আন্তর্জাতিক ইসলামী বিশ্ববিদ্যালয় চট্টগ্রাম

الجامعة الإسلامية العالمية شيتاغونغ

International Islamic University Chittagong

Lab 8

# Lab Report

Department: Computer Science and Engineering.

Course Code: CSE-4876

Course Title: Pattern Recognition & Image Processing

Submitted By:

Name: Farida Nusrat

ID: C201242

Semester: 8th

Section: 8AF

Submitted To:

Mr. Mohammad Mahadi Hasaan

Associate Professor,

Dept. of CSE, IIUC.

Date of Submission: 23 April,2024

**Lab No:** 08

**Lab Title:**

Temporal Processing

- Adding
- Subtracting
- Multiplying
- Signal Averaging
- AND
- OR
- XOR

Geometric Processing

- Clipping
- Rotating

**Introduction:** Temporal characterization occurs when you have a series of images taken at different time. Correlations between the images are often used to monitor the dynamic changes of the object. Geometric image transformation functions use mathematical transformations to crop, pad, scale, rotate, transpose or otherwise alter an image array to produce a modified view of an image.

When an image undergoes a geometric transformation, some or all of the pixels within the source image are relocated from their original spatial coordinates to a new position in the output image. When a relocated pixel does not map directly onto the center of a pixel location, but falls somewhere in between the centers of pixel locations, the pixel's value is computed by sampling the values of the neighboring pixels. This resampling, also known as interpolation, affects the quality of the output image.

In this report, our aim is to accomplish Temporal Processing (Adding, Subtraction, Multiplying, Signal Averaging, AND, OR, XOR) and Geometric Processing (Clipping, Rotating, Flipping, Zooming/Shrinking).

**Methodology:**

1. **Image Loading and Display**

   Two images (img1 and img2) are loaded using OpenCV's imread function. The images are displayed using cv2_imshow.

2. **Image Resizing**

   Both images are resized to a common size of 300x300 pixels using cv2.resize.

3. **Adding**

   The cv2.add function is used to perform element-wise addition of the two images (img1 and img2). The result of the addition operation is displayed using cv2_imshow.

4. **Subtracting**

   The cv2.subtract function is used to perform element-wise subtraction of img2 from img1.The result of the subtraction operation is displayed using cv2_imshow.

5. **Multiplying**

   The cv2.multiply function is used to perform element-wise multiplication of img1 and img2.The result of the multiplication operation is displayed using cv2_imshow.

6. **Signal Averaging**

   A weighted average of img1 and img2 is computed using cv2.addWeighted. The result of the averaging operation is displayed using cv2_imshow.

7. **Bitwise Operations (AND, OR, XOR)**

   Logical AND, OR, and XOR operations are performed using cv2.bitwise_and, cv2.bitwise_or, and cv2.bitwise_xor, respectively. The results of these operations are displayed using cv2_imshow.

8. **Geometric Processing – Clipping:**

   The np.clip function is used to clip the pixel values of an image (img) within a specified range (100 to 200 in this case).The clipped image is displayed alongside the original image using Matplotlib.

9. **Geometric Processing – Rotating:**

   Image Rotation (45, 90, 180 degrees): The cv2.getRotationMatrix2D function is used to generate an affine transformation matrix for rotation. The image is then rotated by 45, 90, and 180 degrees using cv2.warpAffine. Each rotated image is displayed alongside the original image using Matplotlib.

**Code:**

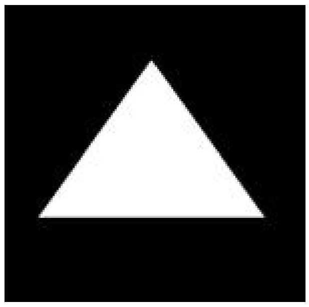
**Import Libraries**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

**Image Include**

```
img1 = cv2.imread("/content/c.png")
img2 = cv2.imread("/content/c1.png")
cv2_imshow(img1)
cv2_imshow(img2)
```

**Result:**



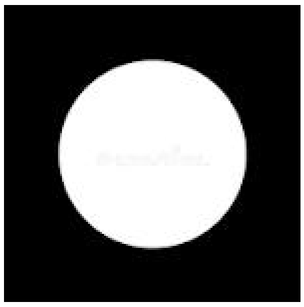
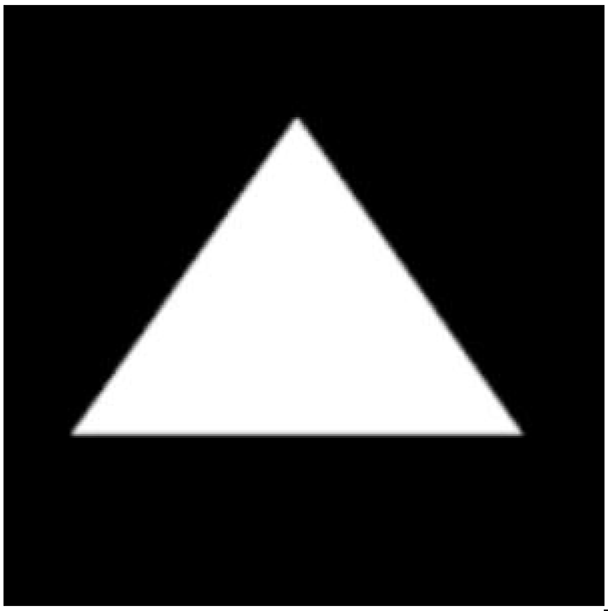
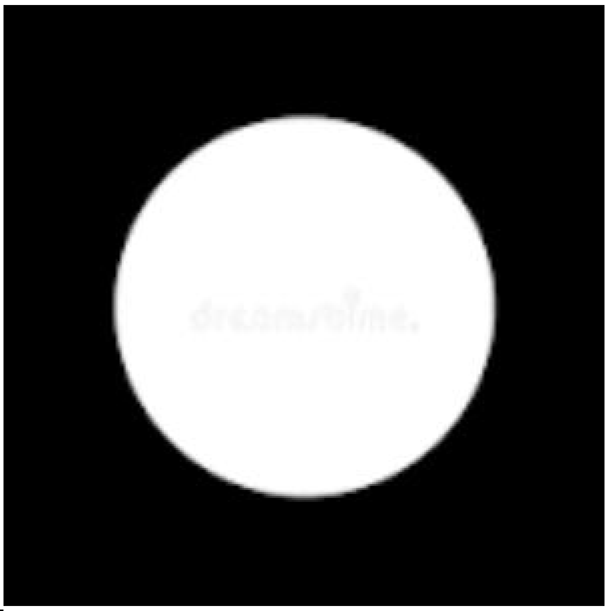


**Image Resizing**

```
img1 = cv2.resize(img1, (300,300))
img2 = cv2.resize(img2, (300,300))
cv2_imshow(img1)
cv2_imshow(img2)
```

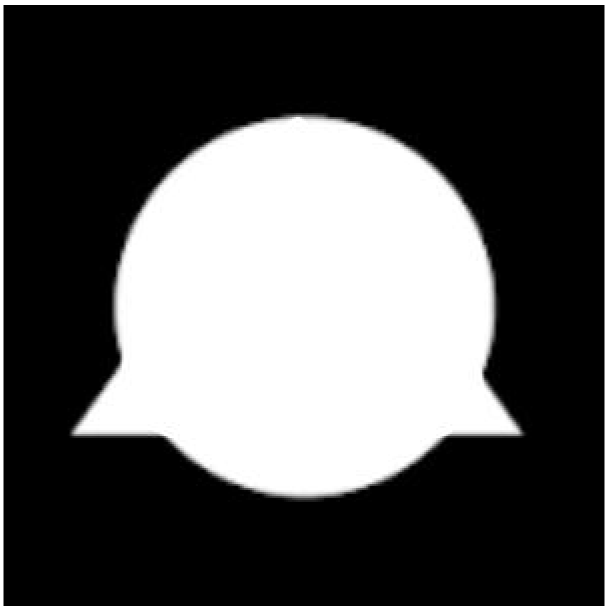
**Result:**

**Adding**

```
img_add = cv2.add(img1,img2)
cv2_imshow(img_add)
```

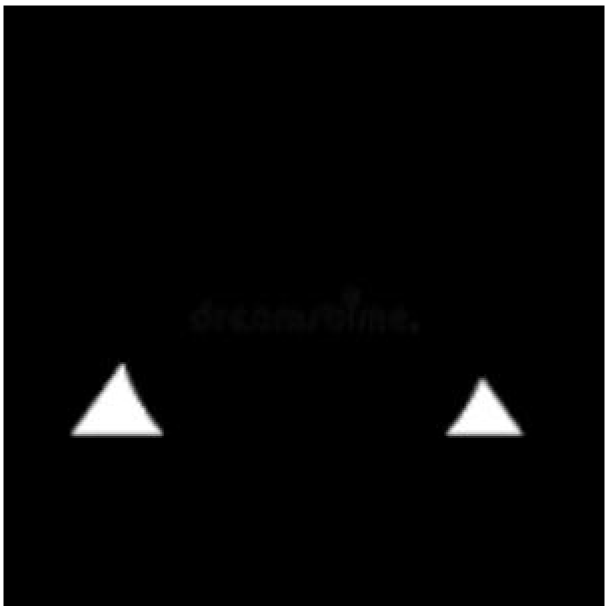
**Result:**



**Subtracting**

```
# Subtracting the images
img_subtract = cv2.subtract(img1, img2)


# Display the subtracted image
cv2_imshow(img_subtract)
```

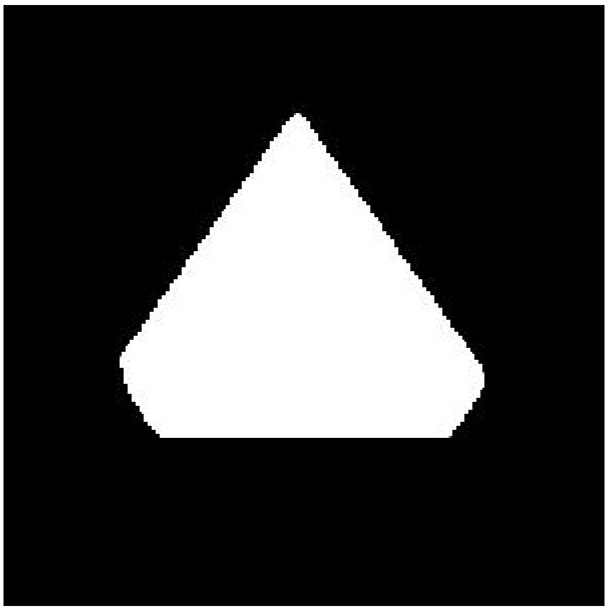
**Result:**



**Multiplying**

```
img_multiply = cv2.multiply(img1, img2)

# Display the multiplied image
cv2_imshow(img_multiply)
```

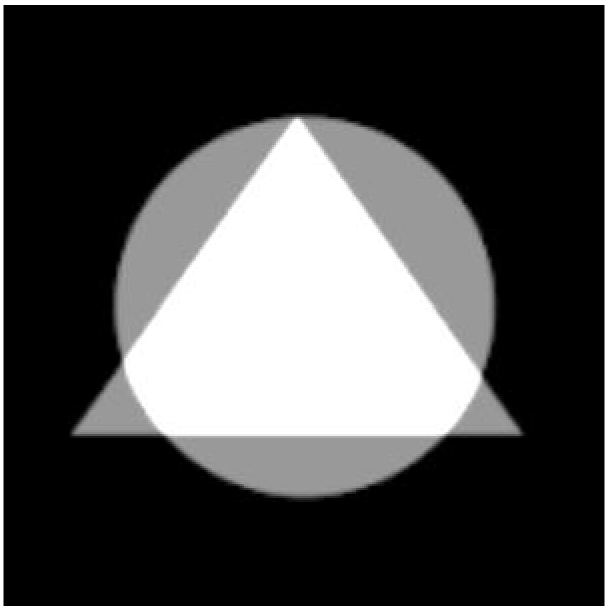
**Result:**



**Signal Averaging**

```
# Compute signal averaging
img_avg = cv2.addWeighted(img1, 0.6, img2, 0.6, 0)

# Display the averaged image
cv2_imshow(img_avg)
```

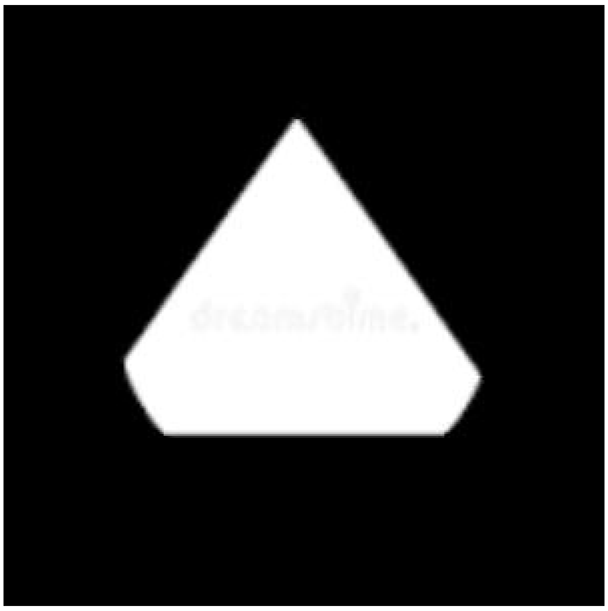
**Result:**



**AND**

```
# Perform logical AND operation
img_and = cv2.bitwise_and(img1, img2)

# Display the ANDed image
cv2_imshow(img_and)
```

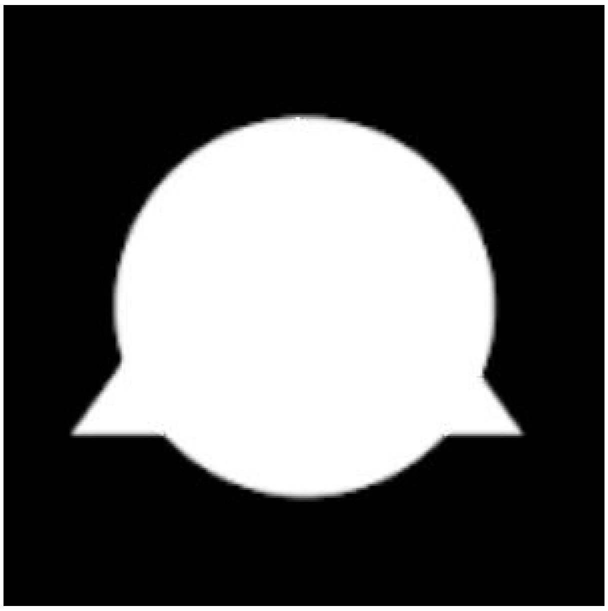
**Result:**



**OR**

```
# Perform logical OR operation
img_or = cv2.bitwise_or(img1, img2)

# Display the ANDed image
cv2_imshow(img_or)
```

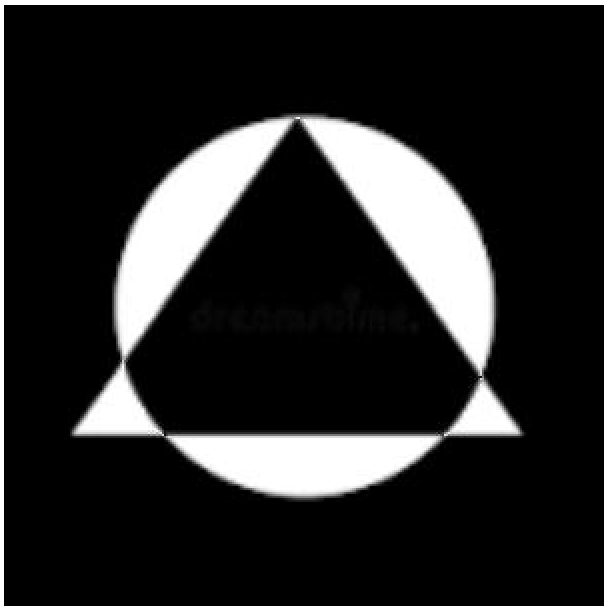
**Result:**



**XOR**

```
# Perform logical OR operation
img_Xor = cv2.bitwise_xor(img1, img2)

# Display the ANDed image
cv2_imshow(img_Xor)
```

**Result:**

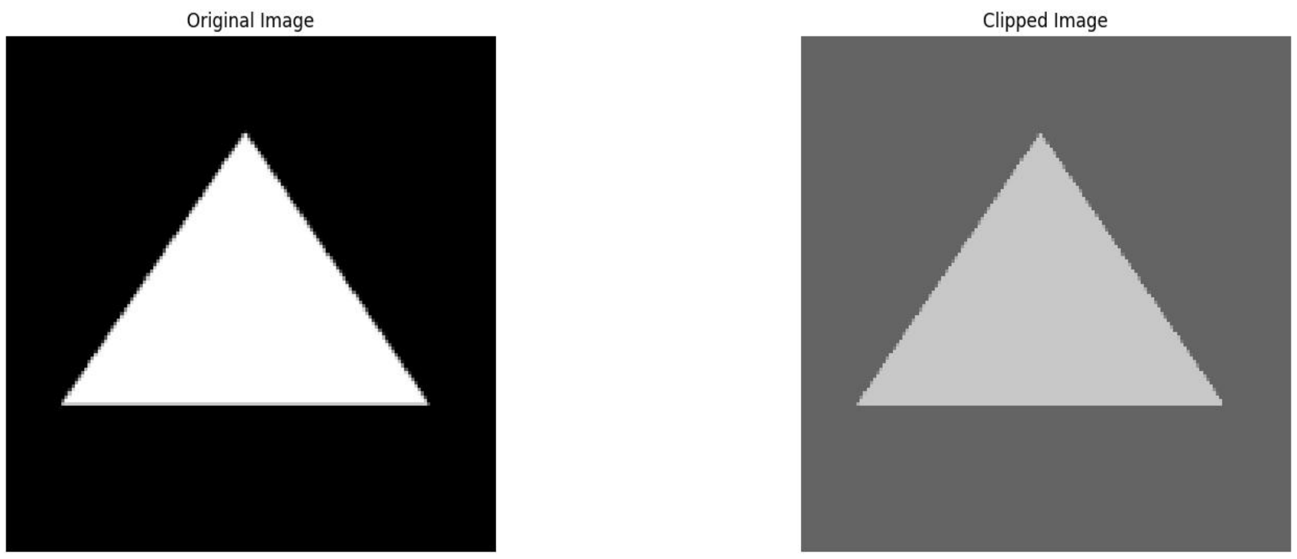


**Geometric Processing – Clipping**

```
# Read the image
img = cv2.imread('/content/c.png')

# Clipping
clipped_img = np.clip(img, 100, 200)  # Adjust the range as needed
```

```
# Display results
plt.figure(figsize=(18, 6))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(clipped_img, cv2.COLOR_BGR2RGB))
plt.title('Clipped Image')
plt.axis('off')
```

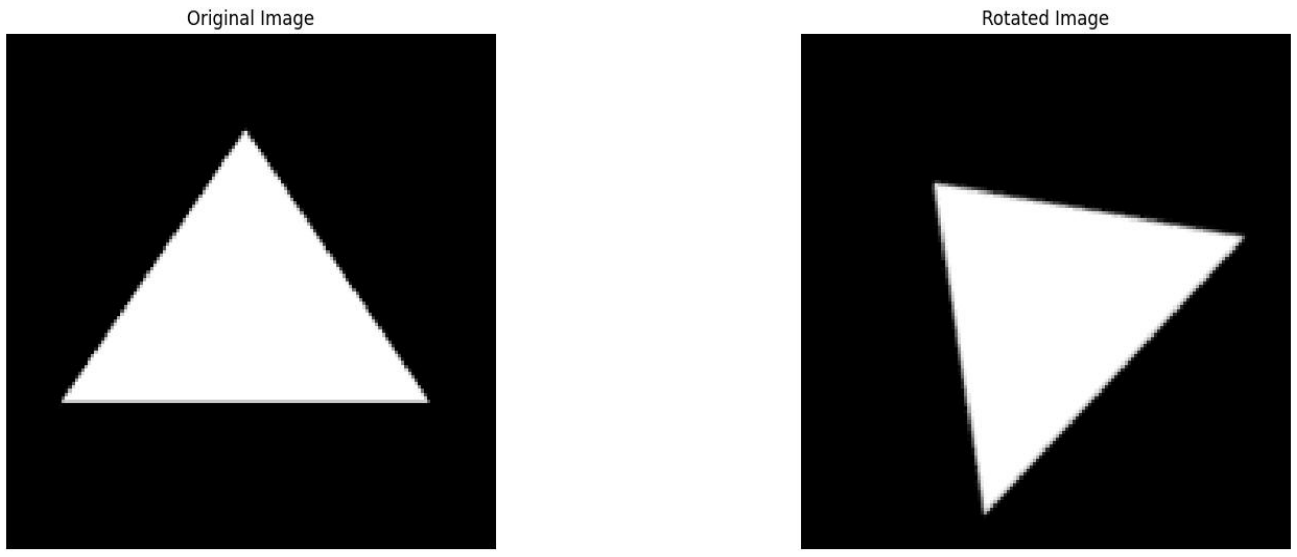
**Result:**



**Rotating 45°**

```
rows, cols = img.shape[:2]

M = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 1)  # Rotate 45 degrees

rotated_img = cv2.warpAffine(img, M, (cols, rows))

# Display results

plt.figure(figsize=(18, 6))

plt.subplot(1, 2, 1)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

plt.title('Original Image')

plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(rotated_img, cv2.COLOR_BGR2RGB))


plt.title('Rotated Image')

plt.axis('off')
```

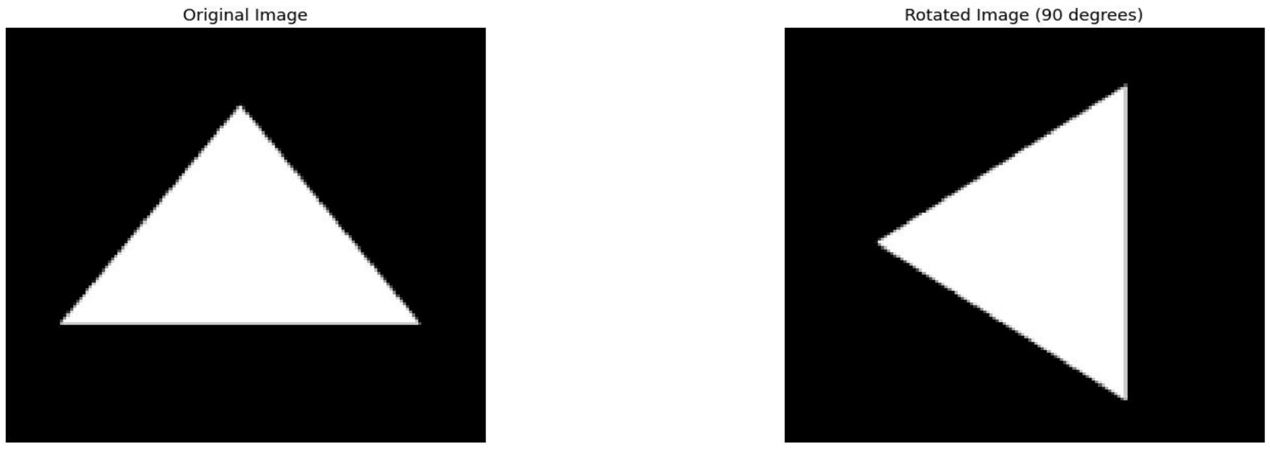
**<u>Result:</u>**



**<u># Rotating 90°</u>**

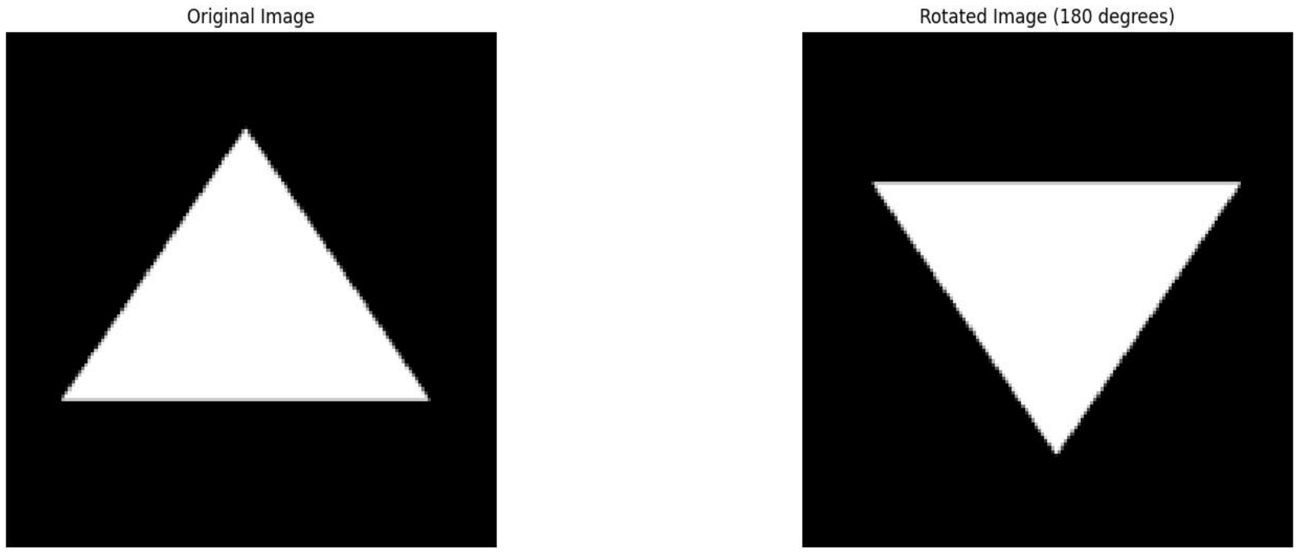
```
rows, cols = img.shape[:2]
M = cv2.getRotationMatrix2D((cols/2, rows/2), 90, 1)  # Rotate 90 degrees
rotated_img = cv2.warpAffine(img, M, (cols, rows))

# Display results
plt.figure(figsize=(18, 6))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(rotated_img, cv2.COLOR_BGR2RGB))
plt.title('Rotated Image (90 degrees)')
plt.axis('off')
```

**<u>Result:</u>**



```
# Rotating 180°
rows, cols = img.shape[:2]
M = cv2.getRotationMatrix2D((cols/2, rows/2), 180, 1)  # Rotate 180 degrees
rotated_img = cv2.warpAffine(img, M, (cols, rows))

# Display results
plt.figure(figsize=(18, 6))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(rotated_img, cv2.COLOR_BGR2RGB))
plt.title('Rotated Image (180 degrees)')
plt.axis('off')
```

**Result:**



**Discussion:** The results demonstrate the effectiveness of various image processing techniques. Resizing allows adjustment of image dimensions, while arithmetic operations manipulate pixel values for desired effects such as brightness adjustment or blending. Signal averaging helps in noise reduction, while logical operations enable image masking and feature extraction. Geometric processing facilitates spatial manipulation of images for tasks like cropping and rotation.