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: Замыкающий фильтр

Студент гр. 3331506/70401 Кондратченко О.О.

Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург

2020

Задание

Используя средствами языка C++ и библиотеками OpenCV, реализовать замыкающий фильтр (Dilate->Erode) для бинарного изображения с ядром, приведенном на рисунке 1. Граница имеет тип *border constant*.

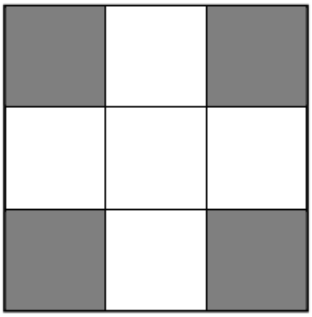

Рисунок 1 – Ядро свертки border constant

Ход работы

Описание алгоритма

Замыкающий фильтр состоит в последовательном применении дилатации и эрозии с одинаковым структурным элементом. Его морфологический эффект заключается в заполнении небольших отверстий в изображении с сохранением формы и размера объектов на изображении.

Суть дилатации сводится к тому, что осуществляется проходом шаблона (3x3 *border constant*) по всему изображения с поиском локального минимума к интенсивностям пикселей изображения, которые “накладываются” шаблоном.

Суть эрозии сводится к тому, что осуществляется проходом шаблона (3x3 *border constant*) по всему изображения с поиском локального максимума к интенсивностям пикселей изображения, которые “накладываются” шаблоном.

Реализация алгоритма

Для описания замыкающего фильтра был создан класс *closeFilter*, изображенный на рисунке 2.

```
class closeFilter
{
public:
    closeFilter();
    ~closeFilter();
    void addBorder();
    void delBorder();
    void setImage(Mat inputImage);
    void setKernel(Mat kernel);
    void showImage();
    Mat getImage();
    void binarizeImage();
    void morphClose();
    void erodeCustom();
    void dilateCustom();
private:
    Mat m_image;
    Mat m_kernel;
};
```

Рисунок 2 – Класс close*Filter*

Алгоритм реализован для бинарных изображений, перед тем как Публичный метод *morphClose()*, код которого представлен на рисунке 3, сначала производится бинаризация изображения, затем последовательное применение дилатации и эрозии, расширив изображение перед морфологическими операциями, а после – удалением границ.

```
void closeFilter::morphClose()
{
    uint32_t start_time = clock();

    binarizeImage();
    addBorder();
    dilateCustom();
    erodeCustom();
    delBorder();

    uint32_t end_time = clock();
    uint32_t search_time = end_time - start_time;
    cout << "Время работы: " << search_time << endl;
}
```

Рисунок 3 – Метод *morphClose()*

Для вычисления расширения изображения реализован метод *addBorder()*, код которого представлен на рисунке 4.

```
void closeFilter::addBorder()
{
    Mat imgWithBorder(m_image.rows + 2, m_image.cols + 2, m_image.type());

    for (int row = 1; row < imgWithBorder.rows - 1; row++)
    {
        for (int col = 1; col < imgWithBorder.cols - 1; col++)
        {
            imgWithBorder.at<uchar>(row, col) = m_image.at<uchar>(row - 1, col - 1);
        }
    }

    for (int row = 0; row < imgWithBorder.rows; row++)
    {
        imgWithBorder.at<uchar>(row, 0) = 0;
        imgWithBorder.at<uchar>(row, imgWithBorder.cols - 1) = 0;
    }

    for (int col = 1; col < imgWithBorder.cols - 1; col++)
    {
        imgWithBorder.at<uchar>(0, col) = 0;
        imgWithBorder.at<uchar>(imgWithBorder.rows - 1, col) = 0;
    }
    m_image = imgWithBorder;
}
```

Рисунок 4 – Метод *addBorder()*

Для реализации дилатации изображения был создан метод *dilateCustom()*, код которого изображен на рисунке 5.

```
void closeFilter::dilateCustom()
{
    Mat imageDilate(m_image.rows, m_image.cols, m_image.type());
    for (int rowsImg = 1; rowsImg < m_image.rows - 1; rowsImg++)
    {
        for (int colsImg = 1; colsImg < m_image.cols - 1; colsImg++)
        {
            int flag = 0;
            for (int rowsKern = 0; rowsKern < 3; rowsKern++)
            {
                for (int colsKern = 0; colsKern < 3; colsKern++)
                {
                    if (m_kernel.at<uchar>(rowsKern, colsKern) == 255 &&
                        m_image.at<uchar>(rowsImg - 1 + rowsKern, colsImg - 1 + colsKern) == 255)
                    {
                        flag++;
                    }
                }
            }
            if (flag)
            {
                imageDilate.at<uchar>(rowsImg, colsImg) = 255;
            }
            else
            {
                imageDilate.at<uchar>(rowsImg, colsImg) = m_image.at<uchar>(rowsImg, colsImg);
            }
        }
    }
    m_image = imageDilate;
}
```

Рисунок 5 – Метод *dilateCustom()*

Для реализации эрозии был создан метод *erodeCustom()*, код которой изображен на рисунке 6.

```
void closeFilter::erodeCustom()
{
    Mat imageErode(m_image.rows, m_image.cols, m_image.type());
    for (int rowsImg = 1; rowsImg < m_image.rows - 1; rowsImg++)
    {
        for (int colsImg = 1; colsImg < m_image.cols - 1; colsImg++)
        {
            int flag = 0;
            for (int rowsKern = 0; rowsKern < 3; rowsKern++)
            {
                for (int colsKern = 0; colsKern < 3; colsKern++)
                {
                    if (m_kernel.at<uchar>(rowsKern, colsKern) == 255 &&
                        m_image.at<uchar>(rowsImg - 1 + rowsKern, colsImg - 1 + colsKern) == 0)
                    {
                        flag++;
                    }
                }
            }
            if (flag)
            {
                imageErode.at<uchar>(rowsImg, colsImg) = 0;
            }
            else
            {
                imageErode.at<uchar>(rowsImg, colsImg) = m_image.at<uchar>(rowsImg, colsImg);
            }
        }
    }
    m_image = imageErode;
}
```

Рисунок 6 – Метод *erodeCustom()*

Для вычисления расширения изображения реализован метод *delBorder()*, код которого представлен на рисунке 7.

```
void closeFilter::delBorder()
{
    Mat imgWithBorder(m_image.rows - 2, m_image.cols - 2, m_image.type());

    for (int row = 1; row < m_image.rows - 1; row++)
    {
        for (int col = 1; col < m_image.cols - 1; col++)
        {
            imgWithBorder.at<uchar>(row - 1, col - 1) = m_image.at<uchar>(row, col);
        }
    }
    m_image = imgWithBorder;
}
```

Рисунок 7 – Метод *delBorder()*

Сравнение с методом из OpenCV

Для сравнения использовалось монохромное изображение разрешением 920 на 400 пикселей (см. рисунок 7).

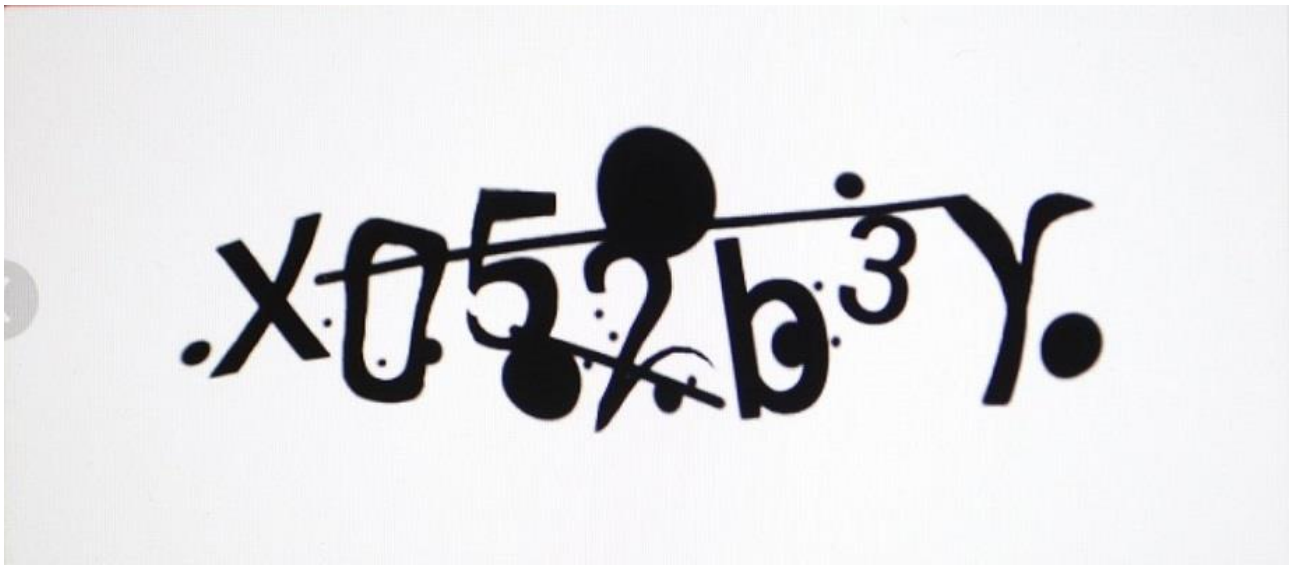

Рисунок 7 – Оригинальное изображение

Изображения, обработанные реализованным фильтром и фильтром из OpenCV, приведены ниже.

Рисунок 8 – Результат собственной реализации

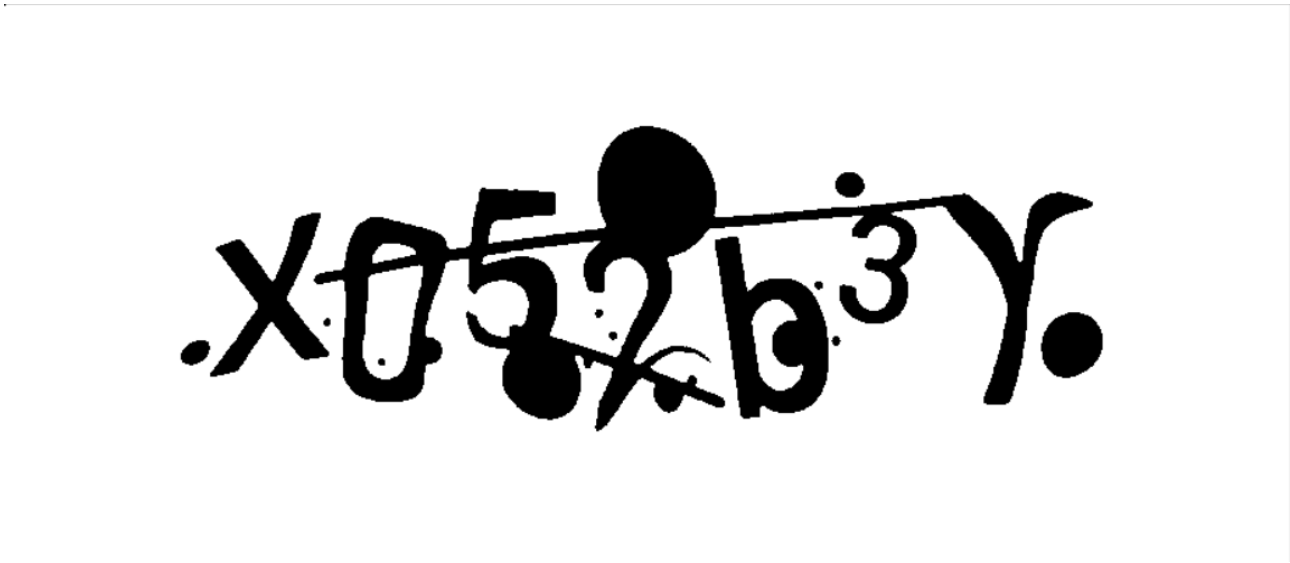

Рисунок 9 – Результат реализации OpenCV

Для сравнения изображений реализована функция *compareSimilarity()*, которая выводит в консоль количество отличающихся пикселей.

При сравнении результатов обработки функция вывела “количество различающихся пикселей”, значение которого равняется 0, т. е. результаты обработки полностью совпадают.

Для сравнения времени выполнения обработки изображения использовали библиотечную функцию *clock*(), которая выводит в консоль количество мс выполнения алгоритма в библиотеке OpenCV и собственной реализации.

Таким образом, результаты определения времени замыкающего фильтра изображения разрешением 920x400 собственной реализации и реализацией из OpenCV представлены в таблице 1.

Таблица 1 – Сравнение времени выполнения

Конфигурация решения	Собственная реализация	OpenCV
Debug	634 мс	4 мс
Release	6 мс	1 мс

Вывод

В результате выполнения лабораторной работы был реализован замыкающий фильтр. Результаты фильтра реализованным методом и методом из OpenCV полностью совпали, однако метод из библиотеки OpenCV более оптимизирован по времени выполнения.