Lab Report 6

Aim

Using different mean filters for removal of noise in MATLAB.

Theory

Different mean filters can be used to remove noise from an image. The most common types of mean filters are the arithmetic mean filter, geometric mean filter, and harmonic mean filter.

Arithmetic mean filter: This filter replaces each pixel with the average value of the neighboring pixels. It is a simple and effective filter for removing random noise. However, it can blur edges and reduce image details.

Geometric mean filter: This filter replaces each pixel with the geometric mean of the neighboring pixels. It is effective at removing multiplicative noise, such as speckle noise in ultrasound or radar images. However, it can also blur the image if the kernel size is too large.

Harmonic mean filter: This filter replaces each pixel with the harmonic mean of the neighboring pixels. It is effective at removing salt-and-pepper noise and preserving edges, but it can produce artifacts in uniform areas of the image.

Alpha-trimmed mean filter: This filter removes a certain percentage of the highest and lowest pixel values in the neighbourhood before calculating the mean. It is effective at removing impulse noise and preserving edges and details, but it may also introduce some smoothing.

Adaptive local noise reduction filter: This filter estimates the local noise level in the image and adapts the filter parameters accordingly. It is effective at removing noise while preserving details and edges.

Midpoint filter: This filter replaces each pixel with the midpoint value of the neighboring pixels. It is effective at removing noise while preserving edges and details, but it may also produce some smoothing.

Contraharmonic mean filter: This filter replaces each pixel with the Contraharmonic mean of the neighboring pixels. It is effective at removing noise of a certain type, such as Gaussian or impulse noise, but it may also produce some artifacts and reduce image sharpness.

To apply a mean filter to an image, a kernel or mask of a specified size is moved over each pixel in the image. The value of each pixel in the new image is then calculated as the mean value of the values of all the pixels within the kernel. The size of the kernel determines the amount of smoothing and the level of detail preservation.

It is important to note that while mean filters are effective at removing noise, they may also introduce unwanted artifacts and reduce image sharpness. It is therefore important to carefully choose the filter type and parameters to achieve the desired level of noise reduction while preserving image details.

CODE

```
% ARITHMATIC MEAN FILTER


clc;clear;close all;
i=uigetfile('*.*');
```

```
I=imread(i);
J=rgb2gray(I);
subplot(1,2,1);
imshow(I);title("Original noisy image");
K=padarray(J,[1,1],0);
K=double(K);
[rows,columns]=size(K);
L=zeros(rows,columns);
for r=2:rows-1
    for c=2:columns-1
        filt=[K(r,c),K(r-1,c),K(r+1,c),K(r,c-1),K(r,c+1),K(r-1,c-
1),K(r-1,c+1),K(r+1,c-1),K(r+1,c+1)];
        x=mean(filt);
        L(r,c)=x;
    end
end
subplot(1,2,2);
M=uint8(L);
imshow(M );title("Final image using arithmetic mean filter");
```

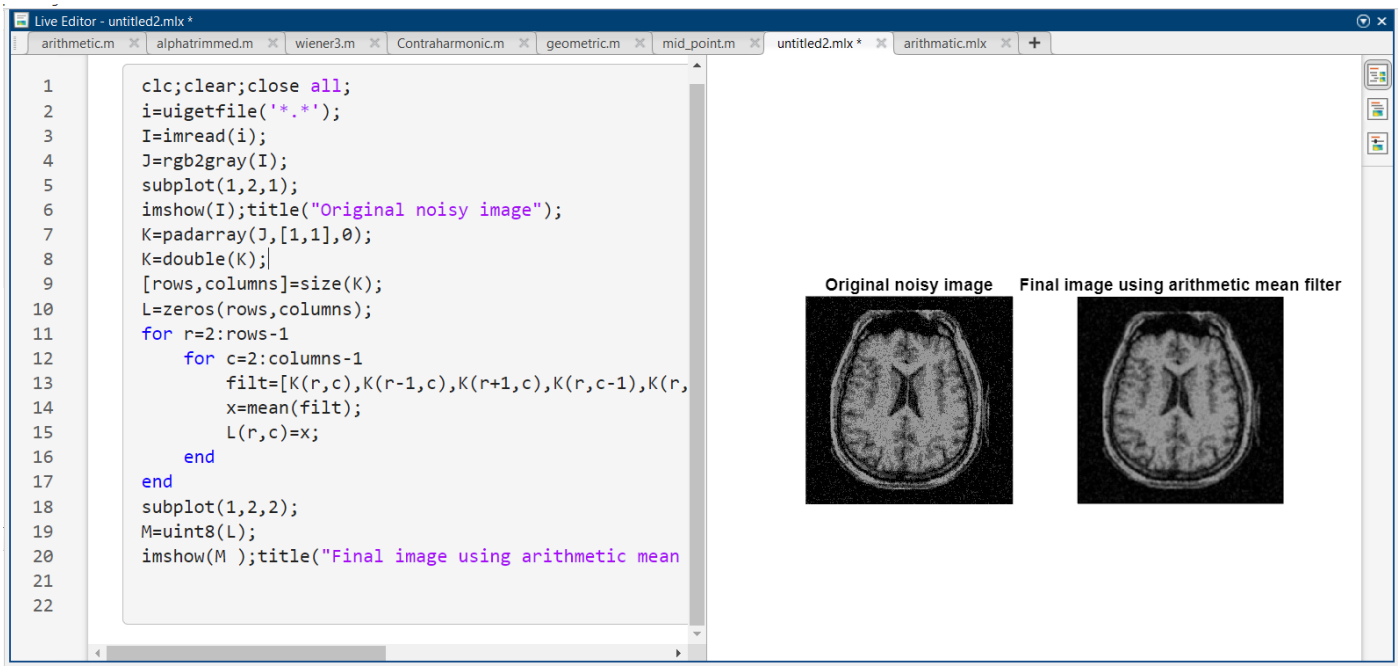

Figure 1: use of arithmetic mean filter to process image in MATLAB

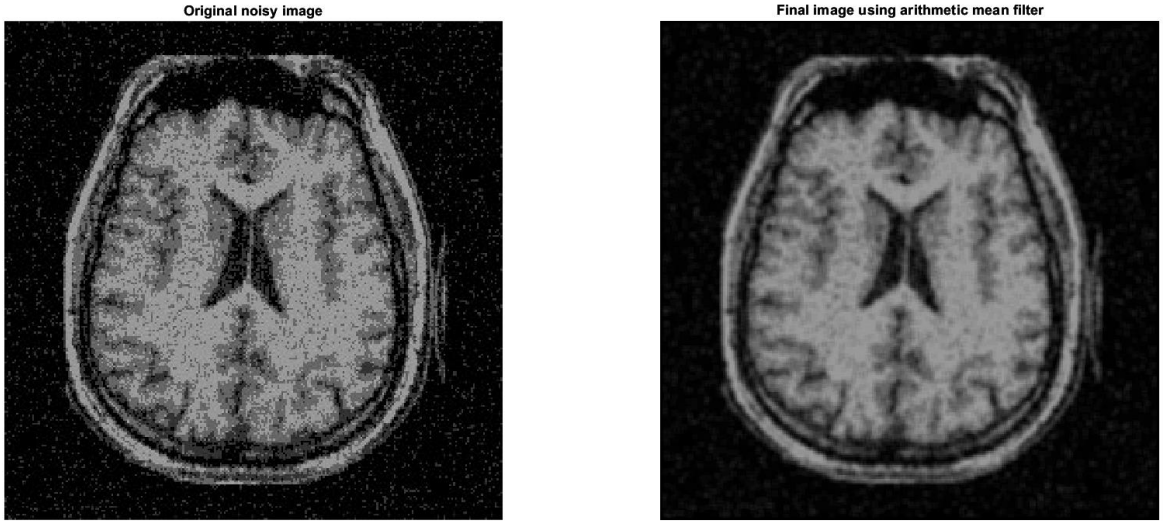

Figure 2: image processing using arithmetic mean filter in MATLAB

CODE

```
% GEOMETRIC MEAN FILTER
clc;clear;close all;
i=uigetfile('*.*');
I=imread(i);
J=rgb2gray(I);
subplot(1,2,1);
imshow(J);title("Original noisy image");
K=padarray(J,[1,1],1);
K=double(K);
[rows,columns]=size(K);
L=zeros(rows,columns);
for r=2:rows-1
    for c=2:columns-1
        filt=[K(r,c),K(r-1,c),K(r+1,c),K(r,c-1),K(r,c+1),K(r-1,c-
1),K(r-1,c+1),K(r+1,c-1),K(r+1,c+1)];
        y=prod(filt);
        x=power(y,1./9);
        L(r,c)=x;
    end
end
subplot(1,2,2);
M=uint8(L);
imshow(M,[]);title("Final image using Geometric mean filter");
```

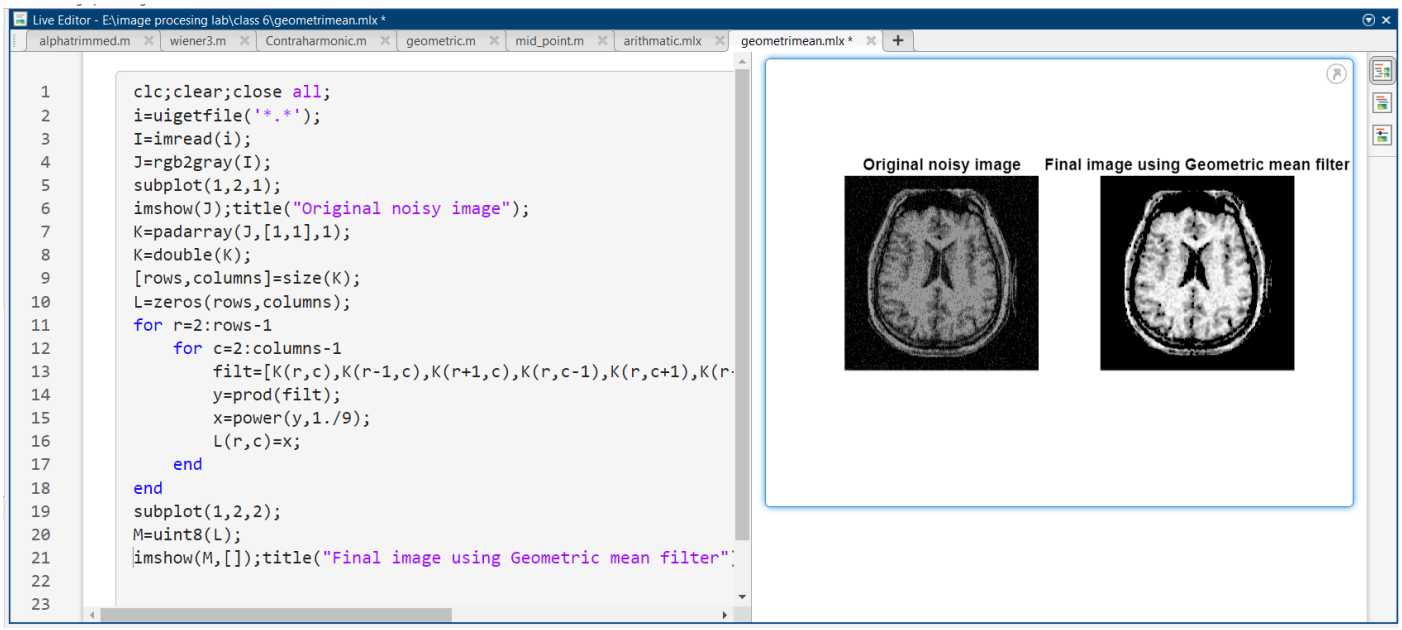

Figure 3: use of geometric mean filter to process image in MATLAB

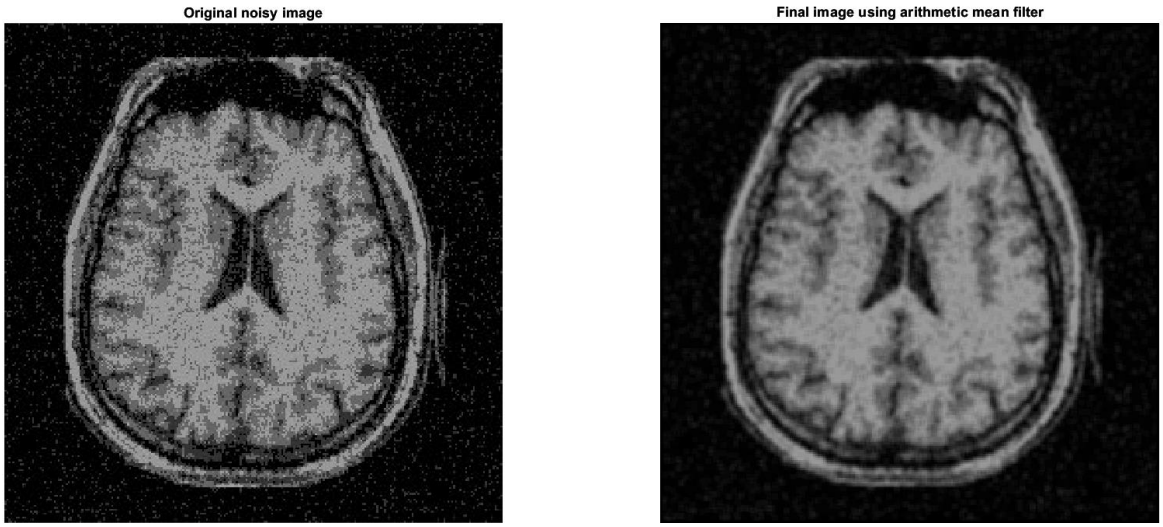

Figure 4: image processing using geometric mean filter in MATLAB

CODE

```
% MID POINT FILTER
clc;clear;close all;
i=uigetfile('*.*');
I=imread(i);
J=rgb2gray(I);
subplot(1,2,1);
imshow(I);title("Original noisy image");
```

```
[rows,columns]=size(J);
K=padarray(J,[1,1],0);
L=zeros(rows,columns);
for r=2:rows-1
    for c=2:columns-1
        filt=[K(r,c),K(r-1,c),K(r+1,c),K(r,c-1),K(r,c+1),K(r-1,c-
1),K(r-1,c+1),K(r+1,c-1),K(r+1,c+1)];
        y1=min(filt); y2=max(filt); x=(y1+y2)./2; L(r-1,c-1)=x;
    end
end
subplot(1,2,2);
M=uint8(L);
imshow(M );title("Final image using Mid_Point filter");
```

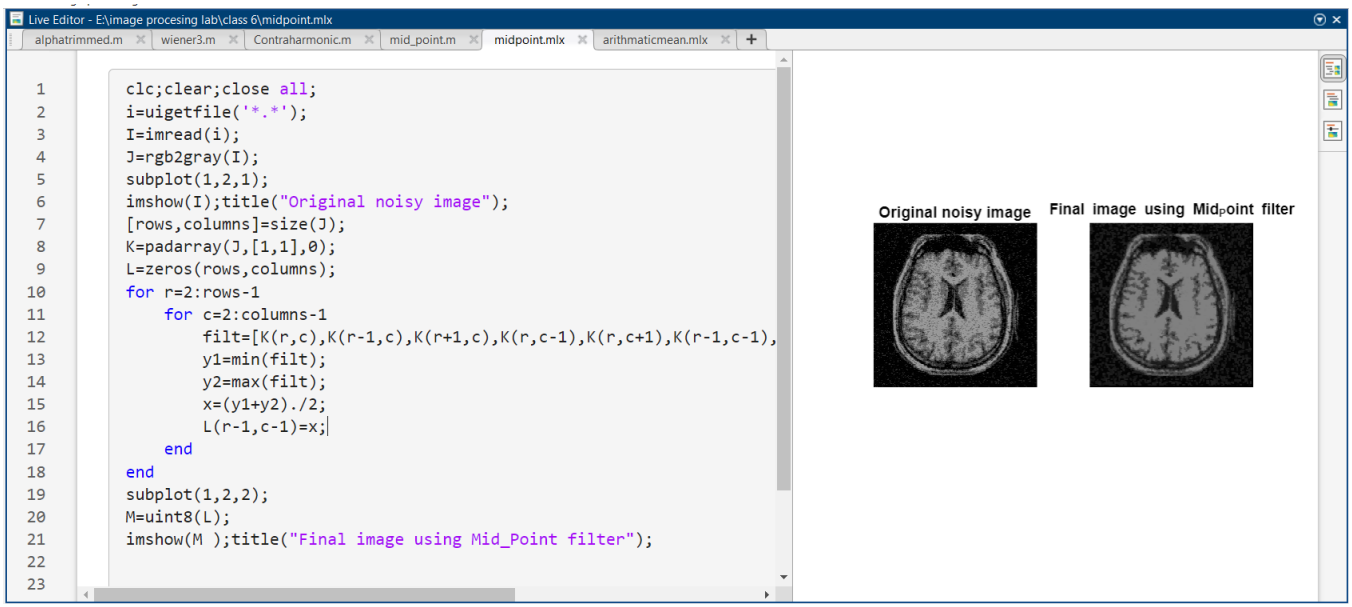

Figure 5: use of mid point filter to process image in MATLAB

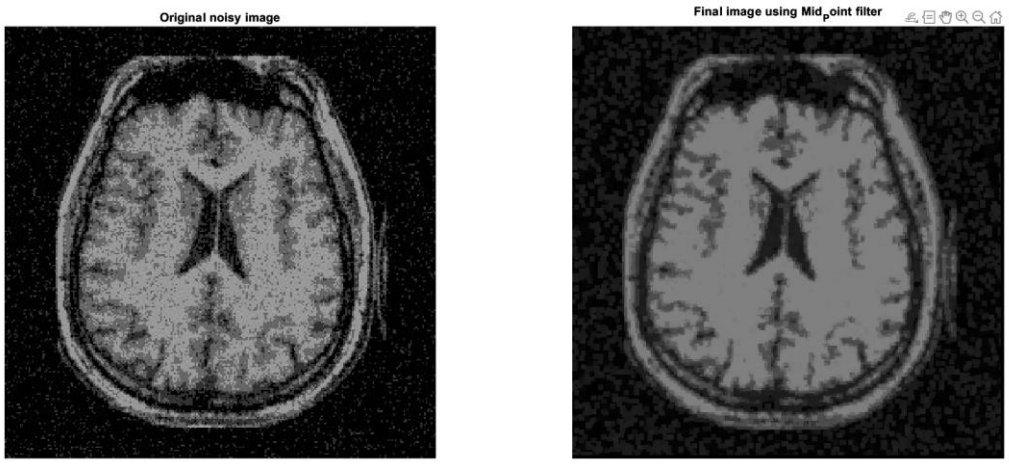

Figure 6: image processing using mid point filter in MATLAB

CODE

```
% HARMONIC MEAN FILTER
clc;clear;close all;
i=uigetfile('*.*');
I=imread(i);
J=rgb2gray(I);
subplot(1,2,1);
imshow(J);title("Original noisy image");
K=padarray(J,[1,1],1);
K=double(K);
[rows,columns]=size(K);
L=zeros(rows,columns);

for r=2:rows-1  for c=2:columns-1
        filt=[K(r,c),K(r-1,c),K(r+1,c),K(r,c-1),K(r,c+1),K(r-1,c-
1),K(r-1,c+1),K(r+1,c-1),K(r+1,c+1)];
        s=sum(1./filt);
        x=9./s;
        L(r,c)=double(x); end end
subplot(1,2,2);
M=uint8(L);
imshow(M,[]);
title("Final image using harmonic mean filter");
```

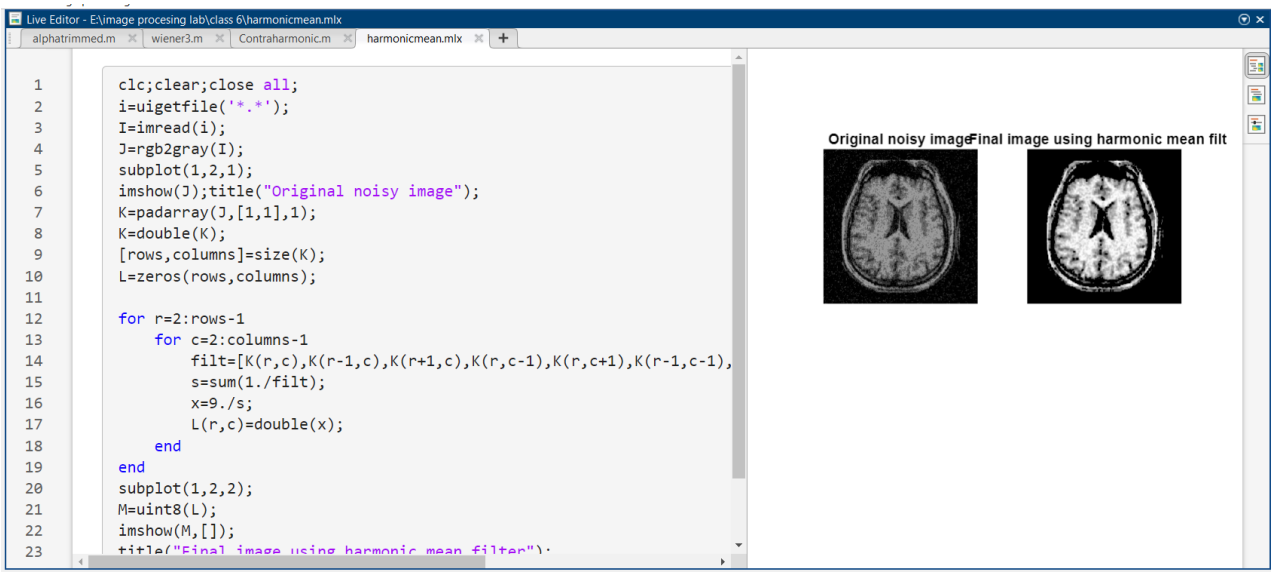

Figure 7: use of harmonic mean filter to process image in MATLAB

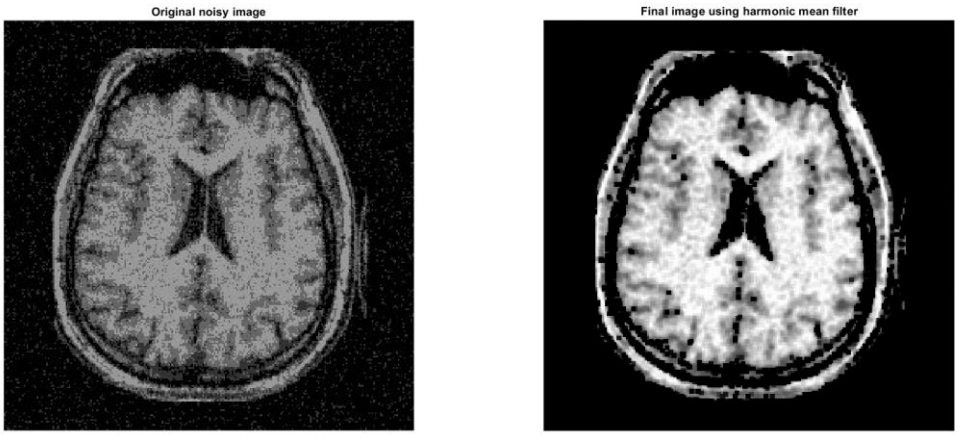

Figure 8: image processing using harmonic mean filter in MATLAB

CODE

```
% CONTRA HARMONIC MEAN FILTER
clc;clear;close all;
i=uigetfile('*.*');
I=imread(i);
J=rgb2gray(I);
subplot(1,2,1);
imshow(I);title("Original noisy image");
K=padarray(J,[1,1],0);
K=double(K);
[rows,columns]=size(K);
L=zeros(rows,columns);
q=-.5;
for r=2:rows-1
    for c=2:columns-1
        filt=[K(r,c),K(r-1,c),K(r+1,c),K(r,c-1),K(r,c+1),K(r-1,c-
1),K(r-1,c+1),K(r+1,c-1),K(r+1,c+1)];
        x=sum(filt.^(q+1) );
        y=sum(filt.^q );
        z=x/y;
        L(r,c)=z;
    end
end
subplot(1,2,2);
M=uint8(L);
imshow(M );title("Final image using Contra-harmonic mean filter");
```

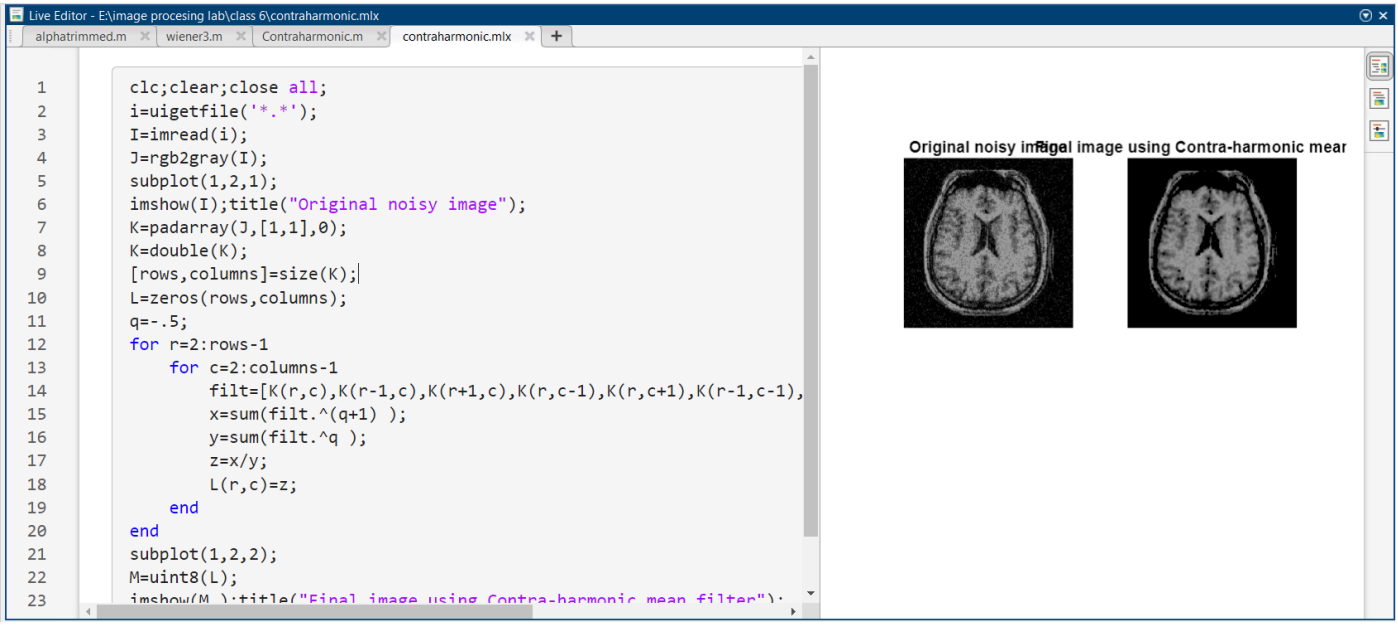

Figure 9: use of contra harmonic mean filter to process image in MATLAB

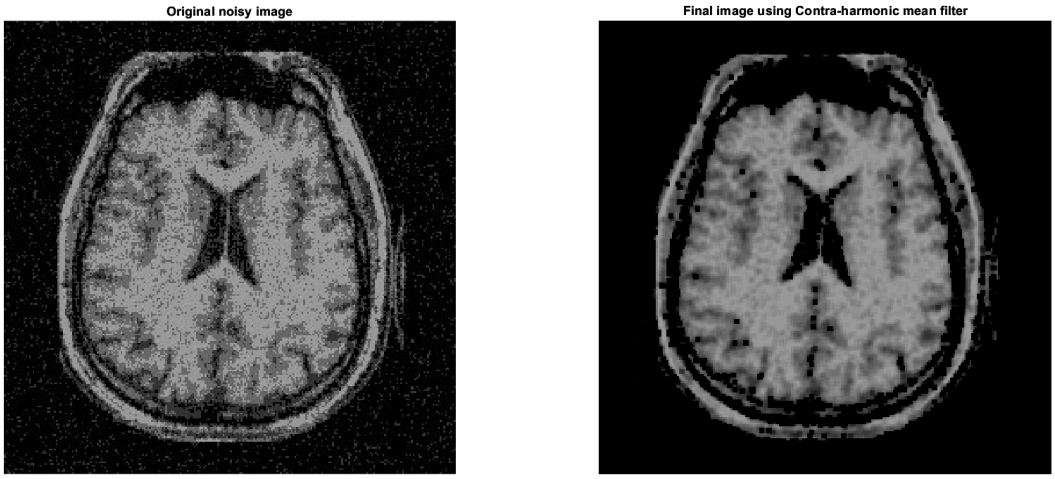

Figure 10: image processing using contra harmonic mean filter in MATLAB

CODE

```
% ALPHA TRIMED FILTER
clc;clear;close all;
i=uigetfile('*.*');
I=imread(i);
J=rgb2gray(I);
subplot(1,2,1);
imshow(I);title("Original noisy image");
K=padarray(J,[1,1],0);
```

```
K=double(K);
[rows,columns]=size(K);
L=zeros(rows,columns);
alpha=2;
d=2.*alpha;
for r=2:rows-1
    for c=2:columns-1
        filt=[K(r,c),K(r-1,c),K(r+1,c),K(r,c-1),K(r,c+1),K(r-1,c-
1),K(r-1,c+1),K(r+1,c-1),K(r+1,c+1)];
        sort(filt);
        x=(1./(9-d))*sum(filt(alpha+1 : 9-alpha));
        L(r,c)=x;
    end
end
subplot(1,2,2);
M=uint8(L);
imshow(M );title("Final image using Alpha-trimmed  filter");
```

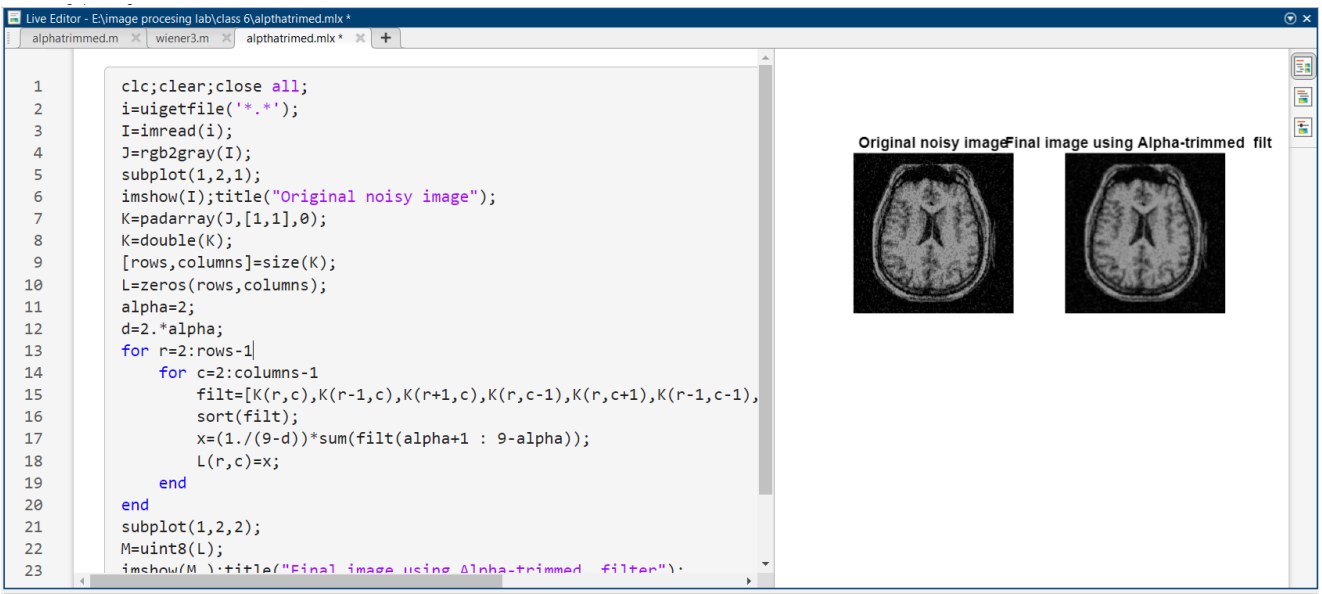

Figure 11: use of alpha trimmed filter to process image in MATLAB

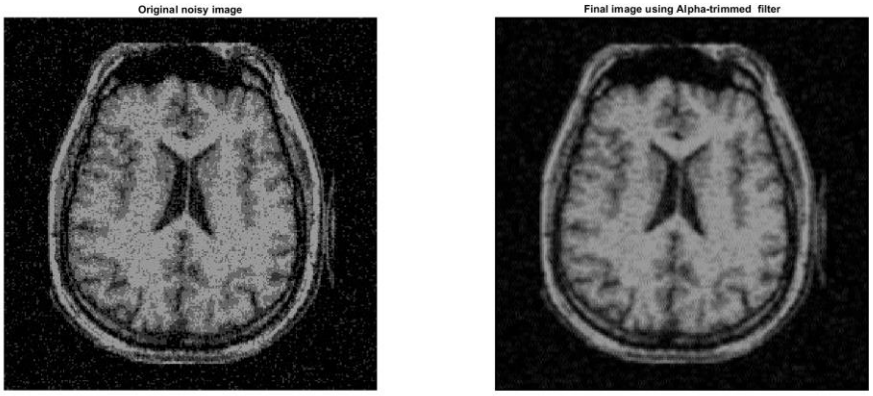

Figure 12: image processing alpha trimmed filter *in MATLAB*

WEINER INVEERSE FILTER

The Weiner inverse filter is a signal processing technique used to restore an image or signal that has been degraded by a known linear time-invariant (LTI) system. The filter is named after Norbert Weiner, who first proposed it in the 1940s. The Weiner inverse filter is designed to remove the effects of degradation caused by an LTI system, such as blurring, distortion, or noise, by filtering the degraded signal. The filter works by estimating the original signal's frequency spectrum and then applying a weighted inverse filter to the degraded signal. The filter's design involves two main steps: estimation of the power spectral density (PSD) of the original signal and estimation of the PSD of the degradation process. The PSD estimates are then used to derive a filter that minimizes the mean square error between the original and filtered signals. The Weiner inverse filter is an optimal linear filter that takes into account the statistical properties of the original signal and the noise present in the degraded signal. It is a powerful tool for signal restoration but can be sensitive to errors in the PSD estimates, which can lead to instability and noise amplification. The Weiner inverse filter is widely used in image processing, audio signal processing, and other fields where signal restoration is necessary. However, it is important to note that the filter is only effective when the degradation process is known and can be modelled accurately.

CODE

```
% WEINER INVEERSE FILTER
clc;clear;close all;
i=uigetfile('*.*');
I=imread(i);
f1=rgb2gray(I);
f=double(f1);

[rows,columns ]=size(f);
% Create a blurred and noisy version of the image
```

```
h = ones(3) / 9;%degradation function
F=fft2(f);
H=fft2(h,rows,columns);
N=25*randn(rows,columns);

G=F.*H ;
g=ifft2(G)+N;
subplot(1,3,1);imshow(f1);title("Original image");
subplot(1,3,2);imshow(uint8(g),[]);title("Degraded Image");

wfilter=(conj(H)./((abs(H).^2)+(abs(N).^2)./(abs(F).^2)));

final=wfilter.*G;
final1=ifft2(final);
subplot(1,3,3);imshow(uint8(final1),[]);title("Restored Image");
```

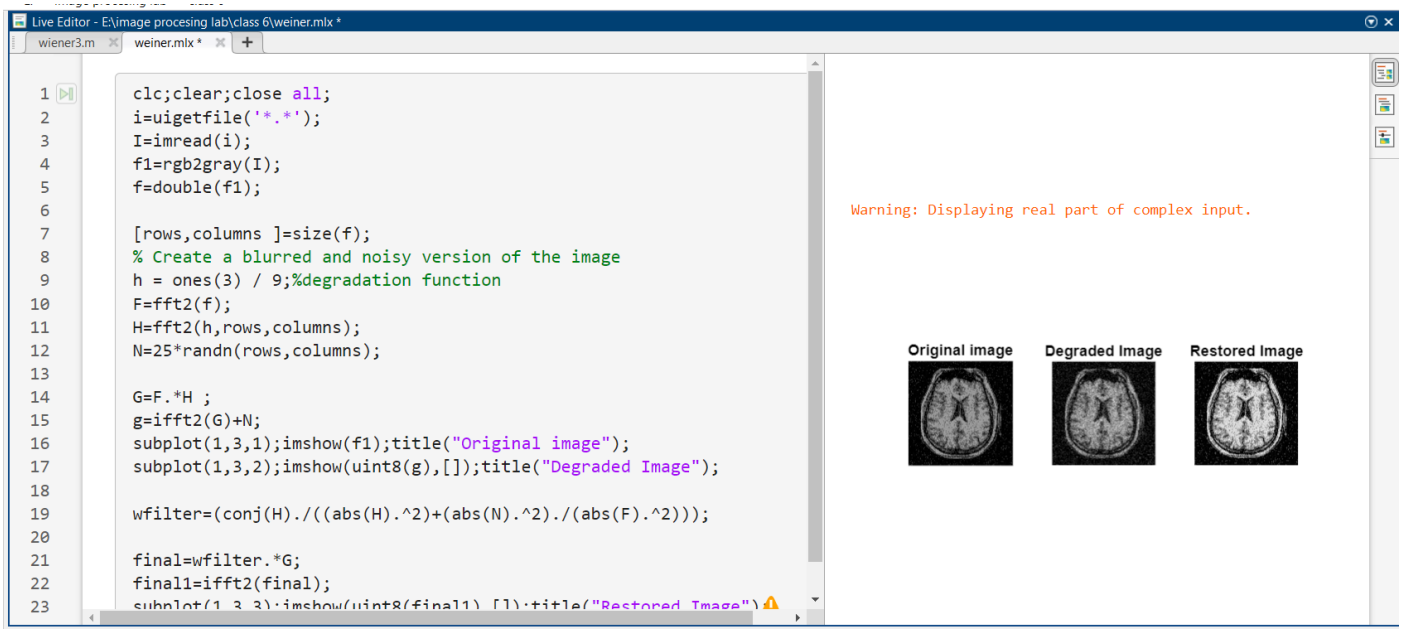

Figure 13: use of weiner inverse filter to process image in MATLAB

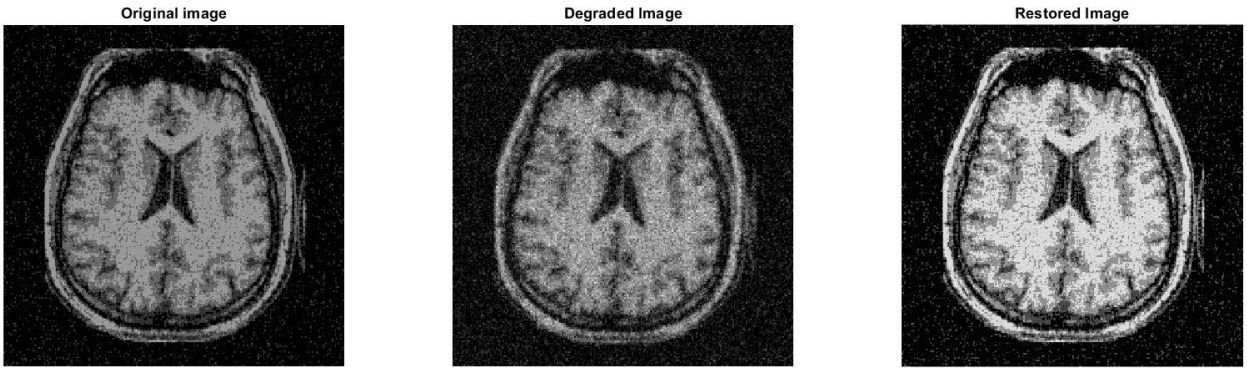

Figure 14: image processing weiner inverse filter in MATLAB

Shreenandan Sahu |120BM0806