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: Фильтрация изображений

Студент гр. 3331506/70401 Паньков И.С.
Преподаватель Варлашин В.В.
« »_____2020 г.

Санкт-Петербург
2020

Цель работы — программная реализация фильтров изображений.

З а д а н и е

Разработать класс, предоставляющий возможности применения к бинарному изображению открывающего фильтра со структурообразующим элементом, представленным на рисунке 1, и заполнением граничных областей по методу border reflect.

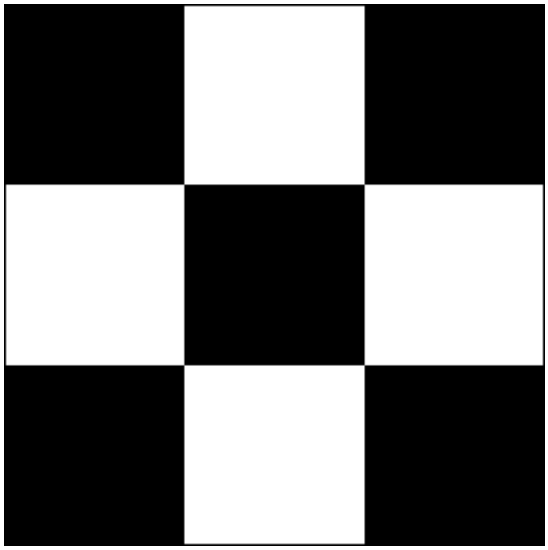

Рисунок 1 — Структурообразующий элемент

Краткие теоретические сведения

Размыкание (открытие) — производная морфологическая операция, представляющая собой последовательное применение двух других операций — эрозии и дилатации (наращивания). Размыкание может быть записано в следующем виде:

$$A \circ B = \left(A \ominus B \right) \oplus B,$$

где A и B — некоторые множества, а применимо к обработке цифровых изображений — изображение и структурообразующий элемент.

Операция эрозии полезна для удаления малых объектов и различных шумов, но у неё есть недостаток: все остающиеся объекты уменьшаются в размере. Этого эффекта можно избежать, если после операции эрозии применить операцию наращивания с тем же структурообразующим элементом. Размыкание отсеивает все объекты, меньшие чем структурный элемент, но при этом позволяет избежать сильного уменьшения размера объектов.

Класс MorphologicalFilter

Для выполнения морфологических операций над изображениями был разработан класс `MorphologicalFilter`, предоставляющий возможности для бинаризации исходных изображений и структурообразующих элементов, методы-сеттеры и методы-геттеры для доступа к фильтруемому изображению, структурообразующему элементу и его якорной точке, а также методы для выполнения элементарных (эрозии и наращивания) и производных (замыкания и размыкания) морфологических операций.

На рисунке 2 представлен листинг класса `MorphologicalFilter`.

```
class MorphologicalFilter
{
public:
    MorphologicalFilter() = default;
    ~MorphologicalFilter() = default;

    void erode();
    void dilate();
    void open();
    void close();

    void setStructuringElement(const cv::Mat structuringElement);
    cv::Mat getStructuringElement() const;

    void setAnchor(const cv::Point2i anchor);
    cv::Point2i getAnchor() const;

    void setImage(const cv::Mat structure);
    cv::Mat getImage() const;

private:
    void fillBorders(cv::Mat& buffer);
    bool compare(const cv::Mat& box, MorphologicalOperation operation);
    cv::Mat binarize(const cv::Mat& image) const;
    cv::Mat normalize(const cv::Mat& image) const;

    cv::Mat m_structuringElement;
    cv::Point2i m_anchor;
    cv::Mat m_image;
    std::vector<int> m_border;
};
```

Рисунок 2 — Листинг класса `MorphologicalFilter`

Морфологическая операция эрозии выполняется с помощью метода `erode`, листинг которого представлен на рисунке 3.

```
void MorphologicalFilter::erode()
{
    if (m_image.empty() == true)
    {
        return;
    }

    if (m_structuringElement.empty() == true)
    {
        return;
    }

    if (m_anchor.x < 0 || m_anchor.x >= m_structuringElement.cols)
    {
        return;
    }

    if (m_anchor.y < 0 || m_anchor.y >= m_structuringElement.rows)
    {
        return;
    }

    auto cols = m_image.cols + m_border[LEFT] + m_border[RIGHT];
    auto rows = m_image.rows + m_border[TOP] + m_border[BOTTOM];

    auto buffer = Mat(rows, cols, CV_8UC1);
    m_image.copyTo(buffer(Rect(m_anchor.x, m_anchor.y, m_image.cols, m_image.rows)));

    fillBorders(buffer);

    for (auto col = 0; col < m_image.cols; col++)
    {
        for (auto row = 0; row < m_image.rows; row++)
        {
            auto box = Mat(buffer, Rect(col, row,
                           m_structuringElement.cols, m_structuringElement.rows));
            if (compare(box, MorphologicalOperation::EROSION) == true)
            {
                m_image.at<uint8_t>(row, col) = 1;
            }
            else
            {
                m_image.at<uint8_t>(row, col) = 0;
            }
        }
    }
}
```

Рисунок 3 — Листинг метода `erode`

Морфологическая операция наращивания выполняется с помощью метода `dilate`, листинг которого представлен на рисунке 4.

```
void MorphologicalFilter::dilate()
{
    if (m_image.empty() == true)
    {
        return;
    }

    if (m_structuringElement.empty() == true)
    {
        return;
    }

    if (m_anchor.x < 0 || m_anchor.x >= m_structuringElement.cols)
    {
        return;
    }

    if (m_anchor.y < 0 || m_anchor.y >= m_structuringElement.rows)
    {
        return;
    }

    auto cols = m_image.cols + m_border[LEFT] + m_border[RIGHT];
    auto rows = m_image.rows + m_border[TOP] + m_border[BOTTOM];

    auto buffer = Mat(rows, cols, CV_8UC1);
    m_image.copyTo(buffer(Rect(m_anchor.x, m_anchor.y, m_image.cols, m_image.rows)));

    fillBorders(buffer);

    for (auto col = 0; col < m_image.cols; col++)
    {
        for (auto row = 0; row < m_image.rows; row++)
        {
            auto box = Mat(buffer, Rect(col, row,
                           m_structuringElement.cols, m_structuringElement.rows));
            if (compare(box, MorphologicalOperation::DILATION) == true)
            {
                m_image.at<uint8_t>(row, col) = 1;
            }
            else
            {
                m_image.at<uint8_t>(row, col) = 0;
            }
        }
    }
}
```

Рисунок 4 — Листинг метода `dilate`

Тестирование работы программы

Тестирование работы алгоритма открывающего фильтра проводилось с помощью программы, листинг которой изображён на рисунке 5.

```
#include "morphological_filter.h"

#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"

using namespace std;
using namespace cv;

int main()
{
    auto filter = MorphologicalFilter();
    auto image = imread("src/images/test.bmp");
    auto structuringElement = imread("src/structuring_elements/my_filter.bmp");
    imshow("Image", image);

    filter.setImage(image);
    filter.setStructuringElement(structuringElement);
    filter.open();

    auto myResult = filter.getImage();
    imshow("My result", myResult);

    auto kernel = Mat();
    auto src = Mat();
    auto openCVResult = Mat();

    cvtColor(image, src, COLOR_BGR2GRAY);
    cvtColor(structuringElement, kernel, COLOR_BGR2GRAY);
    threshold(src, src, 0x7F, 0xFF, THRESH_BINARY);
    threshold(kernel, kernel, 0x7F, 1, THRESH_BINARY_INV);
    morphologyEx(src, openCVResult, MORPH_OPEN, kernel, Point(-1, -1), 1, BORDER_REFLECT);
    imshow("OpenCV Result", openCVResult);

    auto diff = Mat();
    absdiff(myResult, openCVResult, diff);
    absdiff(diff, Mat(diff.rows, diff.cols, diff.type(), Scalar(0xFF)), diff);
    imshow("Difference", diff);

    while (waitKey() != 27);

    return 0;
}
```

Рисунок 5 — Листинг программы тестирования алгоритма

На рисунке 6 представлены соответственно исходное изображение (а), результат действия алгоритма открывающего фильтра (б), результат действия встроенных функций OpenCV (в) и абсолютная разность двух предыдущих изображений, вычтенная из изображения с максимальной интенсивностью пикселей (г).

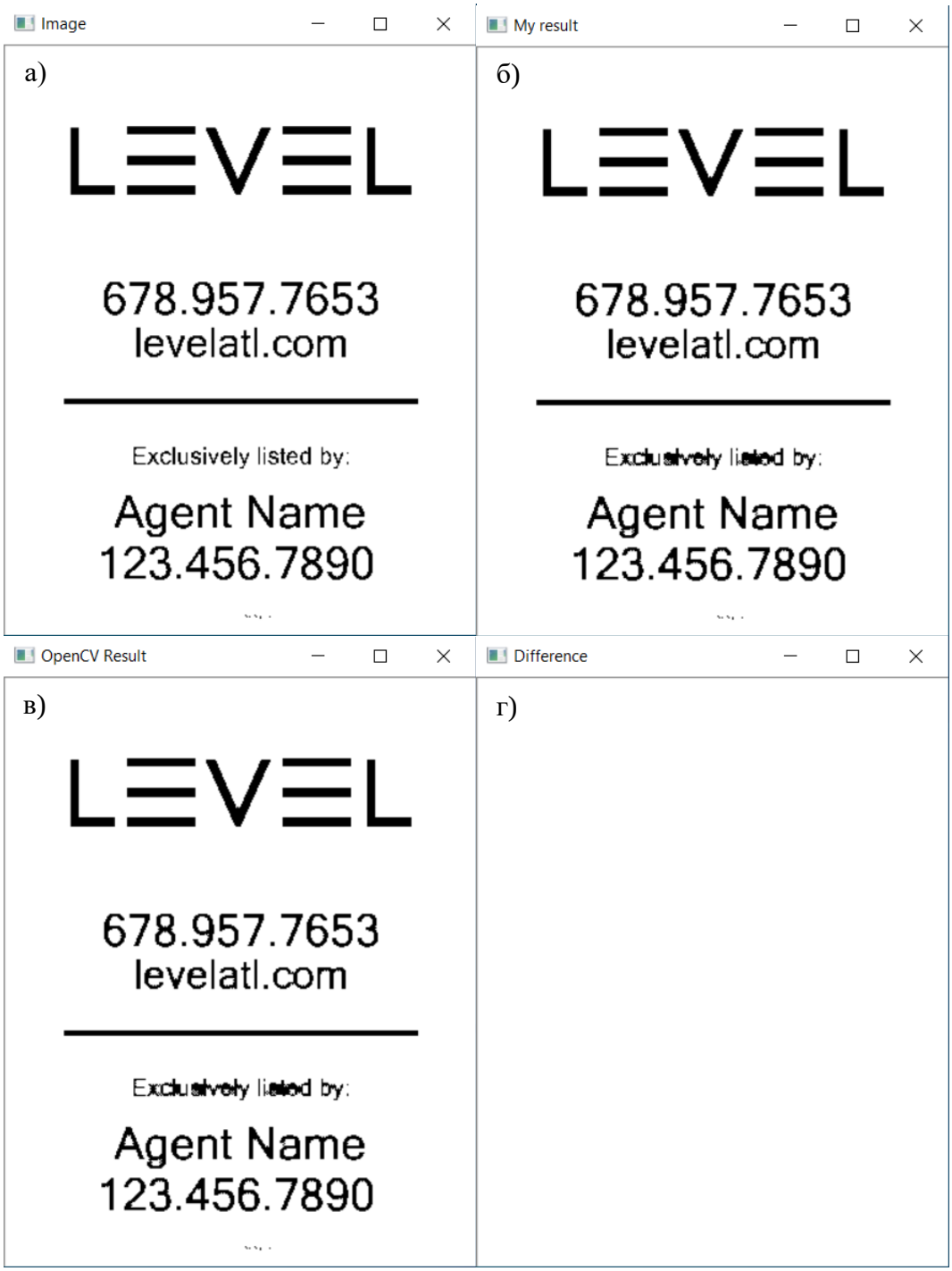

Рисунок 6 — Изображения: а) — исходное, б) — обработанное алгоритмом открывающего фильтра, в) — обработанное алгоритмами встроенных функций OpenCV, г) — абсолютная разность б) и в)

Нетрудно видеть, что результаты работы алгоритмов совпадают, однако значительно различается время их работы: если в случае со встроенными функциями OpenCV время их работы составляет величины порядка единиц миллисекунд, то в случае с реализованным алгоритмом — десятки миллисекунд в релизной сборке и сотни миллисекунд в отладочной сборке программы.