9. Image filtering in the spatial and frequency domains

9.1. Introduction

In this laboratory the convolution operator will be presented. This operator is used in the linear image filtering process applied in the spatial domain (in the image plane by directly manipulating the pixels) or in the frequency domain (applying a Fourier transform, filtering and then applying the inverse Fourier transform. Examples of such filters are: low pass filters (for smoothing) and high pass filters (for edge enhancement).

9.2. The convolution process in the spatial domain

The convolution process implies the usage of a convolution mask/kernel H (usually with symmetric shape and size w*w, cu w=2k+1) which is applied on the source image according to (9.2).

$$I_D = H * I_S \tag{9.1}$$

$$I_D(x,y) = \sum_{i=-k}^{k}\sum_{j=-k}^{k} H(i,j)\cdot I_S(x+i, y+j), \quad x = 0..Height-1, \quad y = 0..Width-1 \tag{9.2}$$

This implies the scanning of the source image I_S, pixel by pixel, ***ignoring the first and last k rows and columns*** (Fig. 9.1) and the computation of the intensity value in the current position *(x,y)* of the destination image I_D using (9.2). The convolution mask is positioned spatially with its central element over the current position *(x,y)*.

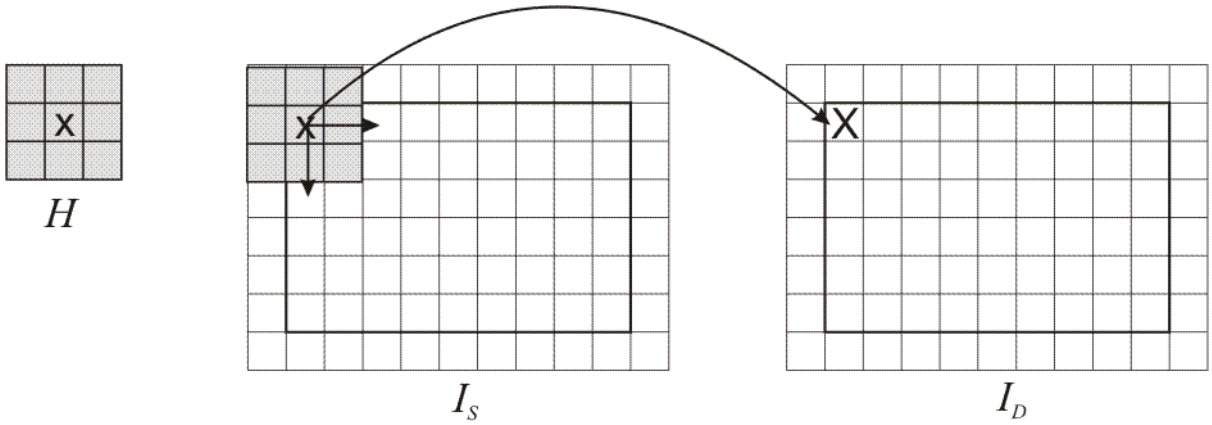


Fig. 9.1 Illustration of the convolution process.

The convolution kernels can have also non-symmetrical shapes (the central/reference element is not positioned in the center of symmetry). Convolution with such kernels applied in a similar way, but such examples will not be presented in the current laboratory.

9.2.1. Low-pass filters

Low-pass filters are used for image smoothing and noise reduction (see the lecture material). Their effect is an averaging of the current pixel with the values of its neighbors, observable as a “blurring” of the output image (they allow to pass only the low frequencies of the image).

All elements of the kernels used for low-pass filtering have positive values. Therefore, a common practice used to scale the result in the intensity domain of the output image is to divide the result of the convolution with the sum of the elements of the kernel:

$$I_D(x,y) = \frac{1}{c} \cdot \sum_{i=-k}^{k} \sum_{j=-k}^{k} H(i,j) \cdot I_S(x+i, y+j) \tag{9.3}$$

Where:

$$c = \sum_{i=-k}^{k} \sum_{j=-k}^{k} H(i,j) \tag{9.4}$$

Examples:
Mean filter (3x3):

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{9.5}$$

Gaussian filter (3x3):

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{9.6}$$

a. **b.** **c.**

Fig. 9.2 a. Original image; b. Result obtained by applying a 3x3 mean filter. c. Result obtained by applying a 5x5 mean filter.

9.2.2. High-pass filters

These filters will highlight regions with step intensity variations, such edges (will allow to pass the high frequencies).

The kernels used for edge detection have the sum of their elements equal to 0:

Laplace filters (edge detection) (3x3):

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{9.7}$$

or

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{9.8}$$

High-pass filters (3x3):

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{9.9}$$

or

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{9.10}$$

a. b. c.

Fig. 9.3 a. The result of applying the Laplace edge detection filter (9.8) on the original image (Fig. 9.2a); b. The result of applying the Laplace edge detection filter (9.8) on the blurred image from Fig. 9.2b (previously filtered with the 3x3 mean filter); c. The result obtained by filtering the original image with the high-pass filter (9.10)

9.3. Image filtering in the frequency domain

The 1D discrete Fourier transform (DFT) of an array of N real or complex numbers is an array of N complex numbers, given by:

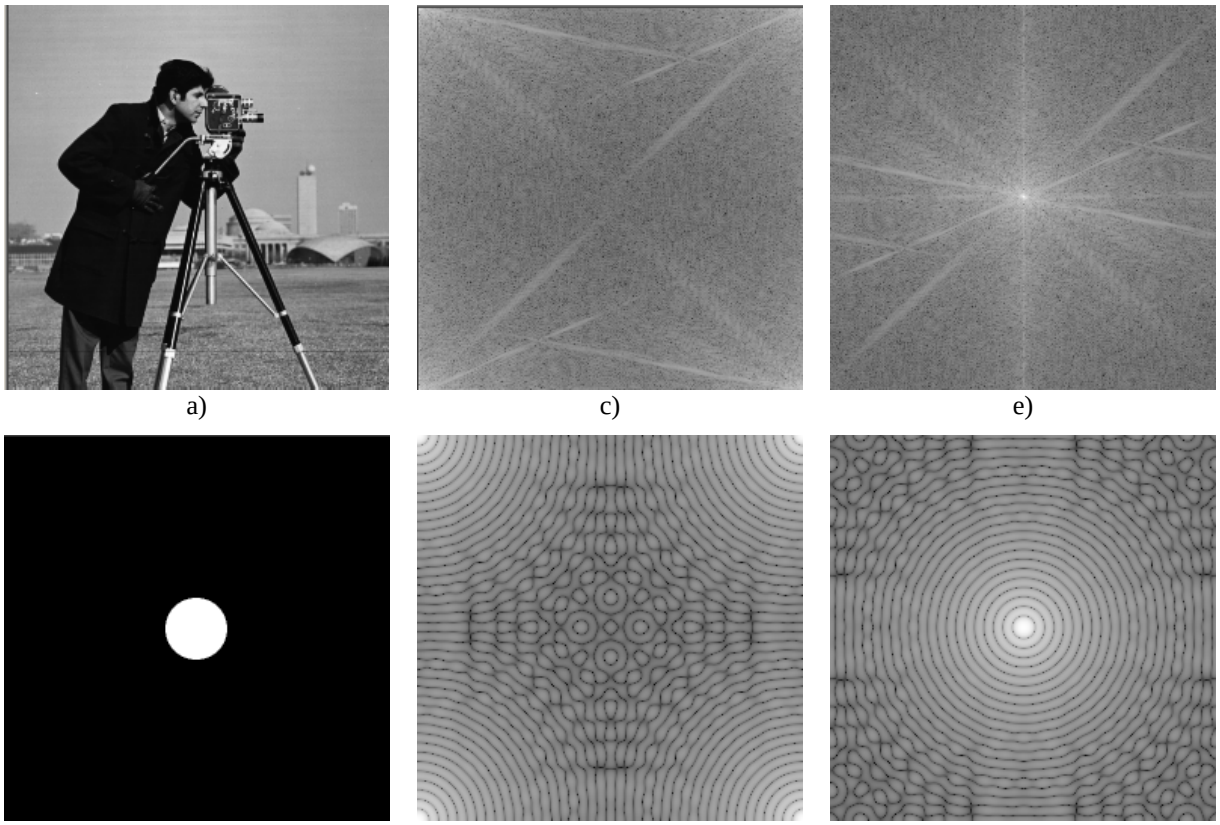
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi jkn}{N}}, \quad k = \overline{0...N-1} \tag{9.11}$$

The inverse discrete Fourier transform (IDFT) is given by:

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} X_k e^{\frac{2\pi jkn}{N}}, \quad n = \overline{0...N-1} \tag{9.12}$$

The 2D DFT is performed by applying the 1D DFT on each row of the input image and then on each column of the previous result. The 2D IDTF is performed by applying the 1D IDFT on each column of the DFT "image" and then on each row of the previous result. The set of complex numbers which are the result of the DFT may also be represented in polar coordinates (magnitude, phase). The set of (real) magnitudes represent the frequency power spectrum of the original array.

The DFT and its inverse are usually performed using the Fast Fourier Transform recursive approach, which reduces the computation time from $O(n^2)$ to $O(n \ln n)$, which represents a significant speed increase, especially in the case of 2D image processing, where a $O(n^2 m^2)$ complexity would be intractable for large images as opposed to the almost linear in number of pixels $O(nm \ln(m+n))$ complexity.



a) c) e)

b) d) f)

Fig. 9.4 a) and b) original images; c) and d) logarithm of magnitude spectra; e) and f) centered logarithm of magnitude spectra

9.3.1. Aliasing

The aliasing phenomenon is a consequence of the Nyquist frequency limit (a sampled signal cannot represent frequencies higher than half the sampling frequency). This means that the higher half of the frequency domain representation is redundant. This fact can also be seen from the identity:

$$X_k = X^*_{N-k} \tag{9.13}$$

(where the asterix denotes complex conjugation) which is true if the input numbers x_k are real. Therefore, the typical 1D Fourier spectrum will contain the low frequency components in both the lower and upper part, with high frequency located symmetrically about the middle. In 2D, the low frequency components will be located near the image corners and the high frequency components in the middle (see Fig. 9.4c, d). This makes the spectrum hard to read and interpret. In order to center the low frequency components spectrum about the middle of the spectrum, one should first perform the transformation on the input data:

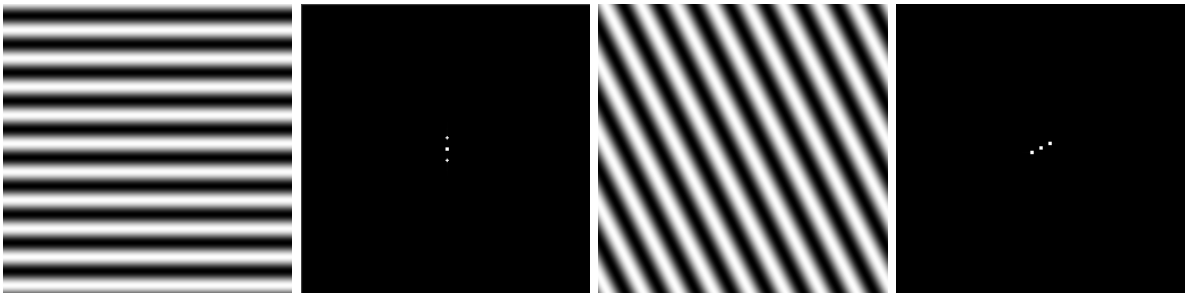
$$x_k \leftarrow (-1)^k x_k \tag{9.14}$$

In 2D the centering transformation becomes:

$$x_{uv} = (-1)^{u+v} x_{uv} \tag{9.15}$$

After applying this centering transform, in 1D the spectrum will contain the low frequency components in the center, and the high frequency components will be located symmetrically toward the left and right ends of the spectrum. In 2D, the low frequency components will be located in the middle of the image, while various high frequency components will be located toward the edges.

The magnitudes located on any line passing through the DFT image center represent the 1D frequency spectrum components of the original image, along the direction of the line. Every such line is therefore symmetrical about its middle (the image center).



a) b) c) d)

Fig. 9.5 Fourier transforms of sine image waves a) and c). The center point in b) and d) represent the DC component, the other two symmetrical points are due to the sine wave frequency.

9.3.2. Ideal low-pass and high-pass filters in frequency domain

The convolution in spatial domain is equivalent to scalar multiplication in frequency domain. Therefore, especially for large convolution kernels, it is computationally convenient to perform convolution in the frequency domain.

The algorithm for filtering in the frequency domain is:

a) Perform the image centering transform on the original image (9.15)
b) Perform the DFT transform
c) Alter the Fourier coefficients according to the required filtering
d) Perform the IDFT transform
e) Perform the image centering transform again (this undoes the first centering transform).

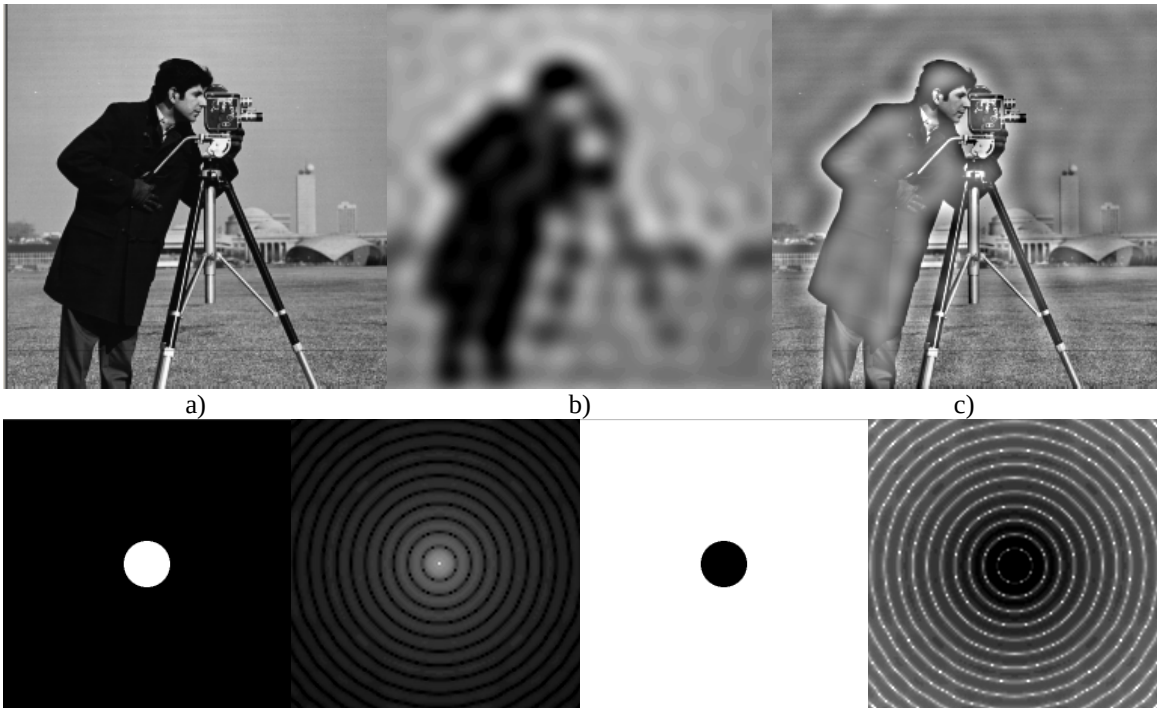
An ideal low pass filter will alter all the Fourier coefficients that are further away from the image center *(W/2, H/2)* than a given distance *R*, by turning them to zero (*W* is the image width and *H* is the image height):

$$X'_{uv} = \begin{cases} X_{uv}\ , & \left(\frac{H}{2}-u\right)^2 + \left(\frac{W}{2}-v\right)^2 \le R^2 \\ 0\ , & \left(\frac{H}{2}-u\right)^2 + \left(\frac{W}{2}-v\right)^2 > R^2 \end{cases} \tag{9.16}$$

An ideal high-pass filter will alter all Fourier coefficients that are at a distance less than *R* from the image center *(W/2, H/2)*, by turning them to 0.

$$X'_{uv} = \begin{cases} X_{uv}\ , & \left(\frac{H}{2}-u\right)^2+\left(\frac{W}{2}-v\right)^2 > R^2 \\ 0\ , & \left(\frac{H}{2}-u\right)^2+\left(\frac{W}{2}-v\right)^2 \le R^2 \end{cases} \tag{9.17}$$

The results of filtering with ideal low- and high-pass filtering are presented in Fig. 9.6 b) and c). Unfortunately, the corresponding spatial filters Fig. 9.6 e) and d) are not FIR (they have an infinite support) and keep oscillating away from their centers. Because of this, the low-pass and high-pass filtered images have a disturbing ringing wavy aspect. In order to correct this, the cutoff in the frequency domain must be smoother, as presented in the next section.



a) b) c)

d) e) f) g)

Fig. 9.6 a) original image; b) result of ideal low-pass filtering; c) result of ideal high-pass filtering; d) ideal low-pass filter in the frequency domain; e) corresponding ideal low-pass filter in the spatial domain; f) ideal high-pass filter in the frequency domain; g) corresponding ideal high-pass filter in the spatial domain

9.3.3. Gaussian low-pass and high-pass filtering in the frequency domain

In the case of Gaussian filtering, the frequency coefficients are not cut abruptly, but smoother cutoff process is used instead. This also takes advantage of the fact that the DFT of a Gaussian function is also a Gaussian function (Fig. 9.7d-g).

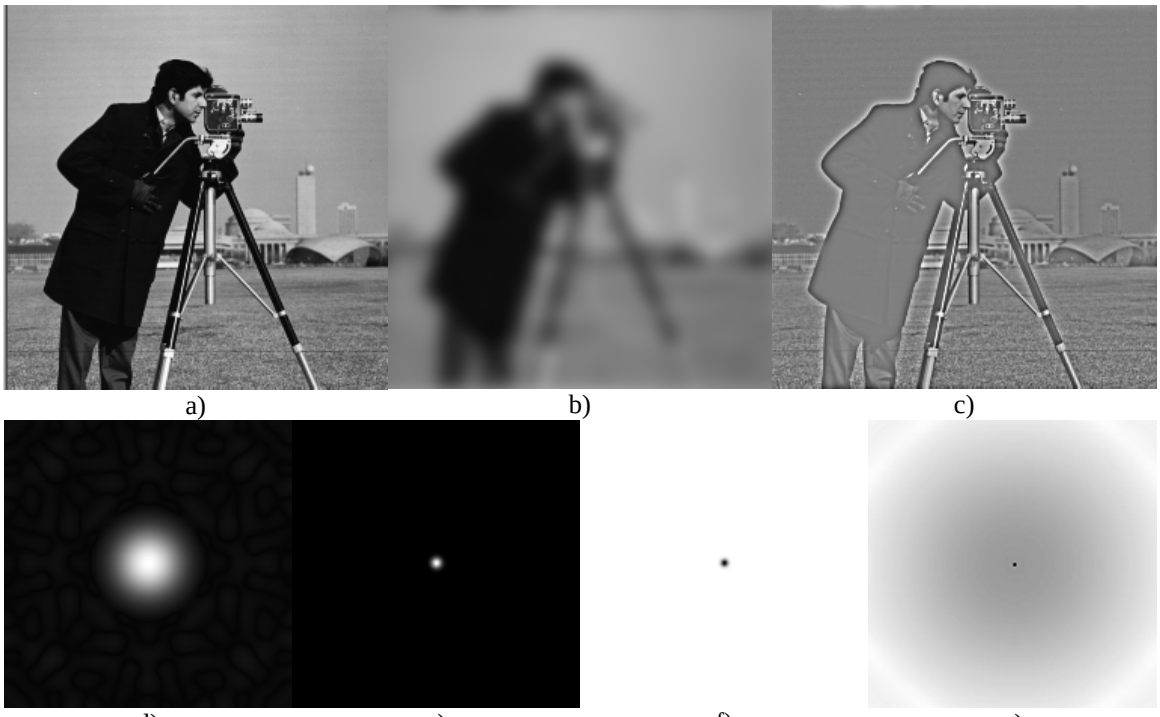
The Gaussian low-pass filter attenuates frequency components that are further away from the image center *(W/2, H/2)*. $A \sim \frac{1}{\sigma}$ where σ is the standard deviation of the equivalent spatial domain Gaussian filter.

$$X'_{uv} = X_{uv} e^{-\frac{\left(\frac{H}{2}-u\right)^2+\left(\frac{W}{2}-v\right)^2}{A^2}} \tag{9.18}$$

The Gaussian high-pass filter attenuates frequency components that are near to the image center *(W/2, H/2)*:

$$X'_{uv} = X_{uv}\left(1-e^{-\frac{\left(\frac{H}{2}-u\right)^2+\left(\frac{W}{2}-v\right)^2}{A^2}}\right) \tag{9.19}$$

Fig. 9.7 shows the results of Gaussian filter. Notice that the ringing (wavy) effect visible in Fig. 9.6 disappeared.



a) b) c)

d) e) f) g)

Fig. 9.7 a) original image; b) result of Gaussian low-pass filtering; c) result of Gaussian high-pass Filtering; d) Gaussian low-pass filter in the frequency domain; e) corresponding Gaussian low-pass filter in the spatial domain; f) Gaussian high-pass filter in the frequency domain; g) corresponding Gaussian high-pass filter in the spatial domain

9.4. Implementation details

9.4.1. Spatial domain filters

Low-pass filters will always have positive coefficients, and therefore, the resulting filtered image will have positive values. You must ensure that the resulting image fits in the desired range (0-255 in our case). In order to ensure this, you must ensure that the coefficients of a low-pass filter sum to 1. If you are using integer operations pay attention to the order of operations! Usually, the division should be the last operation performed in order to minimize the rounding errors!

High-pass filters will have both positive and negative coefficients. You must ensure that the final result is an integer between 0 and 255! There are three possibilities to ensure that the resulting image fits the destination range. The first one is to compute:

$$S_+ = \sum_{F_k>0} F_k, \quad S_- = \sum_{F_k<0} -F_k,$$
$$S = \frac{1}{2\max\{S_+, S_-\}} \tag{9.20}$$
$$I_D(u,v) = S(F * I_S)(u,v) + \left\lfloor \frac{L}{2} \right\rfloor$$

In the formula above S_+ represents the sum of positive filter coefficients and S_- the sum of negative filter coefficients magnitudes. This result of applying the high-pass filter always lies in the interval $[-LS_-, LS_+]$ where L is the maximum image gray level (255). The result of this transform will place scale the result to [*-L/2, L/2*] and then move the 0 level to *L/2*.

Another approach is to perform all operations using signed integers determine the minimum and maximum and then linearly transform the resulting values according to:

$$D = \frac{L(S-\min)}{\max-\min} \tag{9.21}$$

The third approach is to compute the magnitude of the result and saturate everything that exceeds the maximum level *L*.

9.4.2. Frequency domain filters

A library and a header file is supplied for performing the fast Fourier transform. The library is called “dibfft.lib” and the header file is called “dibfft.h”. In order to use the library file you should first copy the “dibfft.lib” and “dibfft.h” files to the “Diblook” folder.

Then right click on the “Diblook” project entry in the workspace window, select “Add > Existing Item…”. It will automatically open the window “Add Existing Item - DibLook”. You should select and add the file “dibfft.h” to the project.

Then the library “dibfft.lib” should be included in the project linker section. Perform right click on the “DibLook” project in the workspace window and select “Properties”. It will automatically open the window “DibLook Property Pages”. In the “Configuration” section choose “All Configurations” and then add the library “dibfft.lib” in the “Linker” section (see Fig. 9.8!).

Finally you should add the header include “dibfft.h” in the “#include” section in the “dibview.cpp” file.

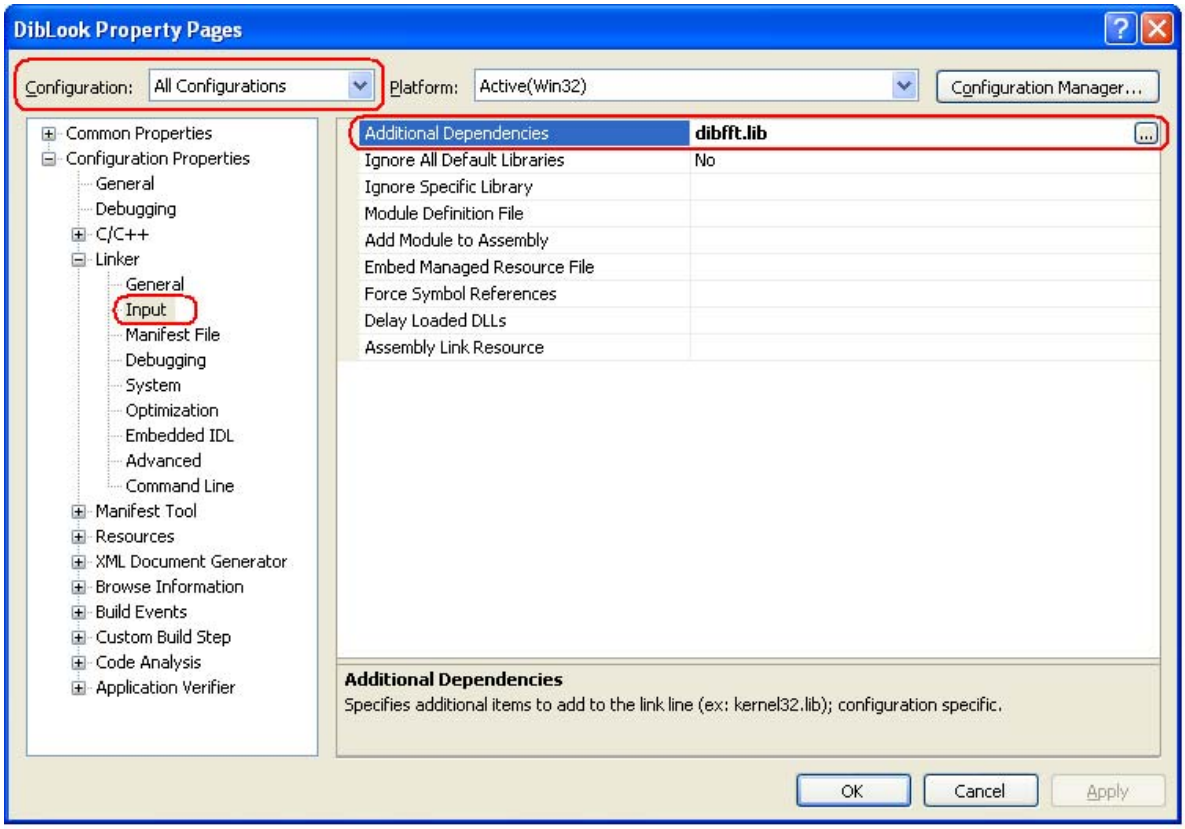


Fig. 9.8 Adding a library file to the project

The library provides the following functions:

```
/*The first two paramers are the image width & height
These functions only work correctly for width, height powers of 2   and >=4.
Parameters 3 and 4 are the input real and imaginary parts arrays
of width*height values. Acceptable value types are unsigned char (BYTE),
float and double. The imaginary part is optional (T*)0 can be provided  instead,
and the input will be assumed as consisting of values of 0.
Parameters  5 and 6 are the output real and imaginary parts.Acceptable value types
are unsigned char (BYTE), float and double. The imaginary part is optional (T*)0
can be provided instead. The imaginary part of the output will be discarded in this
case.*/

/*perform FFT on image rows*/
template<class T> void fftrows(int width, int height, const T *ix, const T *iy,
double *ox, double *oy);

/*perform IFFT on image rows*/
template<class T> void ifftrows(int width, int height, const double *ix, const
double *iy, T *ox, T *oy);

/*perform FFT on image cols*/
template<class T> void fftcols(int width, int height, const T *ix, const T *iy,
double *ox, double *oy);

/*perform IFFT on image cols*/
template<class T> void ifftcols(int width, int height, const double *ix, const
double *iy, T *ox, T *oy);

/*perform FFT on image*/
template<class T> void fftimage(int width, int height, const T *inpx, const T*
inpy, double *ox, double *oy);

/*perform IFFT on image*/
template<class T> void ifftimage(int width, int height, const double *ix, const
double *iy, T *outpx, T *outpy);
```

The functions provided are template based and work for BYTE, float and double inputs/outputs. Imaginary parts are optional for both input and output. Use NULL pointers to specify missing inputs/outputs.

The following code gives an example of FFT followed by an IFFT. The original image should be recovered:

```
BEGIN_PROCESSING();
double *real= new double[dwWidth*dwHeight];
double *imag= new double[dwWidth*dwHeight];
fftimage(dwWidth, dwHeight, lpSrc, (BYTE*)0, real, imag);
ifftimage(dwWidth, dwHeight, real, imag, lpDst, (BYTE*)0);
END_PROCESSING("FFT");
```

A few important aspects of working with frequency domain values:

1. Always use for FFT only grayscale images having both width and height powers of two (example: image “cameraman.bmp” having width=height=256=2^8 pixels)

2. Always perform the centering transform (9.15) before performing the FFT and after performing the IFFT:

```
for(int i=0;i< dwHeight;i++) {
    for(int j=0;j< dwWidth;j++) {
         D[i*w+j] = ((i+j)&1)?-S[i*w+j]:S[i*w+j];
```

3. The DC (0,0) Fourier coefficient highly dominates the other ones. When displaying the magnitude of Fourier coefficients it is better to use the logarithm of the module+1! You should determine the maximum value of the logarithm and scale the remaining values to fit the range

4. The Fourier transform of an image is a complex array of floating point values! Store both real and imaginary values as floating points! When converting back to the spatial domain the imaginary values may be discarded (for usual filters they should be 0 anyway).

9.5. Practical work

1. Implement the convolutions with the kernels from equations (9.5) .... (9.10)
2. Implement a customized convolution operator of size 3x3 using values specified by the user in a dialog box. The scaling coefficient should be computed automatically as either the reciprocal of filter coefficient sum for low pass filters or according to equation (9.20) for high-pass filters.
3. Import the dibfft library into Diblook (see section 9.4.2). Add a processing function that performs the FFT transform of an input image and the transforms the result back to the spatial domain using IFFT. Check if the destination is the same as the source!
4. Add a processing function that computes and displays the logarithm of the magnitude of the Fourier transform of an input image.
5. Add processing functions that perform low- and high-pass filtering in the frequency domain using the ideal and Gaussian filters from equations (9.16)...(9.19).
6. **Save your work. Use the same application in the next laboratories. At the end of the image processing laboratory you should present your own application with the implemented algorithms.**

References

[1]. Umbaugh Scot E, *Computer Vision and Image Processing*, Prentice Hall, NJ, 1998, ISBN 0-13-264599-8
[2] R.C.Gonzales, R.E.Woods, *Digital Image Processing, 2-nd Edition*, Prentice Hall, 2002