

Department of Information and Communication Engineering



Course Code: *ICE-3208*

Course Title: Digital Image and Speech Processing Sessional.

Submitted by:

Name: MD RAHATUL RABBI

Roll: 190609

Session: 2018-2019

3rd year 2nd Semester

Department of ICE,

Pabna university of Science and Technology

Submitted to:

Dr. Md. Imran Hossain

Associate Professor

Department of ICE,

Pabna university of Science and Technology

Teacher's Signature

INDEX

S\L	Experiment Name	Page
01	Display Following Image Operation In MATLAB/Python– i) Histogram Image. ii) Low Pass Filter Image. iii) High Pass Image.	02-04
02	Write A MATLAB/Python Program To Read ‘rice.tif’ Image, Count Number Of Rice And Display Area (Also Specific Range), Major Axis Length, And Perimeter.	05-06
03	Write A MATLAB/Python Program To Read An Image And Perform Convolution With 3×3 Mask.	07-08
04	Write A MATLAB/Python Program To Read An Image And Perform Laplacian Filter Mask.	08-10
05	Write A MATLAB/Python Program To Identify Horizontal, Vertical Lines From An Image.	10-12
06	Write A MATLAB/Python Program To Character Segment Of An Image.	13-14
07	For The Given Image Perform Edge Detection Using Different Operators And Compare The Results.	15-16
08	Write A MATLAB/Python Program To Read Coins.png, Leveling All Coins And Display Area Of All Coins.	16-18
09	Display Following Image Operation In MATLAB/Python – i) Threshold Image ii) Power Enhance Contract Image iii) High Pass Image.	18-19
10	Perform Image Enhancement, Smoothing And Sharpening, In Spatial Domain Using Different Spatial Filters And Compare The Performances.	20-21
11	Write A MATLAB/Python Program To Separation Of Voiced/Un-Voiced/Silence Regions From A Speech Signal.	21-23
12	Write A MATLAB/Python Program And Plot Multilevel Speech Resolution.	23-24
13	Write A MATLAB/Python Program To Recognize Speech Signal.	25-26
14	Write A MATLAB/Python Program For Text-To-Speech Conversion And Record Speech Signal.	26-28

Experiment No: 01

Name of the Experiment: Display Following Image Operation in MATLAB/Python–

- i) Histogram Image. ii) Low Pass Filter Image. iii) High Pass Image.

Objectives:

- (i) To understand and analysis the Histogram Image Operation.
- (ii) To understand and analysis the Low Pass Filter Image Operation.
- (iii) To understand and analysis the High Pass Filter Image Operation

Theory: We will be using MATLAB for the implementation and analysis of these operations as-

Histogram Image: A histogram of an image represents the distribution of pixel intensities. It shows how many pixels in the image have a specific intensity value. In other words, the x-axis of the histogram represents the possible intensity values (ranging from 0 to 255 in an 8-bit grayscale image), and the y-axis represents the number of pixels with that intensity. Histograms provide valuable insights into the image's overall brightness and contrast, and they help in understanding the distribution of various tones in the image.

Low-Pass Filter Image: A low-pass filter is a technique used to reduce high-frequency noise or details in an image while preserving the low-frequency components, which typically represent smoother regions or gradual changes in intensity. Applying a low-pass filter to an image helps to blur or smooth out rapid intensity variations, resulting in a cleaner and less noisy appearance. It is commonly used for noise reduction and preprocessing before further analysis.

High-Pass Filter Image: A high-pass filter is used to enhance the edges, fine details, and high-frequency components in an image while suppressing the low-frequency variations. Applying a high-pass filter makes edges and details more prominent, which can be useful for tasks like edge detection or highlighting specific features in an image.

These image processing techniques play important roles in enhancing, analyzing, and manipulating images for various applications, from photography to medical imaging and computer vision.

Source Code:

i) Histogram Image

```
% ***** Histogram *****_
clc;
clear all;
close all;
%i1=input('image name with location:'); % Get Image from user
im=imread('fruit-2999796.jpg'); % Read source Image
gIm=rgb2gray(im); % Convert RGB to Gray scale image
subplot(2,1,1);
imshow(gIm); % Display Gray scale image
title('Source image');
```

```

sz=size(gIm);                % Image Dimension width by height
n=sz(1)*sz(2);               % Height * Width
for rk=0:2:255
    a=find(gIm==rk);         % Check the condition
    sz=size(a);
    nk=sz(1);                 % Assign the Height of checked image
    pk(rk+1)=nk/n;           % Calculate the Histogram value
end
subplot(2,1,2);
bar(pk);                      % Plot the Histogram
axis([0,256,0,0.01]);        % Axis set-up
title('Histogram');

```

ii) Low Pass Filter Image:

```

% ***** Low Pass Filter *****
getim=imread('fruit-2999796.jpg'); % Read source Image
inim=rgb2gray(getim);             % Convert RGB to Gray scale image
subplot(2,1,1);
imshow(getim);                    % Display Gray scale image
title('Input image');
exim=wextend(2,'zpd',getim,1);    % wextend(mode, boundary, data, extensionLength)
mask=[1 1 1 ; 1 1 1 ; 1 1 1];    % Box filter
[r,c]=size(exim);                 % Image Dimension width by height
for i=1:1:r-2
    for j=1:1:c-2
        subim=exim(i:i+2,j:j+2); % To extract a sub-matrix.
        ele_multi=mask.*double(subim);
        col_sum=sum(ele_multi);    % Column sum
        row_sum=col_sum(1,1)+col_sum(1,2)+col_sum(1,3); % Row sum = (col1+col2+col3)
        ave=row_sum/9;             % Calculate average
        outim(i,j)=ave;            % set Desired value
    end
end
lpassim=uint8(outim);             % Unsigned 8-bit integers, (0 to 255).
subplot(2,1,2),imshow(lpassim);
title('Low pass filtered image');

```

iii) High Pass Filter Image:

```

% ***** High Pass Filter *****
clc
clear all;
close all;
%i1=input('image name with location :'); % Get Image from user
inim=imread('fruit-2999796.jpg');      % Read source Image
inim=rgb2gray(inim);                   % Convert RGB to Gray scale image
subplot(2,1,1),imshow(inim);           % Display Gray scale image
title('Input Image');
exim=wextend(2,'zpd',inim,1);           % wextend(mode, boundary, data, extensionLength)
mask=[-1 -1 -1; -1 8 -1;-1 -1 -1];     % Laplacian kernel

```

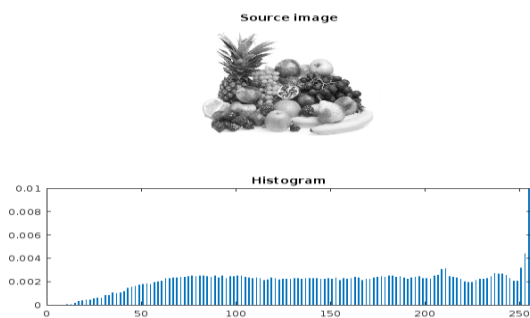
```

[r,c]=size(exim);                % Image Dimension width by height
for i=1:1:r-2
    for j=1:1:c-2
        N1=exim(i:i+2,j:j+2);    % To extract a sub-matrix.
        ele_multi=mask.*double(N1);
        col_sum=sum(ele_multi);    % Column sum
        row_sum=col_sum(1,1)+col_sum(1,2)+col_sum(1,3); %Row sum = (col1+col2+col3)
        ave=row_sum/9;            % Calculate average
        outim(i,j)=ave;          % set Desired value
    end
end
hpassim=uint8(outim);            % Unsigned 8-bit integers, (0 to 255).
subplot(2,1,2),imshow(hpassim);
title('High Pass Filtered Image');

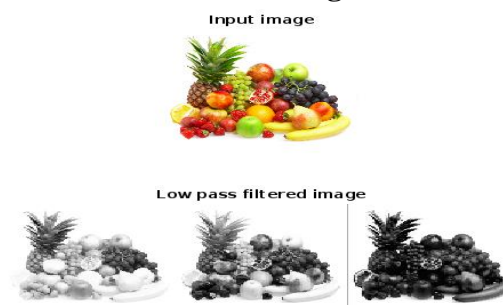
```

Output:

i) Histogram Image



ii) Low Pass Filter Image:



iii) High Pass Filter Image



Experiment No: 02

Name of the Experiment: Write A MATLAB/Python Program To Read 'Rice.Tif' Image, Count Number Of Rice And Display Area (Also Specific Range), Major Axis Length, And Perimeter.

Objectives:

- (i) To understand and analysis of image objects.
- (ii) To understand and analysis the area of an image objects.
- (iii) To understand and analysis about major axis length and perimeter.

Theory: We will be using MATLAB for the implementation and analysis of these operations as-

Reading the Image: The 'rice.tif' image is loaded using an image processing library like OpenCV or skimage. The image is represented as a matrix of pixel values, where each value corresponds to the intensity at a specific location.

Thresholding: Thresholding is applied to convert the image into a binary format, where pixels are either considered foreground (rice) or background.

Region Properties Analysis: For each labeled region (rice grain), various properties can be calculated such as-

- i) **Area:** The number of pixels in the region, indicating the size of the rice grain.
- ii) **Major Axis Length:** The length of the longest line that can be drawn within the region, representing the major dimension of the rice grain.
- iii) **Perimeter:** The total length of the boundary of the region, which gives an idea of the shape complexity.

By combining these steps in program, we can efficiently analyze the 'rice.tif' image, count rice grains, and present relevant information about their properties, aiding in various applications like quality assessment and yield estimation.

Source Code:

```
% Load the 'rice.tif' image
clc;
clear all;
close all;
riceImage = imread('ayTdk.jpg');          % Read source Image

% Convert the image to binary using a threshold
threshold = graythresh(riceImage);
binaryImage = imbinarize(riceImage, threshold);

% Perform morphological operations to clean up the binary image (remove noise)
binaryImage = imopen(binaryImage, strel('disk', 2));
binaryImage = bwareaopen(binaryImage, 100);    % removing small objects
[labels, numRice] = bwlabel(binaryImage);      % Label connected components (rice grains)

% Initialize measurement variables
areas = zeros(1, numRice);
```

```

majorAxisLengths = zeros(1, numRice);
perimeters = zeros(1, numRice);

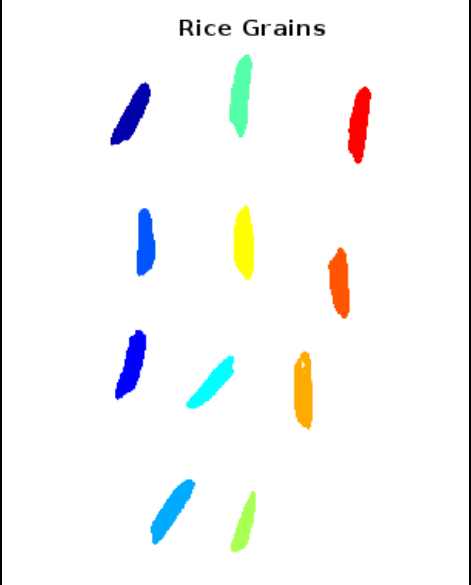
% Calculate measurements for each rice grain
for i = 1:numRice
    riceRegion = (labels == i);
    % Calculate properties of the rice grain
    riceStats = regionprops(riceRegion, 'Area', 'MajorAxisLength', 'Perimeter');
    % Store the measurements in arrays
    areas(i) = riceStats.Area;
    majorAxisLengths(i) = riceStats.MajorAxisLength;
    perimeters(i) = riceStats.Perimeter;
end

% Display the results
fprintf('Number of rice grains: %d\n', numRice);
fprintf('Rice grain measurements:\n');
fprintf('Grain #\tArea\tMajor Axis Length\tPerimeter\n');
for i = 1:numRice
    fprintf('%d\t%.2f\t%.2f\t%.2f\n', i, areas(i), majorAxisLengths(i), perimeters(i));
end

% Display the image with rice grains highlighted
RGB = label2rgb(labels);
imshow(RGB);
title('Rice Grains');

```

Output:

	Number of rice grains: 11			
	Rice grain measurements:			
	<u>Grain</u>	<u>Area</u>	<u>Major-Axis-Length</u>	<u>Perimeter</u>
	1	1533.00	87.33	187.06
	2	1589.00	90.45	194.68
	3	1365.00	86.43	180.37
	4	1587.00	91.93	193.63
	5	1310.00	82.42	174.97
	6	1752.00	103.27	214.55
	7	1054.00	78.83	164.46
	8	1642.00	89.45	191.03
	9	1692.00	96.73	203.28
	10	1517.00	85.95	184.05
	11	1617.00	96.84	203.02

Experiment No: 03

Name of the Experiment: Write A MATLAB/Python Program To Read An Image And Perform Convolution With 3×3 Mask.

Objectives:

- (i) To understand how to read an image in MATLAB.
- (ii) To understand and analysis the convolution process to an image.
- (iii) To understand about 3×3 Mask.

Theory: We will be using MATLAB for the implementation and analysis of these operations as-

Convolution: Convolution is a fundamental operation in image processing and computer vision. In the context of image processing, convolution is used to extract features, enhance details, or apply filters to an image that defined as-

$$g(x,y) = \sum_{s=-a}^a \sum_{t=-b}^b \omega(s,t) f(x-s,y-t)$$
$$g = \omega * f$$

3×3 Mask: A 3×3 mask is a 3 by 3 matrix used for convolution. Each element in the mask corresponds to a weight that determines the contribution of the corresponding pixel and its neighbors to the convolution result.

Convolution Process:

- i) To place the center of the 3×3 mask on the current pixel in the image.
- ii) To multiply the elements of the mask with the corresponding pixel values in the image.
- iii) To sum up the products to get the new value for the central pixel.
- iv) To move the mask to the next pixel and repeat the process until the entire image is covered.

By reading an image and performing convolution with a 3×3 mask, we can achieve various image enhancement or feature extraction operations that play a crucial role in image processing and computer vision tasks.

Source Code:

```
clc;
close all;
clear all;
image = imread('Untitled2.jpeg');    % Read source Image
image = double(image);              % Convert the image to double for convolution
mask = [1, 2, 1; 0, 0, 0; -1, -2, -1]; % Define the 3x3 mask (kernel)

% Separate the RGB channels
redChannel = image(:,:,1);          % To access the first color(Red) channel
greenChannel = image(:,:,2);        % To access the 2nd color(Green) channel
blueChannel = image(:,:,3);         % To access the 3rd color(Blue) channel
% Perform convolution on each channel
```



```

convolvedRed = conv2(redChannel, mask, 'same');
convolvedGreen = conv2(greenChannel, mask, 'same');
convolvedBlue = conv2(blueChannel, mask, 'same');

% Combine the convolved channels back into an RGB image
convolvedImage = cat(3, convolvedRed, convolvedGreen, convolvedBlue);

% Display the original
subplot(1, 2, 1);
imshow(uint8(image));           % Unsigned 8-bit integers, (0 to 255).
title('Original Image');

% Display convolved images
subplot(1, 2, 2);
imshow(uint8(convolvedImage));  % Display the convolved Image
title('Convolved Image');

```

Output:



Experiment No: 04

Name of the Experiment: Write A MATLAB/Python Program To Read An Image And Perform Laplacian Filter Mask.

Objectives:

- (i) To understand how to read an image in MATLAB.
- (ii) To understand and analysis the Laplacian Filter Mask to an image.
- (iii) To understand, how to perform convolution operation on Laplacian Filter Mask.

Theory: We will be using MATLAB for the implementation and analysis of these operations as-

Laplacian Filter: The Laplacian filter is a type of edge detection filter used in image processing. It highlights areas of rapid intensity changes (edges) in an image. It is commonly used for tasks such as edge detection, sharpening, and identifying high-frequency components in an image. Mathematically –

$$L(x, y) = \nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

The Laplacian filter mask typically looks like this for a 3x3 matrix as-

0	1	0
1	-4	1
0	1	0

Convolution: To apply a Laplacian filter mask to an image, we perform a convolution operation.

Procedure: To apply the Laplacian filter mask to an image, follow these steps:

Step-1: To convert the image to grayscale if it's not already.

Step-2: To iterate through each pixel position in the image, excluding the border pixels where filter might not fit entirely.

Step-3: For each pixel, take a small region of the image that matches the size of the Laplacian filter mask.

Step-4: Perform element-wise multiplication between the filter mask and the corresponding region of the image.

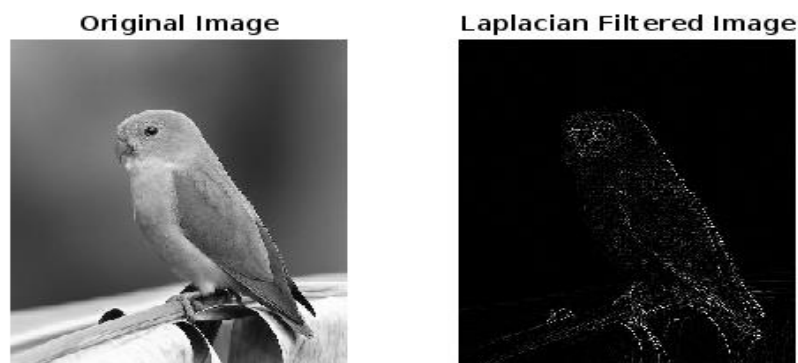
Step-5: To sum up the results of the element-wise multiplications.

Step-6: To place the sum in the output image at the corresponding pixel position.

Source Code:

```
image = imread('pexels-photo-1661179.jpeg');           % Read source Image
% Convert the image to grayscale if it's in color
if size(image, 3) == 3
    grayImage = rgb2gray(image);
else
    grayImage = image;
end
laplacianMask = [0, 1, 0; 1, -4, 1; 0, 1, 0];          % Define the Laplacian filter mask (kernel)
filteredImage = conv2(double(grayImage), laplacianMask, 'same'); % Perform convolution
subplot(1, 2, 1);
imshow(grayImage);                                     % Display the original Image'
title('Original Image');
subplot(1, 2, 2);
imshow(uint8(filteredImage));                           % Display the 'Laplacian Filtered Image'
title('Laplacian Filtered Image');
colormap gray;                                          % To a grayscale colormap
```

Output:



Experiment No: 05

Name of the Experiment: Write A MATLAB/Python Program To Identify Horizontal, Vertical Lines From An Image.

Objectives:

- (i) To Identify Horizontal Lines from an image using MATLAB.
- (ii) To Identify Vertical Lines from an image using MATLAB

Theory: Identifying horizontal and vertical lines from an image is a common task in computer vision and image processing. There are several approaches and techniques that can be used to achieve this goal. Here's a general overview of a possible approach:

i) **Image Preprocessing:**

- ✓ Convert the image to grayscale if it's in color.
- ✓ Apply any necessary noise reduction techniques, such as median filtering.
- ✓ Perform edge detection using techniques like Canny's edge detection.

ii) **Hough Transform:** The Hough Transform is a powerful technique for detecting lines in an image, including horizontal and vertical lines. The transform maps the image space to a parameter space where lines are represented in a different form.

iii) **Peak Detection:** Identify peaks in the Hough parameter space. These peaks represent the lines in the image. Set appropriate thresholds to determine what constitutes a valid peak.

iv) **Line Extraction:**

- ✓ Convert the peaks in the parameter space back to lines in the image space.
- ✓ **For horizontal lines**, the equation would be: $y = mx + b$, where m is close to 0 and b corresponds to the y-intercept.
- ✓ **For vertical lines**, the equation would be: $x = my + b$, where m is close to infinity and b corresponds to the x-intercept.

v) **Visualization:** Draw the detected lines on the original image to visualize the results.

Source Code:

```
clc;
clear;
close all;
getim=imread('LINE3.JPG'); % Read source Image
GI=rgb2gray(getim); % Convert RGB to Gray scale image
subplot(2,1,1),imshow(GI); % Display Gray scale image
title('Input image');

S1=GI;S2=GI;S3=GI;S4=GI; % Initialize the value of GI
M1=[-1 -1 -1;2 2 2;-1 -1 -1]; % Mask For Horizontal line
M2=[-1 -1 2;-1 2 -1;2 -1 -1]; % Mask For +45 degree line
M3=[-1 2 -1;-1 2 -1;1 2 -1]; % Mask For Vartical line
M4=[2 -1 -1;-1 2 -1;-1 -1 2]; % Mask For -45 degree line

%For output image
OutI=GI;
OutI(:,:)=0; % Set all the elements of a 2D matrix with 0

%Get size of Extented matrix
l=1;
Gtext = wextend(2,'zpd',GI,l); % wextend(mode, boundary, data, extensionLength)
Emat=size(Gtext); % Image Dimension width by height
Max_r=Emat(1); % Assign Width
Max_c=Emat(2); % Assign Height
%===== Detect Horinzontal Line =====
for i=1:1:Max_r-2
    for j=1:1:Max_c-2
        x=Gtext(i:i+2,j:j+2); % Get submatrix
        r=M1.*double(x);
        R=sum(sum(r)); % sumRow(sumColumn)
        S1(i,j)=uint8(R); % Unsigned 8-bit integers, (0 to 255).
    end
end
%===== Detect +45 Degree =====
OutI(:,:)=0;
for i=1:1:Max_r-2
    for j=1:1:Max_c-2
        x=Gtext(i:i+2,j:j+2); % Get submatrix
        r=M2.*double(x);
        R=sum(sum(r)); % sumRow(sumColumn)
        S2(i,j)=uint8(R); % Store the R values
    end
end
%===== Detect Vartical Line =====
OutI(:,:)=0;
for i=1:1:Max_r-2
    for j=1:1:Max_c-2
        x=Gtext(i:i+2,j:j+2); % Get submatrix
```

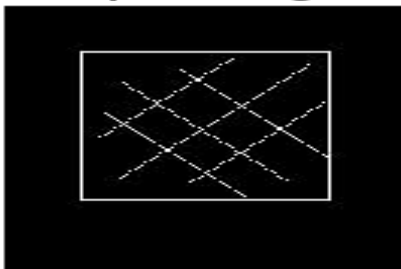
```

    r=M3.*double(x);
    R=sum(sum(r));
    S3(i,j)=uint8(R);    % Store the R values
end
end
%===== Detect -45 Degree Line =====
for i=1:1:Max_r-2
    for j=1:1:Max_c-2
        x=G1ext(i:i+2,j:j+2);    % Get submatrix
        r=M4.*double(x);
        R=sum(sum(r));
        S4(i,j)=uint8(R);    % Store the R values
    end
end
for i=1:1:Max_r-2
    for j=1:1:Max_c-2
        OutI(i,j)=max(max(S1(i,j),S2(i,j)),max(S3(i,j),S4(i,j)));    % Max calculate
    end
end
K=find(OutI>=250);    % find with condition
OutI(:)=0;
OutI(K)=255;
subplot(2,1,2);
imshow(OutI),title('Output image. '); % Display output image

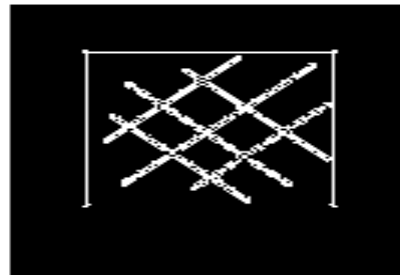
```

Output:

Input image



Output image.



Experiment No: 06

Name of the Experiment: Write A MATLAB/Python Program To Character Segment Of An Image.

Objectives:

- (i) To understand and analysis character segmentation of an image.
- (ii) To understand and analysis character normalization of an image.

Theory: Character segmentation is a crucial step in optical character recognition (OCR) and text recognition systems. Here's a general overview of the theory and process behind character segmentation:

i) **Preprocessing:**

- ✓ Convert the input image to grayscale.
- ✓ Apply noise reduction techniques like Gaussian blur or median filtering to reduce noise and enhance the quality of the text.

ii) **Segmentation Criteria:** Define criteria based on the properties of the connected components to determine whether they represent characters or noise/artifacts. Common criteria include size thresholds, aspect ratio ranges, and proximity to other components.

iii) **Segmentation:** Based on the criteria, separate the connected components into individual characters. This can be achieved using techniques like bounding box extraction, contour analysis, or region growing.

iv) **Character Normalization:** Resize the segmented characters to a consistent size to ensure uniformity for recognition.

v) **Feature Extraction:** Extract relevant features from the segmented characters, such as stroke width, histograms of pixel intensities, or gradient features. These features will be used as input to the OCR system for character recognition.

vi) **Character Recognition:** After segmentation and feature extraction, the recognized features are fed into a character recognition model (such as a learning classifier) to determine the corresponding characters.

Source Code:

```
imagen=imread('image_a.JPG');    % Read source Image
figure(1)
imshow(imagen);                  % Show source image with noise
title('Input Image with Noise')
if size(imagen,3)==3              % RGB image
    imagen=rgb2gray(imagen);      % Convert to gray scale
end
% Convert to binary image
threshold = graythresh(imagen);
imagen = ~im2bw(imagen,threshold);
% Remove all object containing fewer than 30 pixels
imagen = bwareaopen(imagen,30);
```

```

pause(1);figure(2)
imshow(~imagen); % Show image binary image without noise
title('Input Image without Noise');
[L Ne]=bwlabel(imagen); % Label connected components
propied=regionprops(L,'BoundingBox'); % Measure properties of image regions
hold on % To overlay multiple plots on the same set of axes
% Plot Bounding Box
for n=1:size(propied,1)
    rectangle('Position',propied(n).BoundingBox,'EdgeColor','g','LineWidth',2)
end
hold off;pause (1); % To pause the program for 1 second.
% Objects extraction
figure(3)
for n=1:Ne
    [r,c] = find(L==n);
    n1=imagen(min(r):max(r),min(c):max(c));
    imshow(~n1); % show individual character of an image
    pause(1) % To pause the program for 1 second.
end

```

Input:

Input Image with Noise	Input Image without Noise
JUDAS	JUDAS
PRIEST	PRIEST
87755789	87755789
HOLA	HOLA
DIEGO	DIEGO
123	123
1234567	1234567

Output:

12I5L9

Experiment No: 07

Name of the Experiment: For The Given Image Perform Edge Detection Using Different Operators And Compare The Results.

Objectives:

- (i) To understand and analysis the concept of edge detection in image processing.
- (ii) To understand and implement different edge detection operators on given image.

Theory: Edge detection is a fundamental technique in image processing that aims to identify boundaries or transitions between different regions in an image. There are several edge detection operators that can be used to achieve this goal. Here's an overview of the theory behind performing edge detection using different operators and comparing the results.

Sobel Operator: The Sobel operator is a gradient-based edge detection method that computes the gradient magnitude and direction of an image. It uses two convolution masks (kernels) to calculate the gradients in the horizontal and vertical directions. The gradient magnitude is calculated as the square root of the sum of squared gradients in both directions.

$$G[f(x, y)] = \sqrt{G_x^2 + G_y^2}$$

Canny Edge Detector: The Canny edge detector is a multi-stage edge detection algorithm that aims to find edges while suppressing noise. It involves steps like Gaussian blurring to reduce noise, gradient computation using convolution masks (often Sobel), and non-maximum suppression to thin the edges, and hysteresis thresholding to connect and detect final edges.

Laplacian of Gaussian (LoG) Operator: The LoG operator combines Gaussian blurring and the Laplacian operator to detect edges. It convolves the image with a Gaussian filter followed by the Laplacian filter to highlight edges as zero crossings in the second derivative. Mathematically LoG can be written as-

$$LoG(x, y) = -\frac{1}{\pi\sigma^4} \left[1 - \frac{x^2 + y^2}{2\sigma^2} \right] e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

Comparing Results: To compare the results of different edge detection operators- Apply each operator to the same input image and generate edge maps for each.

Source Code:

```
clc;clear;close all;
image = imread('LINE3.JPG');    % Read source Image
if size(image, 3) == 3          % RGB image
    grayImage = rgb2gray(image); % Convert to gray scale
else
    grayImage = image;
end
% Apply edge detection using different operators
sobelEdges = edge(grayImage, 'sobel'); % Sobel Operators
prewittEdges = edge(grayImage, 'prewitt'); % prewitt Operators
```



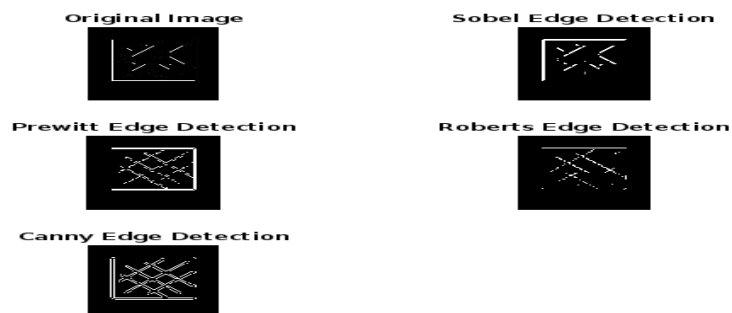
```

robertsEdges = edge(grayImage, 'roberts'); % roberts Operators
cannyEdges = edge(grayImage, 'canny');    % canny Operators

% Display the original image and the edge detection results side by side
subplot(3, 2, 1);
imshow(grayImage);                        % Display Original Image
title('Original Image');
subplot(3, 2, 2);
imshow(sobelEdges);                       % Display Sobel Edge Detection Image
title('Sobel Edge Detection');
subplot(3, 2, 3);
imshow(prewittEdges);                     % Display Prewitt Edge Detection Image
title('Prewitt Edge Detection');
subplot(3, 2, 4);
imshow(robertsEdges);                     % Display Roberts Edge Detection Image
title('Roberts Edge Detection');
subplot(3, 2, 5);
imshow(cannyEdges);                       % Display Canny Edge Detection Image
title('Canny Edge Detection');
colormap gray;                            % To a grayscale colormap

```

Output:



Experiment No: 08

Name of the Experiment: Write A MATLAB/Python Program To Read Coins.png, Leveling All Coins And Display Area Of All Coins.

Objectives:

- (i) To implement algorithms for leveling the coins in the image.
- (ii) To calculate and display the area of all coins in the leveled image

Theory: To read an image file named "Coins.png," extract information about the different coins, level all the coins, and calculate the total display area of all the coins in the image.

Here's a general step-by-step outline of how you might approach this task:

- **Image Reading:** Use a programming language like to read the "Coins.png" image.
- **Coin Detection:** Apply image processing techniques like edge detection, contour detection, or object detection to identify the coins in the image.
- **Coin Analysis:** For each detected coin, calculate its properties like radius, center coordinates, and area. This might involve using geometric measurements and calculations.
- **Coin Leveling:** To level the coins, you'll need to adjust their positions so that they appear as if they are lying flat on a surface.
- **Total Display Area Calculation:** Sum up the areas of all the leveled coins to calculate the total display area.

Source Code:

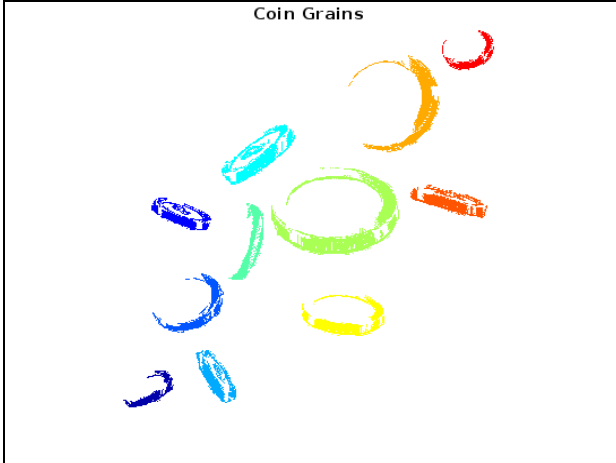
```
coinImage = imread('1.png'); % Read source Image
% Convert the image to binary using a threshold
threshold = graythresh(coinImage);
binaryImage = imbinarize(coinImage, threshold);

% Perform morphological operations to clean up the binary image (remove noise)
binaryImage = imopen(binaryImage, strel('arbitrary', 1)); % strel(shape, parameters); like 'square',
'disk'
binaryImage = bwareaopen(binaryImage, 120); % removing small objects as
bwareaopen(inputImage, minArea);
[labels, numCoin] = bwlabel(binaryImage); % Label connected components (coin grains)
areas = zeros(1, numCoin); % Initialize measurement variables

% Calculate measurements for each Coin grain
for i = 1:numCoin
    riceRegion = (labels == i);
    riceStats = regionprops(riceRegion, 'Area'); % Store the measurements in arrays
    areas(i) = riceStats.Area; % Calculate area
end

% Display the results
fprintf('Number of Coin grains: %d\n', numCoin); % Print number of coin in console
fprintf('Coin Area measurements:\n');
fprintf('Grain \tArea\n');
for i = 1:numCoin
    fprintf('%d\t%.2f\n', i, areas(i)); % Print Area in console
end
RGB = label2rgb(labels);
imshow(RGB); % Display Coin grains highlighted
title('Coin Grains');
```

Output:

	
Number of Coin grains: 11	
Coin Area measurements:	
<u>Grain</u>	<u>Area</u>
1	428.00
2	680.00
3	779.00
4	708.00
5	1265.00
6	748.00
7	2672.00
8	1006.00
9	1459.00
10	909.00
11	442.00

Experiment No: 09

Name of the Experiment: Display following image operation in MATLAB/Python –

- i) Threshold image ii) Power enhance contract image iii) High pass image

Objectives:

- To understand the concept of threshold image.
- To understand power enhance contract image and high pass image.

Theory: We will be using MATLAB for the implementation and analysis of these operations as-

Thresholding: Thresholding is a simple yet powerful technique used in image segmentation. It involves dividing an image into regions based on intensity values. A threshold value is set, and pixels with intensity values above the threshold are assigned to one region, while those below the threshold are assigned to another. Thresholding is commonly used to separate objects from the background, create binary images, and isolate specific features.

Power Law Contrast Enhancement: Power-law contrast enhancement, also known as gamma correction, is a method used to adjust the contrast and brightness of an image. It involves raising the pixel values to a power, which can enhance or compress intensity ranges. This technique is often used to correct non-linearity in image acquisition devices or to improve the visibility of details in dark or bright areas of an image.

High-Pass Filtering: High-pass filtering is an image processing operation that enhances the edges and fine details in an image. It involves attenuating the low-frequency components of

an image while preserving or boosting the high-frequency components. High-pass filters are designed to emphasize intensity variations between neighboring pixels, thus enhancing the edges and highlights in the image.

Source Code:

```
clc;clear;close all;
%=====Original image=====
inputImage = imread('4.jpeg');          % Read source Image
grayImage = rgb2gray(inputImage);      % Convert to gray scale
subplot(221);
imshow(inputImage);                    % Display Original Image
title('Original Image');
%=====Threshold image=====
thresholdValue = 128;
binaryImage = grayImage > thresholdValue; % Thresholding
subplot(222);
imshow(binaryImage);                   % Display Threshold Image
title('Threshold Image');
%=====Power Enhanced Contrast Image=====
%inputImage = imread('4.jpeg');          % Read source Image
%grayImage = rgb2gray(inputImage);      % Convert to gray scale
% Power-law transformation
gamma = 1.5;c = 1;
enhancedImage = c * (double(grayImage) .^ gamma);
enhancedImage = uint8(255 * (enhancedImage / 255).^(1/gamma)); %Unsigned 8-bit integers, (0 to 255).
subplot(223);
imshow(enhancedImage);                 % Display Power Enhanced Contrast Image
title('Power Enhanced Contrast Image');
%=====High Pass Image=====
laplacianKernel = [0 -1 0; -1 4 -1; 0 -1 0]; % Create a high-pass filter kernel (Laplacian)
% Apply the high-pass filter using convolution
highPassImage = conv2(double(grayImage), laplacianKernel, 'same'); % Convolution
highPassImage = uint8(highPassImage); % Unsigned 8-bit integers, (0 to 255)
subplot(224);
imshow(highPassImage);                 % Power Enhanced Contrast Image
title('High Pass Image');
```

Output:



Experiment No: 10

Name of the experiment: Perform image enhancement, smoothing and sharpening, in spatial domain using different spatial filters and compare the performances.

Objectives:

- i) To implement spatial filters in the spatial domain using MATLAB/Python.
- ii) To understand the concepts of image enhancement, smoothing, and sharpening using spatial filters.

Theory: We will be using MATLAB for the implementation and analysis of these operations as-

Image Enhancement: Image enhancement aims to improve the visual quality of an image by highlighting certain features or suppressing unwanted details. Spatial filters for image enhancement typically involve adjusting pixel values based on their neighborhood.

Smoothing: Smoothing, also known as blurring, involves reducing image noise and suppressing fine details. This is achieved by averaging or weighted averaging of pixel values within a neighborhood. Smoothing is useful for noise reduction and preparation of images for further processing.

Sharpening: Image sharpening enhances edges and fine details in an image. This is achieved by emphasizing differences in intensity values between neighboring pixels. High-pass filters are commonly used for image sharpening.

Spatial Filters: Spatial filters are convolution masks that perform operations on an image's pixel values within a local neighborhood. The filter coefficients determine the nature of the operation.

Different types of filter include such as-

- ✓ Averaging (box filter) for smoothing.
- ✓ Gaussian filter for controlled smoothing and noise reduction.
- ✓ Laplacian filter for sharpening.
- ✓ Un-sharp mask filter for enhancing edges.

Source Code:

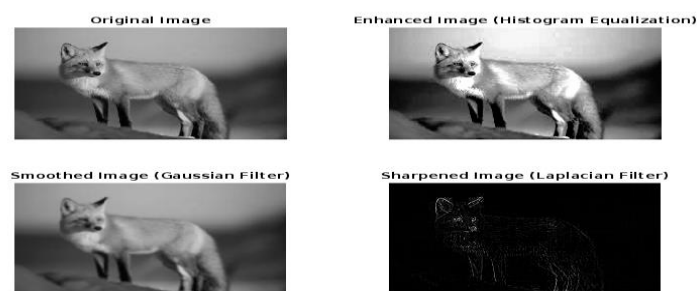
```
clc;close all; clear;
inputImage = imread('Untitled2.jpeg');      % Read source Image
grayImage = rgb2gray(inputImage);          % Convert to gray scale
subplot(221);
imshow(grayImage);                          % Display the original image
title('Original Image');
% =====Image enhancement using histogram equalization=====
enhancedImage = histeq(grayImage);          % To enhance the contrast of an image
subplot(222);
imshow(enhancedImage);                      % Display the Enhanced Image
title('Enhanced Image (Histogram Equalization)');
% ===== Smoothing using Gaussian filter=====
gaussianFilter = fspecial('gaussian', [5 5], 1); % 5x5 Gaussian filter kernel with a standard deviation of 1
```

```

smoothedImage = imfilter(grayImage, gaussianFilter, 'replicate'); % To apply various filters to an image
subplot(223);
imshow(smoothedImage); % Display the Smoothed Image
title('Smoothed Image (Gaussian Filter)');
% =====Sharpening using Laplacian filter=====
laplacianFilter = fspecial('laplacian', 0); % laplacian filter kernel with a standard deviation of 0
sharpenedImage = imfilter(grayImage, laplacianFilter, 'replicate'); % To apply various filters to an image
subplot(224);
imshow(sharpenedImage); % Display the Sharpened Image
title('Sharpened Image (Laplacian Filter)');
set(gcf, 'Position', get(0, 'Screensize')); % Adjust the figure layout

```

Output:



Experiment No: 11

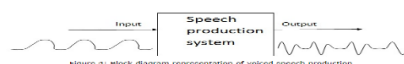
Name of the Experiment: Write A MATLAB/Python Program To Separation Of Voiced/Un-Voiced/Silence Regions From A Speech Signal.

Objectives:

- (i) To identifying and separating voiced/un-voiced/silence regions from a speech signal.
- (ii) To understand about the speech recognition, speaker identification, and emotion recognition technique.

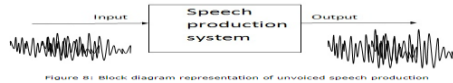
Theory: The process of separating voiced, unvoiced, and silence regions from a speech signal is a crucial step in many speech processing applications, such as speech recognition, speaker identification, and emotion recognition. This separation helps in isolating meaningful speech information from non-speech components.

Voiced Regions: Voiced regions in a speech signal correspond to segments where the vocal cords vibrate periodically, producing harmonically structured sounds.



Example: The sustained sound of a vowel like "aaa" or "eee" usually falls within a voiced region.

Unvoiced Regions: Unvoiced regions consist of segments in which there is no periodic vibration of the vocal cords, resulting in noise-like or turbulent sounds.



Example: The sounds of "ssss" or "shhh" are typically unvoiced, as they are produced by creating turbulence in the airflow without vocal cord vibration.

Silence Regions: Silence regions represent periods within the speech signal where there is no significant audible sound.

Example: The pauses between spoken words or sentences, where no sound is produced, are considered silence regions.

By identifying and separating these regions, speech processing systems can focus on analyzing and extracting relevant information from voiced regions while filtering out noise and non-speech segments.

Source Code:

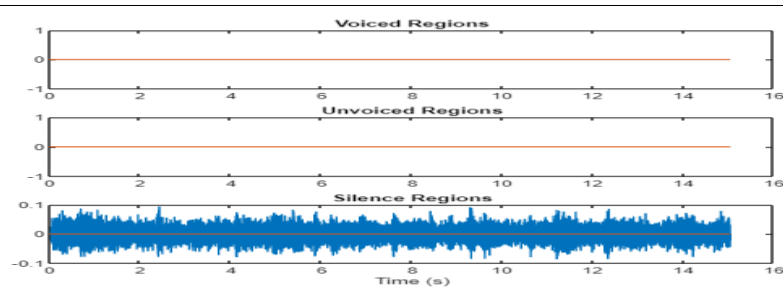
```
clc;clear all;close all;
[speechSignal, Fs] = audioread('intro.mp3'); % Read the speech signal
% Parameters for analysis
frameSize = 0.02; % Frame size in seconds
overlap = 0.5; % Overlap ratio
thresholdEnergy = 0.03; % Energy threshold for silence detection
thresholdZCR = 20; % Zero-crossing rate threshold for voicing detection
frameLength = round(Fs * frameSize);
overlapLength = round(frameLength * overlap);
numFrames = floor((length(speechSignal) - overlapLength) / (frameLength - overlapLength));
% Assign zero
voicedRegions = zeros(size(speechSignal));
unvoicedRegions = zeros(size(speechSignal));
silenceRegions = zeros(size(speechSignal));
for i = 1:numFrames
    startIdx = (i - 1) * (frameLength - overlapLength) + 1;
    endIdx = startIdx + frameLength - 1;
    frame = speechSignal(startIdx:endIdx);
    energy = sum(frame.^2) / frameLength;
    zcr = sum(frame(1:end-1) .* frame(2:end) < 0) / frameLength;
    if energy < thresholdEnergy
        silenceRegions(startIdx:endIdx) = frame;
    elseif zcr > thresholdZCR
        unvoicedRegions(startIdx:endIdx) = frame;
    else
        voicedRegions(startIdx:endIdx) = frame;
    end
end
end
```

```
% Plot the results
time = (0:length(speechSignal)-1) / Fs;
subplot(3,1,1), plot(time, voicedRegions), title('Voiced Regions')
subplot(3,1,2), plot(time, unvoicedRegions), title('Unvoiced Regions')
subplot(3,1,3), plot(time, silenceRegions), title('Silence Regions')
xlabel('Time (s)')

% Play the separated regions
disp("Waiting Please....");
soundsc(voicedRegions, Fs);
disp("Play Original Speech");
pause(length(voicedRegions) / Fs);
soundsc(unvoicedRegions, Fs);
disp("Play Unvoiced Speech");
pause(length(unvoicedRegions) / Fs);
soundsc(silenceRegions, Fs);
disp("Play silence Speech")
```

Output:

Waiting Please....
 Play Original Speech
 Play Unvoiced Speech
 Play silence Speech



Experiment No: 12

Name of the Experiment: Write A MATLAB/Python Program and Plot Multilevel Speech Resolution.

Objectives:

- (i) To understand and analysis about the Multilevel speech resolution.
- (ii) To extract a comprehensive set of features that can be used for various applications.

Theory: Multilevel Speech Resolution seems to be a term that refers to the process of analyzing and understanding speech signals at multiple levels of detail. This approach is often used in speech processing and analysis tasks to capture both fine-grained details and higher-level contextual information.

Here's an overview of how multilevel speech resolution might work:

Low-Level Resolution: At the lowest level, the speech signal is typically analyzed using signal processing techniques to capture basic characteristics such as amplitude, frequency, and time-domain features.

Mid-Level Resolution: In this level, more complex features are extracted from the speech signal. This could involve identifying segments of speech like vowels, consonants, or syllables.

High-Level Resolution: This level involves capturing broader linguistic and semantic features of the speech. It focuses on understanding the context and meaning of the spoken words and phrases.

Contextual Resolution: This level considers the speech signal in the context of the surrounding words and phrases. It involves analyzing longer segments of speech to understand the flow and coherence of the conversation.

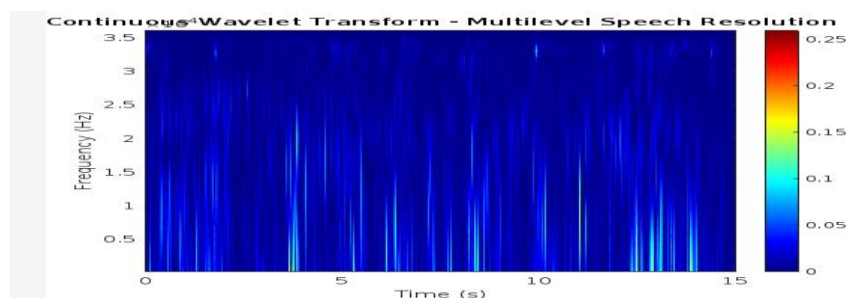
By analyzing and understanding the speech signal at multiple levels, it's possible to extract a comprehensive set of features that can be used for various applications, including speech recognition, language understanding, speaker identification, and more.

Source Code:

```
clc;clear all;close all;
[speechSignal, Fs] = audioread('intro.mp3');    % Read the speech signal
% Parameters for CWT
scales = 1:64;                                % Scale levels
waveletName = 'morl';                          % Wavelet function (change as needed)
cwtCoeffs = cwt(speechSignal, scales, waveletName); % Perform CWT

% Create a time-frequency plot
time = (0:length(speechSignal)-1) / Fs;
freq = scal2frq(scales, waveletName, 1/Fs);
figure;
imagesc(time, freq, abs(cwtCoeffs));
colormap(jet);
colorbar;
set(gca, 'YDir', 'normal');
xlabel('Time (s)');
ylabel('Frequency (Hz)');
title('Continuous Wavelet Transform - Multilevel Speech Resolution');
```

Output:



Experiment No: 13

Name of the Experiment: Write A MATLAB/Python Program To Recognize Speech Signal.

Objectives:

- (i) To identify words or phrases in spoken language in a machine-readable format.
- (ii) To convert a speech signal into word sequences.

Theory: Speech Recognition can be defined as the ability of a machine or program to identify words or phrases in spoken language in a machine-readable format. It is the process of converting a speech signal into word sequences by implementing computer algorithms or programming.

Automatic Speech Recognition, often known as **ASR**, is a technique that converts human speech into text that can be read by using either machine learning or artificial intelligence (AI) technology. Figure-14 shows a basic structure of an ASR system, where a statistical pattern recognition method mainly used. ASR system is consisting of three processes: (1) feature extraction, (2) training and (3) matching.

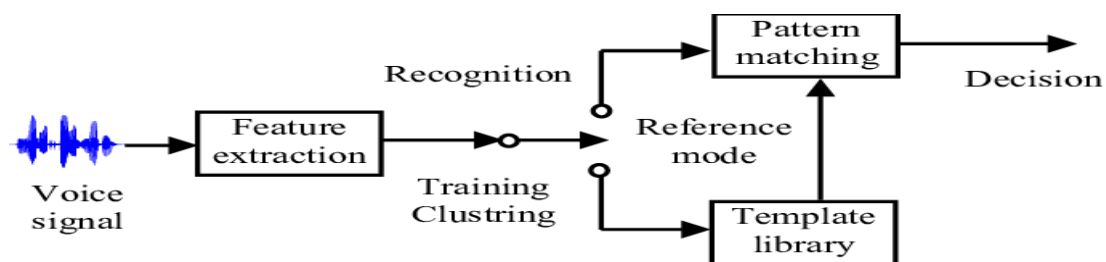


Figure-14: Basic structure of automatic speech recognition system.

Mel-Frequency Cepstral Coefficients (**MFCCs**) are probably the most commonly used technique to represent the speech spectrum in ASR systems. That defined as

$$f_{\text{mel}} = 2595 \log_{10} \left(1 + \frac{f_{\text{Hz}}}{700} \right)$$

Where, f_{mel} is the frequency in Mel, and f_{Hz} in Hz.

Source Code:

```
voice=audioread('one.wav');
x=voice;x=x';x=x(1,:);x=x';
y1=audioread('one.wav');
y1=y1';y1=y1(1,:);y1=y1';
z1=xcorr(x,y1);m1=max(z1);l1=length(z1);
t1=-((l1-1)/2):1:((l1-1)/2);

t1=t1';subplot(3,2,1);plot(t1,z1);
y2=audioread('two.wav');
y2=y2';y2=y2(1,:);y2=y2';
z2=xcorr(x,y2);m2=max(z2);l2=length(z2);
```

```

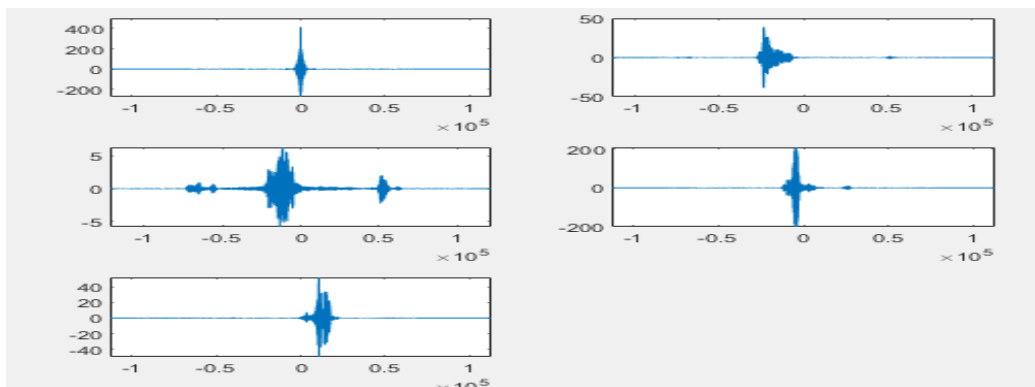
t2=-((l2-1)/2):1:((l2-1)/2);t2=t2';
subplot(3,2,2);plot(t2,z2);
y3=audioread('three.wav');
y3=y3';y3=y3(1,:);y3=y3';z3=xcorr(x,y3);m3=max(z3);
l3=length(z3);t3=-((l3-1)/2):1:((l3-1)/2);
t3=t3';subplot(3,2,3);plot(t3,z3);
y4=audioread('four.wav');
y4=y4';y4=y4(1,:);y4=y4';z4=xcorr(x,y4);
m4=max(z4);l4=length(z4);

t4=-((l4-1)/2):1:((l4-1)/2);t4=t4';subplot(3,2,4);plot(t4,z4);
y5=audioread('five.wav');y5=y5';y5=y5(1,:);y5=y5';z5=xcorr(x,y5);
m5=max(z5);l5=length(z5);t5=-((l5-1)/2):1:((l5-1)/2);t5=t5';subplot(3,2,5);
plot(t5,z5);m5=300;a=[m1 m2 m3 m4 m5];m=max(a);h=audioread('allow.wav');

if m<=m1
    soundsc(audioread('one.wav'),50000)
    soundsc(h,50000)
elseif m<=m2
    soundsc(audioread('two.wav'),50000)
    soundsc(h,50000)
elseif m<=m3
    soundsc(audioread('three.wav'),50000)
    soundsc(h,50000)
elseif m<=m4
    soundsc(audioread('four.wav'),50000)
    soundsc(h,50000)
elseif m<m5
    soundsc(audioread('five.wav'),50000)
    soundsc(h,50000)
else
    soundsc(audioread('denied.wav'),50000)
end

```

Output:



Experiment No: 14

Name of the Experiment: Write A MATLAB/Python Program for Text-To-Speech Conversion and Record Speech Signal.

Objectives:

- (i) To convert an arbitrary input text into intelligible and natural sounding speech.
- (ii) To convert written text into spoken language.
- (iii) To record speech signal from user voice.

Theory: A Text-to-Speech (TTS) conversion system is a technology that converts written text into spoken language. TTS systems enable computers to generate natural-sounding speech that mimics human speech patterns, intonations, and accents.

Here's a detailed explanation of the components and processes involved in a typical TTS conversion system-

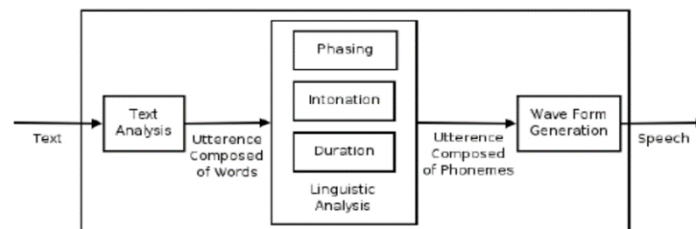


Figure-15: Block diagram of a Text-to-Speech Conversion.

Text Processing and Analysis: The process begins with the input text that needs to be converted into speech. The input text is analyzed to identify linguistic features such as words, sentences, and punctuation.

Linguistic Analysis: The system analyzes the linguistic features to understand the structure of the text, including grammar, syntax, and semantics. This helps in determining the appropriate prosody (intonation, rhythm, stress) for generating natural-sounding speech.

Phoneme Duration and Prosody Generation: The system assigns durations to each phoneme based on linguistic rules and the desired speech rate. It also generates prosody information, which includes pitch contour, stress patterns, and rhythm, to make the speech sound more natural and expressive.

Output Generation: The final synthesized speech is output as an audio waveform. This waveform can be saved as an audio file or directly played through a speaker.

Source Code:

```
voice = 'Pabna University of Science and Technology'; % Input voice
sampleRate = 44100; % You can adjust this based on your preferences
NET.addAssembly('System.speech');
mySpeaker=System.Speech.Synthesis.SpeechSynthesizer;
```

```

mySpeaker.Rate=3;
mySpeaker.Volume=100;
Speak(mySpeaker, voice); % Text to Speech
% Record the speech
recordTime = 5; % Duration of recording in seconds
recObj = audiorecorder(sampleRate, 16, 1); % Start recording
disp('Recording speech...');
recordblocking(recObj, recordTime);
disp('Recording completed.');
```

```

recordedSignal = getaudiodata(recObj); % Get the recorded speech signal
audiowrite('recorded_speech.wav', recordedSignal, sampleRate); % Save the recorded speech signal to a file (optional)
time = (0:length(recordedSignal)-1) / sampleRate;
figure;
plot(time, recordedSignal); % Plot the recorded speech signal
xlabel('Time (s)');
ylabel('Amplitude');
title('Recorded Speech Signal');
```

Output:

