MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

Course Title: DSP Lab

Course Code: CSE4104

Report No: 02

Report Title: Lab Report on Image Processing Techniques.

LAB REPORT

Submitted by	Submitted to
Name :Alamgir Hosain ID:CE21012 Session: 2020-21 Year: 4 th Semester: 1 st Department: Computer Science and Engineering. Mawlana Bhashani Science and Technology University.	Dr. Mahbuba Begum Associate Professor, Department of Computer Science and Engineering, Mawlana Bhashani Science and Technology University.

Lab Report: Solve simple equations.

Problem Definition
To solve a simple linear algebraic equation using MATLAB.
The equation is: $2x + 3 = 11$

Objective

1. To understand the use of MATLAB for solving basic algebraic equations.
2. To learn how to define variables and perform arithmetic operations in MATLAB.
3. To determine the value of the unknown variable x in the given linear equation.

Tools Used

1. Software : MATLAB R2023b
2. Operating System : Windows
3. Editor : MATLAB Editor

Theory
A linear equation is an algebraic equation in which each term is either a constant or the product of a constant and a single variable ,general form :ax+b=c

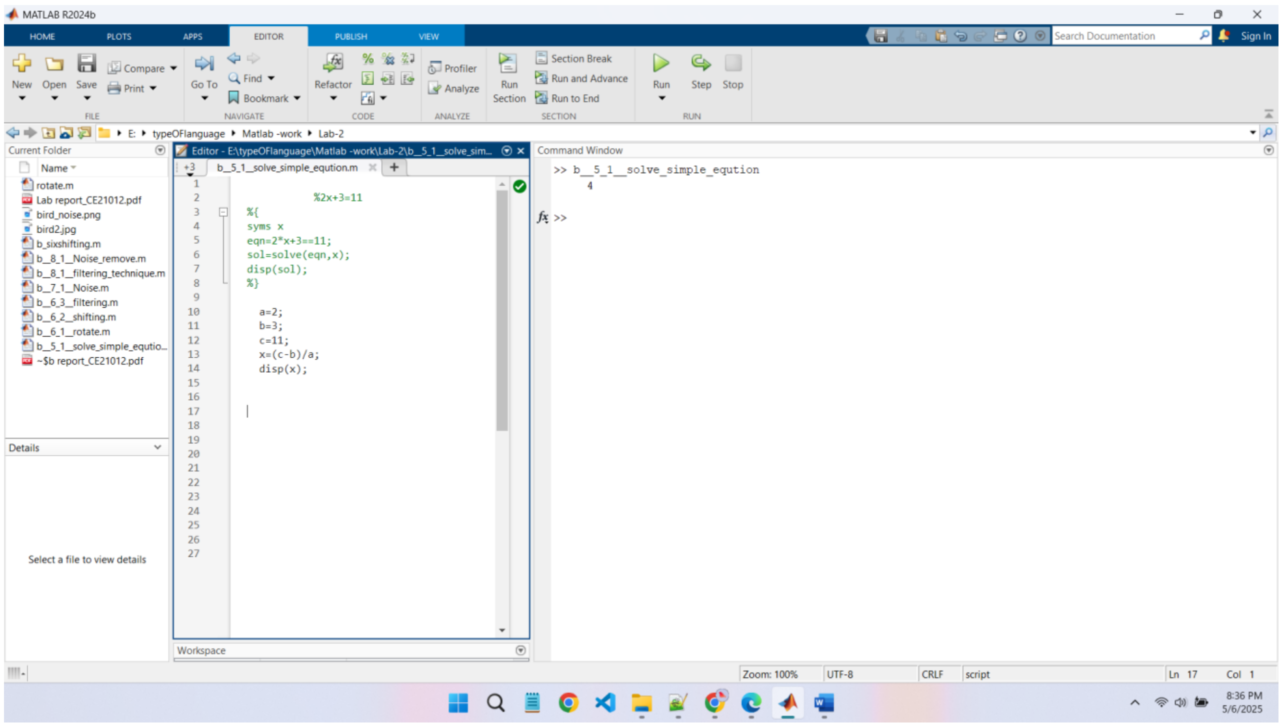
Method / Procedure:

1. Open Matlab
2. Define the coefficients and constants.
3. Assign values to the variables representing the equation components.

Code

```
           %2x+3=11
%{
syms x
eqn=2*x+3==11;
sol=solve(eqn,x);
disp(sol);
%}

  a=2;
  b=3;
  c=11;
  x=(c-b)/a;
  disp(x);
```

Result (Screenshot of entire screen)



Editor - E:\typeOFlanguage\Matlab -work\Lab-2\b__5_1__solve_sim...

b__5_1__solve_simple_eqution.m

```
              %2x+3=11
%{
syms x
eqn=2*x+3==11;
sol=solve(eqn,x);
disp(sol);
%}

  a=2;
  b=3;
  c=11;
  x=(c-b)/a;
  disp(x);
```

Command Window

```
>> b__5_1__solve_simple_eqution
     4

fx >>
```

Result Discussion

The experiment successfully demonstrated solving a simple linear equation using MATLAB. The correct value of x = 4 was obtained, validating both the mathematical approach and the implementation in MATLAB.

Lab Report: Image processing operations: Image rotation, shifting,filtering (average, median, LPGF, mean filter).

Problem Definition

To perform fundamental image processing operations on a digital image using MATLAB. The operations include:

1. Image rotation
2. Image shifting (translation)
3. Filtering using average, median, low-pass Gaussian filter (LPGF), and mean filter.

Objective

1. To understand the basic functionality of image processing.
2. To understand the application of spatial domain transformations in image processing.
3. To implement image rotation and shifting using MATLAB.
4. To apply and compare different image filters to remove noise or enhance image features.

Tools Used

1. Software : MATLAB R2023b
2. Operating System : Windows
3. Editor : MATLAB Editor

Theory

1. **Image Rotation:** Image rotation turns the image by a certain angle around its center. Rotating an image by 90 degrees clockwise involves shifting the column into rows of the image matrix and 180 degrees means flipping both rows and columns.
2. **Image Shifting (Translation):**Shifting an image moves it horizontall or vertically or diagonally.
3. **Filtering:**Filtering is used to enhance image quality or remove noise.
   i. **Average Filter or Mean Filter** : The average filter replaces each pixel's value with the average of its neighboring pixels within a 3x3 kernel.
   ii. **Median Filter** : The median filter replaces each pixel's value with the median value of its neighboring pixels within a 3x3 kernel.
   iii. **LPGF (Low-Pass Gaussian Filter):** Applies a Gaussian kernel to smooth the image, reducing high-frequency noise.

Method/Procedure

1. Rotation :
   i. Initialize a new matrix for the rotated image.
   ii. Use nested loops to iterate over each pixel in the original image, placing the pixels in their new positions in the rotated image matrix..

2. Shifting :
   i. Create a new matrix for the shifted image.
   ii. Use nested loops to shift the pixel values by a specified amount to the right.

3. Filtering :
   i. Average Filter : Use nested loops to iterate over the image and apply the average filter using a 3x3 kernel.
   ii. Median Filter : Similar to the average filter but replacing each pixel with the median of its 3x3 neighborhood, sorted manually.

4. Display Results : All the processed images (grayscale, rotated, shifted, average filtered, and median filtered) are displayed in a 2x3 subplot for comparison.

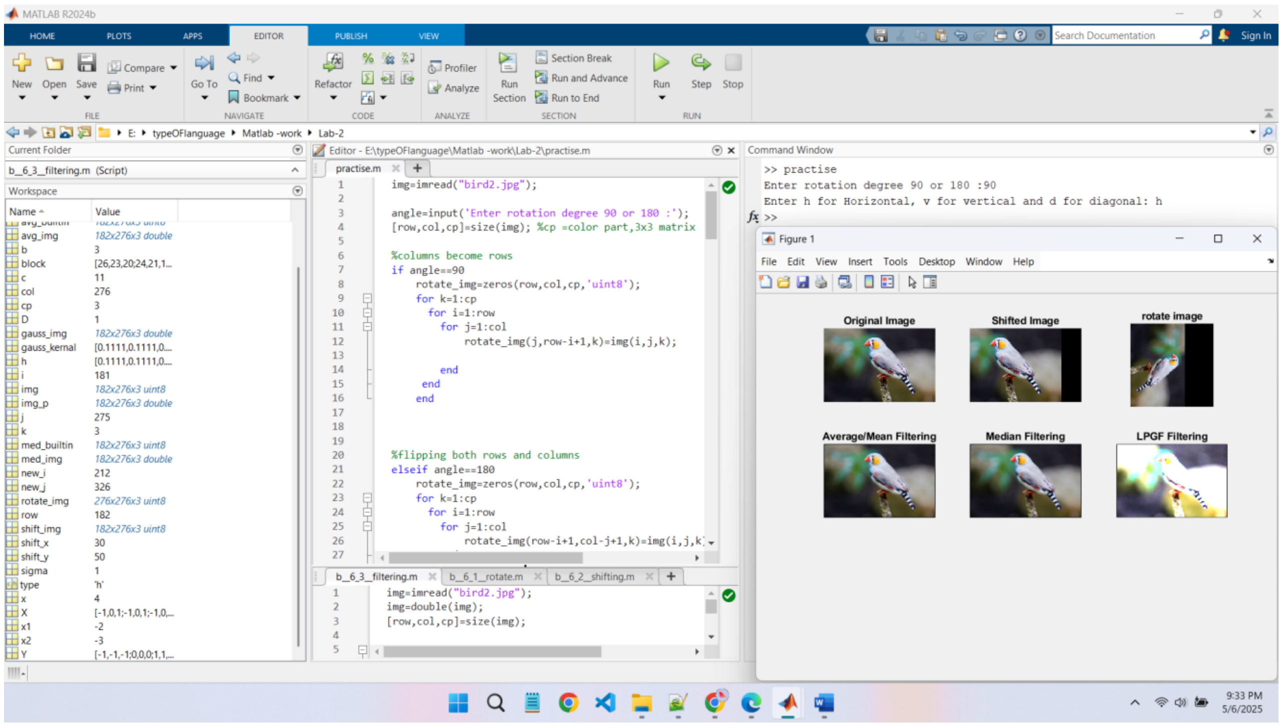
Code

```
img=imread("bird2.jpg");
angle=input('Enter rotation degree 90 or 180 :');
[row,col,cp]=size(img); %cp =color part,3x3 matrix
%columns become rows
if angle==90
    rotate_img=zeros(row,col,cp,'uint8');
    for k=1:cp
      for i=1:row
        for j=1:col
            rotate_img(j,row-i+1,k)=img(i,j,k);
        end
     end
    end
%flipping both rows and columns
elseif angle==180
    rotate_img=zeros(row,col,cp,'uint8');
    for k=1:cp
      for i=1:row
        for j=1:col
            rotate_img(row-i+1,col-j+1,k)=img(i,j,k);
        end
       end
    end
else
    error('enter 90 or 180 ');
end
type = input('Enter h for Horizontal, v for vertical and d for
diagonal: ', 's');
% Shift amounts
shift_x = 30;  % Vertical shift (downward)
```

```
shift_y = 50;  % Horizontal shift (to the right)
% Initialize shifted image
shift_img = zeros(size(img), "uint8");
% Horizontal shift
% move pixels down by subtracting shift_y from the col index.
if lower(type) == 'h'
    for k = 1:cp
        for i = 1:row
            for j = 1:col
                new_j = j + shift_y;  % Changed to + for right
shift
                if new_j > 0 && new_j <= col
                    shift_img(i, j, k) = img(i, new_j, k);
                end
            end
        end
    end
% Vertical shift ,
% move pixels down by subtracting shift_x from the row index.
elseif lower(type) == 'v'
    for k = 1:cp
        for i = 1:row
            for j = 1:col
                new_i = i + shift_x;  % Changed to + for
downward shift
                if new_i > 0 && new_i <= row
                    shift_img(i, j, k) = img(new_i, j, k);
                end
            end
        end
    end
% Diagonal shift
elseif lower(type) == 'd'
    for k = 1:cp
        for i = 1:row
            for j = 1:col
                new_i = i + shift_x;
                new_j = j + shift_y;
                if new_i > 0 && new_i <= row && new_j > 0 &&
new_j <= col
                    shift_img(i, j, k) = img(new_i, new_j, k);
                end
            end
        end
    end
else
    error('Invalid input. Please enter h, v, or d.');
end
img_p=double(img);
%        _________________Without Builtin_________________
%avegrage filtering=mean filtering
% create 3x3 avegare block
avg_img=zeros(size(img_p));
for k=1:cp
    for i=2:row-1
        for j=2:col-1
            block=img_p(i-1:i+1  , j-1:j+1 , k);
            avg_img(i,j,k)=sum(block(:))/9;
        end
    end
end
%median Filtering
med_img=zeros(size(img_p));
for k=1:cp
    for i=2:row-1
        for j=2:col-1
            block=img_p(i-1:i+1  , j-1:j+1 , k);
            med_img(i,j,k)=median(block(:));
        end
    end
end
% LPGF-Low Pass Gaussian Filter)
% sigma = 1.0 controls blur intensity.
% G(x,y)=e(-x2+y2/2*sigma2

gauss_img=zeros(size(img_p));
sigma=1.0;
[X,Y]=meshgrid(-1:1 , -1:1);
gauss_kernal=exp(-(X^2+Y^2)/(2*sigma^2));
gauss_kernal=gauss_kernal/sum(gauss_kernal(:));% Normalize to
sum = 1
for k=1:cp
    for i=2:row-1
        for j=2:col-1
             block=img_p(i-1:i+1  , j-1:j+1 , k);
            gauss_img(i,j,k)=sum(sum(block*gauss_kernal));
        end
    end
```

```
end
% Display results
figure;
subplot(3, 3, 1), imshow(img), title("Original Image");
subplot(3, 3, 2), imshow(shift_img), title("Shifted Image");
subplot(3,3,3),imshow(rotate_img),title('rotate image');
subplot(3, 3, 4), imshow(uint8(avg_img)), title("Average/Mean
Filtering");
subplot(3, 3, 5), imshow(uint8(avg_img)), title("Median
Filtering");
subplot(3, 3, 6), imshow(uint8(gauss_img)), title("LPGF
Filtering");
```

Result/Output



Editor - E:\typeOFlanguage\Matlab -work\Lab-2\practise.m

```
img=imread("bird2.jpg");

angle=input('Enter rotation degree 90 or 180 :');
[row,col,cp]=size(img); %cp =color part,3x3 matrix

%columns become rows
if angle==90
    rotate_img=zeros(row,col,cp,'uint8');
    for k=1:cp
      for i=1:row
        for j=1:col
            rotate_img(j,row-i+1,k)=img(i,j,k);

        end
      end
    end



%flipping both rows and columns
elseif angle==180
    rotate_img=zeros(row,col,cp,'uint8');
    for k=1:cp
      for i=1:row
        for j=1:col
            rotate_img(row-i+1,col-j+1,k)=img(i,j,k
```

b_6_3_filtering.m

```
img=imread("bird2.jpg");
img=double(img);
[row,col,cp]=size(img);
```

Command Window

```
>> practise
Enter rotation degree 90 or 180 :90
Enter h for Horizontal, v for vertical and d for diagonal: h
>>
```

Workspace

Name	Value
avg_img	182x276x3 double
b	3
block	[26,23,20;24,21,1...
c	11
col	276
cp	3
D	1
gauss_img	182x276x3 double
gauss_kernal	[0.1111,0.1111,0....
h	[0.1111,0.1111,0....
i	181
img	182x276x3 uint8
img_p	182x276x3 double
j	275
k	3
med_builtin	182x276x3 uint8
med_img	182x276x3 double
new_i	212
new_j	326
rotate_img	276x276x3 uint8
row	182
shift_img	182x276x3 uint8
shift_x	30
shift_y	50
sigma	1
type	'h'
x	4
X	[-1,0,1;-1,0,1;-1,0,...
x1	-2
x2	-3
Y	[-1,-1,-1;0,0,0;1,1,...

Result Discussion

Each type of noise was successfully added to the image, and their effects were analyzed.
Each noise type has different visual characteristics, and these differences are useful in understanding noise in digital images and how to handle them in image processing tasks.

Lab Report: Adding noise in an image (SPN, SN, PN, GN)

Problem Definition

This experiment aims to apply various types of noise—Salt-and-Pepper, Gaussian, Poisson, and Speckle—to an image and study their impact.

Objective

1. To understand the characteristics of different types of noise in images.
2. To apply and visualize Salt-and-Pepper, Gaussian, Poisson, and Speckle noise.
3. To compare the impact of each noise type on the original image.
4. To prepare the image for further filtering or denoising experiments.

Tools Used

1. Software: MATLAB R202x (any version with Image Processing Toolbox)
2. Hardware**:** Computer with MATLAB installed.

Theory

1. **Salt-and-Pepper Noise (SPN):**
   i. Randomly turns pixels black (0) or white (1).
   ii. Caused by sharp and sudden disturbances in the image signal.
2. **Gaussian Noise (GN)**
   i. Continuous noise with a normal distribution.
   ii. gn_image=original + noise
   iii. Mean=average of the noise usually 0
   iv. variance=higher v. higher noise
3. **Speckle Noise (SN)**
   i. sn_image = original + (original × noise)
   ii. Multiplicative noise, common in radar and ultrasound images.
4. **Poisson Noise (PN)**
   i. pn = poissrnd(img * 255) / 255;
   ii. pn = min(max(pn, 0), 1)

Method/Procedure

1. Adding Salt and Pepper Noise :

A random number is generated for each pixel, and if it is less than a set threshold (5% of the total pixels), the pixel is replaced with either 0 (black) or 255 (white).

2. Adding Gaussian Noise :
   Gaussian noise is generated using the Box-Muller transform. Random numbers are generated, and the noise is scaled by a standard deviation value (20 in this case).
3. Mean Filtering :

A 3x3 kernel is used to compute the average of each pixel's neighborhood to reduce noise.

5. Median Filtering : A 3x3 window is used to replace each pixel with the median value of the surrounding pixels to reduce Salt and Pepper noise.

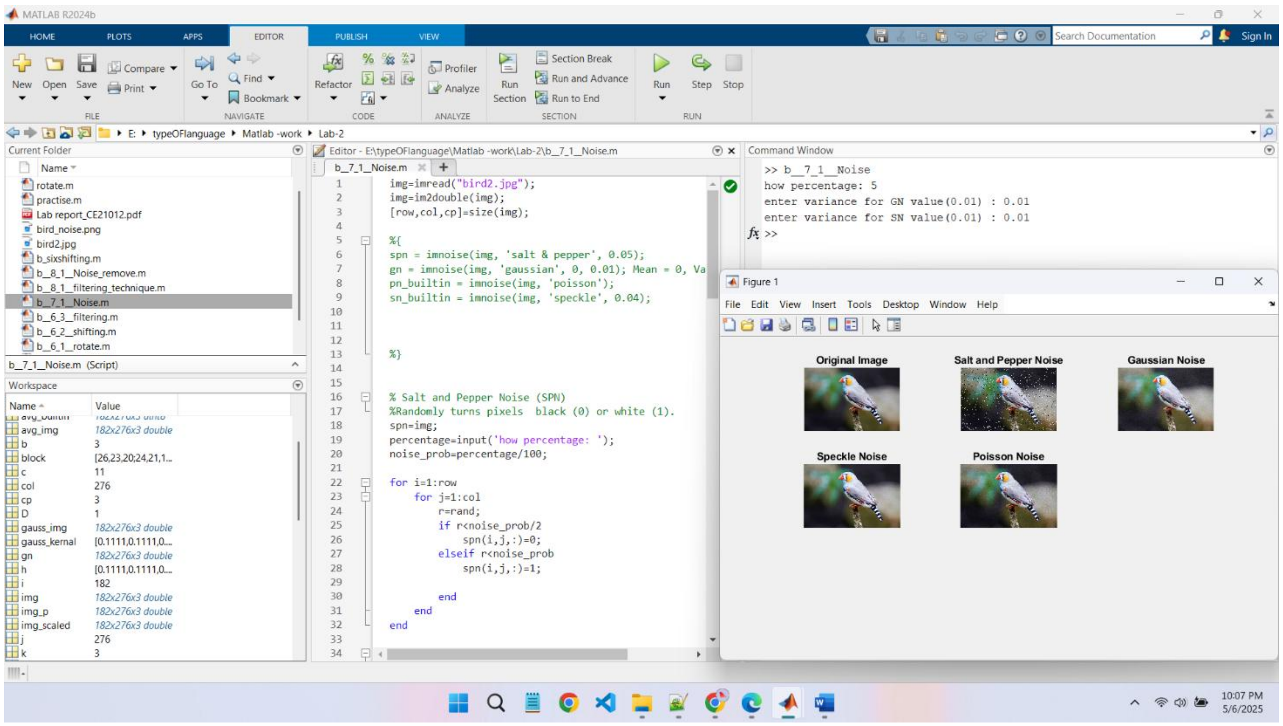
6. Display Results : The original image, noisy images, and filtered images are displayed using the subplot() function to compare the effects of the noise and filters.

Code

```
img=imread("bird2.jpg");
img=im2double(img);
[row,col,cp]=size(img);

% Salt and Pepper Noise (SPN)
%Randomly turns pixels  black (0) or white (1).
spn=img;
percentage=input('how percentage: ');
noise_prob=percentage/100;

for i=1:row
    for j=1:col
        r=rand;
        if r<noise_prob/2
            spn(i,j,:)=0;
        elseif r<noise_prob
            spn(i,j,:)=1;

        end
    end
end

% Gaussian Noise (GN)
  % gn_image=original + noise
  % Mean=average of the noise usually 0
  % variance=higher v. higher noise
% Continuous noise with a normal distribution.

variance=input('enter variance for GN value(0.01) : ');
gn=img+sqrt(variance)*rand(size(img));
gn=min(max(gn,0),1); % Clip values to stay between 0 and 1
%4. Speckle Noise (SN)
    % sn_image = original + (original × noise)

variance2=input('enter variance for SN value(0.01) : ');
sn=img+img.*sqrt(variance2).*rand(size(img));
```

```
sn=min(max(sn,0),1); % Clip values to stay between 0 and 1

% Poisson Noise (PN)
%pn = poissrnd(img * 255) / 255;
%pn = min(max(pn, 0), 1);

img_scaled = img * 255; % Scale to [0,255]
pn = zeros(size(img));

%To generate a Poisson random variable with mean λ:
for i = 1:row
    for j = 1:col
        for k = 1:cp
            lambda = img_scaled(i,j,k);
            L = exp(-lambda);
            p = 1;
            n = 0;
            while p > L
                n = n + 1;
                p = p * rand();
            end
            pn(i,j,k) = (n - 1);  % Poisson sample
        end
    end
end
pn = pn / 255;            % Scale back to [0,1]
pn = min(max(pn, 0), 1); % Clip values
subplot(3, 3, 1), imshow(img), title("Original Image");
subplot(3, 3, 2), imshow(spn), title("Salt and Pepper Noise");
subplot(3, 3, 3), imshow(gn), title("Gaussian Noise");
subplot(3, 3, 4), imshow(sn), title("Speckle Noise");
subplot(3, 3, 5), imshow(pn), title("Poisson Noise");
```

Result/Output



```
img=imread("bird2.jpg");
img=im2double(img);
[row,col,cp]=size(img);

%{
spn = imnoise(img, 'salt & pepper', 0.05);
gn = imnoise(img, 'gaussian', 0, 0.01); Mean = 0, Va
pn_builtin = imnoise(img, 'poisson');
sn_builtin = imnoise(img, 'speckle', 0.04);



%}


% Salt and Pepper Noise (SPN)
%Randomly turns pixels  black (0) or white (1).
spn=img;
percentage=input('how percentage: ');
noise_prob=percentage/100;

for i=1:row
    for j=1:col
        r=rand;
        if r<noise_prob/2
            spn(i,j,:)=0;
        elseif r<noise_prob
            spn(i,j,:)=1;

        end
    end
end
```

```
>> b__7_1__Noise
how percentage: 5
enter variance for GN value(0.01) : 0.01
enter variance for SN value(0.01) : 0.01
>>
```

Result Discussion

1. Salt-and-Pepper Noise (SPN): Introduces sharp black and white dots; mainly affects image clarity and fine details.
2. Gaussian Noise (GN): Adds smooth, grainy distortion across the image; common in electronic imaging.
3. Poisson Noise (PN): Signal-dependent noise; more visible in brighter areas, simulates low-light or photon-limited scenarios.
4. Speckle Noise (SN): Grainy, multiplicative distortion; affects image contrast, common in radar/medical imaging.

Lab Report: Remove noise using filtering techniques.

Problem Definition

This experiment aims to remove different types of noise using three filtering techniques: Gaussian, Median, and Mean (Averaging) filters.

Objective

1. To apply Gaussian, Median, and Mean filters on noisy images.
2. To observe how each filter handles different types of noise (especially SPN and GN).
3. To compare their effectiveness in preserving image details while reducing noise.

Tools Used

5. Software: MATLAB R202x (any version with Image Processing Toolbox)
6. Hardware**:** Computer with MATLAB installed.

Theory

1. Mean Filter:A linear filter that replaces each pixel with the average of its neighbors.Good for Gaussian noise but blurs edges.

2. Gaussian Filter:Uses a Gaussian kernel to compute a weighted average . Smooths the image and reduces high-frequency noise while preserving more structure than mean filter.

3. Median Filter:A non-linear filter that replaces each pixel with the median of neighboring pixels.Extremely effective for removing Salt-and-Pepper noise without blurring edges**.**

Method/Procedure

1. Open MATLAB.

2. Mean Filtering : Traverse the image using a 3×3 sliding window. Replace each center pixel with the average of its 8 neighbors and itself.

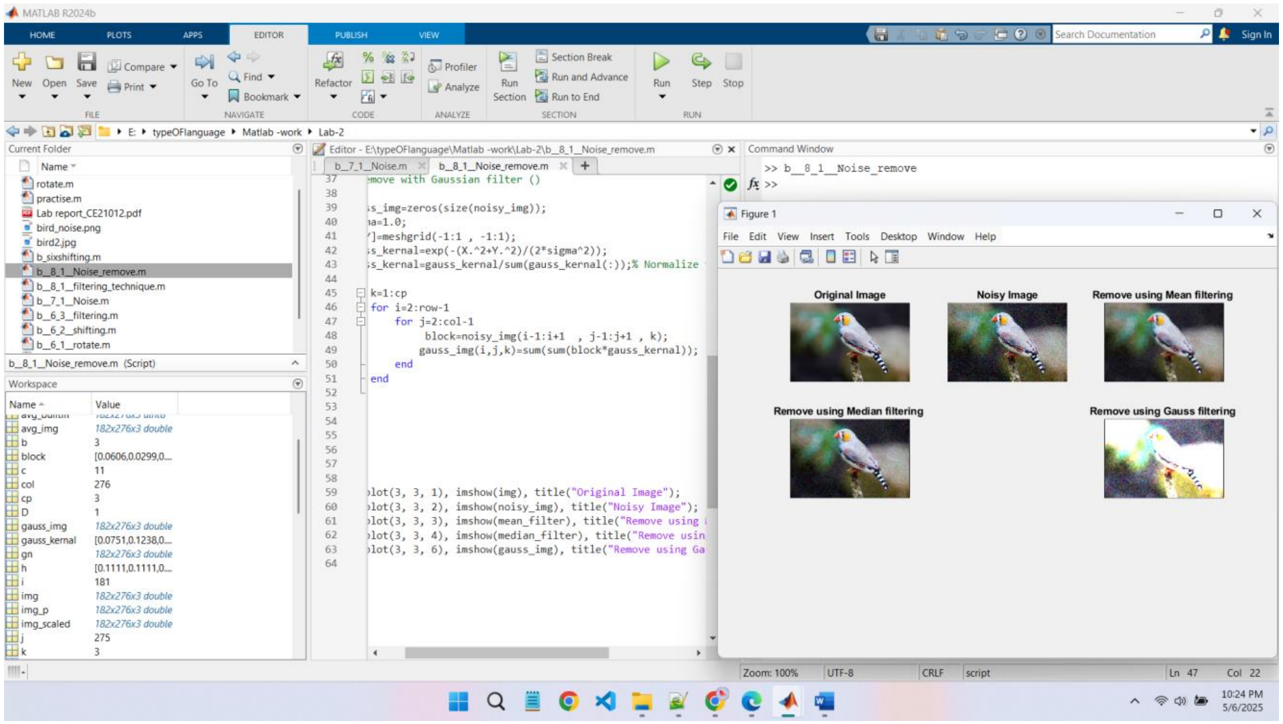
3. Gaussian Filtering :Manually create a normalized 3×3 Gaussian kernel with σ = 1. Convolve this kernel with the image using nested loops.

4. Display Results : Use subplot() and imshow() to visualize the original, mean filtered, and Gaussian filtered images side by side.

Code

```
img=imread("bird2.jpg");
img=im2double(img);
[row,col,cp]=size(img);
noisy_img = imnoise(img, 'gaussian', 0, 0.01);

%{
mean_builtin = imfilter(noisy_img, fspecial('average', [3 3]),
'replicate');
median_builtin = medfilt2(noisy_img, [3 3]);


%}

%Remove with Mean filter
mean_filter = zeros(size(img));
for k=1:cp
 for i = 2:row-1
    for j = 2:col-1
        block = noisy_img(i-1:i+1, j-1:j+1,k);
        mean_filter(i, j,k) = mean(block(:));
    end
 end
end


% Remove with Median filter
median_filter = zeros(size(img));
for k = 1:cp
    for i = 2:row-1
        for j = 2:col-1
            block = noisy_img(i-1:i+1, j-1:j+1, k);
            median_filter(i, j, k) = median(block(:));
        end
    end
end


% Remove with Gaussian filter ()

gauss_img=zeros(size(noisy_img));
sigma=1.0;
[X,Y]=meshgrid(-1:1 , -1:1);
gauss_kernal=exp(-(X.^2+Y.^2)/(2*sigma^2));
gauss_kernal=gauss_kernal/sum(gauss_kernal(:));%  Normalize  to
sum = 1

for k=1:cp
    for i=2:row-1
        for j=2:col-1
             block=noisy_img(i-1:i+1  , j-1:j+1 , k);
            gauss_img(i,j,k)=sum(sum(block*gauss_kernal));
        end
    end
end
subplot(3, 3, 1), imshow(img), title("Original Image");
subplot(3, 3, 2), imshow(noisy_img), title("Noisy Image");
subplot(3, 3, 3), imshow(mean_filter), title("Remove using Mean
filtering ");
subplot(3,  3,  4),  imshow(median_filter),  title("Remove  using
Median filtering ");
subplot(3, 3, 6), imshow(gauss_img), title("Remove using Gauss
filtering ");
```

Result/Output(Screenshot of entire screen)



Editor - E:\typeOFlanguage\Matlab -work\Lab-2\b__8_1__Noise_remove.m

```
emove with Gaussian filter ()

s_img=zeros(size(noisy_img));
ia=1.0;
']=meshgrid(-1:1 , -1:1);
s_kernal=exp(-(X.^2+Y.^2)/(2*sigma^2));
s_kernal=gauss_kernal/sum(gauss_kernal(:));% Normalize

k=1:cp
for i=2:row-1
    for j=2:col-1
         block=noisy_img(i-1:i+1  , j-1:j+1 , k);
        gauss_img(i,j,k)=sum(sum(block*gauss_kernal));
    end
end






plot(3, 3, 1), imshow(img), title("Original Image");
plot(3, 3, 2), imshow(noisy_img), title("Noisy Image");
plot(3, 3, 3), imshow(mean_filter), title("Remove using
plot(3, 3, 4), imshow(median_filter), title("Remove usin
plot(3, 3, 6), imshow(gauss_img), title("Remove using Ga
```

Command Window

```
>> b__8_1__Noise_remove
>>
```

Workspace

Name	Value
avg_img	182x276x3 double
b	3
block	[0.0606,0.0299,0....
c	11
col	276
cp	3
D	1
gauss_img	182x276x3 double
gauss_kernal	[0.0751,0.1238,0....
gn	182x276x3 double
h	[0.1111,0.1111,0....
i	181
img	182x276x3 double
img_p	182x276x3 double
img_scaled	182x276x3 double
j	275
k	3

Result Discussion

1. Mean Filter: Reduces Gaussian noise but blurs edges significantly.
2. Gaussian Filter: Preserves structure better than mean filter while reducing Gaussian noise.
3. Median Filter: Very effective for Salt-and-Pepper noise; preserves edges and removes noise thoroughly.