WS 2019/20	Exercises Digital Image Processing	Solution Exercise No. 8

Filtering in the spatial domain (Continued)

Targets of Exercise:

Targets of Exercise:

General

- Edge filter
- Internal and visual representation
- Creating sharp images
- Comparing the results of different edge filters

Solution for task 8.1a

Task: *Which type of edge filter is known to you?*

The most common filters are *Laplace*, *Sobel* and *Roberts* filter. Table 1 shows different filter masks.

−1	−1	−1
−1	8	−1
−1	−1	−1

(a)

−1	0
0	1

(b)

0	−1
1	0

(c)

−1	−2	−1
0	0	0
1	2	1

(d)

−1	0	1
−2	0	2
−1	0	1

(e)

Table 1: (a) Laplace filter mask, (b) and (c) Roberts filter masks, (d) and (e) Sobel filter masks

Solution for task 8.1b

Task: *What is the type of pattern for the filter response of an edge filter? Describe different methods to generate an edge diagram.*

The response of the edge filter is a matrix in the size of the original image with positive and negative values outside the normal image domain (*spectrum*). This filter response is generated to filter an image, which has been filtered with the mask (*Mask*). An important aspect of filter responses should be taken into consideration is that there is no change at the matrix values around zero. On the other hand, the higher (for positive values) and the lower (for negative values) of the matrix values are, the greater the color difference and an edge is found. To create an edge image, there are different procedures:

1) The first method scales the range of the filter response to be [0,255]. The resulting images produced are typically gray-scale images whose edges as bright.

2) In the second method, the amount of all values will be considered, so that there are only positive values in the matrix. When creating the visual representation, the matrix values are transmitted as image color values, where all values greater than 255 set to be 255 (*Clipping*). For such edge image, 0 (= black) represents homogeneous area (no edges), while 255 (= white) represents large color difference (edges).

   Note: values are clipped with no absolute value function (eliminating the negative edges).

Solution for task 8.1c

Task: *Which possibilites exist to generate the rectified edge image? Why edge image can not be visualized?*

To create a rectified edge image (sharp image), the filter response is added to the image. The result can exceed the spectrum of normal gray image. Therefore, it must be adjusted after the addition by one of the following procedures:

1) As with the edge images, it is possible to scale the result to the range [0,255]. The result is also a gray-scale image in which the edges are more emphasized, but not as strong as in an edge image.

2) The second procedure is done by clipping without the absolute value function. The formal of the clipping calculation is as follows:

$$f(x,y) = \begin{cases} 0, & \text{if } f(x,y) < 0 \\ 255, & \text{if } f(x,y) > 255 \\ f(x,y), & \text{else} \end{cases}$$

The edge image is merely a visual representation of the filter response. The filter response includes information outside the normal image range which can not be display. Generating the edge image by one of the discussed procedure will not turn the unviewable information to be visible through conversions. Therefore, these methods will lose such information and make a further processing is impossible.

Solution for task 8.1d

Task: *Implement the functions discussed in tasks b) und c). Generate both the edge image and rectified edge image.*

The required functions are the scaling of image values, the calculation of the absolute values, the clipping of image vlaues. The function `scale(GrayImage& input)` has been implemented already in the task (5). It has been adapted so that only one image is passed as a parameter, which is then modified. The `add(GrayImage& image)` funcion make addition process in the rectified edge image method.

Definition of interfaces::

```
// Scaling of gray values of the input image.
void scale (GrayImage& input)

// Calculation of the absolute values for all values of the input image.
void abs (GrayImage& input)

// Clipping all values under 0 and above 255.
void clip (GrayImage& input)
```

Implementation:

```
//
//
//
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);
    }
}
```

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

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

```
//
//
//
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;
    }
}
```

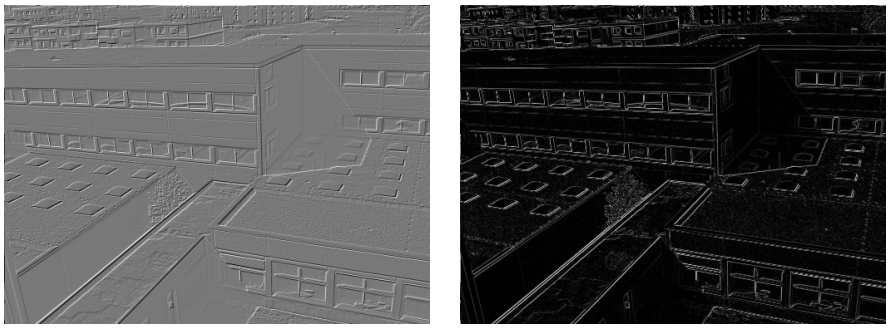
The following call computes the filter response of the horizontal Robert filter and presents the scaled edge image. In order to show the absolute values of an edge image, `abs (...)` function are used.

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

```
/* Exercise 07                                */
/* ********************************** */

cout << "Folgende Funktionen waehlen:\n";
cout << "1 - Laplace Filter erzeugen\n";
cout << "2 - vert. Sobel Filter erzeugen\n";
cout << "3 - hori. Sobel Filter erzeugen\n";
cout << "4 - vert. Roberts Filter erzeugen\n";
cout << "5 - hori. Roberts Filter erzeugen\n";
cout << "-------------------------------------\n";
cout << "6 - Filter auf Bild anwenden und Filterbild skalieren\n";
cout << "7 - Filter auf Bild anwenden und fuer Filterbild absolute Werte verwenden\n";
cout << "8 - Filter auf Bild anwenden, Filterbild auf Bild addieren und skalieren\n";
cout << "9 - Filter auf Bild anwenden, Filterbild auf Bild addieren und clippen\n";

int choice;
cin >> choice;

string filename;
GrayImage image;

if (choice > 5)
{
    cout << "Image file name : ";
    cin >> filename;
    image.load(filename);
}

string filtername;
cout << "Filtername: ";
cin >> filtername;

switch (choice)
{
    case 1:
    {
        GrayImage mask = GrayImage(3,3);
        makeFilter(mask, LAPLACE);
        saveFilterMask(mask, filtername);
        break;
    }
    case 2:
    {
        GrayImage mask = GrayImage(3,3);
        makeFilter(mask, SOBEL_VERT);
        saveFilterMask(mask, filtername);
        break;
    }
    case 3:
    {
        GrayImage mask = GrayImage(3,3);
        makeFilter(mask, SOBEL_HORI);
        saveFilterMask(mask, filtername);
        break;
    }
    case 4:
    {
        GrayImage mask = GrayImage(2,2);
        makeFilter(mask, ROBERTS_VERT);
        saveFilterMask(mask, filtername);
        break;
    }
    case 5:
    {
        GrayImage mask = GrayImage(2,2);
        makeFilter(mask, ROBERTS_HORI);
        saveFilterMask(mask, filtername);
        break;
    }
    case 6:
    {
        GrayImage mask;
        loadFilterMask(filtername, mask);

        GrayImage result = GrayImage(image.getWidth(), image.getHeight());
        filter(image, result, mask);
        scale(result);
```

```
            result.show();
            result.save();
            break;
        }
        case 7:
        {
            GrayImage mask;
            loadFilterMask(filtername, mask);

            GrayImage result = GrayImage(image.getWidth(), image.getHeight());
            filter(image, result, mask);
            abs(result);

            result.show();
            result.save();
            break;
        }
        case 8:
        {
            GrayImage mask;
            loadFilterMask(filtername, mask);

            GrayImage result = GrayImage(image.getWidth(), image.getHeight());
            filter(image, result, mask);
            image.add(result);
            scale(image);

            image.show();
            image.save();
            break;
        }
        case 9:
        {
            GrayImage mask;
            loadFilterMask(filtername, mask);

            GrayImage result = GrayImage(image.getWidth(), image.getHeight());
            filter(image, result, mask);
            image.add(result);
            clip(image);

            image.show();
            image.save();
            break;
        }
    }
    cout << "FINISHED.\n";
    return 0;
}
```

Figure 1 shows the two filter responses.



(a) (b)

Figure 1: Rsults of using horizontal Robert filter, (a) using range scaled and (b) absolute values.

The following Figure shows the results of rectified edge image using the horizontal Robert filter. Here, `scale (...)` may be replaced by `clip (...)`.

```
// Bildinhalt kopieren, damit ist die Randbehandlung erledigt
GrayImage result = GrayImage(input.getWidth(), input.getHeight());

switch (choice)
{
    case 1:
        cout << "Selectionsort gestartet - ";
        filter (input, result, xsize, ysize, SELECTIONSORT);
```

Figure 2 shows the two versions of the rectified edge image.

(a) (b)

Figure 2: Sharpning images using horizontal Robert filter, (a) by scalling and (b) by clipping.

Solution for task 8.1e

Task: *Apply the different edge filters on different images and interpret the results.*

Depending on the filter, vertical, horizontal or diagonal structures will be emphasized. It is easy to recognize especially with the different versions of *uni.bmp* . The following figure shows the results (The images were rotated, filtered, and again turned back to compare them):

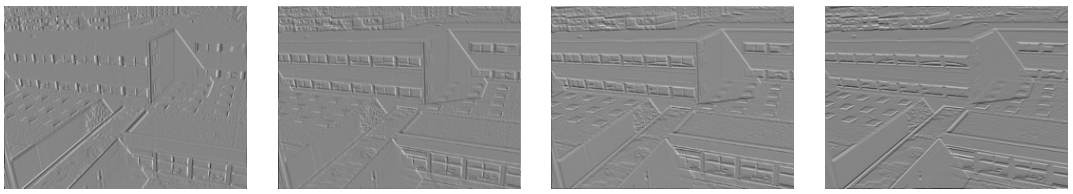
-1	0	1
-2	0	2
-1	0	1

(a)

0	-1
1	0

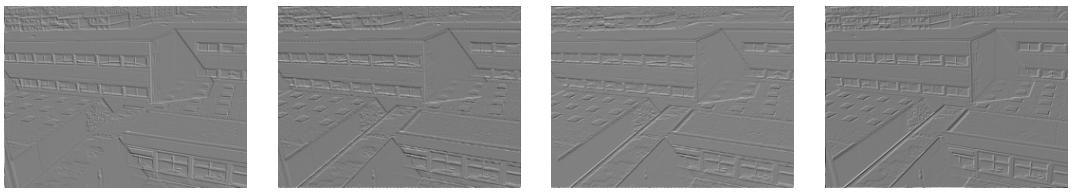
(b)

Table 2: Image filters, (a) Sobel and (b) Roberts 3, 4 and 5



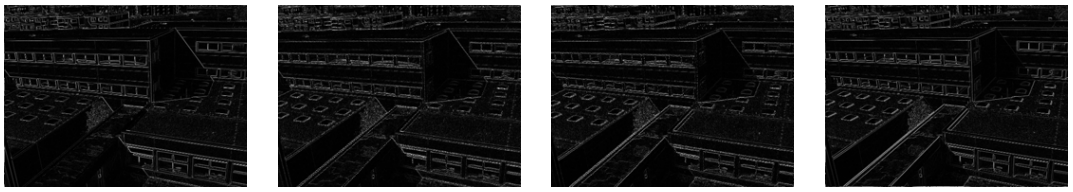
(a) (b) (c) (d)

Figure 3: Result of filtering with Sobel filter (scaled). (a) Filtering with normal image, (b) Image rotated 30°, (c) Image rotated 60°, (d) Image rotated 90°.



(a) (b) (c) (d)

Figure 4: Result of filtering with Roberts filter (scaled). (a) Filtering with normal image, (b) Image rotated 30°, (c) Image rotated 60°, (d) Image rotated 90°.



(a) (b) (c) (d)

Figure 5: Result of filtering with Roberts filter (clipped). (a) Filtering with normal image, (b) Image rotated 30°, (c) Image rotated 60°, (d) Image rotated 90°.