Practical Lesson 3

Topic: Learning Preliminary Processing of Two-Dimensional Signals in MATLAB/Python

1. Purpose of the Practical Work

- To study basic techniques of preliminary processing of 2D signals (images).
- To implement simple operations such as reading, displaying, resizing, grayscale conversion, and filtering.
- To develop practical programming skills in MATLAB/Python for image processing.

1. Introduction to Two-Dimensional Signals

Signal processing is one of the most fundamental areas in information technology, telecommunications, and computer science. Signals can exist in one dimension (such as audio signals), two dimensions (such as images), or even more dimensions in advanced applications like 3D medical imaging or video processing. A two-dimensional (2D) signal is a signal that varies over two independent variables. Most commonly, these variables represent spatial coordinates, such as the horizontal and vertical directions of an image. Thus, digital images are the most well-known and widely studied type of 2D signal.

A digital image can be described as a finite two-dimensional array of intensity values, where each element of the array corresponds to a pixel. Each pixel carries information about the brightness or color at a specific point in space. Because of this structure, image processing can be studied as a branch of two-dimensional signal processing.

Two-dimensional signal processing has wide applications in fields such as:

- Medical imaging (X-rays, CT scans, MRI)
- Remote sensing (satellite imagery, aerial photography)
- Telecommunications (video transmission, error correction, compression)
- Industrial automation (quality inspection of products, robotics vision)

- Artificial intelligence and machine learning (object detection, face recognition, autonomous driving)

Before advanced operations such as feature extraction, segmentation, or recognition can be performed, images need to be preprocessed. Preprocessing ensures that the raw data becomes suitable for further analysis by reducing noise, improving quality, and transforming it into the required format. This stage is called preliminary processing of 2D signals.

2. Representation of Two-Dimensional Signals

Mathematically, a 2D signal can be represented as a function:

$f(x,y)$

where x and y are spatial coordinates, and the function value $f(x,y)$ represents intensity (brightness) at that location.

- In grayscale images, intensity values are typically integers ranging from 0 (black) to 255 (white).
- In binary images, values are either 0 or 1.
- In color images, each pixel is represented by three intensity values corresponding to the Red, Green, and Blue (RGB) channels.

For example, a color image with resolution 640×480 has 640 columns and 480 rows, and each pixel contains three values, forming a 3D data structure.

In computer memory, images are stored as matrices (in MATLAB) or as NumPy arrays (in Python). This makes them convenient to manipulate using linear algebra and matrix-based operations.

3. Types of Images

Before discussing preprocessing, it is essential to understand the different types of digital images used in 2D signal processing.

3.1 Binary Images

- Contain only two possible values: 0 (black) and 1 (white).
- Used for shape analysis, masks, or object detection after segmentation.
- Example: Document scanning for character recognition.

3.2 Grayscale Images

- Represented by a single intensity channel, ranging from 0–255.
- Easier to process compared to color images because only one channel is involved.
- Example: Medical images (X-ray scans).

3.3 Color Images

- Represented by three channels: Red, Green, Blue (RGB).
- Each pixel is a combination of three intensity values.
- More complex for processing but more informative.
- Example: Natural photographs.

3.4 Other Image Types

- Indexed images (use colormaps).
- Multispectral or hyperspectral images (contain multiple channels, used in remote sensing).

4. Importance of Image Preprocessing

Raw images often contain distortions or imperfections due to various factors such as:

- Noise: Caused by sensors, transmission errors, or environmental factors.
- Low contrast: Makes it difficult to distinguish details.
- Uneven brightness: Due to poor lighting conditions.
- Blurriness: Result of camera focus issues or motion.

- Redundancy: Images may be too large in resolution, requiring resizing for efficient processing.

Preprocessing ensures:

1. Noise reduction → Cleaner signals.
2. Image enhancement → Easier feature detection.
3. Standardization → Preparing images to a fixed size and intensity range.
4. Improved accuracy in further processing stages (e.g., segmentation, recognition).

Thus, preprocessing is a critical first step before applying any advanced image processing techniques.

5. Common Image Preprocessing Operations

In practical applications, several operations are used during the preliminary stage of image processing. The most essential ones are discussed below.

5.1 Reading and Displaying an Image

The first step is to import an image into the programming environment.

- In MATLAB: imread() is used for reading images, and imshow() for displaying.
- In Python (OpenCV): cv2.imread() is used for reading, and cv2.imshow() for displaying.

Example in MATLAB:

```
img = imread('image.jpg');
imshow(img);
```

Example in Python:

```
import cv2
```

```
from google.colab.patches import cv2_imshow


img =
cv2.imread('/content/drive/MyDrive/Fundamentals_of_Image_processing/Data
(1)/smallproject/pano1.jpg')

cv2_imshow(img)
```

5.2 Converting to Grayscale

Grayscale conversion reduces complexity by transforming a color image (3 channels) into a single-channel image. This is useful when color is not important.

- Formula for grayscale conversion (weighted average):

Gray=0.2989R+0.5870G+0.1140BGray = 0.2989R + 0.5870G + 0.1140BGray=0.2989R+0.5870G+0.1140B

This reflects human perception, as the eye is more sensitive to green than red or blue.

- MATLAB function: rgb2gray()
- Python (OpenCV): cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

5.3 Resizing and Cropping

Images may need to be resized for computational efficiency or for matching dimensions with other datasets.

- Resizing changes the overall size of the image.
- Cropping extracts a region of interest (ROI).

Applications:

- Face recognition (crop faces from photos).
- Medical imaging (focus on tumor region).

- Object detection in robotics.

Code:

```
import cv2
img = cv2.imread('/content/drive/MyDrive/Fundamentals_of_Image_processing/Data (1)/smallproject/pano1.jpg')
resized_img = cv2.resize(img, (900, 1600))  # width va height kerakli o‘lchamlar
print("Resizied Image:")
cv2_imshow(resized_img)
```

5.4 Noise in Images

Noise is any unwanted random variation in pixel values. Sources of noise include:

- Salt-and-pepper noise: Random white and black pixels.
- Gaussian noise: Statistical variations in intensity.
- Speckle noise: Common in radar and ultrasound images.

Noise must be reduced before analysis.

5.5 Noise Removal Using Filters

Filtering is the process of modifying an image to emphasize certain features or reduce unwanted ones.

a) Mean Filter

- Averages pixel values in a local neighborhood.

- Smooths the image but may blur edges.

b) Gaussian Filter

- Uses a weighted average with Gaussian distribution.
- Provides smoother results than the mean filter.

c) Median Filter

- Replaces each pixel with the median of its neighbors.
- Very effective against salt-and-pepper noise.

Python Example:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


img = cv2.imread('/content/drive/MyDrive/Fundamentals_of_Image_processing/Data
(1)/smallproject/pano1.jpg')

img = cv2.resize(img, (400, 300))  #


def add_salt_pepper_noise(image, prob):

    output = np.copy(image)

    black = 0

    white = 255

    probs = np.random.rand(*image.shape[:2])


    output[probs < prob] = black



    output[probs > 1 - prob] = white
```

```
    return output

noisy_img = add_salt_pepper_noise(img, prob=0.02)  # 2% noise

denoised_img = cv2.medianBlur(noisy_img, 3)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
noisy_rgb = cv2.cvtColor(noisy_img, cv2.COLOR_BGR2RGB)
denoised_rgb = cv2.cvtColor(denoised_img, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(15,5))

plt.subplot(1,3,1)
plt.imshow(img_rgb)
plt.title('Original Image')
plt.axis('off')

plt.subplot(1,3,2)
plt.imshow(noisy_rgb)
plt.title('Salt-and-Pepper Noise')
plt.axis('off')

plt.subplot(1,3,3)
plt.imshow(denoised_rgb)
plt.title('After Median Filtering')
plt.axis('off')

plt.show()
```

5.6 Brightness and Contrast Adjustment

- Brightness adjustment → Adds or subtracts a constant value to all pixels.
- Contrast adjustment → Expands the range of intensity values.

For example, an image with pixel values between 100–150 can be stretched to 0–255, making details clearer.

Applications:

- Medical imaging (enhancing tumors in X-ray scans).
- Satellite imaging (improving visibility of terrain).

5. Control Questions

1. What is a two-dimensional signal?
2. Why is preprocessing important in image processing?
3. What is the difference between grayscale and color images?
4. Which function is used to resize images in MATLAB/Python?
5. What is the purpose of a median filter?
6. What is the difference between Gaussian and median filtering?
7. How do you display an image in MATLAB and Python?