List of Experiments
B.E. (Comp.) Sem VII
Subject: Digital Signal and Image Processing Lab

PAGE NO :
DATE : / /

Experiment No-1

Aim: To Sample and Reconstruct the Analog Signal.

Theory:

What is sampling?

- In signal processing, Sampling is the reduction of a continuous-time signal to a discrete time signal.
- A common example is the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete time signal)
- A sample is a value or set of values at a point in time and / or space
- A sampler is a subsystem or operation that extracts samples (a discrete time signal)
- A Theoretical ideal sampler produces samples equivalent to the instantaneous Value of the continous signal at the desired points.
- The original signal is retrievable from a sequence of samples, up to the Nyquist limit, by passing the sequence of samples through a type of low pass filter called a Reconstruction filter

PAGE NO.:
DATE: / /

Concept of Digital frequency

Unlike of analog frequency a digitally generated frequency does not have infinite resolution.

- A digital frequency can only take discrete values A digital sine wave, for example, can only take on discrete values for frequency, phase and amplitude.
- The frequency resolution of a digitally generated sinusoidal is limited by the period of the digital sample clock $T = \frac{1}{fs}$

and the precision of the sinusoidal wave from amplitual is limited by the bit width of each digital samples. The normalised values of these limits are respectively 0.5 are 1.0 cycles samples.

Aliasing and how to hand Aliasing.

- In signal processing and related disciplines aliasing is an effect that causes different signals to become indistinguisable when samples.
- It also often refers to the distortion or artifact that results when a signal reconstructed from sample is different from the original continuous signal.
- Aliasing can occur in signals samples in time for instance digital audio and in

PAGE NO.:
DATE: / /

referred to as temporal aliasing. It can also occur in spatially sampled signals (eg moire pattern in digital images) this type of aliasing is called spatial aliasing.

Anti aliasing and Nyquist Theorem:

- An anti-aliasing filter (AAF) is a filter used before a signal sampler to restrict the bandwidth of a signal to approximately or completely satisfy the Nyquist-shannon sampling theorem over the band of interest.
- Since the theorem states that unambiguous reconstruction of the signal from its samples is possible when the powers of frequencies above the Nyquist frequency is zero, a real anti-aliasing
- A realizable anti-aliasing filter will typically either permit aliasing to occur or else attenuate some in-band frequencies close to the Nyquist limit.
- For this many samples practical systems higher than would be theoretically required by a perfect AAF in order to ensure that all frequencies of interest can be reconstructed a practice called oversampling.

example:

Consider the signal (Analog signal)
$x(t) = 4\cos 200\pi t - 3\sin 600\pi t + 2\cos 1200\pi t$
$f_s = 700$ samples/seconds.

Solution : $T = \frac{1}{f_s}$

$x(nT) = 4 \cos 2000\pi nT - 3\sin 6000\pi t + 8 \sin 12000\pi nt$

$T = 1/500$

$x(n) = 4 \cos \frac{2\pi}{5} nT - 3 \sin \frac{6\pi}{5} nT + 8 \sin \frac{12\pi}{5} nt \; T$

$\sin \frac{6\pi}{5}$ and $\frac{12\pi}{5}$ are greater than π,

we map them to $\frac{4\pi}{5}$ and $\frac{2\pi}{5}$ respectively,

Sampled signal

$x(n) = 4\cos \frac{2\pi}{5} nT - 3 \sin \frac{4\pi}{5} nT + 8 \sin \frac{2\pi}{5} nT$

$x_R(t) = 4 \cos \frac{2\pi}{5} 500t - 3 \sin \frac{4\pi}{5} 500t + 8 \sin \frac{2\pi}{5} 100t$

Reconstructed signal

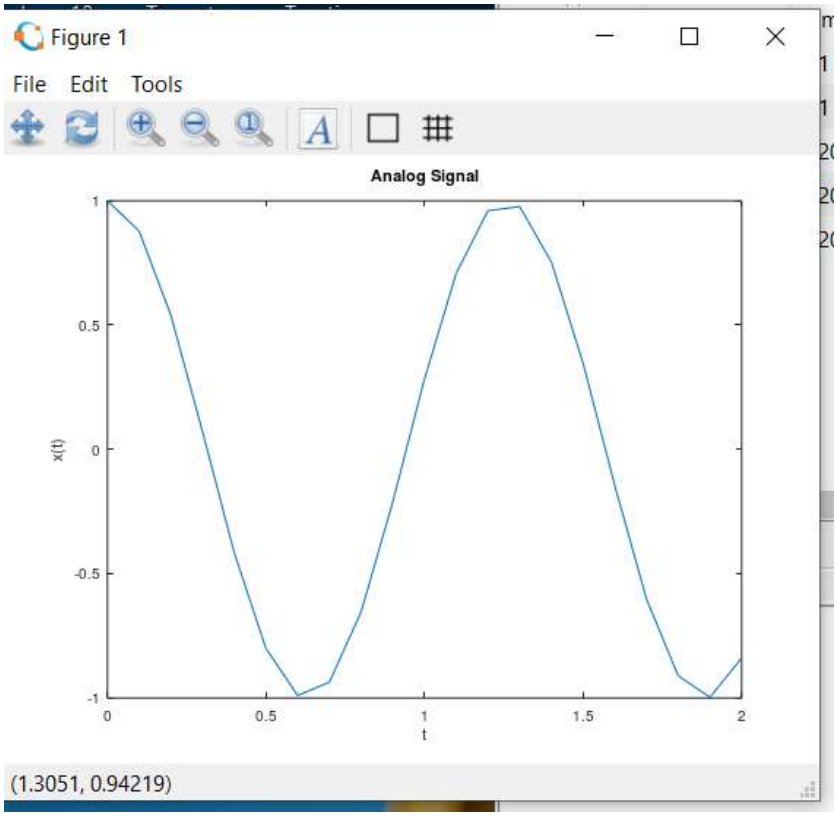
$x_R(t) = 4\cos 200\pi t - 3\sin 400\pi t + 8\sin 200\pi t$

C21-85

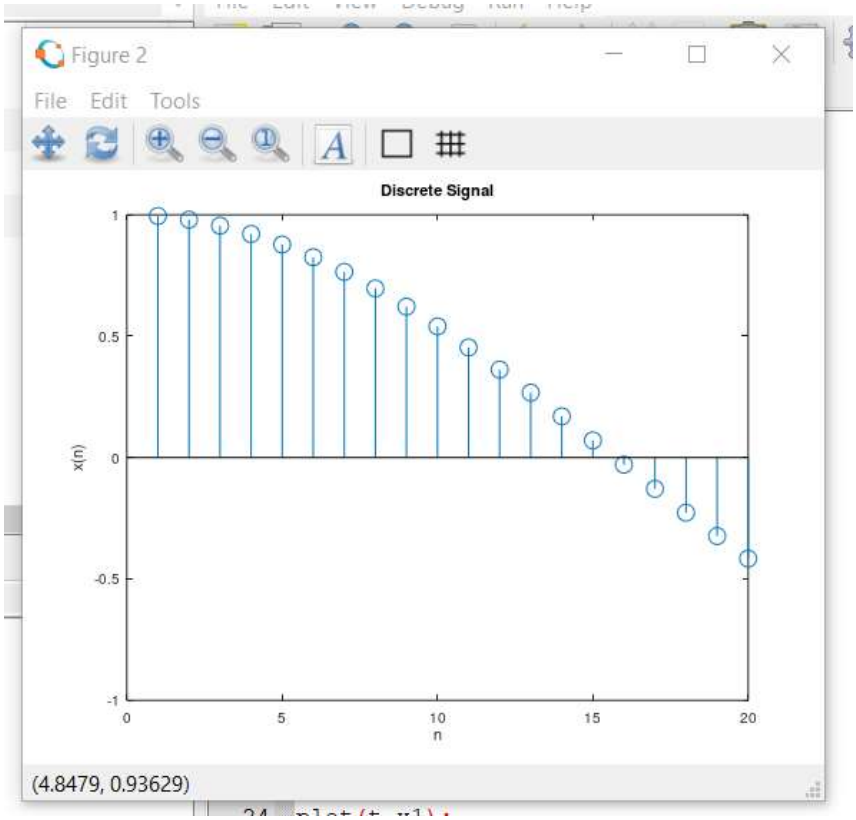
Program Of Sampling and Reconstruction

```
clc
clear all
t = [0:0.1:2]
x1 = cos(5*t);
figure(1);
plot(t,x1);
title('Analog Signal');
xlabel('t');
ylabel('x(t)');
Fs = 50;
n = [1:1:20];
T = 1/Fs;
t = n*T;
x1 = cos(5*n/Fs);
figure(2);
stem(n,x1);
title('Discrete Signal');
xlabel('n');
ylabel('x(n)');
n = t/T;
n = t*Fs;
x1 = cos(5*t*Fs);
figure(3);
plot(t,x1);
title('Reconstructed Signal');
xlabel('t');
ylabel('x(t)');
```

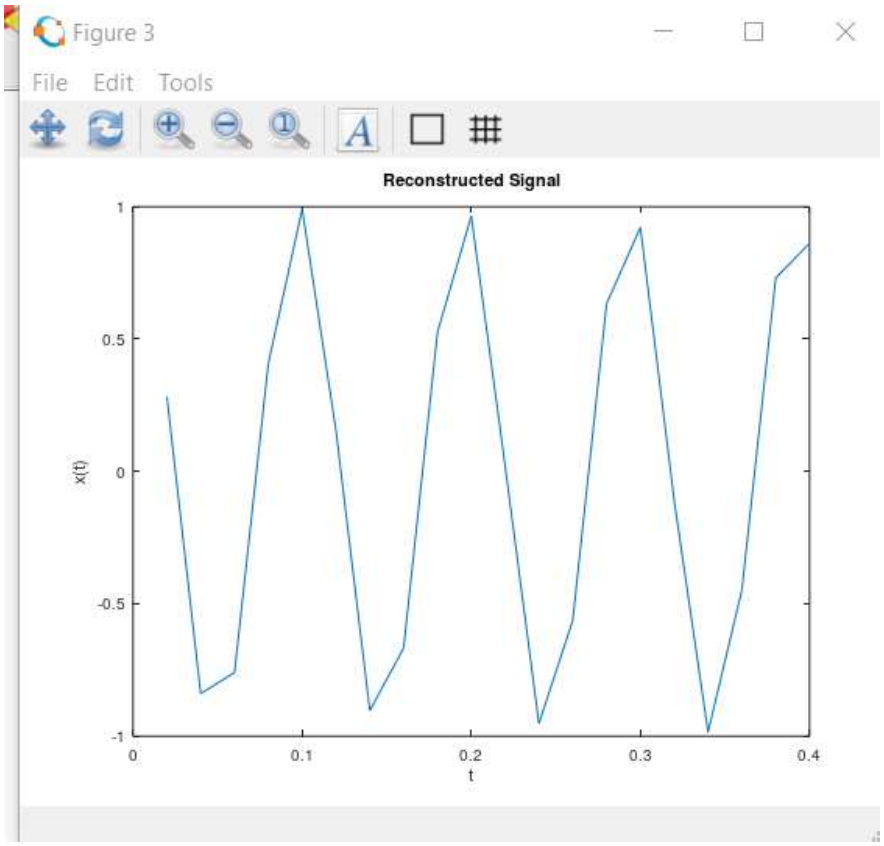
Output Figure-1 (Analog Signal)



Output Figure-1 (Discrete Signal)



Output Figure-3 (Reconstructed Signal)



C22-85

EXPERIMENT – 2

Aim - Implement Discrete Correlation

Theory -

Correlation is a measure of similarity between two signals.

The concept of correlation in general quantifies the similarity of two spatial- or time-dependent signals Xx and Yy. The main property of correlation is that both signals do not have to depend on each other;
only statements regarding their similarity can be given. Correlation is, however, not adequate to reveal the existence of a causal relationship between Xx and Yy.

Correlation between signals indicates the measure up to which the given signal resembles another signal. In other words, if we want to know how much similarity exists between the signals 1 and 2, then we need to find out the correlation of Signal 1 with respect to Signal 2 or vice versa.

The general formula for correlation is

$$\int_{-\infty}^{\infty} x_1(t)x_2(t-\tau)dt$$

There are two types of correlation:

- Auto correlation
- Cross correlation

Auto Correlation Function

It is defined as correlation of a signal with itself. Auto correlation function is a measure of similaritybetween a signal & its time delayed version. It is represented with R(τ).

Consider a signal x(t). The auto correlation function of x(t) with its time delayed version is given by

$$R_{11}(\tau) = R(\tau) = \int_{-\infty}^{\infty} x(t)x(t-\tau)dt$$

Where τ = searching or scanning or delay parameter.

Cross Correlation Function

Cross correlation is the measure of similarity between two different signals. Consider two signals x1(t)and x2(t). The cross correlation of these two signals is given by

$$R_{12}(\tau) = \int_{-\infty}^{\infty} x_1(t) x_2(t-\tau)\, dt$$

Example: Correlation using convolution table method x(n) = {3, 5, 1, 2} (0 is at 5) y(n) = {1, 4, 3} (0 is at 1)

	3	4	1
3	9	12	3
5	15	20	5

1	3	4	1
2	6	8	2

Rxy(n) = {9, 27, 26, 15, 9, 2} (0 is at 15)

Program

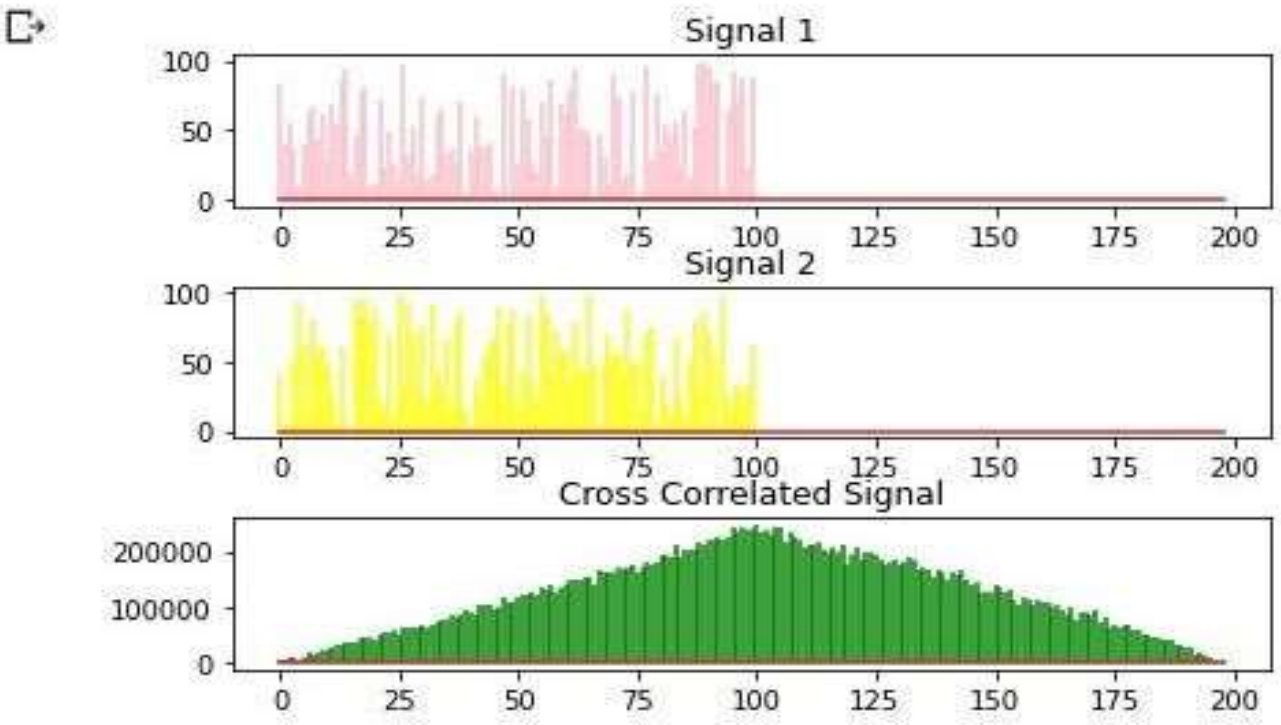
```
import matplotlib.pyplot as plt
from random import sample
def correlate(x, y):
    n = len(x)
    m = len(y)
    x = [0] * (m - 1) + x + [0] * (m - 1)
    y = y + [0] * (n - 1 + m - 1)
    z =[0] * (n + m - 1)
    for i in range(n + m - 1):
        z[i] = sum([i * j for i, j in zip(x, y)])
        y = y[-1:] + y[:-1]
    return z

def stemPlot(axis, x, y, color):
    axis.stem( x, y, linefmt = color, markerfmt = " ", use_line_collection = True)

samples = 100
X = list(range(samples))
Y1 = sample(X, samples)
Y2 = sample(X, samples)
C = correlate(Y1, Y2)
Y1 += [0] * (samples - 1)
Y2 += [0] * (samples - 1)
X = list(range(2 * samples - 1))
figure, axes = plt.subplots(3)
figure.tight_layout()
axes[0].set_title('Signal 1')
axes[1].set_title('Signal 2')
axes[2].set_title('Cross Correlated Signal')
stemPlot(axes[0], X, Y1, 'pink')
stemPlot(axes[1], X, Y2, 'yellow')
stemPlot(axes[2], X, C, 'green')

plt.show()
figure.savefig('EXP2.png', dpi = 600)
```

Output -



Conclusion – Implemented Discrete Correlation Successfully.

C22-85

Exp-3

Aim: To perform Discrete Convolution.

Theory:

It is a mathematical operation to calculate the output of any linear time invariant by finding the zero-state response of related linear time invariant systems.

It is based on concept of linearity and time invariance and assumes that system information is known as in terms of impulse response h(n)

A discrete time system performs an transformation operation on input signal based on predefined criteria to produce a modified output signal.

Example:

Tabular Method

	4	2	1	3
1	4	2	1	3
2	8	4	2	6
2	8	4	2	6
1	4	2	1	3

$y(n) = \{4, 10, 13, 13, 10, 7, 3\}$ 0 is at top

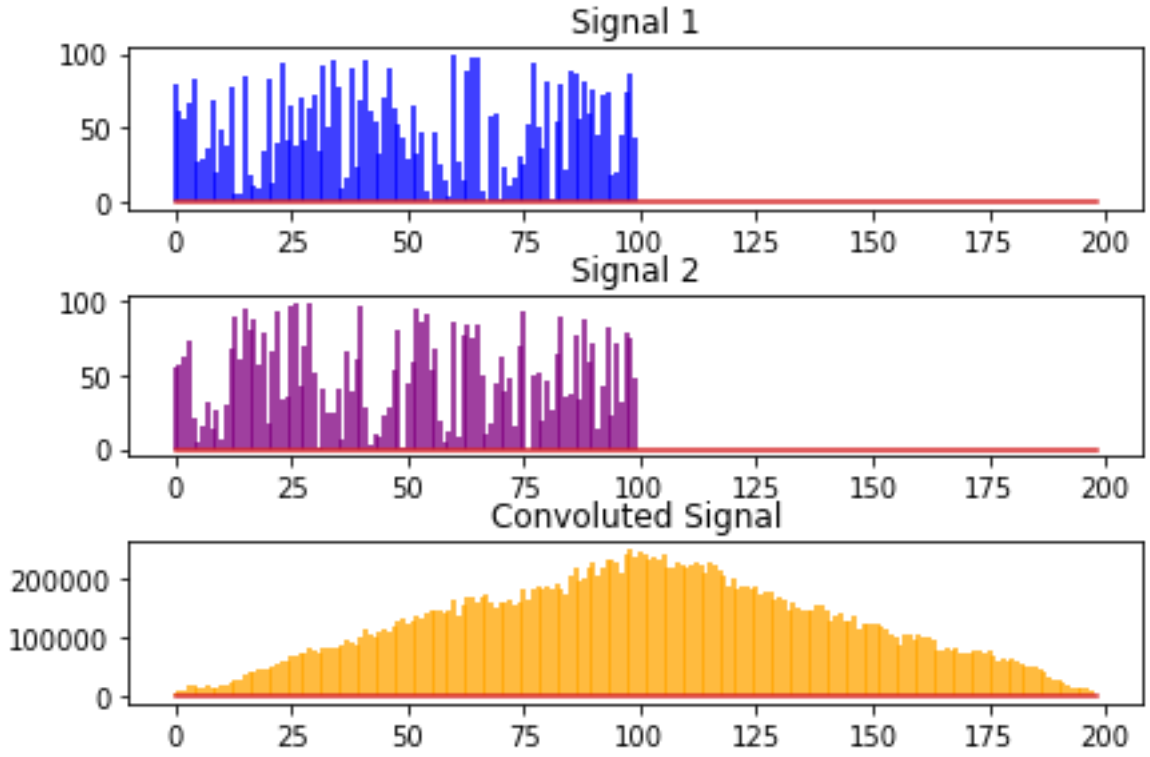
Conclusion: Discrete convolution is successfully implemented.

Code:

```
import random
import matplotlib.pyplot as pltdef
convolute(x, y):
    n = len(x)
    m = len(y)
    dp = []
    for i in range(n):
        dp.append([])
        for j in range(m):
            dp[i].append(x[i] * y[j])
    z = [0] * (n + m - 1)
    for i in range(n):
        for j in range(m):
            z[i + j] += dp[i][j]
    return z
def stemPlot(axis, x, y, color):
    axis.stem(
        x,
        y,
        linefmt = color,
        markerfmt = " ",
        use_line_collection = True
```

```
    )
samples = 100
X = list(range(samples))
Y1 = random.sample(X, samples)
Y2 = random.sample(X, samples)
C = convolute(Y1, Y2)
Y1 += [0] * (samples - 1)
Y2 += [0] * (samples - 1)
X = list(range(2 * samples - 1))
figure, axes = plt.subplots(3)
figure.tight_layout()
axes[0].set_title('Signal 1')
axes[1].set_title('Signal 2')
axes[2].set_title('Convoluted Signal')
stemPlot(axes[0], X, Y1, 'blue')
stemPlot(axes[1], X, Y2, 'purple')
stemPlot(axes[2], X, C, 'orange')
plt.show()
figure.savefig('EXP3.png', dpi = 600)
```

Output


Signal 1
100
50
0
0
25
50
75
100
125
150
175
200
Signal 2
100
50
0
0
25
50
75
100
125
150
175
200
Convoluted Signal
200000
100000
0
0
25
50
75
100
125
150
175
200

PAGE NO.:
DATE: / /

Experiment no 7

Aim: To implement Discrete Fourier Transformation

Theory:

An N-point DFT is expressed as the multiplication $X = Wx$ where x is the original input signal W is the N-by-N square DFT matrix and X is the DFT of the signal.

$$W = \left(\frac{W^{jk}}{\sqrt{N}}\right) \; j, k = 0 \ldots N-1$$

or

The transformation matrix W can be defined as equivalently,

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & w^3 & \cdots & w^{n-1} \\ 1 & w^2 & w^3 & w^4 & \cdots & w^{2(n-1)} \\ 1 & w^3 & w^4 & w^5 & \cdots & w^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & w^{3(n-1)} & \cdots & w^{(n-1)(n-1)} \end{bmatrix}$$

$-2\pi i/N$

where $\omega = e$ is a primitive N^{th} root of unity in which $i^2 = -1$ we can avoid writing large exponents for ω using the fact that for any exponent x and n we have the identity $\omega^{x \bmod n} = \omega$

This is the vandermonde matrix for the roots of unity, upto the normalization factor. Note that the normalization factor in front of the sum. 1 And the sign of the exponent in $\sqrt{N}$ are merely convention, and differ in some treatment. All of the following discussion applies regardless of the convention, with atmost minor adjustment. The only important thing is that the forward and inverse transform have opposite-sign exponent, and that the product of their normalization factor be $\frac{1}{N}$. However, the $\frac{1}{\sqrt{N}}$ choice here makes the resulting DFT matrix unitary, which is convenient in many circumstances.

Example: Determine 2 point, 4 point, 8 point DFT for sequence,

$x(n) = u(n) - u(n-2)$

Magnitude $= |X(k)| = (Real^2 + Image^2)$

Phase $= \tan^{-1}(\text{imag}/\text{real})$

PAGE NO.:
DATE: / /

solution

for $n = 2$,

$x(n) = \{1, 1\}$

$$W = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

2 point DFT matrix.

$x(k) = W x(n)$

$= \{2, 0\}$

magnitude = $\{2, 0\}$

Phase = $\{0, 0\}$

for $n = 4$,

$x(n) = \{1, 1, 0, 0\}$

$$W = \frac{1}{\sqrt{4}} \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix}$$

4 point DFT matrix.

$X(k) = W^* x(n)$

$= \{2, 1-j, 0, 1+j\}$

Magnitude = $\{2, 1.41, 0, 1.41\}$

Phase = $\{0, -\frac{\pi}{4}, 0, \frac{\pi}{4}\}$

for n=8.

x(n) = {1, 1, 1, 1, 0, 0, 0, 0}

8 point DFT matrix

$$N = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^8 & W^{10} & W^{12} & W^{14} \\ W^0 & W^3 & W^6 & W^9 & W^{12} & W^{15} & W^{18} & W^{21} \\ W^0 & W^4 & W^8 & W^{12} & W^{16} & W^{20} & W^{24} & W^{28} \\ W^0 & W^5 & W^{10} & W^{15} & W^{20} & W^{25} & W^{30} & W^{35} \\ W^0 & W^6 & W^{12} & W^{18} & W^{24} & W^{30} & W^{36} & W^{42} \\ W^0 & W^7 & W^{14} & W^{21} & W^{28} & W^{35} & W^{42} & W^{49} \end{bmatrix}$$

X(k) = W^{kn} x(n) = {4, 1-j(1+√2), 0, 1+j(1-√2), 0, 1+j(1-√2), 0, 1+j(1+√2)}

Magnitude = {4, 2.613, 0, 1.082, 0, 1.082, 0, 2.613}.

Phase = {0, -1.178, 0.392, 0, 0.390, 0, -1.178, 0}.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings('ignore')

from random import sample, randint

PRECISION = 1e-10

def twiddleFactorMatrix(point, unitary = False):

    def changePrecision(n):
        r, i = 1, 1j
        if abs(n.real) < PRECISION:
            r = 0
        if abs(n.imag) < PRECISION:
            i = 0j
        return (r * n.real) + (i * n.imag)

    def omega(N):
        return changePrecision(
            np.power(np.e, -2 * np.pi * 1j / N)
        )

    w = omega(point)
    tfm = [[0 for j in range(point)] for i in range(point)]
    for i in range(point):
        tfm[i][0] = np.power(w, 0)
        for j in range(1, point):
            tfm[i][j] = changePrecision(tfm[i][j - 1] * np.power(w, i))
```

```
    print(f'Twiddle Factor Matrix for {point} Point DFT')
    tfm = np.around(tfm, decimals = 3)
    for row in tfm:
        print(*row)
    print()
    if unitary:
        return (tfm, 1 / np.power(point, 0.5))
    return (tfm, 1)

def signal(point):
    X = list(range(point))
    Y = [randint(0, 1) for x in range(point)]
    return X, Y

def dft(x, y):
    n = len(y)
    z = [0 for i in range(n)]
    for i in range(n):
        z[i] = sum([j * k for j, k in zip(x[i], y)])
    return np.around(z, decimals = 3)

def magnitude(x):
    return list(map(
        lambda n: np.power((np.power(n.real, 2) + np.power(n.imag, 2)), 0.5),
        x
    ))

def phase(x):
    n = len(x)
    p = [0 for i in range(n)]
```

```
    for i in range(n):
        if x[i].real == 0 and x[i].imag < 0:
            a = np.NINF
        elif x[i].real == 0 and x[i].imag >= 0:
            a = np.PINF
        else:
            a = x[i].imag / x[i].real
        p[i] = np.arctan(a)
    return p

def stemPlot(axis, x, y, color):
    axis.stem(
        x,
        y,
        linefmt = color,
        markerfmt = " ",
        use_line_collection = True
    )

points = [2, 4, 8]

for point in points:
    tfm, factor = twiddleFactorMatrix(point)
    X, Y = signal(point)
    DFT = dft(tfm, Y)
    M, P = magnitude(DFT), phase(DFT)
    figure, axes = plt.subplots(4)
    figure.tight_layout()

    axes[0].set_title('Signal')
    axes[1].set_title('Discrete Fourier Transform')
```

```
axes[2].set_title('Magnitude Plot')
axes[3].set_title('Phase Plot')

stemPlot(axes[0], X, Y, 'blue')
stemPlot(axes[1], X, DFT, 'orange')
stemPlot(axes[2], X, M, 'green')
stemPlot(axes[3], X, P, 'red')

plt.show()
figure.savefig(f'DSIP_EXP_4_{point}PT.png', dpi = 600)
```

Output :

RESTART: C:\Users\Gaurav Poojary\OneDrive\Desktop\Experiments\DSIP\DSIP_EXP_4.py

Twiddle Factor Matrix for 2 Point DFT

(1+0j) (1+0j)

(1+0j) (-1+0j)

Twiddle Factor Matrix for 4 Point DFT

(1+0j) (1+0j) (1+0j) (1+0j)

(1+0j) -1j (-1+0j) 1j

(1+0j) (-1+0j) (1+0j) (-1+0j)

(1+0j) 1j (-1+0j) -1j

Twiddle Factor Matrix for 8 Point DFT

(1+0j) (1+0j) (1+0j) (1+0j) (1+0j) (1+0j) (1+0j) (1+0j)

(1+0j) (0.707-0.707j) -1j (-0.707-0.707j) (-1+0j) (-0.707+0.707j) 1j (0.707+0.707j)

(1+0j) -1j (-1+0j) 1j (1+0j) -1j (-1+0j) 1j

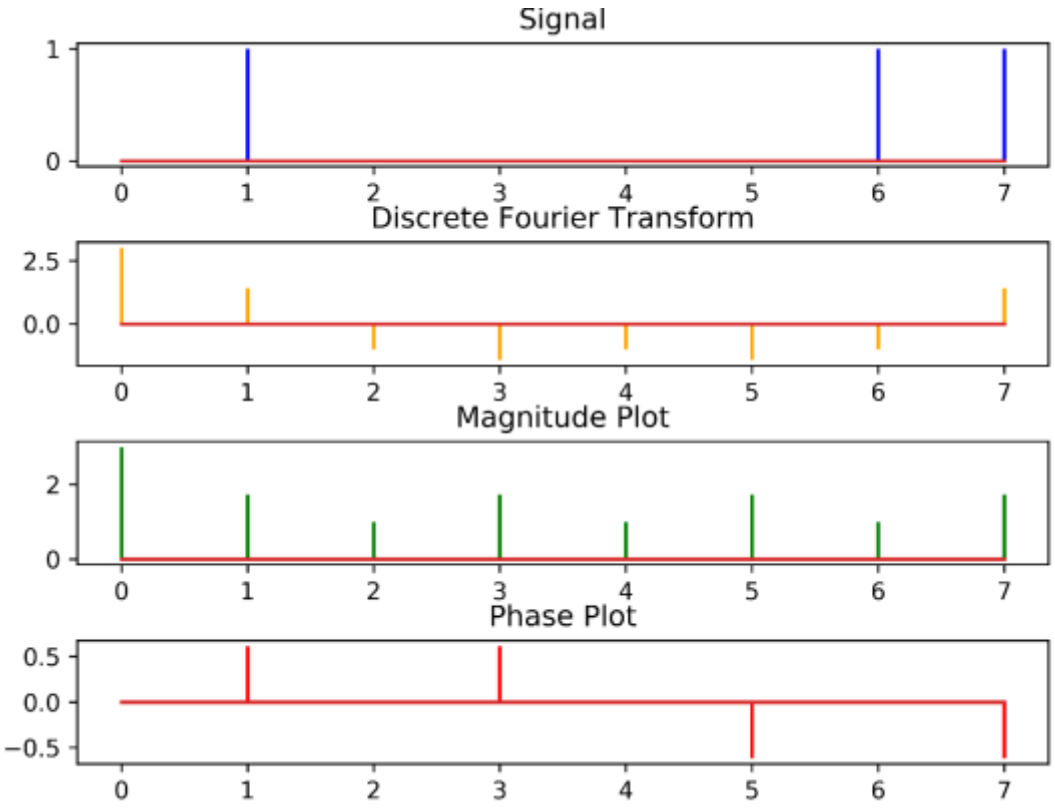
(1+0j) (-0.707-0.707j) 1j (0.707-0.707j) (-1+0j) (0.707+0.707j) -1j (-0.707+0.707j)

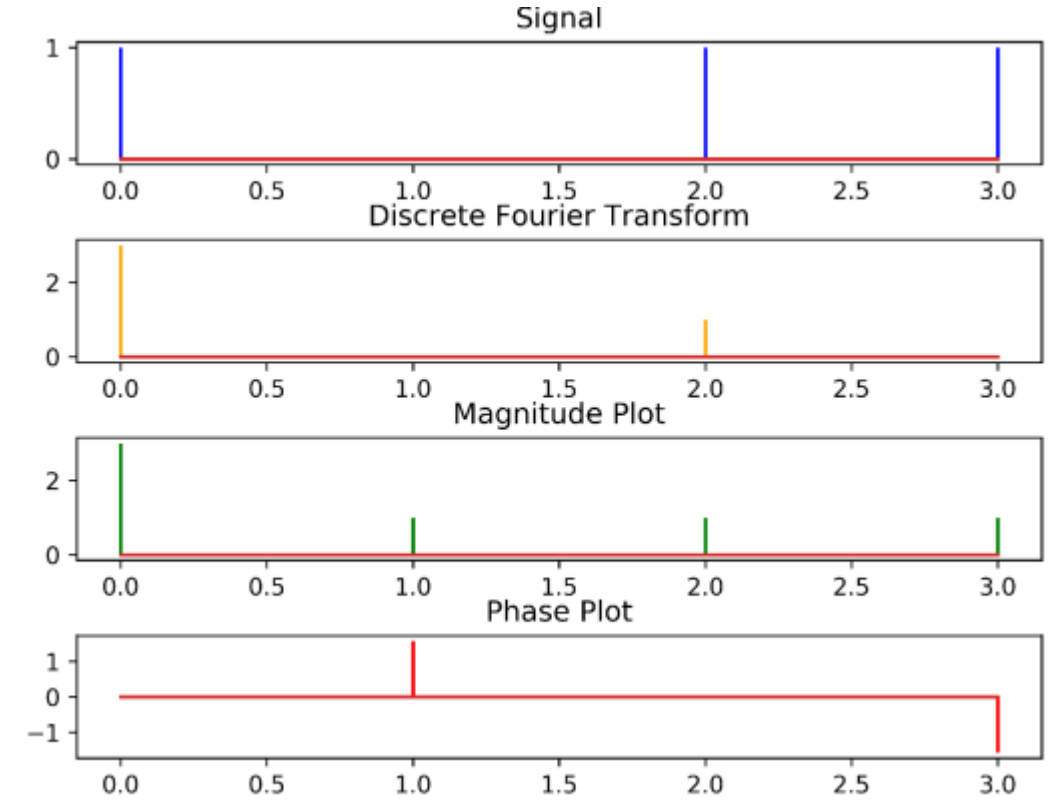
(1+0j) (-1+0j) (1+0j) (-1+0j) (1+0j) (-1+0j) (1+0j) (-1+0j)

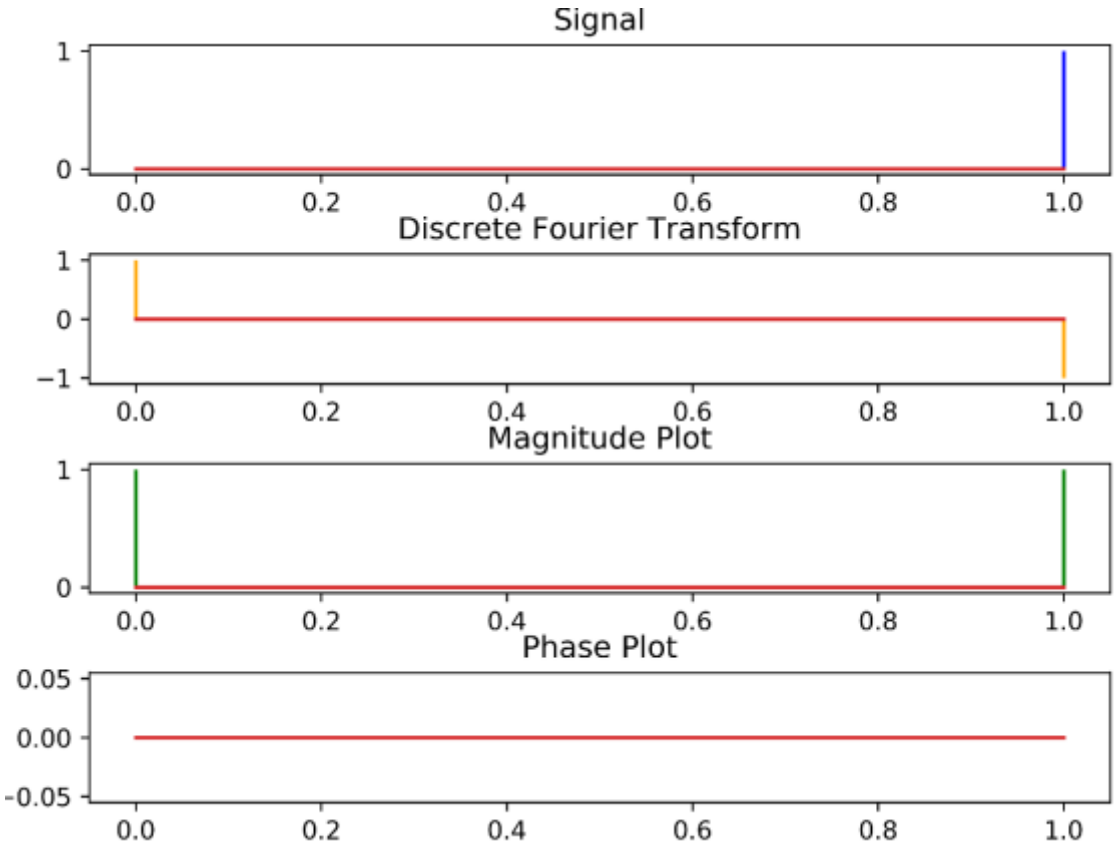
(1+0j) (-0.707+0.707j) -1j (0.707+0.707j) (-1+0j) (0.707-0.707j) 1j (-0.707-0.707j)

(1+0j) 1j (-1+0j) -1j (1+0j) 1j (-1+0j) -1j

(1+0j) (0.707+0.707j) 1j (-0.707+0.707j) (-1+0j) (-0.707-0.707j) -1j (0.707-0.707j)


Signal
Discrete Fourier Transform
Magnitude Plot
Phase Plot


Signal
Discrete Fourier Transform
Magnitude Plot
Phase Plot


Signal
1
0
0.0
0.2
0.4
0.6
0.8
1.0
Discrete Fourier Transform
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0
Magnitude Plot
1
0
0.0
0.2
0.4
0.6
0.8
1.0
Phase Plot
0.05
0.00
-0.05
0.0
0.2
0.4
0.6
0.8
1.0

Exp 5 -: Implement fast- fourier transformation

Theory:

The discrete fourier transformation (DTF) can be written as follows.

$$x[k] = \sum_{n=0}^{N-1} x[N] e^{\frac{-j 2\pi kn}{N}}$$

To determine the DTF of a discrete signal $x[n]$, we multiply each of its value by e raised to some function of n. We then sum the results obtained for given n. If we used a computer to calculate the discrete fourier transform of a signals it would need to perform N (multiplication) * N (additions) = $O(N^2)$ operations. As the name implies, the fast fourier transform (FFT) is an algorithm that determines discrete fourier transform of an input significantly

discrete fourier transform $O(N^2)$

fast fourier transform $O(N \log_2 N)$

N	1000	10^6	10^9
N^2	10^6	10^{12}	10^{18}
$N \log_2 N$	10^4	20×10^6	30×10^9

Say it took 1 ns to perform one operation

It would take the fast fourier transform algorithm approximately 30 seconds to compute the

discrete fourier transformation for problem of size $N = 10^9$. In contrast the regular algorithm would need several decades.

10^{10} ns → 31.7 years.

30×10^9 ns → 30 seconds.

Fast fourier transform algorithm

Suppose, we separated the fourier transform into its even and odd indexed sub-sequences.

$n = 2r$ if even

$n = 2r+1$ if odd

where $r = 1, 2, \ldots N/2 - 1$

After performing a bit of algebra, we end up with the summation of two terms. The advantage of this approach lies in the fact that the even and odd indexed sub-sequences can be computed concurrently

Suppose $N = 8$. To visualize the flow of data with this we can make use of a butterfly diagram. We compute the discrete fourier transform for the even and odd terms simultaneously. Then we calculate $x[k]$ using the formula from above

We can express the gain in terms of Big O notations as follows. The first term comes from the fact that we compute the discrete.

fourier transform on half the original input. For the final step, it takes N steps to add up the fourier transform for a particular k. We account for this by adding N to the final product.

Notice how, we were able to cut the time taken to compute the fourier transform by a factor of 2. We can further improve the algorithm by applying the divide and conquer approach, having the computational cost each time.

Program:

```
from math import *

N = int(input("Enter the length of x(n):"))
print("Enter the signal x(n):")
x = [int(i) for i in input().split()]
print("Enter the type of FFT:")
print("1. 2-point\n2. 4-point\n3. 8-point")
ch = int(input("Enter your choice:"))

def DFT_matrix(N):
    W = [0]*N
    pi = 3.14
    for k in range(N):
        if k==0:
            W[k] = 1
        else:
            W[k] = complex(cos((2*pi*k)/N),-sin((2*pi*k)/N))
    for k in range(N):
        if(round(W[k].imag) == 0):
            W[k] = float(round(W[k].real,2))
        else:
            W[k] =
complex(float(round(W[k].real,2)),float(round(W[k].imag,2)))
    return W

def two_point_FFT(N,x):
    X = [0]*N
    W = DFT_matrix(N)
    for k in range(N):
        X[k] = x[0] + (W[k] * x[1])
    return X

def four_point_FFT(N,x):
    x_even = [x[0],x[2]]
    x_odd = [x[1],x[3]]
    print("x_even:",x_even)
    print("x_odd:",x_odd)

    G = two_point_FFT(2,x_even)
    H = two_point_FFT(2,x_odd)
    #periodicity of G and H
    G.extend(G)
    H.extend(H)
    W = DFT_matrix(4)
    X = [0]*N
    for k in range(N):
        X[k] = G[k] + W[k] * H[k]
    return X

def eight_point_FFT(N,x):
    x_even = [x[0],x[2],x[4],x[6]]
    x_odd = [x[1],x[3],x[5],x[7]]
    print("x_even:",x_even)
    print("x_odd:",x_odd)

    G = four_point_FFT(4,x_even)
    H = four_point_FFT(4,x_odd)
    #periodicity of G and H
    G.extend(G)
```

```
    H.extend(H)
    W = DFT_matrix(8)
    X = [0]*N
    for k in range(N):
        X[k] = G[k] + W[k] * H[k]
    return X

if ch == 1:
    X = two_point_FFT(2,x)
elif ch == 2:
    X = four_point_FFT(4,x)
else:
    X = eight_point_FFT(8,x)

print(str(N) + " point FFT is given by:" + str(X))
print("Total number of complex additions are:",(N * log(N,2)))
print("Total number of complex multiplications are:",((N/2) * log(N,2)))
```

Output:

1) 2-point

Enter the length of x(n):2

Enter the signal x(n):

8 14

Enter the type of FFT:

1. 2-point

2. 4-point

3. 8-point

Enter your choice:1

2 point FFT is given by:[22.0, -6.0]

Total number of complex additions are: 2.0

Total number of complex multiplications are: 1.0

2) 4-point

Enter the length of x(n):4

Enter the signal x(n):

6 13 18 22

Enter the type of FFT:

1. 2-point

2. 4-point

3. 8-point

Enter your choice:2

x_even: [6, 18]

x_odd: [13, 22]

4 point FFT is given by:[59.0, (-12+9j), -11.0, (-12-9j)]

Total number of complex additions are: 8.0

Total number of complex multiplications are: 4.0

3) 8-point

Enter the length of x(n):8

Enter the signal x(n):

3 5 10 12 15 20 23 27

Enter the type of FFT:

1. 2-point

2. 4-point

3. 8-point

Enter your choice:3

x_even: [3, 10, 15, 23]

x_odd: [5, 12, 20, 27]

x_even: [3, 15]

x_odd: [10, 23]

x_even: [5, 20]

x_odd: [12, 27]

8 point FFT is given by:[115.0, (-12+34.3j), (-15+14j), (-12+8.299999999999997j), -13.0, (-12-8.299999999999997j), (-15-14j), (-12-34.3j)]

Total number of complex additions are: 24.0

Total number of complex multiplications are: 12.0

Exp no 6.

Aim To implement image negation, gray level slicing and thresholding

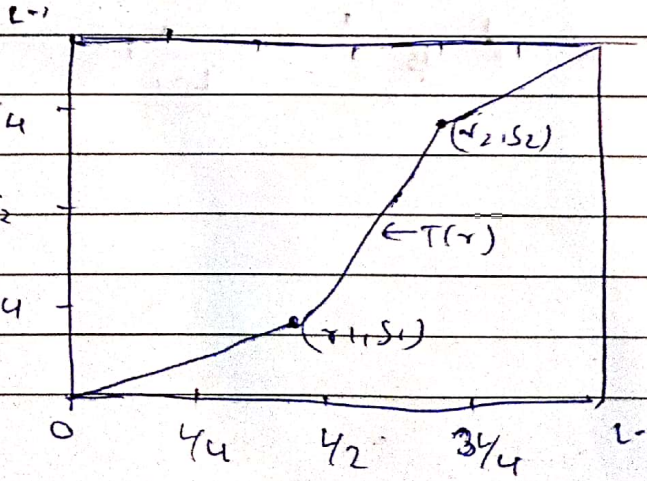
Theory Point processing techniques are among the simplest of all image enhancement techniques

1 Negation of an image

The simple operation in image processing is to compute the negation of an image It can be done by reversing the pixel values from black to white to black. Intensity of output image decreases as intensity of input image increases.

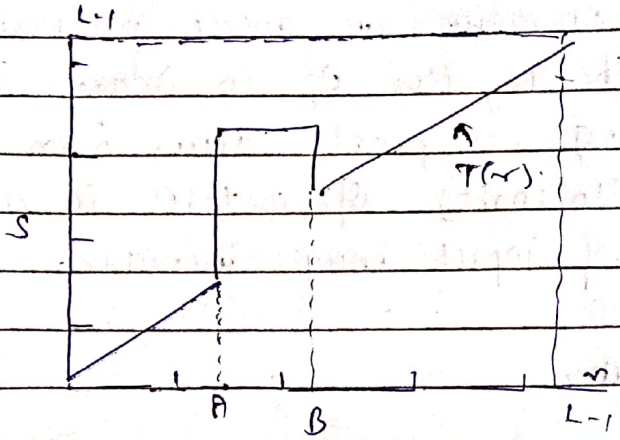
2) thresholding

It is a process of extracting a part of an image which contains information. In this transformation the threshold level is set and pixel values below threshold level is to be taken as 0 and above values are taken as 255. Intermediate values of r_1 r_2 s_1 and s_2 produce various degrees of gray levels and output which affects the contrast as shown in fig.

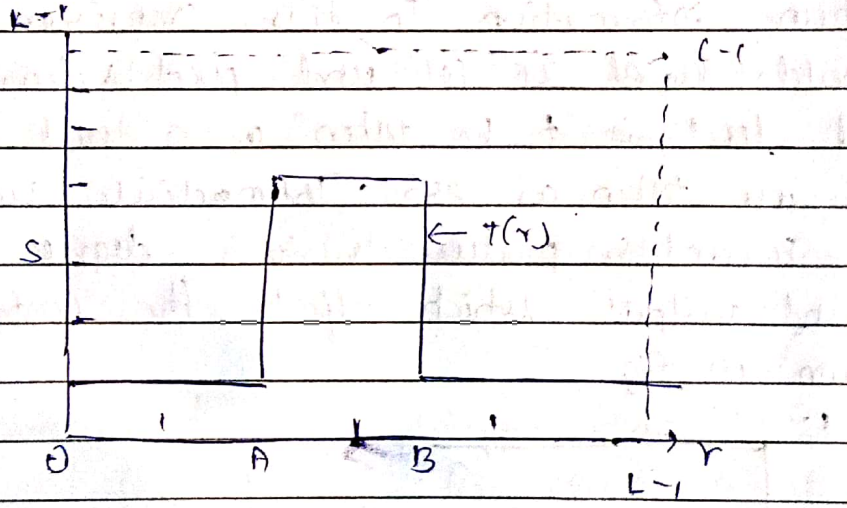


Gray level slicing with Background:

It is equivalent to band pass filtering. It manipulates groups of intensity levels in an image upto specific range by diminishing rest or by leaving them alone. This transformation is applicable in medical images and satellite images such as X ray, flows, CT scan. Two diff approaches are adopted for gray level scaling



4 Gray level slicing without bg:



#Program

```
clc
clear all
img = imread('C:\Users\Gaurav Poojary\OneDrive\Desktop\Trupti\gp.jpg');
a1=rgb2gray(img);
[row col]=size(a1);
a=double(a1);

c=255;
b=c-a;
figure(1)
imshow(img);
figure(2)
imshow(a1);
title("Grayscale Image");
figure(3)
imshow(uint8(b));
title("Digital Negative");

T=input("Enter the threshold value:");
for i=1:1:row
  for j=1:1:col
    if(a1(i,j)<T)
       a1(i,j)=0;
    else
       a1(i,j)=255;
    endif
  endfor
endfor
figure(4)
imshow(a1);
```

```
title("Threshold image");
V1=input("Enter the threshold value1:");
V2=input("Enter the threshold value2:");
for i=1:1:row
  for j=1:1:col
    if(a1(i,j)>V1 && a1(i,j)<V2)
      a2(i,j)=255;
    else
      a2(i,j)=0;
    endif
  endfor
endfor
for i=1:1:row
  for j=1:1:col
    if(a1(i,j)>V1 && a1(i,j)<V2)
      a3(i,j)=255;
    else
      a3(i,j)=a1(i,j);
    endif
  endfor
endfor
figure(5)
imshow(a2);
title("Gray level slicing without Background");
figure(6)
imshow(a3);
title("Gray level slicing with Background");
```

Command Window

Enter the threshold value:150

Enter the threshold value1:135

Enter the threshold value2:205

>>

Fig 1

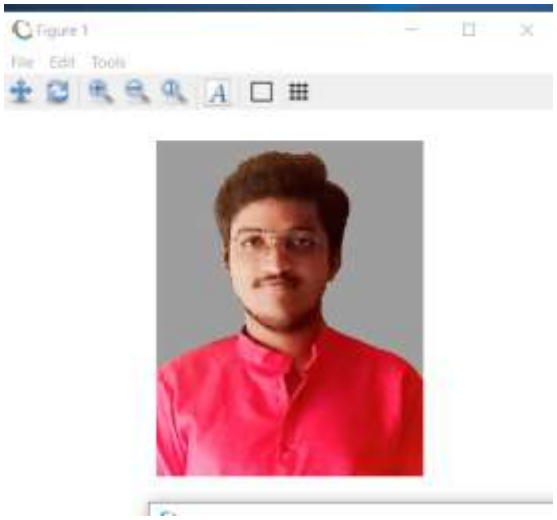


Fig 2

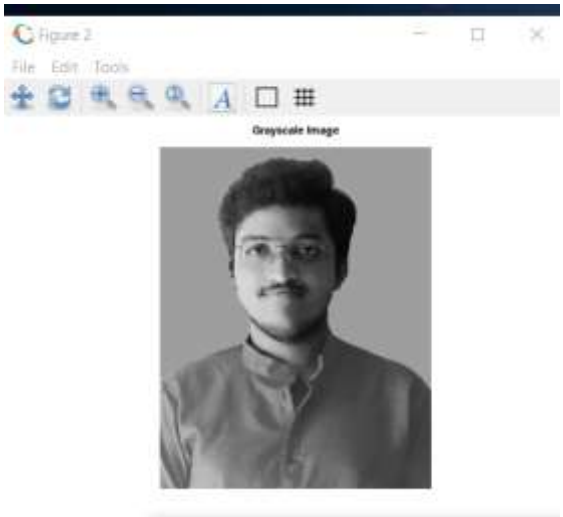



Figure 6
(94.996, 500.64
File Edit Tools
Gray level slicing with Background

Fig 6

Fig 5

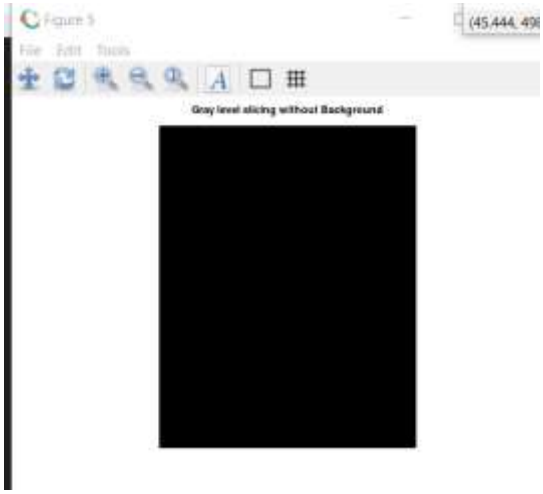


Figure 5
File Edit Tools
Gray level slicing without Background

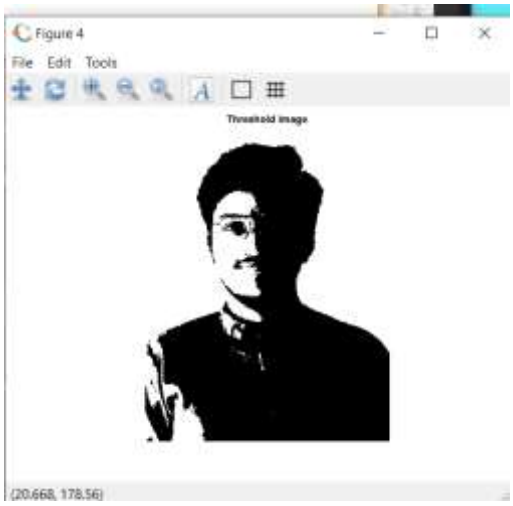


Figure 4
File Edit Tools
Threshold Image
(20.668, 178.56)

Fig 4

Fig 3

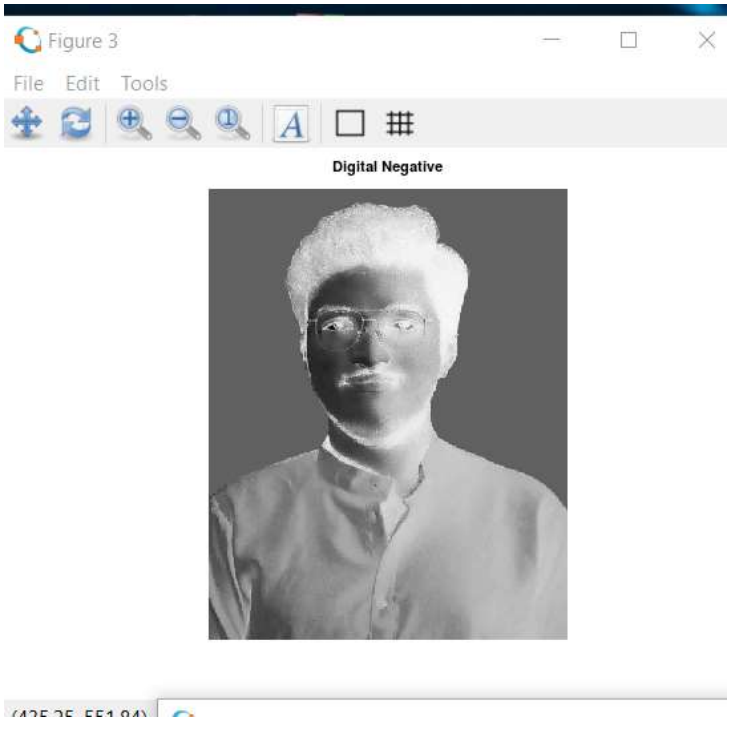


Figure 3
File Edit Tools
Digital Negative

DATE: / /

Experiment no - 07.

Aim: To implement contrast stretching dynamic Range Compression and Bit plane Siling

Theory:

1. contrast stretching

Contrast streching is used to increase the dynamic range of grey level in an image. It is required due to poor contrast. Poor contrast normally occurs due to: poor or non-uniform illumination non-linear dynamic range in an image sensor, wrong setting of lens aperture This technique simply increase the contrast of an image by making dark regions darker and brighter regions brighter. The linear normalization of a gray scale digital image is performed according to the formula.

$$In = (I - min) \frac{newMax - newMin}{max - min} + newMin$$

Example.

Assume: If newMax = 100,

newMin = 200

DATE: / /

10	192	107
112	56	205
82	250	142

100	145	140
		5
142	119	181
130	200	155

2. Dynamic Range compression:

Dynamic range in photography describes the ratio between the maximum and minimum measurable high intensities (white and black respectively). In the real world, one never encounters true white or black only varying degrees of light source intensity and subject reflectivity. Therefore the concept of dynamic range becomes more complicated and depends on whether you are describing a capture device (such as a camera or a scanner), a display device (such as a camera or a computer display) or the subject itself. One aim of HDR is to present a similar range of luminance to that experienced through the human visual system. The human eye through non-linear response adaption of the iris and other methods, adjusts constantly to a environment. The brain continuously interprets this information so that a viewer can see in a wide range of light conditions.

PAGE NO:
DATE: / /

Using $s = c\log(r+1)$

Example:

Assume: $c = 2$

Just dividing by 255, applying the transform then multiplying by 255 to get,

10	192	109
112	56	205
82	250	142

19	236	178
185	101	300
142	348	225

3. Bit plane slicing:

A bit-plane of a digital discrete signal such as an image or sound, is a set of bits corresponding to a given bit position in each of the binary numbers representation the signal. For example for 16-bit data representation there are 16 bit planes. The first bit plane contains the set of most significant bit and the 16th contains the least significant bit.

It is possible to see that the first bit plane gives the roughest but the most critical approximation of values of a medium, and the higher the number

of the bit plane the less is its contribution to the final stage. Thus adding a bit plane gives a better approximation.

Digital steganography makes use of bit planes and the phenomena that too complicated visual patterns cannot be perceived as "shape-information". It replaces complex areas on the bit plane of the vessel image with other complex data patterns.

example.

Assume: Message to hide = "A"
Binary of "A" = 01000001
so the 2nd and 8th pixel values are increased. Why? As their last bit plane was 0

10	192	107
112	56	205
187	240	142

10	193	107
112	56	205
187	251	142

Code

Bitplane

```
img = imread('C:\Users\Gaurav Poojary\OneDrive\Desktop\Trupti\gp.jpg');
img = rgb2gray(img);
dimg = double(img);
s= 1;
z = mod(dimg,2);
z = 0;
for i=1:8
bit_plane = mod(floor(dimg/(2^(i-1))),2);
% Right Shift by One Bit(Floor to make Binary Image )
subplot(2,4,i);
imshow(bit_plane);
title(['Bit Plane ',num2str(s)]);
s = s+1;
z = z+bit_plane*(2^(i-1));
end
figure;
imshow(img);
title('Original Image');
```

Contrast

```
img = imread('C:\Users\Gaurav Poojary\OneDrive\Desktop\Trupti\gp.jpg');
gray_img = rgb2gray(img);
MP = 255;
a = min(gray_img(:));
b = max(gray_img(:));
R = b-a;
scale = R/MP;
p = (gray_img - a) .* scale;
p = round(p);
```

```
#{
h1 = hist(uint8(gray_img));
figure;
plot(h1)
h2 = hist(uint8(p));
figure;
plot(h2)
#}
figure;
imshow(uint8(gray_img));
title("Original image");
figure;
imshow(uint8(p));
title("Contrast Stretched image");
```

Dynamic Range

```
clc
clear all
img = imread('C:\Users\Gaurav Poojary\OneDrive\Desktop\Trupti\gp.jpg');
img2 = rgb2gray(img);
figure(1)
imshow(img);
title('Orginal Image');
x = 150/255;
y = 200/255;
[n,m] = size(img2);
for i=1:n
  for j=1:m
  img2(i,j)=img2(i,j)/255;
  end
```

```
end
for i=1:n
  for j=1:m
    if(img2(i,j)>x && img2(i,j)<y)
    img2(i,j) = img2(i,j)-(30/255);
  end
end
end
for i=1:n
  for j=1:m
  img2(i,j)=img2(i,j)*255;
  end
end
img2=floor(img2)
img2=uint8(img2)
figure(2)
imshow(img2);
title('After Enhancement');
```

Output

Bit plane

Contrast

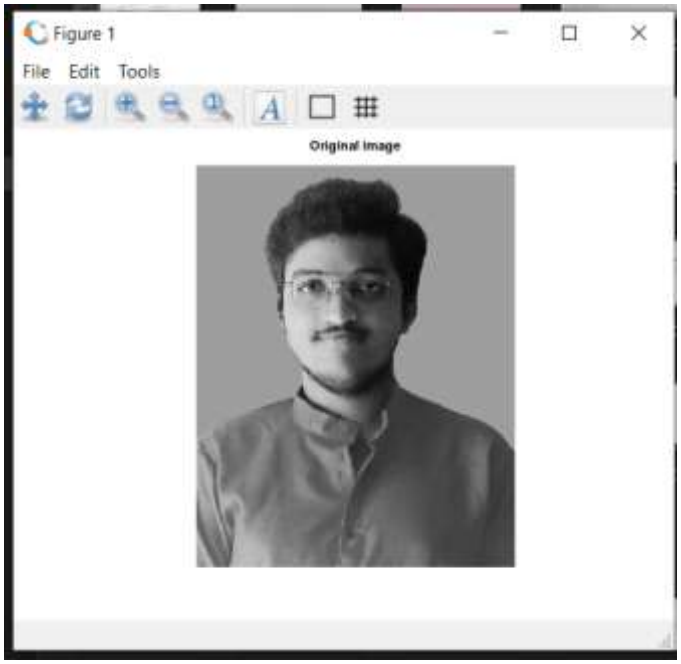
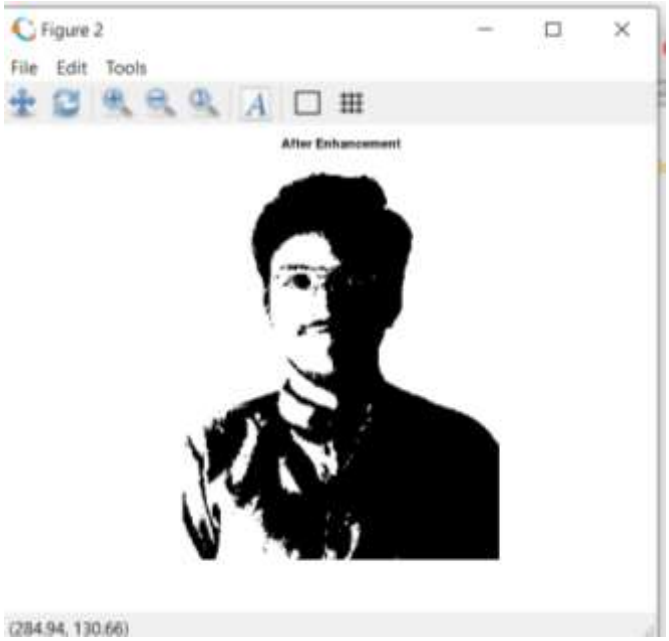


Figure 1
File Edit Tools
Original Image


Figure 2
File Edit Tools
Contrast Stretched Image
(246.95, 416.4)

Dynamic range



72 - 85

Experiment 8

Theory :

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram. Histogram equalization is one of the best methods for image enhancement. It provides better quality of image without loss of any information.

This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out of the most frequent intensity values.

This method is useful in images with bgs and foregrounds that are both bright or both dark. In particular, the method can lead to better views of bone structure in x-ray images and to better detail in photographs that are over or under exposed.

Consider a discrete grayscale image $f(x)$ and let n_i be the no of occurrences of gray level i.

The probability of an occurrence of a pixel of level i in image is

$$p_x(i) = p(x=i) = \frac{n_i}{N} \quad 0 \le i \le L$$

L being the no of gray levels in the image, M being the total no of pixels in the image and $p_x(i)$ being its first the image's histogram for pixel value i, normalized to [0,1].

$$cdf_x(i) = \sum_{j=0}^{i} p_x(x=j)$$

This is also the image's cumulated normalized histogram. The general histogram equalization formula is:

$$h(v) = round\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

Eg: L = 256

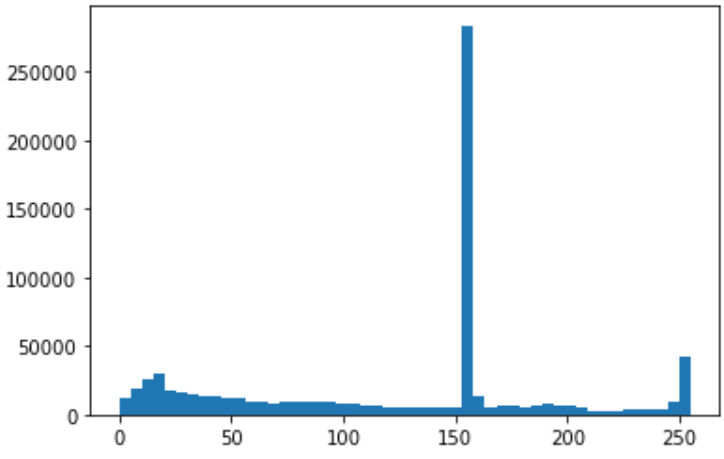
I/P image

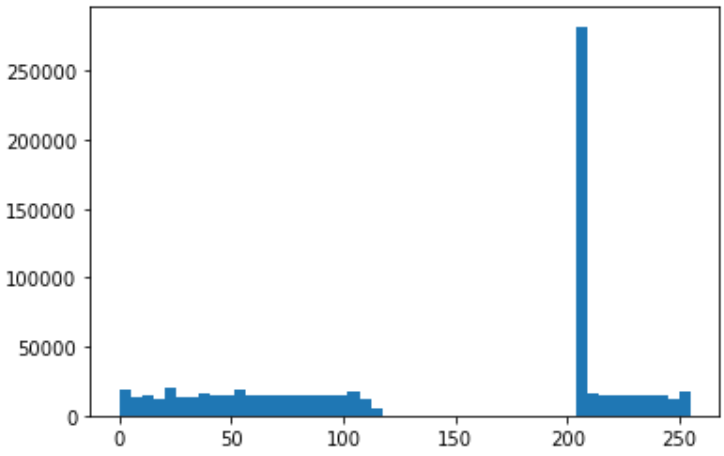
10	192	107
112	56	205
82	250	142

Pixel value	count	PDF	CDF	h(v)
10	1	1/9	1/9	28
192	1	1/9	2/9	57
107	1	1/9	3/9	85
112	1	1/9	4/9	113
56	1	1/9	5/9	142
205	1	1/9	6/9	170
82	1	1/9	7/9	198
250	1	1/9	8/9	227
142	1	1/9	9/9	255

Output

Histograms





CS22-85

Experiment no 9

Aim: Implementation of image smoothening and sharpening

Theory :-

In image processing a kernel convolution matrix is a small matrix used for blurring, processing and sharpening, edge detection etc.

(i) Image smoothing: It is often used to reduce the noise within an image or to produce a less pixelated image.

→ Mean smoothing methods are based on low pass filters.

(ii)

→ To perform this we consider adjacent pixels and the following kernel is used:

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

(iii) Image sharpening: It is an effect applied to digital images to give the image sharper appearance.

→ Image edges can be classified from at least a small amount of sharpening.

→ To perform a high pass filter the following kernel is used

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

→ At the edges of an image we mix pixel to form a neighbourhood.

→ Approaches to deal with it are:
* Omit missing pixels - only works with many filters and flow processing
* Pad the image - By with a black pixels.
* Replicate Border pixels.
* Truncate the image.
* Allow pixels wrap around the image.

5) Input image

10	192	107
112	56	205
82	250	142

→ Applying the smoothening kernel:

41	75	62
77	125	105
55	91	72

Output

Exp 10

Aim:- To implement sobel and Prewitt mask filter

Theory:- → Prewitt operator is used for edge detection in an image ie horizontal and vertical edges.

→ Edges are calculated by using the difference btw corresponding pixel intensities of image.

→ All the mask that are used for edge detection are also called derivation mask because the changes in a signal can only be calculated using differentiation.

→ All the mask should have the following:

(i) Opposite sign should be present in mask

(ii) Sum weight means no edge detection.

(iii) Sum of masks should be equal to 0

Vertical direction

-1	0	1
-1	0	1
-1	0	1

horizontal direction

-1	-1	-1
0	0	0
1	1	1

→ Sobel operator is similar to prewitt operator and is also a derivation mask used for edge detection

→ It can also be used for vertical or horizontal detection of edge.

→ The major diff b/w 2 is that in sobel operators the coefficient of masks are not fixed and they can be adjusted to our own requirement

unless they don't violate any property of derivative mask.

Vertical direction

-1	0	1
-2	0	2
-1	0	1

Horizontal direction

-1	-2	-1
0	0	0
1	2	1

Eg I/p image

10	192	107
112	56	205
82	250	102

→ Applying sobel Horizontal mask.

255	255	255
202	223	128
0	0	0

Program:

```
import numpy as np
import cv2


def showAndWrite(image, title, filename):
    cv2.imshow(title, image)
    cv2.waitKey(0)
    cv2.imwrite(filename, image)


def filter2D(src, kernel):


    def kMooreCoords(k, x, y, n, m):
        k1 = k // 2
        for i in range(-k1, k1 + 1):
            x1 = x + i
            if not -1 < x1 < n:
                x1 = -1
            for j in range(-k1, k1 + 1):
                y1 = y + j
                if not -1 < y1 < m:
                    y1 = -1
                yield (x1, y1)


    k = len(kernel)
    flatKernel = sum(kernel, [])
```

```
    n, m = src.shape
    dst = np.zeros(src.shape, dtype = np.uint8)
    for i in range(n):
        for j in range(m):
            pixel = 0
            for value, coord in zip(flatKernel, kMooreCoords(k, i, j, n, m)):
                if -1 not in coord:
                    pixel += value * src[coord[0], coord[1]]
            if pixel < 0:
                pixel = 0
            if pixel > 255:
                pixel = 255
            dst[i, j] = pixel
    return dst


kernels = {
    'sobelH': ([-1, -2, -1],
               [ 0,  0,  0],
               [ 1,  2,  1]),
    'sobelV': ([-1,  0,  1],
               [-2,  0,  2],
               [-1,  0,  1]),
    'prewittH': ([-1, -1, -1],
                 [ 0,  0,  0],
                 [ 1,  1,  1]),
```

```
        'prewittV': ([-1,    0,    1],
                     [-1,    0,    1],
                     [-1,    0,    1])
}

operations = {
        'sobelH': {
                'function': filter2D,
                'title': 'Sobel Horizontal Mask',
                'filename': 'DSIP_EXP_10_SH.jpg'
        },
        'sobelV': {
                'function': filter2D,
                'title': 'Sobel Vertical Mask',
                'filename': 'DSIP_EXP_10_SV.jpg'
        },
        'prewittH': {
                'function': filter2D,
                'title': 'Prewitt Horizontal Mask',
                'filename': 'DSIP_EXP_10_PH.jpg'
        },
        'prewittV': {
                'function': filter2D,
                'title': 'Prewitt Vertical Mask',
                'filename': 'DSIP_EXP_10_PV.jpg'
```

```
        }
}


image = cv2.resize(cv2.imread('arpit_original.jpeg', 0), (0, 0), fx = 0.5, fy = 0.5)
#     image = np.array([[10, 192, 107], [112, 56, 205], [82, 250, 142]], dtype = np.uint8)
cv2.imshow('Original', image)
cv2.waitKey(0)


for oper in operations:
        showAndWrite(
                operations[oper]['function'](image, kernels[oper]),
                operations[oper]['title'],
                operations[oper]['filename']
        )
```

Output

Sobel Horizontal:

Sobel Vertical:

Perwitt Horizontal:

Perwitt Vertical:

Assignment No: 1

Q.1) Define Signal, System & Signal processing

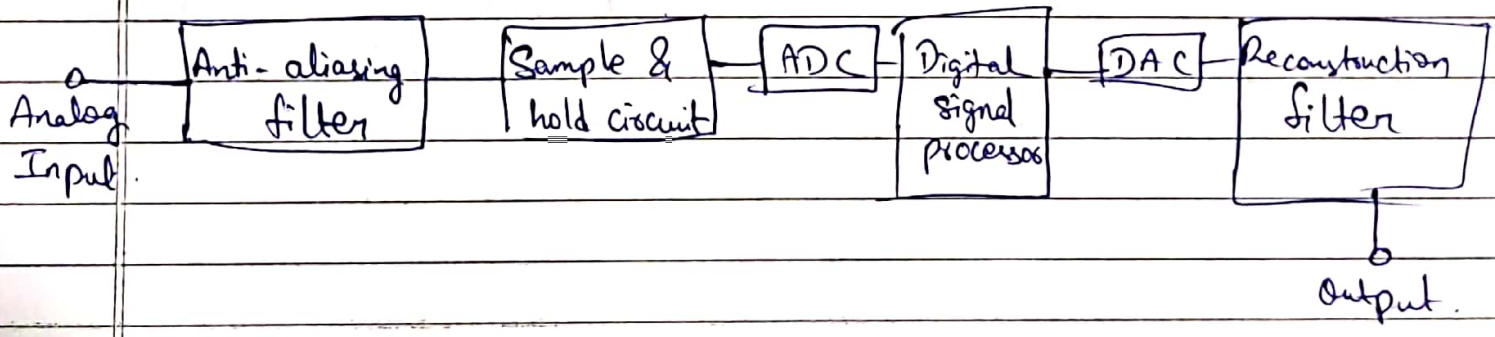
⇒ A signal is defined as any physical quantity that varies with time, space or any other independent variable. Anything that carries some information can be called signal.

- System is defined as a device or a combination of devices which produces signals & also performs operations on signals to get desired output.
- The production of a desired output signal from an input signal is known as signal processing.

Q.2) Explain digital signal processing system.

⇒ Digital signal processing system is the process of representing discrete mathematical sequence of numbers & analyzing, modifying & extracting the information contained in the signal by carrying out algorithmic operations & processing in the signal.

The basic block diagram is:



→ The input signal is analog from some transducer. It is denoted by $x(t)$.
→ The anti-aliasing filter is a low pass filter used to remove any high frequency & also removes the band-limit for the signal.
→ The sample & hold circuit takes samples of the signals & it keeps the voltage level of the input signal as constant which is required for ADC.
→ The ADC converts the signal from analog to digital.

Q.3) Compare Digital signal Processing with Analog Signal processing.

=>

Digital Signal Processing	Analog Signal processing
i) It is more accurate	It is less accurate.
ii) It is more flexible	It is less flexible.
iii) It can be easily replicated.	It is tedious to be replicated.
iv) Easily upgradable.	Difficult to upgrade.
v) It has a more complex system.	It has a less complex system.
vi) It requires more power consumption.	It requires less power consumption
vi) It is more versatile.	It is less versatile.

Q.4) Explain the applications of Digital Signal Processing.

⇒ Digital signal processing are used in a variety of applications some of them are:
- Speech recognition, synthesis & Analysis
- Image processing applications
- Robotic vision
- Biomedical applications like Analysing ECE, CT, etc.
- Industrial applications like robotics, CNN
- Military applications like Radar Signal processing & secure communications.
- Data Compression.

Q.5) Explain image file formats.

⇒ Images obtained from a camera are stored on the host in different file formats. It is a structure which defines how information is stored & how it is displayed to users.
- All formats contain 2 major things - a header & image data.
- The header gives us the information about the kind of image. It begins with binary code or ascii string.

1) BMP: The Bitmap Image file, device independent bitmap file format is a raster graphics image format to store the bitmap digital images independent of device. Invented by Microsoft, it is a loss less format but requires large space for high quality file images.

ii) JPEG: It is a commonly used method for compression for digital images, particularly used for photography uses.
The degree of compression can be adjusted allowing a tradeoff between storage size & image quality.

iii) PNG: It stands for Portable Network Graphics.
They are ideal for internet use.
They are compressed in a loss less format & thus retain all detail.
They are allow partial & total transparency thus allowing overlays & loges.

Assignment No: 2

Q Write a note on Image zooming by replication & interpolation.

=> i) Linear Interpolation:.

In this method, instead of replicating each pixel, average of the two adjacent pixels along the rows is taken & placed between the two pixels. The same operation is then performed in along the columns.

Interpolation along rows is given as.

$$V_1(m, 2n) = u(m, n); \quad 0 \le m \le M-1$$

$$V_1(m, 2n+1) = \frac{1}{2}[u(m,n) + v(m, n+1)]; \quad 0 \le m \le M-1, \quad 0 \le n \le N-1$$

For eg:

1	0	2	0	3	0	4	0
0	0	0	0	0	0	0	0
5	0	6	0	7	0	8	0
0	0	0	0	0	0	0	0
9	0	8	0	6	0	7	0
0	0	0	0	0	0	0	0
0	0	1	0	2	0	3	0
0	0	0	0	0	0	0	0

=> modified image

1	1.5	2	2.5	3	3.5	4	2
0	0	0	0	0	0	0	0
5	5.5	6	6.5	7	7.5	8	4
0	0	0	0	0	0	0	0
9	8.5	8	7	6	6.5	7	3.5
0	0	0	0	0	0	0	0
0	0.5	1	1.5	2	2.5	3	1.5
0	0	0	0	0	0	0	0

We now proceed to find the average values along the columns.

1	1.5	2	2.5	3	3.5	4	2
3	3.5	4	4.5	5	5.5	6	3
5	5.5	6	6.5	7	7.5	8	4
7	7	7	6.25	6.5	7	7.5	3.75
9	8.5	8	7	6	6.5	7	3.5
4.5	4.5	4.5	4.25	4	4.5	5	2.5
0	0.5	1	1.5	2	2.5	3	1.5
0	0.25	0.5	0.75	1	1.25	1.5	0.75

If we compare the results of replication & interpolation it is clear that the patchiness that was present in the replicated image is much less in the interpolated image. Hence we can conclude that zooming by interpolation is more effective than zooming by replication. To implement zooming by interpolation, we simply run an interpolation mask on the zero interlaced image.

1/4	1/2	1/4
1/2	1	1/2
1/4	1/2	1/4

ii) Replication:

In replication, we simply replicate each pixel an then replicate each row. Consider the pseudo-image shown below

1	2	3	4
5	6	7	8
9	8	6	7
0	1	2	3

We start from the first row. We replicate each pixel & then replicate each row.

The first row now looks like:

1 1 2 2 3 3 4 4

We now replicate this row to get.

1 1 2 2 3 3 4 4
1 1 2 2 3 3 4 4

performing operation on the entire image we get,

1	1	2	2	3	3	4	4
1	1	2	2	3	3	4	4
5	5	6	6	7	7	8	8
5	5	6	6	7	7	8	8
9	9	8	8	6	6	7	7
9	9	8	8	6	6	7	7
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3

Hence, a 4x4 image is zoomed to a 8x8 image. This method can be repeated to get bigger images. Remember, this is an image enhancement technique & hence no new data is added. In the zoomed pseudo-image, we observe that as we increase the size of the image, clusters of grey levels are formed which are discernible to the observer. The first step is to interlace the original image with zeroes. This is known as zero interlacing. In this we add zeroes after every pixel & then add a complete row of zeroes. Adding zeroes to every other pixel of the first row, we get.

1 0 2 0 3 0 4 0

This is also known as zero interlacing the columns as we add zero at every other column. Now inserting a row full of zeroes gives us. This is known as zero interlacing the rows as we add zeroes at every other row.

1	0	2	0	3	0	4	0
0	0	0	0	0	0	0	0
5	0	6	0	7	0	8	0
0	0	0	0	0	0	0	0
9	0	8	0	6	0	7	0
0	0	0	0	0	0	0	0
0	0	1	0	2	0	3	0
0	0	0	0	0	0	0	0

This image is known as the zero interlaced image. On this image we run a replication mask given below

1	1
1	1

The final zoomed image is.

1	1	2	2	3	3	4	4
1	1	2	2	3	3	4	4
5	5	6	6	7	7	8	8
5	5	6	6	7	7	8	8
9	9	8	8	6	6	7	7
9	9	8	8	6	6	7	7
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3

To get above result, we need to add a row of zeroes at the top & a column of zeroes at the beginning of the zeroes interlaced image making it 9×9.

Replication by zooming = Interlacing the rows → Interlacing the columns —convolve→ Mask

Q Compare Smoothening & sharpening filters.

=>

Smoothening filters	Sharpening filters.
-> Smoothening filter is used for blurring & noise reduction	Sharpening filter is used for removing blurring & highlighting the edges.
eg: low pass filter	eg: High pass filter.
-> It attenuates the high frequency	It attenuates low frequency.
-> low frequency is preserved in it.	High frequency is preserved
-> It allows the frequency below cut-off frequency to pass through it.	It allows frequencies above cut off frequency to pass through it.
-> It helps in removal of aliasing effect.	It helps in removal of noise.

Q Explain segmentation based on discontinuities

=> 1) Point Detection:-

Points, lines & edges are all high frequencies components & hence.. we need masks which are basically high pass. Hence the masks that we present here for point, line & edge detection have the properties of a high mask mainly, the sum of the coefficients of the mask has to be equal to zero in all the three cases.

-1	-1	-1
-1	8	-1
-1	-1	-1

$|R| \geq T$

where R is derived from the standard convolution formula:

$$R = w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9$$

$$R = \sum_{i=1}^{9} w_i z_i$$

We take |R| because we want to detect both the kinds of points on a black background as well as black points on a white background. T is a non-negative threshold which is defined by the user.

2) Line Detection:

Detection of lines can be done using the masks shown below. In an image, lines can be in any direction & detecting these lines would need different masks.

-1	-1	-1
2	2	2
-1	-1	-1

horizontal

-1	-1	2
-1	2	-1
2	-1	-1

+45°

-1	2	-1
-1	2	-1
-1	2	-1

vertical

2	-1	-1
-1	2	-1
-1	-1	2

-45°

Hence, we notice that all these masks have a total sum equal to zero, therefore they are all high pass masks. The first mask corresponds strongly to lines that are oriented horizontally. The second mask would correspond to lines at an angle of 45°. the third mask would respond strongly to lines that are vertical & the last to the lines that are at -45° angle.

3) Edge Detection:-

More than isolated points & lines, it is the detection of edges that form an important part of image segmentation. An edge can be defined as a set of connected pixel that form a boundary between two disjoint regions.

Here, the set of pixels that separate shaded region from white region is called as edge. The figure represents a step edge. Images obtained from digitization of optical images do not posses step edges.
Real world signals are not band-limited i.e. $F(v,v) \neq 0$ outside an internal & maximum frequency present in it is infinite. Hence, Nyquist criteria is not met.

It is this anti aliasing filter that reduces slope of the edge. Due to this, real world images when captured through a camera, will not have step edges.

In practice, edges are modelled as having a ramp like profile.
Here, the slope of ramp is inversely proportional to the degree of blurring. Given below are some of the two ~~Difficult~~ dimensional discrete domain edge model.

a	a	a	a	a	b	b	b	b	b
a	a	a	a	a	b	b	b	b	b
a	a	a	a	a	b	b	b	b	b
a	a	a	a	a	b	b	b	b	b
a	a	a	a	a	b	b	b	b	b

Vertical step edge

a	a	a	a	b	b	b	b	b	b
a	a	a	a	a	b	b	b	b	b
a	a	a	a	a	b	b	b	b	b
a	a	a	a	a	a	b	b	b	b
a	a	a	a	a	a	a	b	b	b

Diagonal step edge

Q $x_1(n) = \{1, 2, 3, 1\}$ $x_2(n) = \{1, 1, 1\}$.

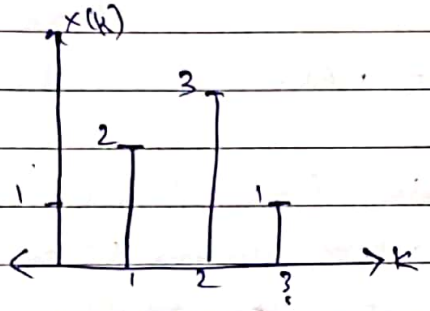
=> Graphical method

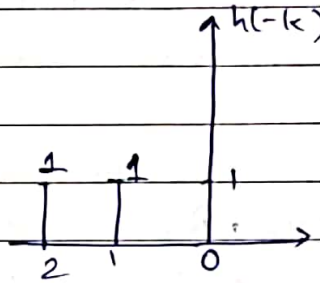
let $x(n) = x_1(n)$, $h(n) = x_2(n)$

linear convolution = $y(n)$ Range: $x_1 = 0$, $x_n = 3$, $h_1 = 0$, $h_n = 2$

$\therefore y(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$ $y_1 = x_1 + h_1 = 0$, $y_n = 5$

we draw $x(k)$ & $h(-k)$ & obtain $y(0)$

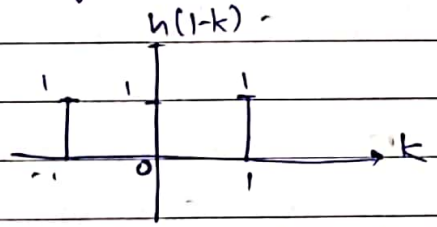




$y(0) = (1)(1) = 1$

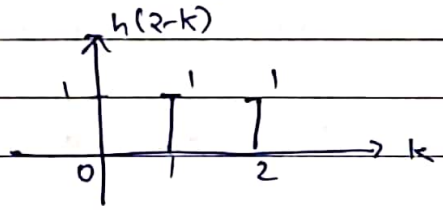
Shifting right:

Obtaining $y(1)$



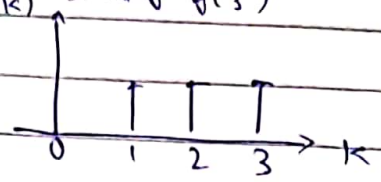
$y(1) = (1)(1) + (1)(2) = 3$

Obtaining $y(2)$



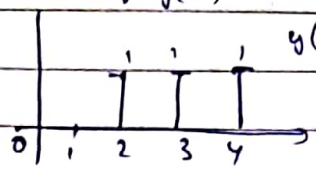
$y(2) = (1)(1) + (1)(2) + (1)(3) = 6$

h(3-k) Obtaining y(3)



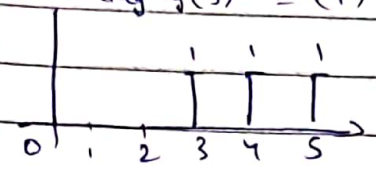
$y(3) = (1)(2)+(1)(3)+(1)(1)$
$= 6$

Obtaining y(1)



$y(4) = (1)(3)+(1)(1)$
$= 4.$

Obtaining $y(5) = (1)(1) = 1$



For y(6), there is no overlap so we will stop shifting right.
For y(-1) also there is no overlap from y(0) we will not shift left.

$\therefore y(n) = \{1, 3, 6, 6, 4, 1\}$

Formula method: length of $y(n) = 6$
length of $x(n) = 4$, length of $h(n) = 3$
$\therefore$ as per formula method:
length of $y(n)$ = length of $x(n)$ + length of $h(n) - 1 = 4+3-1 = 6$

$\Sigma x(n) = 1+2+3+1 = 7$
$\Sigma h(n) = 1+1+1 = 3$
$\Sigma x(n) \times \Sigma h(n) = 7 \times 3 = 21$
$\because \Sigma y(n) = \Sigma x(n) \times \Sigma h(n)$
$\Sigma y(n) = 1+3+6+6+4+1 = 21$
hence obtained result is correct.

2) $x(k) = \{10, -2, 0, 2\}$. Compute the energy of signal $(x(n))$

→ From Parseval's Energy theorem, we have

$$E = \sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|.$$

$$\therefore E = \frac{1}{4}\left\{ |x(0)|^2 + |x(1)|^2 + |x(2)|^2 + |x(3)|^2 \right\}$$

$$= \frac{1}{4}\{100 + 4 + 0 + 4\}$$

$$E = 27$$

Q Since the length of signal is 4,

$N = 4 \rightarrow$ 4-point DFT.

$\therefore$ 4x4, twiddle factor $W_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$

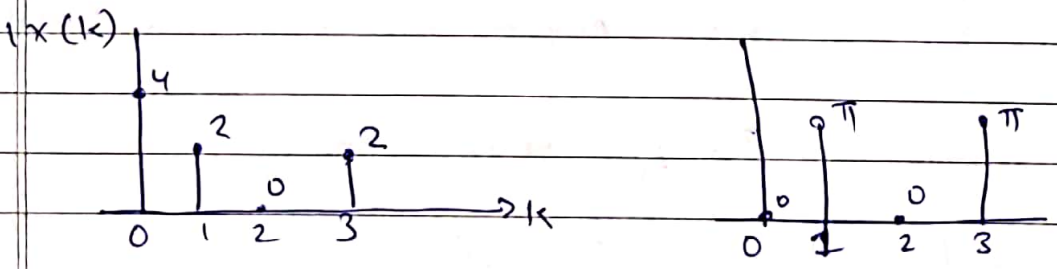
$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

$$= \begin{bmatrix} (1)0 + (1)(1) + (1)2 + 1(1) \\ (1)0 + (-j)1 + (-1)2 + (j)1 \\ (1)0 + (-1)1 + (1)2 + (-1)1 \\ (1)0 + (j)1 + (-1)2 + (-j)1 \end{bmatrix} = \begin{bmatrix} 4 \\ -2 \\ 0 \\ -2 \end{bmatrix}$$

$x(k) = \{4, -2, 0, -2\} = \{4\angle 0, 2\angle \pi, 0\angle 0, 2\angle \pi\}$.

$|X(k)| = \{4, 2, 0, 2\}$

$\angle x(k)| = \{0, \pi, 0, \pi\}$.



Magnitude spectrum Phase spectrum.