Elex7815 Laboratory manual

Digital Image & Video Processing

Bachelor of Engineering

School of Energy

Author: John Dian, Ph. D., P. Eng.

Lists of lab experiments:

1. Image manipulation
2. Point processing
3. Histogram processing
4. Image sub sampling
5. Image quantization
6. Spatial Filtering
7. Frequency-domain filtering
8. Noise removal
9. DCT transformation
10. Motion estimation

LAB 1

Image manipulation

Introduction

Digital grayscale or color image can be saved in memory for storage and processing. A digital grayscale image can be represented as a 2-D matrix. Two possible ways to represent a color images are RGB and indexed image. A RGB image can be represented by three separate 2-D matrices. An indexed color can be represented by a 2-D matrix and a color palette.

We use MATLAB (MATrix LABoratory) to read, write and process grayscale and color images. MATLAB is a data analysis, prototyping, and visualization tool with built in support for matrices and matrix operation. MATLAB provides a development environment based on its high-level programming language. MATLAB uses various toolbox programs to extend its basic capabilities. Example of these toolbox programs are signal processing toolbox, control toolbox and image processing toolbox.

The image processing toolbox (IPT) is a collection of functions that extend the basic capability of MATLAB environment to enable specialized image and video processing operation. IPT has built-in functions to read and write gray-scale or color image files in most popular formats in variety of data classes. It also has several functions for displaying images and excellent tools to inspect the pixel values. IPT has a large number of functions for processing digital image & video signals.

Task1: Reading and writing grayscale images

1- Read the image Barbara.jpg using **imread** function and use **imfinfo** function to find the structure of this image. The **imfinfo** function provides too many information about this image. Record the result and find out what you can understand from it.

```
I=imread (‘barbara.jpg‘);
imfinfo ((‘barbara.jpg‘)
```

2- Why do you use the semicolon operator after **imread** statement? What happens if you don’t use semicolon?

3- When MATLAB read and image, it puts the pixel values in an array for further processing. These pixel values can belong to different data classes depending on the image. The most common dataset for gray scale images are **unit8** (1 byte per pixel in the range of [0 255]) and **double** (8 bytes per pixel, in the range of [0.0 1.0]). The following table summarizes some ways to get information about an image. Use the following functions and explain what each function provides. What types of dataset the image ‘barbara.jpg‘ has?

Whos	To get information about size, type, and bytes, of all variables.
whos I	For information about an image stored in I.
size(I)	To get the size of the image stored in I.

class(I)	To get type of data stored in I

4- If you want to view an image, you can use the **imshow** function, which opens a separate window displaying the image. For instance: **imshow(I)** will show Barbar.jpg. **imshow (I,[ ])**, shows an scaled and improved version of image I for display purposes, while **imshow (I, [a, b])** display an image with all pixel values lower than **a** as black and any values greater than **b** as white. Use these function and show the Barbara image.

5- If you type **imtool(I)**, you will start up the image tool. You can get a close up view of the pixels of a particular region using the "**Inspect pixel values**" option from the toolbar. You can also resize the view area by dragging the corners of a box on the main image. If the box is small enough, the close up view will show pixel values. Click the middle of the box and drag the cursor around on the image to see that brighter areas are closer to 255 and darker areas are closer to 0.Use the **imtool** function and explain about your experiment.

6- If you want to save an image matrix (stored in memory) to an image file, you can use **imwrite** function. This function can be used to convert the format or even the quality of the image. Read a gray scale image 'lena.tiff' in tiff format and save it in another formats. The statements below show some of the parameters of this function.

```
I_Lena= imread('lena.tiff');
Imwrite(I_Lena,'lena1.jpg');
Imwrite(I_Lena,'lena2.jpg', 'quality',5);
Imwrite(I_Lena,'lena3.jpg', 'quality',80);
```

Find the size of each of the files that you saved. Also show all the images in one figure using subplot command. Explain about the quality and the size.

Task 2: Reading color images

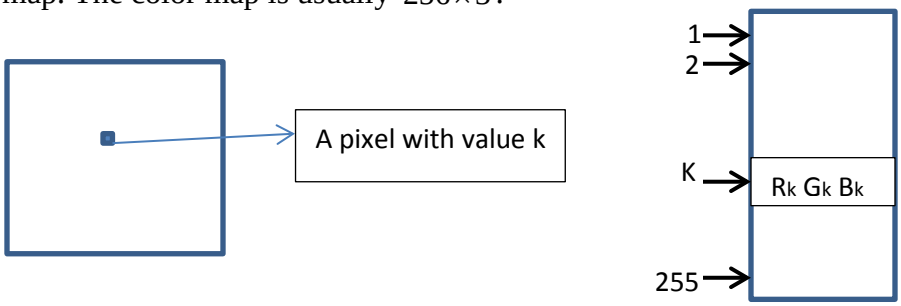
There can be different types of data stored in the image matrix. Table below lists different types of images and their possible dataclasses.

Binary	logical 0's and 1's (0 is black and 1 is white)
Grayscale (intensity)	Values have ranges as follows: single or double arrays [0, 1] uint8 [0, 255] uint16 [0, 65 535] int16 [-32 768, 32 767]
Indexed (pseudocolor)	values in image are indices into a colormap with length of p single or double arrays [1, p] logical, uint8, or uint16 [0, p-1]
Truecolor (RGB)	Values have the following range: single or double arrays [0, 1] uint8 [0, 255] uint16 [0, 65 535]

For each pixel in grayscale format, there is a single value stored in a two dimensional matrix, from a range of values appropriate to a data type. For instance, if the data type is unit8, then the values are from 0 to 255.

In the previous table, two types of color image are shown. The first one is trucolor or RGB and the second one is indexed image.For each pixel, in RGB color image, there are 3 values as intensity of Red, Green and Blue (RGB). Therefore, an RGB image of size $M \times N$ can be stored in a $M \times N \times 3$ array. If the RGB image has a unit 8 data class, all intensity values are in range of [0 255]. For instance, for a pure red pixel, the intensity value of red is 255, and the intensity values for green/blue are zero.

An indexed image of size $M \times N$, can be stored in $M \times N$ array of indices (pointers) to a secondary color map. The color map is usually 256×3.



1- What type of color do the image ‘fruits.png’ have? Read image "fruits.png" into I2 and find the size of this image.
2- Find out the value of the pixel intensity at location (93,180) in image fruits.png. Remember, the red intensity is represented by I2(93,180,1), the green intensity is represented by I2(93,180,2) and the blue intensity is represented by I2(93,180,3).
3- Show the red channel image, green image channel and blue image channel of this image. What color the red objects would appear in red channel? What about white objects?
4- Read an RGB color image and convert it to indexed image (see table below) and save the file in indexed format.you can use ‘fruits.png’ as an RGB image. Call the indexed image as ‘fruits2.png’. Read the saved indexed image in variable I3 and map as **[I3, map]=imread(‘fruits2.png‘)** . Use **imshow(I3, map)** to display the indexed image.
5- The following table lists commands provided by the Image Processing Toolbox that convert between the different color-formats. Read ‘onion.png’ image and try to understand how to use the following functions to convert the images.

`gray2ind()`	Convert from intensity format to indexed format.
`ind2gray()`	Convert from indexed format to intensity format.
`ind2rgb()`	Convert from indexed format to RGB format.
`mat2gray()`	Convert a regular matrix to intensity format by scaling.
`rgb2gray()`	Convert from RGB format to intensity format.
`rgb2ind()`	Convert from RGB format to indexed format.
`im2bw()`	Convert an intensity/indexed/RGB image to a thresholded black & white binary image

6- If a RGB image is converted to an indexed image, do you think that the quality will be changed? why

Task 3: Getting familiar with image processing toolbox

1- Image processing toolbox (IPT) provides many functions that can be used in image processing. It also has a good help section which can be very helpful to understand the parameters and functionality of the toolbox's functions. IPT also provide several demos that show the capabilities of the IPT. Go to start->toolboxes→ image processing→demoes or help->documentation->image processing toolbox->example (depending on the IPT version that you are using). Theses demoes give you a quick and easy way to understand how IPT functions can be used and combined to develop a MATLAB code for an application. Choose three demoes and evaluate the code. In your report explain what you learned about the demoes. Remember,
   a. Any block of codes in these demoes can be evaluated by highlighting the code and selecting **Evaluate selection option**.
   b. You can open an editor (File→ new→ script) and copy the code blocks there and evaluate the complete code.
   c. Read the comments of the code and follow the demo to get a good picture of image processing algorithm. The syntax of most of the functions used may be unfamiliar to you at this point.

Task 4: Pixel intensity along one row and its correlation

1- Write a MATLAB program to read an image and plot the profile of pixel intensities along a specific row. Consider the 'cameraman.tif ' as a test image and plot the pixel intensities along row 164.

2- Observe the pixel values in this plot. Do you see any correlation among the pixel intensities of this row? What is the reason for that?

Optional:

3- Complete your MATLAB program to plot the normalized correlation among the row intensities of part 1, x(1:N), over m pixel replacements. Consider m to be 128. N is the number of columns of your image. The correlation at displacement **k** can be calculated as

$$corr(k) = \frac{sum[x(k:N).x(1:N-k+1)]}{N-k+1} \qquad k = 1,..,m$$

The normalized correlation can be calculated as

$$normalized\_corr(k) = \frac{corr(k)}{Max[\ corr(k)]|k = 1,..m}$$

4- Do you think that pixels may have a high correlation along the columns as well? Is it true to say that the basic principle behind image data compression is to de-correlate the pixels and encode the resulting de-correlated image for transmission or storage? What do you think is the difference between various compression schemes?

LAB 2

Point Processing

Introduction

Image enhancement techniques are used to improve the presentation of image detail to human eyes or machines. One of the techniques for image enhancements is point processing. In point processing, the value of a pixel **r** in original image is changed to a new value **s** using a transformation which can be shown as:

$$s = T\,[r]$$

Examples of the transformation that can be used in point processing are:

- Linear point processing $s = c.\,r + b$
  - Negative (contrast reverse) c=-1, b=255
- Power law (Gamma transformation) $s = c.\,r^{\gamma}$
- Log transform $s = c.\,\log(1+r)$
- Piecewise linear transformation $\begin{cases} s = c_1 r + b_1 & 0 < r < a_1 \\ \vdots & \\ s = c_n r + b_n & r > a_n \end{cases}$
- Thresholding $\begin{cases} s = 0 & r \le r_m \\ s = 255 & r > r_m \end{cases}$
- Auto contrast adjusting $s = \dfrac{L-1}{r_{\max} - r_{\min}}(r - r_{\min})$, where L-1 be the highest gray value in the input image
- Contrast stretching, amplitude scaling $s = 1/((1+(m/r))^{e})$

Task1- Point processing transformations

1- Choose two of the following and create a graph to show the relationship between **s** and **r** $(0 \le r \le 255)$ for
   a) Negative transformation
   b) For a logarithmic transformation, use value of c=0.1, 1, 10
   c) For a gamma transformation. Use γ =0.5, 0.1,2, 10. Find the value of c for each case.
   d) Contrast stretching with m=0.5, e=1, 2, 3, 4, 5

2- A linear point processing with c=1 and b=60 adds a constant value to the entire pixels in the image. You can use **imadd** function to do so in MATLAB. Read an image and add a constant value to all the pixel values within image. Plot the original and processed image.

3- What are the minimum and maximum pixel values in the original and processed image in step 2. How many pixels had value 255 in the original image, how many in processed image. Explain your result.

4- The negative transformation of an image produces a negative image of the input image. This image also called contrast reverses. MATLAB has **imcomplement** function which produces its negative of an input image. Write a Matlab code to read the image 'Tire.tif' as an image and produce the negative image. Show the original and negative image.

5- Write a MATLAB code to read the image 'tire.tif' and use gamma transformation with three different values of gamma as 0.4, 1 and 3. Plot all the images and explain the result.

6- Write a MATLAB code to read tire.tiff image and using 3 different values for **c ,** study the effect of logarithmic transformation on this image. Plot the original image and the three transformed images. Choose the values of c=1, 2 and 5.

7- Using which value in step 6, you can see the radial lines on the inside of the tire? Why you couldn't see them in other images?

8- Write a MATLAB function to do the following point processing function. If any pixel value is less than a threshold T, the pixel is replaced by its negative value. The pixel values equal or greater than Threshold T would not be changed. Apply this function to an image of your choice and explain about your result. Use T as 0, 255 and 100.

9- Consider a sub-image with intensity values as shown below, where 4 is the intensity value in location (1,1), 2 is the intensity value in location (1,2), 3 is intensity value in location (2,1) and 6 is intensity value in location (2,2). Write a MATLAB program to calculate a value at location (1.7 , 1.2) using

   a. Two 1-D bilinear interpolation
   b. 2-D bilinear interpolation

10- Write a MATLAB code to make the following image of size 256 x 256. The rectangle inside the image is gray color and the size of 20 x 50. The center of this rectangle is the center of the image. Rotate this image by $\alpha = 10^{\circ}$ clock wise and $\alpha = -80^{\circ}$ counter clockwise. Use inverse interpolation and bilinear interpolation. Do not use MATLAB rotate function.

Optional

11- In contrast stretching transformation, **e** controls the slope of the function and **m** is the mid-line where you want to switch from dark values to light values. Write a MATLAB code to read an image and applies three different contrast stretching transformations. Choose **m** value as the mean of the image intensities. Choose **e** values as 4, 5 and10. Make sure that your image is in double class type.

12- Explain why using e=10 in step 11, the transformation function becomes more like a thresholding function (the resulting image is more black and white than grayscale).

LAB 3

Histogram processing

Introduction

Histogram is a way of visualizing the predominant intensities of an image. As a definition, image histogram is a count of the number of pixels that are at certain intensity.

Histogram equalization is a mathematical method by which we change the histogram of the input image in such a way that the processed image to have a uniform distribution or flat histogram. Histogram equalization can be used for image enhancement. The idea behind Histogram Equalization tries to evenly distribute the occurrence of pixel intensities in an image so that the entire range of intensities is used more fully. In other word, we are trying to give each pixel intensity equal opportunity. Especially for images with a wide range of values with detail clustered around a few intensities, histograms will improve the contrast in the image. In MATLAB, the function to perform histogram equalization is **histeq(I).**

Histogram matching (also called Histogram Specification) is a process where an image is modified such that its histogram matches that of another (reference) image histogram. A common application of this is to match the images from two sensors with slightly different responses, or from a sensor whose response changes over time. In MATLAB, **histeq** function can also be used for histogram matching. In this case, **Histeq** function accepts a second parameter which show the way you want the histogram of an image to look like. For example, `M(1:256)=1; g=histeq(I,M);` commands produce the result that is similar to the histogram equalization, since the elements in vector M is all 1. However, using the different dataset as the second parameter in **histeq** function can be used for histogram matching.

Local histogram equalization (LHE) uses a sliding window method in which, for each pixel, local histograms are computed from the windowed neighborhood to produce a local grey-level remapping for each pixel. Then the intensity of the pixel at the center of the neighborhood is changed according to the local gray-level remapping for that pixel. LHE is capable of great contrast enhancement. LHE-based methods are generally requiring more computation than histogram equalization method.

Task 1 – Histogram

1- MATLAB easily displays image histograms using the function **imhist** (I). Using this function, plot the histogram of the following images. barbara.jpg, Tire_gray.jpg, pout_gray.jpg and eight_gray.jpg. How the histograms are different?
2- Which image would you expect to be darkest? Which image would you expect to be brightest?

Task 2: Histogram Equalization

1- Write a MATLAB function to calculate the histogram of an image without using the **imhist** function and calculate the histogram equalization without using **histeq** function.

2- Compare the histogram that you obtained in step 1 with the ones that you can obtain using **imhist** and **histeq** fuctions? Are the results the same?
3- Write a MATLAB code to apply histogram equalization to an image with predominately low range intensities such as ‘tire_gray.jpg’. Plot the resulting image and resulting histogram. Explain about the result. Also apply histogram equalization to an image with predominately mid-range intensities such as ‘pout_gray.jpg’. Plot the resulting image and resulting histogram. Explain about the result.

Task 3- Histogram Matching

1- Read two images and plot their histogram and the image.
2- Write a MATLAB code that make the histogram of the second image similar to the first image.

Optional

Task 4: local histogram equalization

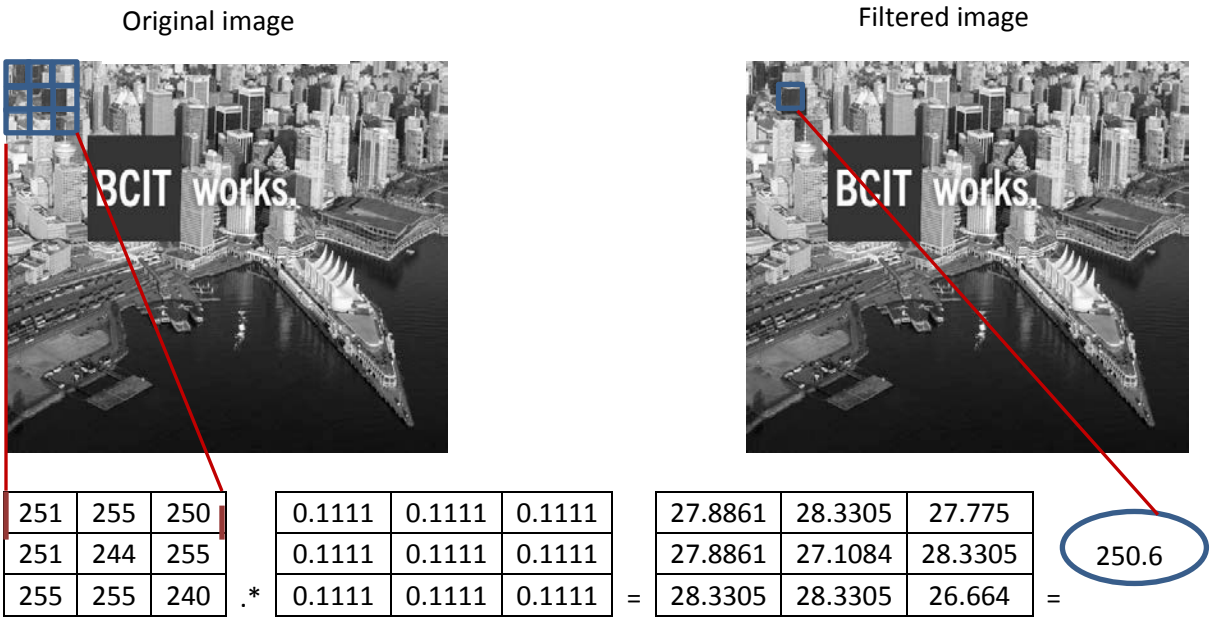
1- Write a MATLAB code to read an image and generate local histogram equalized image using window size of $M \times N$. Read the image ‘tire.tiff’ as an example and choose M=10 and N=20.
2- Compare the result with the histogram equalized image in task 2.
3- Also write a MATLAB code to compute the local histogram processing by block-based histogram equalization. Compare your resut with the result of Step 1.

LAB 4

Spatial filtering

Introduction

Spatial filtering is a technique that uses a pixel and its neighbors to select a new value for the pixel. It is also known as neighborhood processing. Neighborhood processing is an appropriate name because the value of each pixel will be modified by performing an operation or applying a filter to only those pixels in predetermined neighborhood of that pixel. The result of the operation is one value, which becomes the modified value of the pixel. The simplest type of spatial filtering is called linear filtering. It attaches a weight to the pixels in the neighborhood of the pixel of interest, and these weights are used to blend those pixels together to provide a new value for the pixel of interest. Linear filtering can be uses to smooth, blur, sharpen, or find the edges of an image. Sometimes a linear filter is not enough to solve a particular problem, and non-liner filtering may be applied. The following diagram is meant to illustrate in further details how a simple spatial filter is applied on a pixel. The value of 244 in the center has changed to 251 after applying the spatial filtering with a window which all its coefficients are 0.1111.



The breakdown of how the resulting value of 251 (rounded up from 250.66) was calculated mathematically is:

```
= 251*0.1111 + 255*0.1111 + 250*0.1111 + 251*0.1111 + 244*0.1111 + 255*0.1111
+ 255*0.1111 + 255*0.1111 + 240*0.1111= 250.66
```

The filter should be applied to the entire pixels within an image by sliding the filter window over the image. There is an inherent problem when you are working with the image boundaries. The problem is that some of the "neighbors" are missing. As you can see in the next image, there are no upper neighbors or neighbors to the left. Two solutions to solve this problem exist. One is zero padding and the second one is replicating. In zero padding, the missing value is replaced by zero. The disadvantage of zero padding is that it leaves dark artifacts around the edges of the filtered image (with white background). In replicating, the missing value is replaced by the

image boundary values. As a note, if your filter were larger than 3x3, then the "border padding" would have to be extended.

1- Write a Matlab function to get the original image and a 3x3 filter matrix and performs the spatial filtering. Consider that the original image should be padded with 0's.
2- Write a MATLAB program to read an image and using the function on Step 1, plot the filtered image. Use the filtered window as [1/9 1/9 1/9; 1/9 1/9 1/9; 1/9 1/9 1/9]. Use the image 'bcitworks.jpg' as the image of your choice.
3- Find a pixel within the original image and calculate the value of the pixel after applying the filter. Did you get the same value in the filtered image. Show your calculation.
4- Explain about your result. What is the result of spatial filtering using the window above?
5- Instead of using the function you wrote in step 1, you can use a function that comes as part of the Image Processing Toolbox (IPT). This function is called **imfilter** (image, w). This function also accepts some additional parameters which are summarized below. Use this function and explain about your result.

Options	Description
Filtering mode	
'corr'	Filtering is done using correlation. This is the default.
'conv'	Filtering is done using convolution.
Boundary Options	
P	Zerro padding; The boundaries of the input image are extended by padding with a value, P (written without quotes). This is the default, with value 0.
'replicate'	The size of the image is extended by replicating the values in its outer border.
'symmetric'	The size of the image is extended by mirror-reflecting it across its border.
'circular'	The size of the image is extended by treating the image as one period a 2-D periodic function.
Size Options	
'full'	The output is of the same size as the extended (padded) image.
'same'	The output is of the same size as the output. This is achieved by limiting the excursions of the center of the filter mask to points contained in the original image. This is the default.

6- What is the difference between zero-padding, replicate, symmetric boundary options. For a filter of size 3x3, does 'replicate' and 'symmetric' yield the same results?

7- You can define the filters for spatial filtering manually or you can call a function that will create certain common filter matrices for you. The function, called **fspecial** , requires an argument that specifies the kind of filter you would like. A full description of **fspecial** is available in MATLAB help—type:

   doc **fspecial**

   Summarize this function in some lines.

8- Find the following special filters and write the filtering window.
   a. h=fspecial('motion', 20, 45);
   b. h=fspecial('unsharp');
   c. h=fspecial('sobel');

9- Apply a 5x5 averaging filter using the command **h=fspecial('average',5)**to an image and use the following four different boundary options (zero padded, replicate, symmetric and circular). Explain about your result. Explain why there is a darker border along the bottom and right-hand edge in the zero-padded image? Use peppers.png as your image.

10- Filtering mode for **imfilter** function can be set to correlation or convolution. Correlation is the default filtering mode for this function. If convolution mode needs to be used, then the function **imfilter(image,filter_parameter,'conv')** needs to be used. The only difference between the correlation and convolution is the fact that in case of convolution the filtered should be rotated 180. Read an image and apply filtering mode once as convolution and once as correlation. You can use the following code for this purpose which will be applied to image i. Find a 3X3 window of the original image and show how **imfilter** function has calculated the new values for this 3x3 window using convolution and correlation. Use bcitworks.jpg as your image.

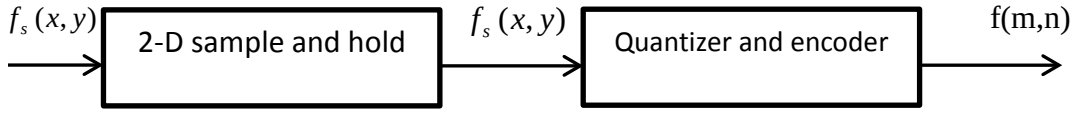
```
h=[1 2 3,   4 5 6,   7 8 9];
h=h/45;
result_corr=imfilter(i,h);   % correlation is the default,
result_conv=imfilter(i,h,'conv');
```

LAB 5

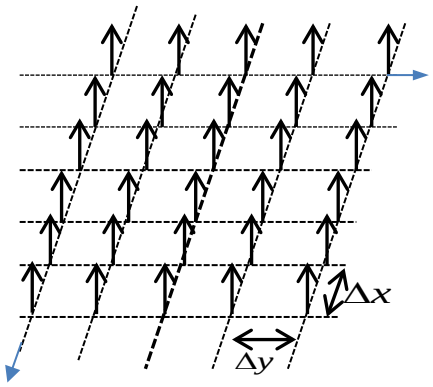
Image sub-sampling and aliasing

Introduction

An image $f(x,y)$,$-\infty \le x, y \le \infty$ taken by a camera is a continuous distribution of light intensity in the two continuous coordinates x and y. Even though practical images are of finite size, we assume here that the extent of the image is infinite to be more general. In order to acquire a digital image from $f(x,y)$, it must be sampled in the spatial coordinates and then the sample values must be quantized and represented in digital form, as shown below. The process of discretizing the 2-D function of $f(x,y)$ in the two spatial coordinates is called *sampling* and the process of representing the sample values in digital form is known as quantizing and encoding.



Let us first study the sampling process. The sampled image will be denoted by $f_s(x,y)$ and can be expressed as $f_s(x,y) = f(x,y)\big|_{x=m\Delta x,\, y=n\Delta n}$ $-\infty \le m, n \le \infty$. The values of m, n are integer whole numbers, and Δx and Δy are the spacing between samples in the two dimensions, respectively. Since the spacing is constant the resulting sampling process is known as uniform sampling. The sampling introduced here represents sampling the image with an impulse array and is called ideal or impulse sampling. In an ideal image sampling, the sample width approaches zero. The sampling process can be interpreted as multiplying $f(x, y)$ by a sampling function $s(x, y)$, which, uses a rectangle grid as shown in figure below, an array of impulses spaced uniformly at integer multiples of Δx and Δy in the two coordinates, respectively. An impulse function (Dirac delta function) in the two-dimensional (2D) domain is defined by $\delta(x,y) = 0 \;\; x, y \neq 0$



The ideal sampling function can be expressed as

$$s(x,y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta(x - m\Delta x, y - n\Delta y).$$

The sampled image $f_s(x,y)$ can be written as

$$f_s(x,y) = f(x,y) \times s(x,y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m\Delta x, n\Delta y)\;\; \delta(x - m\Delta x, y - n\Delta y)$$

In order for us to recover f (*x, y*) from the sampled image $f_s(x,y)$, we need to determine the maximum spacing Δx and Δy. This is done easily if we use the Fourier transform. So, let $F(u,v)$ be the 2D Fourier transform of $f(x, y)$ as defined by

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, e^{-j(ux+vy)}\, dx\, dy$$

The Fourier transform of the 2D sampling function s(x,y) can be shown as

$$FT[S(x,y)] = \frac{1}{\Delta x \Delta y} \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} \delta(u - m\frac{2\pi}{\Delta x}, v - n\frac{2\pi}{\Delta y})$$

where $2\pi/\Delta x$ and $2\pi/\Delta y$ are the sampling frequencies in radians per unit distance in the respective spatial dimensions. Because the sampled image is the product of the continuous image and the sampling function ,

$$F_s(u,v) = \frac{1}{\Delta x \Delta y} \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} F(u - m\frac{2\pi}{\Delta x}, v - n\frac{2\pi}{\Delta y})$$

As it can be seen, the Fourier transform of the ideally sampled image is obtained by first replicating the Fourier transform of the continuous image at multiples of the sampling frequencies in the two dimensions, and then scaling the replicas by $1/\Delta x \Delta y$, and finally adding the resulting functions. Since the Fourier transform of the sampled image is continuous, it is possible to recover the original continuous image from the sampled image by filtering the sampled image by a suitable linear filter. If the continuous image is lowpass, then exact recovery is feasible by an ideal lowpass filter, provided the continuous image is band limited to say $u_s/2$ and $v_s/2$ in the two spatial frequencies, respectively. That is to say, if

$$F(u,v) = \begin{cases} non-zero & |u| \le \dfrac{\pi}{\Delta x},\ |v| \le \dfrac{\pi}{\Delta y}/2 \\ Zero & otherwise \end{cases}$$

then *f* (*x, y*) can be recovered from the sampled image by filtering it by an ideal low-pass filter . Sampling theorem guarantees exact recovery of a continuous image, which is low-pass and band limited. If the sampling rates are below the Nyquist rates, namely, then the continuous image cannot be recovered exactly from its samples by filtering and so the reconstructed image suffers from a distortion known as *aliasing distortion.*

Even though sampling using rectangle grid is used often, it is possible to use nonrectangular grids, such as hexagonal sampling grid to acquire a digital image. It can be shown that in some applications, nonrectangular sampling can produce better result as compared to rectangular sampling. The process of converting pixels from rectangular to hexagonal sampling grids and vice versa can be done using linear matrix operation. For example, we can use the following transformation to find a new location (h_1, h_2) in non-rectangular grid where a pixel at location (R_1, R_2) in a rectangle grid needs to be transferred. It is clear that range for the hexagonal grid is not the same as the range of rectangular grid.

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$$

Task 1: aliasing distortion

1) Read the image 'barbara.jpg' image and record the size of the image?
2) Write a MATLAB program to down sample the image by a factor of *M (samples at integer multiple of M). R*econstruct the full size image using **imresize** function from the down-sampled image. Use bicubic as the interpolation method used by the **imresize** function. Choose a value of 4 for *M*.
3) Show the original image and the reconstructed image after down sampling
4) Write a MATLAB program to pre-filter the original image with a low pass filter and then down sample the filtered image and reconstruct it again.
5) Show the filtered image and the reconstructed pre-filtered image after sub subsampling
6) Explain the reason of filtering of an image before sub sampling.
7) Discuss the results.

Task 2: formula –based image sampling

1) Write a MATLAB code to generate an image using the following formula $f(x, y) = \cos(2\sqrt{10}\ \pi x)$. Rotate this image by $\alpha = \arctan(1/3)$ then sample this image. Sample with different values and explain about your result.

Task 3: Nonrectangular Sampling Grids

2) Write a MATLAB code to make an image of 256 × 256 array of pixels of alternating black and white dots. Show the image.
3) Using a linear transformation, convert pixels in the rectangular array into the hexagonal array. This will generate a hexagonal sampling grid. Display this sampling grid as an image.
4) Zoom the images you found in step 1 and 2 to have a better view of the grid in the rectangular and hexagonal grid.

Optional

Task 4: Image sampling in Nonrectangular Sampling Grid

1) Read the image 'barbara.jpg' as your original image for this task and write a MATLAB code to convert the image from the original rectangular sampling to hexagonal sampling grid using the following equation
2) Display the original image and corresponding hexagonally sampled image
3) If M and N are the number of rows and columns of the original image, respectively, what would be the size of the hexagonally sampled image?

LAB 6

Image quantization

Introduction

The image samples have a continuum of values. That is, the samples are analog and must be represented in digital format for storage or transmission. Because the number of bits for representing each image sample should be limited, the analog samples must first be quantized to a finite number of levels and then the particular level must be coded in binary number. Since the quantization process compresses the continuum of analog values to a finite number of discrete values, some distortion is introduced in the qauntization process. This distortion is known as quantization noise and might cause an artifact known as contouring.

In simple terms, a scalar quantizer receives an input analog sample and outputs a digital value. The digital output value is a close approximation to the input analog value. The output values are predetermined corresponding to the range of the input and the number of bits allowed in the quantizer. A scalar quantizer $Q(.)$ maps of input decision intervals d_k, $k = 1,...,L+1$ to output or reconstruction levels r_k, $k = 1,...,L$. Thus, $Q(x) = \hat{x} = r_k$. We assume that the quantizer output levels are chosen such that $r_1 \prec r_2 \prec ... \prec r_L$. The number of bits required to address any one of the output levels is $B = \lceil \log_2 L \rceil$, where $\lceil x \rceil$ is the nearest integer is equal to or larger than *x*. A design of a quantizer amounts to specifying the decision intervals and corresponding output levels and a mapping rule.

A Lloyd–Max quantizer is an optimal quantizer in terms of the mean square error (MSE). It depends on the input probability density function (pdf) and the number of levels allowed in the quantizer. Since, the input analog sample is a random variable; it can be described by its pdf represented by $p_x(x)$. The quantizer design is as follows: Let x be a random variable with a continuous pdf. The optimal MSE or Lloyd–Max quantizer design amounts to determining the input decision intervals d_k and corresponding reconstruction levels r_k for given L levels such that the MSE is minimized.

$$MSE = E((x-\hat{x})^2) = \sum_{j=1}^{L} \int_{d_j}^{d_{j+1}} (x-r_j)^2 \; p_x(x)\, dx$$

It can be proven that, MSE can be minimized under the following condition

$$r_j = \frac{\int_{d_j}^{d_{j+1}} x \; p_x(x)\, dx}{\int_{d_j}^{d_{j+1}} p_x(x)\, dx}$$

When the pdf of the analog sample is uniform, the decision intervals and output levels of the quantizer can be computed analytically. In this case, the decision intervals are all equal as well as

the intervals between the output levels and the quantizer is called a *uniform* quantizer. The uniform pdf of the input image is given by

$$p(x) = \frac{L}{x_{\max} - x_{\min}} = \frac{L}{d_{L+1} - d_1}$$

Where L is number of levels and B is the number of bits assigned for each level.It can be shown that the

$$r_j = d_j + \frac{d_{j+1} - d_j}{2} = d_j + \frac{\Delta}{2}$$

$$2^B = L \;\Rightarrow\; \Delta = \frac{d_{L+1} - d_1}{L} = \frac{d_{L+1} - d_1}{2^B}$$

$$d_j = \frac{d_{j+1} - d_{j-1}}{2}$$

Therefore, the intervals between the output levels are also equal to Δ. Thus, the design of the optimal quantizer for a signal with uniform distribution is carried out as follows:

1. Divide the input range (*x*min, *x*max) into *L* equal steps Δ with $d_1 = x_{\min}$ and $d_{L+1} = x_{\max}$
2. Determine the *L* output levels as $r_j = d_j + \frac{\Delta}{2}$.
3. Quantize a sample *x* according to $Q(x) = r_j$, $d_j \le x \le d_{j+1}$.

For the uniform quantizer, the quantization error $e = x - \hat{x}$ has the range $(-\frac{\Delta}{2}, \frac{\Delta}{2})$, and it is uniformly distributed as well. The performance of a uniform quantizer can be discussed in terms of the achievable signal-to-noise ratio (SNR). It can be proved that the SNR of a uniform quantizer is

$$SNR = \frac{(x_{\max} - x_{\min})^2}{\Delta^2} = 2^{2B}$$

In dB, a B-bit uniform quantizer yields an $SNR = 10 \log 2^{2B} = 6B\ (dB)$. It can be seen that each additional bit in the quantizer improves its SNR by 6 dB. Therefore, the Lloyd–Max quantizer for signals with uniform pdf gives an SNR of 6 dB/bit of quantization.

Task 1: Uniform Quantizer

1. Write a MTLAB program to read the image 'barbara.jpg' with 8 bits/pixel and re-quantize it to *B* bits/pixel using a uniform quantizer. Assume *B* to be 3 and 5 bits.
2. Display both the original and re-quantized images and observe the differences, if any.
3. Plot the SNR in dB due to quantization noise. For plotting, vary *B* from1 to 7 bits.
4. Explain the quantization step size, the decision boundaries and output levels, for the uniform quantizer with B bits of quantization.

5. Find the slope of SNR plot based on dB/bit.
6. The uniform quantizer is optimal only when the pdf of the input image is uniform. Plot the histogram of the image 'barbara.jpg' and explain if your uniform quantizer is optimal.
7. Find and plot the error due to quantization for 5 bpp. What you expect the quantization error range should be?
8. Plot the input decision intervals versus the reconstruction levels for the uniform quantizer. Is it a type of stair case form?

Task 2: Dithering

When an image is quantized coarsely, we notice contouring effects. This happens whether we use a uniform or a non-uniform quantizer. One way to mitigate contouring is to add a small amount of dither or pseudo random noise to the input image before quantization and then quantize the image. We can then subtract the dither from the quantized image. The dither perturbs the input by a small amount such that pixels having more or less the same values in a neighborhood fall into different decision regions and are, therefore, assigned slightly different output levels. This eliminates or minimizes the contouring.

1. Write a MATLAB program to Quantize the "Barbara.jpg" image to 4 bpp using a uniform dithered quantizer
2. Complete your program to subtract the dither after quantization.
3- Compare the result when same image quantized to 4 bpp without dithering
4- Discuss the result in Step 1,2and 3. Does dithering eliminate contouring without increasing quantization bits?

Optional

Task 3: Optimized Quantizer

A uniform quantizer is optimized if, the pdf of the analog sample is uniform. In this case, the intervals between any two consecutive decision regions as well as the intervals between any two consecutive output levels were constant.

When the pdf of the input analog samples is not uniform, then the quantization steps are not constant. In this case , there is a need to design a nonuniform quantizer which is referred to as pdf optimized quantizer. One such procedure is called the Lloyd algorithm .

1. Design an optimal quantizer for an input image such as Barbara with 8 bpp using Lloyd algorithm. Write a MATLAB program for this purpose. The algorithm for LIoyd quantizer is explained below:

   Given an input image and number of levels L of quantization:

   a) Choose an initial set R_1 of L output levels, initial average distortion T, $\varepsilon \leq 0$, and set the value of k = 1.

b) Partition the input pixels into regions D_k using the output set R_k according to the nearest neighbor condition, via, $D_k = \{d(f, r_k) \le d(f, r_l) \quad l \ne k\}$

c) From the partition regions found in step 2, find the new output levels R_{k+1} as the centroids of the input partitions.

d) Calculate the new average distortion due to the new output levels. If the absolute difference between the previous and current average distortions relative to the previous average distortion is less than ε, stop. The input decision boundaries are computed as the midpoints of the output levels. Otherwise, set k to k + 1 and go to step b.

2. Display the re-quantized images at 3 and 5 bpp and observe the differences, if any.
3. Plot the SNR due to quantization for bits 1–7 .
4. Draw a plot of the decision regions versus output levels of your quantizer for L = 32 levels.
5. Does non-uniform quantizer perform better than the uniform quantizer for the same number of bits of quantization?

LAB 7

Frequency-domain filtering

Introduction

The Fourier transform is a mathematical transformation which can be used to find the frequency information of a digital two dimensional image. The result of applying the Fourier transformation to an image is a two dimensional matrix which can also be represented as an image. MATLAB uses a function called **fft2** to calculate the 2D Fourier transformation.

Before designing a filter in frequency domain, you should understand what frequencies of the image need to be filtered. In other words, you should decide whether you need a low-pass, band-pass or high pass filter. You also need to choose the type of filter that you want to implement. Ideal filter, Gaussian or Butterworth are examples of different types of filter. Depends on the filter type, you must decide the parameter of the filter.

Low pass filters attenuate the high frequency of the image. The result of applying a low pass filter to an image is blurring. The degree of blurring depends of the filter type and the parameters used in a particular filter type. High pass filters attenuate the low frequency of the image. The result of applying a high pass filter to an image is sharpening. The degree of sharpening depends on the filter type and the parameters used in a particular filter type.

Ideal filter have sharp transition from pass band to stop band which result on undesirable image artifacts such as ringing artifacts. Gaussian filters have smooth transition from pass band to stop band. Butterworth filters have some parameters that can change the shape of their transition from pass band to stop band.

Task1 - Display the spectrum of your image

1- Load an image and generate its Fourier transform coefficients. Put the coefficients in an array called FT-Cof. Use “Cameraman.png” as your image. Use **fft2** function to calculate the fourier transformation. It should be noted that you need to convert the image to a data-class accepted as an input to fft2 function such as double data-class.
2- Are the FT coefficients are in [0 255] range. Find out the minimum and maximum values of the FT coefficients calculated in step 1.
3- Display the spectrum of your image as an image. To do so, you need to
    a. Shift the DC component to the center of the image using ft_shift=fftshift(ft) function
    b. Remap the result to grayscale [0 255] range using options of imshow function. Such as imshow (log(1+abs(ft_shift),[]))
4- If you would use imshow ((ft_shift),[])) instead of what you used in 3.b, how the plot would look like? Can you explain why the remap used in 3.b using the log function is better ?

Task 2- Display the distance matrix

1. To specify and implement frequency domain filters, you must generate a matrix that represents the distance of each pixel from the center of the image. Write a MATLAB function for this purpose. Call the function distmatix. This function should generate a distance matrix , the same size of the image.
2. Plot a 3D-mesh plot of the distance matix. Note that you need to shift the distance matrix before plotting it..

Task 3- Low pass filter design

Part A: Create an ideal low pass filter

1- Create an ideal low pass filter which can be applied to image i. use the following code to create the filter. Explain how this code can generate an ideal low pass filter.

```
[M,N]=size(i);
D=distmatrix(M,N);
H=zeros(M,N);
radius =35;
ind=D<= radius;
H(ind)=1;
HD=double(H);
```

2- Plot the filter's frequency response using

```
Imshow( fftshift(H)),Title('ideal low pass filter');
```

3- Apply the filter to your image. If i_dft is the DFT of your image, then you can use the following command for this purpose.

```
DFT_Filt = HD.* i_dft;
```

4- Convert the FT of the filtered image back to spatial domain by

```
i2=uint8(real(ifft2(DFT_Filt)));
```

5- How the filtered image compare to the original image. Can you see any artifacts?

Part B: Create a Gaussian low pass filter

1- Create a Gaussian low pass filter which can be applied to image i. use the following code to create the filter. Explain how this code can generate an ideal low pass filter.

```
Sigma= 30;
H_gua= exp(-(D.^2)/(2*(Sigma^2)));
figure, imshow(fftshift(H_gua)), title('Gausian Low Pass');
```

2- Repeat steps 2-5 of part A for Gausian low pass filter.

Part C: Create a Butterworth low pass filter

1- Create a Butterworth low pass filter which can be applied to image i. use the following code to create the filter. Explain how this code can generate an ideal low pass filter.

```
d0=35, n=3;
H_but=1./( 1 + (D./d0) .^(2*n) );
figure, imshow(fftshift(H_but)), title('Butterworth Low Pass');
```

2- Repeat steps 2-5 of part A for Butterworth low pass filter.

Task 4- High pass filter design

The techniques that you need to use to design a high pass filter are somehow similar to the ones used in the design of low pass filter. However, there is an important issue that must be considered in the design of high pass filters. A high pass filter attenuates low frequency, including the DC component of the image. This will result to set the average value of the image to zero. This problem can be solved using a technique known as high frequency emphasis filtering. In this technique after applying the high pass filter, the result in frequency domain is multiplied by a constant (let say b) and then be added by another constant (let say a).

Part A: Create an ideal high pass filter

1- Create an ideal low pass filter which can be applied to image i. use the following code to create the filter. Explain how this code can generate an ideal low pass filter.

```
H=ones(M,N);
radius=30;
ind=D<=radius;
H(ind)=0;
a=1;
b=1;
HD=double(a+(b.*H));
```

2- Plot the filter's frequency response using

```
Imshow( fftshift(HD),Title('ideal high pass filter');
```

3- Apply the filter to image I, using the following command

```
DFT_Filt = HD.* i_dft;
```

4- Convert the FT of the filtered image back to spatial domain by

```
i4=real(ifft2(DFT_Filt));
```

5- How the filtered image compare to the original image. Can you see any artifacts?

Part B: Create a Gaussian high pass filter

3- Create a Gaussian high pass filter which can be applied to image i. use the following code to create the filter. Explain how this code can generate an ideal low pass filter.

```
Sigma= 30;
H_gua= 1- exp(-(D.^2)/(2*(Sigma^2)));
H-gua_hfe=a+(b.*H_gau);
figure, imshow(fftshift(H_gua)), title('Gausian high Pass');
```

4- Repeat steps 2-5 of part A for Gausian high pass filter.

Part C: Create a Butterworth high pass filter

1- Create a butterworth high pass filter which can be applied to image i. use the following code to create the filter. Explain how this code can generate an ideal low pass filter.

```
d0=30, n=2;
H_but=1./( 1 + (d0./D) .^(2*n) );
H_but_hfe = a+(b.*H_but);
```

2- Repeat steps 2-5 of part A for Butterworth high pass filter.

LAB 8

DCT transformation

Introduction

The most common DCT definition of a 1-D sequence of length N is

$$C(u) = \alpha(u)\sum_{x=0}^{N-1} f(x)\cos[\frac{\pi(2x+1)u}{2N}] \qquad \alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \mu = 0 \\ \sqrt{\frac{2}{N}} & \mu \neq 0 \end{cases}$$

It can be seen from this formula that the first transform coefficient $C(u=0)$ is the average value of the sample sequence. In literature, this value is referred to as the DC Coefficient. All other transform coefficients are called the AC Coefficients.

Lets ignore the f(x) and α(u) and concentrate about the term $\sum_{x=0}^{N-1}\cos[\frac{\pi(2x+1)\mu}{2N}]$. If N=8, Figure 8-1 shows the plot of this term for different values of u. The top-left waveform of Figure 8-1, is for $(u=0)$ which shows the DC value, whereas, all other waveforms (u = 1,2,…,7) give waveforms at progressively increasing frequencies.

These waveforms are called the *cosine basis function, or DCT basis images*. Note that these basis functions are orthogonal. Hence, multiplication of any waveform shown in Figure 8-1 with another waveform followed by a summation over all sample points yields a zero (scalar) value, whereas multiplication of any waveform in Figure 8-1 with itself followed by a summation yields a constant (scalar) value. Orthogonal waveforms are independent, that is, none of the basis functions can be represented as a combination of other basis functions.

For image processing application, we are interested in 2-D DCT transforms. The 2-D DCT is given by

$$C(u,v) = \alpha(u)\,\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos[\frac{\pi(2x+1)u}{2N}]\cos[\frac{\pi(2y+1)v}{2N}] \qquad u,v = 0,\ 1,\ 2,\ ...,\ N-1$$

The 2-D basis functions can be generated by multiplying the horizontally oriented 1-D basis functions (shown in Figure below) with vertically oriented set of the same functions.

Explain the difference between DCT and DFT? Why DCT is more useful in image processing?

Task 1: Compute and display the 8 basis images for the 1D transforms DCT

1- Write a MATLAB code that produce the basis function shown in Figure 8-1
2- Write a MATLAB program that shows basis functions are orthogonal

Task 2: Compute and display the 8 × 8 basis images for the 2D transforms DCT

1- Write a MATLAB code to compute and display the 8X8 basis images for 2-D DCT.
2- For better demonstration, magnify each 8 × 8 basis image by a factor of 2 in both dimensions and scale the values between 0 to 255.

Task 3: Read an intensity image and expand it in terms of the DCT basis images

1- Read in the Barbara image, compute the 2D DCT of each 8 × 8 block using the raster scanning pattern, and expand each such block in terms of the computed basis images.
2- Once the entire image is reconstructed using the basis images, calculate the MSE between the original image and the reconstructed image. Plot the mean square error (MSE) in reconstructing the image using from one to N × N basis images. Take N to be 8.
3- To visualize the difference in the quality of the reconstructed image, use 1 and 13 basis images in the reconstruction

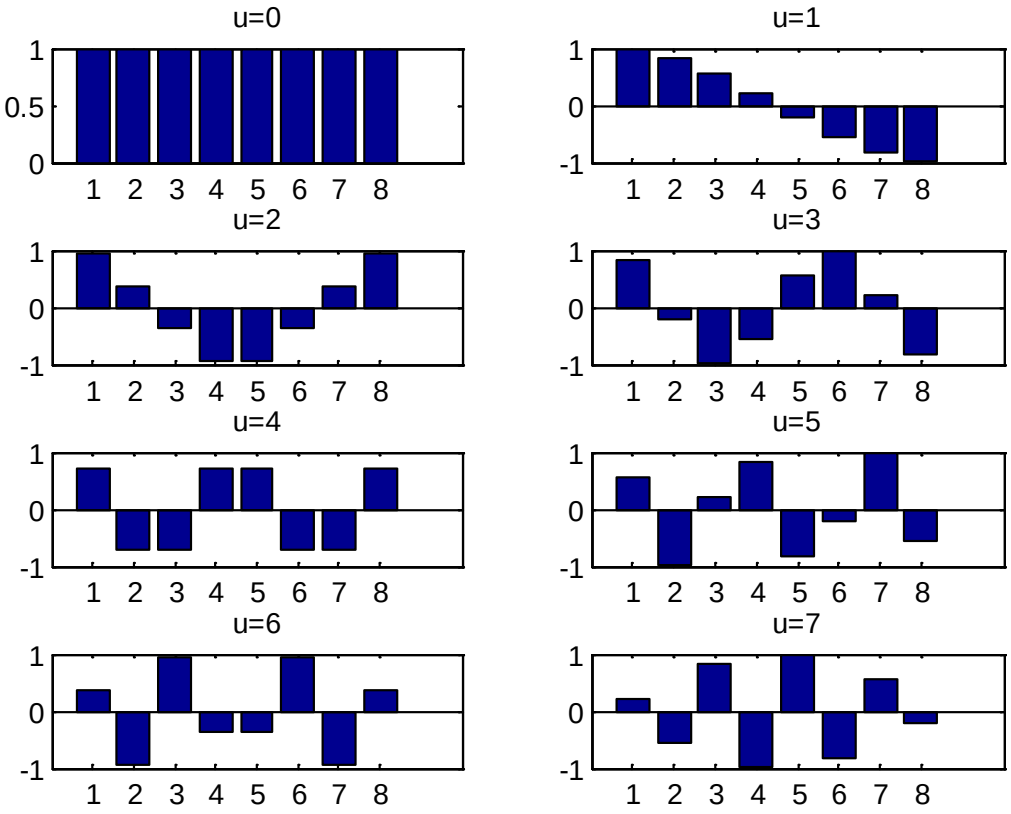


Figure 8-1

LAB 9

Noise Image restoration and Removal

Introduction

Noise is an undesired artifact that may contain in an image. The presence of noise in an image can be due to several sources. For instance, thermal noise can be generated in sensors, while periodic noise can be generated in communication channels. There are also various types of noise. Noise can be treated as a random variable whose probability density function (PDF) or histogram describes the shapes and distribution across the range of pixel values. Examples of noise with specific PDF function are Gaussian noise, Impulse noise (Salt and pepper), uniform noise, Rayleigh noise, Gamma noise and Exponential noise.

Task 1: PDF graph of different noise

1- Plot the PDF function of the following noise signals using MATLAB

- Uniform noise

$$P_u(z) = \begin{cases} \dfrac{1}{b-a} & a \le z \le b \\ 0 & otherwise \end{cases}$$

- Salt and pepper noise

$$P_{sp}(z) = \begin{cases} P_p & z = p \\ P_s & z = s \\ 0 & otherwise \end{cases}$$

- Gaussian noise

$$P_g(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(z-\bar{z})^2/2\sigma^2}$$

- Rayleigh noise

$$P_r(z) = \begin{cases} \dfrac{2}{b}(z-a)e^{-(z-a)^2/b} & z \ge a \\ 0 & otherwise \end{cases}$$

- Gamma or Erlang noise

$$P_E(z) = \begin{cases} \dfrac{a^b z^{b-1}}{(b-1)\perp} e^{-az} & z \ge 0 \\ 0 & otherwise \end{cases}$$

- Exponential noise $$P_{\exp}(z) = \begin{cases} ae^{-az} & z \geq 0 \\ 0 & otherwise \end{cases}$$

2- Load an image, add Gaussian noise to the image and display the original image and the image with noise. Use **imnoise(I,'gaussian',0,0.001)** function to add Gaussian noise to image I. Also add salt and pepper noise using the **imnoise** function.

3- Arithmetic mean filter, also known as averaging filter is a simple process of replacing each pixel value in the image with the average of NxN window surrounding the pixel. Apply an averaging filter (arithmetic mean filter) to the image and explain the result in the following cases:
   a. Using **fspecial** and **imfilter** functions with a default window of 3x3
   b. Using **fspecial** and **imfilter** functions with a default window of 5x5

Task 2: Filter the image which has noise

1- Load the \cameraman" image in MATLAB and assign it to the variable f. Contaminate the image with zero mean additive white Gaussian noise (AWGN) of standard deviation 20 using the command g=f+20*randn(size(f)). The image g is to be de-noised using the operation

$$\hat{f}[n,m] = \frac{\sum_{k=-L}^{L}\sum_{l=-L}^{L} h[k,l]\, g[n-k,m-l]}{\sum_{k=-L}^{L}\sum_{l=-L}^{L} h[k,l]}$$

We would like to apply the following three filters.

$$h_1[k,l] = \exp\ (-\frac{k^2+l^2}{2\sigma^2})$$

$$h_2[k,l] = \exp\ (-\frac{\{g(n-k,m-l)-g(n,m)\}^2}{2\rho^2})$$

$$h_3[k,l] = \exp\ (-\frac{k^2+l^2}{2\sigma^2})\ \exp\ (-\frac{\{g(n-k,m-l)-g(n,m)\}^2}{2\rho^2})$$

The filter **h1** is the Gaussian filter. The filters **h2** and **h3** correspond to the well-known "Sigma" filter and the "Bilateral" filter.

1- Write a MATLAB code to filter the image g using each of the filters. For h1, use σ = 1 and L = 3. For h2, use ρ = 40 and L = 3. For h3, use σ = 2, ρ = 50 and L=6. Use symmetric boundary condition to find the estimate of the pixel intensities at the boundaries (Hint: You can use the function padarray.m for the symmetric boundary extension).

2- Display the resulting images and compute the mean squared error in each case.

3- Comment on your results. Which filter do you think has been able to eliminate the noise as well as preserve edges and why?

LAB 10

Motion estimation

Introduction

In a video sequence, the motion estimation between two consecutive frame can be estimated using various algorithms. One of these algorithms is the Exhaustive search block matching (EBMA).In this method, for each block in the first frame, all candidate blocks in the second frame will be searched and the one with the minimum amount of error will be selected. The idea is that the block in the first frame has been moved to the selected block in the second frame. The vector representing this displacement is called the motion vector.
The candidate blocks in the second frame are within a certain range from the corresponding location of the block in the first frame. The candidate blocks are with one-pixel displacement from each other.
EBMA algorithm is very slow. So many fast algorithms have been designed to estimate the motion vectors faster. Examples of these algorithms are 2D log search and three step search method.

Task 1: Reading a video clip and calculating the motion vectors

1- Read two consecutive frame of ‘gforman.avi ‘ video. For example frames 7 and 8. Crop the frames to the size of 352 by 288. Find out if the frames are in color format or grayscale format. If they are in color format, change them to gray-scale format.

```
FirstVideo = VideoReader('gforeman.avi');
Firstframe= read(FirstVideo, 7);
Secondframe= read(FirstVideo, 8);
```

2- Display these frames

```
figure
Subplot(1,2,1), imshow(Firstframe), title( ‘first Frame’);
Subplot(1,2,2), imshow(Secondframe), title( ‘second Frame’);
```

3- Estimate the motion between these two frames using EBMA algorithm. For this purpose, write a MATLAB m file to do the EBMA motion estimation. Record the time to process the motion using this method in a variable.

```
blocksize=[8 8];
p=16;
FrameHeight=352;
FrameWidth=288;

tic
[predictFrame, mv_d, mv_o]= ebma( Secondframe, Firstframe, blocksize,
p, accuracy);
Process_time=toc
```

4- What class is the **predictFrame** frame? Is it the same dataclass as FirstFrame and SecondFrame? If it is not the same class, convert it to the same dataclass as other ones.

5- Display the motion vectors overlaid on the first frame using the following commands

```
figure, imshow(Firstframe)
hold on
quiver(mv_o(1,:), mv_o(2,:), mv_d(1,:),mv_d(1,:)), title('Motion vectors using EBMA');
```

6- Display the predicted frame and calculate the error by subtracting the predicted frame from the second frame. Also calculate PSNR .

```
figure, imshow(predictFrame), title('Predicted frame');
errorframe=imabsdiff(Firstframe, predictFrame);
PSNR=10*log10 (255*255/mean(mean((errorframe.^2))))
```

7- Change the blocksize=[8 8] to [4 4],[16 16] and [32 32]. Record the process time and PSNR. Explain the result. What is the best block size for calculating motion estimation?

8- Calculate the size of mv_o and mv_d motion vectors when different block size is used in Step 7. Use MATLAB to find the size of these vectors.

9- Change the value of p to 8. Run your program and find out the process time , the value of PSNR and the size of motion vector. Explain about your result.