Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Отчёт

по лабораторной работе №2

Дисциплина: Техническое зрение
Тема: Моделирование движения робота с использованием библиотеки OpenCV

Студенты гр. 3331506/70401 Тестерева М.Н.
Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург
2020 г.

Задание

Реализовать сглаживающий квадратный фильтр (box filter) с ядром 3x3 и якорем в центре ядра. Обработка границ – завернуть за край (wrap border).

Выполнение задания

Считываем исходное изображение, создаем заготовки под изображения для фильтрации:

```
Mat Original_Image = imread("Lenna.png", IMREAD_GRAYSCALE);
Mat Filtered_Image = Original_Image.clone();
Mat Lib_Filtered_Image = Original_Image.clone();
```

Задаём размеры ядра:

```
Size filter_size(3, 3);
```

Создаём счётчик для измерения времени работы функций:

```
TickMeter timer_ms;
```

Выведем исходное изображение:

```
imshow("Original image", Original_Image);
```

Обработаем изображение с помощью встроенной функции OpenCV blur.

Сначала сбросим и запустим счётчик времени, вызовем обрабатывающую функцию, остановим счётчик и выведем в отдельном окне – результат фильтрации, в командном окне – время работы функции.

```
timer_ms.reset();
timer_ms.start();
blur(Lib_Filtered_Image, Lib_Filtered_Image, filter_size);
timer_ms.stop();
imshow("With library filtered image", Lib_Filtered_Image);
cout << "Lib_filter time = " << timer_ms.getTimeMilli() << " ms" << endl;
```

Обработаем изображение фильтром, созданным по заданию.

Порядок действий идентичен, за исключением функции, обрабатывающей границы изображения:

```
timer_ms.reset();
timer_ms.start();
Filtered_Image = Box_Filter(Original_Image, filter_size);
timer_ms.stop();
//обработка границ
Filtered_Image = Wrap_border(Filtered_Image);
imshow("For task filtered image", Filtered_Image);
cout << "Task_box_filter time = " << timer_ms.getTimeMilli() << " ms" << endl;
```

Сравниваем обработанные функцией библиотеки и пользовательской функцией изображения:

```
Mat Comparative_Image = Compare(Filtered_Image, Lib_Filtered_Image);
imshow("Comparative image", Comparative_Image);
```

Непосредственно функция, реализующая пользовательский фильтр:

```
static Mat Box_Filter(Mat Image, Size filter_size) {
    int i_width = Image.cols;
    int i_height = Image.rows;

    Mat Filtred_Image(i_height, i_width, CV_8UC1);

    for (int i = 0; i < i_height - filter_size.height; i++)
    {
        for (int j = 0; j < i_width - filter_size.width; j++)
        {
            Rect rp(i, j, filter_size.width, filter_size.height);
            Mat roi = Image(rp);
            Scalar roi_avg = mean(roi);
            Filtred_Image.at<uchar>(Point(i + filter_size.width / 2, j + filter_size.height / 2)) = floor(roi_avg[0] + 0.5);
        }
    }
        return Filtred_Image;
}
```

Функция *mean* возвращает среднее значение по каналам независимо друг от друга и возвращает в массив.

Непосредственно функция, реализующая обработку границ:

```
static Mat Wrap_border(Mat Filtered_Image) {
    Rect cut(2, 2, Filtered_Image.cols - 4, Filtered_Image.rows - 4);
    Size buff_size(Filtered_Image.cols - 4, Filtered_Image.rows - 4);
    Mat buffer_Img(buff_size, Filtered_Image.type());
    Mat Cut_Filtered_Image = Filtered_Image(cut);
    copyTo(Cut_Filtered_Image, buffer_Img, Cut_Filtered_Image);
    //imshow("cuted", Cut_Filtered_Image);
    copyMakeBorder(buffer_Img, Cut_Filtered_Image, 2, 2, 2, 2, BORDER_WRAP);
    //imshow("wraped", Cut_Filtered_Image);
    return Cut_Filtered_Image;
}
```

Непосредственно функция, реализующая сравнение:

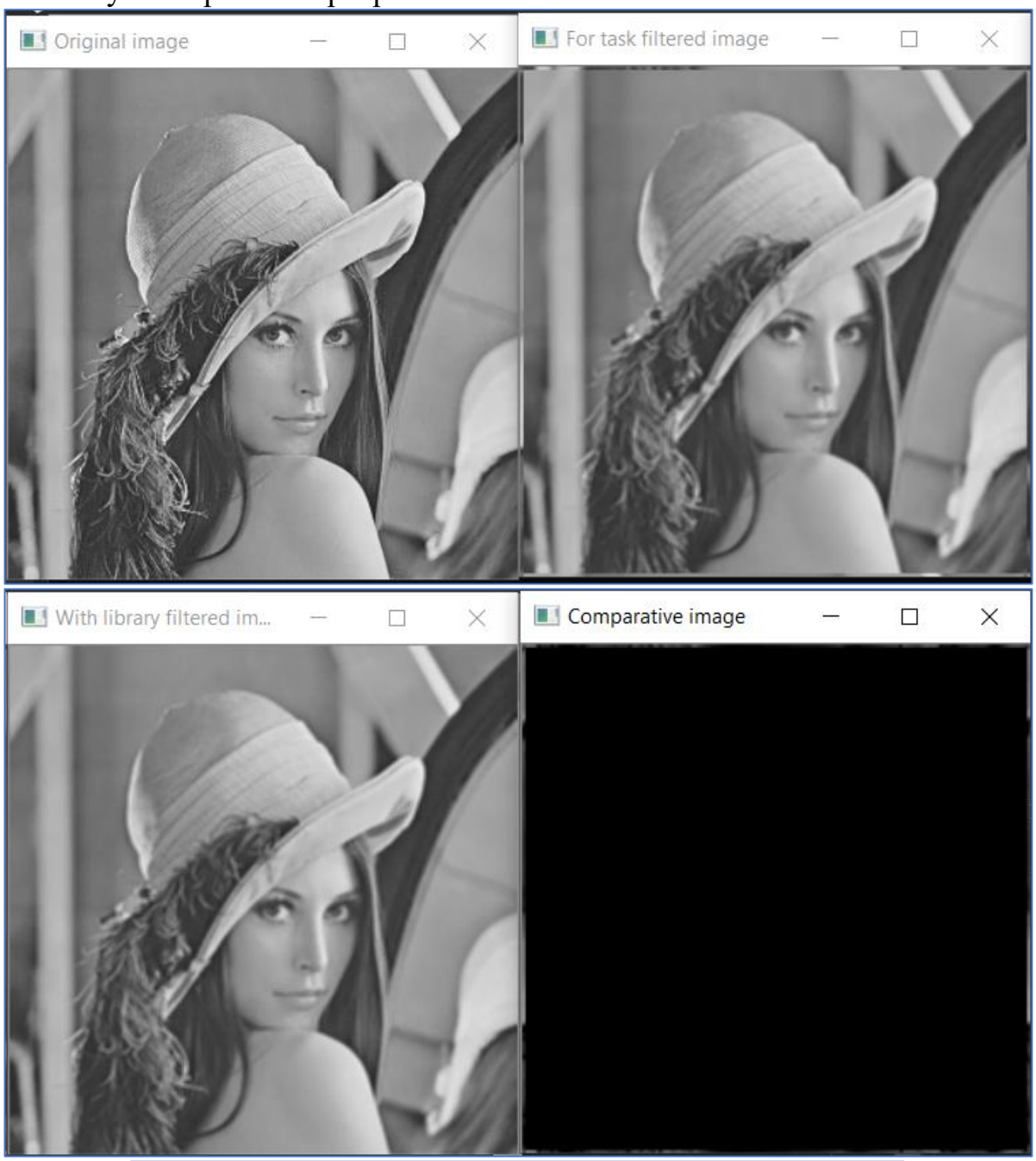
```
static Mat Compare(Mat Image1, Mat Image2) {
    int pixel_sum = 0;
    int pixel_correct = 0;

    if (Image1.cols == Image2.cols) { cout << "Same size" << endl; }
    else { cout << "Not the same size" << endl; }

    for (int i = 0; i < Image1.cols; i++)
    {
        for (int j = 0; j < Image1.rows; j++)
        {
            if (int(Image1.at<uchar>(i,j)) == int(Image2.at<uchar>(i,j)))
            {
                pixel_correct++;
            }
            pixel_sum++;
        }
    }
    cout << ((pixel_correct * 100.0) / pixel_sum) << " % of similarity";

    Mat comperative;
    absdiff(Image1, Image2, comperative);
    return comperative;
}
```

Результат работы программы:




```
C:\Users\Maria\source\repos\Filter_2\x64\Release\Filter_2.exe
Lib_filter time = 0.1415 ms
Task_box_filter time = 11.595 ms
Same size
97.3667 % of similarity
```

Результат 97% объясняется увеличенной на 1 px рамкой пользовательского фильтра.