

Introduction to Digital Image Processing

Exercise No. 2: Frequency domain

Due: 18-12-05

Purpose

The purpose of this exercise is to help you understand the concept of the frequency domain by doing some manipulation of images. The exercise will have 4 parts : first two parts on basic manipulation of images in the frequency domain and the second two parts are about image restoration: overcoming blur in images. The last part of the exercise (Inverse-Filtering in the spatial domain) is optional and you can get a bonus of 20 points for doing it.

Image Resize using the frequency domain

In this section you are required to perform image resizing in the frequency domain. Your program should get any image, convert it to gray scale and resize it by a given factor. The resized image should be displayed as grayscale image. You should use the function `my_im_read` you wrote in the first exercise to read the image and convert it to grayscale (please correct any errors you may had with your previous code). The resizing operation should be done in the frequency domain in the way that was explained in class.

The function should be called as follows.:

```
ex2resizeImg(name, factor)
```

where:

- **name** is the input image filename
- **factor** is a positive scale factor

This function should not return any arguments but should open 2 figures displaying the image and the resized image. The function should call an internal function as follows

```
f = ex2resizeFreq(im, factor)
```

where:

- **im** is the grayscale input image.
- **factor** is a positive scale factor
- **f** is an image which is the result of the resize operation

The function should not display anything and it should return the grayscale image in the appropriate size, e.g. if the factor is 1.5 the image should be 1.5 times bigger than the original image and if the factor is 0.3 the function should return the image in 30% of the original image size (round to the nearest integer).

Please note that your program should receive images from odd and even sizes and it should work properly on both types. It is expected that an image that was resized in factor f (f is larger than 1) and the resulting image was resized again in factor $1/f$ will produce image very similar to the original image (up to quantization errors).

In the README file you should describe what operations you did in the frequency domain and what equivalent operations should have been done if the resizing was performed in the spatial domain.

Find image translation parameters using the spatial domain

In this section you should find the translation parameters of an 2 images $im1$, $im2$ where $im2$ was received after translating $im1$ in $(x0,y0)$ pixels (e.g. $im1(x,y) = im2(x+x0,y+y0)$). Finding the translation parameters should be done in the frequency domain, base on the translation property of the Fourier Transform (as you proved in the theoretical exercise).

It is sufficient to use a very small number of pixels (2 pixels) wisely selected to find those parameters or in order to get a more stable solution solving larger equation system. You are required to perform the simple option (solving based on value of 2 pixels). To test your solution you should generate your own test images, both synthetic and real images and and in the README file analyze the performance of this solution: under what conditions did you get good estimation for the translation.

The function should be called as follows:

`[u0, v0] = ex2findTrans(name1, name2)`

where:

- **name1** is the first input image ($im1$) filename
- **name2** is the second input image ($im2$) filename (the first image with translation of $u0,v0$)

The function should return $u0$ and $v0$ which are the translation component in X and in Y respectively.

Inverse-Filtering

Image blur happens, for example in cheap web cameras (due to bad optics), in bad weather, and when the camera moves quickly during the exposure (motion blur). In many cases, as we showed on class, when the blur is uniform over the image, it can be modeled by a convolution:

$$g = f * h + n$$

where:

- * marks convolution.
- g is the input blurry image.
- f is the original unknown image.
- h is the convolution blur filter.
- n is the noise added to the image.

So in order to estimate image g from image f the following simplifying assumptions are made:

- The convolution blur h is known.
- The noise is independent of the image.
- The noise pixels are identically independently distributed with normal distribution and zero mean.
- The noise looks similar to what you see on an un-tuned television. In the frequency domain, this means that typically the power spectrum of the noise image is close to uniform.

This task is called inverse-filtering or de-convolution, since one tried to do the inverse of the convolution blur filter.

Inverse-Filtering in the frequency domain

The imaging process can be presented in the frequency domain by:

$$G(u, v) = F(u, v)H(u, v) + N(u, v)$$

where:

- G is the Fourier transform of the input image g .
- F is the Fourier transform of the unknown original image f .
- H is the Fourier transform of the blur filter h .
- N is the Fourier transform of the noise added to the image.

Thus in order to recover the image, one can use the following:

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)} = F(u, v) + \frac{N(u, v)}{H(u, v)}$$

Let us mark

$$D(u, v) = \frac{N(u, v)}{H(u, v)}$$

Using $\hat{F}(u, v)$ is problematic, since in frequencies where $|D(u, v)|$ is very large, we get more noise than information. So we need to assume something about the noise and the image. We will assume that for low frequencies, $|F(u, v)|$ is much larger than $|D(u, v)|$, while for high frequencies it is the other way around. There are several reasons for this assumption:

- h is a low pass filter, and so $|H(u, v)|$ is large for low frequencies and small for high frequencies.
- A typical power spectrum of a “natural” image is exponentially decreasing, i.e. it is very large for low frequencies and close to zero for high frequencies.
- The power spectrum of noise is flat or uniform. This is typical to noise like the one in an un-tuned television. Note: in this exercise you don’t need to estimate the noise spectrum explicitly.

Thus, in order to cancel the blur, while avoiding amplification of the noise, we use $\hat{F}(u, v)$ when it is reliable, and otherwise use $G(u, v)$. We will use one of 2 methods for handling the noise: direct inverse filter and wiener inverse filter.

In this part you should write a program that performs inverse-filtering in the frequency domain. It should be called as follows.:

`f=ex2_freq(useWeiner, g, h, tu, tv, K, origIm)`

where: **useWeiner** is a parameter either 1 or 0 representing whether to restore the image using wiener restoration method (1) or using direct method (0). **g** is the input image filename and **h** the blur filter (a matrix). **f** is the restored image. The **thresholds** tu, tv are used only in the direct inverse filtering to define which frequencies are considered low (so $\hat{F}(u, v)$ is used) and which are high (so $G(u, v)$ is used). (u, v) is a low frequency iff $|u| < tu, |v| < tv$. Note that we refer to frequencies on the shifted representation, where the frequency $(0, 0)$ is in the middle of the image. See 'fftshift' in matlab for usage of this representation. The ranges of tu and tv are $[0, 1]$. For example, $(0, 0)$ means a window of size 0 (no restoration), $(1, 1)$ refers to the whole image (restore all frequencies), and $(0.5, 0.5)$ means a square window in the middle of the image. **K** is a parameter used when the wiener filter method is to be used and it is represent a constant representing the factor between the spectrum of the image to the spectrum of the noise (see the definition of the wiener filter as defined in class). **origIm** is an optional argument: original image filename used for testing the restoration result when we have the original image available.

The function should also display in one figure (using subplot):

1. The blurred image g .
2. The restored image f .
3. original Image before degradation (if given).
4. The log of the power spectrum of g .
5. The log of the power spectrum of f .

6. The log of the power spectrum of the original image (if given)

You can find in the following web page under the directory Ex2Examples an example of blurred image and a program showing how it was created. You can use it to test your program

Inverse-Filtering in the spatial domain

Note: This part is a bonus part and you can get up to 20 point. The de-convolution problem can be presented in the spacial domain as a linear set of equations that need to be solved:

$$g(x, y) = \sum \sum f(x - \delta x, y - \delta y)h(\delta x, \delta y)$$

So that each pixel in the input image gives one equation. Since there is a large number of unknown variables (as the size of the image), a simple inversion of the equations matrix is computationally expensive. Moreover, the frequency domain teaches us that if we invert the matrix, in high frequencies the noise will be amplified.

Instead, we use a simple iterative method. We start with an initial solution (e.g. $f^{(0)} = g$), and iteratively update the solution by the formula:

$$f^{(n+1)} = f^{(n)} + (g - f^{(n)} * h) * h'$$

Where h' is taken to be the reflected version of h (For symmetric kernels, $h' = h$). when this system converges, that is $f^{(n+1)}$ equals $f^{(n)}$ up to a small error, we get that: $(g - f^{(n)} * h) * h' = 0$, so $f^{(n)}$ satisfies our model.

There is another detail missing, which is what to do in the image borders. One way, is to follow the convolution assumption that the image is cyclic. Another way is to assume that out of the borders of the image there are black pixels (We will call it *zero padding*). A third way is to duplicate the border. For example, for an image f of size 10×10 , we assume $f(-3, 2) = f(1, 2)$, $f(13, 2) = f(10, 2)$, $f(3, 12) = f(3, 10)$ etc.

In the second part of the restoration section you will implement this method. it should be called:

`f=ex2_spatial(g,h,thresh,borderMethod)`

where:

- **g** is the input image filename.
- **h** is the blur filter
- **thresh** is the threshold for convergence. The algorithm should stop when

$$\frac{1}{S} \sum_{x,y} (f^{(n+1)}(x, y) - f^{(n)}(x, y))^2 < thresh$$

where S is the image size (the number of pixels in the image).

- **borderMethod** is the way the border should be treated. It can be either -1 (Cyclicity), or 0 (Zero-Padding) or 1 (Duplicate border)

Your program should return the result. It should not display any image.

Readme and tips

Your README file should include your analysis of question asked in the first and second parts of the exercise (if you prefer you can hand the theoretical analysis not as a README text file but in another format like a word document. If you do so please state it in your README file). It should also include the difference between the different possibilities of inverse filtering:

A. In the frequency domain, what is the difference between the results using wiener and direct inverse filtering? Which method would you use?

B. In the frequency domain, what is the effect in changing t_u, t_v ? What were the values of t_u and t_v that gave you the best results ? What is the value of K that gave you good results if you implemented part 4:

C. What is the difference between the results in the spatial domain and frequency domain ? Which method would you use?

D. In the spacial domain, what is the difference between the results when using different border strategies (cyclic image, duplicate, zero padding)? What happens in the image and near the borders when the threshold you use is very small?

In order to help you get to know some useful matlab functions related to this exercise you are advised to look on the help for the following functions: `fft2`, `fftshift`, `ifftshift`, `random`, `find`, `conv2`, `imfilter`.