Table of Contents

IVP Assignment 3

```
% Name: Chanakya Ajit Ekbote
% Institute: IIT Bhubaneswar
% Date: 23.10.2020
% Degree: Btech
% Branch: Electronics and Communication
% Roll Number: 17EC01041
```

Creating a new environment

```
clc;
clear all;
close all;
```

Functions Created: sharpening_with_laplacian_filter

```
% Function that sharpens the image using Laplacian via the Frequency
 Domain

function [img] = sharpening_with_laplacian_filter(image)
    % image: We assume that it is a grayscale image.

    dft_image = fftshift(fft2(image));
    [row, col] = size(dft_image);
    mid_row = (1 + row) / 2;
    mid_col = (1 + col) / 2;
    filter = zeros(size(dft_image));

    % Creates the laplacian filter for image sharpening.
    for i = 1:row
```

```
        for j = 1:col
            filter(i, j)  = 1 + 4*pi^2*((mid_row - i)^2 + (mid_col-
j)^2);
        end
    end

    % ifftshift shifts the fft2d back and then perform the idft.
    img = real(ifft2(ifftshift(filter .* double(dft_image))));
```

Functions Created: low_pass_gaussian_filter_smoothening

```
% Function that smoothes the image using a low pass gaussian filter

function [img] = low_pass_gaussian_filter_smoothening(image, var)
    % image: We assume that it is a grayscale image.
    % var: Gives the variance of the 2D gaussian distribution.

    dft_image = fftshift(fft2(image));
    [row, col] = size(dft_image);
    mid_row = (1 + row) / 2;
    mid_col = (1 + col) / 2;
    filter = zeros(size(dft_image));

    % Creates the low pass gaussian filter for smoothening.
    for i = 1:row
        for j = 1:col
            filter(i, j) = exp(-((mid_row - i)^2 + (mid_col-j)^2) / (2
 * var^2));
        end
    end

    % ifftshift shifts the fft2d back and then perform the idft.
    img = real(ifft2(ifftshift(filter .* dft_image)));
```

Functions Created: high_pass_gaussian_filter_sharpening

```
% Function that sharpens the image using a high pass gaussian filter

function [img] = high_pass_gaussian_filter_sharpening(image, var)
    % image: We assume that it is a grayscale image.
    % var: Gives the variance of the 2D gaussian distribution.
```

```
    dft_image = fftshift(fft2(image));
    [row, col] = size(dft_image);
    mid_row = (1 + row) / 2;
    mid_col = (1 + col) / 2;
    filter = zeros(size(dft_image));

    % Creates the high pass gaussian filter for sharpening.
    for i = 1:row
        for j = 1:col
            filter(i, j)  = 1- exp(-((mid_row - i)^2 + (mid_col-
j)^2) ...
                / (2 * var^2));
        end
    end

    % ifftshift shifts the fft2d back and then perform the idft.
    img = real(ifft2(ifftshift(filter .* dft_image)));
```

Functions Created: erosion

```
% Function that performs the morphological operation erosion and
 returns
% the processed image.

function [img] = erosion(image, struct_elem)
    % image: Contains the binary image
    % struct_elem: Contains the structuring element upon which the
    % morphological operation is being done. The assumption is that
 the
    % struct_elem is odd.

    % The logic is that if we take the structuring element and
 multiply it
    % with the same window of the original image (element wise) and if
 the
    % sum of that is equal to the sum of all the values in the
 structuring
    % element then it meand A intersection B is true for the entire
 region.
    % Else it is false

    img = uint8(zeros(size(image)));
    [row_img, col_img] = size(image);
    [row_struct_elem, col_struct_elem] = size(struct_elem);

    mid_row = double(uint8((row_struct_elem + 1) / 2));
    mid_col = double(uint8((col_struct_elem + 1) / 2));
    sum_struct_elem = sum(sum(struct_elem));
```

```
    for i = 1:(row_img - row_struct_elem + 1)
        for j = 1:(col_img - col_struct_elem + 1)
            sum_ = image(i:(i+row_struct_elem-1), j:(j
+col_struct_elem-1)) ...
                .* struct_elem;
            sum_ = sum(sum(sum_));
            if (sum_ == sum_struct_elem)
                img((i+mid_row-1), (j+mid_col-1)) = 1;
            end
        end
    end
```

Functions Created: dilation

```
% Function that performs the morphological operation dilation and
 returns
% the processed image.

function [img] = dilation(image, struct_elem)
    % image: Contains the binary image
    % struct_elem: Contains the structuring element upon which the
    % morphological operation is being done. The assumption is that
 the
    % struct_elem is odd.

    % The logic is that if we take the structuring element and
 multiply it
    % with the same window of the original image (element wise) and if
 the
    % sum of that is greater than 0, then it means that A intersection
 B
    % lies in A.

    img = zeros(size(image));
    [row_img, col_img] = size(image);
    [row_struct_elem, col_struct_elem] = size(struct_elem);

    mid_row = double(uint8((row_struct_elem + 1) / 2));
    mid_col = double(uint8((col_struct_elem + 1) / 2));

    for i = 1:(row_img - row_struct_elem + 1)
        for j = 1:(col_img - col_struct_elem + 1)
            sum_ = image(i:i+row_struct_elem-1, j:j
+col_struct_elem-1) ...
                .* struct_elem;
            sum_ = sum(sum(sum_));
            if (sum_ > 0)
                img(i+mid_row-1, j+mid_col-1) = 1;
```

```
            end
        end
    end

end
```

Image Imports

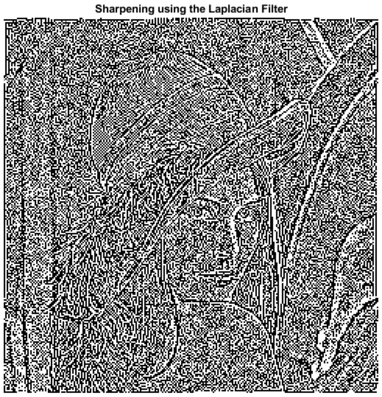
```
lena = imread('C:\Chanakya\Projects\ivp-assignments
\Assignment-3\images\lena_gray_256.tif');
orig_fingerprint = rgb2gray(imread('C:\Chanakya\Projects\ivp-
assignments\Assignment-3\images\fingerprint.jpg'));
```

Sharpening using the Laplacian Filter

The image can be sharpened in both the time domain and the frequency domain. In this case we convert the Laplacian to the frequency domain sharpen it and then convert it back to the time domain via the inverse dft transform. We use the following in the frequency domain:

$$H(u, v) = (1 + 4 * \pi^2 * (u^2 + v^2)(F(u, v))$$

```
% Calling the sharpening_with_laplacian_filter function
sharpened_lena_laplacian = sharpening_with_laplacian_filter(lena);

% Plotting the images
figure('Name', 'Sharpening using Laplacian Filter', 'units', ...
    'normalized','outerposition', [0 0 1 1]);

subplot(1,2,1)
imshow(lena);
title('Original Image');

subplot(1,2,2)
imshow((mat2gray(uint8(sharpened_lena_laplacian))));
title('Sharpening using the Laplacian Filter');
```



Smoothening using the Low Pass Gaussian Filter

The images can be blurred via the frequency domain via the Low Pass Gaussian Filter. The smoothened image can be obtained via the the IDFT after filteration.

$$H(u,v) = exp^{\frac{-D(u,v)^2}{2*D_0^2}} * F(u,v)$$

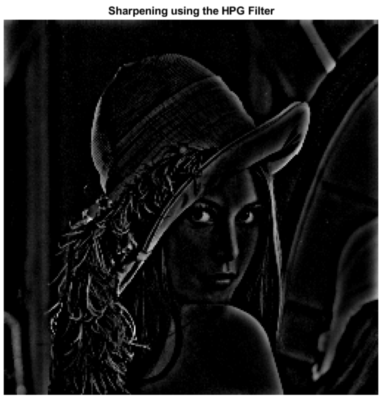
```
% Calling the low_pass_gaussian_filter_smoothening function
smoothened_lena = low_pass_gaussian_filter_smoothening(lena, 10);

% Plotting the images
figure('Name', 'Smoothening using LPG Filter', 'units', ...
    'normalized','outerposition', [0 0 1 1]);

subplot(1,2,1)
imshow(lena);
title('Original Image');

subplot(1,2,2)
imshow((mat2gray(uint8(smoothened_lena))));
title('Smoothening using the LPG Filter');
```

Sharpening Using the High Pass Gaussian Filter

The images can be sharpened via the frequency domain via the High Pass Gaussian Filter. The smoothened image can be obtained via the the IDFT after filteration.

$$H(u,v) = (1 - exp^{\frac{-D(u,v)^2}{2*D_0^2}}) * F(u,v)$$

```
% Calling the high_pass_gaussian_filter_sharpening function
sharpened_lena_hpg =  high_pass_gaussian_filter_sharpening(lena, 10);

% Plotting the images
figure('Name', 'Sharpening using HPG Filter', 'units', ...
    'normalized','outerposition', [0 0 1 1]);

subplot(1,2,1)
imshow(lena);
title('Original Image');

subplot(1,2,2)
imshow((mat2gray(uint8(sharpened_lena_hpg))));
title('Sharpening using the HPG Filter');
```



Removing the noise using Opening: Erosion followed by dilation

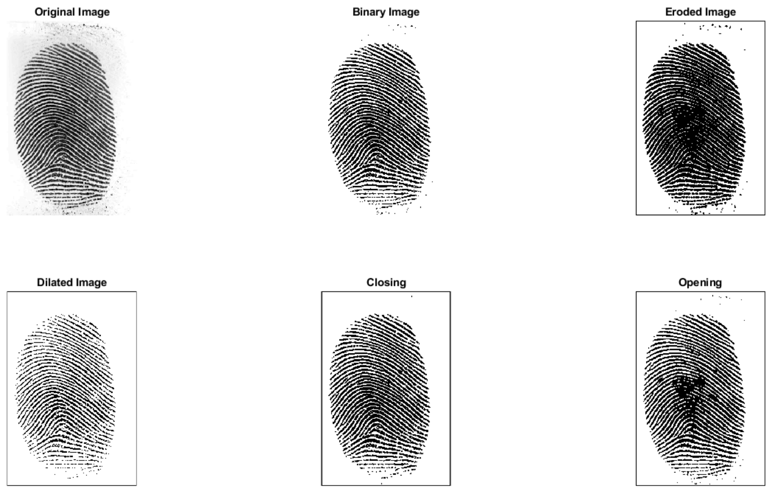
Erosion and Dilation are operations that are done for basically removing unecessary protrusions and filling in holes. Closing is used for filling in holes and then resizing the it back to the original image. Opening is used for removing protrusions and resizing the images back to the original images. The equations for opening and closing are as follows:

$$A \circ B = (A \ominus B) \oplus B$$

$$A \bullet B = (A \oplus B) \ominus B$$

```
% Creating the structuring element
struct_elem = uint8([0, 1, 0; 1, 1, 1; 0, 1, 0]);

% Making the fingerprint image a binary image
fingerprint = uint8(orig_fingerprint>128);

% Calling the erosion and dilation function
eroded_image = erosion(fingerprint, struct_elem);
dilated_image = dilation(fingerprint, struct_elem);

% Closing and Dilation
% Erosion and Dilation and are defined opposite here. That's because
 the
% fingerprint is black and the outer region is white. Therefore, we
 have to
% consider the opposite.
opening = dilation(uint8(erosion(fingerprint, struct_elem)),
 struct_elem);
closing = erosion(uint8(dilation(fingerprint, struct_elem)),
 struct_elem);
```

```matlab
% Plotting the images
figure('Name', 'Erosion and Dilation', 'units', ...
    'normalized','outerposition', [0 0 1 1]);

subplot(2,3,1)
imshow(orig_fingerprint);
title('Original Image');

subplot(2,3,2)
imshow(mat2gray(fingerprint));
title('Binary Image');

subplot(2,3,3)
imshow((mat2gray(uint8(eroded_image))));
title('Eroded Image');

subplot(2,3,4)
imshow((mat2gray(uint8(dilated_image))));
title('Dilated Image');

subplot(2,3,5)
imshow((mat2gray(uint8(closing))));
title('Closing');

subplot(2,3,6)
imshow((mat2gray(uint8(opening))));
title('Opening');
```



Conclusion

From these experiments we can see that we can perform smoothening as well as sharpening via the frequency domain. We can observe that sometimes it is easier to process the image in the frequency domain. Moroever, we can also see that we can use operations like opening and closing to embellish the images.

Published with MATLAB® R2020b