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: Открывающий фильтр

Студент гр. 3331506/70401 Водорезов Г.И.

Преподаватель Варлашин В. В.

« »_____2020 г.

Санкт-Петербург

2020

Задание

Пользуясь средствами языка C++ и библиотеки OpenCV, реализовать открывающий фильтр с границей типа border reflect для бинарного изображения с ядром приведенном на рисунке 1.

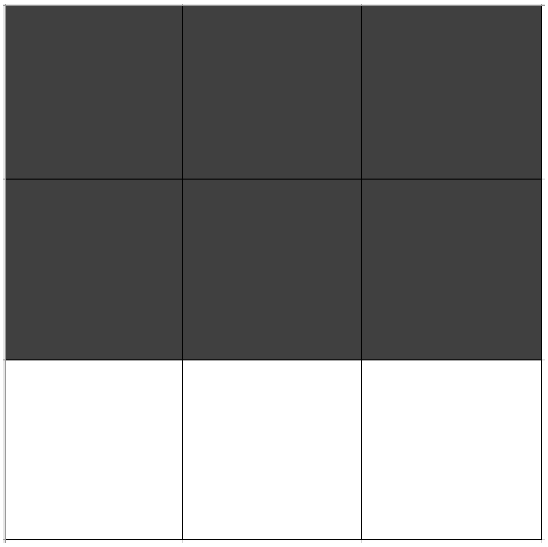

Рисунок 1 – Ядро фильтра

Ход работы

Описание алгоритма

Открывающий фильтр состоит в последовательном применении эрозии и дилатации с одинаковым структурным элементом. Его морфологический эффект заключается в удалении малых изолированных частей фигуры.

Применение эрозии сводится к проходу шаблоном по всему изображению и применению оператора поиска локального минимума к интенсивностям пикселей изображения, которые накрываются шаблоном.

Применение дилатации сводится к проходу шаблоном по всему изображению и применению оператора поиска локального максимума к интенсивностям пикселей изображения, которые накрываются шаблоном.

Реализация алгоритма

Для описания открывающего фильтра был создан класс *openingFilter*, изображенный на рисунке 2.

```
class openingFilter
{
public:
    void addBoarder();
    void deleteBoarder();
    void erodeCustom();
    void dilateCustom();
    void erodeOpencv();
    void dilateOpencv();
    void openingCustom(Mat src, Mat kernel);
    void openingOpenCV(Mat src, Mat kernel);

    void setImage(Mat src);
    void setKernel(Mat kernel);
    Mat getImage();
    Mat getKernel();
private:
    Mat m_image;
    Mat m_kernel;
};
```

Рисунок 2 – Класс *openingFilter*

Данный класс реализован для бинарных изображений, перед тем как воспользоваться методом *openingCustom()* необходимо бинаризовать изображение. Код метода *openingCustom()* представлен ниже.

```
void openingFilter::openingCustom(Mat src, Mat kernel)
{
    setKernel(kernel);
    setImage(src);
    addBoarder();
    erodeCustom();
    dilateCustom();
    deleteBoarder();
}
```

Рисунок 3 – Метод *openingCustom()*

Методы, используемые в *openingCustom()*, рассматриваются ниже.

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

```
void openingFilter::addBoarder()
{
    Mat buffer(m_image.rows+2, m_image.cols+2, m_image.type());

    for (int i = 0; i < m_image.rows; i++)
        for (int j = 0; j < m_image.cols; j++)
            buffer.at<uchar>(i + 1, j + 1) = m_image.at<uchar>(i, j);

    for (int i = 0; i < m_image.rows; i++)
    {
        buffer.at<uchar>(i + 1, 0) = m_image.at<uchar>(i, 0);
        buffer.at<uchar>(i + 1, buffer.rows - 1) = m_image.at<uchar>(i, m_image.rows - 1);
    }

    for (int j = 0; j < m_image.cols; j++)
    {
        buffer.at<uchar>(0, j + 1) = m_image.at<uchar>(0, j);
        buffer.at<uchar>(buffer.cols - 1, j + 1) = m_image.at<uchar>(m_image.cols - 1, j);
    }

    m_image = buffer;
}
```

Рисунок 4 – Метод *addBoarder()*

Для осуществления эрозии был реализован метод *erodeCustom()*, код которой изображен на рисунке 5.

```
void openingFilter::erodeCustom()
{
    Mat imageFiltered(m_image.rows, m_image.cols, m_image.type());

    for (int i = 1; i < m_image.rows - 1; i++)
        for (int j = 1; j < m_image.cols - 1; j++)
        {
            int flag = 0;
            for (int m = 0; m < 3; m++)
                for (int n = 0; n < 3; n++)
                    if (m_kernel.at<uchar>(m, n) == 255 && m_image.at<uchar>(i - 1 + m, j - 1 + n) == 0)
                        flag++;
            if (flag) imageFiltered.at<uchar>(i, j) = 0;
            else imageFiltered.at<uchar>(i, j) = m_image.at<uchar>(i, j);
        }
    m_image = imageFiltered;
}
```

Рисунок 5 – Метод *erodeCustom()*

Для реализации дилатации изображения был создан метод *dilatCustom()*, код которого изображен на рисунке 6.

```
void openingFilter::dilateCustom()
{
    Mat imageFiltered(m_image.rows, m_image.cols, m_image.type());

    for (int i = 1; i < m_image.rows - 1; i++)
        for (int j = 1; j < m_image.cols - 1; j++)
        {
            int flag = 0;
            for (int m = 0; m < 3; m++)
                for (int n = 0; n < 3; n++)
                    if (m_kernel.at<uchar>(m, n) == 255 && m_image.at<uchar>(i - 1 + m, j - 1 + n) == 255)
                        flag++;
            if (flag) imageFiltered.at<uchar>(i, j) = 255;
            else imageFiltered.at<uchar>(i, j) = m_image.at<uchar>(i, j);
        }
    m_image = imageFiltered;
}
```

Рисунок 6 – Метод *dilateCustom()*

Сравнение с методом из OpenCV

Для обработки использовалось классическое изображение Лены Седерберг разрешением 800 на 800 пикселей (см. рисунок 7).

Рисунок 7 – Оригинальное изображение

Изображения, обработанные самописным фильтром и фильтром из OpenCV, приведены ниже на рисунках 8 и 9.

Рисунок 8 – Результат собственной реализации

Рисунок 9 – Результат реализации OpenCV

Для точного сравнения изображений реализован вспомогательная функция *comapreSimilarity()*. Функция выводит в консоль количество отличающихся пикселей. При сравнении результатов обработки данная функция вывела число 0, что соответствует полностью совпадающим изображениям.

Для сравнения времени выполнения обработки изображения использовался функцией *compareTime()*, которая выводит в консоль количество миллисекунд выполнения алгоритма в библиотеке OpenCV и собственной реализации.

Результаты определения времени открывающего фильтра изображения разрешением 800*800 собственной реализации и реализацией из OpenCV представлены в таблице 1.

Таблица 1 – Сравнение времени выполнения

Конфигурация решения	Собственная реализация	OpenCV
Debug	3265 мс	6 мс
Release	36 мс	1 мс

Различное время выполнения в различных конфигурациях связано с тем, что в конфигурации Release включена максимальная оптимизация, а в Debug оптимизация отключена полностью.

Вывод

В результате выполнения лабораторной работы был реализован открывающий фильтр. Результаты фильтра реализованным методом и методом из OpenCV полностью совпали, однако метод из OpenCV оптимальнее по времени выполнения.