“Heaven’s Light is Our Guide ”

Rajshahi University of Engineering & Technology

Submitted To:

Rizoan Toufiq

Assistant professor of

Department of Computer Science & Engineering

Course Title : Digital Image Processing

Course No : CSE 4106

Submitted By:

Sabit Ahmed

Roll:1503056

Sec: A

Department of Computer Science & Engineering

Rajshahi University of Engineering & Technology

5. Spatial Filtering

5.1 Smoothing Filters

Smoothing filters are used for blurring and noise reduction. Blurring is used in preprocessing tasks, such as removal of small details from an image prior to (large) object extraction, and bridging of small gaps in lines or curves. Noise reduction can be accomplished by blurring with a linear filter and also by nonlinear filter.

5.1.1 Gaussian Filter

The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian (`bell-shaped') hump. The steps are following:
Design the kernel by the equation:

$$\frac{1}{2\pi\sigma^2}.e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

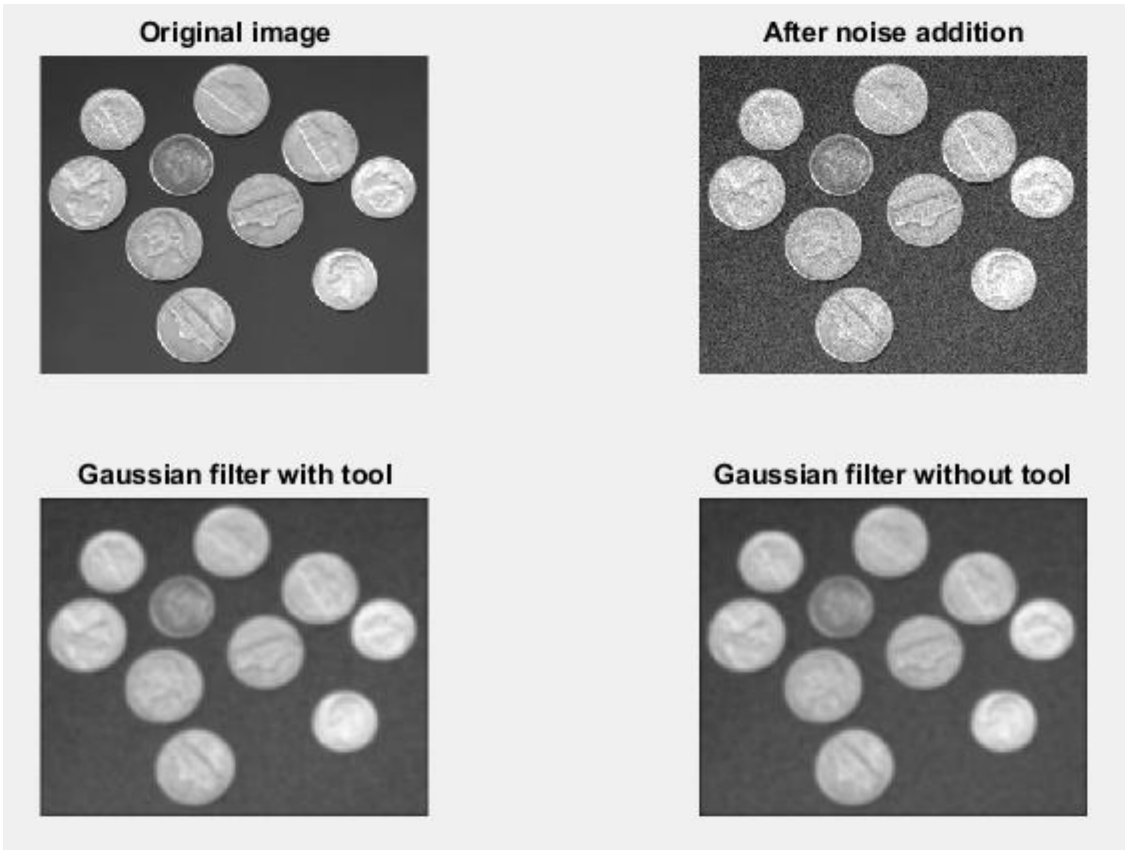
Then convolve the kernel and the local region in the image. On convolution of the local region and the Gaussian kernel gives the highest intensity value to the center part of the local region and the remaining pixels have less intensity as the distance from the center increases.
Sum up the result and store it in the current pixel location and perform the above steps for all part of the region.

Code

```
function gaussian_filter()
%Gaussian filter using MATLAB built_in function
Img = imread('coins.png');
A = imnoise(Img,'Gaussian',0.04,0.003);
figure,imshow(Img);title('Original image');
figure,imshow(A);title('After noise addition');
H = fspecial('Gaussian',[9 9],1.76);
GaussF = imfilter(A,H);
figure,imshow(GaussF);title('Gaussian filter with tool');
%Without built_in function
Img = imread('coins.png');
A = imnoise(Img,'Gaussian',0.04,0.003);
I = double(A);
sigma = 1.76;
sz = 4;
[x,y]=meshgrid(-sz:sz,-sz:sz);

M = size(x,1)-1;
N = size(y,1)-1;
Exp_comp = -(x.^2+y.^2)/(2*sigma*sigma);
```

```
Kernel= exp(Exp_comp)/(2*pi*sigma*sigma);
Output=zeros(size(I));
I = padarray(I,[sz sz]);
for i = 1:size(I,1)-M
    for j =1:size(I,2)-N
        Temp = I(i:i+M,j:j+M).*Kernel;
        Output(i,j)=sum(Temp(:));
    end
end
Output = uint8(Output);
figure,imshow(Output);title('Gaussian filter without tool');
end
```

Output



Discussion

Noise is added with the original image and then use the kernel with convolution and then the output is filtered blurred image. Here, Gaussian filtering with tool and without tool are almost similar with each other as both blurred the noisy image successfully.

5.2 Order-statistic Filters

Order-statistic filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result.

5.2.1 Max and Min Filter

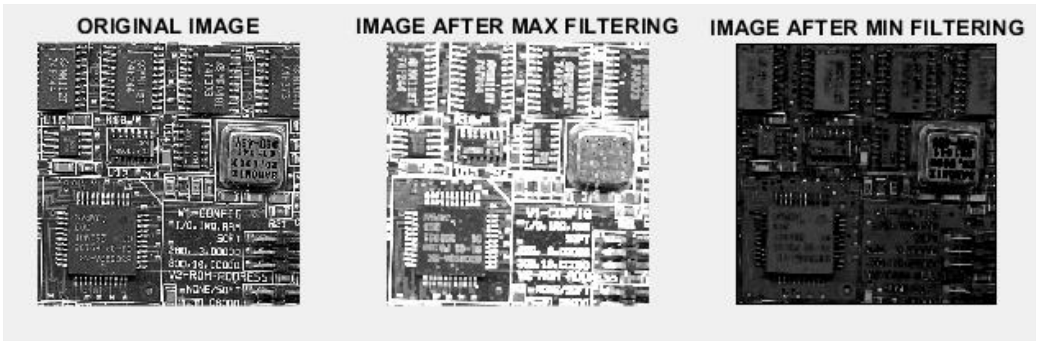
Theory

Max filter is useful for finding the brightest points in an image. Min filter is used for the opposite purpose of max filter. Max filter finds the maximum value in the area encompassed by the filter and reduces the pepper noise as a result of the max operation. These are the opposite for min filter.

Code

```
function max_and_min_filter()
A = imread('board.tif');
A = rgb2gray(A(1:300,1:300,:));
B=zeros(size(A));
C=zeros(size(A));
modifyA=padarray(A,[1 1]);
modifyA=double(modifyA);
        x=[1:3]';
        y=[1:3]';
for i= 1:size(modifyA,1)-2
    for j=1:size(modifyA,2)-2
       window = modifyA(i:i+2,j:j+2);
       B(i,j)=max(window(:));
       C(i,j)=min(window(:));

    end
end
B=uint8(B);
C=uint8(C);
figure;
subplot(2,3,1),imshow(A),title('ORIGINAL IMAGE');
subplot(2,3,2),imshow(B),title('IMAGE AFTER MAX FILTERING');
subplot(2,3,3),imshow(C),title('IMAGE AFTER MIN FILTERING');
```

Output



Discussion

We can see from the figure, after max filtering it has become brighter as it finds the maximum values from an image and the opposite has occurred after min filtering.

5.2.2 Median Filter

Theory

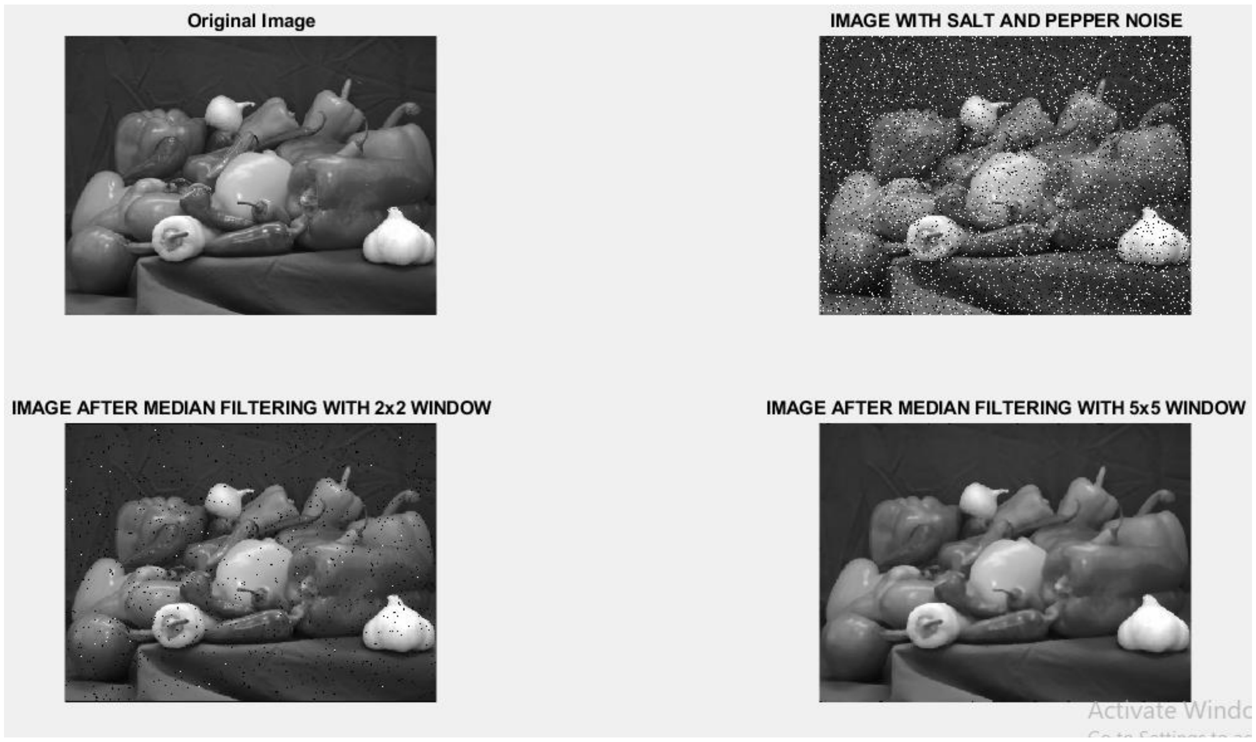
A median filter is a nonlinear filter used for signal smoothing. It is particularly good for removing impulsive type noise from a signal. The main idea of the median filter is to run through the matrix entry by entry, replacing each entry with the median of neighboring entries.

Code

```
I = imread('peppers.png');
I = rgb2gray(I);
A = imnoise(I,'Salt & pepper',0.1);
M1=2;
N1=2;
modifyA1=padarray(A,[floor(M1/2),floor(N1/2)]);
B = zeros([size(A,1) size(A,2)]);
med_indx = round((M1*N1)/2);
for i=1:size(modifyA1,1)-(M1-1)
    for j=1:size(modifyA1,2)-(N1-1)
        temp=modifyA1(i:i+(M1-1),j:j+(N1-1),:);
        tmp_sort = sort(temp(:));
        B(i,j) = tmp_sort(med_indx);
    end
end
M2=5;
N2=5;
modifyA2=padarray(A,[floor(M2/2),floor(N2/2)]);
C = zeros([size(A,1) size(A,2)]);
med_indx = round((M2*N2)/2);

for i=1:size(modifyA2,1)-(M2-1)
    for j=1:size(modifyA2,2)-(N2-1)


        temp=modifyA2(i:i+(M2-1),j:j+(N2-1));
        tmp_sort = sort(temp(:));
        C(i,j) = tmp_sort(med_indx);




    end
end
B=uint8(B);
C=uint8(C);
figure;
```

```
subplot(2,2,1),imshow(I),title('Original Image');
subplot(2,2,2),imshow(A),title('IMAGE WITH SALT AND PEPPER NOISE');
subplot(2,2,3),imshow(B),title('IMAGE AFTER MEDIAN FILTERING WITH 2x2 WINDOW');
subplot(2,2,4),imshow(C),title('IMAGE AFTER MEDIAN FILTERING WITH 5x5 WINDOW');
```

Output


Original Image
IMAGE WITH SALT AND PEPPER NOISE
IMAGE AFTER MEDIAN FILTERING WITH 2x2 WINDOW
IMAGE AFTER MEDIAN FILTERING WITH 5x5 WINDOW
Activate Windo

Discussion

We can see from the figure, the image filtered with 2x2 window still contains salt and pepper noise but the edges are still sharp. But the image filtered with 5x5 window is free of noise but the edges are smoothed to certain extent.

5.3 Sharpening Spatial Filters

Image sharpening is an effect applied to digital images to give them a sharper appearance. It increases the contrast between bright and dark regions to bring out features.

5.3.1 Image Sharpening using second order derivative (Laplacian)

Theory

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection. The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single gray level image as input and produces another gray level image as output.

Laplacian equation is defined by:

$$g(x,y)=f(x,y)+c\left[\Delta^2 f(x,y)\right]$$

Fig F

Where f(x,y) is the input image g(x,y) is the sharpened image and c= -1 for the above mentioned filter masks and the derivative operator Laplacian for an Image is defined as:

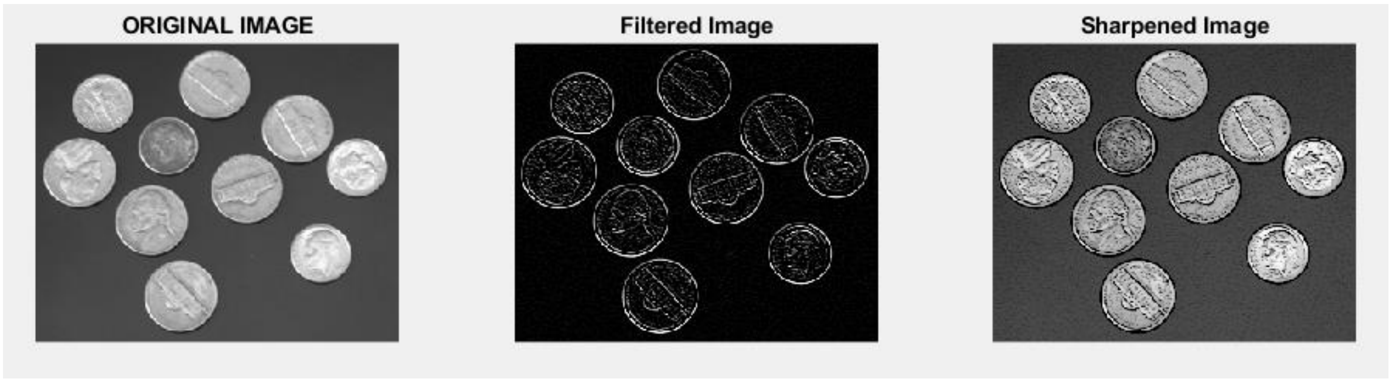
$$\Delta^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Fig A

Code

```
function laplacian_image_sharpening()
A=imread('coins.png');
figure,imshow(A);
I1=A;
I=zeros(size(A));
I2=zeros(size(A));
F1=[0 1 0;1 -4 1; 0 1 0];
F2=[1 1 1;1 -8 1; 1 1 1];
A=padarray(A,[1,1]);
A=double(A);
for i=1:size(A,1)-2
    for j=1:size(A,2)-2
        I(i,j)=sum(sum(F1.*A(i:i+2,j:j+2)));
    end
end
I=uint8(I);
figure,imshow(I);title('Filtered Image');
B=I1-I;
figure,imshow(B);title('Sharpened Image');
end
```

Output



Discussion

The above figure, the Laplacian derivative equation has produced grayish edge lines and other areas are made dark (background). The Filter Image is combined with the original input image thus the background is preserved and the sharpened image is obtained.

5.3.2 Unsharp Masking and Highboost Filtering

Theory

The unsharp filter is a simple sharpening operator which derives its name from the fact that it enhances edges (and other high frequency components in an image) via a procedure which subtracts an unsharp, or smoothed, version of an image from the original image. Unsharp masking consists of blurring the original image, subtracting the blurred image from the original (the resulting difference is called the mask) and adding the mask to the original.

Mask is obtained by:

$$g_{mask}(x,y) = f(x,y) - \bar{f}(x,y)$$

And after adding the mask, the equation is:

$$g(x,y) = f(x,y) + k \times g_{mask}(x,y)$$

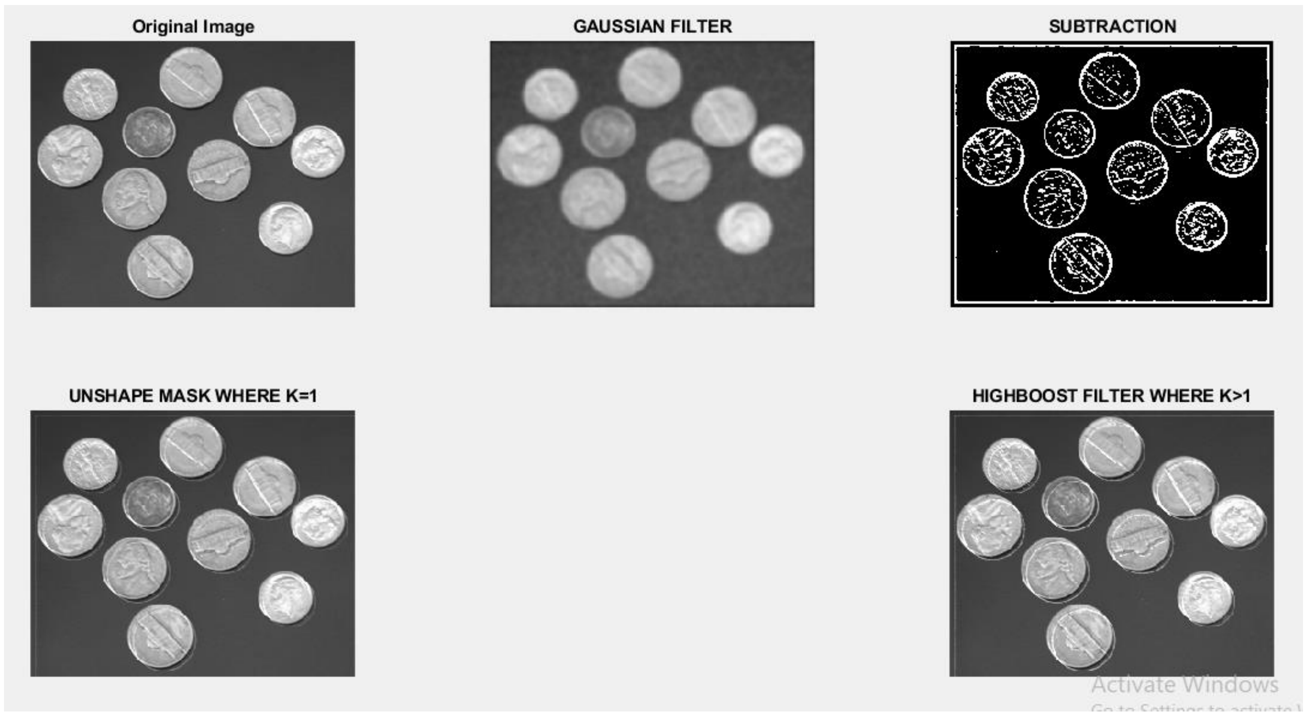
Where k=1. When k > 1, the process is referred to as **Highboost Filtering**.

Code

```
function unsharp_mask()
Img = imread('coins.png');
A = imnoise(Img,'Gaussian',0.04,0.003);
I = double(A);
sigma = 1.76;
```

```
sz = 4;
[x,y]=meshgrid(-sz:sz,-sz:sz);

M = size(x,1)-1;
N = size(y,1)-1;
Exp_comp = -(x.^2+y.^2)/(2*sigma*sigma);
Kernel= exp(Exp_comp)/(2*pi*sigma*sigma);
Output=zeros(size(I));
I = padarray(I,[sz sz]);
for i = 1:size(I,1)-M
    for j =1:size(I,2)-N
        Temp = I(i:i+M,j:j+M).*Kernel;
        Output(i,j)=sum(Temp(:));
    end
end
Output = uint8(Output);
Output2=Img-Output;
Output2 = padarray(Output2,[sz sz]);
Output2=double(Output2);
Img1=double(Img);
Output3=zeros(size(I));
k=1;
for i = 1:size(Img1,1)-M
    for j =1:size(Img1,2)-N
        Temp(i:i+M,j:j+M) = Img1(i:i+M,j:j+M)+k.*Output2(i:i+M,j:j+M);
        Output3=Temp;
    end
end
Output4=zeros(size(I));
k=2;
for i = 1:size(Img1,1)-M
    for j =1:size(Img1,2)-N
        Temp(i:i+M,j:j+M) = k.*Output2(i:i+M,j:j+M);
    end
end
Output4=double(Img)+Temp;
Output3 = uint8(Output3);
Output4 = uint8(Output4);
figure,imshow(Img),title('Original Image');
figure;
subplot(2,2,1),imshow(Output),title('GAUSSIAN FILTER');
subplot(2,2,2),imshow(Output2),title('SUBTRACTION');
subplot(2,2,3),imshow(Output3),title('UNSHAPE MASK WHERE K=1');
subplot(2,2,4),imshow(Output4),title('HIGHBOOST FILTER WHERE K>1');
end
```

Output



Discussion

The original image is made into a blurred one with Gaussian Filter and then subtracting it from the original one got the mask needed for unshape masking. Adding the mask with the original image we get **Unshape Mask** where k=1. Similarly, we get **Highboost Filter** if k>1.

6. Filtering in Frequency Domain

6.1 Discrete Fourier Transformation

Theory

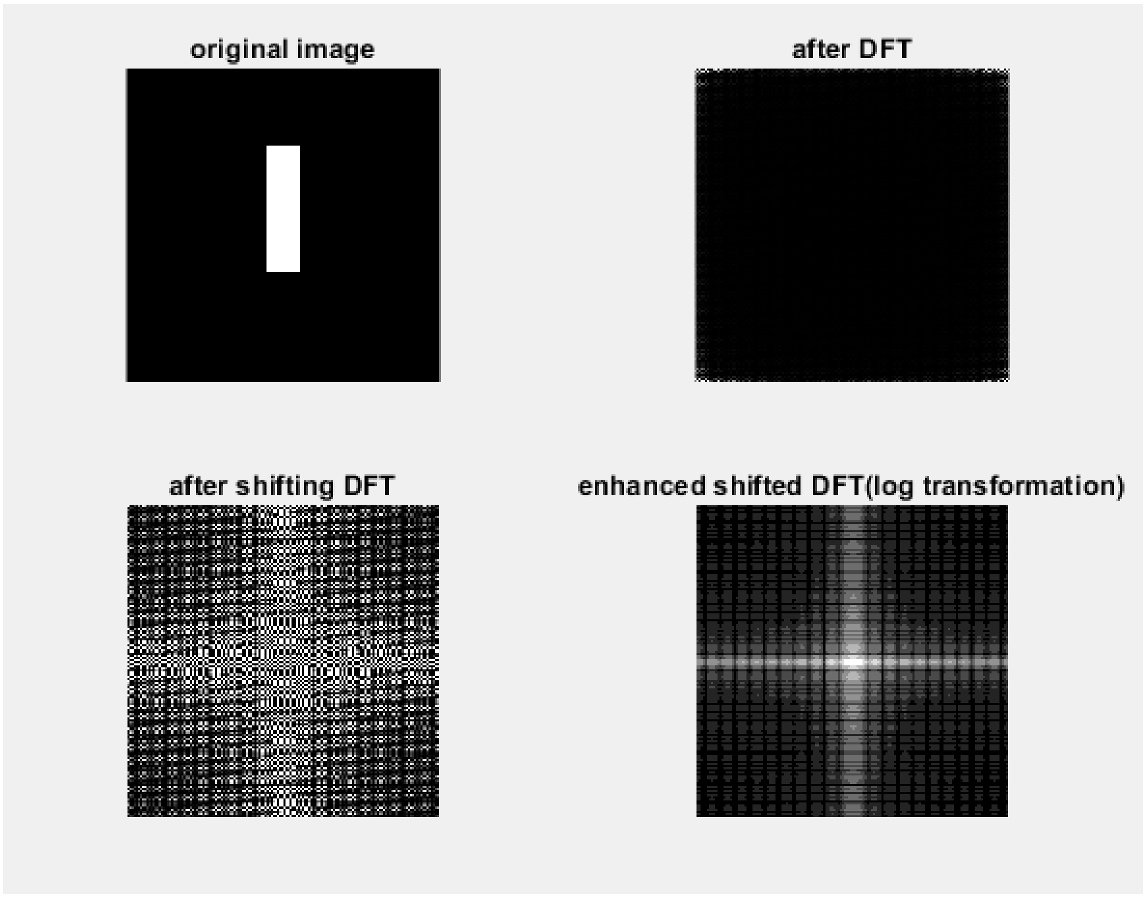
The Discrete Fourier Transform (DFT) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency. The DFT is therefore said to be a frequency domain representation of the original input sequence.

Code

```
function DFT()
f=zeros(200,200);
f(50:130,90:110)=1;
Q=0;
```

```
[M,N]=size(f);
for k=0:1:M-1
    for l=0:1:N-1
        for x=0:1:M-1
            for y=0:1:N-1
                F= f(x+1,y+1)* exp(-1i*2*pi*(k*x/M + l*y/N));
                Q=Q+F;
            end
        end
        im(k+1,l+1)=Q;
        Q=0;
    end
end
subplot(2,2,1),imshow(f);
title('original image');
subplot(2,2,2),imshow(uint8(im));
title('after DFT');
F=fftshift(im);
subplot(2,2,3),imshow(F);
title('after shifting DFT');
subplot(2,2,4),imshow(uint8(log(1+abs(F))),[]);
title('enhanced shifted DFT(log transformation)');
end
```

Output



Discussion

At first the original image is shown. Then the spectrum showing the bright spots around the four corners of the second image. The third image is of centered spectrum. The final is the result showing the increased detail after a log transformation.

6.2 Image Smoothing using Frequency Domain Filters

Smoothing (blurring) is achieved in the frequency domain by lowpass filtering. A lowpass filter passes only the low frequency of an image. There are three types of lowpass filters: Ideal, Butterworth and Gaussian lowpass filter.

6.2.1 Butterworth Lowpass Filter

Theory

The Butterworth filter has a parameter called the filter order. For high order values, the Butterworth filter approaches the ideal filter. For lower order values, the Butterworth filter is more like a Gaussian filter. Butterworth filter mask is obtained by:

$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}}$$

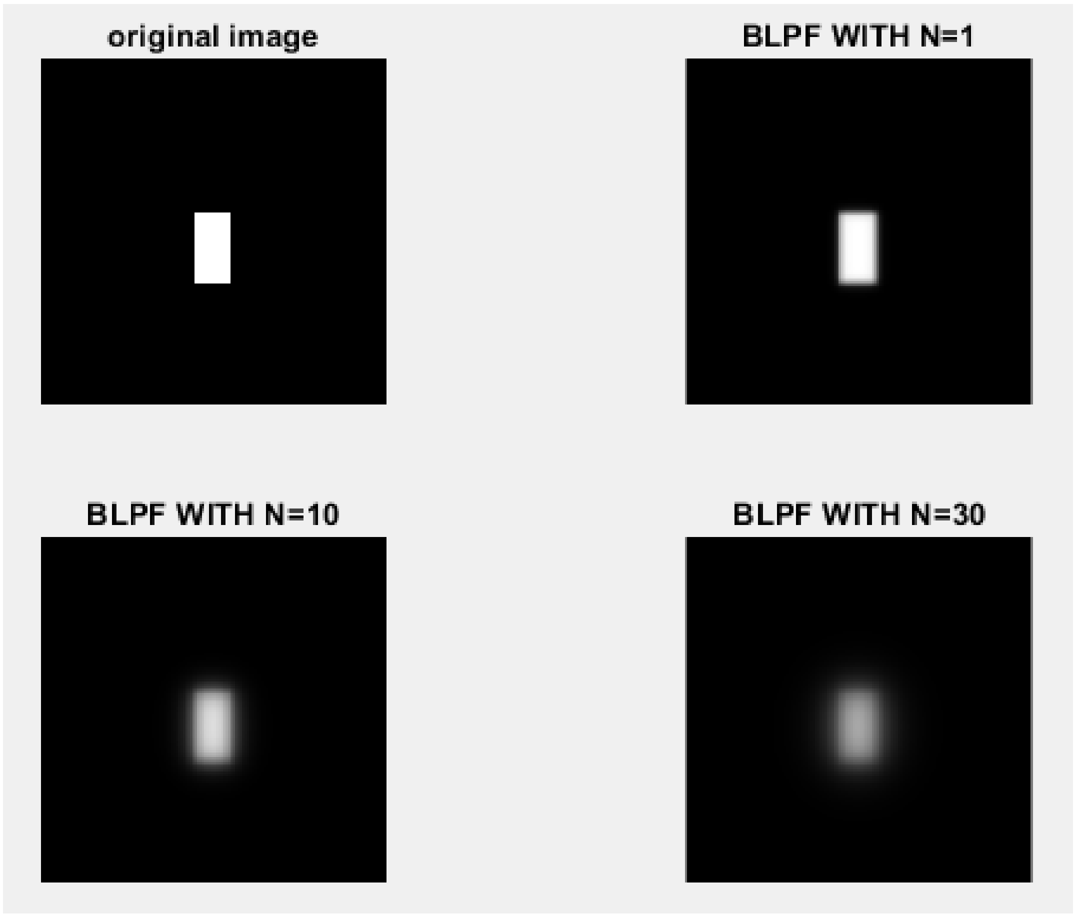
Where D(u,v) is the distance from (0,0) to (u,v), D0 is the cut-off frequency and n is the order of the filter.

Code

```
function butterworth_filter()
f=zeros(200,200);
f(50:130,90:110)=1;
fd=double(f);
[m,n]=size(f);
F1=0;
for u=0:1:m-1
    for v=0:1:n-1
        for x=0:1:m-1
            for y=0:1:n-1
                F= fd(x+1,y+1)* exp(-1i*2*pi*(u*x/m + v*y/n));
                F1=F1+F;
            end
         end
        im(u+1,v+1)=F1;
        F1=0;
    end
end
F=fftshift(im);
[P,Q]=size(F);
D0=20;
```

```
n1=1;
n2=10;
n3=30;
for u=0:P-1
   for v=0:Q-1
        D(u+1,v+1)=(((u-(P/2)).^2)+((v-(Q/2)).^2)).^0.5;
        H1(u+1,v+1)=1/(1+(D(u+1,v+1)/D0).^2*n1);
        H2(u+1,v+1)=1/(1+(D(u+1,v+1)/D0).^2*n2);
        H3(u+1,v+1)=1/(1+(D(u+1,v+1)/D0).^2*n3);
   end
end
result1=F.*H1;
r1=ifft2(result1);
result2=F.*H2;
r2=ifft2(result2);
result3=F.*H3;
r3=ifft2(result3);
subplot(3,2,2),imshow(abs(r1));title('BLPF WITH N=1');
subplot(3,2,3),imshow(abs(r2));title('BLPF WITH N=10');
subplot(3,2,4),imshow(abs(r3));title('BLPF WITH N=30');
subplot(3,2,1),imshow(f);title('original image');
end
```

Output



Discussion

The above figure is spatial representation of Butterworth lowpass filter of order 1, 10 and 30 where the cutoff frequency D0 is 20. The intensity in the center is increasing with the order of the filter. The original image size was of 200x200.