**Introduction to Image Processing**

**Lab 6**

## 1. Discrete Fourier Transform

This is a brief review of the Fourier transform. An in-depth discussion of the Fourier transform is best left to your class instructor.

The general idea is that the image (f(x,y) of size M x N) will be represented in the frequency domain (F(u,v)). The equation for the two-dimensional discrete Fourier transform (DFT) is:

DFT equation

The concept behind the Fourier transform is that any waveform can be constructed using a sum of sine and cosine waves of different frequencies. The exponential in the above formula can be expanded into sines and cosines with the variables u and v determining these frequencies.

The inverse of the above discrete Fourier transform is given by the following equation:

Inverst DFT equation

Thus, if we have F(u,v), we can obtain the corresponding image (f(x,y)) using the inverse, discrete Fourier transform.

Things to note about the discrete Fourier transform are the following:

* the value of the transform at the origin of the frequency domain, at F(0,0), is called the dc component
  + F(0,0) is equal to MN times the average value of f(x,y)
  + in MATLAB, F(0,0) is actually F(1,1) because array indices in MATLAB start at 1 rather than 0
* the values of the Fourier transform are complex, meaning they have real and imaginary parts. The imaginary parts are represented by i, which is defined solely by the property that its square is −1, ie:

http://www.cs.uregina.ca/Links/class-info/425/Lab5/Equations/imaginary_definition.png

* we visually analyze a Fourier transform by computing a **Fourier spectrum** (the magnitude of F(u,v)) and display it as an image.
  + the Fourier spectrum is symmetric about the origin
* the fast Fourier transform (FFT) is a fast algorithm for computing the discrete Fourier transform.
* MATLAB has three functions to compute the DFT:
  + fft -for one dimension (useful for audio)
  + fft2 -for two dimensions (useful for images)
  + fftn -for n dimensions
* MATLAB has three related functions that compute the inverse DFT:
  + ifft
  + ifft2
  + ifftn

### 1.1 How to Display a Fourier Spectrum using MATLAB

The following table is meant to describe the various steps behind displaying the Fourier Spectrum.

|  |  |
| --- | --- |
| **MATLAB code** | **Image Produced** |
| %Create a black 30x30 image  f=zeros(30,30);  %With a white rectangle in it.  f(5:24,13:17)=1;  imshow(f,'InitialMagnification', 'fit') | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/Spectrum/rectangle.jpg |
| %Calculate the DFT.  F=fft2(f);  %There are real and imaginary parts to F.  %Use the abs function to compute the magnitude  %of the combined components.  F2=abs(F);  figure, imshow(F2,[], 'InitialMagnification','fit') | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/Spectrum/rectangle_spectrum1.jpg |
| %To create a finer sampling of the Fourier transform,  %you can add zero padding to f when computing its DFT  %Also note that we use a power of 2, 2^256  %This isbecause the FFT -Fast Fourier Transform -  %is fastest when the image size has many factors.  F=fft2(f, 256, 256);  F2=abs(F);  figure, imshow(F2, []) | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/Spectrum/rectangle_spectrum2.jpg |
| %The zero-frequency coefficient is displayed in the  %upper left hand corner. To display it in the center,  %you can use the function fftshift.  F2=fftshift(F);  F2=abs(F2);  figure,imshow(F2,[]) | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/Spectrum/rectangle_spectrum3.jpg |
| %In Fourier transforms, high peaks are so high they  %hide details. Reduce contrast with the log function.  F2=log(1+F2);  figure,imshow(F2,[]) | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/Spectrum/rectangle_spectrum4.jpg |

To get the results shown in the last image of the table, you can also combine MATLAB calls as in:

f=zeros(30,30);

f(5:24,13:17)=1;

F=fft2(f, 256,256);

F2=fftshift(F);

figure,imshow(log(1+abs(F2)),[])

Notice in these calls to imshow, the second argument is empty square brackets. This maps the minimum value in the image to black and the maximum value in the image to white.

## 2. Frequency Domain Versions of Spatial Filters

See section 14.3.5, 14.5.1, and 14.5.2 in your textbook

The following convolution theorem shows an interesting relationship between the spatial domain and frequency domain:

Convolution in the spatial domain

and, conversely,

Multiplication in the spatial domain is equivalent to convolution in the Frequency Domain.

where the symbol "\*" indicates convolution of the two functions. The important thing to extract out of this is that the multiplication of two Fourier transforms corresponds to the convolution of the associated functions in the spatial domain. This means we can perform linear spatial filters as a simple component-wise multiply in the frequency domain.

This suggests that we could use Fourier transforms to speed up spatial filters. This only works for large images that are correctly padded, where multiple transformations are applied in the frequency domain before moving back to the spatial domain.

When applying Fourier transforms padding is very important. Note that, because images are infinitely tiled in the frequency domain, filtering produces wraparound artefacts if you don't zero pad the image to a larger size. The paddedsize function below calculates a correct padding size to avoid this problem. The paddedsize function can also help optimize the performance of the DFT by providing power of 2 padding sizes. See paddesize's help header for details on how to do this.

### 2.1 Basic Steps in DFT Filtering

The following summarize the basic steps in DFT Filtering (taken directly from page 121 of Digital Image Processing Using MATLAB):

1. Obtain the padding parameters using function [paddedsize](http://www.cs.uregina.ca/Links/class-info/425/Lab5/M-Functions/paddedsize.m):   
   PQ=paddedsize(size(f));
2. Obtain the Fourier transform of the image with padding:  
   F=fft2(f, PQ(1), PQ(2));
3. Generate a filter function, H, the same size as the image
4. Multiply the transformed image by the filter:  
   G=H.\*F;
5. Obtain the real part of the inverse FFT of G:  
   g=real(ifft2(G));
6. Crop the top, left rectangle to the original size:  
   *g=g(1:size(f, 1), 1:size(f, 2));*

### 2.2 Example: Applying the Sobel Filter in the Frequency Domain

For example, let's apply the Sobel filter to the following picture in both the spatial domain and frequency domain.



\*Note, this code relies on [paddedsize.m](http://www.cs.uregina.ca/Links/class-info/425/Lab5/M-Functions/paddedsize.m)

|  |  |
| --- | --- |
| **Spatial Domain Filtering** | **Frequency Domain Filtering** |
| %Create the Spacial Filtered Image  f = imread('entry2.png');  h = fspecial('sobel');  sfi = imfilter(double(f),h, 0, 'conv');  %Display results (show all values)  figure,imshow(sfi, []);  Palm fourier-based thresholded edge map  %The abs function gets correct magnitude  %when used on complex numbers  sfim = abs(sfi);  figure, imshow(sfim, []);  Palm spatial edge directions  %threshold into a binary image  figure, imshow(sfim > 0.2\*max(sfim(:)));  Palm spatial edge magnitudes | %Create the Frequency Filtered Image  f = imread('entry2.png');  h = fspecial('sobel'); PQ = paddedsize(size(f)); F = fft2(double(f), PQ(1), PQ(2)); H = fft2(double(h), PQ(1), PQ(2)); F\_fH = H.\*F; ffi = ifft2(F\_fH); ffi = ffi(2:size(f,1)+1, 2:size(f,2)+1);  %Display results (show all values)  figure, imshow(ffi,[])  Palm  %The abs function gets correct magnitude  %when used on complex numbers  ffim = abs(ffi);  figure, imshow(ffim, []);  Palm fourier-based edge directions  %threshold into a binary image  figure, imshow(ffim > 0.2\*max(ffim(:)));  Palm fourier-based edge magnitudes |

You will notice that both approaches result in a similar looking, if not identical filtered image. You may have to adjust how you crop the image slightly as shown in the example above.

## 3. Frequency Domain Specific Filters

See section 14.5.3 in your textbook, and Chapter 4 and Section 5.4 in Digital Image Processing Using MATLAB

As you have already seen, based on the property that multiplying the FFT of two functions from the spatial domain produces the convolution of those functions, you can use Fourier transforms as a fast convolution on large images. Note that on small images it is faster to work in the spatial domain.

However, you can also create filters directly in the frequency domain. There are three commonly discussed filters in the frequency domain:

* Lowpass filters, sometimes known as smoothing filters
* Highpass filters, sometimes known as sharpening filters
* Notch filters, sometimes known as band-stop filters

### 3.1 Lowpass Filters

Lowpass filters:

* create a blurred (or smoothed) image
* attenuate the high frequencies and leave the low frequencies of the Fourier transform relatively unchanged

Three main lowpass filters are discussed in Digital Image Processing Using MATLAB:

1. ideal lowpass filter (ILPF)
2. Butterworth lowpass filter (BLPF)
3. Gaussian lowpass filter (GLPF)

The corresponding formulas and visual representations of these filters are shown in the table below. In the formulae, D0 is a specified nonnegative number. D(u,v) is the distance from point (u,v) to the center of the filter.

|  |  |  |
| --- | --- | --- |
| **Lowpass Filter** | **Mesh** | **Image** |
| Ideal:General Highpass Filter - relates Highpass filters to lowpass filters | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/lp_ideal_mesh.png | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/lp_ideal.png |
| Butterworth:http://www.cs.uregina.ca/Links/class-info/425/Lab5/Equations/butterworthLPF.gif | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/lp_btw_mesh.png | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/lp_btw.png |
| Gaussian:DFT | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/lp_gaussian_mesh.png | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/lp_gaussian.png |

### 3.2 Highpass Filters

Highpass filters:

* sharpen (or shows the edges of) an image
* attenuate the low frequencies and leave the high frequencies of the Fourier transform relatively unchanged

The highpass filter (Hhp) is often represented by its relationship to the lowpass filter (Hlp):

Gaussian Filter Equation

Because highpass filters can be created in relationship to lowpass filters, the following table shows the three corresponding highpass filters by their visual representations:

|  |  |  |
| --- | --- | --- |
| **Lowpass Filter** | **Mesh** | **Image** |
| Ideal | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/hp_ideal_mesh.png | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/hp_ideal.png |
| Butterworth | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/hp_btw_mesh.png | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/hp_btw.png |
| Gaussian | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/hp_gaussian_mesh.png | http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/hp_gaussian.png |

### 3.3 Notch Filters

Notch filters:

* are used to remove repetitive "Spectral" noise from an image
* are like a narrow highpass filter, but they "notch" out frequencies other than the dc component
* attenuate a selected frequency (and some of its neighbors) and leave other frequencies of the Fourier transform relatively unchanged

Repetitive noise in an image is sometimes seen as a bright peak somewhere other than the origin. You can suppress such noise effectively by carefully erasing the peaks. One way to do this is to use a notch filter to simply remove that frequency from the picture. This technique is very common in sound signal processing where it is used to remove mechanical or electronic hum, such as the 60Hz hum from AC power. Although it is possible to create notch filters for common noise patterns, in general notch filtering is an ad hoc procedure requiring a human expert to determine what frequencies need to be removed to clean up the signal.

## 4. Exercises

### Part 1: Identifying and Using High and Low Pass Filters

1. Download the following image **97.jpg** and store it in MATLAB's "Current Directory".   
   
2. Identify which of the following is the result of a lowpass or highpass Butterworth filter and reproduce the results.

|  |  |
| --- | --- |
| Image 1 | Image 2 |
| Lost my mind | Lost my mind: Blurry |

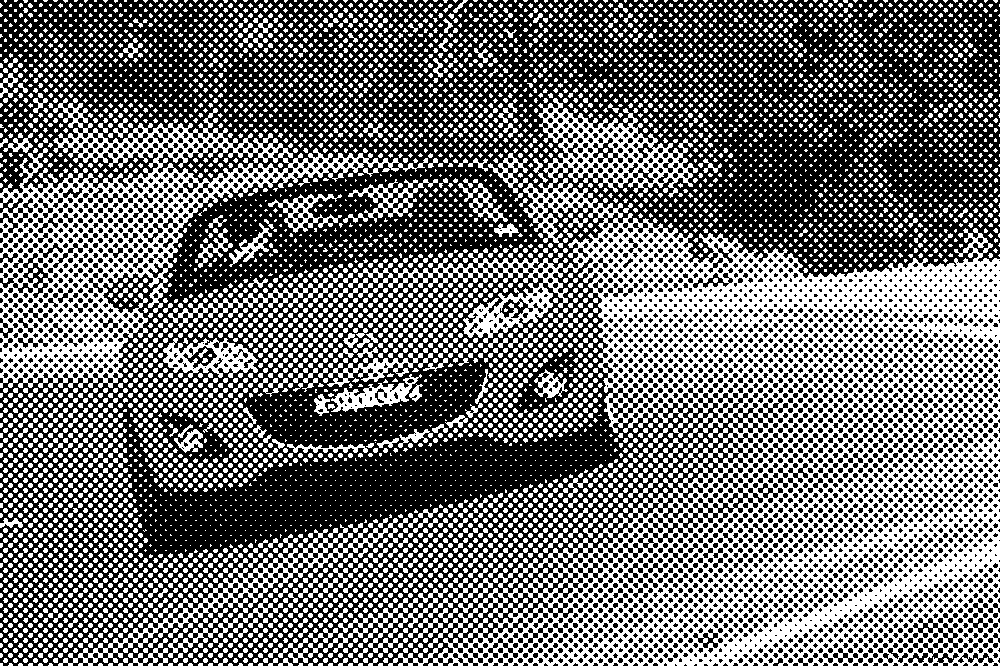
1. Display the Fourier spectrum for 97.jpg

#### Deliverables:

* Identification of high and low pass filters in above images
* Reproduced highpass and lowpass filter for 97.jpg
* Fourier spectrum for 97.jpg

### Part 2: CSI Style Image Enhancement

You are the image processing expert for a local police department. A detective has reopened a cold case and part of his evidence is a newspaper print of a car. You have been asked to do some CSI style magic to see if you can learn the suspect's license plate number or see his face.

1. Download the image **halftone.png** and store it in MATLAB's "Current Directory".  
   [](http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/halftone.png)
2. Use a set of notch filters to remove the peaks from the image's Fourier transform.   
   **TIPS:**
   * create a function that takes a list of peaks as an argument
   * the peaks form a repetitive pattern. Figure out the pattern to save time.
3. Fine tune your results by trying varying widths of the three notch filter types. Provide a best effort solution for each one. Also provide a blurring based solution in Photoshop.
4. The following is one possible solution:  
   [](http://www.cs.uregina.ca/Links/class-info/425/Lab5/Picts/sol.png)

#### Deliverables:

* A script that creates and displays 3 best effort notch filter sets for each of the three notch filter types.
* Photoshop/Gimp created blur of the original image.