
 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Design Butterworth and Chebyshev filter using the bilinear transformation method.	
Experiment No: 04	Date:	Enrollment No: 92200133030

Aim: Design Butterworth and Chebyshev filters using the bilinear transformation method.

Theory:-

- The bilinear transformation method is commonly used to design analog filters and then convert them into digital filters. This method maps the analog frequency response to the digital frequency response using a bilinear transformation.
- The Butterworth and Chebyshev filters are two commonly used filter types. The Butterworth filter has a maximally flat frequency response in the passband, while the Chebyshev filter allows for a sharper transition between the passband and the stopband at the expense of ripples in either the passband or stopband.
- The steps involved in designing Butterworth and Chebyshev filters using the bilinear transformation method are as follows:
- Specify the desired filter specifications, such as the filter order, cutoff frequency, and filter type (Butterworth or Chebyshev).
- Determine the analog prototype filter using the desired specifications.
- Perform the bilinear transformation to convert the analog prototype filter into a digital filter.
- Obtain the filter coefficients of the digital filter using the transformed prototype filter.
- Plot the filter's magnitude response and impulse response.
- Save the filter coefficients (optional).
- Flowchart:
- The flowchart for the program will consist of the following steps:
- Specify the desired filter specifications, such as the filter order, cutoff frequency, and filter type.
- Design the analog prototype filter using the `scipy.signal.butter` or `scipy.signal.cheby1` function.
- Perform the bilinear transformation using the `scipy.signal.bilinear` function to convert the analog filter to a digital filter.
- Obtain the filter coefficients of the digital filter.
- Plot the filter's magnitude response and impulse response.
- Save the filter coefficients (optional).
- Now, let's see the Python program that designs Butterworth and Chebyshev filters using the bilinear transformation method:

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Design Butterworth and Chebyshev filter using the bilinear transformation method.	
Experiment No: 04	Date:	Enrollment No: 92200133030

Programm:-

```

import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import bilinear, butter, cheby1, freqz, lfilter

def design_butterworth_filter(filter_order, cutoff_frequency, sampling_frequency):
    # Normalize the cutoff frequency (cutoff_frequency / Nyquist frequency)
    nyquist_frequency = sampling_frequency / 2
    normalized_cutoff = cutoff_frequency / nyquist_frequency

    # Design the analog Butterworth filter
    analog_b, analog_a = butter(
        filter_order, normalized_cutoff, analog=False, btype="low"
    )

    # Perform the bilinear transformation
    digital_b, digital_a = bilinear(analog_b, analog_a, sampling_frequency)

    return digital_b, digital_a

def design_chebyshev_filter(filter_order, cutoff_frequency, sampling_frequency, ripple):
    # Normalize the cutoff frequency
    nyquist_frequency = sampling_frequency / 2
    normalized_cutoff = cutoff_frequency / nyquist_frequency

    # Design the analog Chebyshev filter
    analog_b, analog_a = cheby1(
        filter_order, ripple, normalized_cutoff, analog=False, btype="low"
    )

    # Perform the bilinear transformation
    digital_b, digital_a = bilinear(analog_b, analog_a, sampling_frequency)


    return digital_b, digital_a

def plot_filter_response(digital_b, digital_a, sampling_frequency):
    # Compute the frequency response of the filter
    frequency, magnitude_response = freqz(digital_b, digital_a, fs=sampling_frequency)

    # Plot the magnitude response
    plt.figure(figsize=(10, 6))
    plt.plot(frequency, np.abs(magnitude_response))
    plt.title("Filter Magnitude Response")
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude")
    plt.grid(True)
    plt.show()

    # Compute and plot the impulse response
    impulse = np.zeros(100)
    impulse[0] = 1 # Create a unit impulse

```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Design Butterworth and Chebyshev filter using the bilinear transformation method.	
Experiment No: 04	Date:	Enrollment No: 92200133030

```

impulse_response = lfilter(digital_b, digital_a, impulse)

plt.figure(figsize=(10, 6))
plt.plot(impulse_response)
plt.title("Filter Impulse Response")
plt.xlabel("Samples")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()


# Specify the desired filter specifications
filter_order = 4 # Filter order
cutoff_frequency = 1000 # Cutoff frequency in Hz
sampling_frequency = 8000 # Sampling frequency in Hz
ripple = 0.5 # Ripple factor for Chebyshev filter

# Design the Butterworth filter
digital_b, digital_a = design_butterworth_filter(
    filter_order, cutoff_frequency, sampling_frequency
)
# Plot the Butterworth filter's magnitude response and impulse response
plot_filter_response(digital_b, digital_a, sampling_frequency)

# Design the Chebyshev filter
digital_b, digital_a = design_chebyshev_filter(
    filter_order, cutoff_frequency, sampling_frequency, ripple
)
# Plot the Chebyshev filter's magnitude response and impulse response
plot_filter_response(digital_b, digital_a, sampling_frequency)

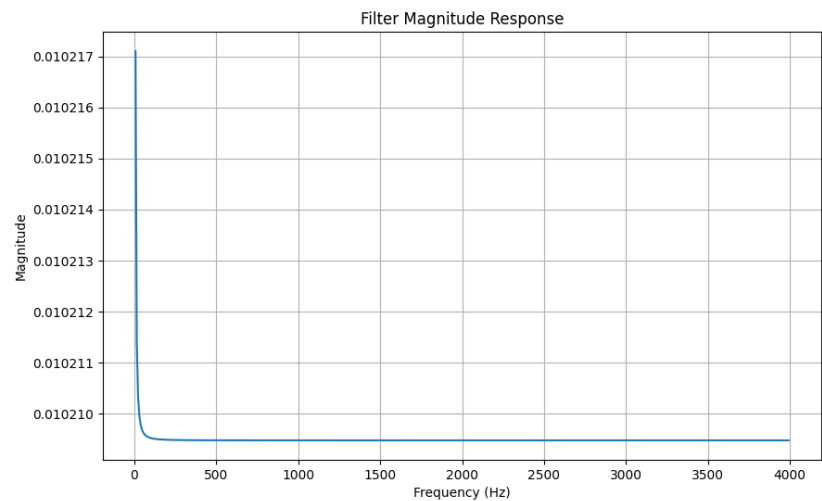
# Save the filter coefficients (optional)
filter_path = "filter_coefficients.txt"
np.savetxt(filter_path, np.vstack((digital_b, digital_a)), delimiter=",")
print(f"Filter coefficients saved at: {filter_path}")

```

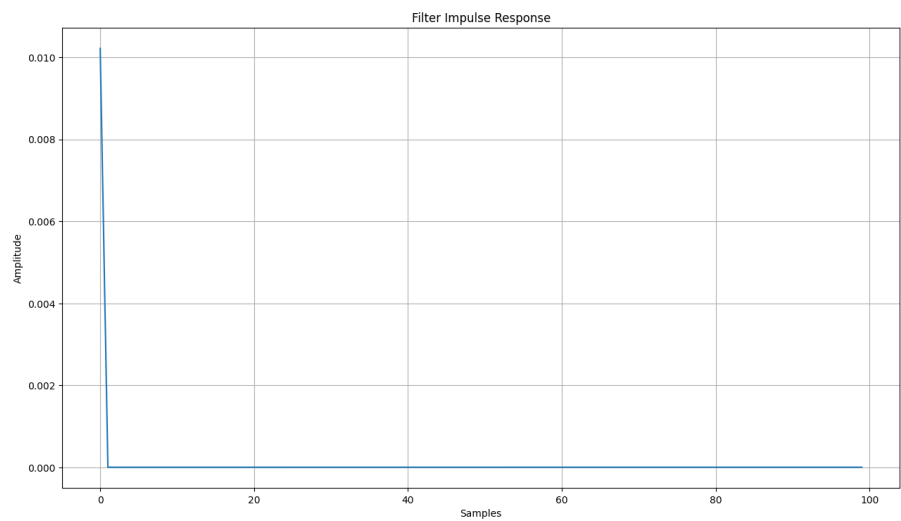
 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Design Butterworth and Chebyshev filter using the bilinear transformation method.	
Experiment No: 04	Date:	Enrollment No: 92200133030


Output :-

Butterworth Filter Magnitude Response :-

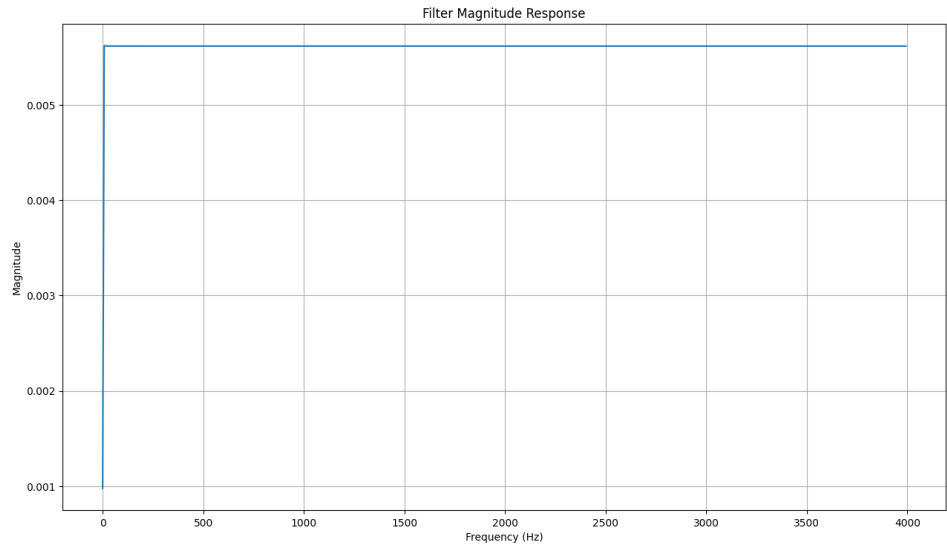


Butterworth Filter Impulse Response :-

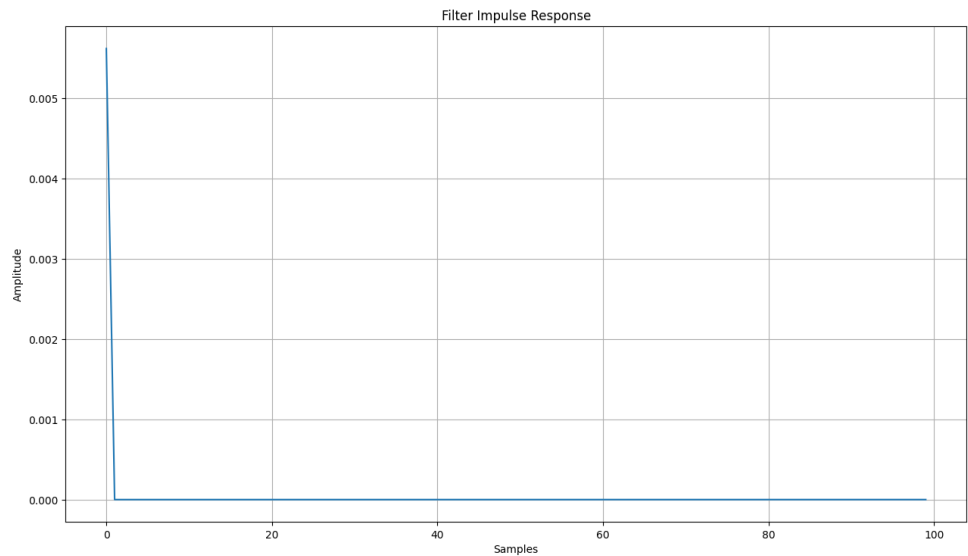



 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Design Butterworth and Chebyshev filter using the bilinear transformation method.	
Experiment No: 04	Date:	Enrollment No: 92200133030

Chebyshev Filter Magnitude Response :-




Chebyshev Filter Impulse Response :-



 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Design Butterworth and Chebyshev filter using the bilinear transformation method.	
Experiment No: 04	Date:	Enrollment No: 92200133030

Filter Coefficients :-

5.621724098249910630e-03,-2.248408570661841743e-02,3.372191305732654548e-02,-2.247846538775357186e-02,5.618913938795535787e-03
1.0000000000000000e+00,-4.000320181991981805e+00,6.000960591637167774e+00,-4.000960637301628431e+00,1.000320227656443350e+00

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Design FIR filter with windowing method.	
Experiment No: 05	Date:	Enrollment No: 92200133030

Aim: Design FIR filter with windowing method.

Theory:-

- The windowing method is a commonly used technique for designing FIR filters. It involves designing an ideal frequency response and then applying a window function to obtain a practical FIR filter.
- The steps involved in designing an FIR filter using the windowing method are as follows:
- Specify the desired filter specifications, such as the cutoff frequency, filter length, and window type.
- Design an ideal frequency response that meets the desired specifications.
- Apply a window function to the ideal frequency response to obtain the filter coefficients.
- Normalize the filter coefficients to ensure stability.
- There are various window functions available, such as the rectangular, Bartlett, Hann, Hamming, and Blackman windows, among others. The choice of window function depends on the desired trade-off between the main lobe width and sidelobe suppression.

Program:-

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import firwin, freqz


# Filter parameters
cutoff_frequency = 0.2 # Normalized cutoff frequency (0.0 to 0.5)
filter_length = 31 # Number of filter taps (odd for symmetry)

# Design the FIR filter using Hanning window method
filter_coefficients = firwin(filter_length, cutoff=cutoff_frequency, window="hann")

# Plot the impulse response of the filter
plt.figure(figsize=(10, 5))

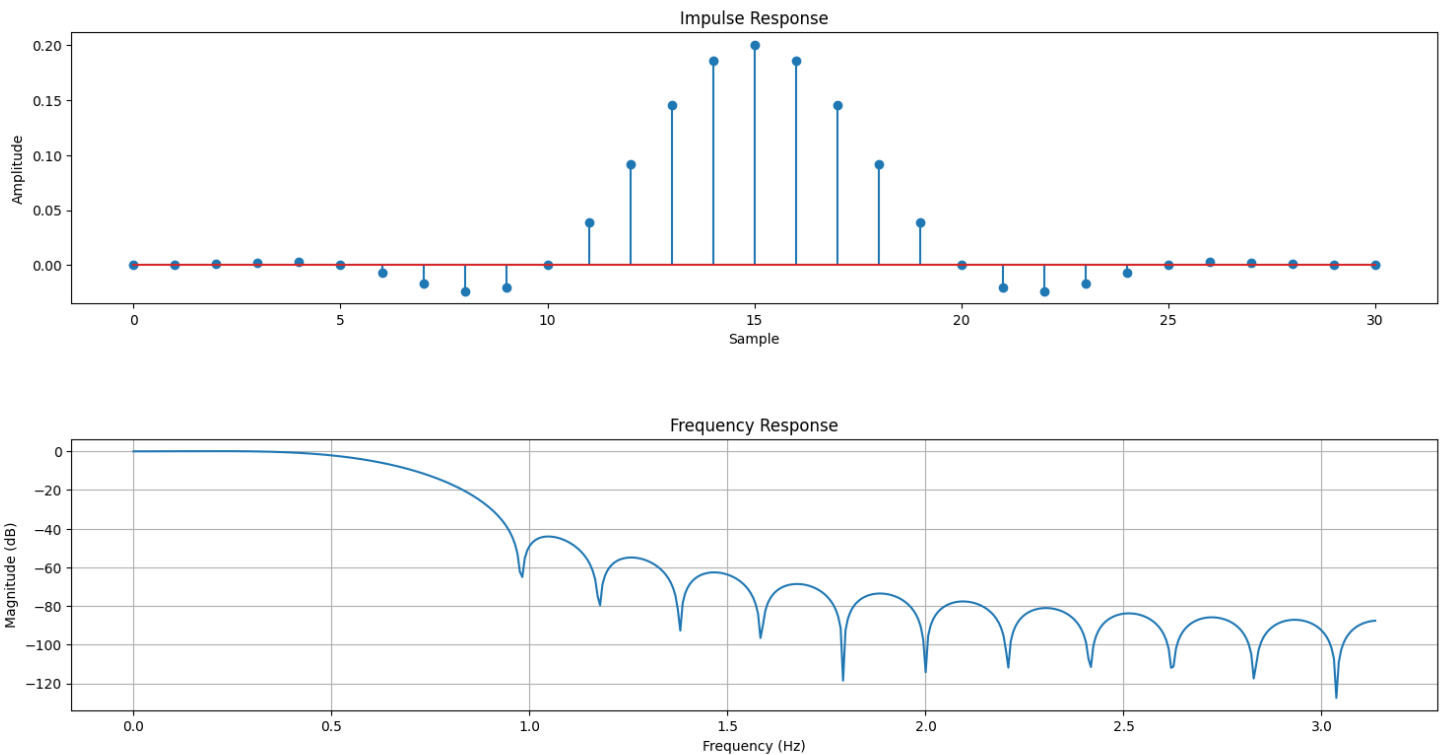
# Impulse response
plt.subplot(2, 1, 1)
plt.stem(filter_coefficients, use_line_collection=True)
plt.title("Impulse Response")
plt.xlabel("Sample")
plt.ylabel("Amplitude")

# Frequency response
plt.subplot(2, 1, 2)
frequencies, response = freqz(filter_coefficients)
plt.plot(frequencies, 20 * np.log10(np.abs(response)))
plt.title("Frequency Response")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude (dB)")
plt.grid(True)
```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Design FIR filter with windowing method.	
Experiment No: 05	Date:	Enrollment No: 92200133030


```
# Final layout and display
plt.tight_layout()
plt.show()
```

Output :-



Filter Coefficients :-

5.621724098249910630e-03,-2.248408570661841743e-02,3.372191305732654548e-02,-2.247846538775357186e-02,5.618913938795535787e-03
 1.0000000000000000e+00,-4.000320181991981805e+00,6.000960591637167774e+00,-
 4.000960637301628431e+00,1.000320227656443350e+00

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Perform Gray Level Operations Images.	
Experiment No: 07	Date:	Enrollment No: 92200133030

Aim: Perform Gray Level Operations Images.

Theory:-

- Gray-level operations involve manipulating the pixel values of an image to enhance or modify its appearance. These operations are commonly used in image processing tasks such as contrast adjustment, brightness correction, and image thresholding.

Programm:-

```
import cv2

def perform_gray_level_operation(image, operation):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    if operation == 'contrast':
        contrast_image = cv2.equalizeHist(gray_image)
        processed_image = cv2.cvtColor(contrast_image, cv2.COLOR_GRAY2BGR)
    elif operation == 'brightness':
        alpha = 1.5 # brightness factor
        processed_image = cv2.convertScaleAbs(gray_image, alpha=alpha)
        processed_image = cv2.cvtColor(processed_image, cv2.COLOR_GRAY2BGR)
    elif operation == 'thresholding':
        _, threshold_image = cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)
        processed_image = cv2.cvtColor(threshold_image, cv2.COLOR_GRAY2BGR)
    else:
        print("Invalid operation. Available operations: 'contrast', 'brightness', 'thresholding'")
        return None


    return processed_image

# Load the input image
image_path = './Images.jpg'
input_image = cv2.imread(image_path)



# Perform gray level operation
operation_type = 'contrast' # Change this to the desired operation: 'contrast', 'brightness', 'thresholding'
output_image = perform_gray_level_operation(input_image, operation_type)

if output_image is not None:
    # Display the processed image
    cv2.imshow('Processed Image', output_image)
    cv2.waitKey(0)


    # Save the processed image (optional)
    output_path = 'output_image.jpg'
    cv2.imwrite(output_path, output_image)
    print(f"Processed image saved at: {output_path}")
```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Perform Gray Level Operations Images.	
Experiment No: 07	Date:	Enrollment No: 92200133030

Output:-

Original Image	Processed Image
	

Conclusion :-

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate smoothing and sharpening operation on images using spatial filters.	
Experiment No: 08	Date:	Enrollment No: 92200133030

Aim: Simulate smoothing and sharpening operation on images using spatial filters.

Theory:-

- Smoothing and sharpening are common image enhancement techniques used to modify the spatial characteristics of an image.

Smoothing:-

- Smoothing, also known as blurring, is used to reduce noise and remove fine details in an image. It works by applying a low-pass filter that averages the pixel values within a neighborhood, thus producing a blurred effect.

Sharpening:-

- Sharpening is used to enhance the edges and fine details in an image. It works by applying a high-pass filter that amplifies the differences between neighboring pixel values, thus increasing the contrast and emphasizing edges.

Programm:-

```
import cv2 # type: ignore
import numpy as np

# Load the image
image = cv2.imread('./Images.jpg', 0)


# Define a low-pass filter kernel (Gaussian)
kernel_size = 5
sigma = 1.5
low_pass_filter = cv2.getGaussianKernel(kernel_size, sigma)
low_pass_filter = low_pass_filter * low_pass_filter.T

# Apply the low-pass filter to the image
smoothed_image = cv2.filter2D(image, -1, low_pass_filter)

# Define a high-pass filter kernel (Laplacian)
high_pass_filter = np.array([[0, -1, 0],
                             [-1, 4, -1],
                             [0, -1, 0]], dtype=np.float32)

# Apply the high-pass filter to the image
sharpened_image = cv2.filter2D(image, -1, high_pass_filter)

# Save the images
cv2.imwrite("original_image.jpg", image)
```




 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate smoothing and sharpening operation on images using spatial filters.	
Experiment No: 08	Date:	Enrollment No: 92200133030

```
cv2.imwrite("smoothed_image.jpg", smoothed_image)
cv2.imwrite("sharpened_image.jpg", sharpened_image)


# Wait for a key press and then close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()

print("Images saved as 'original_image.jpg', 'smoothed_image.jpg', and 'sharpened_image.jpg'")
```

Output :-

Original Image	Smoothed Image	Sharpened Image
		

Conclusion :-

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Apply non-linear filters on images and investigate its application in noise-removal.	
Experiment No: 09	Date:	Enrollment No: 92200133030

Aim: Apply non-linear filters on images and investigate its application in noise-removal.

Theory:-

- Non-linear filters are image processing techniques used for noise removal by considering the local neighborhood of each pixel. Unlike linear filters, non-linear filters modify pixel values based on their relationship with neighboring pixels, allowing them to effectively suppress different types of noise.

Programm:-

```
import cv2
import numpy as np


# Load the noisy image
noisy_image = cv2.imread("./Images.jpg", 0) # Load as grayscale
if noisy_image is None:
    raise FileNotFoundError(
        "The image './Images.jpg' could not be loaded. Check the file path."
    )

# Apply a median filter to remove noise
filtered_image = cv2.medianBlur(noisy_image, 5) # 5x5 neighborhood window size



# Save the filtered image
cv2.imwrite("filtered_image.jpg", filtered_image)


# Wait for a key press and then close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()

print("Filtered image saved as 'filtered_image.jpg'")
```


 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Apply non-linear filters on images and investigate its application in noise-removal.	
Experiment No: 09	Date:	Enrollment No: 92200133030

Output:-

Original Image	Non Linear Image
	

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate smoothing and sharpening operations on images using frequency domain filters.	
Experiment No: 10	Date:	Enrollment No: 92200133030

Aim: Simulate smoothing and sharpening operations on images using frequency domain filters.

Theory:-

- Smoothing and sharpening operations can also be performed in the frequency domain using filters such as the Gaussian filter and the Laplacian filter.

Smoothing in Frequency Domain:-

- To perform smoothing in the frequency domain, we apply a low-pass filter that attenuates high-frequency components. This helps to blur the image and reduce noise. One common approach is to use a Gaussian filter in the frequency domain.

Sharpening in Frequency Domain:-


- To perform sharpening in the frequency domain, we apply a high-pass filter that enhances high-frequency components. This amplifies the edges and fine details in the image. One common approach is to use a combination of a high-pass filter and the original image.

Program:-

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def apply_gaussian_filter(image, sigma):
    image = np.float32(image)
    frequency_domain = cv2.dft(image, flags=cv2.DFT_COMPLEX_OUTPUT)
    shifted_frequency_domain = np.fft.fftshift(frequency_domain)
    rows, cols = image.shape
    crow, ccol = rows // 2, cols // 2
    mask = np.zeros((rows, cols, 2), np.float32)
    for i in range(rows):
        for j in range(cols):
            distance_squared = (i - crow) ** 2 + (j - ccol) ** 2
            gaussian_value = np.exp(-distance_squared / (2 * sigma**2))
            mask[i, j] = [gaussian_value, gaussian_value]
    filtered_frequency_domain = shifted_frequency_domain * mask
    shifted_filtered_frequency_domain = np.fft.ifftshift(filtered_frequency_domain)
    filtered_image = cv2.idft(
        shifted_filtered_frequency_domain, flags=cv2.DFT_SCALE | cv2.DFT_REAL_OUTPUT
    )
    filtered_image = cv2.normalize(filtered_image, None, 0, 255, cv2.NORM_MINMAX)
    return np.uint8(filtered_image)

def apply_sharpening_filter(image, strength):
    image = np.float32(image)
    frequency_domain = cv2.dft(image, flags=cv2.DFT_COMPLEX_OUTPUT)
    shifted_frequency_domain = np.fft.fftshift(frequency_domain)
```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate smoothing and sharpening operations on images using frequency domain filters.	
Experiment No: 10	Date:	Enrollment No: 92200133030

```

rows, cols = image.shape
crow, ccol = rows // 2, cols // 2
mask = np.ones((rows, cols, 2), np.float32)
mask[crow - strength : crow + strength, ccol - strength : ccol + strength] = 0
filtered_frequency_domain = shifted_frequency_domain * mask
shifted_filtered_frequency_domain = np.fft.ifftshift(filtered_frequency_domain)
filtered_image = cv2.idft(
    shifted_filtered_frequency_domain, flags=cv2.DFT_SCALE | cv2.DFT_REAL_OUTPUT
)
filtered_image = cv2.normalize(filtered_image, None, 0, 255, cv2.NORM_MINMAX)
return np.uint8(filtered_image)

image_path = "./Images.jpg"
input_image = cv2.imread(image_path, 0)
if input_image is None:
    raise FileNotFoundError(
        f"The image '{image_path}' could not be loaded. Check the file path."
    )

sigma = 20
smoothed_image = apply_gaussian_filter(input_image, sigma)


strength = 20
sharpened_image = apply_sharpening_filter(input_image, strength)

combined_image = np.hstack((input_image, smoothed_image, sharpened_image))
plt.imshow(combined_image, cmap="gray")
plt.title("Original | Smoothed | Sharpened")
plt.axis("off")
plt.show()




smoothed_path = "smoothed_image.jpg"
sharpened_path = "sharpened_image.jpg"
cv2.imwrite(smoothed_path, smoothed_image)
cv2.imwrite(sharpened_path, sharpened_image)

print(f"Smoothed image saved at: {smoothed_path}")
print(f"Sharpened image saved at: {sharpened_path}")


```


 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate smoothing and sharpening operations on images using frequency domain filters.	
Experiment No: 10	Date:	Enrollment No: 92200133030

Output:-

Original Image	Sharpened Image	Smoothed Image
		

Conclusion :-

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate Dilation, Erosion, Opening and Closing Operation on Images.	
Experiment No: 11	Date:	Enrollment No: 92200133030

Aim: Simulate Dilation, Erosion, Opening and Closing Operation on Images.

Theory:-

- Dilation, erosion, opening, and closing are morphological operations used in image processing to manipulate the shape and size of objects in an image.

Dilation:-

- Dilation expands the boundaries of objects in an image by adding pixels to the object's boundaries. It is achieved by sliding a structuring element over the image and assigning the maximum pixel value within the neighborhood of each pixel.

Erosion:-

- Erosion shrinks the boundaries of objects in an image by removing pixels from the object's boundaries. It is achieved by sliding a structuring element over the image and assigning the minimum pixel value within the neighborhood of each pixel.

Opening:-

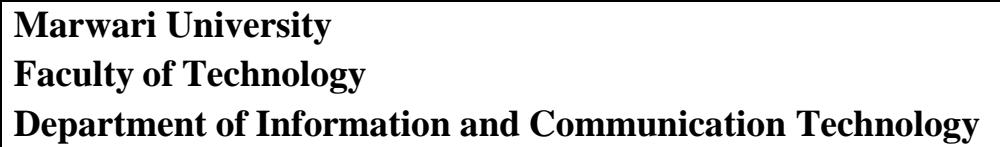
- Opening is an erosion operation followed by a dilation operation. It is used to remove noise and small objects from the image while preserving the larger objects' shapes.

Closing:-

- Closing is a dilation operation followed by an erosion operation. It is used to close small gaps and holes within objects while preserving the overall object shapes.






Programm:-


```
import cv2
import numpy as np
image_path = "./Images.jpg"
image = cv2.imread(image_path, 0)
if image is None:
    raise FileNotFoundError(f"The image '{image_path}' could not be loaded. Check the file path.")
kernel = np.ones((5, 5), np.uint8)
erosion = cv2.erode(image, kernel, iterations=1)
dilation = cv2.dilate(image, kernel, iterations=1)
opening = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
closing = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
cv2.imwrite("erosion.jpg", erosion)
cv2.imwrite("dilation.jpg", dilation)
cv2.imwrite("opening.jpg", opening)
cv2.imwrite("closing.jpg", closing)
print("Images saved: 'erosion.jpg', 'dilation.jpg', 'opening.jpg', 'closing.jpg'")
```



Aim: Simulate Dilation, Erosion, Opening and Closing Operation on Images.

Enrollment No: 92200133030

Original Image	Erosion Image	Dilation Image
		
Opening Image	Closing Image	
		

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate Hit or Miss Transformation on images.	
Experiment No: 12	Date:	Enrollment No: 92200133030

Aim: Simulate Hit or Miss Transformation on images.

Theory:-

- The Hit or Miss transformation is a morphological operation used to detect specific patterns or shapes in an image. It is commonly used for shape recognition or pattern matching.
- The operation requires two structuring elements: one for foreground (hits) and another for background (misses). The foreground structuring element represents the desired pattern to be matched, while the background structuring element represents the complement of the pattern.
- During the transformation, the foreground structuring element is matched against the foreground pixels in the image, and the background structuring element is matched against the background pixels. Only those pixels that match both structuring elements are considered hits.



Programm:-


```
import cv2
import numpy as np

image = cv2.imread("./Images.jpg", 0)
pattern = np.array([[ -1, 1, -1], [1, 1, 1], [ -1, 1, -1]], dtype=np.int8)
result = cv2.morphologyEx(image, cv2.MORPH_HITMISS, pattern)
cv2.imwrite("hit_miss_transformation_result.jpg", result)


print("The Result is Saved")
```

Output:-

Original Image	Hit Miss Transformation
	

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate Hit or Miss Transformation on images.	
Experiment No: 12	Date:	Enrollment No: 92200133030

Conclusion :-

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate Boundary Extraction on images.	
Experiment No: 13	Date:	Enrollment No: 92200133030

Aim: Simulate Boundary Extraction on images.

Theory:-



- Boundary extraction is a morphological operation used to extract the boundary or contour of objects in an image. It highlights the boundaries between object regions and the background, providing important information about the shape and structure of objects
- The boundary extraction operation can be achieved by subtracting the input image from its morphological dilation. This highlights the pixels that are on the object boundaries, as the dilation operation expands the object while maintaining its shape.


Programm:-

```
import cv2
import numpy as np

image = cv2.imread("./Images.jpg", cv2.IMREAD_GRAYSCALE)
image_blurred = cv2.GaussianBlur(image, (5, 5), 0)
edges = cv2.Canny(image_blurred, threshold1=30, threshold2=100)
boundary_image = np.zeros_like(image)
boundary_image[edges > 0] = 255
cv2.imwrite("boundary_image_result.jpg", boundary_image)
print("The boundary image is saved as 'boundary_image_result.jpg'")
```

Output:-

Original Image		Boundary Extraction	
			

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	Aim: Simulate Boundary Extraction on images.	
Experiment No: 13	Date:	Enrollment No: 92200133030

Conclusion :-