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

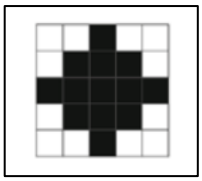
Студенты гр. 3331506/70401 Каретный Я.М.
Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург
2020 г.

Задание

Необходимо реализовать морфологическую операцию – открывающий фильтр (Erode → Dilate) с ядром diamond 5x5 и обработкой границ border constant. Форма ядра:



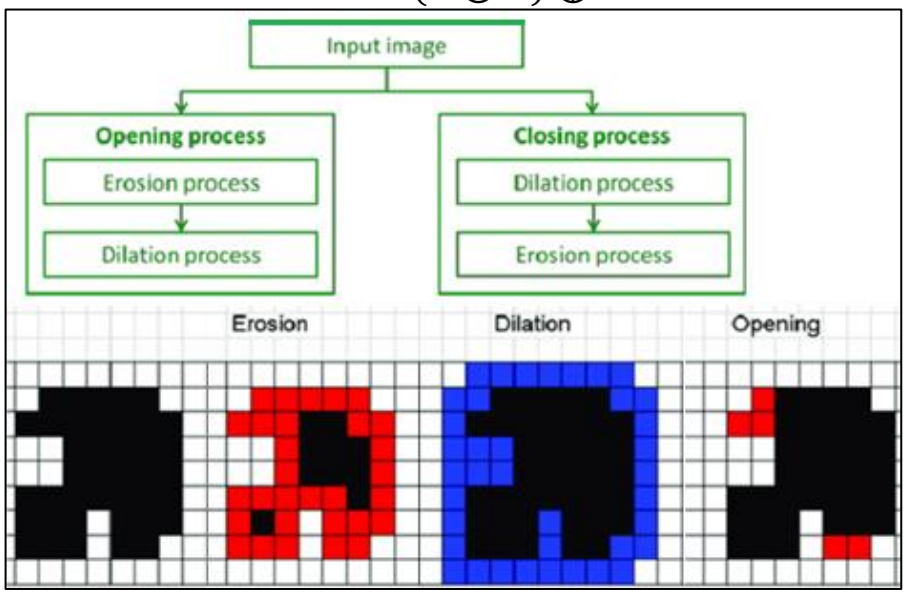
Описание

В открывающем фильтре сначала производится операция сужения (эрозия), а затем операция расширения (дилатация).

В результате из исходного множества удаляются граничные точки, т.е. результирующее множество меньше исходного на некоторые малые по размеру выступающие части.

Морфологическая операция открытия:

$$A \circ B = (A \ominus B) \oplus B$$



Реализация

Заданное ядро:

```
uint8_t kernelData[25] = { 255, 255,  0 , 255, 255,
                           255,  0 ,  0 ,  0 , 255,
                            0 ,  0 ,  0 ,  0 ,  0 ,
                           255,  0 ,  0 ,  0 , 255,
                           255, 255,  0 , 255, 255 };
Mat kernel(5, 5, CV_8UC1, kernelData);
```

Бинаризация исходного изображения:

```
Mat OriginalImg = imread("Test_image1.jpg", IMREAD_GRAYSCALE);
threshold(OriginalImg, OriginalImg, 128, 255, THRESH_BINARY);
```

Встроенная функция OpenCV, реализующая отрывающий фильтр, применение с измерением времени:

```
timer1.reset();
timer1.start();
Mat OpenCVFilterImg = library_opening_filter(OriginalImg, kernel);
timer1.stop();
imshow("OpenCV filtered image", OpenCVFilterImg);
cout << "OpenCV filter = " << timer1.getTimeMilli() << " ms" << endl;
```

описание:

```
static Mat library_opening_filter(Mat Image, Mat kernel) {
    Mat f_img(Image.rows, Image.cols, Image.type());
    morphologyEx(Image, f_img, MORPH_OPEN, kernel, Point(-1, -1), 1, BORDER_CONSTANT);
    return f_img;
}
```

Созданная по заданию функция, реализующая открывающий фильтр, применение с измерением времени:

```
timer1.reset();
timer1.start();
Mat MyFilterImg = custom_opening_filter(OriginalImg, kernel);
timer1.stop();
imshow("My filtered image", MyFilterImg);
cout << "My filter = " << timer1.getTimeMilli() << " ms" << endl;
```

описание:

- функция эрозии:

```
static Mat custom_erode(Mat Image, Mat kernel) {
    Mat f_img(Image.rows, Image.cols, Image.type());
    for (int i = 2; i < f_img.rows - 2; i++)
        for (int j = 2; j < f_img.cols - 2; j++) {
            int flag = 0;
            for (int m = 0; m < 5; m++) {
                for (int n = 0; n < 5; n++) {
                    if ((kernel.at<uchar>(m, n) == 255) && (Image.at<uchar>(i - 2 + m, j - 2 + n) == 0))
                        flag++;
                }
            }
            if (flag) f_img.at<uchar>(i, j) = 0;
            else f_img.at<uchar>(i, j) = Image.at<uchar>(i, j);
        }
    return f_img;
}
```

- функция дилатации:

```
static Mat custom_dilate(Mat Image, Mat kernel) {
    Mat f_img(Image.rows, Image.cols, Image.type());
    for (int i = 2; i < f_img.rows - 2; i++)
        for (int j = 2; j < f_img.cols - 2; j++) {
            int flag = 0;
            for (int m = 0; m < 5; m++) {
                for (int n = 0; n < 5; n++) {
                    if (kernel.at<uchar>(m, n) == 255 && Image.at<uchar>(i - 2 + m, j - 2 + n) == 255)
                        flag++;
                }
            }
            if (flag) f_img.at<uchar>(i, j) = 255;
            else f_img.at<uchar>(i, j) = Image.at<uchar>(i, j);
        }
    return f_img;
}
```

- функция фильтра:

```
static Mat custom_opening_filter(Mat Image, Mat kernel) {
    Mat f_img1(Image.rows+4, Image.cols+4, Image.type());
    //add border
    copyMakeBorder(Image, f_img1, 2, 2, 2, 2, BORDER_CONSTANT);
    //erode
    f_img1 = custom_erode(f_img1, kernel);
    //dilate
    f_img1 = custom_dilate(f_img1, kernel);
    //delete border
    Rect cut_rect(2, 2, f_img1.cols - 4, f_img1.rows - 4);
    Size buff_size(f_img1.cols - 4, f_img1.rows - 4);
    Mat buff_img(buff_size, Image.type());
    Mat f_img2 = f_img1(cut_rect);

    return f_img2;
}
```

Использование функции, реализующей сравнение двух изображений:

```
Mat DiffImg = simile(MyFilterImg, OpenCVFilterImg);
imshow("Diff image", DiffImg);
```

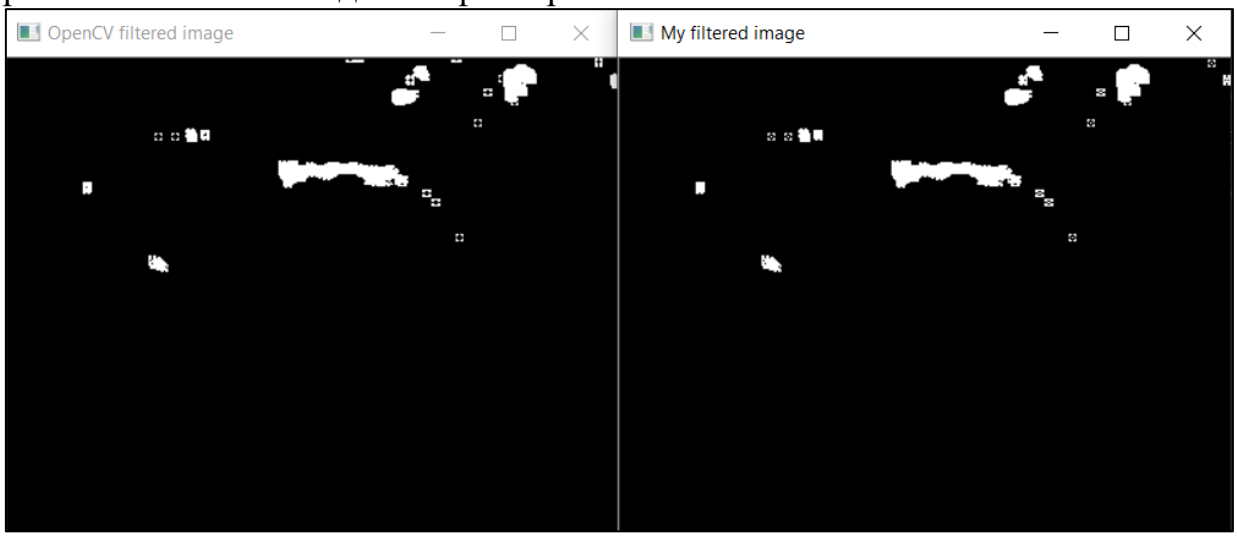
описание:

```
static Mat simile(Mat Image1, Mat Image2) {

    Mat c_img(Image1.rows, Image1.cols, Image1.type());
    cv::absdiff(Image1, Image2, c_img);

    float uncorrect = 0;
    float pixels = 0;
    for (int i = 0; i < c_img.rows; i++)
        for (int j = 0; j < c_img.cols; j++)
        {
            if (c_img.at<uchar>(i, j) != 0) { uncorrect++; }
            pixels++;
        }
    float sim = ((pixels - uncorrect) / pixels) * 100;
    cout << "Similar (%): " << sim << endl;
    return c_img;
}
```

Результаты

Исходное бинарное изображение:

Изображения, полученные после фильтрации встроенным и реализованным по заданию фильтрами:



Изображение-сравнение:


```
OpenCV filter = 0.2392 ms
My filter = 5.7304 ms
Similar (%): 99.9252
```