Experiment No. 10

Title - Design, Implement and Analyse Image Processing with an example

---

Theory -

Image processing is a field in computer vision that involves analysing, manipulating, and enhancing digital images to extract useful information or improve their visual quality. Here's an in-depth overview covering the steps, importance, advantages, and mathematical backgrounds in image processing:

Steps in Image Processing

1. Image Acquisition: Capture images using cameras, scanners, or load existing images from files.
2. Preprocessing: Includes steps like noise reduction, colour space conversions, and image enhancement.
3. Segmentation: Partition the image into meaningful regions or objects.
4. Feature Extraction: Identify and extract relevant features like edges, textures, or shapes.
5. Object Recognition/Classification: Identify objects or patterns using extracted features.
6. Post-processing: Restore or reconstruct images and visualise processed results.

Significance

- Medical Imaging: Diagnosing diseases through X-rays, MRIs, etc.
- Remote Sensing: Satellite imagery for weather forecasting, agriculture, etc.
- Security and Surveillance: Object detection, facial recognition, etc.
- Entertainment and Media: Image editing, video processing, etc.

Advantages

- Information Extraction: Reveals hidden details, patterns, or structures.
- Automation: Facilitates automated analysis or decision-making.
- Enhanced Visualisation: Improves visual quality for better human interpretation.
- Efficiency: Speeds up processes that would be time-consuming manually.

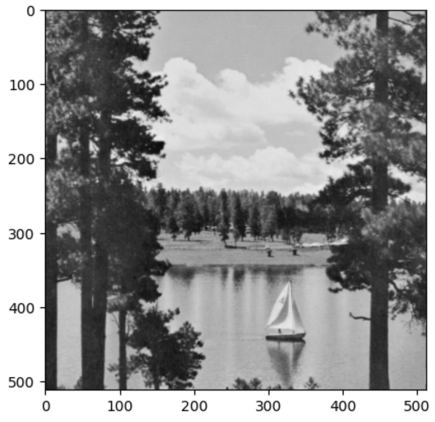
Mathematical Background

- Linear Algebra: Matrix operations for image transformations (scaling, rotation, shear, blur etc.).
- Calculus: Differential equations for edge detection (Sobel, Prewitt, Canny, etc.).
- Probability & Statistics: Noise reduction, filtering, and pattern recognition.
- Fourier Transform: Frequency domain analysis for signal representation.

Implementation

```
# Import Libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
#Load image
image=cv2.imread('lake.tif')
print(image.shape)
plt.imshow(image,cmap='gray')
```

```
> (512, 512, 3)
```



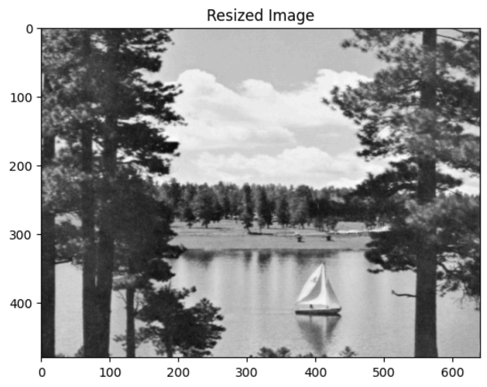
```
# Grayscale Conversion
gray_image=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
print(gray_image.shape)
```

```
> (512, 512)
```

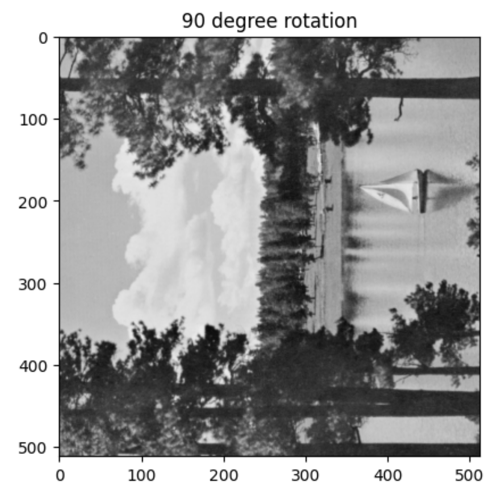
```
# Image Resizing
new_size = (640, 480)
# Resize the image
resized_image = cv2.resize(gray_image, new_size, interpolation=cv2.INTER_AREA)
print(resized_image.shape)
```

```
> (480, 640)
```

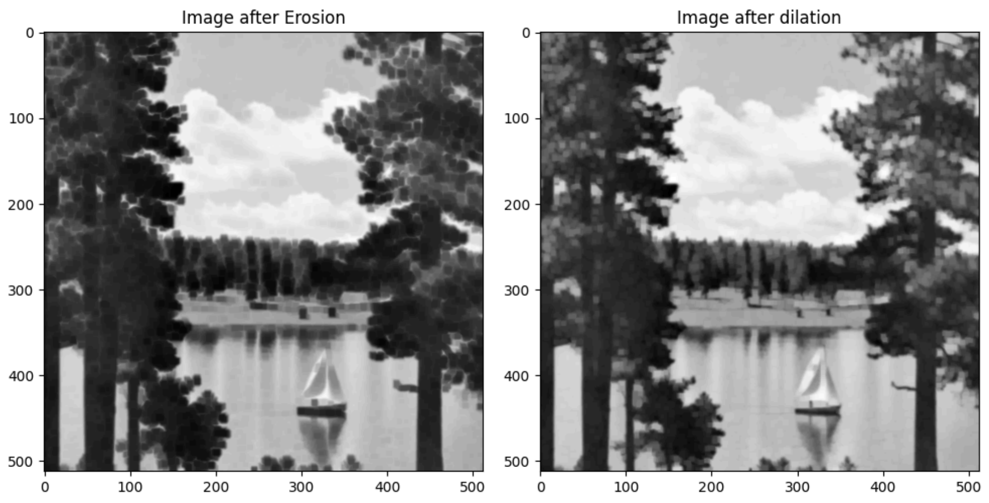
```
plt.imshow(resized_image,cmap='gray')
plt.title('Resized Image')
```



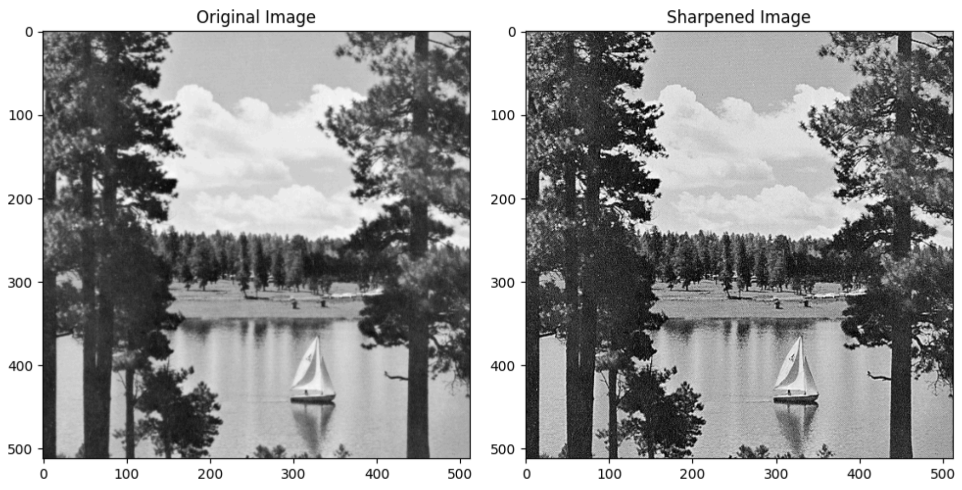
```
# Image Rotation
# Define the angle and center of rotation
angle = 90
center = (image.shape[1] // 2, image.shape[0] // 2)

# Calculate the rotation matrix
rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1)

# Rotate the image
rotated_image = cv2.warpAffine(image, rotation_matrix, (image.shape[1], image.shape[0]))

# Show the original and rotated images
plt.title('90 degree rotation')
plt.imshow(rotated_image,cmap='gray')
```



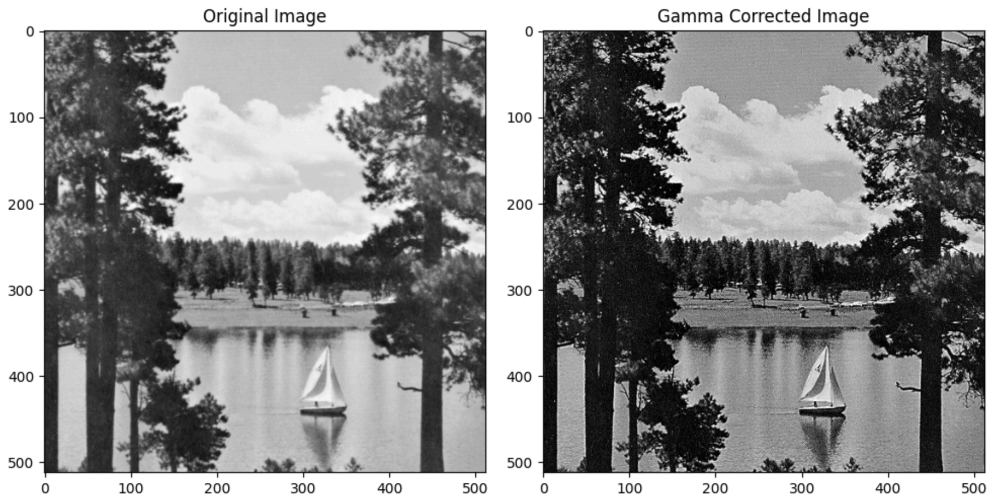
```
# Apply dilation and erosion to remove noise
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

erosion_kernel = np.ones((5, 5), np.uint8)
erosion_image = cv2.erode(blurred_image, erosion_kernel, iterations=1)

dilation_kernel = np.ones((5, 5), np.uint8)
dilation_image = cv2.dilate(erosion_image, dilation_kernel, iterations=1)

plt.figure(figsize=(10,15))
plt.subplot(1,2,1)
plt.title('Image after Erosion')
plt.imshow(erosion_image,cmap='gray')
plt.subplot(1,2,2)
plt.title('Image after dilation')
plt.imshow(dilation_image,cmap='gray')
plt.tight_layout()
```



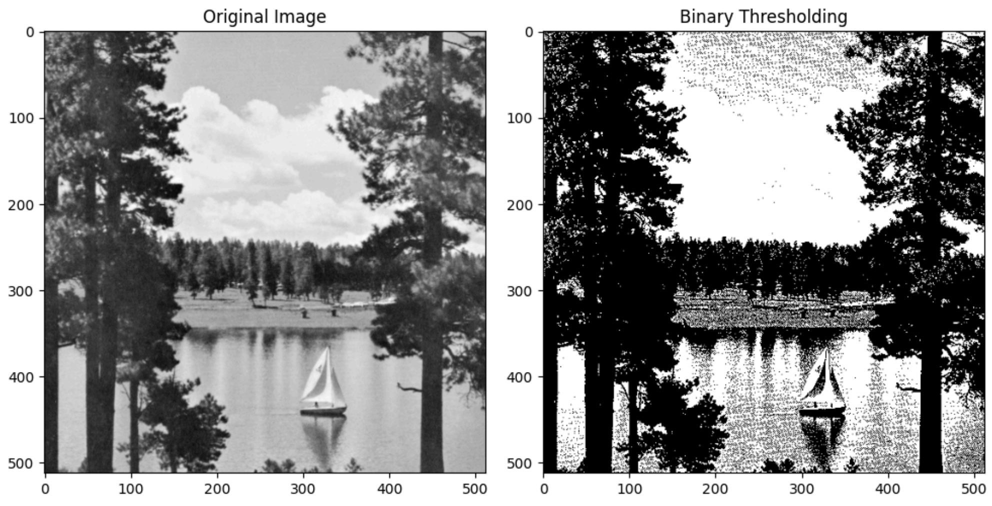
```
# Image Sharpening
# Define the sharpening kernel (Laplacian kernel)
sharpening_kernel = np.array([[0, -1, 0],
                              [-1, 5, -1],
                              [0, -1, 0]])
# Apply the sharpening kernel using filter2D
sharpened_image = cv2.filter2D(gray_image, -1, sharpening_kernel)

# Display the original and sharpened images
plt.figure(figsize=(10,15))
plt.subplot(1,2,1)
plt.title('Original Image')
plt.imshow(gray_image,cmap='gray')
plt.subplot(1,2,2)
plt.title('Sharpened Image')
plt.imshow(sharpened_image,cmap='gray')
plt.tight_layout()
```



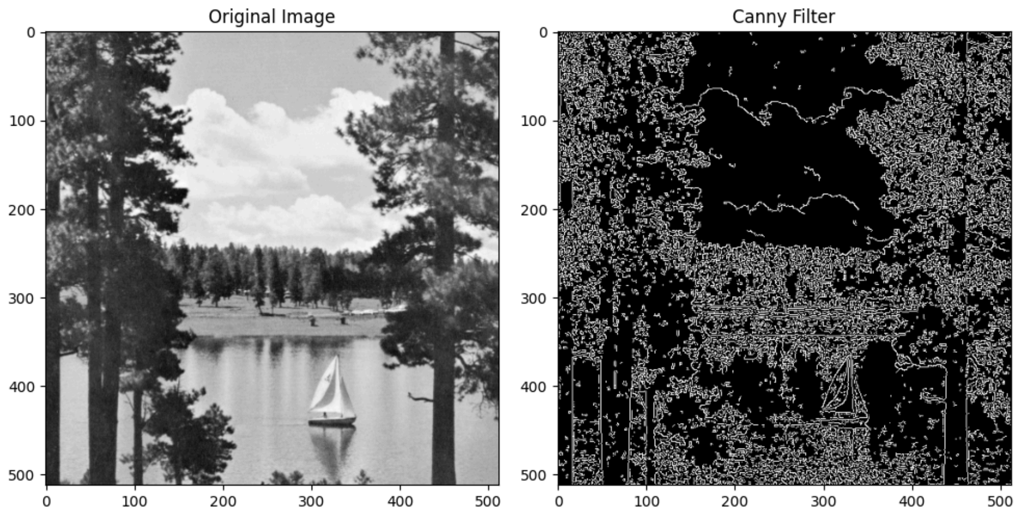
```
# Gamma Correction
# Apply power law transformation
gamma_corrected = np.power(sharpened_image / 255.0, 1.5) * 255.0
gamma_corrected = np.uint8(gamma_corrected)

plt.figure(figsize=(10,15))
plt.subplot(1,2,1)
plt.title('Original Image')
plt.imshow(gray_image,cmap='gray')
plt.subplot(1,2,2)
plt.title('Gamma Corrected Image')
plt.imshow(gamma_corrected,cmap='gray')
plt.tight_layout()
```



```
# Binary Thresholding
# Use Binary thresholding to manipulate each pixel values
threshold_value = 127
thresholded_image = np.where(gamma_corrected < threshold_value, 0, 255)
plt.figure(figsize=(10,15))
plt.subplot(1,2,1)
plt.title('Original Image')
plt.imshow(gray_image,cmap='gray')
plt.subplot(1,2,2)
plt.title('Binary Thresholding')
plt.imshow(thresholded_image,cmap='gray')
plt.tight_layout()
```



```
# Use canny filter to detect edges

edges = cv2.Canny(gamma_corrected.astype(np.uint8), threshold1=100, threshold2=200)

plt.figure(figsize=(10,15))
plt.subplot(1,2,1)
plt.title('Original Image')
plt.imshow(gray_image,cmap='gray')
plt.subplot(1,2,2)
plt.title('Canny Filter')
plt.imshow(edges,cmap='gray')
plt.tight_layout()
```



Conclusion

Image processing encompasses a broad range of techniques and algorithms, combining various mathematical theories and computational methodologies. It plays a pivotal role across diverse fields, offering solutions for analysis, interpretation, and manipulation of digital images. The continuous advancements in algorithms and computational power further expand its applications and potential impact in numerous industries and everyday life.