Vietnam National University Ho Chi Minh City
Ho Chi Minh City University of Technology
Faculty of Mechanical Engineering
Mechatronics Department

Computer Vision

Course ID: ME4201
Semester 232 – Class P01

Assignment 1

Group 4

No.	Student name	Student ID
1	Nguyễn Đức Đạt	2111009
2	Nguyễn Phước Đạt	2111014
3	Nguyễn Trọng Phúc	2152883
4	Đinh Ngọc Quỳnh Anh	2151002

List of Exercises

Exercise 1.

(Note: The websites in this exercise may take a while to load all the necessary information.)

The field of view (FOV) of a camera is a rectangle of shape $m \times n$. Therefore, the minimum FOV must be a rectangle inscribed by the arcs of the working space. So, the value of FOV is

$$\text{FOV} = (\text{Arc diameter} + \text{Rectangle length}) \times (\text{Arc diameter}) = 3000 \times 1000 \text{ mm}$$

Enter parameters such as FOV, smallest feature (1 mm), moving object... at the website Basler http://tinyurl.com/2p8pv8m6. From there, the website outputs 8 cameras according to the given requirements (visit http://tinyurl.com/4x7xadjk to see the results). The results are divided into 3 groups as shown below:

Intitial FOV (mm)	3000×1000		
Model	Basler boost boA8100-16cc/16cm	Basler boost boA6500-36cc/36cm	Basler boost boA9344-30cc/30cm/70cc/70cm
Sensor size (mm)	26.2×17.4	21×15.8	29.9×22.4
Sensor diagonal (mm)	31.45	26.28	37.4
Camera resolution (px)	8192×5460	6580×4935	9344×7000
Select FOV width (mm)	3000		
New FOV height (mm)	$3000 \cdot \frac{5460}{8192} = 2000$	$3000 \cdot \frac{4935}{6580} = 2250$	$3000 \cdot \frac{7000}{9344} = 2248$
Camera FOV (mm)	3000×2000	3000×2250	3000×2248
Select fixed focal length (mm)	35		
Working distance (mm)	$\frac{35\sqrt{3000^2 + 2000^2}}{31.45} = 4013$	$\frac{35\sqrt{3000^2 + 2250^2}}{26.28} = 4995$	$\frac{35\sqrt{3000^2 + 2248^2}}{37.4} = 3509$

Table 1. Properties and calculation values of Basler cameras

The formulas used in the table above are:

- **Camera FOV** FOV height $= \text{FOV width} \cdot \dfrac{\text{Camera resolution height}}{\text{Camera resolution width}}$.
- **Working distance** Working distance $= \dfrac{\text{FOV} \times \text{Focal length}}{\text{Sensor size}}$.

Then, we access the Basler website http://tinyurl.com/28prxsze, choose appropriate camera and input the parameters Object width (3000 mm), Focal length (35 mm) to check the calculated value:

Model	Formula's result	Website's result	Results link
boA8100-16cc/16cm	4013 mm	4043 mm	Click here
boA6500-36cc/36cm	4995 mm	5035 mm	Click here
boA9344-30cc/30cm/70cc/70cm	3509 mm	3547 mm	Click here

Table 2. Working distance results

Exercise 2.

The matrix of the picture is $\begin{bmatrix} 40 & 20 & 30 & 30 \\ 80 & 40 & 100 & 110 \\ 120 & 160 & 40 & 150 \\ 220 & 230 & 240 & 250 \end{bmatrix}$.

a) The histogram equalization table of the picture matrix:

Pixel	Number of pixels	Probability	Cumulative density function (CDF)	New pixel
20	1	0.0625	1	0
30	2	0.125	3	34
40	3	0.1875	6	85
80	1	0.0625	7	102
100	1	0.0625	8	119
110	1	0.0625	9	136
120	1	0.0625	10	153
150	1	0.0625	11	170
160	1	0.0625	12	187
220	1	0.0625	13	204
230	1	0.0625	14	221
240	1	0.0625	15	238
250	1	0.0625	16	255

Table 3. Histogram equalization table

The new pixel values are calculated according to the following formula:

$$r(\text{Pixel}) = \frac{(\text{CDF} - \text{CDF}_{\text{min}})(\text{Number of gray levels used} - 1)}{\text{Number of elements in the matrix} - \text{CDF}_{\text{min}}}$$

From **Table 3**, we have $\text{CDF}_{\text{min}} = 1$. There are 16 elements in the matrix, and OpenCV libraries use 256 gray levels for histogram equalization. So, the formula becomes:

$$r(\text{Pixel}) = 17(\text{CDF} - 1)$$

Here are some values calculated according to the formula above:

$$\begin{aligned} r(40) &= 17(6-1) = 85 \\ r(100) &= 17(8-1) = 119 \\ r(220) &= 17(13-1) = 204 \\ r(240) &= 17(15-1) = 238 \end{aligned}$$

The image matrix after histogram equalization:

$$\begin{bmatrix} 85 & 0 & 34 & 34 \\ 102 & 85 & 119 & 136 \\ 153 & 187 & 85 & 170 \\ 204 & 221 & 238 & 255 \end{bmatrix}$$

b) The following code listing uses C++ and OpenCV 4.9.0 to perform histogram equalization in section a):

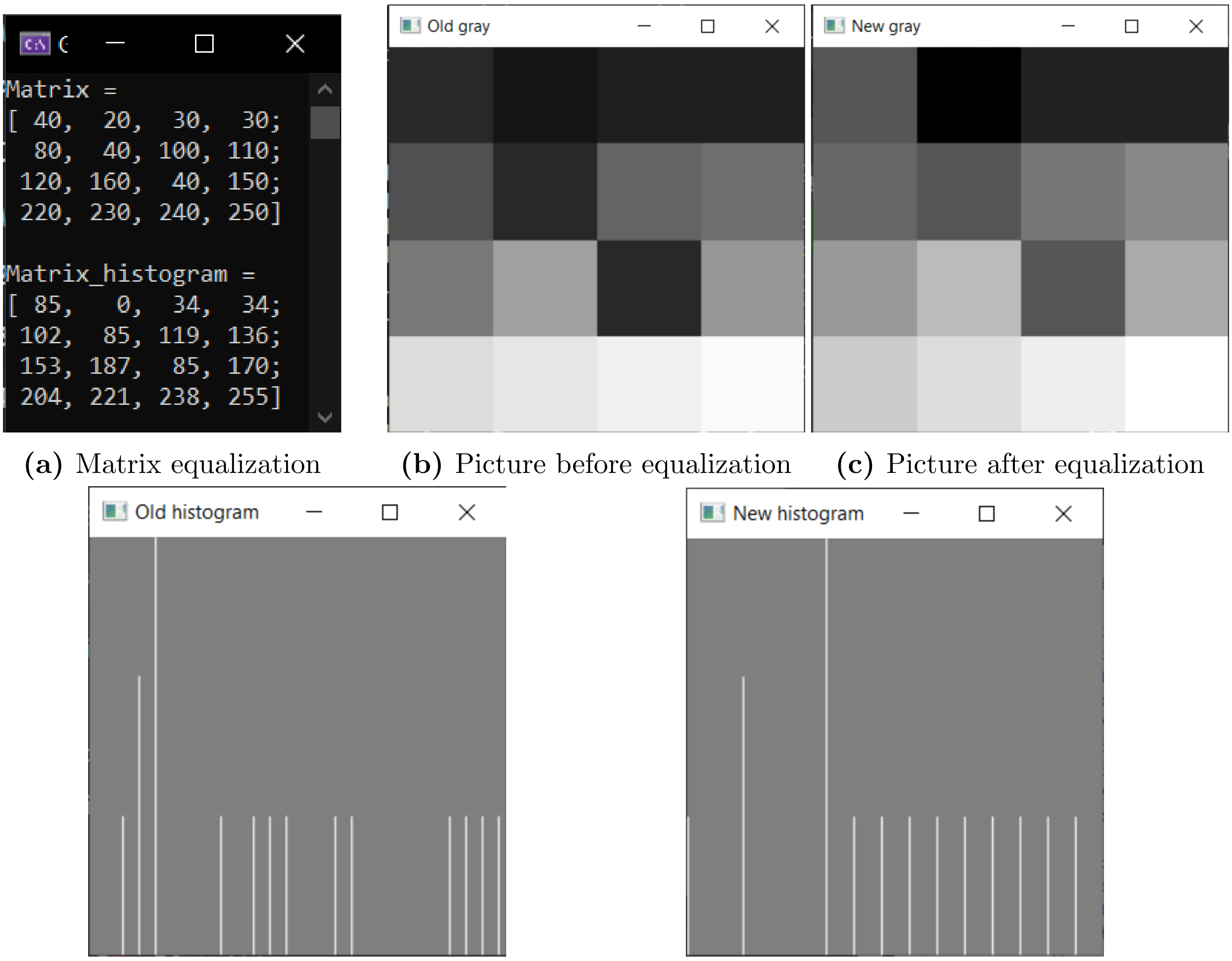
Code listing 1. Exercise 2.b)

```
#include "opencv2/opencv.hpp"

using namespace cv;
using namespace std;

void histogram(string const& name, Mat const& Image)
{
  int bin = 255;
  int histsize[] = { bin };
  float range[] = { 0,255 };
  const float* ranges[] = { range };
  Mat hist;
  int chanel[] = { 0 };
  int hist_heigt = 256;
  Mat hist_image = Mat::zeros(hist_heigt, bin, CV_8SC3);
  calcHist(&Image, 1, chanel, Mat(), hist, 1, histsize, ranges, true, false);
  double max_val = 0;
  minMaxLoc(hist, 0, &max_val);
  for (int i = 0; i < bin; i++)
  {
    float binV = hist.at<float>(i);
    int height = cvRound(binV * hist_heigt / max_val);
    line(hist_image, Point(i, hist_heigt - height), Point(i, hist_heigt), Scalar::all(255));
  }
  imshow(name, hist_image);
}
int main(int argv, char** argc)
{
  float img[16] = { 40,20,30,30,
            80,40,100,110,
            120,160,40,150,
            220,230,240,250 };
  Mat gray_img = Mat(4, 4, CV_32F, img);
  Mat gray_img_his;
  namedWindow("Old gray", WINDOW_FREERATIO);
  namedWindow("New gray", WINDOW_FREERATIO);
  gray_img.convertTo(gray_img, CV_8UC1);
  equalizeHist(gray_img, gray_img_his);
  cout << "Matrix = " << endl << "" << gray_img << endl << endl;
  cout << "Matrix_histogram = " << endl << "" << gray_img_his << endl << endl;
  imshow("Old gray", gray_img);
  imshow("New gray", gray_img_his);
  histogram("Old histogram", gray_img);
  histogram("New histogram ", gray_img_his);
  waitKey();
}
```

The results of the code above:



(a) Matrix equalization **(b)** Picture before equalization **(c)** Picture after equalization

(d) Histogram before equalization **(e)** Histogram after equalization

Figure 1. The picture before and after histogram equalization

c) We choose the value $T_0 = \dfrac{\max + \min}{2} = \dfrac{250 + 20}{2} = 135$ then we divide the elements of the matrix into 2 groups:

- Group of elements smaller than T_0: $40, 20, 30, 30, 80, 40, 100, 110, 120, 40$. The average value of this group is

$$m_1 = \frac{40 + 20 + 30 + 30 + 80 + 40 + 100 + 110 + 120 + 40}{10} = 61$$

- Group of elements greater than T_0: $160, 150, 220, 230, 240, 250$. The average value of this group is

$$m_2 = \frac{160 + 150 + 220 + 230 + 240 + 250}{6} = 208.33$$

The value $T_1 = \dfrac{m_1 + m_2}{2} = \dfrac{61 + 208.33}{2} = 134.67$. So $\Delta T = |T_1 - T_0| = |134.67 - 135| = 0.33$

We continue dividing the elements into 2 groups:

- Group of elements smaller than T_1: $40, 20, 30, 30, 80, 40, 100, 110, 120, 40$. The average value of this group is

$$m_1 = \frac{40 + 20 + 30 + 30 + 80 + 40 + 100 + 110 + 120 + 40}{10} = 61$$

- Group of elements greater than T_1: $160, 150, 220, 230, 240, 250$. The average value of this group is

$$m_2 = \frac{160 + 150 + 220 + 230 + 240 + 250}{6} = 208.33$$

The value $T_2 = \dfrac{m_1 + m_2}{2} = \dfrac{61 + 208.33}{2} = 134.67$. So $\Delta T = |T_2 - T_1| = 0$.
Therefore, the Otsu thresholding value is $T = 134.67$ and the result matrix is shown below:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 255 & 0 & 255 \\ 255 & 255 & 255 & 255 \end{bmatrix}$$

The following code listing uses C++ and OpenCV 4.9.0 to check the result above:

Code listing 2. Exercise 2.c)

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

int main() {
    Mat source = (Mat_<uint8_t>(4, 4) << 40, 20, 30, 30,
        80, 40, 100, 110,
        120, 160, 40, 150,
        220, 230, 240, 250);
    Mat dst;
    double thresh = 0, maxValue = 255;
    long double thres = threshold(source, dst, thresh, maxValue, THRESH_OTSU);

    cout << "Image matrix = " << endl << "" << source << endl;
    cout << "Otsu Threshold T = " << thres << endl;
    cout << "Otsu Threshold result" << endl << "" << dst << endl;

    waitKey(0);
    return 0;
}
```

The result from the code above:

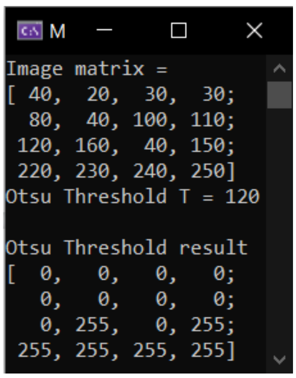


Figure 2. Otsu Thresholding result from OpenCV

Exercise 3.

The matrix of the picture is $\begin{bmatrix} 1 & 2 & 3 & 4 & 10 & 11 \\ 4 & 10 & 11 & 16 & 12 & 18 \\ 16 & 12 & 18 & 22 & 24 & 25 \\ 23 & 24 & 25 & 25 & 4 & 7 \end{bmatrix}$ and the template is $\begin{bmatrix} 4 & 10 & 11 \\ 16 & 12 & 18 \\ 23 & 24 & 25 \end{bmatrix}$.

Denote the element in the x-th row and y-th column of the image matrix is $I(x, y)$ and the element in the x'-th row and y'-th column of the template matrix is $T(x', y')$. The Normalized Cross-Correlation (NCC) coefficients are determined according to the following formula:

$$R(x, y) = \frac{\sum_{x',y'} [T(x', y') \cdot I(x + x' - 1, y + y' - 1)]}{\sqrt{\sum_{x',y'} T^2(x', y') \cdot \sum_{x',y'} I^2(x + x' - 1, y + y' - 1)}}$$

Substitute the $\sum_{x',y'} T^2(x', y') = 4^2 + 10^2 + 11^2 + 16^2 + 12^2 + 18^2 + 23^2 + 24^2 + 25^2 = 2691$ into the $R(x, y)$ formula, we obtain:

$$R(x, y) = \frac{\sum_{x',y'} [T(x', y') \cdot I(x + x' - 1, y + y' - 1)]}{\sqrt{2691 \sum_{x',y'} I^2(x + x' - 1, y + y' - 1)}}$$

Here are some values obtained from the formula above:

$$R(1, 1) = \frac{1 \cdot 4 + 2 \cdot 10 + 3 \cdot 11 + 4 \cdot 16 + 10 \cdot 12 + 11 \cdot 18 + 16 \cdot 23 + 12 \cdot 24 + 18 \cdot 25}{\sqrt{2691(1^2 + 2^2 + 3^2 + 4^2 + 10^2 + 11^2 + 16^2 + 12^2 + 18^2)}} = 0.953826374$$

$$R(1, 2) = \frac{2 \cdot 4 + 3 \cdot 10 + 4 \cdot 11 + 10 \cdot 16 + 11 \cdot 12 + 16 \cdot 18 + 12 \cdot 23 + 18 \cdot 24 + 22 \cdot 25}{\sqrt{2691(2^2 + 3^2 + 4^2 + 10^2 + 11^2 + 16^2 + 12^2 + 18^2 + 22^2)}} = 0.969316528$$

$$R(1, 3) = \frac{3 \cdot 4 + 4 \cdot 10 + 10 \cdot 11 + 11 \cdot 16 + 16 \cdot 12 + 12 \cdot 18 + 18 \cdot 23 + 222 \cdot 24 + 24 \cdot 25}{\sqrt{2691(3^2 + 4^2 + 10^2 + 11^2 + 16^2 + 12^2 + 18^2 + 22^2 + 24^2)}} = 0.978928905$$

$$R(2, 1) = \frac{4 \cdot 4 + 10 \cdot 10 + 11 \cdot 11 + 16 \cdot 16 + 12 \cdot 12 + 18 \cdot 18 + 23 \cdot 23 + 24 \cdot 24 + 25 \cdot 25}{\sqrt{2691(4^2 + 10^2 + 11^2 + 16^2 + 12^2 + 18^2 + 23^2 + 24^2 + 25^2)}} = 1$$

The NCC coefficients matrix is shown below:

$$\begin{bmatrix} R(1,1) & R(1,2) & R(1,3) & R(1,4) \\ R(2,1) & R(2,2) & R(2,3) & R(2,4) \end{bmatrix} = \begin{bmatrix} 0.953826374 & 0.969316528 & 0.978928905 & 0.999848171 \\ 1 & 0.982228851 & 0.888346493 & 0.802686196 \end{bmatrix}$$

The position of the image is similar to the template is $R(2, 1)$. Applying binary thresholding to the NCC coefficients matrix with the threshold value $T = 0.98$ and the maximum value 1, we obtain the thresholded matrix below:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

The following code listing is a C++ code using OpenCV 4.9.0 to check the calculation above:

Code listing 3. Exercise 3

```
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

int main()
{
    Mat img = (Mat_<float>(4, 6) << 1, 2, 3, 4, 10, 11,
        4, 10, 11, 16, 12, 18,
        16, 12, 18, 22, 24, 25,
        23, 24, 25, 25, 4, 7);
    Mat temp = (Mat_<float>(3, 3) << 4, 10, 11,
        16, 12, 18,
        23, 24, 25);

    Mat ncc;
    matchTemplate(img, temp, ncc, TM_CCORR_NORMED);
    cout << "NCC coefficients matrix :" << endl << ncc << endl;
    double minVal, maxVal;
    Point minLoc, maxLoc;
    minMaxLoc(ncc, &minVal, &maxVal, &minLoc, &maxLoc);
    cout << "The position of the image is similar to the template is R(" << maxLoc.y + 1 << "," << maxLoc.x + 1 << ")" << endl;
    Mat bin;
    threshold(ncc, bin, 0.98, 1, THRESH_BINARY);
    vector<Point> locs;
    findNonZero(bin, locs);
    cout << "The positions with values greater than 0.98 are:" << endl;
    for (auto loc : locs) {
        cout << "R(" << loc.y + 1 << "," << loc.x + 1 << ")\t";
    }
    cout << "\nObtain thresholding the NCC coefficients matrix" << endl << "" << bin << endl;
    waitKey();
}
```

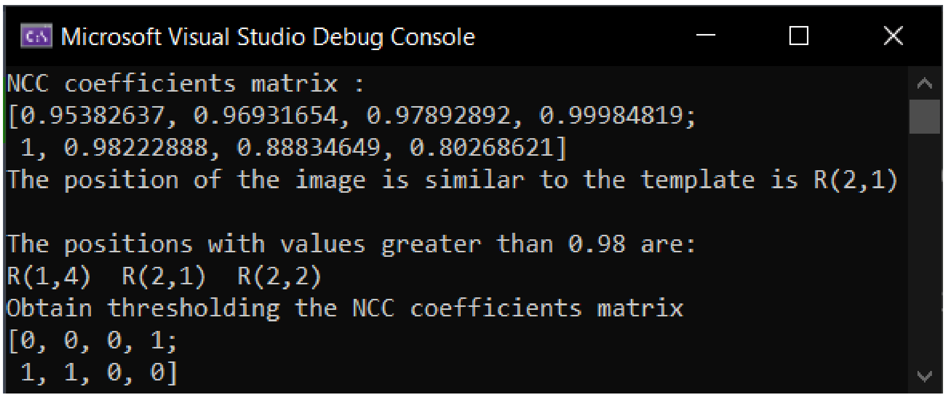


Figure 3. Template matching with NCC coefficients

Exercise 4.

The matrix of the picture is $\begin{bmatrix} 9 & 10 & 11 & 10 & 9 & 10 & 11 \\ 10 & 9 & 100 & 100 & 100 & 10 & 11 \\ 10 & 100 & 10 & 11 & 10 & 100 & 11 \\ 10 & 9 & 100 & 100 & 100 & 13 & 11 \\ 10 & 10 & 10 & 10 & 10 & 13 & 11 \end{bmatrix}$. Adding duplicate border to the image matrix $\begin{bmatrix} 9 & 9 & 10 & 11 & 10 & 9 & 10 & 11 & 11 \\ 9 & 9 & 10 & 11 & 10 & 9 & 10 & 11 & 11 \\ 10 & 10 & 9 & 100 & 100 & 100 & 10 & 11 & 11 \\ 10 & 10 & 100 & 10 & 11 & 10 & 100 & 11 & 11 \\ 10 & 10 & 9 & 100 & 100 & 100 & 13 & 11 & 11 \\ 10 & 10 & 10 & 10 & 10 & 10 & 13 & 11 & 11 \\ 10 & 10 & 10 & 10 & 10 & 10 & 13 & 11 & 11 \end{bmatrix}$.

a) The Gaussian Filter function is $G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$. Substituting $\sigma = 0.3$ into $G_\sigma(x,y)$, then we calculate the kernel matrix:

$$\begin{bmatrix} G_{0.3}(-1,-1) & G_{0.3}(0,-1) & G_{0.3}(1,-1) \\ G_{0.3}(-1,0) & G_{0.3}(0,0) & G_{0.3}(1,0) \\ G_{0.3}(-1,1) & G_{0.3}(0,1) & G_{0.3}(1,1) \end{bmatrix} = \begin{bmatrix} 0.000026 & 0.006836 & 0.000026 \\ 0.006836 & 1.768388 & 0.006836 \\ 0.000026 & 0.006836 & 0.000026 \end{bmatrix}$$

The sum of all the elements of the matrix is 1.79584. Therefore, we obtain the filter kernel matrix:

$$\frac{1}{1.79584}\begin{bmatrix} 0.000026 & 0.006836 & 0.000026 \\ 0.006836 & 1.768388 & 0.006836 \\ 0.000026 & 0.006836 & 0.000026 \end{bmatrix}$$

The matrix after filtering with Gaussian kernel is

$$\begin{bmatrix} 9.007643 & 9.997518 & 11.332445 & 10.345263 & 9.355418 & 10.001339 & 10.996164 \\ 9.993696 & 9.700515 & 98.968198 & 99.313280 & 98.964392 & 10.689050 & 10.997488 \\ 10.342585 & 98.619280 & 11.034269 & 11.675240 & 11.034343 & 98.642151 & 11.338822 \\ 9.997518 & 9.700515 & 98.964392 & 99.313280 & 98.979663 & 13.654627 & 11.008953 \\ 9.999985 & 9.997518 & 10.343924 & 10.345263 & 10.355448 & 12.982143 & 11.007673 \end{bmatrix}$$

The elements of the matrix after filtering are calculated using the convolution formula. Here are some of the calculated element values:

$$a(1,1) = \frac{9\cdot 0.000026 + 9\cdot 0.006836 + 10\cdot 0.000026 + 9\cdot 0.006836 + 9\cdot 1.768388}{1.79584} + \frac{10\cdot 0.006836 + 10\cdot 0.000026 + 10\cdot 0.006836 + 9\cdot 0.000026}{1.79584} = 9.007643$$

$$a(1,2) = \frac{9\cdot 0.000026 + 10\cdot 0.006836 + 11\cdot 0.000026 + 9\cdot 0.006836 + 10\cdot 1.768388}{1.79584} + \frac{11\cdot 0.006836 + 10\cdot 0.000026 + 9\cdot 0.006836 + 100\cdot 0.000026}{1.79584} = 9.997518$$

$$a(2,2) = \frac{9\cdot 0.000026 + 10\cdot 0.006836 + 11\cdot 0.000026 + 10\cdot 0.006836 + 9\cdot 1.768388}{1.79584} + \frac{100\cdot 0.006836 + 10\cdot 0.000026 + 100\cdot 0.006836 + 10\cdot 0.000026}{1.79584} = 9.700515$$

$$a(3,3) = \frac{9\cdot 0.000026 + 100\cdot 0.006836 + 100\cdot 0.000026 + 100\cdot 0.006836 + 10\cdot 1.768388}{1.79584} + \frac{11\cdot 0.006836 + 9\cdot 0.000026 + 100\cdot 0.006836 + 100\cdot 0.000026}{1.79584} = 11.034269$$

The following code listing is a C++ code using OpenCV 4.9.0 to check the calculation above:

Code listing 4. Exercise 4.a)

```cpp
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

int main()
{
    float sigma = 0.3;
    Mat src = (Mat_<float>(5, 7) << 9, 10, 11, 10, 9, 10, 11,
        10, 9, 100, 100, 100, 10, 11,
        10, 100, 10, 11, 10, 100, 11,
        10, 9, 100, 100, 100, 13, 11,
        10, 10, 10, 10, 10, 13, 11);
    Mat dst;
    GaussianBlur(src, dst, Size(3, 3), sigma, sigma, BORDER_REPLICATE);
    cout << "Image Matrix" << endl << "" << src << endl;
    cout << "Gaussian Blur" << endl << "" << dst << endl;
    waitKey();
}
```

```
Microsoft Visual Studio Debug Console
Image Matrix
[9, 10, 11, 10, 9, 10, 11;
 10, 9, 100, 100, 100, 10, 11;
 10, 100, 10, 11, 10, 100, 11;
 10, 9, 100, 100, 100, 13, 11;
 10, 10, 10, 10, 10, 13, 11]
Gaussian Blur
[9.0076437, 9.9975176, 11.332444, 10.345263, 9.3554182, 10.00134, 10.996163;
 9.9936962, 9.7005138, 98.968185, 99.313263, 98.964386, 10.689051, 10.997487;
 10.342585, 98.619278, 11.034269, 11.67524, 11.034344, 98.642143, 11.338821;
 9.9975176, 9.7005138, 98.964378, 99.313263, 98.979652, 13.654627, 11.008953;
 9.9999857, 9.9975176, 10.343925, 10.345263, 10.355448, 12.982142, 11.007672]
```

Figure 4. Gaussian Blur filtering result from OpenCV

b) The Sobel operator uses two different kernel matrices:

- Taking the derivative with respect to x: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
- Taking the derivative with respect to y: $M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Using the Sobel M_x operator, we obtain the image after filtering:

$$\begin{bmatrix} 2 & 96 & 91 & -6 & -90 & -83 & 4 \\ 89 & 182 & 93 & -2 & -91 & -175 & -86 \\ 178 & 180 & 4 & 0 & 1 & -176 & -179 \\ 88 & 180 & 93 & 0 & -82 & -176 & -95 \\ -1 & 90 & 91 & 0 & -78 & -86 & -8 \end{bmatrix}$$

Here are some values that were calculated through the Sobel M_x operator:

$a(1,1) = 9 \cdot (-1) + 9 \cdot 0 + 10 \cdot 1 + 9 \cdot (-2) + 9 \cdot 0 + 10 \cdot 2 + 10 \cdot (-1) + 10 \cdot 0 + 9 \cdot 1 = 2$
$a(2,2) = 9 \cdot (-1) + 10 \cdot 0 + 11 \cdot 1 + 10 \cdot (-2) + 9 \cdot 0 + 100 \cdot 2 + 10 \cdot (-1) + 100 \cdot 0 + 10 \cdot 1 = 182$

Using the Sobel M_y operator, we obtain the image after filtering:

$$\begin{bmatrix} 2 & 88 & 267 & 360 & 272 & 91 & 0 \\ 93 & 180 & 89 & 2 & 93 & 181 & 90 \\ 0 & 0 & 0 & 0 & 3 & 6 & 3 \\ -90 & -180 & -91 & -2 & -88 & -174 & -87 \\ 1 & -88 & -269 & -360 & -270 & -90 & 0 \end{bmatrix}$$

Here are some values that were calculated through the Sobel M_y operator:

$a(1,1) = 9 \cdot (-1) + 9 \cdot (-2) + 10 \cdot (-1) + 9 \cdot 0 + 9 \cdot 0 + 10 \cdot 0 + 10 \cdot 1 + 10 \cdot 2 + 9 \cdot 1 = 2$
$a(2,2) = 9 \cdot (-1) + 10 \cdot (-2) + 11 \cdot (-1) + 10 \cdot 0 + 9 \cdot 0 + 100 \cdot 0 + 10 \cdot 1 + 100 \cdot 2 + 10 \cdot 1 = 180$

Code listing 5. Exercise 4.b)

```
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

int main()
{
    Mat src = (Mat_<double>(5, 7) << 9, 10, 11, 10, 9, 10, 11,
        10, 9, 100, 100, 100, 10, 11,
        10, 100, 10, 11, 10, 100, 11,
        10, 9, 100, 100, 100, 13, 11,
        10, 10, 10, 10, 10, 13, 11);

    Mat sobelx, sobely;
    Sobel(src, sobelx, CV_64F, 1, 0, 3, 1, 0, BORDER_REPLICATE);
    Sobel(src, sobely, CV_64F, 0, 1, 3, 1, 0, BORDER_REPLICATE);
    cout << "Sobel x = " << endl << "" << sobelx << endl << endl;
    cout << "Sobel y = " << endl << "" << sobely << endl << endl;
    waitKey();
}
```

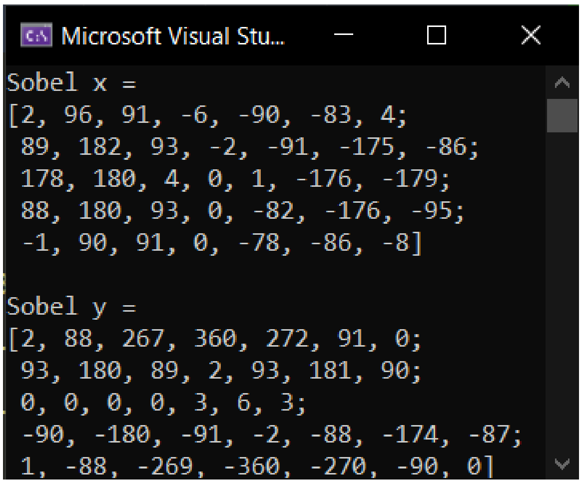


Figure 5. Sobel filtering result from OpenCV