DIGITAL IMAGE PROCESSING
ELL715
ASSIGNMENT 5

10 November 2023

Group Members

1. Bhavik Shankla (2020MT60873)
2. Ritika Soni (2020MT10838)
3. Sai Kiran Gunnala (2020MT60889)
4. Sai Niketh Varanasi (2020MT60895)
5. Yash Pravin Shirke (2020MT60986)

1

1.1 Explanation

This MATLAB code first resizes the signature image to 128 by 128 pixels and then converts it to grayscale. To highlight the signature curve, the image is then binarized and goes through thinning operations. The centre of mass is then determined by the code after extracting the coordinates of the black pixels that represent the signature. After that, the signature is moved to the origin, and the covariance matrix for the modified coordinates is computed. The rotation angle is calculated from this matrix's minimum eigenvalue, and the signature is then rotated in accordance with that value. Relocating the signature to its original location is the last step. Using this MATLAB code we can make signatures robust which can be used in applications that recognise or verify signatures.

1.2 MATLAB Code

```
SigImage=imread('sig.jpg'); % Load the image file and store it as the variable SigImage.
figure,imshow(SigImage);

I2=imresize(SigImage,[128 ,128]);
figure,imshow(I2);

I3=rgb2gray(I2);
I3=im2double(I3);
I3=imbinarize(I3);                       %converting image to black and white
I3 = bwmorph(~I3, 'thin', inf);          %thining the image
I3=~I3;
figure,imshow(I3);

i=1;
k=1;

while i<=128
    j=1;
```

```
    while j<=128
        if I3(i, j)==0
            u(k)=i;
            v(k)=j;
            k=k+1;
            I3(i, j)=1;
        end
        j=j+1;
    end
    i=i+1;
end

C=[u;v];%the curve of the signature
N=k-1;%the number of pixels in the signature
oub=0;
for i=1:N
    oub= oub+ C(1,i);
end
oub=oub/N; %the original x co-ordinate center of mass of the image
ovb=0;
for i=1:N
    ovb= ovb+ C(2,i);
end
ovb=ovb/N;   %the original y co-ordinate center of mass of the image

%moving the signature to the origin
for i=1:N
    u(i)=u(i)-oub+1;
    v(i)=v(i)-ovb+1;
end
% the new curve of the signature
C=[u;v];

ub=sum(C(1,:))/N;
vb=sum(C(2,:))/N;
ubSq=sum((C(1,:)-ub).^2)/N;
vbSq=sum((C(2,:)-vb).^2)/N;

uvb=0;
for i=1:N
    uvb=uvb+ (u(i)*v(i));
end
uvb=uvb/N;
M=[ubSq uvb;uvb vbSq];
%calculating minimum eigen value of the matrix
minIgen=min(abs(eig(M)));

MI=[ubSq-minIgen uvb;uvb vbSq-minIgen];
theta=(atan((-MI(1))/MI(2))*180)/pi;

thetaRad=(theta*pi)/180;
%% rotating the signature and passing the new co-ordinates
for i=1:N
    v(i)=(C(2,i)*cos(thetaRad))-(C(1,i)*sin(thetaRad));
    u(i)=(C(2,i)*sin(thetaRad))+(C(1,i)*cos(thetaRad));
end
C=[u;v];

%moving the signature to its original position

for i=1:N
    u(i)=round(u(i)+oub-1);
    v(i)=round(v(i)+ovb-1);
```

```
end
```

2

2.1 Explanation

We used controlled dilation and erosion as basic morphological operations to improve images and pull out features. The approach begins with the preprocessing step of padding the original image based on the given structuring element. This ensures that the structuring element can be properly placed at each pixel location during subsequent operations.

For regulated dilation, we iterate over the original image, placing the reflected structuring element at each pixel. The reflected structuring element accounts for any symmetry present, eliminating the need for modification. At each iteration, we calculate the cardinality of the intersection between the structuring element and the corresponding region in the image. If this count exceeds a predefined strictness parameter (denoted as 's'), we set the pixel value in the resulting image to 1; otherwise, it is set to 0. This process effectively makes the edges of regions in the binary image bigger, and the strictness parameter lets you control how much it grows.

Conversely, regulated erosion involves placing the structuring element at each pixel and finding the intersection pixel count with the complemented original image. Here, the structuring element is not reflected. If the intersection count is below the specified strictness parameter ('s'), the pixel value in the resulting image is set to 1; otherwise, it is set to 0. Regulated erosion is meant to make the edges of areas in the binary image smaller, and the strictness parameter lets you change how the erosion works.

This methodical approach gives a flexible and controlled framework for controlled erosion and dilation, so the morphological operations can be made to fit the features of the input image.

2.2 Python Code

```python
pip install Pillow

import cv2
import numpy as np
from skimage import io
from skimage.morphology import binary_dilation, binary_erosion, square,
    rectangle, disk, diamond
from google.colab.patches import cv2_imshow
from PIL import Image

#reading the image
image = cv2.imread('book1.png')

# Open an image file
input_image_path = '/content/book1.png'
output_image_path = '/content/output_image.png'
image = Image.open("/content/book1.png")

new_size = (114*3, 89*3)

# Resize the image
resized_image = image.resize(new_size)

resized_array = np.array(resized_image)

cv2_imshow(resized_array)
cv2.waitKey(0)
cv2.destroyAllWindows()

gray_image = cv2.cvtColor(resized_array, cv2.COLOR_BGR2GRAY)
_, binary_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY)

```

```python
import numpy as np
# Iterate through each element in the array
for i in range(binary_image.shape[0]):
    for j in range(binary_image.shape[1]):
        # Check if the pixel value is 255
        if binary_image[i, j] == 255:
            # Change it to 1
            binary_image[i, j] = 1
# print(binary_image)


def pad_binary_image(image, top_padding, bottom_padding, left_padding, right_padding):
    original_height, original_width = image.shape

    #calculating the new dimensions with padding
    padded_height = original_height + top_padding + bottom_padding
    padded_width = original_width + left_padding + right_padding

    #creating a new blank image filled with zeros
    padded_image = np.zeros((padded_height, padded_width), dtype=np.uint8)

    #copying the original binary image to the center of the padded image
    padded_image[top_padding:top_padding + original_height, left_padding:left_padding + original_width] = image
    # print("Padded iamge is \n")
    # print(padded_image)
    return padded_image

def count_intersection_pixels(image, struc_ele, x_offset, y_offset):
    #ensuring that the structuring element fits within the image
    if y_offset + struc_ele.shape[1]-1 > image.shape[1] or x_offset + struc_ele.shape[0]-1 > image.shape[0]:
        raise ValueError("Structuring element does not fit within the image at the specified coordinates.")

    #creating a region of interest (ROI) in the image
    roi = image[x_offset : x_offset + struc_ele.shape[0], y_offset : y_offset + struc_ele.shape[1]]

    #counting the number of intersection pixels (pixels with a value of 1 in both image and the structuring element)
    intersection_count = np.sum(roi & struc_ele)
    #print("Intersection count is \n")
    #print(intersection_count)
    return intersection_count

#updated dilation function
def updated_dilation(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width), dtype=np.uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    # we run on the image and check the cardinality of the 1 pixels when structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else 0
    for x in range(image.shape[0]): # height
```

```python
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(padded_image,
                struc_ele, x, y)
            if intersection_count >= s:
                resulting_image[x, y] = 255

    return resulting_image

#updated erosion function
def updated_erosion(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width),
        dtype=np.uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    compliment_image = complemented_image =
        np.logical_not(padded_image).astype(int)
    # we run on the image and check the cardinality of the 1 pixels when
        structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else
        0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(compliment_image,
                struc_ele, x, y)
            if intersection_count <= s:
                resulting_image[x, y] = 255

    return resulting_image

# normal dilation function
def normal_dilation(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width),
        dtype=np.uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    # we run on the image and check the cardinality of the 1 pixels when
        structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else
        0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(padded_image,
                struc_ele, x, y)
            if intersection_count >= 1: # taking intersection greater than
                equal to 1
                resulting_image[x, y] = 255

    return resulting_image

# normal erosion function
def normal_erosion(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
```

```
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width),
        dtype=np.uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    # print(padded_image)
    compliment_image = complemented_image =
        np.logical_not(padded_image).astype(int)
    # print(compliment_image)
    # we run on the image and check the cardinality of the 1 pixels when
        structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else
        0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(compliment_image,
                struc_ele, x, y)
            if intersection_count == 0:
                resulting_image[x, y] = 255

    return resulting_image

import matplotlib.pyplot as plt

def display_images(result1, result2, result3, result4, custom_title):
    # Create a figure with 2 rows and 2 columns
    fig, axs = plt.subplots(2, 2, figsize=(10, 10))

    # Display images in each subplot with custom titles
    axs[0, 0].imshow(result1, cmap='gray')
    axs[0, 0].set_title(f'Updated Dilation - {custom_title}')

    axs[0, 1].imshow(result3, cmap='gray')
    axs[0, 1].set_title(f'Normal Dilation - {custom_title}')

    axs[1, 0].imshow(result2, cmap='gray')
    axs[1, 0].set_title(f'Updated Erosion - {custom_title}')

    axs[1, 1].imshow(result4, cmap='gray')
    axs[1, 1].set_title(f'Normal Erosion - {custom_title}')

    # Hide the axes labels
    for ax in axs.flat:
        ax.label_outer()

    # Show the plot
    plt.show()

#square structuring element
square_struc_ele = square(5)
#rectangle structuring element
rectangle_struc_ele = rectangle(5, 3)
#circle structuring element
circle_struc_ele = disk(3)
#diamond structuring element
diamond_struc_ele = diamond(3)
#plus structuring element
plus_struc_ele = np.array([[0, 0, 1, 0, 0],
                           [0, 0, 1, 0, 0],
                           [1, 1, 1, 1, 1],
                           [0, 0, 1, 0, 0],
```

```
                            [0, 0, 1, 0, 0]], dtype=bool)


result1 = updated_dilation(binary_image, 20, square_struc_ele)
result2 = updated_erosion(binary_image, 20, square_struc_ele)
result3 = normal_dilation(binary_image, 1, square_struc_ele)
result4 = normal_erosion(binary_image, 20, square_struc_ele)

display_images(result1, result2, result3, result4, "Square Structuring Element")

result1 = updated_dilation(binary_image, 15, rectangle_struc_ele)
result2 = updated_erosion(binary_image, 10, rectangle_struc_ele)
result3 = normal_dilation(binary_image, 1, rectangle_struc_ele)
result4 = normal_erosion(binary_image, 15, rectangle_struc_ele)

display_images(result1, result2, result3, result4, "Rectangle Structuring Element")

result1 = updated_dilation(binary_image, 20, circle_struc_ele)
result2 = updated_erosion(binary_image, 20, circle_struc_ele)
result3 = normal_dilation(binary_image, 1, circle_struc_ele)
result4 = normal_erosion(binary_image, 40, circle_struc_ele)


display_images(result1, result2, result3, result4, "Circular Structuring Element")

circle_struc_ele.size

result1 = updated_dilation(binary_image, 20, diamond_struc_ele)
result2 = updated_erosion(binary_image, 20, diamond_struc_ele)
result3 = normal_dilation(binary_image, 1, diamond_struc_ele)
result4 = normal_erosion(binary_image, 40, diamond_struc_ele)

diamond_struc_ele.size

display_images(result1, result2, result3, result4, "Diamond Structuring Element")


result1 = updated_dilation(binary_image, 7, plus_struc_ele)
result2 = updated_erosion(binary_image, 7, plus_struc_ele)
result3 = normal_dilation(binary_image, 1, plus_struc_ele)
result4 = normal_erosion(binary_image, 25, plus_struc_ele)

plus_struc_ele.size

display_images(result1, result2, result3, result4, "Plus Structuring Element")

def dilate_then_erode(input_image, s1, s2, structuring_element):
    # Perform dilation
    dilated_image = updated_dilation(input_image, s1, structuring_element)

    # Perform erosion on the dilated image
    final_image = updated_erosion(dilated_image, s2, structuring_element)

    return final_image

result1 = dilate_then_erode(binary_image, 20,20, square_struc_ele)
result2 = dilate_then_erode(binary_image, 15,10, rectangle_struc_ele)
result3 = dilate_then_erode(binary_image, 20,20, circle_struc_ele)
result4 = dilate_then_erode(binary_image, 20, 20, diamond_struc_ele)
```

```
result5 = dilate_then_erode(binary_image, 7,7, plus_struc_ele)


# Create a figure with 2 rows and 3 columns
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

# Display images in each subplot with custom titles
axs[0, 0].imshow(result1, cmap='gray')
axs[0, 0].set_title('Square')

axs[0, 1].imshow(result2, cmap='gray')
axs[0, 1].set_title('Rectangle')

axs[0, 2].imshow(result3, cmap='gray')
axs[0, 2].set_title('Circle')

axs[1, 0].imshow(result4, cmap='gray')
axs[1, 0].set_title('Diamond')

axs[1, 1].imshow(result5, cmap='gray')
axs[1, 1].set_title('Plus')

# Hide the axes labels
for ax in axs.flat:
    ax.label_outer()

# Hide the last subplot (if necessary, adjust this based on your layout)
axs[1, 2].axis('off')

# Show the plot
plt.show()
```

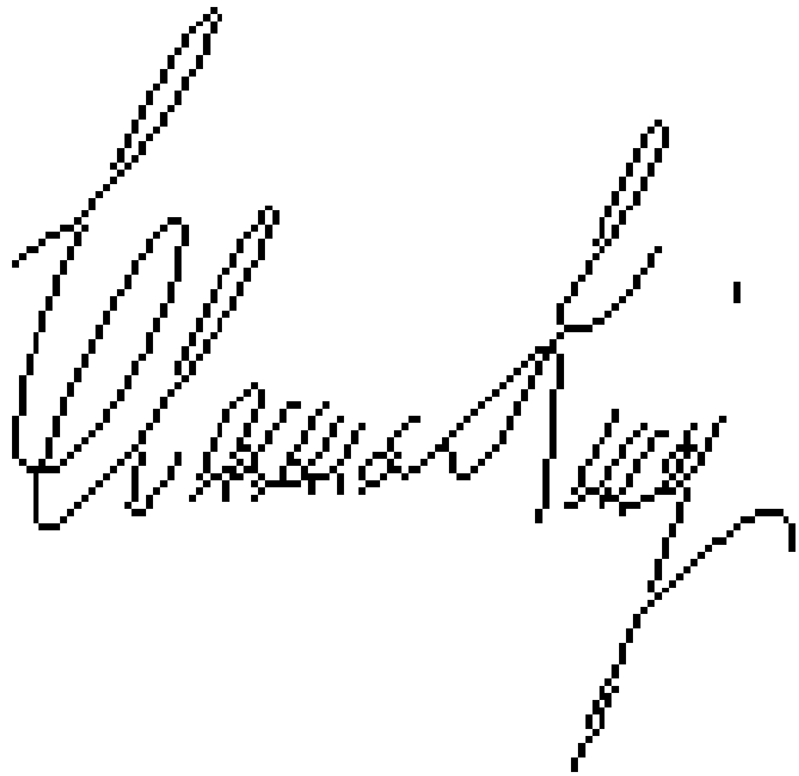
2.3 Results

We have improved the performance of the algorithm by replacing regular operations with regulated ones and fine-tuning a parameter called "strictness" based on specific criteria. This optimization, achieved by using regulated operations strategically, has upgraded the effectiveness of the morphological algorithm. By swapping out certain parts with regulated operations and adjusting them appropriately, we have obtained better results.

```
SigImage=imread('sig.jpg'); % Load the image file and store it as the variable SigImage.
figure,imshow(SigImage);
```

```
I2=imresize(SigImage,[128 ,128]);
figure,imshow(I2);
```

```
I3=rgb2gray(I2);
I3=im2double(I3);
I3=imbinarize(I3);                    %converting image to black and white
I3 = bwmorph(~I3, 'thin', inf);       %thining the image
I3=~I3;
figure,imshow(I3);
```



```
i=1;
k=1;

while i<=128
    j=1;
    while j<=128
        if I3(i, j)==0
            u(k)=i;
            v(k)=j;
            k=k+1;
            I3(i, j)=1;
        end
        j=j+1;
    end
    i=i+1;
end

C=[u;v];%the curve of the signature
N=k-1;%the number of pixels in the signature
```

```
oub=0;
for i=1:N
    oub= oub+ C(1,i);
end
oub=oub/N; %the original x co-ordinate center of mass of the image
ovb=0;
for i=1:N
    ovb= ovb+ C(2,i);
end
ovb=ovb/N;   %the original y co-ordinate center of mass of the image
```

```
%moving the signature to the origin
for i=1:N
    u(i)=u(i)-oub+1;
    v(i)=v(i)-ovb+1;
end
% the new curve of the signature
C=[u;v];

ub=sum(C(1,:))/N;
vb=sum(C(2,:))/N;
ubSq=sum((C(1,:)-ub).^2)/N;
vbSq=sum((C(2,:)-vb).^2)/N;
```

```
uvb=0;
for i=1:N
    uvb=uvb+ (u(i)*v(i));
end
uvb=uvb/N;
M=[ubSq uvb;uvb vbSq];
%calculating minimum eigen value of the matrix
minIgen=min(abs(eig(M)));

MI=[ubSq-minIgen uvb;uvb vbSq-minIgen];
theta=(atan((-MI(1))/MI(2))*180)/pi;

thetaRad=(theta*pi)/180;
%% rotating the signature and passing the new co-ordinates
for i=1:N
    v(i)=(C(2,i)*cos(thetaRad))-(C(1,i)*sin(thetaRad));
    u(i)=(C(2,i)*sin(thetaRad))+(C(1,i)*cos(thetaRad));
end
C=[u;v];

%moving the signature to its original position

for i=1:N
    u(i)=round(u(i)+oub-1);
    v(i)=round(v(i)+ovb-1);
end
```

vdxsvf4ph

November 10, 2023

[223]:
```
pip install Pillow
```

```
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages
(9.4.0)
```

[224]:
```
import cv2
import numpy as np
from skimage import io
from skimage.morphology import binary_dilation, binary_erosion, square,␣
↪rectangle, disk, diamond
from google.colab.patches import cv2_imshow
from PIL import Image
```

[225]:
```
#reading the image
image = cv2.imread('book1.png')
```

[226]:
```
# Open an image file
input_image_path = '/content/book1.png'
output_image_path = '/content/output_image.png'
image = Image.open("/content/book1.png")

new_size = (114*3, 89*3)

# Resize the image
resized_image = image.resize(new_size)

resized_array = np.array(resized_image)

cv2_imshow(resized_array)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[227]:
```
gray_image = cv2.cvtColor(resized_array, cv2.COLOR_BGR2GRAY)
_, binary_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY)
```

[228]:
```
import numpy as np
# Iterate through each element in the array
for i in range(binary_image.shape[0]):
    for j in range(binary_image.shape[1]):
        # Check if the pixel value is 255
        if binary_image[i, j] == 255:
            # Change it to 1
            binary_image[i, j] = 1
# print(binary_image)
```

[229]:
```
def pad_binary_image(image, top_padding, bottom_padding, left_padding,␣
 ↪right_padding):
    original_height, original_width = image.shape

    #calculating the new dimensions with padding
    padded_height = original_height + top_padding + bottom_padding
    padded_width = original_width + left_padding + right_padding

    #creating a new blank image filled with zeros
    padded_image = np.zeros((padded_height, padded_width), dtype=np.uint8)
```

```
    #copying the original binary image to the center of the padded image
    padded_image[top_padding:top_padding + original_height, left_padding:
 ↪left_padding + original_width] = image
    # print("Padded iamge is \n")
    # print(padded_image)
    return padded_image
```

[230]:
```
def count_intersection_pixels(image, struc_ele, x_offset, y_offset):
    #ensuring that the structuring element fits within the image
    if y_offset + struc_ele.shape[1]-1 > image.shape[1] or x_offset + struc_ele.
 ↪shape[0]-1 > image.shape[0]:
        raise ValueError("Structuring element does not fit within the image at␣
 ↪the specified coordinates.")

    #creating a region of interest (ROI) in the image
    roi = image[x_offset : x_offset + struc_ele.shape[0], y_offset : y_offset +␣
 ↪struc_ele.shape[1]]

    #counting the number of intersection pixels (pixels with a value of 1 in␣
 ↪both image and the structuring element)
    intersection_count = np.sum(roi & struc_ele)
    #print("Intersection count is \n")
    #print(intersection_count)
    return intersection_count
```

[231]:
```
#updated dilation function
def updated_dilation(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width), dtype=np.
 ↪uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    # we run on the image and check the cardinality of the 1 pixels when␣
 ↪structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else 0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(padded_image,␣
 ↪struc_ele, x, y)
            if intersection_count >= s:
                resulting_image[x, y] = 255
```

```python
    return resulting_image
```

[232]:

```python
#updated erosion function
def updated_erosion(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width), dtype=np.
 ↪uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    compliment_image = complemented_image = np.logical_not(padded_image).
 ↪astype(int)
    # we run on the image and check the cardinality of the 1 pixels when␣
 ↪structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else 0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(compliment_image,␣
 ↪struc_ele, x, y)
            if intersection_count <= s:
                resulting_image[x, y] = 255

    return resulting_image
```

[233]:

```python
# normal dilation function
def normal_dilation(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width), dtype=np.
 ↪uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    # we run on the image and check the cardinality of the 1 pixels when␣
 ↪structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else 0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(padded_image,␣
 ↪struc_ele, x, y)
```

```
            if intersection_count >= 1: # taking intersection greater than␣
↪equal to 1
                resulting_image[x, y] = 255

    return resulting_image
```

[234]:
```
# normal erosion function
def normal_erosion(image, s, struc_ele):
    struc_height, struc_width = struc_ele.shape
    struc_x = struc_width//2
    struc_y = struc_height//2
    original_height, original_width = image.shape

    resulting_image = np.zeros((original_height, original_width), dtype=np.
↪uint8)

    padded_image = pad_binary_image(image, struc_y, struc_y, struc_x, struc_x)
    # print(padded_image)
    compliment_image = complemented_image = np.logical_not(padded_image).
↪astype(int)
    # print(compliment_image)
    # we run on the image and check the cardinality of the 1 pixels when␣
↪structuring element is placed at each pixel in the image
    # if this cardinality is greater than or equal to s then we keep it 1 else 0
    for x in range(image.shape[0]): # height
        for y in range(image.shape[1]): # width
            intersection_count = count_intersection_pixels(compliment_image,␣
↪struc_ele, x, y)
            if intersection_count == 0:
                resulting_image[x, y] = 255

    return resulting_image
```

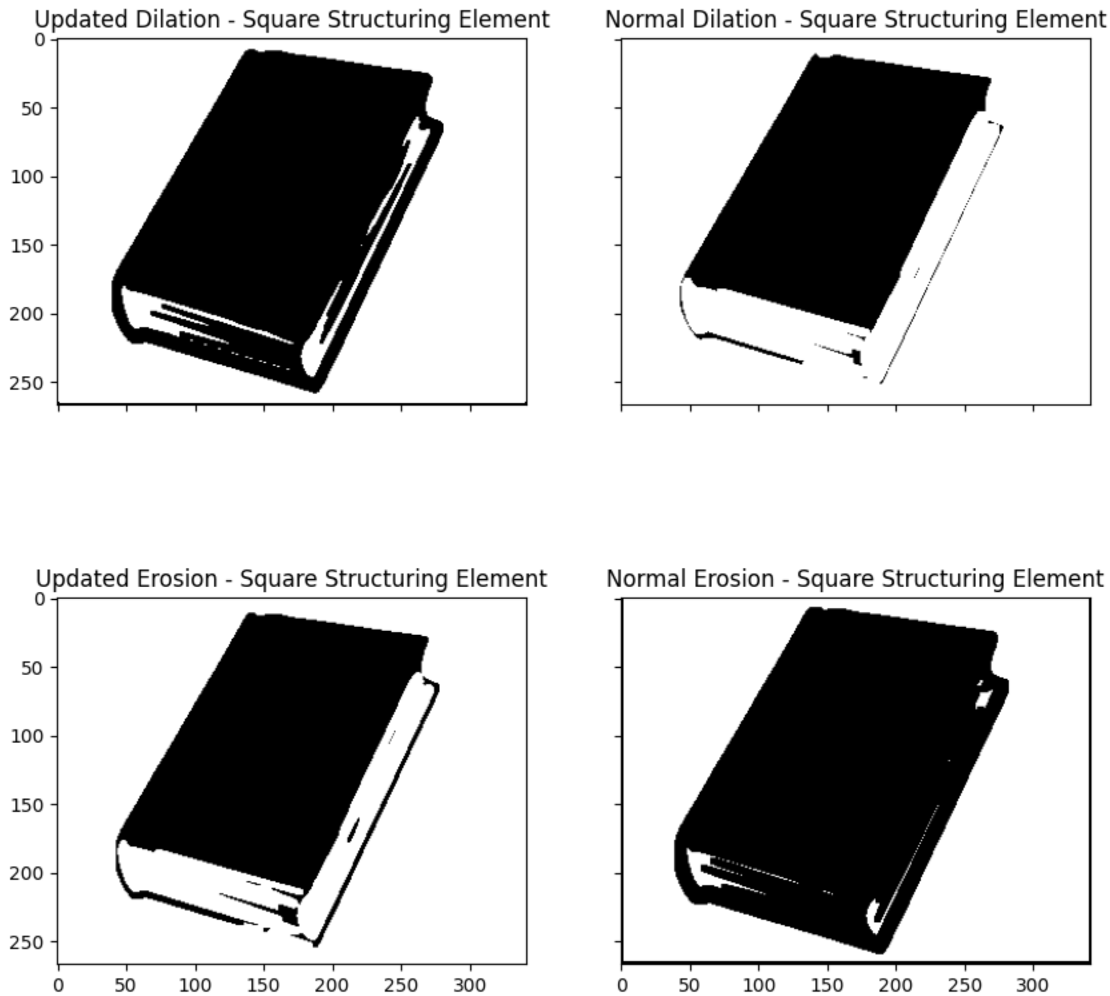
[235]:
```
import matplotlib.pyplot as plt

def display_images(result1, result2, result3, result4, custom_title):
    # Create a figure with 2 rows and 2 columns
    fig, axs = plt.subplots(2, 2, figsize=(10, 10))

    # Display images in each subplot with custom titles
    axs[0, 0].imshow(result1, cmap='gray')
    axs[0, 0].set_title(f'Updated Dilation - {custom_title}')

    axs[0, 1].imshow(result3, cmap='gray')
    axs[0, 1].set_title(f'Normal Dilation - {custom_title}')
```

```
    axs[1, 0].imshow(result2, cmap='gray')
    axs[1, 0].set_title(f'Updated Erosion - {custom_title}')

    axs[1, 1].imshow(result4, cmap='gray')
    axs[1, 1].set_title(f'Normal Erosion - {custom_title}')

    # Hide the axes labels
    for ax in axs.flat:
        ax.label_outer()

    # Show the plot
    plt.show()
```

[236]:
```
#square structuring element
square_struc_ele = square(5)
#rectangle structuring element
rectangle_struc_ele = rectangle(5, 3)
#circle structuring element
circle_struc_ele = disk(3)
#diamond structuring element
diamond_struc_ele = diamond(3)
#plus structuring element
plus_struc_ele = np.array([[0, 0, 1, 0, 0],
                           [0, 0, 1, 0, 0],
                           [1, 1, 1, 1, 1],
                           [0, 0, 1, 0, 0],
                           [0, 0, 1, 0, 0]], dtype=bool)
```

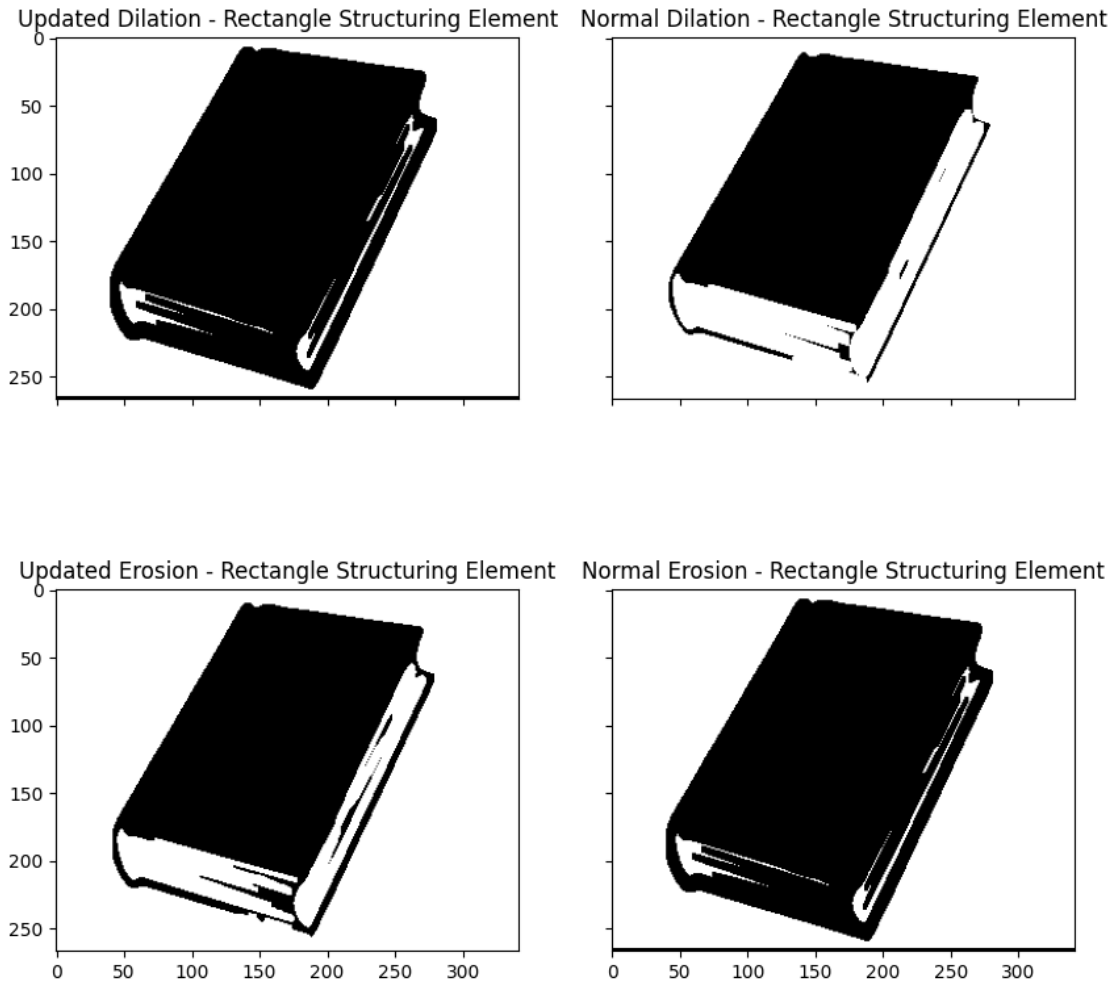
[237]:
```
result1 = updated_dilation(binary_image, 20, square_struc_ele)
result2 = updated_erosion(binary_image, 20, square_struc_ele)
result3 = normal_dilation(binary_image, 1, square_struc_ele)
result4 = normal_erosion(binary_image, 20, square_struc_ele)
```

[238]:
```
display_images(result1, result2, result3, result4, "Square Structuring Element")
```



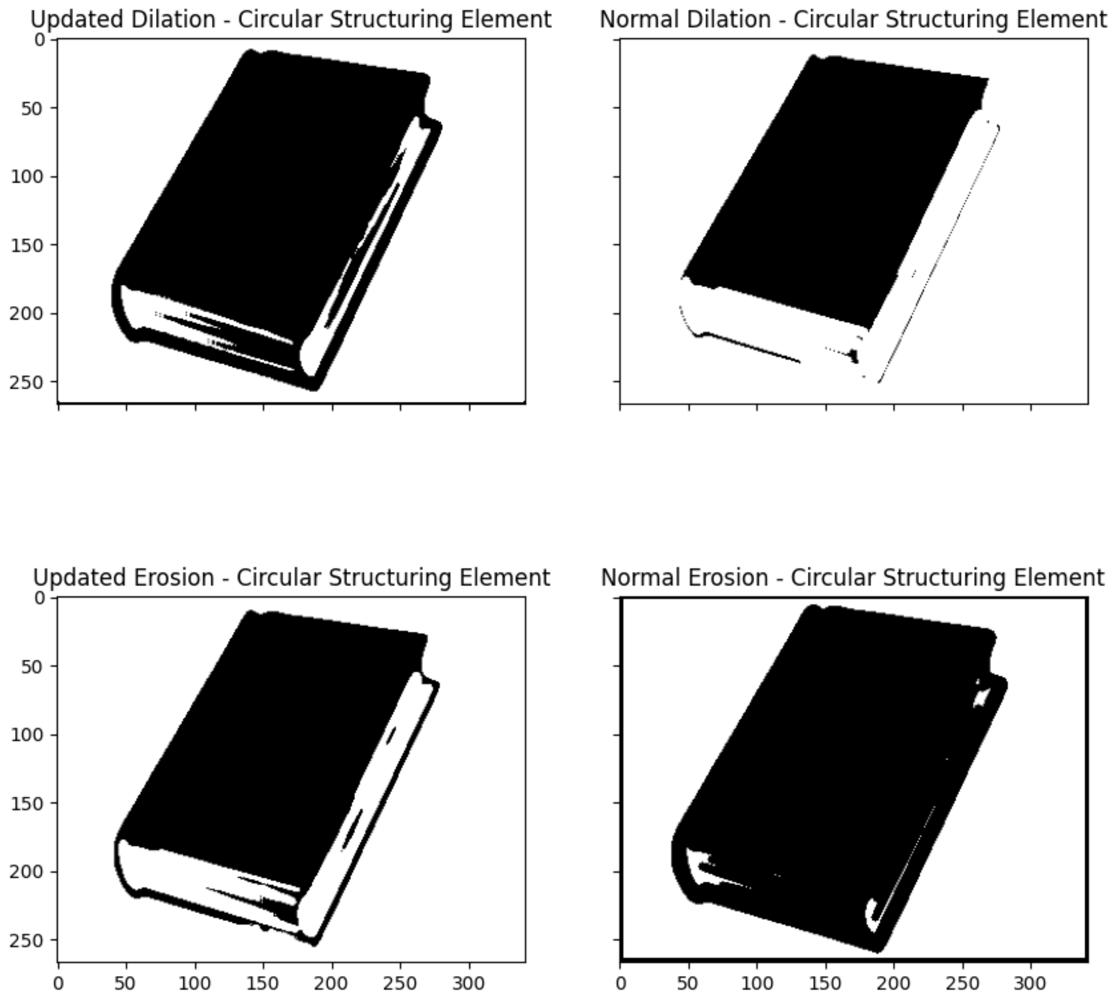
[239]:
```
result1 = updated_dilation(binary_image, 15, rectangle_struc_ele)
result2 = updated_erosion(binary_image, 10, rectangle_struc_ele)
result3 = normal_dilation(binary_image, 1, rectangle_struc_ele)
result4 = normal_erosion(binary_image, 15, rectangle_struc_ele)
```

[240]:
```
display_images(result1, result2, result3, result4, "Rectangle Structuring␣
↪Element")
```



[241]:
```
result1 = updated_dilation(binary_image, 20, circle_struc_ele)
result2 = updated_erosion(binary_image, 20, circle_struc_ele)
result3 = normal_dilation(binary_image, 1, circle_struc_ele)
result4 = normal_erosion(binary_image, 40, circle_struc_ele)
```

[242]:
```
display_images(result1, result2, result3, result4, "Circular Structuring␣
↪Element")
```



[243]:
```
circle_struc_ele.size
```

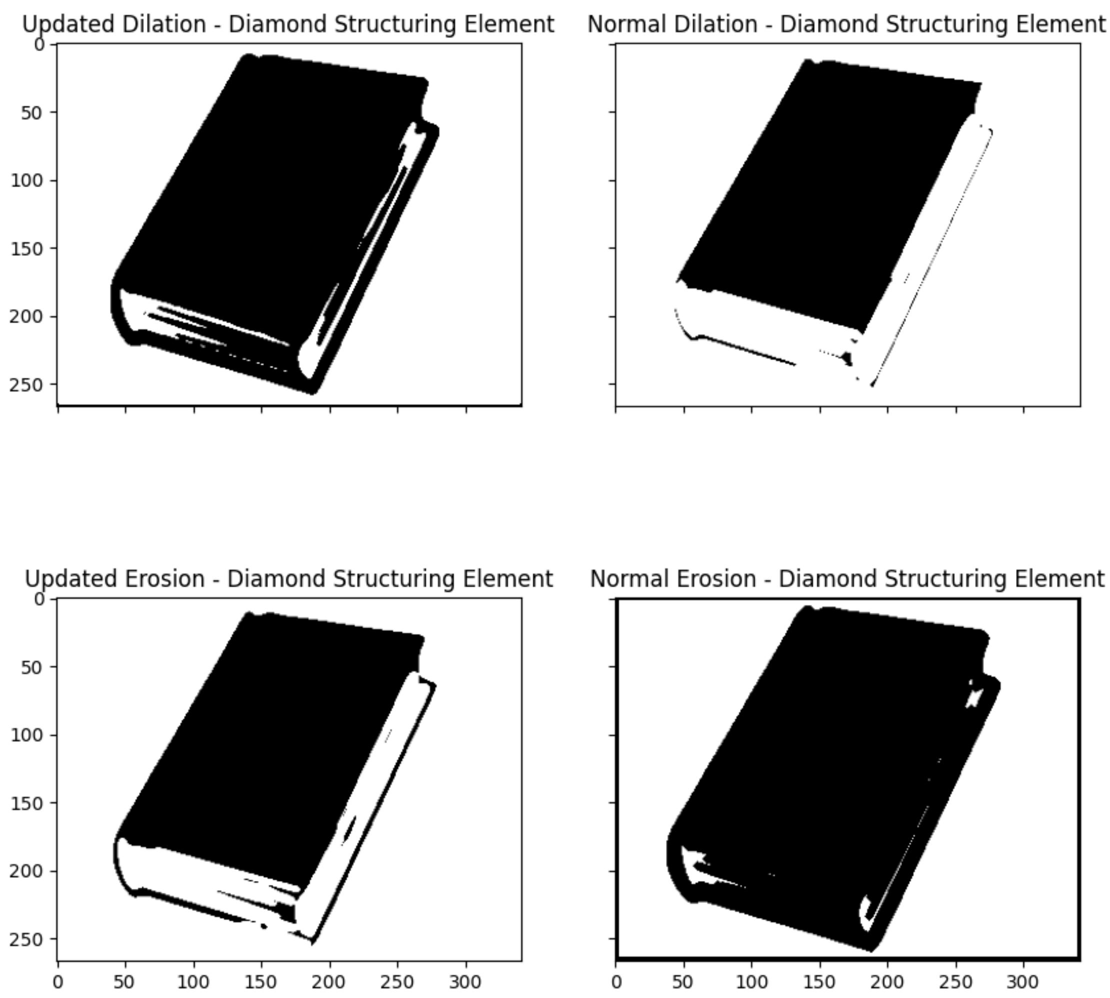
[243]: 49

[244]:
```
result1 = updated_dilation(binary_image, 20, diamond_struc_ele)
result2 = updated_erosion(binary_image, 20, diamond_struc_ele)
result3 = normal_dilation(binary_image, 1, diamond_struc_ele)
result4 = normal_erosion(binary_image, 40, diamond_struc_ele)
```

[245]:
```
diamond_struc_ele.size
```

[245]: 49

[246]:
```
display_images(result1, result2, result3, result4, "Diamond Structuring␣
↪Element")
```

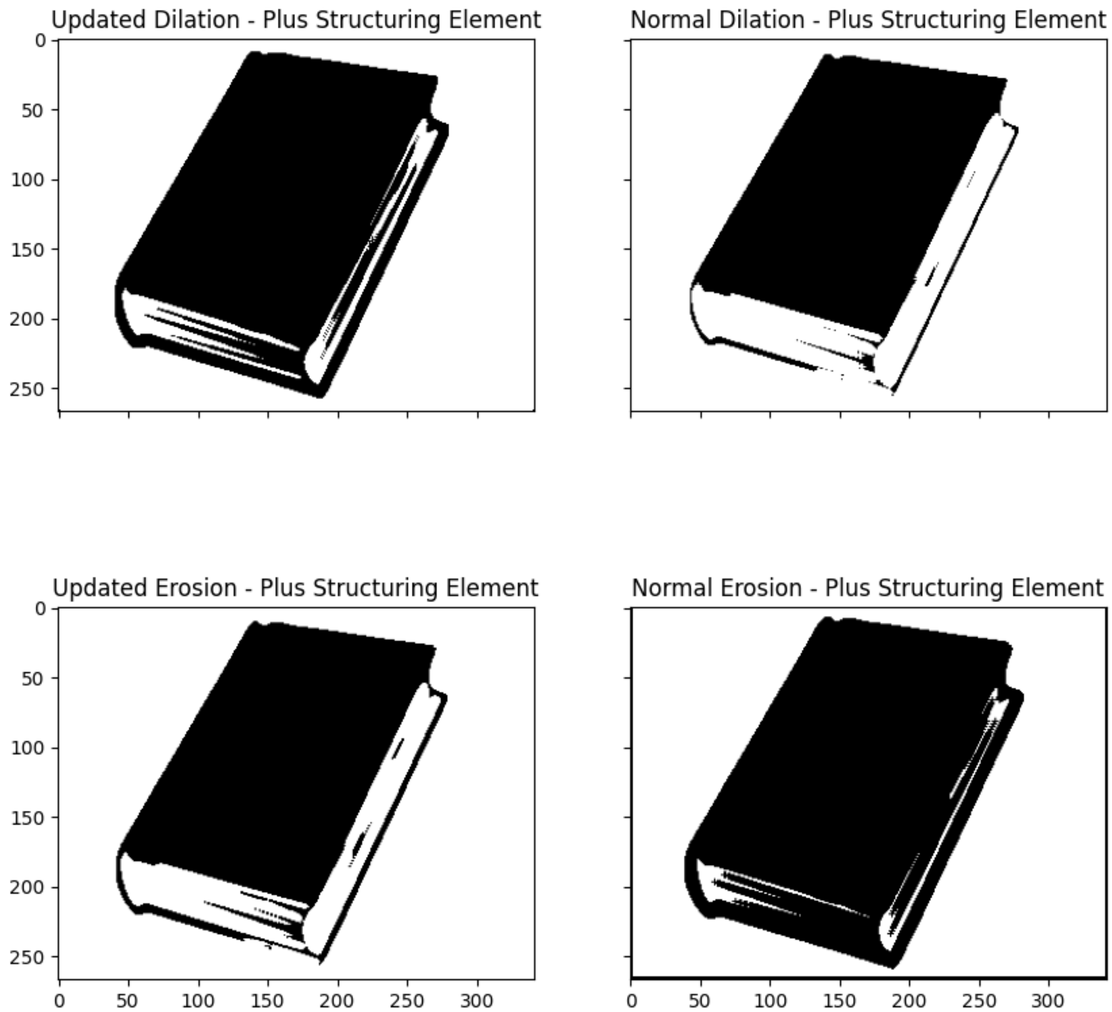


[247]:
```
result1 = updated_dilation(binary_image, 7, plus_struc_ele)
result2 = updated_erosion(binary_image, 7, plus_struc_ele)
result3 = normal_dilation(binary_image, 1, plus_struc_ele)
result4 = normal_erosion(binary_image, 25, plus_struc_ele)
```

[248]:
```
plus_struc_ele.size
```

[248]: 25

[249]:
```
display_images(result1, result2, result3, result4, "Plus Structuring Element")
```

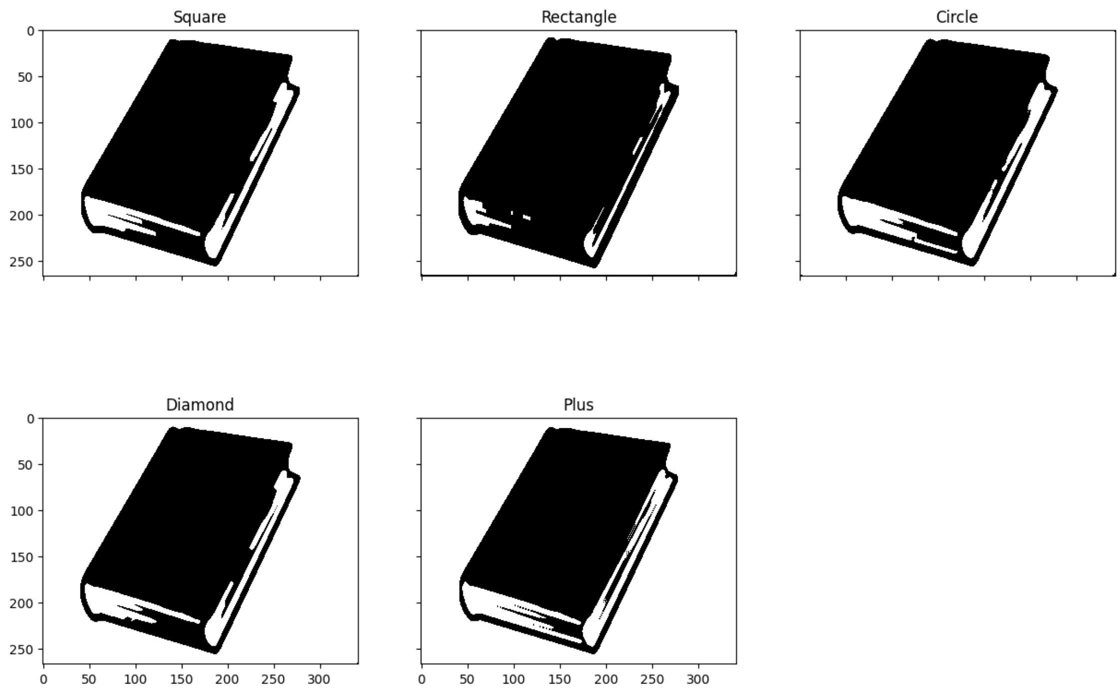


[250]:
```python
def dilate_then_erode(input_image, s1, s2, structuring_element):
    # Perform dilation
    dilated_image = updated_dilation(input_image, s1, structuring_element)

    # Perform erosion on the dilated image
    final_image = updated_erosion(dilated_image, s2, structuring_element)

    return final_image
```

[251]:
```python
result1 = dilate_then_erode(binary_image, 20,20, square_struc_ele)
result2 = dilate_then_erode(binary_image, 15,10, rectangle_struc_ele)
result3 = dilate_then_erode(binary_image, 20,20, circle_struc_ele)
result4 = dilate_then_erode(binary_image, 20, 20, diamond_struc_ele)
result5 = dilate_then_erode(binary_image, 7,7, plus_struc_ele)
```

[252]:

```python
# Create a figure with 2 rows and 3 columns
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

# Display images in each subplot with custom titles
axs[0, 0].imshow(result1, cmap='gray')
axs[0, 0].set_title('Square')

axs[0, 1].imshow(result2, cmap='gray')
axs[0, 1].set_title('Rectangle')

axs[0, 2].imshow(result3, cmap='gray')
axs[0, 2].set_title('Circle')

axs[1, 0].imshow(result4, cmap='gray')
axs[1, 0].set_title('Diamond')

axs[1, 1].imshow(result5, cmap='gray')
axs[1, 1].set_title('Plus')

# Hide the axes labels
for ax in axs.flat:
    ax.label_outer()

# Hide the last subplot (if necessary, adjust this based on your layout)
axs[1, 2].axis('off')

# Show the plot
plt.show()
```


Square
Rectangle
Circle
Diamond
Plus
0
50
100
150
200
250
300