2D Linear imaging systems

Aim

The aim of this exercise is to learn how the general imaging process works, how the observed image is a filtered version of the real object. The exercise should also provide basic knowledge of 2D filtering and 2D Fourier analysis using Matlab.

Deadline

As announced on It's Learning.

Introduction

If you see an object through a camera lens, binoculars or similar, and the lens is not properly focused, the image will appear diffuse. This is an example of how an imaging system (the lens) affects the appearance of the object.

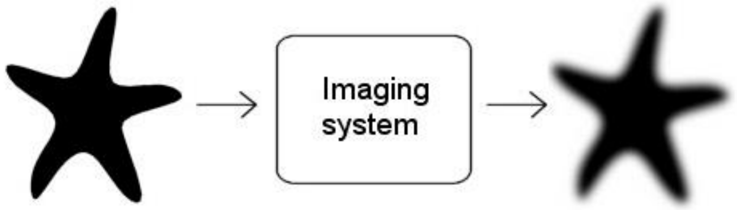

Figure 1: The imaging system affects the way the object appears.

An imaging system can be described by its point spread function (PSF). The PSF describes how an infinitely small point object is seen through the imaging system. Assuming a linear imaging system, one can build the total image of the object by summing PSFs of the individual points that together form the total object. This process is a convolution sum between the real object and the PSF. The convolution sum is identical to linear filtering, with the PSF as the filter.

$$\text{image} = \text{object} \otimes \text{PSF}$$

In this exercise we will define our own PSF and see how it affects an image. We will use the same method of analysis as in the *Pulse-echo exercise*, but in two spatial dimensions instead of one temporal dimension. That is, we will use 2D Fourier analysis in the *x* and *y* directions in the image. By dealing with two dimensional images, the convolution process can be seen as 2D linear filtering. The PSF can be defined as the filter in exercise one, but now with two dimensions. The images we will use are all digitized grey scale images, where each point in the image has a given brightness value. An example of such an image is given in *Figure 2.*

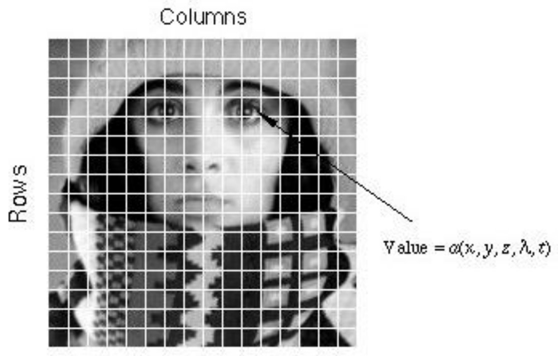

Figure 2: Digitization of a continuous image. The pixel at coordinates [*m*=10, *n*=3] has the integer brightness value 110.

Part one: 2D Fourier transform.

In this part we will play a litte with the 2D Fourier transform. The spatial domain Fourier transform is exactly the same as the time domain transform, but for some reason it seems harder to visualize spatial frequency. It is defined continuously as:

$$\Im\{f(x,y)\} = F(u,v) = \int\int_{-\infty}^{\infty} f(x,y)\exp[-j2\pi(ux+vy)]dxdy$$

And the inverse transform is defined as:

$$\Im^{-1}\{F(u,v)\} = f(x,y) = \int\int_{-\infty}^{\infty} F(u,v)\exp[j2\pi(ux+vy)]dudv$$

1. Use Microsoft Paint to draw your favorite lines, and save your images as a .bmp files to your working directory. Make several images with just lines that are horizontal, vertical and rotated and with different line widths (ex.5-6 images). Draw the lines in different positions on screen. Use white lines on a black background and save the images as monochrome .bmp files (choose this in the save-file dialog).
2. Load an image you've made into the Matlab environment and show it in a figure window. Hints: (type *help <function>* in Matlab to get help on a function)
   a. Use *imread()* to load the image, and *imagesc()* to show it in a figure. Use *double()* to convert the data from integer to double precision format.
   b. Use *colormap(gray(255))* to set the color map to gray scale.
3. 2D Fourier transform the image and show it in a Matlab figure window. The image of the transform must be shown using a decibel scale. Hints:
   - Use *fft2()* to obtain the 2D Fourier transform.
   - Use *fftshift()* to get zero-frequency in the middle of the image.
   - Normalize to 1 as maximum value, compress logarithmically, and scale the data before showing it using the *image()* (not *imagesc*) function. See *(*)* for more on this.
   - Don't forget to set the color map to gray scale.

   (*) To scale the image properly we set the dynamic range manually. This is done using the following matlab code:

   normGs = the normalized Fourier transform.
   dynRange = 60; % dynamic range in dB

NGray = 255; % number of grays in colormap.
imagePowerSpecter = NGray*(1 + 20*log10(normGs)/dynRange);
Using this the maximum value (0 dB) wil be NGray, and –dynRange dB will be zero.

4. Repeat the steps 2-3 for all the images you made in step 1.

5. What do you observe? Use the results to imply the translational and rotational properties of the 2D Fourier transform.

Part two: 2D Low pass filtering.

In this part we will make the PSF as a low pass filter in both the *x* and the *y* direction in the image, and see how this imaging system will affect the appearance of an object.

1. Load the image **oving2_lena.jpg** into Matlab and show it in a figure window. Assume that the image proportions is 30cm in height and width. Make the axis accordingly.
   Hints:
   To get the right proportions in the figure use the command *axis('image')* after using the *image()* command. Use this on all figures including the power-spectrum figures.

2. Apply the 2D FFT to the image and show the result in a Matlab figure. The axis shall show the spatial frequency.

3. Design the PSF as a 2D low pass filter, i.e. make a 2D low pass filter**.**
   Hints:
   A simple low pass filter is the neighborhood averaging filter. This filter makes a specific pixel have the average value of the pixels in the region around it. To implement it, we make a matrix containing the weights of the surrounding pixels. These weights correspond to the filter coefficients in a FIR filter. To ensure proper scaling, the total pixel value should be scaled by the number of pixel values used as input to the filter. If *f(x,y)* is the current pixel, its value will be given by the following formula:

$$f(x,y) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha(i,j)\cdot f\left(x+i-\left\lceil\frac{N}{2}\right\rceil, y+j-\left\lceil\frac{N}{2}\right\rceil\right)$$

   Where α (*i, j*) contains the filter coefficients, and *i* and *j* are the indices in the 2D filter matrix. *N* is here an odd number which gives the size of the matrix.
   The coefficients of our matrix are all equal to one divided by the total number of coefficients. To make this matrix use the *ones(...)* function in Matlab.

4. Apply the 2D FFT to the filter matrix and show it in a matlab figure. What can you say about how the filter will work in the u and v directions by inspecting this power spectrum?

5. Filter the image using the filter from 3.
   Hints:
   This can be done in two ways. Either by doing the convolution directly in the time domain using conv2(…'same'), or filter2(…). Or by multiplication of the Fourier transforms in the frequency domain and then applying the inverse transform using ifft2(…).

Convolution in the time domain is multiplication in frequeny domain and vice versa. If any ”complex problems” appears remember to take the absolute value.

6. Apply the 2D FFT to the resulting image and show the result in a matlab figure. The axis shall show the spatial frequency. What has happened?

7. Show the resulting image in a matlab figure. What has happened? What happens if you vary the size of the PSF?