image clock2.jpg. Convert it to grayscale and then convert it into a binary image using a threshold of 0.5.
clock2 = io.imread('images/clock2.jpg')
clock2_gray = color.rgb2gray(clock2)
threshold = 0.5
clock2_binary = clock2_gray > threshold
```

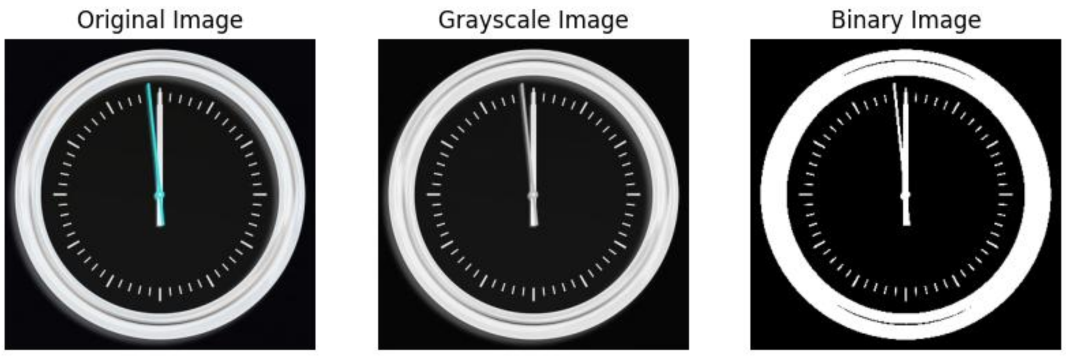


Figure 1 Comparison of original image, grayscale image, and binary image

(b) Use an SE as [1 1 1; 1 1 1; 1 1 1] to obtain the binary morphological er osion, dilation, opening and closing of the binary image. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

```
# (b) Use an SE as [1 1 1; 1 1 1; 1 1 1] to obtain the binary morphological erosion, dilation,
opening and closing of the binary image. Display them in the same figure with sub-figures. Add
the corresponding title to the sub-figure.
se1 = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
erosion1 = morphology.binary_erosion(clock2_binary, footprint=se1)
dilation1 = morphology.binary_dilation(clock2_binary, footprint=se1)
opening1 = morphology.binary_opening(clock2_binary, footprint=se1)
closing1 = morphology.binary_closing(clock2_binary, footprint=se1)

# Display binary operations with se1
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
axs[0, 0].imshow(erosion1, cmap='gray')
axs[0, 0].set_title('Erosion with SE [1 1 1; 1 1 1; 1 1 1]')
axs[0, 0].axis('off')

axs[0, 1].imshow(dilation1, cmap='gray')
axs[0, 1].set_title('Dilation with SE [1 1 1; 1 1 1; 1 1 1]')
axs[0, 1].axis('off')

axs[1, 0].imshow(opening1, cmap='gray')
axs[1, 0].set_title('Opening with SE [1 1 1; 1 1 1; 1 1 1]')
axs[1, 0].axis('off')

axs[1, 1].imshow(closing1, cmap='gray')
axs[1, 1].set_title('Closing with SE [1 1 1; 1 1 1; 1 1 1]')
axs[1, 1].axis('off')

plt.show()
```

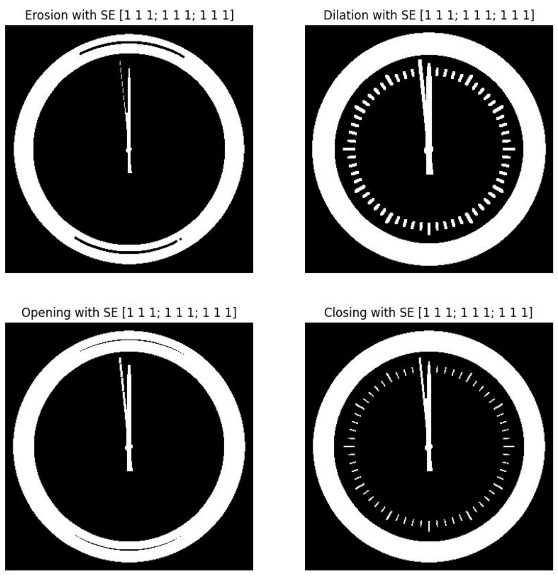

Erosion with SE [1 1 1; 1 1 1; 1 1 1]
Dilation with SE [1 1 1; 1 1 1; 1 1 1]
Opening with SE [1 1 1; 1 1 1; 1 1 1]
Closing with SE [1 1 1; 1 1 1; 1 1 1]

Figure 2 Comparison of Different Parameters

(c) Use an SE as [1 0 0; 0 1 0; 0 0 1] to obtain the binary morphological erosion, dilation, opening and closing of the binary image. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

```python
# (c) Use an SE as [1 0 0; 0 1 0; 0 0 1] to obtain the binary morphological erosion, dilation, opening and closing of the binary image. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.
se2 = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
erosion2 = morphology.binary_erosion(clock2_binary, footprint=se2)
dilation2 = morphology.binary_dilation(clock2_binary, footprint=se2)
opening2 = morphology.binary_opening(clock2_binary, footprint=se2)
closing2 = morphology.binary_closing(clock2_binary, footprint=se2)

# Display binary operations with se2
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
axs[0, 0].imshow(erosion2, cmap='gray')
axs[0, 0].set_title('Erosion with SE [1 0 0; 0 1 0; 0 0 1]')
axs[0, 0].axis('off')

axs[0, 1].imshow(dilation2, cmap='gray')
axs[0, 1].set_title('Dilation with SE [1 0 0; 0 1 0; 0 0 1]')
axs[0, 1].axis('off')

axs[1, 0].imshow(opening2, cmap='gray')
axs[1, 0].set_title('Opening with SE [1 0 0; 0 1 0; 0 0 1]')
axs[1, 0].axis('off')

axs[1, 1].imshow(closing2, cmap='gray')
axs[1, 1].set_title('Closing with SE [1 0 0; 0 1 0; 0 0 1]')
axs[1, 1].axis('off')

plt.show()
```

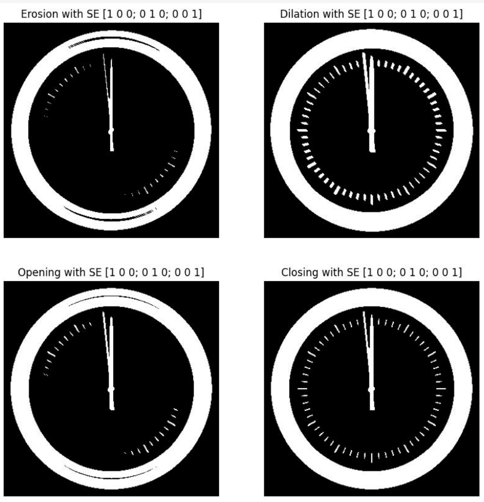

Erosion with SE [1 0 0; 0 1 0; 0 0 1]
Dilation with SE [1 0 0; 0 1 0; 0 0 1]
Opening with SE [1 0 0; 0 1 0; 0 0 1]
Closing with SE [1 0 0; 0 1 0; 0 0 1]

Figure 3 Comparison of Different Parameters

(d) Load the RGB image zebras.jpg. Convert it to grayscale.

```
# (d) Load the RGB image zebras.jpg. Convert it to grayscale.
zebras = io.imread('images/zebras.jpg')
zebras_gray = color.rgb2gray(zebras)
```

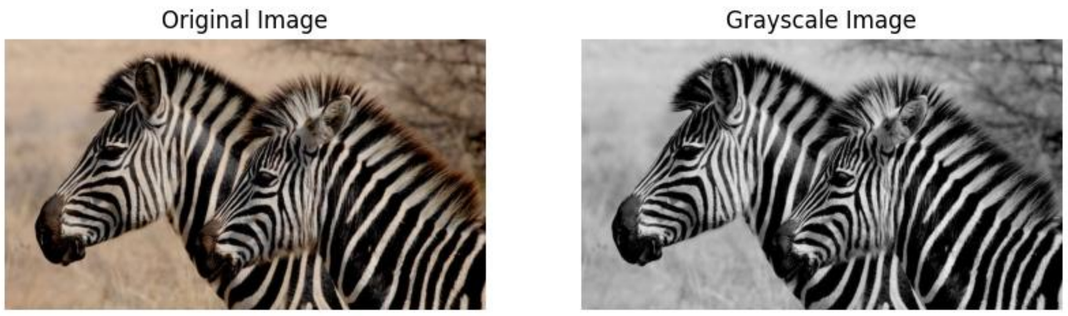


Figure 4 Original image and grayscale image

(e) Use a 3x3 SE to obtain the grayscale morphological erosion, dilation, opening and closing of grayscale image. Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

```
# (e) Use a 3x3 SE to obtain the grayscale morphological erosion, dilation, opening and closing
of grayscale image. Display them in the same figure with sub-figures. Add the corresponding title
to the sub-figure.
se3 = morphology.square(3)
erosion3 = morphology.erosion(zebras_gray, footprint=se3)
dilation3 = morphology.dilation(zebras_gray, footprint=se3)
opening3 = morphology.opening(zebras_gray, footprint=se3)
closing3 = morphology.closing(zebras_gray, footprint=se3)

# Display grayscale operations with se3
fig, axs = plt.subplots(2, 2, figsize=(12, 8))
axs[0, 0].imshow(erosion3, cmap='gray')
axs[0, 0].set_title('Erosion with 3x3 SE')
axs[0, 0].axis('off')

axs[0, 1].imshow(dilation3, cmap='gray')
axs[0, 1].set_title('Dilation with 3x3 SE')
axs[0, 1].axis('off')

axs[1, 0].imshow(opening3, cmap='gray')
axs[1, 0].set_title('Opening with 3x3 SE')
axs[1, 0].axis('off')

axs[1, 1].imshow(closing3, cmap='gray')
axs[1, 1].set_title('Closing with 3x3 SE')
axs[1, 1].axis('off')
```

```
plt.show()
```

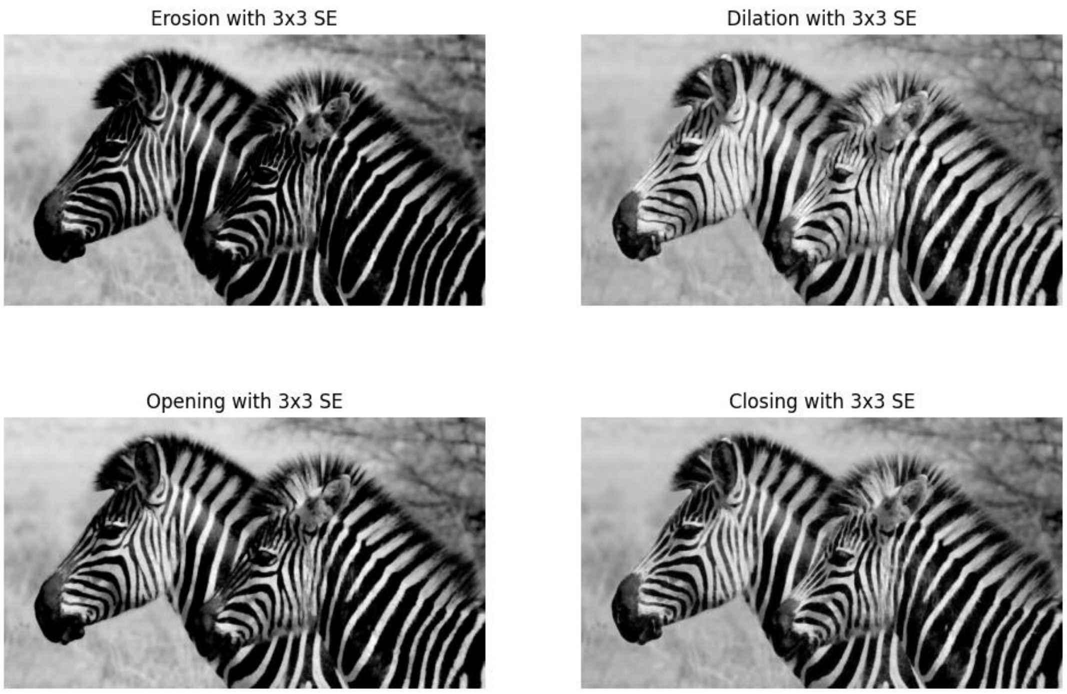


Figure 5 Comparison of different morphological treatments using 3x3 SE

(f) Use a 7x7 SE to obtain the grayscale morphological erosion, dilation, openi ng and closing of grayscale image. Display them in the same figure with sub-figure s. Add the corresponding title to the sub-figure. (Hint: skimage.morphology.binary_er osion, skimage.morphology.binary_dilation, skimage.morphology.binary_opening, skima ge.morphology.binary_closing, skimage.morphology.erosion, skimage.morphology.dilatio n, skimage.morphology.opening and skimage.morphology.closing)

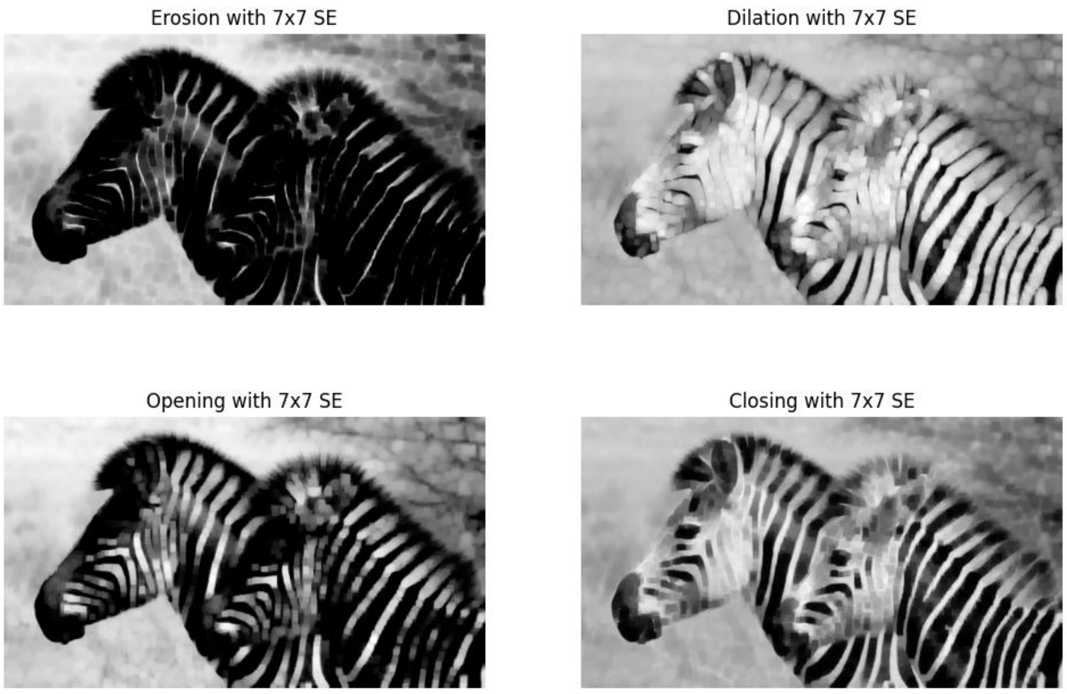


Figure 6 Comparison of different morphological treatments using 7x7 SE

(2) **Boundary Extraction and Hole Filling.**

(a) Load the image boundary.tif. Convert it into a binary image.

```
# (a) Load the image boundary.tif. Convert it into a binary image.
boundary_image = Image.open('images/boundary.tif')
# Convert to grayscale
boundary_image = boundary_image.convert('L')
# Convert to binary
binary_boundary_image = np.array(boundary_image) > 128
```

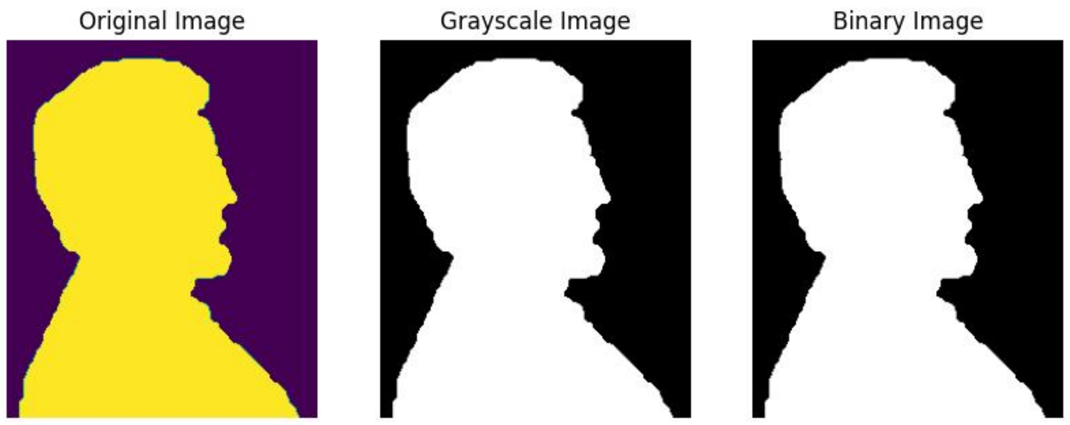


Figure 7 Original image, grayscale image, and binary image

(b) Use a 5x5 SE of 1s to obtain the binary morphological erosion of the binary image.

```
# (b) Use a 5x5 SE of 1s to obtain the binary morphological erosion of the binary image.
selem = np.ones((5, 5))
eroded_image = morphology.binary_erosion(binary_boundary_image, selem)
```

(c) Obtian the boundary of the binary image via performing the difference between the binary image and its erosion image.

```
# (c) Obtain the boundary of the binary image via performing the difference between the binary
image and its erosion image.
boundary = binary_boundary_image.astype(int) - eroded_image.astype(int)
```

(d) Load the image holefiling.tif. Convert it into a binary image.

```
# (d) Load the image holefiling.tif. Convert it into a binary image.
hole_filling_image = Image.open('images/holefiling.tif')
hole_filling_image = hole_filling_image.convert('L')  # Convert to grayscale
binary_hole_filling_image = np.array(hole_filling_image) > 128  # Convert to binary
```

(e) Fill the holes in binary objects. (Hine:scipy.ndimage.binary_fill_holes)

```
# (e) Fill the holes in binary objects.
filled_image = binary_fill_holes(binary_hole_filling_image)
```

(f) Display them in the same figure with sub-figures. Add the corresponding tit le to the sub-figure.

```
# (f) Display them in the same figure with sub-figures. Add the corresponding title to the
sub-figure.
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(binary_boundary_image, cmap='gray')
axes[0].set_title('Binary Boundary Image')
axes[0].axis('off')

axes[1].imshow(boundary, cmap='gray')
axes[1].set_title('Boundary of Binary Image')
axes[1].axis('off')

axes[2].imshow(filled_image, cmap='gray')
axes[2].set_title('Filled Image')
axes[2].axis('off')

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

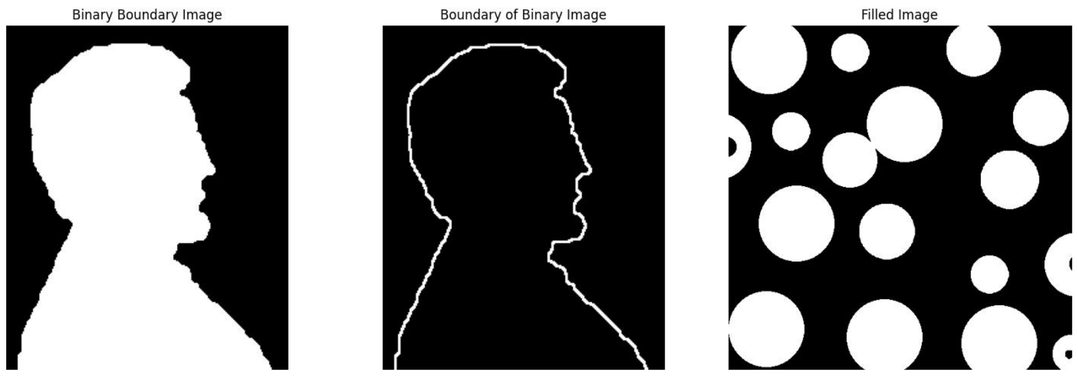


Figure 8 Results Display

(3) **Thining and Convex Hull.**

(a) Load the image CT.tif. Convert it into a binary image.

```
# (a) Load the image CT.tif. Convert it into a binary image.
ct_image = Image.open('images/CT.tif')
ct_image = ct_image.convert('L')  # Convert to grayscale
binary_ct_image = np.array(ct_image) > 128  # Convert to binary
```

(b) Perform morphological thinning of a binary image. (Hint:skimage.morphology.thin, set the parameter of max_iter to 5, 50, and none, respectively.)

```
# (b) Perform morphological thinning of a binary image.
thinned_5 = morphology.thin(binary_ct_image, max_num_iter=5)
thinned_50 = morphology.thin(binary_ct_image, max_num_iter=50)
thinned_none = morphology.thin(binary_ct_image)
```

(c) Compute the convex hull image of a binary image. (Hint:skimage.morphology.convex_hull_image)

```
# (c) Compute the convex hull image of a binary image.
convex_hull_image = morphology.convex_hull_image(binary_ct_image)
```

(d) Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

```
# (d) Display them in the same figure with sub-figures. Add the corresponding title to the
sub-figure.
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

axes[0].imshow(binary_ct_image, cmap='gray')
axes[0].set_title('Binary CT Image')
axes[0].axis('off')

axes[1].imshow(thinned_5, cmap='gray')
axes[1].set_title('Thinned Image (5 iterations)')
axes[1].axis('off')

axes[2].imshow(thinned_50, cmap='gray')
axes[2].set_title('Thinned Image (50 iterations)')
axes[2].axis('off')

axes[3].imshow(thinned_none, cmap='gray')
axes[3].set_title('Thinned Image (None)')
axes[3].axis('off')

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

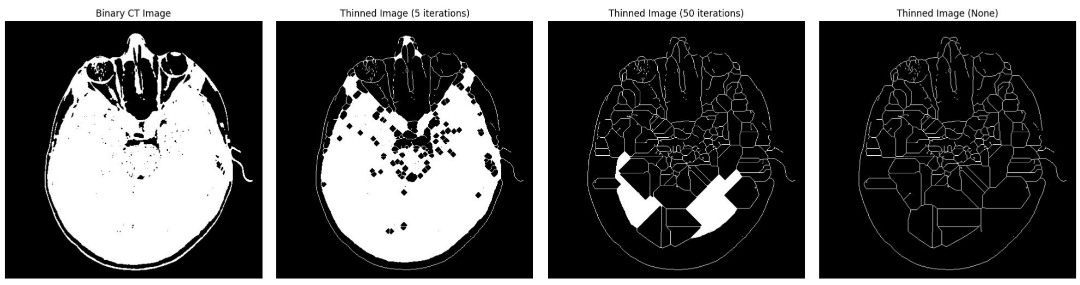


Figure 9 Presentation of Thinning and Convex Hull Results

```
# Display convex hull in a separate figure
fig, ax = plt.subplots(1, 1, figsize=(5, 5))

ax.imshow(convex_hull_image, cmap='gray')
ax.set_title('Convex Hull Image')
ax.axis('off')
```

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

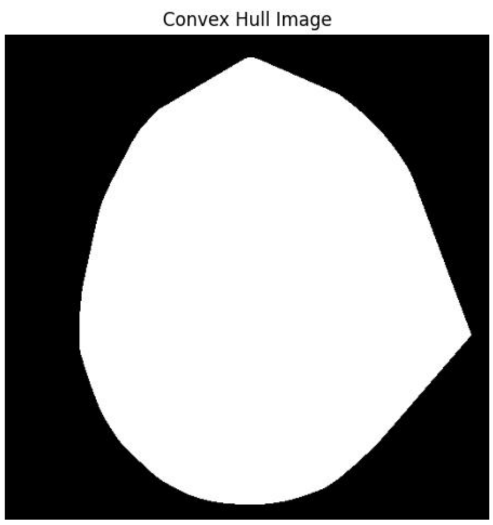


Figure 10 Morphological processing results

(4) **Top-hat (white top-hat) and Bottom-hat (black top-hat) Transformation.**

(a) Load the grayscale image rice.tif. Convert it into a binary image with a threshold of 0.5.

```python
# (a) Load the grayscale image rice.tif. Convert it into a binary image with a threshold of 0.5.
rice_image = Image.open('images/rice.tif')
rice_image = rice_image.convert('L')  # Ensure it is in grayscale
rice_image_np = np.array(rice_image) / 255.0  # Normalize to range [0, 1]
binary_rice_image = rice_image_np > 0.5  # Convert to binary
```

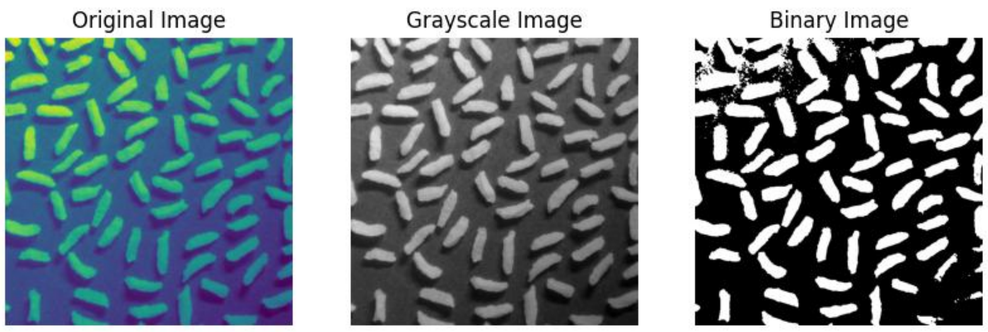


Figure 11 Original image, grayscale image, and binary image

(b) Perform top-hat transformation to the grayscale image. (Hint:skimage.morphology.white_tophat, set the parameter of footprint to skimage.morphology.square(81))

```python
# (b) Perform top-hat transformation to the grayscale image.
footprint = morphology.square(81)
top_hat_image = morphology.white_tophat(rice_image_np, footprint=footprint)
```

(c) Obtain thresholded top-hat image and convert it into a binary image.

```python
# (c) Obtain thresholded top-hat image and convert it into a binary image.
# Use Otsu's method for thresholding
thresholded_top_hat = top_hat_image > filters.threshold_otsu(top_hat_image)
```

(d) Perform bottom-hat transformation to the grayscale image. (Hint:skimage.morphology.black_tophat, set the parameter of footprint to skimage.morphology.square(81))

```python
# (d) Perform bottom-hat transformation to the grayscale image.
bottom_hat_image = morphology.black_tophat(rice_image_np, footprint=footprint)
```

(e) Obtain thresholded bottom-hat image and convert it into a binary image.

```python
# (e) Obtain thresholded bottom-hat image and convert it into a binary image.
# Use Otsu's method for thresholding
thresholded_bottom_hat = bottom_hat_image > filters.threshold_otsu(bottom_hat_image)
```

(f) Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

```python
# (f) Display them in the same figure with sub-figures. Add the corresponding title to the
sub-figure.
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

axes[0].imshow(binary_rice_image, cmap='gray')
axes[0].set_title('Binary Rice Image')
axes[0].axis('off')
axes[1].imshow(top_hat_image, cmap='gray')
axes[1].set_title('Top-hat Transformed Image')
axes[1].axis('off')
axes[2].imshow(thresholded_top_hat, cmap='gray')
axes[2].set_title('Thresholded Top-hat Image')
axes[2].axis('off')
axes[3].imshow(bottom_hat_image, cmap='gray')
axes[3].set_title('Bottom-hat Transformed Image')
axes[3].axis('off')

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

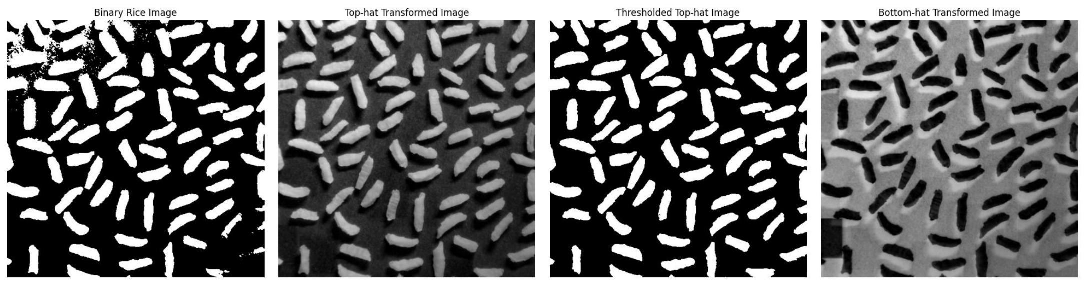

Binary Rice Image
Top-hat Transformed Image
Thresholded Top-hat Image
Bottom-hat Transformed Image

Figure 12 Display of results for Top-hat and Bottom-hat

```python
# Display thresholded bottom-hat image in a separate figure
```

```python
fig, ax = plt.subplots(1, 1, figsize=(5, 5))


ax.imshow(thresholded_bottom_hat, cmap='gray')
ax.set_title('Thresholded Bottom-hat Image')
ax.axis('off')


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

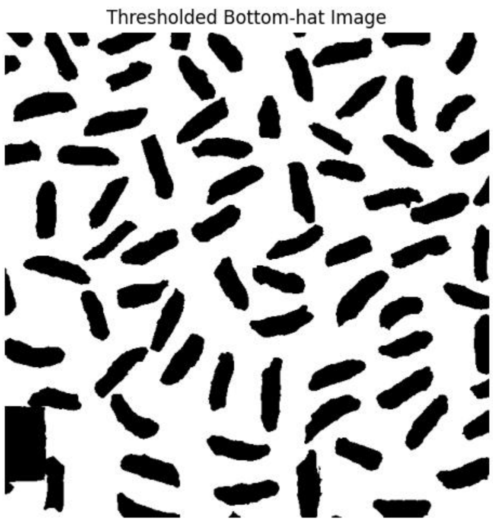


Figure 13 Thresholded Bottom-hat Image

(5) **Edge Detection.**

(a) Load the image building.tif.

```python
# (a) Load the image building.tif.
building_image = Image.open('images/building.tif')
building_image = building_image.convert('L')  # Ensure it is in grayscale
building_image_np = np.array(building_image)
```

(b) Find edges in an image using the Roberts' cross operators, Sobel operators, Prewitt operators, and the Canny algorithm. (Hint:skimage.filters.roberts, skimage.filters.sobel, skimage.filters.prewitt, skimage.feature.canny) (c) Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

```python
# (b) Find edges in an image using the Roberts' cross operators, Sobel operators, Prewitt operators,
and the Canny algorithm.
edges_roberts = filters.roberts(building_image_np)
edges_sobel = filters.sobel(building_image_np)
edges_prewitt = filters.prewitt(building_image_np)
# Normalize for Canny edge detection
edges_canny = feature.canny(building_image_np / 255.0)
# (c) Display them in the same figure with sub-figures. Add the corresponding title to the
sub-figure.
fig, axes = plt.subplots(1, 4, figsize=(20, 5))
```

```python
axes[0].imshow(edges_roberts, cmap='gray')
axes[0].set_title('Roberts Edge Detection')
axes[0].axis('off')

axes[1].imshow(edges_sobel, cmap='gray')
axes[1].set_title('Sobel Edge Detection')
axes[1].axis('off')

axes[2].imshow(edges_prewitt, cmap='gray')
axes[2].set_title('Prewitt Edge Detection')
axes[2].axis('off')

axes[3].imshow(edges_canny, cmap='gray')
axes[3].set_title('Canny Edge Detection')
axes[3].axis('off')

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

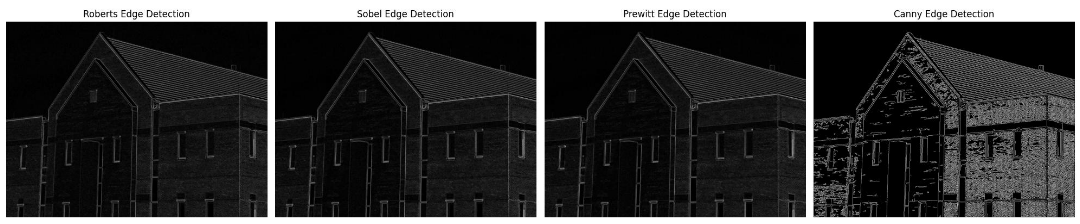


Figure 14 Boundary extraction display

(6) **Edge-Based Segmentation.**

(a) Load the coins image in skimage. (Hint:skimage.data.coins())

```python
# (a) Load the coins image in skimage.
coins_image = data.coins()
```

(b) Use Canny algorithm to obtain the edge image. (Hint:skimage.feature.canny, set the parameter of sigma to 3)

```python
# (b) Use Canny algorithm to obtain the edge image.
edges_canny = feature.canny(coins_image, sigma=3)
```

(c) Fill the holes to obtain the segmented image. (Hint:scipy.ndimage.binary_fill_holes)

```python
# (c) Fill the holes to obtain the segmented image.
segmented_image = binary_fill_holes(edges_canny)
```

(d) Display the images in the same figure with sub-figures. Add the corresponding title to the sub-figure.

```python
# (d) Display the images in the same figure with sub-figures. Add the corresponding title to the sub-figure.
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(coins_image, cmap='gray')
axes[0].set_title('Original Coins Image')
axes[0].axis('off')

axes[1].imshow(edges_canny, cmap='gray')
axes[1].set_title('Canny Edge Detection')
axes[1].axis('off')

axes[2].imshow(segmented_image, cmap='gray')
axes[2].set_title('Segmented Image')
axes[2].axis('off')

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

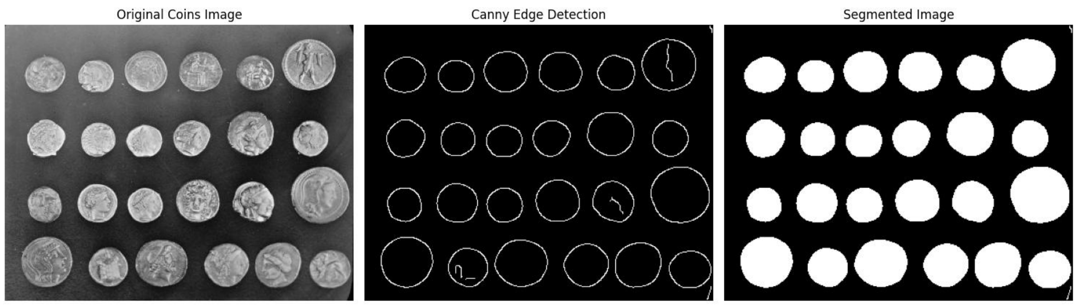


Figure 15 Boundary segmentation results

(7) **Hough transform.**

(a) Load the image triangle_circle.png. Convert it to grayscale.

```python
# (a) Load the image triangle_circle.png and convert it to grayscale
image_path = 'images/triangle_circle.png'
image = io.imread(image_path)

# Check if image has an alpha channel (RGBA format)
if image.shape[2] == 4:
    image = image[..., :3]  # Remove the alpha channel
gray_image = color.rgb2gray(image)
```

(b) Perform a straight-line Hough transform to the grayscale image. (Hint:skimage.transform.hough_line)

```python
# (b) Perform a straight line Hough transform to the grayscale image
h, theta, d = transform.hough_line(gray_image)
```

(c) Obtain the peaks in a straight-line Hough transform. (Hint:skimage.transform.hough_line_peaks)

```
# (c) Obtain the peaks in a straight line Hough transform
peaks = transform.hough_line_peaks(h, theta, d)
```

(d) Highlight the detected lines by red color. (Hint:matplotlib.pyplot.axline)

```
# (d) Highlight the detected lines by red color
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(image)

for _, angle, dist in zip(*peaks):
    y0 = (dist - 0 * np.cos(angle)) / np.sin(angle)  # y = (r - x*cos(theta)) / sin(theta)
    y1 = (dist - gray_image.shape[1] * np.cos(angle)) / np.sin(angle)  # y = (r - x*cos(theta))
/ sin(theta)
    ax.axline((0, y0), (gray_image.shape[1], y1), color='r')

ax.set_title('Hough Transform for Lines')
ax.axis('off')

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

(e) Display them in the same figure with sub-figures. Add the corresponding title to the sub-figure.

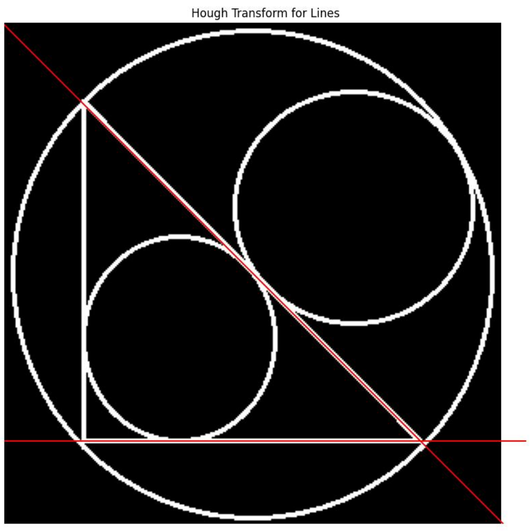


Figure 16 Hough transform Results

(8) **Thresholding-Based Segmentation.**

(a) Load the image shade_text1.tif.

```
# (a) Load the image shade_text1.tif
image_path = 'images/shade_text1.tif'
image = io.imread(image_path)
```

```python
# Check if the image is grayscale (2 channels) or RGB (3 channels)
if image.ndim == 2:
    gray_image = image  # It's already grayscale
else:
    gray_image = color.rgb2gray(image)  # Convert RGB to grayscale
```

(b) Perform automatic image thresholding by Otsu's method to get the threshol d value. (Hint:skimage.filters.threshold_otsu)

```python
# (b) Perform automatic image thresholding by Otsu's method
otsu_threshold = filters.threshold_otsu(gray_image)
```

(c) Transform the grayscale image into a binary image using the threshold valu e obtained by Otsu's method.

```python
# (c) Transform the grayscale image into a binary image using Otsu's method
binary_otsu = gray_image > otsu_threshold
```

(d) Compute a moving average threshold. (Hint:cv2.blur)

```python
# (d) Compute a moving average threshold
moving_average_threshold = cv2.blur(gray_image, (50, 50))
```

(e) Obtain the result of local thresholding using moving averages. (f) Display t hem in the same figure with sub-figures. Add the corresponding title to the sub-fig ure.

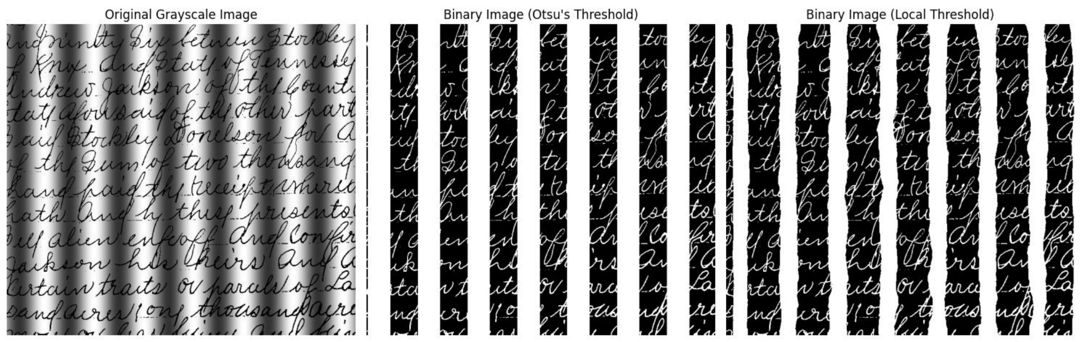
```python
# (e) Obtain the result of local thresholding using moving averages
binary_local = gray_image > moving_average_threshold
# (f) Display them in the same figure with sub-figures
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Original Image
axes[0].imshow(gray_image, cmap='gray')
axes[0].set_title('Original Grayscale Image')
axes[0].axis('off')

# Otsu's Thresholding
axes[1].imshow(binary_otsu, cmap='binary')
axes[1].set_title('Binary Image (Otsu\'s Threshold)')
axes[1].axis('off')

# Local Thresholding with Moving Average
axes[2].imshow(binary_local, cmap='binary')
axes[2].set_title('Binary Image (Local Threshold)')
axes[2].axis('off')
```

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



Original Grayscale Image
Binary Image (Otsu's Threshold)
Binary Image (Local Threshold)

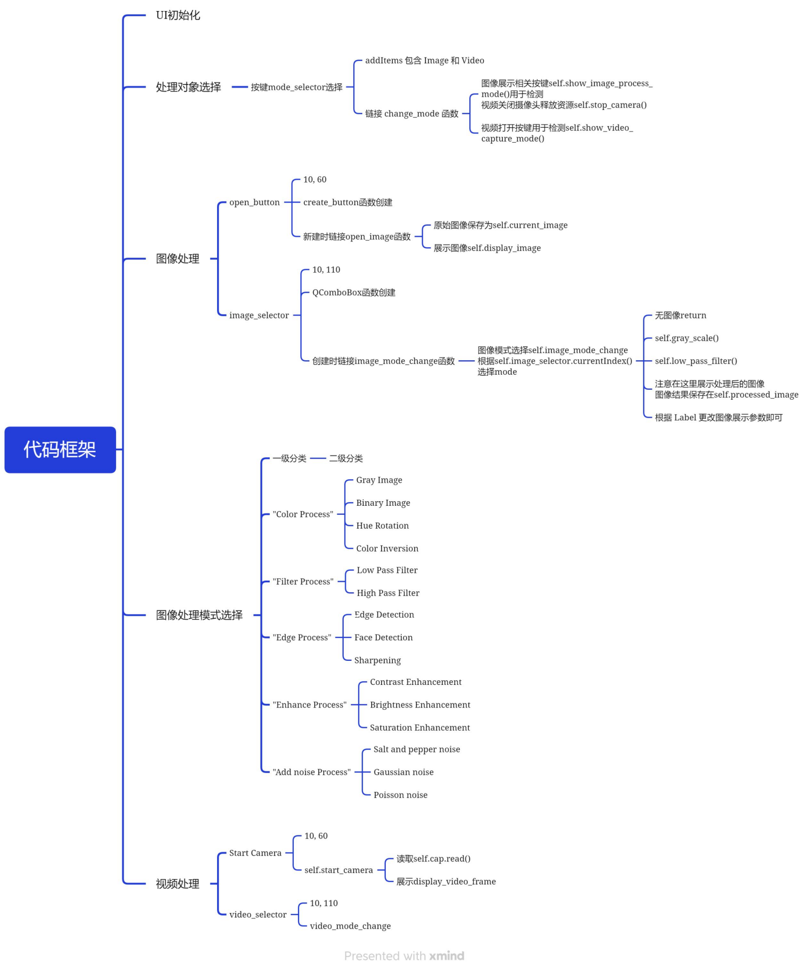
Figure 17 Threshold segmentation results

(9) **Image Processing Software with GUI.**

(a) Use QT or other tools to design an image processing toolbox with GUI. There is no limitation on implementations.

(b) The GUI contain at least one basic image processing function, can it can display the input image and the processed image.

(c) Submit a video recording to demonstrate the function of your GUI design in your report.



```python
import sys
import cv2
import random
import numpy as np
from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtGui import QPixmap, QImage
from PIL import Image, ImageOps, ImageFilter, ImageEnhance
from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel, QWidget, QPushButton,
QFileDialog, QComboBox, QSlider


class ImageProcessorGUI(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("贾苏健的 Image Processing Toolbox")
        self.setGeometry(100, 100, 1500, 900)

        self.main_widget = QWidget(self)
        self.setCentralWidget(self.main_widget)

        # 创建模式选择框
        self.mode_selector = QComboBox(self.main_widget)
        self.mode_selector.setGeometry(10, 10, 120, 40)
        self.mode_selector.addItems(["Image Process", "Video Capture"])
        self.mode_selector.currentIndexChanged.connect(self.change_mode)

        # 设置图像显示标签
        self.image_label = QLabel(self.main_widget)
        self.image_label.setGeometry(150, 10, 600, 800)  # x, y, width, height
        self.output_label = QLabel(self.main_widget)
        self.output_label.setGeometry(760, 10, 600, 800)  # x, y, width, height
        self.video_label = QLabel(self.main_widget)
        self.video_label.setGeometry(150, 10, 600, 800)  # x, y, width, height
        self.video_output = QLabel(self.main_widget)
        self.video_output.setGeometry(760, 10, 600, 800)  # x, y, width, height

        self.open_button = self.create_button("Open Image", 10, 60, self.open_image)
        # 创建图像处理模式选择按键
        self.image_selector = QComboBox(self.main_widget)
        self.image_selector.setGeometry(10, 110, 120, 40)
        self.image_selector.addItems(["Color Process", "Filter Process", "Edge
Process",  "Enhance Process", "Add noise Process"])
        self.image_selector.currentIndexChanged.connect(self.image_mode_change)
        # 创建颜色处理模式选择按键
        self.color_process_selector = QComboBox(self.main_widget)
        self.color_process_selector.setGeometry(10, 160, 120, 40)
        self.color_process_selector.addItems(["Gray Image", "Binary Image", "Hue Rotation",
"Color Inversion"])
        self.color_process_selector.currentIndexChanged.connect(self.image_mode_change)
        # 创建滤波处理模式选择按键
        self.filter_process_selector = QComboBox(self.main_widget)
        self.filter_process_selector.setGeometry(10, 160, 120, 40)
        self.filter_process_selector.addItems(["Low Pass Filter", "High Pass Filter"])
        self.filter_process_selector.currentIndexChanged.connect(self.image_mode_change)
        # 创建边缘处理模式选择按键
        self.edge_process_selector = QComboBox(self.main_widget)
        self.edge_process_selector.setGeometry(10, 160, 120, 40)
        self.edge_process_selector.addItems(["Edge Detection", "Sharpening"])
        self.edge_process_selector.currentIndexChanged.connect(self.image_mode_change)
        # 创建增强处理模式选择按键
        self.enhance_process_selector = QComboBox(self.main_widget)
        self.enhance_process_selector.setGeometry(10, 160, 120, 40)
        self.enhance_process_selector.addItems(["Contrast Enhancement", "Brightness
Enhancement", "Saturation Enhancement"])
        self.enhance_process_selector.currentIndexChanged.connect(self.image_mode_change)
        # 创建添加噪声处理模式选择按键
        self.noise_process_selector = QComboBox(self.main_widget)
        self.noise_process_selector.setGeometry(10, 160, 120, 40)
        self.noise_process_selector.addItems(["Salt and pepper noise", "Gaussian noise",
"Poisson noise"])
        self.noise_process_selector.currentIndexChanged.connect(self.image_mode_change)
```

```python
        # self.start_button = self.create_button("Start Camera", 10, 60, self.open_camera)
        # 创建视频处理模式选择按键
        self.video_selector = QComboBox(self.main_widget)
        self.video_selector.setGeometry(10, 60, 120, 40)
        self.video_selector.addItems(["Edge Detection", "Face Detection"])
        self.video_selector.currentIndexChanged.connect(self.video_mode_change)
```

```python
        # 添加滑条
        self.slider = QSlider(Qt.Horizontal, self.main_widget)
        self.slider.setGeometry(150, 750, 1000, 40)
        self.slider.setRange(1, 100)  # 将范围调整为 1 到 100，增加了滑块的精度
        self.slider.setSingleStep(1)  # 设置每次滑动的步长为 1，使其更连续
        self.slider.valueChanged.connect(self.image_mode_change)
        self.slider.hide()  # 默认隐藏滑条
        # 添加保存图像按钮
        self.save_button = self.create_button("Save Image", 10, 210, self.save_image)

        # 初始化为 Gray Scale 模式
        self.current_image = None
        self.processed_image = None
        self.timer = QTimer()
        self.cap = None
        self.change_mode(0)

    def create_button(self, text, x, y, callback):
        button = QPushButton(text, self.main_widget)
        button.setGeometry(x, y, 120, 40)  # x, y, width, height
        button.clicked.connect(callback)
        return button

    def change_mode(self, index):
        if index == 0:  # Image Process mode
            self.show_image_process_mode()
            self.stop_camera()
        elif index == 1:  # Video Capture mode
            self.show_video_capture_mode()
            self.start_camera()

    def show_image_process_mode(self):
        # Hide video related widgets
        self.video_label.hide()
        self.video_output.hide()
        # self.start_button.hide()
        self.video_selector.hide()

        # Show image processing related widgets
        self.image_label.show()
        self.output_label.show()
        self.open_button.show()
        self.image_selector.show()
        self.save_button.show()
        self.slider.hide()  # Show slider initially to allow selection based on filter type
        self.image_mode_change()  # Apply the initial filter selection to adjust the slider
visibility


    def show_video_capture_mode(self):
        # Hide image processing related widgets
        self.image_label.hide()
        self.output_label.hide()
        self.open_button.hide()
        self.image_selector.hide()
        self.color_process_selector.hide()
        self.filter_process_selector.hide()
        self.edge_process_selector.hide()
        self.enhance_process_selector.hide()
        self.noise_process_selector.hide()
        self.slider.hide()
        self.save_button.hide()

        # Show video related widgets
        self.video_label.show()
        self.video_output.show()
        # self.start_button.show()
        self.video_selector.show()

    def open_image(self):
        options = QFileDialog.Options()
        filename, _ = QFileDialog.getOpenFileName(self, "Open Image", "",
                                                  "Image Files (*.png *.jpg *.jpeg *.bmp *.gif)",
options=options)
        if filename:
            self.current_image = Image.open(filename)
            self.display_image(self.current_image, self.image_label)
            self.image_mode_change()

    # 用于显示图像
    def display_image(self, image, label):
        if image.mode != "RGB":
            image = image.convert("RGB")
        image = image.convert("RGBA")
        image_data = image.tobytes("raw", "RGBA")
        qimage = QImage(image_data, image.size[0], image.size[1], QImage.Format_RGBA8888)
        pixmap = QPixmap.fromImage(qimage)
        label.setPixmap(pixmap.scaled(label.width(), label.height(), Qt.KeepAspectRatio))

    # 用于更改图像处理模式时调用
    def image_mode_change(self):
        if not self.current_image:
            return

        process_mode = self.image_selector.currentIndex()
        if process_mode == 0:  # Color Process
            self.color_process_selector.show()
            self.filter_process_selector.hide()
            self.edge_process_selector.hide()
            self.enhance_process_selector.hide()
            self.noise_process_selector.hide()
        elif process_mode == 1:  # Filter Process
            self.color_process_selector.hide()
            self.filter_process_selector.show()
            self.edge_process_selector.hide()
            self.enhance_process_selector.hide()
            self.noise_process_selector.hide()
        elif process_mode == 2:  # Edge Process
            self.color_process_selector.hide()
            self.filter_process_selector.hide()
            self.edge_process_selector.show()
            self.enhance_process_selector.hide()
            self.noise_process_selector.hide()
        elif process_mode == 3:  # Enhance Process
            self.color_process_selector.hide()
            self.filter_process_selector.hide()
            self.edge_process_selector.hide()
            self.enhance_process_selector.show()
            self.noise_process_selector.hide()
        elif process_mode == 4:  # Add noise Process
            self.color_process_selector.hide()
            self.filter_process_selector.hide()
            self.edge_process_selector.hide()
            self.enhance_process_selector.hide()
            self.noise_process_selector.show()
```

```
        if process_mode == 0:  # Color Process
            color_process_mode = self.color_process_selector.currentIndex()
            if color_process_mode == 0:
                self.slider.hide()
                self.gray_scale()
            elif color_process_mode == 1:
                self.slider.show()
                self.binary_image()
            elif color_process_mode == 2:
                self.slider.show()
                self.hue_rotation()
            elif color_process_mode == 3:
                self.slider.hide()
                self.color_inversion()
        elif process_mode == 1:  # Filter Process
            self.slider.show()
            filter_process_mode = self.filter_process_selector.currentIndex()
            if filter_process_mode == 0:
                self.low_pass_filter()
            elif filter_process_mode == 1:
                self.high_pass_filter()
        elif process_mode == 2:  # Edge Process
            edge_process_mode = self.edge_process_selector.currentIndex()
            if edge_process_mode == 0:
                self.slider.hide()
                self.edge_detection()
            elif edge_process_mode == 1:
                self.slider.show()
                self.sharpening()
            # elif edge_process_mode == 2:
            #     self.face_detection()
        elif process_mode == 3:  # Enhance Process
            self.slider.show()
            enhance_process_mode = self.enhance_process_selector.currentIndex()
            if enhance_process_mode == 0:
                self.contrast_enhancement()
            elif enhance_process_mode == 1:
                self.brightness_enhancement()
            elif enhance_process_mode == 2:
                self.saturation_enhancement()
        elif process_mode == 4:  # Add noise Process
            self.slider.show()
            add_noise_process_mode = self.noise_process_selector.currentIndex()
            if add_noise_process_mode == 0:
                self.add_salt_and_pepper_noise()
            elif add_noise_process_mode == 1:
                self.add_gaussian_noise()
            elif add_noise_process_mode == 2:
                self.add_poisson_noise()
```

```
    # 用于更改视频处理模式时调用
    def video_mode_change(self):
        if not self.cap or not self.cap.isOpened():
            return
        # 不要在这里更新摄像头帧，而是确保定时器连接到update_camera_frame
        self.update_camera_frame()

    def gray_scale(self):
        if self.current_image:
            gray_image = ImageOps.grayscale(self.current_image)
            self.display_image(gray_image, self.output_label)
            self.processed_image = gray_image

    def binary_image(self):
        if self.current_image:
            threshold = self.slider.value() * 255 / 100  # Retrieve slider value and scale to
[0, 255]
            grayscale_image = self.current_image.convert("L")  # 转换为灰度图像
            binary_image = grayscale_image.point(lambda p: p > threshold and 255)  # 应用阈值
处理

            self.display_image(binary_image, self.output_label)
            self.processed_image = binary_image

    def hue_rotation(self):
        if self.current_image:
            hue_factor = self.slider.value() / 20.0  # Retrieve slider value
            enhanced_image = ImageEnhance.Color(self.current_image).enhance(hue_factor)

            self.display_image(enhanced_image, self.output_label)
            self.processed_image = enhanced_image

    def color_inversion(self):
        if self.current_image:
            inverted_image = ImageOps.invert(self.current_image)
            self.display_image(inverted_image, self.output_label)
            self.processed_image = inverted_image

    def low_pass_filter(self):
        if self.current_image:
            # 根据滑条的值设置滤波器强度
            strength = self.slider.value() / 10.0
            filtered_image =
self.current_image.filter(ImageFilter.GaussianBlur(radius=strength))
            self.display_image(filtered_image, self.output_label)
            self.processed_image = filtered_image

    def high_pass_filter(self):
        if self.current_image:
            strength = self.slider.value()  # 获取滑条当前值作为滤波器强度
            filtered_image = self.current_image.filter(ImageFilter.FIND_EDGES)
            filtered_image = filtered_image.filter(ImageFilter.UnsharpMask(radius=strength *
10.0, percent=150, threshold=3))
            self.display_image(filtered_image, self.output_label)
            self.processed_image = filtered_image

    def edge_detection(self):
        if self.current_image:
            filtered_image = self.current_image.filter(ImageFilter.FIND_EDGES)
            self.display_image(filtered_image, self.output_label)
            self.processed_image = filtered_image

    def sharpening(self):
        if self.current_image:
            strength = self.slider.value() / 10.0  # 获取滑条当前值并缩放到合适范围
            sharpened_image = self.current_image.filter(ImageFilter.SHARPEN)
            sharpened_image = ImageEnhance.Sharpness(sharpened_image).enhance(strength)
            self.display_image(sharpened_image, self.output_label)
            self.processed_image = sharpened_image

    def contrast_enhancement(self):
        if self.current_image:
            strength = self.slider.value() / 50.0 + 1.0  # 获取滑条当前值并缩放到合适范围
            enhanced_image = ImageEnhance.Contrast(self.current_image).enhance(strength)
            self.display_image(enhanced_image, self.output_label)
            self.processed_image = enhanced_image

    def brightness_enhancement(self):
        if self.current_image:
            strength = self.slider.value() / 50.0 + 1.0  # 获取滑条当前值并缩放到合适范围
            enhanced_image = ImageEnhance.Brightness(self.current_image).enhance(strength)
            self.display_image(enhanced_image, self.output_label)
            self.processed_image = enhanced_image

    def saturation_enhancement(self):
        if self.current_image:
            strength = self.slider.value() / 50.0 + 1.0  # 获取滑条当前值并缩放到合适范围
            enhanced_image = ImageEnhance.Color(self.current_image).enhance(strength)
            self.display_image(enhanced_image, self.output_label)
            self.processed_image = enhanced_image

    def add_salt_and_pepper_noise(self):
        if self.current_image:
            strength = self.slider.value() / 10.0  # Retrieve slider value
            if self.current_image.mode != "RGB":
                image = self.current_image.convert("RGB")
            else:
                image = self.current_image.copy()

            width, height = image.size
            pixels = image.load()
            num_pixels = int(width * height * (strength / 100.0))  # Adjust noise intensity based
on slider (range 1-10)

            for _ in range(num_pixels):
                x = random.randint(0, width - 1)
                y = random.randint(0, height - 1)
                if random.random() < 0.5:
                    pixels[x, y] = (0, 0, 0)  # Pepper noise, black
                else:
                    pixels[x, y] = (255, 255, 255)  # Salt noise, white

            self.display_image(image, self.output_label)
            self.processed_image = image

    def add_gaussian_noise(self):
        if self.current_image:
            strength = self.slider.value() / 10.0 * 25  # Retrieve slider value and scale to [0,
250]
            noisy_image = self.current_image.copy()
            width, height = noisy_image.size
            noise = np.random.normal(0, strength, (height, width, 3))
            noisy_image = np.array(noisy_image) + noise
            noisy_image = np.clip(noisy_image, 0, 255)
            noisy_image = Image.fromarray(noisy_image.astype('uint8'), 'RGB')

            self.display_image(noisy_image, self.output_label)
            self.processed_image = noisy_image

    def add_poisson_noise(self):
        if self.current_image:
            strength = 10.1 - self.slider.value() / 10.0  # Retrieve slider value
            noisy_image = self.current_image.copy()

            # Convert to numpy array and apply Poisson noise to each channel
            noisy_array = np.array(noisy_image)
            for channel in range(noisy_array.shape[2]):
                noisy_array[:, :, channel] = np.random.poisson(noisy_array[:, :, channel] * 
strength) / strength

            # Clip values to [0, 255] and convert back to uint8
            noisy_array = np.clip(noisy_array, 0, 255).astype(np.uint8)

            # Convert NumPy array back to PIL Image
            noisy_image = Image.fromarray(noisy_array)

            # Display the noisy image
            self.display_image(noisy_image, self.output_label)
            self.processed_image = noisy_image

    def save_image(self):
        if self.processed_image:
            options = QFileDialog.Options()
            save_filename, _ = QFileDialog.getSaveFileName(self, "Save Image", "",
                                                           "Image Files (*.png *.jpg *.jpeg *.bmp 
*.gif)",
                                                           options=options)
            if save_filename:
                self.processed_image.save(save_filename)

    # def open_camera(self):
    #     if not self.cap:
    #         self.cap = cv2.VideoCapture(0)
    #     if not self.timer.isActive():
    #         self.timer.timeout.connect(self.update_camera_frame)

    def start_camera(self):
        if not self.cap:
            self.cap = cv2.VideoCapture(0)
        if not self.timer.isActive():
            self.timer.timeout.connect(self.update_camera_frame)  # 连接定时器的槽函数
            self.timer.start(30)  # 30ms

    def stop_camera(self):
        if self.timer.isActive():
            self.timer.stop()
        if self.cap:
            self.cap.release()
            self.cap = None
        self.video_label.clear()
        self.video_output.clear()

    def update_camera_frame(self):
        if self.cap:
            ret, frame = self.cap.read()
            if ret:
                self.display_video_frame(frame, self.video_label)
                # 根据选择的视频处理模式进行处理
                video_process = self.video_selector.currentIndex()
                if video_process == 0:
                    self.video_edge_detection(frame)
                elif video_process == 1:
                    self.video_face_detection(frame)

    def video_edge_detection(self, frame):
        edges = cv2.Canny(frame, 100, 200)
        self.display_video_frame(edges, self.video_output, is_gray=True)

    def video_face_detection(self, frame):
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
        self.display_video_frame(frame, self.video_output)
    # 此函数只用于展示视频
    def display_video_frame(self, frame, label, is_gray=False):
        if is_gray:
            qimage = QImage(frame.data, frame.shape[1], frame.shape[0],
QImage.Format_Grayscale8)
        else:
            rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            qimage = QImage(rgb_image.data, rgb_image.shape[1], rgb_image.shape[0],
QImage.Format_RGB888)
        pixmap = QPixmap.fromImage(qimage)
        label.setPixmap(pixmap.scaled(label.width(), label.height(), Qt.KeepAspectRatio))
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = ImageProcessorGUI()
    window.show()
    sys.exit(app.exec_())
```

实验分析与结论（Analysis and Conclusion）：

1. Morphological Operations (Erosion, Dilation, Opening, Closing)

In this experiment, various morphological operations were applied using different structuring elements (SEs) on images, including erosion, dilation, opening, and closing. Here are the key findings:

Clock Image Processing: Using SEs [1 1 1; 1 1 1; 1 1 1] and [1 0 0; 0 1 0; 0 0 1] for morphological operations highlighted directional influences on the binary image. The former emphasized all-directional effects, while the latter focused on diagonal directions.

Zebra Image Processing: Applying different-sized SEs (3x3 and 7x7) on the grayscale image showed varying impacts on image edges and details. Smaller SEs preserved more details, whereas larger SEs blurred details but highlighted overall shapes.

These experiments underscored the importance of morphological operations in adjusting image morphology and detail through SE selection and operations.

2. Boundary Extraction and Hole Filling

Boundary Extraction: Employing a 5x5 SE of ones for binary erosion followed by subtraction effectively extracted object boundaries from the binary image.

Hole Filling: Filling holes in binary objects using `scipy.ndimage.binary_fill_holes` ensured complete and connected representation of objects, enhancing image integrity.

These experiments demonstrated effective techniques for handling object boundaries and internal gaps in images through morphological operations.

3. Thinning and Convex Hull

Thinning: Thinning the binary image using `skimage.morphology.thin` with different iteration counts highlighted the trade-off between shape preservation and pixel reduction, crucial for shape analysis.

Convex Hull: Computing the convex hull of binary images using `skimage.morphology.convex_hull_image` provided a minimal polygon representation useful for shape characterization.

These operations illustrated methods to enhance shape analysis and recognition through morphological techniques.

4. Top-hat and Bottom-hat Transformation

Top-hat Transformation: Performing top-hat transformation using `skimage.morphology.white_tophat` emphasized bright regions in the grayscale image, useful for detecting subtle features like rice grains.

Bottom-hat Transformation: Applying bottom-hat transformation with `skimage.morphology.black_tophat` highlighted dark areas, aiding in background noise reduction and object localization.

These transformations enhanced local contrast and facilitated detailed feature extraction in images.

5. Edge Detection

Edge Detection: Utilizing Roberts, Sobel, Prewitt operators, and the Canny algorithm for edge detection on the building image showcased different strengths in edge preservation, noise robustness, and overall performance.

These techniques provided insights into selecting appropriate edge detection methods based on image characteristics and application needs.

6. Edge-Based Segmentation

Edge-Based Segmentation:Employing the Canny algorithm for edge detection followed by hole filling produced segmented images of coins, effectively isolating objects for automated recognition and analysis.

This method illustrated a robust approach to image segmentation using edge information.

7. Hough Transform

Hough Transform: Applying the straight-line Hough transform to grayscale images of triangles and circles detected and visualized geometric shapes using red-colored lines, aiding in shape localization and description.

These experiments highlighted the utility of mathematical transforms for extracting geometric features from images.

8. Thresholding-Based Segmentation

Thresholding-Based Segmentation: Automatic thresholding using Otsu's method segmented shade_text1.tif into binary regions, followed by local thresholding for improved segmentation accuracy based on local intensity variations.

These techniques automated object separation based on grayscale intensity, enhancing segmentation reliability.

9. Image Processing Software with GUI

Image Processing Software with GUI: Designing an image processing toolbox with a GUI interface using QT provided intuitive interaction for basic and advanced image processing tasks. This interface facilitated efficient image manipulation and analysis for users.

These experiments underscored the importance of user-friendly interfaces in streamlining image processing workflows.

指导教师批阅意见：

成绩评定：

实验态度 10 分	实验步骤及代码 40 分	实验数据与结果 40 分	实验分析与结论 10 分

指导教师签字：李斌

2024　年　6　月 30 日

备注：