**CAP 5400 - Digital Image Processing**

**Project 4: Fourier Transform and frequency filtering**

Yu Liang

Department of Computer Sc. and Engineering.

University of South Florida,

Tampa, Florida

**1. Introduction**

The purpose of this homework is to experiment with Fourier Transform and frequency filtering. My program apply F.T. to an image using OpenCV library. The program display absolute value of the amplitude of the Fourier domain as an image. Also my program apply inverse D.F.T. and display resulting image. This work on grey level images with one ROI. I can’t figure out how to apply two ROIs.

My program operate within specified ROI, the rest of image should not be affected. My program apply D.F.T. to each ROI and display absolute value of the amplitude of the Fourier domain before and after filtering operation. My program apply inverse D.F.T. and display resulting (combined) image.

My program add low-pass and high-pass filtering to your choice of options. Assume circular filter and allow user to input one cut-off frequency for low-pass/high-pass (filter does not affect the phase). It also adds band-stop filter by allowing user input for 2 cut-off frequencies.

**2. Description of algorithms**

**2.1. The Fourier Transform**

The Fourier Transform will decompose an image into its sinus and cosines components. In other words, it will transform an image from its spatial domain to its frequency domain. The idea is that any function may be approximated exactly with the sum of infinite sinus and cosines functions. The Fourier Transform is a way how to do this. Mathematically a two dimensional images Fourier transform is:

F(k,l) = \displaystyle\sum\limits_{i=0}^{N-1}\sum\limits_{j=0}^{N-1} f(i,j)e^{-i2\pi(\frac{ki}{N}+\frac{lj}{N})}

e^{ix} = \cos{x} + i\sin {x}

Here f is the image value in its spatial domain and F in its frequency domain. The result of the transformation is complex numbers. Displaying this is possible either via a real image and a complex image or via a magnitude and a phase image. However, throughout the image processing algorithms only the magnitude image is interesting as this contains all the information we need about the images geometric structure. Nevertheless, if you intend to make some modifications of the image in these forms and then you need to retransform it you’ll need to preserve both of these.

Make place for both the complex and the real values. The result of a Fourier Transform is complex. This implies that for each image value the result is two image values (one per component). Moreover, the frequency domains range is much larger than its spatial counterpart. Therefore, we store these usually at least in a float format. Therefore we’ll convert our input image to this type and expand it with another channel to hold the complex values.

Make the Discrete Fourier Transform. Transform the real and complex values to magnitude. A complex number has a real (Re) and a complex (imaginary - Im) part. The results of a DFT are complex numbers. The magnitude of a DFT is:

M = \sqrt[2]{ {Re(DFT(I))}^2 + {Im(DFT(I))}^2}

Switch to a logarithmic scale. It turns out that the dynamic range of the Fourier coefficients is too large to be displayed on the screen. We have some small and some high changing values that we can’t observe like this. Therefore the high values will all turn out as white points, while the small ones as black. To use the gray scale values to for visualization we can transform our linear scale to a logarithmic one:

M_1 = \log{(1 + M)}

Crop and rearrange. Remember, that at the first step, we expanded the image? Well, it’s time to throw away the newly introduced values. For visualization purposes we may also rearrange the quadrants of the result, so that the origin (zero, zero) corresponds with the image center.

Normalize. This is done again for visualization purposes. We now have the magnitudes, however this are still out of our image display range of zero to one. We normalize our values to this range using the normalize() function.

**2.2. Low, high, band pass filter**

Thus we were able to lowpass filter images (e.g. blur them) by making the camera lens go out-of-focus. In a sense, therefore, the camera lens is an example of a continuous analog lowpass filter. Now we will examine the effects of lowpass filters implemented by digital signal processing:

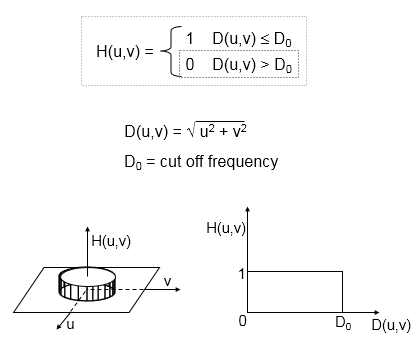
Typically, filters are classified by examining their properties in the frequency domain:

(1) Low-pass

(2) High-pass

(3) Band-pass

(4) Band-stop

****

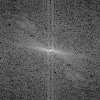
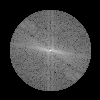
**3. Results**

**3.1. Sobel Edge Detection**

First we applied Low pass filter with threshold value of 40 on the lena image as below. Figure b shows the gradient amplitude as intensity image for Fourier domain before low-pass filter and after lowpass filter in ROI (60,60,100,100), T=40. We can see the result in image c.



1. Original image

1. Fourier domain before low-pass filter and after lowpass

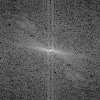
filter in ROI(60,60,100,100), T=40.

1. Result of low-pass filter with ROI

Second, we applied high pass filter with threshold value of 40 on the lena image as below. Figure b shows the gradient amplitude as intensity image for Fourier domain before high-pass filter and after highpass filter in ROI (60,60,100,100), T=40. We can see the result in image c.



1. Original image

1. Fourier domain before high-pass filter and after highpass

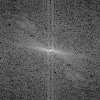
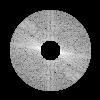
filter in ROI(60,60,100,100), T=40.

1. Result of high-pass filter with ROI

Finally, we applied band pass filter with threshold value of 40 on the lena image as below. Figure b shows the gradient amplitude as intensity image for Fourier domain before high-pass filter and after highpass filter in ROI (60,60,100,100), T=10 to 40. We can see the result in image c.



1. Original image

1. Fourier domain before band-pass filter and after band pass

filter in ROI(60,60,100,100), T=10 to 40.

1. Result of band-pass filter with ROI

**4. Discussion**

The OpenCV DFT can easily transform a normal domain into frequency domain. The low-pass filter, high-pass filter and band-pass filter can be easily implemented in the Fourier domain.

The low-pass filter is like smoothing filter, the high-pass filter is like edge detection. The band-pass filter is like smoothing and edge detection.

**README**

Project 4

Fourier Transform and frequency filtering

\*\*\*\*\*

This software is architectured as follows. This software can work on grad server.

DIPCODE/OpenCV - OpenCV code, input image, output image

\*\*\* INSTALATION \*\*\*

On Linux

Enter the OpenCV directory in terminal and run:

g++ dft\_low.cpp -o dft\_low `pkg-config opencv --cflags --libs`

to compile dft\_low

g++ dft\_high.cpp -o dft\_high `pkg-config opencv --cflags --libs`

to compile dft\_high

g++ dft\_band.cpp -o dft\_band `pkg-config opencv --cflags --libs`

to compile dft\_band

\*\*\* FUNCTIONS \*\*\*

implement the following filters. (Not use the function from OpenCV) functions should operate within ROI (one is enough).

For each filter, you should have three output images:

1. the amplitude of the Fourier domain before filtering (no need to combine with the original image),

2. the amplitude of the Fourier domain after filtering (no need to combine with the original image),

3. the output result of the ROI combines with the original image.

Filters:

Low-pass filter,

High-pass filter,

Band-stop filter,

ROI\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*low or high:

60 60 100 100 40 50 # Parameters for ROI1: x, y, sx, sy, T, NULL

#need a space after 50#

\*\*band:

60 60 100 100 10 40 50 # Parameters for ROI1: x, y, sx, sy, T\_low, T\_high, NULL

#need a space after 50#

\*\*\* Run the program:

./dft\_low

./dft\_high

./dft\_band