Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with 'A' Grade (CGPA: 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2024-25

Name: Shashwat Shah SAP ID: 60004220126

Course: **Digital Signal Processing and Applications Laboratory** Course Code: DJ19CEL701

Year: BTech Sem: VII Batch: C22

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11
Course Outcome	1	2	2	3	4	4	4	4	5	All	
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	5	4	5	5	5	5	5	5	5	5	5
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	5	4	4	4	5	5	4	5	5	5	5
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	4	5	3	5	3	3	5	5	5	5
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	3	5	3	3	3	3	3	3	3	3	4
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	-	-	-	-	-	-	-	-	-	-	-
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	3	5	3	5	3	3	3	3	3	3	4
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	-	-	-	-	-	-	-	-	-		-
Total	21	22	21	20	20	19	18	20	21	21	23
Signature of the faculty member											

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Total Term-work (25) = 21
Sign of the Student:

Ruhinaik 18/11/24.

Signature of the Faculty member:
Name of the Faculty member: Ruhina Karani

Signature of Head of the Department
Date:

Experiment 1

Shashwat Shah
60004220126
BE Comps C22

Aim : Construction of standard DT signals (sine, cosine, step, unit, unit impulse, ramp), and single operations.

Theory : Discrete time (DT) signals are representations of continuous signals in digital form, sampled at specific intervals.

→ Sine and cosine signals.
These are periodic waveforms used extensively in signal processing

→ Unit step signals.
The unit step signal $u(n)$ is a discrete signal that starts from zero and stays to a certain point

→ Unit impulse signal
Also known as delta function $\delta(n)$ it is zero everywhere, except at $n=0$ where it is 1

→ Ramp signal.
It increases linearity with time, useful in control systems and for simulating acceleration over time.

→ Signal operations
Addition and subtraction

→ Multiplication.

→ Time scaling - It stretches or compresses the signal in the time domain.

Conclusion - By constructing and visualizing these DT signals, we gain and understand of how each signal behaves over discrete intervals and how they can be combined or altered through basic signal operations.

NAME: Shashwat Shah SAP ID: 60004220126
DIV/BATCH:C22

DIGITAL SIGNAL PROCESSING
(DSP) EXPERIMENT 01

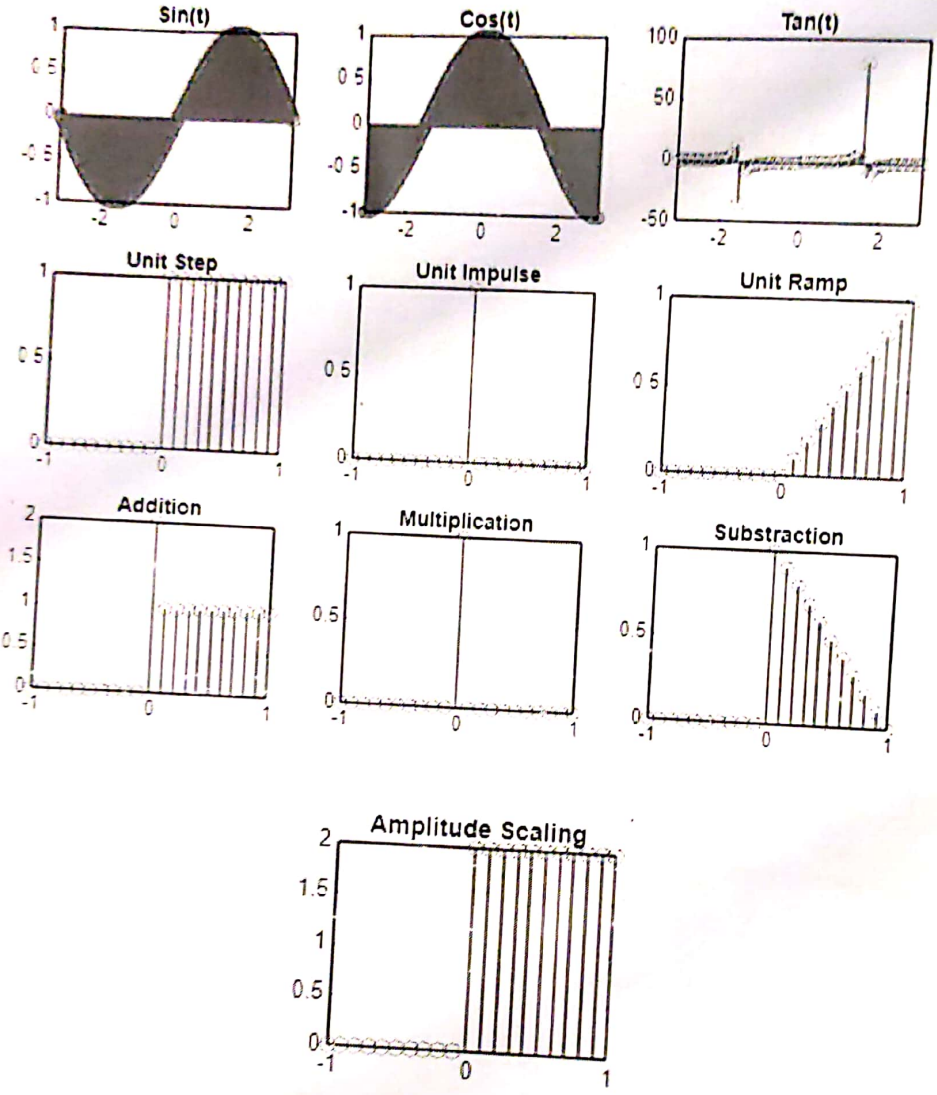
CODE:

```
t=-pi:0.01:pi; y =
sin(t);
subplot(3,3,1),stem(t,y);
title('Sin(t)');

t=-pi:0.01:pi; y =
cos(t);
subplot(3,3,2),stem(t,y);
title('Cos(t)');

t=-pi:0.1:pi; y =
tan(t);
subplot(3,3,3),stem(t,y);
title('Tan(t)');

t = -1:0.1:1;
unitstep = t>=0;
subplot(3,3,4),stem(t,unitstep); title('Unit
Step');

impulse = t==0;
subplot(3,3,5),stem(t,impulse); title('Unit
Impulse');

ramp = t.*unitstep;
subplot(3,3,6),stem(t,ramp); title('Unit
Ramp');

add = impulse+unitstep;
subplot(3,3,7),stem(t,add); title('Addition');

mul = impulse.*unitstep;
subplot(3,3,8),stem(t,mul);
title('Multiplication');

sub = unitstep-ramp;
subplot(3,3,9),stem(t,sub); title('Substraction');
```

Amplitude Scaling

```
mul = 2.*unitstep; subplot(3,3,8),stem(t,mul);
title('Multiplication');
```

OUTPUT:



)DE:

)efine the original signal x(n) n = 0:8; % n
:es from 0 to 8
1:9; % x(n) is defined from 1 to 9

ake input for the expansion factor b
nput('Enter the expansion factor b: ');

ompute the expanded signal x(n/b) expanded_n = n / b;

terpolate to find the expanded x values
nded_x = interp1(n, x, expanded_n, 'linear', 'extrap');

t x(n/b) to 0 when n/b is not an integer expanded_x(mod(expanded_n, 1) ~= 0) = 0;

Experiment 2

Shashiwat Shah
60004220126
BE Comps B

Aim: Discrete Auto / Cross correlation between two signals.

Theory: Correlation is a statistical operation that measures the degree to which two signals are similar

- Auto correlation - This is a measure of how a signal correlates with itself over time.

The formula is given by

$$R_{xx}(m) = \sum_{n=-\infty}^{\infty} x(n) \cdot x(n-m)$$

where, $x(n)$ is the discrete signal and m is the time lag.

Cross correlation.

It measures the similarity between two different signals as a function of the time-lag applied to one of them.

The formula is given by

$$R_{xy}(m) = \sum_{n=-\infty}^{\infty} x(n) \cdot y(n-m)$$

where $x(n)$ and $y(n)$ are the two signals being compared.

Conclusion - In this experiment, we implemented disc auto and cross correlation between two signals to study their similarities and alignment properties. We observed, through auto correlation we can identify periodic patterns and through cross correlation we can get insights into the degree of alignment and possible phase shift of the signal.

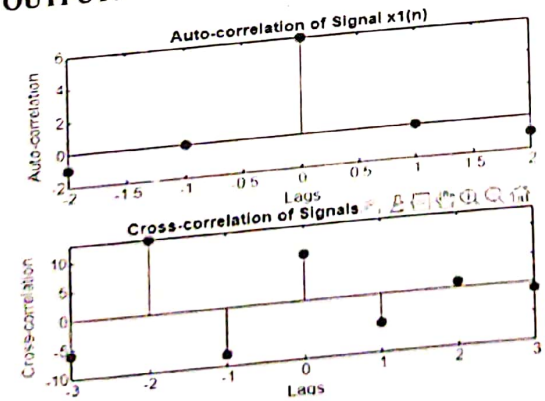
NAME: Shashwat Shah SAP ID: 60004220126
DIV/BATCH:C22
DATE: 05/08/24

DIGITAL SIGNAL PROCESSING (DSP) EXPERIMENT 02

CODE:

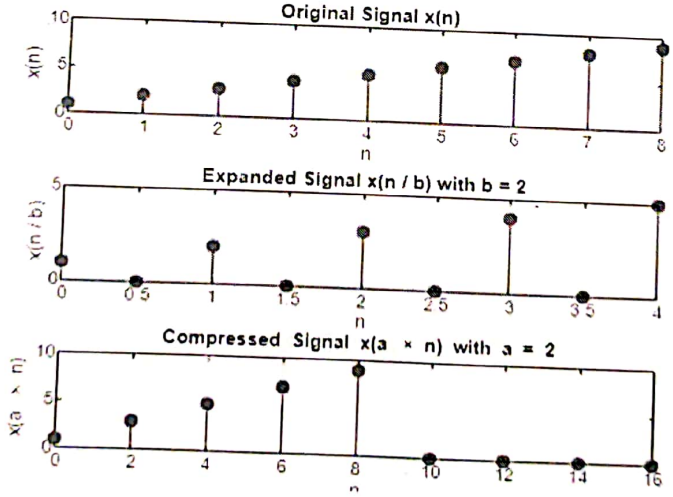
```
% Define the original signal x(n) for auto-correlation x1 = [-1 2 1]; % signal
x1(n)

% Compute the auto-correlation of the signal x1 [auto_corr_x1, lags_x1] =
xcorr(x1);

% Define the original signals x(n) and y(n) for cross-correlation x2 = [-3 2 -1 1]; % signal
x2(n)
y = [-1 0 -3 2];  % signal y(n)

% Compute the cross-correlation of the signals x2 and y [cross_corr_xy, lags_xy] =
xcorr(x2, y);

% Create a figure to show both auto-correlation and cross-correlation figure;

% Plot the auto-correlation of x1
subplot(2,1,1);
stem(lags_x1, auto_corr_x1, 'filled'); title('Auto-
correlation of Signal x1(n)'); xlabel('Lags');
ylabel('Auto-correlation'); grid on;

% Plot the cross-correlation of x2 and y subplot(2,1,2);
stem(lags_xy, cross_corr_xy, 'filled');
title('Cross-correlation of Signals x2(n) and y(n)'); xlabel('Lags');
ylabel('Cross-correlation'); grid on;
```

OUTPUT:



```
% Take input for the compression factor a
a = input('Enter the compression factor a: ');

% Compute the compressed signal x(a*n) compressed_n =
n * a;
compressed_x = interp1(n, x, compressed_n, 'linear', 0);

% If any value in compressed_n exceeds the maximum n, set corresponding x to 0 compressed_x(compressed_n > max(n))
= 0;

% Create a figure to show all three signals figure;

% Plot the original signal subplot(3,1,1);
stem(n, x, 'filled'); title('Original Signal x(n)');
xlabel('n');
ylabel('x(n)'); grid on;

% Plot the expanded signal subplot(3,1,2);
stem(expanded_n, expanded_x, 'filled');
title(['Expanded Signal x(n / b) with b = ', num2str(b)]); xlabel('n');
ylabel('x(n / b)'); grid on;

% Plot the compressed signal subplot(3,1,3);
stem(compressed_n, compressed_x, 'filled');
title(['Compressed Signal x(a \times n) with a = ', num2str(a)]); xlabel('n');
ylabel('x(a \times n)'); grid on;
```

OUTPUT:



Experiment 3

Shashwat Sha
60004220126
BE Comps (22

Aim : Discrete convolution (linear & circular)

Theory : Convolution is a mathematical operation tha describes the two-way signal combine.

→ Linear convolution.

This combins two signals to form a new signa considering the total length of both signals.

The discrete linear convolution of two signal $x(n)$ $h(n)$ is given by - $y(n) = \sum_{k} x(k) \cdot h(n-k)$

→ Circular convolution

Circular convolution treats signals as if they are period wrapping around when they exceed a certain length The formula is similar to linear convolution bu includes modulo operation to handle periodicity

$$y(n) = x(n) \textcircled{N} h(n)$$

Conclusion : By applying these methods, we observed how linear convolution extends the length output signal, while the circular convolution wra the signal due to periodicity, producing a result same length as i/p sequences.

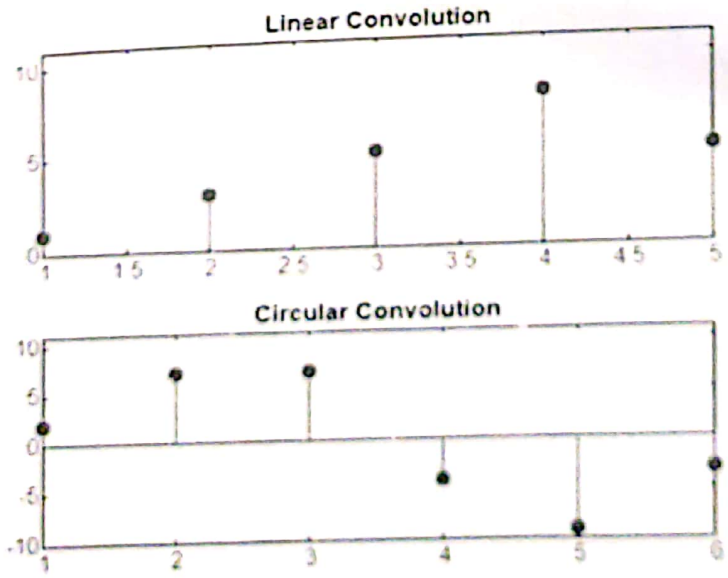
NAME: Shashwat Shah SAP ID: 60004220126
DIV/BATCH:C22
DATE: 26/08/24

DIGITAL SIGNAL PROCESSING (DSP) EXPERIMENT 03

CODE:

```
x = [1 2 3 5];
y = [1 1];
linear_c = conv(x,y);

xpad = [x zeros(1,6-length(x))];
ypad = [y zeros(1,6-length(y))];

subplot(2,1,1) stem(linear_c,'filled')
ylim([0 11])
title('Linear Convolution')


x1 = [1 2 0 -3];
x2 = [2 3 1];
x1pad = [x1 zeros(1,6-length(x1))]; x2pad = [x2
zeros(1,6-length(x2))];
circular_c = ifft(fft(x1pad).*fft(x2pad));

subplot(2,1,2) stem(circular_c,'filled')
ylim([-10 11]) title('Circular
Convolution')
```

OUTPUT:



Experiment 4

Shashwat Shah
60004220126
BE comps C22

Aim: Discrete Fourier Transform on Discrete time signals.

Theory: The discrete fourier transform is a fundamental tool in signal processing that transforms a time domain signal into its frequency domain representation. Time and frequency Domains.

In the time domain, signals are viewed as functions of time. However many practical applications require analyzing the frequency content of the signals such as communication system, audio processing and vibration analysis.

The DFT formula.

The DFT of a discrete time signal $x(n)$ is given by.

$$X(k) = \sum_{n=0}^{N-1} x(n) \quad e^{-j\frac{2\pi kn}{N}}$$

where

$X(k)$ is the DFT, N is the number of samples and k represent the frequency index.

Conclusion : This experiment demonstrates the utilities the DFT in understanding the spectral characte of signal including the identification of dominant frequ and periodicity. Overall, the application of the DFT in signal analysis is crucial for task such as filtering, signal reconstruction and spectral analysis in digital signal processing.

NAME: Shashwat Shah SAP ID: 60004220126
DIV/BATCH:C22
Date : 23/09/24

DIGITAL SIGNAL PROCESSING (DSP)
EXPERIMENT 04

AIM: To apply Discrete Fourier Transform on DT signal

CODE:

```
% Function to calculate DFT manually function X =

DFT_manual(x)

    N = length(x);  % Number of points
    X = zeros(1, N);  % Initialize the result array for k = 0:N-1
        for n = 0:N-1
            angle = 2 * pi * k * n / N;
            X(k+1) = X(k+1) + x(n+1) * exp(-1j * angle);
     end   end

end

N4 = 4; % N=4 N8
= 8; % N=8

% Input signal for N=4
disp('Enter 4 values for the signal (N=4):'); x4 = input('Signal:
');

if length(x4) ~= N4
    disp('Please enter exactly 4 values.'); else
    % Calculate DFT manually for N=4 dft_manual_4 =
    DFT_manual(x4); disp('Manual DFT result for
    N=4:'); disp(dft_manual_4);

    % Calculate using FFT (direct function) dft_fft_4 =
    fft(x4);
    disp('FFT result for N=4:');
    disp(dft_fft_4);

    % Compare results
    if isequal(round(dft_manual_4, 10), round(dft_fft_4, 10)) disp('The manual DFT and
        FFT results match for N=4.');
    else
        disp('The manual DFT and FFT results do not match for N=4.');
    end
end

% Input signal for N=8
disp('Enter 8 values for the signal (N=8):'); x8 = input('Signal:
');
```

```
    if length(x8) ~= N8
        disp('Please enter exactly 8 values'); else
        % Calculate DFT manually for N=8 dft_manual_8 =
        DFT_manual(x8); disp('Manual DFT result for
        N=8:'); disp(dft_manual_8);

        % Calculate using FFT (direct function) dft_fft_8 =
        fft(x8);
        disp('FFT result for N=8:');
        disp(dft_fft_8);

        % Compare results
        if isequal(round(dft_manual_8, 10), round(dft_fft_8, 10)) disp('The manual DFT and
            FFT results match for N=8.');
        else
            disp('The manual DFT and FFT results do not match for N=8.');
        end
end
```

OUTPUT:

```
Signal:
[1 2 3 4]
Manual DFT result for N=4:
  10.0000 + 0.0000i  -2.0000 + 2.0000i  -2.0000 - 0.0000i  -2.0000 - 2.0000i

FFT result for N=4:
  10.0000 + 0.0000i  -2.0000 + 2.0000i  -2.0000 + 0.0000i  -2.0000 - 2.0000i

The manual DFT and FFT results match for N=4.
Enter 8 values for the signal (N=8):
Signal:
[1 2 3 4 5 6 7 8]
Manual DFT result for N=8:
  36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i  -4.0000 - 0.0000i  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i

FFT result for N=8:
  36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i  -4.0000 + 0.0000i  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i

The manual DFT and FFT results match for N=8.
```

Experiment- 5

Shashwat Shah
60004220126
BE Comps C22

Aim: Image Negation, grey level slicing and thresholding.

Theory: These image processing techniques help enhance and extract features from image.

→ Image Negation

In image negative, each pixel value is subtracted from the maximum pixel value, resulting in an inverted image.

→ Gray level slicing - Gray level scaling is a technique used to highlight a specific range of intensities in an image.

→ Thresholding

Thresholding is a technique which converts a grayscale image into a binary image by turning all pixels above a certain intensity value white (1) and all other black (0).

Conclusion: In this experiment, we applied image negative grey scale level slicing and thresholding techniques. Image negative inverted pixel values, grey level slicing highlighted specific intensity ranges and thresholding converted image to binary for object detection. These techniques are essential for image manipulation and analysis in field like computer vision and image enhancement.

NAME: Shashwat Shah SAP ID: 60004220126
DIV/BATCH:C22DATE: 14/10/24
23/09/24

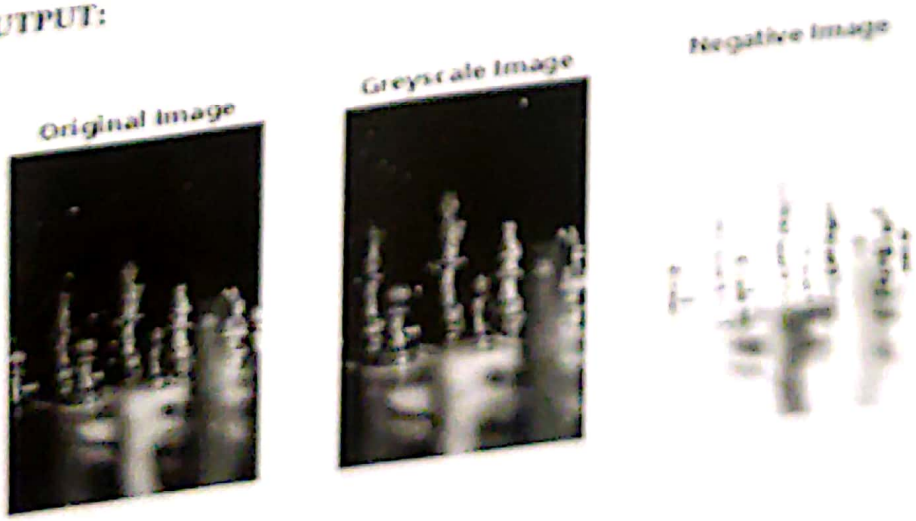
DIGITAL SIGNAL PROCESSING (DSP)
EXPERIMENT 06

AIM: To Image Negative, Grey Level Slicing & Thresholding

CODE:
IMAGE NEGATIVE

```
image_path = 'image1.jpg'; image =
imread(image_path);

% Convert to grayscale if
size(image, 3) == 3
    grey_image = rgb2gray(image);
end
% Number of intensity levels L = 256;

% Convert to negative
negative_image = L - 1 - grey_image; output_path =
'negative_image.jpg'; imwrite(negative_image,
output_path);

figure; subplot(1, 3, 1);
imshow(image);
title('Original Image');

subplot(1, 3, 2); imshow(grey_image);
title('Greyscale Image');

subplot(1, 3, 3); imshow(negative_image);
title('Negative Image');
```

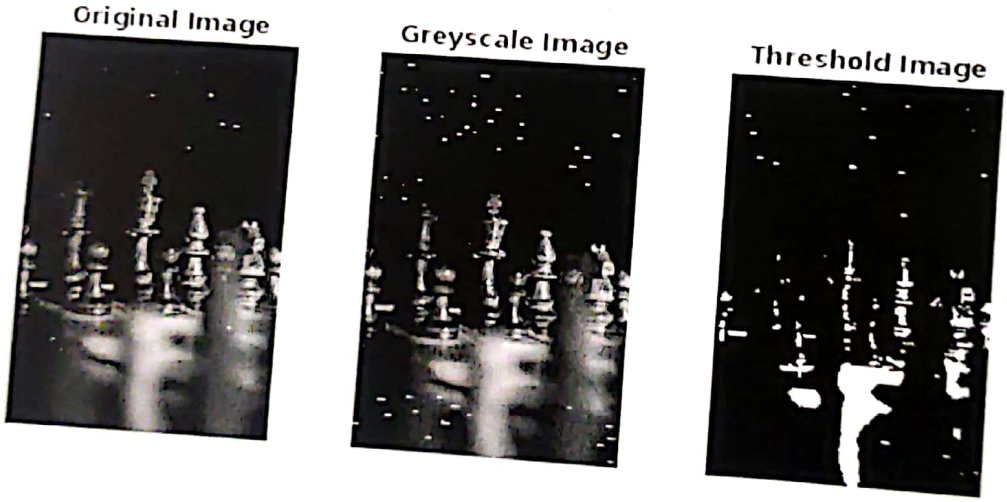
OUTPUT:



THRESHOLDING

```
image_path = 'image1.jpg'; image =
imread(image_path);

% Convert to grayscale if
size(image, 3) == 3
    grey_image = rgb2gray(image);
end

% Number of intensity levels L = 256;
thres = 127;

% Apply thresholding
threshold_image = grey_image >= thres; threshold_image =
uint8(threshold_image) * (L - 1);

output_path2 = 'threshold_image.jpg';
imwrite(threshold_image, output_path2);

figure; subplot(1, 3, 1);
imshow(image);
title('Original Image');

subplot(1, 3, 2); imshow(grey_image);
title('Greyscale Image');

subplot(1, 3, 3); imshow(threshold_image);
title('Threshold Image');
```

OUTPUT:



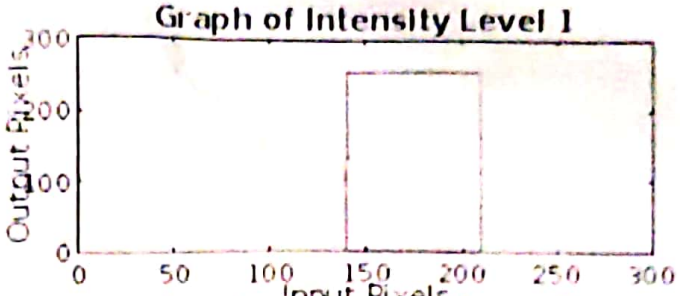
GREY LEVEL SLICING

```
function plot_transformations_in_batches(input_img, x_values, transformations, batch_size)
    num_transforms = size(transformations, 1);
    num_batches = ceil(num_transforms / batch_size);  % Calculate how many batches are needed

    % Convert input image to double for consistent arithmetic input_img =
    double(input_img);

    for batch = 1:num_batches
        figure('Position', [100, 100, 1400, 800]);  % Adjust size for space start_idx = (batch - 1) *
        batch_size + 1;
        end_idx = min(batch * batch_size, num_transforms);

        for i = start_idx:end_idx
            transform_func = transformations{i, 1}; transform_title =
            transformations{i, 2};

            % Apply the transformation to the image transformed_img =
            arrayfun(transform_func, input_img); y_values =
            arrayfun(transform_func, x_values);

            % Plot the transformed image subplot(batch_size, 2, 2*(i -
            start_idx) + 1); imshow(transformed_img, []);
            title(transform_title);

            % Plot the transformation graph subplot(batch_size, 2, 2*(i -
            start_idx) + 2); plot(x_values, y_values);
            xlabel('Input Pixels'); ylabel('Output
            Pixels'); grid on;
            title(['Graph of ' transform_title]);

        end
    end

end

% Load and process image img =
imread('image1.jpg'); gray_image =
rgb2gray(img);

% Define transformations transformations = {
    @(r) (r > 140 & r < 210) * 255, 'Intensity Level 1';
    @(r) ~(r > 140 & r < 210) * 255, 'Intensity Level 2';
    @(r) (r > 140 & r < 210) * 255 + (r <= 140 | r >= 210) .* double(r), 'Intensity Level 3';
    @(r) (r > 140 & r < 210) .* double(r) + (r <= 140 | r >= 210) * 255, 'Intensity Level 4';
    @(r) (r < 150) * 0 + (r >= 150) * 255, 'Threshold at 150';
    @(r) (r < 60) * 0 + (r >= 60) * 255, 'Threshold at 60';
};

x_values = linspace(0, 255, 500);  % For plotting transformation curves

% Plot transformations in batches of 3 to avoid overloading batch_size = 3;
```

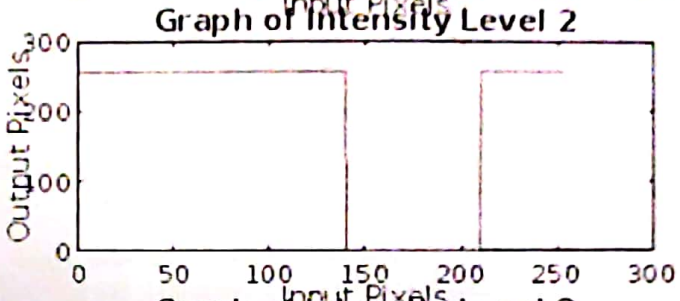
plot_transformations_in_batches(gray_image, x_values, transformations, batch_size);

OUTPUT:

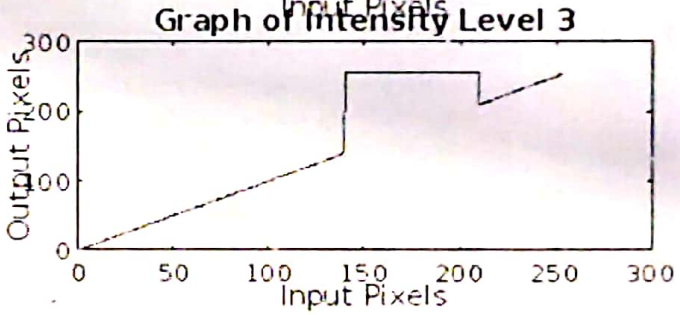
Intensity Level 1



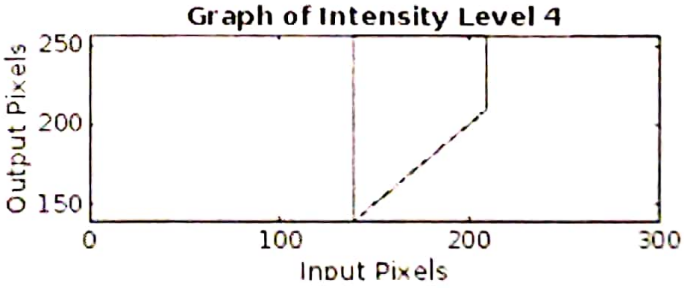
Intensity Level 2



Intensity Level 3



Intensity Level 4



Experiment: 6

Shashwat Shah
60004220126
BE comps C22

Aim: Histogram stretching on an Image.

Theory: Histogram stretching also known as contrast stretching is a technique used in image processing to improve the contrast of an image by stretching the range of intensity value.

→ Image Histogram

It represents the distribution of pixel intensity values. For grey scale images, it shows how often each pixel intensity (0 - 255) occur.

→ Histogram stretching.

The main goal of histogram stretching is to enhance the contrast by expanding the narrow intensity range to cover the full range to cover the possible values. (0 - 255). The process can be defined as,

$$S = \frac{(r - r_{min})}{(r_{max} - r_{min})} \cdot (S_{max} - S_{min}) - S_{min}$$

where r is the input pixel intensity, r_{min} and r_{max} are the minimum and maximum intensities of the original image and S_{min} and S_{max} are the desired new minimum and maximum intensity value.

Conclusion - Overal, this experiment highlighted how histogram manipulation can improve image quality making features that were initially obscure more distinguishable and thus making it a valuable tool for image enhancement applications

NAME: Shashwat Shah SAP ID: 60004220126 DIV/BATCH:

DIGITAL SIGNAL PROCESSING (DSP)
EXPERIMENT 06

AIM: To perform Histogram Stretching on an Image.

CODE:

```
# Function to plot histogram from scratch
def plot_histogram2(image, title):
    # Calculate the histogram
    hist = np.zeros(256, dtype=int)
    for pixel in image.ravel():
        hist[pixel] += 1

    # Plot the histogram
    plt.plot(hist, color='gray')
    plt.title(title)
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.xlim([0, 256])
    plt.grid(True)

# Plot histograms for the original and stretched images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plot_histogram2(image, 'Original Image Histogram')
plt.subplot(1, 2, 2)
plot_histogram2(stretched_image, 'Stretched Image Histogram')
plt.tight_layout()
plt.show()
```

OUTPUT:

Experiment 7

Shashwat Sha
60004220126
BE comps C22

Aim : Histogram equalization for an image.

Theory : Histogram equalization is an image proces techniques used to enhance the contrast of a image by redistributing its pixels intensity valu

→ Principle

The basic idea behind histogram equalization is the mapping of the pixel intensities of an image so that they follow a uniform distribution.

→ Equalization process.

The steps for performing histogram equalization can be summarized as:

1) Calculate the histogram of the given image.
2) Compute the cumulative distribution function (CDF based on the histogram.
3) Normalize CDF, to span the full range of pixel intensities (0 to 255)
4) Use of normalized CDF, to map the original pixel intensities to new values creating the equalised image.

→ Benefits and Applications - It enhances the visibility of details in an image by making the intensity distribu more uniform, which can significantly improve image clarity in situations where there are shows

Conclusion - In this experiment, By redistributing the intensity values, we transformed a low contrast image into one with improved detail and visibility

NAME: Shashwat Shah SAP ID: 60004220126 DIV/BATCH:C22

DIGITAL SIGNAL PROCESSING (DSP)
EXPERIMENT 07

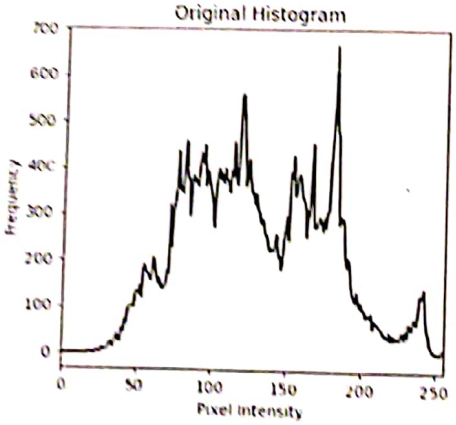
AIM: To implement Histogram Equalization for an Image.

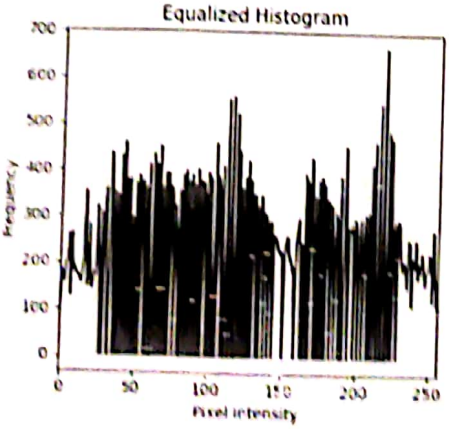
CODE:

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

def histogram_equalization(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Calculate the histogram of the original image original_histogram, bins =
    np.histogram(image.flatten(), bins=256,
range=[0, 256])

    # histogram equalization flat_image =
    image.flatten()
    histogram = original_histogram / original_histogram.sum() cdf = histogram.cumsum()
    cdf_normalized = np.round(cdf * 255).astype(np.uint8) equalized_image =
    cdf_normalized[flat_image].reshape(image.shape)

    # Calculate the histogram of the equalized image equalized_histogram, bins_eq
    =
np.histogram(equalized_image.flatten(), bins=256, range=[0, 256])

    # Display the original image, its histogram, the equalized image, and its histogram
    plt.figure(figsize=(8, 8))

    # Original Image and Histogram plt.subplot(2, 2,
    1) plt.title('Original Image') plt.imshow(image,
    cmap='gray') plt.axis('off')

    plt.subplot(2, 2, 2) plt.title('Original Histogram')
    plt.plot(original_histogram, color='black') plt.xlim([0, 256])
    plt.xlabel('Pixel Intensity')
```

```
    plt.ylabel('Frequency')

    # Equalized Image and Histogram
    plt.subplot(2, 2, 3)
    plt.title('Equalized Image')
    plt.imshow(equalized_image, cmap='gray')
    plt.axis('off')

    plt.subplot(2, 2, 4)
    plt.title('Equalized Histogram')
    plt.plot(equalized_histogram, color='black')
    plt.xlim([0, 256])
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()


image_path = 'image1.jpg'
histogram_equalization(image_path)
```

OUTPUT:





Experiment 8

Shashwat Shah
6000 4220126
BE comps C22

Aim: Justify: Average filter is used to remove salt and noise

Theory: Salt and Pepper noise also known as impulse noise is characterized by randomly occurring white and black pixels. An averaging filter, also known as the mean filter, is commonly employed in image processing to reduce noise. It works by replacing pixel values. This has a smoothening effect on the image, reducing variations between adjacent pixels and thus softening noise.

Justification

The averaging filter can smooth out some of the effects of salt and pepper noise by averaging the black and white noise pixels with surrounding pixels. However, it has significant drawbacks.

1) Blurring - The averaging filter tends to blur edges and fine details in the image, as it replaces each pixel with the average of its neighbours, even if there is no noise

2) Ineffectiveness with salt and pepper - Since salt and pepper involves extreme values (0 to 255) averaging with surrounding non-noise pixels often result in a loss of information. Instead of removing the noise entirely, it creates new pixel values that don't necessarily correspond to the original noise.

Conclusion: The use of an averaging filter to remove sa
& pepper noise is generally not ideal. While it
can reduce the visibility of the noise, it also causes
a significant loss of image detail due to the blurring
effect.

NAME: Shashwat Shah SAP ID: 60004220126
DIV/BATCH:

DIGITAL SIGNAL PROCESSING (DSP)
EXPERIMENT 08

AIM: To implement image smoothing / image sharping using low pass and high pass filter.

CODE:

IMAGE SMOOTHING USING LOW PASS FILTER

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

# Function to add Gaussian noise
def add_gaussian_noise(image, mean=0, sigma=25):
    noisy_image = image + np.random.normal(mean, sigma,
image.shape).astype(np.uint8)
    return np.clip(noisy_image, 0, 255)

# Function to apply low-pass averaging filter
def apply_averaging_filter(image, kernel_size=5):
    return cv2.blur(image, (kernel_size, kernel_size))

# Load an image
image = cv2.imread('chess.jpg', cv2.IMREAD_GRAYSCALE)

# Check if image was loaded
if image is None:
    print("Error: Image not found.")
else:
    # Add Gaussian noise to the image
    noisy_image = add_gaussian_noise(image)

    # Apply low-pass averaging filter to remove noise
    filtered_image = apply_averaging_filter(noisy_image)

    # Plot the images
    plt.figure(figsize=(12, 8))

    plt.subplot(1, 3, 1)
    plt.title('Original Image')
    plt.imshow(image, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 3, 2)
```

```
plt.title('Noisy Image')
plt.imshow(filtered_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('Filtered Image')
plt.imshow(noisy_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

OUTPUT:

IMAGE SMOOTHING USING HIGH PASS FILTER

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

image_path = 'image3.jpg'
original_image = cv2.imread(image_path)
original_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)

# Apply Gaussian Blur to create a low-pass filtered image
kernel_size = (5, 5)  # Size of the Gaussian kernel
low_pass = cv2.GaussianBlur(original_image, kernel_size, sigmaX=0)

# Create the high-pass filtered image
high_pass = cv2.subtract(original_image, low_pass)

# Sharpen the original image by adding the high-pass image
sharpened_image = cv2.add(original_image, high_pass)
plt.figure(figsize=(15, 5))

# Original image
plt.subplot(1, 3, 1)
plt.imshow(original_image)
plt.title('Original Image')
plt.axis('off')
```

Experiment 9

Shashwat Shah
60004220126
BE Comps C22

stify - Prewitt & Sobel filters work good on noisy images.

ory: Prewitt and sobel filters are first order derivaty filters used in edge detection for image processing. These filters highlight regions of high spatial frequency. which often correspond to edge in an image. Both filters are designed to approximate the gradient of the image intensity ction, helping to identify areas where there is a sharp change in intensity i.e edges.

ustification -

hen applying the Prewitt and Sobel filters to nosy Images, the following observations can be made.

Sensitivity to Noise

Both prewitt and sobel filters are sensitive to noise, especially high frequency noise such as Gaussian noise This is because noise introduces abrupt intensity changes across the image, which the filters may incorrectly identify as edge.

Performance in noisy conditions

In noisy images, the output of prewitt and sobel filters become cluttered with fake edge cause by noise This leads to poor performance in detecting the actual structure of the image.

Conclusion : Using the Prewitt and Sobel filters on noi
does not yield good result. These filters are desi
to highlight edges based on intensity gradients, bu
noise introduces gradients that are mistaken for e

NAME: Shashwat Shah SAP ID: 60004220126 DIV/BATCH:C22 DATE: 14/10/24

DIGITAL SIGNAL PROCESSING (DSP)
EXPERIMENT 09

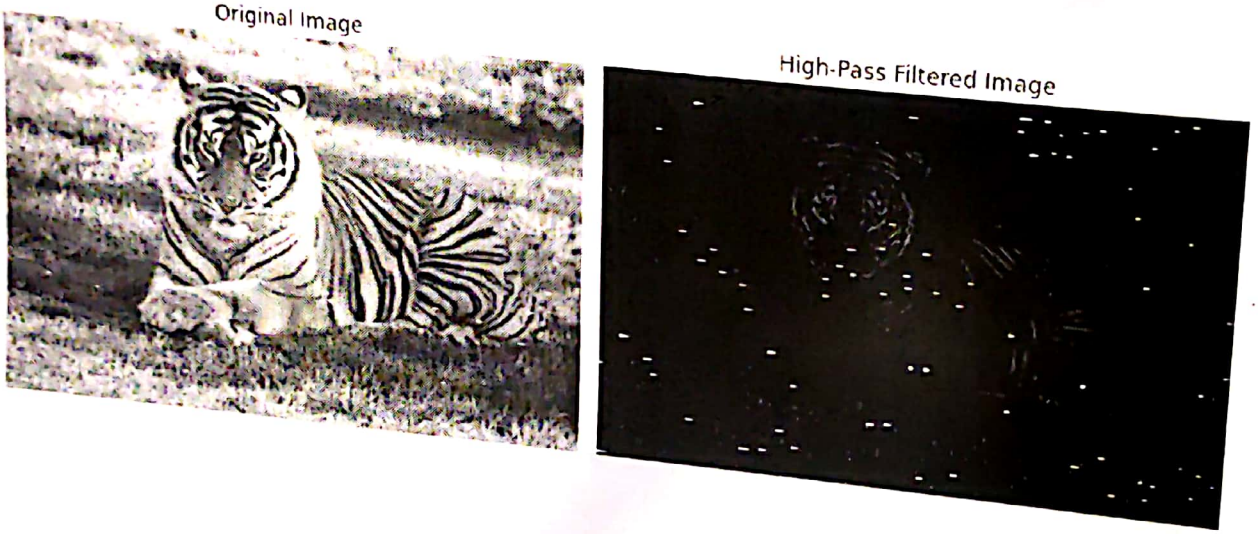
AIM: To implement Edge detection using Sobel & Perwitt masks.

CODE:
SOBEL

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

# Load the uploaded image
image_path = 'image2.jpg'  # Replace with the actual path if needed image = cv2.imread(image_path)

# Step 1: Display the original image plt.figure(figsize=(10, 10))
plt.subplot(3, 3, 1), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)),
plt.title('Original Image') plt.axis('off')

# Step 2: Convert the image to grayscale and display it gray_image = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY) plt.subplot(3, 3, 2), plt.imshow(gray_image, cmap='gray'),
plt.title('Grayscale Image')
plt.axis('off')

# Step 3: Apply Sobel X (fx) on grayscale image sobel_x_kernel = np.array([[-1, 0,
1],
                                    [-2, 0, 2],
                                    [-1, 0, 1]])

r1 = cv2.filter2D(gray_image, -1, sobel_x_kernel)  # Applying Sobel X plt.subplot(3, 3, 3),
plt.imshow(r1, cmap='gray'), plt.title('Sobel X (r1)')
plt.axis('off')

        # Step 4: Apply Sobel Y (fy) on  grayscale image
        sobel_y_kernel = np.array([[-1,  -2, -1],
                                   [0,  0, 0],
                                   [1,  2, 1]])

r2 = cv2.filter2D(gray_image, -1, sobel_y_kernel)  # Applying Sobel Y
```

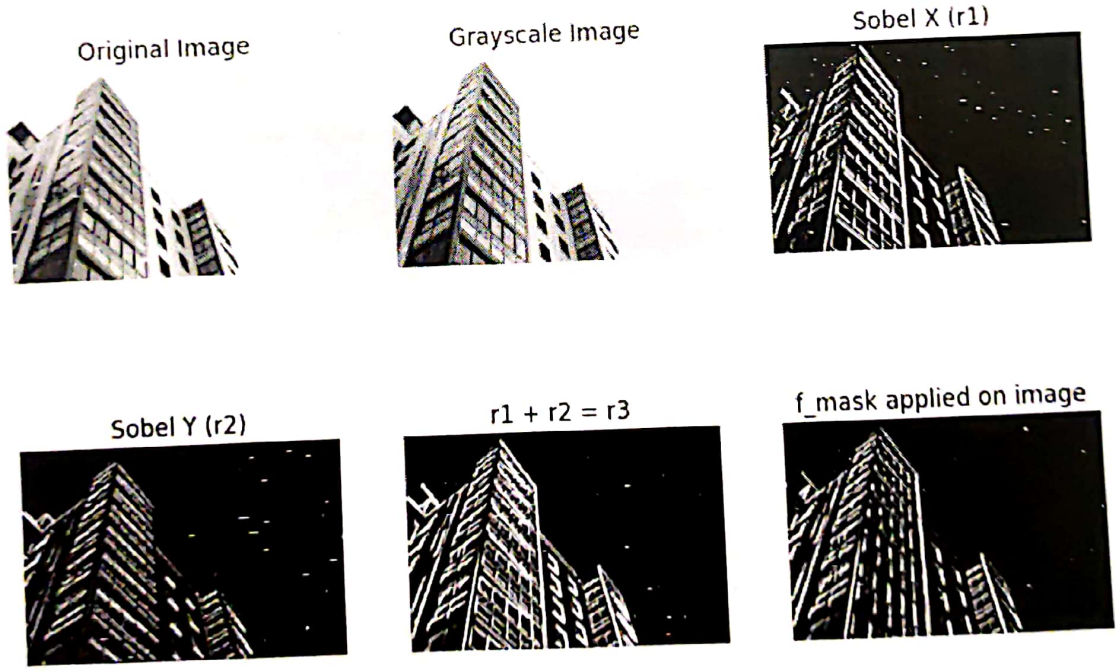
```
# High-pass filtered image
plt.subplot(1, 3, 2)
plt.imshow(high_pass)
plt.title('High-Pass Filtered Image')
plt.axis('off')

plt.tight_layout()
plt.show()
```

OUTPUT:



```
lt.subplot(3, 3, 4), plt.imshow(r2, cmap='gray'), plt.title('Sobel Y (r2)')
lt.axis('off')

 Step 5: Add r1 and r2 to obtain r3 and display
3 = cv2.add(np.abs(r1), np.abs(r2))  # Adding r1 and r2 plt.subplot(3, 3, 5), plt.imshow(r3,
:map='gray'), plt.title('r1 + r2 = r3')
plt.axis('off')

# Step 6: Add fx and fy to obtain f_mask
f_mask = sobel_x_kernel + sobel_y_kernel  # Adding Sobel X and Y kernels
# plt.subplot(3, 3, 6), plt.imshow(f_mask, cmap='gray'), plt.title('f_mask = fx + fy')
plt.axis('off')

# Step 7: Apply f_mask on the grayscale image and display f_mask_applied =
cv2.filter2D(gray_image, -1, f_mask)  # Applying combined mask
plt.subplot(3, 3, 6), plt.imshow(f_mask_applied, cmap='gray'), plt.title('f_mask applied on image')
plt.axis('off')

# Show all the steps in one figure plt.show()
```

OUTPUT:

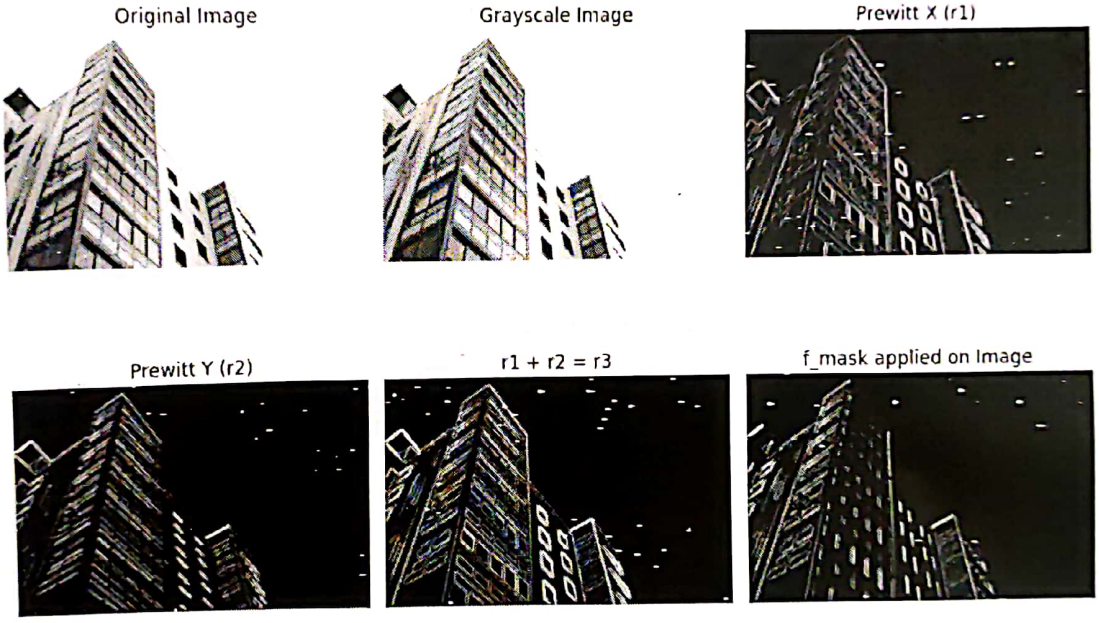


PERWITT

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

# Step 1: Load and display the original image image =
cv2.imread('image2.jpg') plt.figure(figsize=(10, 10))
plt.subplot(3, 3, 1), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)),
plt.title('Original Image') plt.axis('off')

# Step 2: Convert the image to grayscale and display it gray_image = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY) plt.subplot(3, 3, 2), plt.imshow(gray_image, cmap='gray'),
plt.title('Grayscale Image')
plt.axis('off')

# Define Prewitt kernels prewitt_x = np.array([[ -
1, 0, 1],
                                    [ -1, 0, 1],
                                    [ -1, 0, 1]])

prewitt_y = np.array([[ -1, -1, -1],
                                    [ 0, 0, 0],
                                    [1, 1, 1]])

# Step 3: Apply Prewitt operator (derivative in x direction) and display the result (r1)
r1 = cv2.filter2D(gray_image, cv2.CV_64F, prewitt_x)
r1 = np.abs(r1)  # Absolute values to handle negative edges plt.subplot(3, 3, 3), plt.imshow(r1,
cmap='gray'), plt.title('Prewitt X (r1)')
plt.axis('off')

# Step 4: Apply Prewitt operator (derivative in y direction) and display the result (r2)
r2 = cv2.filter2D(gray_image, cv2.CV_64F, prewitt_y)
r2 = np.abs(r2)  # Absolute values to handle negative edges plt.subplot(3, 3, 4), plt.imshow(r2,
cmap='gray'), plt.title('Prewitt Y (r2)')
plt.axis('off')

# Step 5: Add r1 and r2 to get r3, and display r3 r3 = cv2.add(r1, r2)
plt.subplot(3, 3, 5), plt.imshow(r3, cmap='gray'), plt.title('r1 + r2 = r3')
plt.axis('off')
```

```
# Step 6: Add Prewitt X and Prewitt Y to obtain the filter mask
f_mask = prewitt_x + prewitt_y
plt.axis('off')
f_mask_applied = cv2.filter2D(gray_image, cv2.CV_64F, f_mask)

# Display the result of applying f_mask on the grayscale image
plt.subplot(3, 3, 6), plt.imshow(np.abs(f_mask_applied), cmap='gray'),
plt.title('f_mask applied on Image')
plt.axis('off')


plt.tight_layout()
plt.show()
```

OUTPUT:



Assignment 1

Shashwat Shah
60004220126
BE comps C22

Q1 Explain Parsevals Energy Theorum.

→ Parsevals Energy theorum is a fundamental principle in signal processing and fourier analysis that relate total energy of signal in time domain to total energy of its representation in the frequency domain.

Definition.

It states that total energy of a signal is equal to tot energy of its fourier transform.

$$\int_{-\infty}^{\infty} |x(t)|^2 \, dt = \int_{-\infty}^{\infty} |X(f)^2| \, df.$$

Where.

$x(t)$ is time domain signal & $X(f)$ is its fourier transform.

Time domain energy is calculated as

$$E = \int_{-\infty}^{\infty} |x(t)|^2 \, dt.$$

The theorum illustrates the principle of energy conservation in signal indicating that regardless of whether the signal is analyzed in time or frequency domain the total energy remains invariant.

It aid in analyzing the distribution of energy such as audio compression

3 applications of DSP.

(1) Audio & speech processing
This involves manipulating sound signals to improve quality enable communication or extract meaningful information. DSP techniques are essential (1) filtering. (2) Speech recognition are some of the examples.

(2) Statistical signal processing.
This deals with analyzing and interpreting signals that are affected by noise and uncertainty. It uses statistical methods
(1) Estimation (2) Detection these help identify presence of signal amidst noise.

(3) Data compression.
This involves reducing amount of data required to represent a signal without significantly degrading quality.
(1) Transform coding (2) Lossless compression.

Write short notes on:

) Hotelling / KL transform
This is a statistical technique used for dimensionality reduction and feature extraction.
It involves for decomposition of a multivariate random process into orthogonal components.

ii) Moving average method for thresholding
This is used for thresholding. is determined based on average value.

iii) Iterative thresholding.
It is an adaptive thresholding technique used for image segmentation that adjusts threshold based on pixel intensities iteratively.

iv) Multivariate thresholding.
This is an extension of traditional thresholding. Instead of relying on a single intensity value this method utilizes multiple features.

v) JPEG compression algorithm.
Widely used lossy image compression technique that reduces file size while maintaining acceptable image quality. The image is converted from RGB to YCbCr. Each 8x8 block of pixels undergo DCT.