WS 2019/20	Exercises Digital Image Processing I	Solution Exercise No. 10

Discrete Fourier Transform

Targets of Exercise:

General

- Using complex-images
- Discrete Fourier Transform
- Translation property of Fourier transform

Solution for task 10.1a

Task: *for the frequency domain filter, define the following filter transfer functions:*

For all filters: $D(u,v) = \sqrt{\left[\left(u - \frac{M}{2}\right)^2 + \left(v - \frac{N}{2}\right)^2\right]}$ (distance of point (u,v) to the center) and the *Cut-Off frequency* $D_0 > 0$.

Ideal lowpass filter

$$H(u,v) = \begin{cases} 1, & \text{if } D(u,v) \leq D_0 \\ 0, & \text{if } D(u,v) > D_0 \end{cases} \tag{1}$$

Butterworth lowpass filter

$$H(u,v) = \frac{1}{1 + \left[\frac{D(u,v)}{D_0}\right]^{2n}} \tag{2}$$

Gauss lowpass filter

$$H(u,v) = e^{-\frac{D^2(u,v)}{2{D_0}^2}} \tag{3}$$

The high pass filters are very similar to the low-pass filters :

Ideal high pass filters

$$H(u,v) = \begin{cases} 0, & \text{if } D(u,v) \leq D_0 \\ 1, & \text{if } D(u,v) > D_0 \end{cases} \tag{4}$$

Butterworth high pass filters

$$H(u,v) = \frac{1}{1 + \left[\frac{D_0}{D(u,v)}\right]^{2n}} \tag{5}$$

Gauss high pass filters

$$H(u,v) = 1 - e^{-\frac{D^2(u,v)}{2{D_0}^2}} \tag{6}$$

Solution for task 10.1b

Task: *Implement one or more functions such that the matrices generates transfer functions from exercise part (a)!*

A filter in the frequency domain is stored as an object of class `ComplexImage` .There are low-pass and high pass filter implemented in exercise 10.1a, using D_0 As a parameter for the cut-off frequency, n for the order ,and $\sigma = D_0$ (see lecture notes).

Definition of interfaces:

```
// Write the transfer function of a filter of type <filter> in the
// given Matrix, with Cut-Off frequency <d0> and order <n>.
void buildFilterTransferFunction (ComplexImage& input, FilterType filter, int d0, int n)
```

Implementation:

```
//
//
//
enum FilterType {ILPF = 1, IHPF = 2, BLPF = 3, BHPF = 4, GLPF = 5, GHPF = 6};

//
//
//
void buildFilterTransferFunction (ComplexImage& input, FilterType filter, int d0, int n)
{
    int      width  = input.getWidth();
    int      height = input.getHeight();
    Complex* data   = input.getData();

    int      m2     = width / 2;
    int      n2     = height / 2;

    for (int v=0; v<height; v++)
    {
        for (int u=0; u<width; u++)
        {
            float Duv = sqrt((double)((u-m2)*(u-m2) + (v-n2)*(v-n2)));

            if (filter == ILPF)
            {
                if (Duv <= d0)  data[v*width+u].re = 1;
                else            data[v*width+u].re = 0;
            }

            if (filter == IHPF)
            {
                if (Duv <= d0)  data[v*width+u].re = 0;
                else            data[v*width+u].re = 1;
            }

            if (filter == BLPF) data[v*width+u].re = 1.0 / (1.0 + powf((Duv/d0), 2.0*n));
            if (filter == BHPF) data[v*width+u].re = 1.0 / (1.0 + powf((d0/Duv), 2.0*n));


            if (filter == GLPF) data[v*width+u].re = expf(-(Duv*Duv)/(2.0 * d0 * d0));
            if (filter == GHPF) data[v*width+u].re = 1.0 - expf(-(Duv*Duv)/(2.0 * d0 * d0));

            data[v*width+u].im = 0;
        }
    }
}

// ------------------------------------------------------------------------

//
```

Solution for task 10.1c

Task: *Write a function that transforms the filter transfer function from exercise (b) into the real space function!*

The representation of a frequency filter in the spatial domain is done by the algorithm from the lecture notes. Before Fourier transformation can be made, the input filter must be centered, then calculates the inverse Fourier transform. Finally, once again takes place-centering, so that the image is properly displayed.
Definition of interfaces:

```
// Transform the given filter <input_frequency> from the
// frequency domain to the spatial domain. The function return <output_spatial>
// to be represented in the spatial domain.

void spatialRepresentation (ComplexImage& input_frequency,
                            ComplexImage& output_spatial);
```

Implementation:

```
//
//
//
void spatialRepresentation (ComplexImage& input_frequency, ComplexImage& output_spatial)
{
    ComplexImage tmp(input_frequency);

    fourier_center(tmp);

    inverse_fourier_transform(tmp, output_spatial);

    fourier_center(output_spatial);
}

// ---------------------------------------------------------------------------

//
//
//
void clip (GrayImage& input)
{
    float* data = input.getData();
    int    size = input.getSize();

    for (int i=0; i<size; i++)
    {
        if (data[i] > 255) data[i] = 255;
        if (data[i] < 0)   data[i] = 0;
    }
}

// ---------------------------------------------------------------------------

//
//
//
void scale (GrayImage& input)
{
    int    size = input.getSize();
    float* data = input.getData();

    float  mini = min(input);
    float  maxi = max(input);

    for (int i=0; i<size; i++)
    {
        data[i] = 255.0 * (data[i] - mini) / (maxi - mini);
    }
}
```

```
// ----------------------------------------------------------------------------

//
//
//
enum PreviewType {CLIP, SCALE};
```

Solution for task 10.1d

Task: *Write down the function that represents the filter mask of a ComplexImage object into a GrayImage objects and also show them on the screen.*

To view the complex-images or filter in the spatial domain, the `ComplexImage`object must be decomposed in real and imaginary parts. The display appears as a gray-scale image, where the image must be scaled or clipped to exploit the full spectrum and to eliminate extreme values, if necessary.

Definition of interfaces:

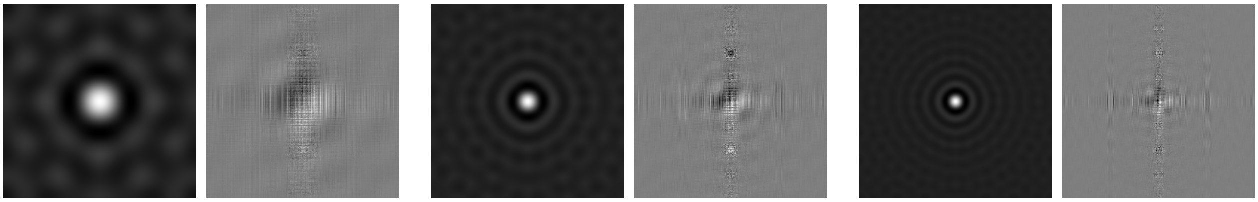
```
// Decompose the complex-image <image> in an image of type GrayImage
// and show it. <preview> will define if the image will be scaled
// or cliped.
void viewComplexImage (ComplexImage& input, PreviewType preview)
```

Implementation:

```
//
//
//
void viewComplexImage (ComplexImage& input, PreviewType preview)
{
    int width  = input.getWidth();
    int height = input.getHeight();

    GrayImage real_img(width, height);
    GrayImage imag_img(width, height);

    real_part(input, real_img);
    imag_part(input, imag_img);


    if (preview == SCALE) scale(real_img);   // Scale or
    if (preview == CLIP)  clip(real_img);    // Clip?

    real_img.show();
    real_img.save();

    if (preview == SCALE) scale(imag_img);   // Scale or
    if (preview == CLIP)  clip(imag_img);    // Clip?

    imag_img.show();
    imag_img.save();
}

// --------------------------------------------------------------------------

//
//
//
void viewComplexImageMagnitude (ComplexImage& input, PreviewType preview)
{
    int width  = input.getWidth();
    int height = input.getHeight();

    GrayImage real_img(width, height);
    GrayImage imag_img(width, height);
    GrayImage magnitude_img(width, height);

    real_part(input, real_img);
    imag_part(input, imag_img);
    fourier_spectrum(input, magnitude_img);

    if (preview == SCALE) scale(real_img);
    if (preview == CLIP)  clip(real_img);

    real_img.show();
    real_img.save();

    if (preview == SCALE) scale(imag_img);
    if (preview == CLIP)  clip(imag_img);

    imag_img.show();
    imag_img.save();

    float maxi = max(magnitude_img);

    magnitude_img.mul(255.0 / maxi);

    magnitude_img.show();
    magnitude_img.save();
}
```

Solution for task 10.1e

Task: *Generate responses for different values of D_0, σ and n (depending on the type of filter) and show them on the screen. Compare the results for: $D_0 = 5$, $D_0 = 10$, $D_0 = 15$, $n = 2$, $n = 10$, $\sigma = 2$, $\sigma = 5$ und $\sigma = 10$.*



(a) (b) (c)

Abbildung 1: Real and imaginary part of the spatial presentation of an ideal low pass filter, with (a) $D_0 = 5$, (b) $D_0 = 10$ and (c) $D_0 = 15$

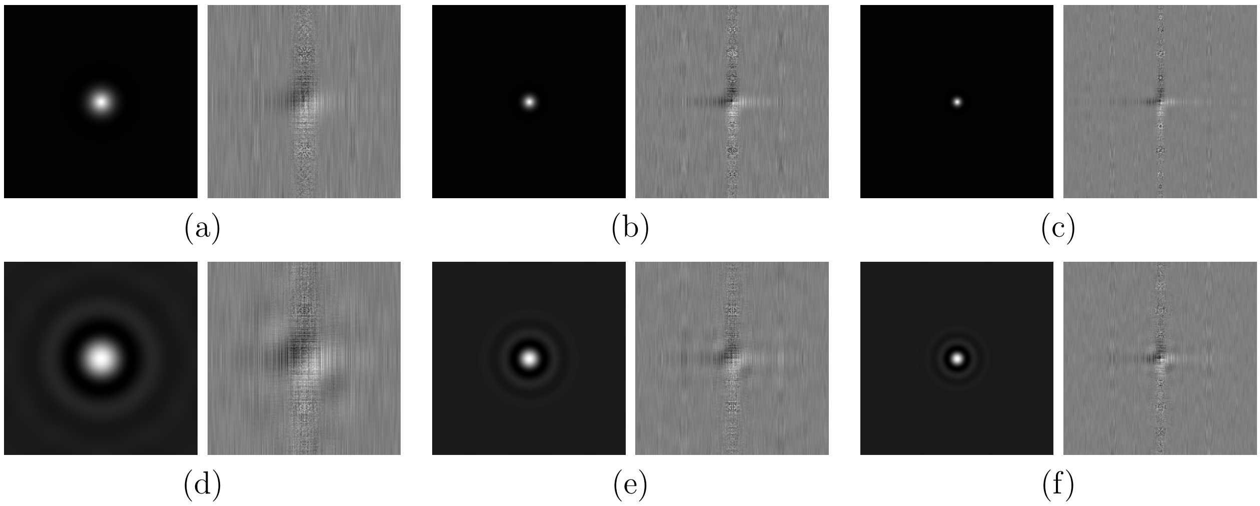


Abbildung 2: Real and imaginary part of the spatial presentation of an Butterworth low pass filter, with (a) $D_0 = 5, n = 2$, (b) $D_0 = 10, n = 2$, (c) $D_0 = 15, n = 2$, (d) $D_0 = 5, n = 10$, (e) $D_0 = 10, n = 10$ und (f) $D_0 = 15, n = 10$

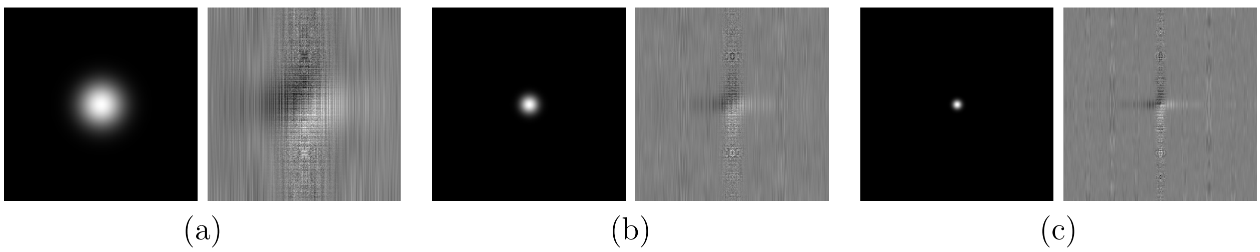


Abbildung 3: Real and imaginary part of the spatial presentation of an Gauss low pass filter, with (a) $D_0 = 5, \sigma = 2$, (b) $D_0 = 5, \sigma = 5$ and (c) $D_0 = 5, \sigma = 10$. Other values for D_0 hardly deliver visible differences.

Main Program:

```
int main (int argc, char** argv)
{
    /* ********************************** */
    /* Exercise 9                         */
    /* ********************************** */

    int width  = 128;
    int height = 128;
    ComplexImage transferFunction(width, height);
    ComplexImage spatialFilterMask(width, height);

    int input;
    cout << "Filtertyp (1 (ILPF), 2 (IHPF), 3 (BLPF), 4 (BHPF), 5 (GLPF), 6 (GHPF)): ";
    cin >> input;

    // Filter Typ setzen
    FilterType filtertype = static_cast<FilterType>(input);

    // D0 und n abfragen
    int   d0    = 1;
    int   n     = 1;

    cout << "D0: ";
    cin >> d0;

    if (filtertype == BLPF || filtertype == BHPF)
    {
        cout << "n: ";
        cin >> n;
    }

    buildFilterTransferFunction(transferFunction, filtertype, d0, n);

    spatialRepresentation(transferFunction, spatialFilterMask);

    viewComplexImage(spatialFilterMask, SCALE);

    cout << "FINISHED.\n";
    return 0;
}
```