Abu Dhabi University

CEN 464 - Digital Signal Processing

Lab Report 1 & 2

Introduction to Matrix Digital Image Representation and Digital Image Processing

Authors:
Muhammad Obaidullah 1030313

Supervisor:
Dr. Mohammed Assad Ghazal

Section 1

June 8, 2014

Contents

Abstract

In this lab we were introduced to Image Processing Toolbox provided in MATLAB. We were exposed to simple image cropping, various types filtering and applying 2 dimensional Fast Fourier Transform on an image.

1 Objectives of the lab

- Be able to create a matrix using zeros, ones, and the range operator :
- Be able to alter/change some areas in the matrix using the range operator :
- Use items 1 and 2 to build the 2D lowpass, highpass, bandpass, and band reject filters
- Understand the difference between a gray-level and true color images
- Be able to display images and to find the values/coordinates of a pixel
- Be able to relate the visual appearance (e.g., color) to the values in the matrix representation
- Be able to cut/crop part of an image
- Be able to apply non-overlapping block operations to an image using blkproc
- Be able to apply overlapping block operations to an image using nlfilter
- Understand that some operations are nonrecoverable/uninvertable
- Understand that lowpass filtering in the time domain is averaging in the frequency domain
- Find the frequency response of a system using its input and output images

2 List of equipment used

- A Computer.
- MATLAB.
- Image Processing Toolbox.

3 Creating matrices

To create a matrix x of zeros with 10 rows and 5 columns, we type the following command into MATLAB:

```
x = zeros(10,5)
```

To create a matrix y of ones with 10 rows and 5 columns, we type the following command into MATLAB:

```
x = ones(10,5)
```

```
>> x = zeros(10,5)

x =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

```
>> y = ones(10,5)

y =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

To create a matrix z of values going from -5 to 5, we type the following command into MATLAB:

```
z = -5:5
```

To change the orientation of the matrix, ie. to change rows to columns and columns to rows. We take the transpose of the matrix:

```
z = z'
```

```
>> z = -5:5

z =

    -5    -4    -3    -2    -1     0     1     2     3     4     5

>> z = z'

z =

    -5
    -4
    -3
    -2
    -1
     0
     1
     2
     3
     4
     5
```

4 Modifying matrices

To alter a part of the matrix, we type the following command:

```
x = zeros(10,5)
x(4:7,2:4) = 1
```

```
>> x = zeros(10,5)

x =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

```
>> x(4:7,2:4) = 1

x =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     1     1     1     0
     0     1     1     1     0
     0     1     1     1     0
     0     1     1     1     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

5 Building simple 2D filters

5.1 Low-pass & high-pass filter

Building a low-pass filter is easy as the center pixel's value is determined by the pixels nearby or conventionally called as the "neighborhood" of the pixel. A matrix of zeros is created and then the center part is made one:

```
lpf = zeros(256,256);
lpf(64:192,64:192) = 1;
imshow(lpf);
```

In s high-pass filter the center pixel's value is not determined by the neighborhood but the pixels farthest away. A matrix of ones is created and then the center part is made zero:

```
hpf = ones(256,256);
hpf(64:192,64:192) = 0;
imshow(hpf);
```

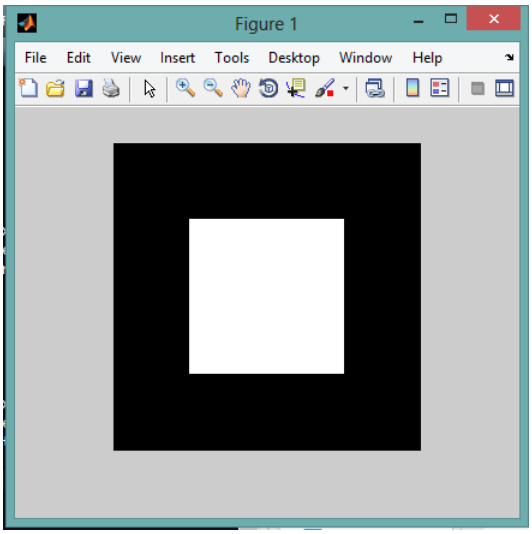

Figure 1: Low-pass filter

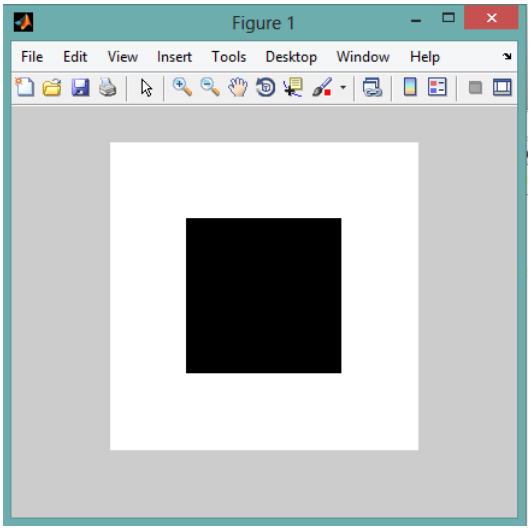

Figure 2: High-pass filter

5.2 Band-pass & band-reject filter

Building a band-pass filter is just like allowing the pixels surrounding the center to pass but the pixels further away go to be rejected. A matrix of zeros is created and then the center part is made one:

```
bpf = zeros(256,256);
bpf(64:192,64:192) = 1;
bpf(96:160,96:160) = 0;
imshow(bpf)
```

In s high-pass filter the center pixel's value is not determined by the neighborhood but the pixels farthest away. A matrix of ones is created and then the center part is made zero:

```
brf = ones(256,256);
brf(64:192,64:192) = 0;
brf(96:160,96:160) = 1;
imshow(brf)
```

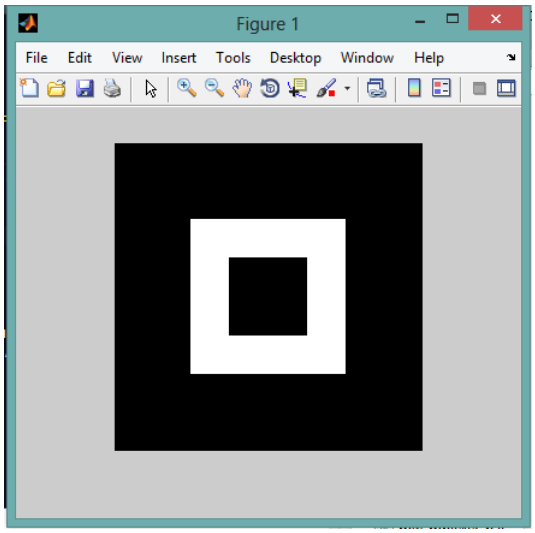

Figure 3: band-pass filter

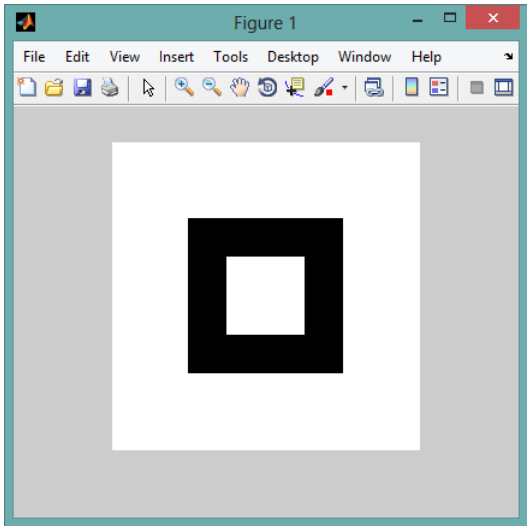

Figure 4: band-reject filter

6 Gray-level and RGB image

Any digital image is composed of pixels spread in two dimensional space. Any one of the pixel is represented by its x and y positions on the screen. In gray-level image, a pixel has a value determining how close to black or white color a pixel is. If the pixel values are unsigned integer 8-bits, then value 255 is the whitest and value 0 is blackest. Any value in between is a shade of gray. On the contrary, a color image is composed of pixels which often have three values, the red value, the green value, and the blue value. These values indicate how close to the primary color the pixel is. For example, if a pixel has R = 255 G = 0 B = 0 values then it means the pixel is entirely red.

There are some image formats which also allow an additional value for each pixel to be stored in an image. This is the alpha value, which determines the transparency of the pixel. A value of 255 indicates that the pixel is completely opaque while the 0 value indicates that the pixel is completely transparent. This fourth alpha channel is crucial in many image applications and visual effects because it allows dynamics of the background and sense of reality.

For an image to be processed by MATLAB, it is often recommended to convert the pixel value from unsigned integer to double and map every value from 0-255 to 0-1.[1] This allows the image matrix to become double so that is image operations and processing is done on it, it can candle decimal values. At the end of image processing, the image is converted back to 0-255 range so that the real world displays can handle them.

We can load some images from the MATLAB library and convert a RGB color image into a gray-level image by typing the following code:

```
peppers = imread('peppers.png');
imtool(peppers);
pepgray = rgb2gray(peppers);
imtool(pepgray);
```

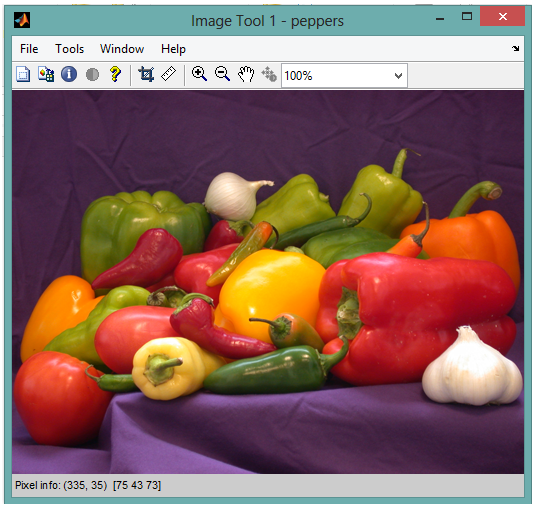

Figure 5: RGB image

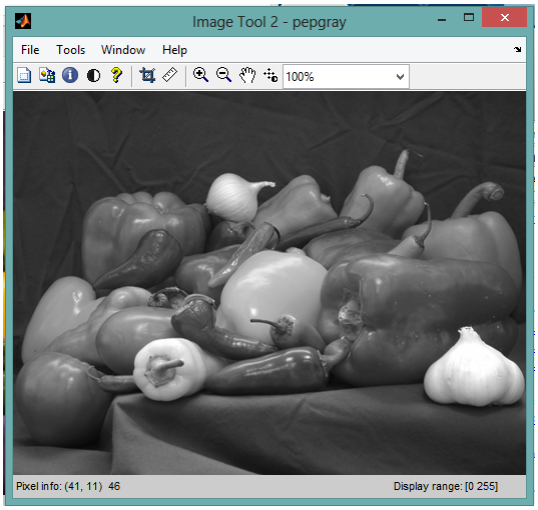

Figure 6: Gray-level image

7 2D position values and pixel values of an RGB image

Any one of the pixel is represented by its x and y positions on the screen. In gray-level image, a pixel has a value determining how close to black or white color a pixel is. If the pixel values are unsigned integer 8-bits, then value 255 is the whitest and value 0 is blackest. Any value in between is a shade of gray. On the contrary, a color image is composed of pixels which often have three values, the red value, the green value, and the blue value. These values indicate how close to the primary color the pixel is. For example, if a pixel has R = 255 G = 0 B = 0 values then it means the pixel is entirely red.

imtool() command in MATLAB allows images to be shown and at the same time, when the user hover the mouse over a pixel, the x,y position co-ordinates, and the RGB values of the pixel under the cursor are shown.

```
imtool(imread('peppers.png'));
```

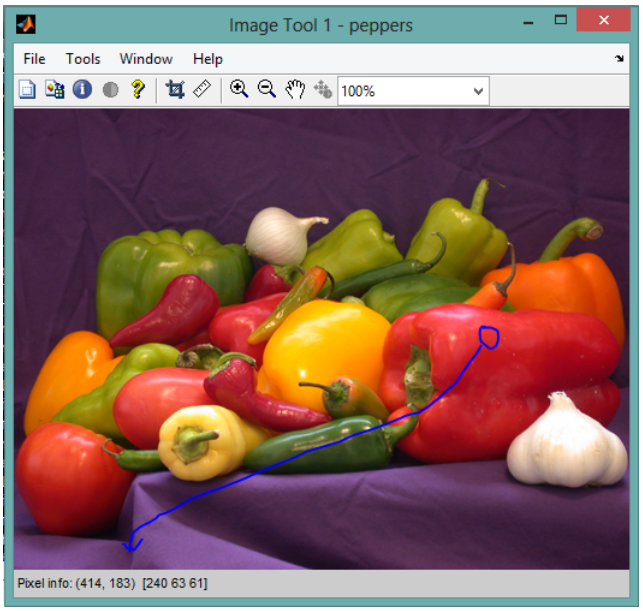

Figure 7: RGB image with selected pixel values. x and y position values are written in round bracket while the square bracket first value tells about Red channel value, Green channel value, and Blue channel value. As the selected pixel is red, the value for red channel is quite high compared to other channels.

8 Cutting/Cropping part of an image

If we think of an image as a matrix, we can select some elements of the matrix and copy them into a new and smaller matrix. This is known as cropping an image. All computer image editing software use this technique behind the scenes. What a user essentially does is to select the corners of bounding box by drawing a rectangle on the image and then the software selects the pixels inside that rectangle, copies them onto a new matrix and displays it.

The reason red image of garlic is white is because the garlic is white, hence is a combination of all primary colors in high quantity.

Let's do this cropping for ourselves in MATLAB:

```
% read the image into MATLAB
peppers = imread('peppers.png');
% select part of the x,y but select all of RGB
garlic = peppers(230:314,407:511,:);
% Show the garlic by advanced imtool
imtool(garlic)
% select part of the x,y but select red channel
redgarlic = garlic(:,:,1);
% Show the red garlic by advanced imtool
imtool(redgarlic)
% select part of the x,y but select blue channel
bluegarlic = garlic(:,:,3);
% Show the blue garlic by advanced imtool
imtool(bluegarlic)
```

Figure 8: The cropped peppers image with all RGB colors.

Figure 9: The cropped peppers image with only red pixel value.

Figure 10: The cropped peppers image with only blue pixel value.

9 Doing block processing on an image

Block processing is very powerful technique in image processing as it allows the program to focus and process a part of an image at a time. If we think of about block processing, it is kind of like doing convolution in 2 dimensional space. There are two kinds of block processing we can do.

- Non-overlapping block processing

The function we are going to use is blockproc(A, [M,N], FUN). This function takes a matrix A as an input argument and applies the function FUN on the matrix A with the sample size of M,N. The size of sample can be thought of as a brush size.

So first let's make an averager function to apply to a block:

```
function y = averager(x)
  % This function simply finds a 2 dimensional average of the matrix supplied to it
    y = mean2(x);
end
```

Let's make a function to pass:

```
% This function takes a block structure and averages the pixels around it to produce
    the value at the center
fun = @(block_struct) averager(block_struct.data(:,:));
```

Now, let's apply this function on the peppers image

```
% Read image into MATLAB
peppers = imread('peppers.png');
% Convert the peppers image into gray-level image
peppers_gray = rgb2gray(peppers);
% Show gray-level image
imtool(peppers_gray);
% Apply block processing with brush size of 3,3
result_image = blockproc(peppers_gray,[3,3],fun);
% Convert the resulting image from double to unsigned integer 8-bits
result_image = uint8(result_image);
% Show the resulting image in imtool window
imtool(result_image);
```

Figure 11: The original gray-level peppers.png image.

Figure 12: The block processed gray-level peppers.png image.

Let's make another function to pass:

```
% This function takes a matrix and returns a pixel at the center of the matrix
function y = sampler(x)
    %imshow(x);
    [rows,cols] = size(x);
    rowinmiddle = ceil(rows/2);
    colinmiddle = ceil(cols/2);
    y = x(rowinmiddle,colinmiddle);
end
```

Now, let's apply this function on the peppers image

```
% Read image into MATLAB
peppers = imread('peppers.png');
% Convert the peppers image into gray-level image
peppers_gray = rgb2gray(peppers);
% Show gray-level image
imtool(peppers_gray);
% Apply block processing with brush size of 3,3
processed_image = blkproc(peppers_gray,[3 3],@sampler);
% Show the resulting image in imtool window
imtool(processed_image);
```

Figure 13: The original gray-level peppers.png image.

Figure 14: The block processed gray-level peppers.png image.

This block processing results in a much sharper result than the averager block process. This is because the average or mean is kind of low-pass filtering of the image.

Non-overlapping block operation As we can notice from the block processed image, the resulting image is much smaller and has less pixel information. This is because of the fact that a single block results in a single pixel on the resulting image. The bigger the sample size, the smaller is the resulting image.

10 Doing nlfiltering on an image

Let's apply sampler function on the peppers image using non-local filter

```
% Read image into MATLAB
peppers = imread('peppers.png');
% Convert the peppers image into gray-level image
peppers_gray = rgb2gray(peppers);
% Show gray-level image
imtool(peppers_gray);
% Apply non-local filtering with brush size of 20,20
processed_image = nlfilter(peppers_gray,[20 20],@sampler);
% Show the resulting image in imtool window
imtool(uint8(processed_image));
```

Figure 15: The original gray-level peppers.png image.

Figure 16: The non-local filter processed gray-level peppers.png image.

We can immediately conclude that the images are not changed, that is because the function sampler returns the pixel at the center back to the filter and the filter places the pixel on the same position again.

Let's apply averager function on the peppers image using non-local filter

```
% Read image into MATLAB
peppers = imread('peppers.png');
% Convert the peppers image into gray-level image
peppers_gray = rgb2gray(peppers);
% Show gray-level image
imtool(peppers_gray);
% Apply non-local filtering with brush size of 20,20
processed_image = nlfilter(peppers_gray,[20 20],@averager);
% Show the resulting image in imtool window
imtool(uint8(processed_image));
```

Figure 17: The original gray-level peppers.png image.

Figure 18: The non-local filter processed gray-level peppers.png image.

We can immediately conclude that the image is blurred and there are no sudden changes of color. It is because of the fact that the averager is kind of low-pass filter and allows small changes of colors to pass while smoothing or lowering the amplitude of the high frequency regions.

11 Non-recoverable image data

```
% Read image into MATLAB
peppers = imread('peppers.png');
% Convert the peppers image into gray-level image
peppers_gray = rgb2gray(peppers);
% Process the gray-level image using non local filter
processed_image = nlfilter(peppers_gray,[20 20],@averager);
% Create a predefined high-pass filter
H = fspecial('unsharp');
% Try to recover the image
recovered_image = imfilter(processed_image,H);
% Show the resulting image in imtool window
imtool(uint8(recovered_image));
```

Figure 19: The original gray-level peppers.png image.

Figure 20: The recovered gray-level peppers.png image.

As we can see, after low-pass filtering and then trying to recover the information did not work for us. The information is permanently lost and cannot be recovered.

12 Find the frequency response

```
cameraman = imread('cameraman.tif');
figure('name','Original Image');
imshow(cameraman);
camera_double = im2double(cameraman);
F = fft2(camera_double);
F = fftshift(F); % Center FFT
F = abs(F); % Get the magnitude
F = log(F+1); % Use log, for perceptual scaling, and +1 since log(0) is undefined
F = mat2gray(F); % Use mat2gray to scale the image between 0 and 1
figure('name','FFT Image');
imshow(cameraman);
imshow(F,[]); % Display the result
```

Figure 21: The original image.

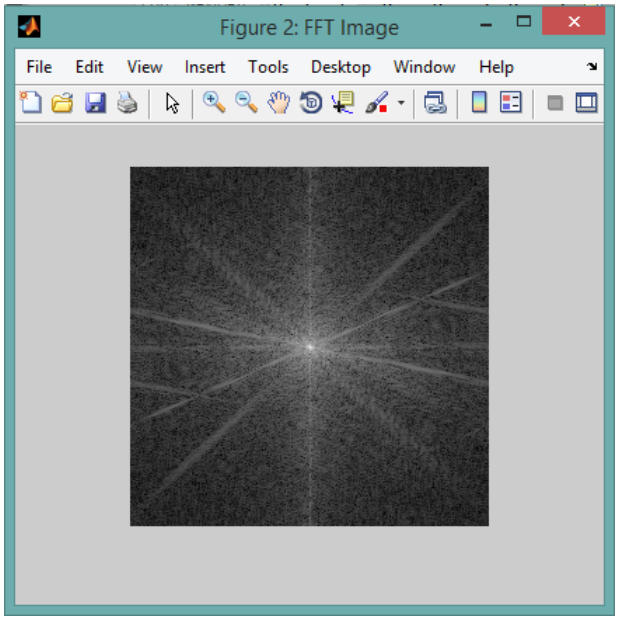

Figure 22: The frequency plot of the image.

References

[1] Mathworks. *Image Processing Toolbox*. [Electronic]. Available: `http://www.mathworks.com/products/datasheets/pdf/image-processing-toolbox.pdf` [2 June 2014].

[2] Dr.Tim Morris. *Image Processing with MATLAB*. Page 2. Available: `http://studentnet.cs.manchester.ac.uk/ugt/COMP27112/doc/matlab.pdf` [2 June 2014].

[3] Huidan Cao & Yuming Shen. *Application of MATLAB image processing technology in sewage monitoring system* [Electronic]. Available: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5274144&isnumber=5273983` [2 June 2013].