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: Разработка морфологического замыкающего фильтра с заданным ядром

Студент гр. 3331506/70401 Якименко Г.К.

Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург

2020

Задача

Задание к лабораторной работе представлено на Рисунке 1.

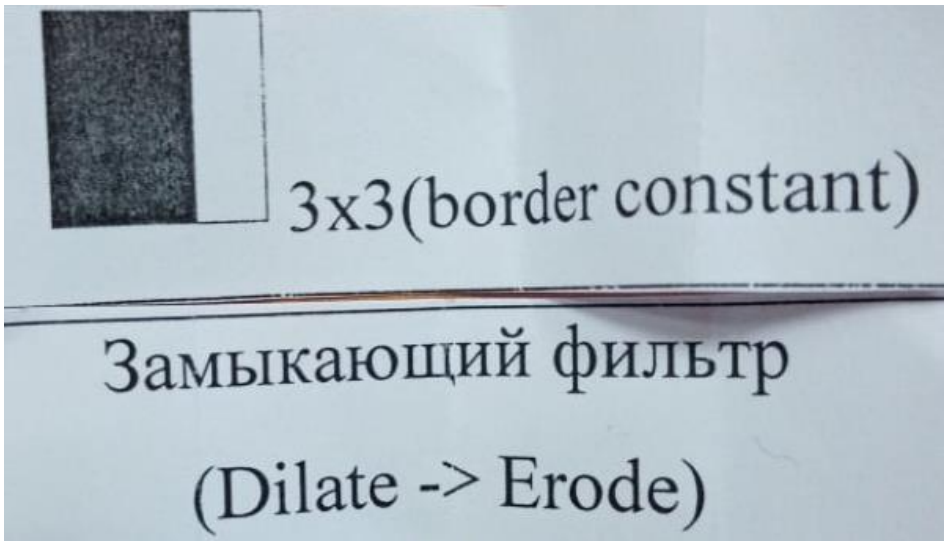

Рисунок 1 – задание к лабораторной работе

Описание алгоритма

1. Ниже представлен класс MyFilter, реализующий замыкающий фильтер:

```
class MyFilter
{
private:
    cv::Mat m_defaultImage;
    cv::Mat m_grayImage;
    cv::Mat m_binImage;
    cv::Mat m_dilatedImage;
    cv::Mat m_erodedImage;
    cv::Mat m_difrenceImg;
    cv::Mat m_result;

    cv::Mat m_CVdilatedImage;
    cv::Mat m_CVerodedImage;
    cv::Mat m_CVresult;

    bool m_kernel[3][3];

    int m_cols;
    int m_rows;

public:
```

```
    MyFilter();
    MyFilter(cv::Mat image, bool kernel[3][3]);

    void dilate();
    void erode();
    void fillBorder();
    void closeFilter();

    void CVdilate();
    void CVerode();
    void CVcloseFilter();

    void showDefaultImg();
    void showGrayImg();
    void showBinImg();
    void showDilatedImg();
    void showErodedImg();
    void showDifrence();

    void showCVdilatedImg();
    void showCVerodedImg();

    void showResult();
    void showCVresult();
};
```

2. Конструктор класса MyFilter() принимает изображение, над которым мы собираемся проводить манипуляции. Так как изображение может оказаться цветным, конструктор подвергает изображение обработке, переводя его из трехканального изображения в одноканальное в оттенках серого. И завершающим этапом предварительной обработки является пороговая бинаризация изображения в оттенках серого. Код конструктора приведен ниже:

```
MyFilter::MyFilter(cv::Mat image, bool kernel[3][3])
{
    //Передача изображения классу
    m_defaultImage = image.clone();

    //Передача ядра классу
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            m_kernel[i][j] = kernel[i][j];

    //Размеры изображения
    m_rows = m_defaultImage.rows;
    m_cols = m_defaultImage.cols;
```

```
    //Изображение в оттенках серого
    cvtColor(m_defaultImage, m_grayImage, COLOR_RGB2GRAY);

    //Бинаризованное изображение
    cvtColor(m_defaultImage, m_binImage, COLOR_RGB2GRAY);
    for (int i = 0; i < m_rows; i++)
    {
        for (int j = 0; j < m_cols; j++)
        {
            if (m_binImage.at<uint8_t>(i, j) < 125) {
                m_binImage.at<uint8_t>(i, j) = 0;
            }
            else
            {
                m_binImage.at<uint8_t>(i, j) = 255;
            }
        }
    }

    m_dilatedImage = m_binImage.clone();
    m_erodedImage = m_binImage.clone();
    m_difrenceImg = m_binImage.clone();
    m_result = m_binImage.clone();
    m_CVresult = m_binImage.clone();
}
```

3. Следующий шаг – дилатация бинаризованного изображения. Код функции дилатации представлен ниже:

```
void MyFilter::dilate()
{
    //Цикл для прохода по пикселям изображения
    for (int i_image = 0; i_image < m_rows - 2; i_image++)
    {
        for (int j_image = 0; j_image < m_cols - 2; j_image++)
        {
            //Цикл для прохода по ядру
            for (int i_kernel = 0; i_kernel < 3; i_kernel++)
            {
                for (int j_kernel = 0; j_kernel < 3; j_kernel++)
                {
                    if (m_kernel[i_kernel][j_kernel] &&
m_binImage.at<uint8_t>(i_image + i_kernel, j_image + j_kernel) == 0)
                        m_dilatedImage.at<uint8_t>(i_image + 1,
j_image + 1) = 0;
                }
            }
        }
    }
}
```

4. Далее - эрозия изображения после дилатации. Код функции эрозии представлен ниже:

```
void MyFilter::erode()
{
    m_erodedImage = m_dilatedImage.clone();
    //Цикл для прохода по пикселям изображения
    for (int i_image = 0; i_image < m_rows - 2; i_image++)
    {
        for (int j_image = 0; j_image < m_cols - 2; j_image++)
        {
            //Цикл для прохода по ядру
            for (int i_kernel = 0; i_kernel < 3; i_kernel++)
            {
                for (int j_kernel = 0; j_kernel < 3; j_kernel++)
                {
                    if (m_kernel[i_kernel][j_kernel] &&
m_dilatedImage.at<uint8_t>(i_image + i_kernel, j_image + j_kernel) ==
255)
                        m_erodedImage.at<uint8_t>(i_image + 1, j_image
+ 1) = 255;
                }
            }
        }
    }

    m_result = m_erodedImage.clone();
}
```

5. Заключительный шаг – заливка границ изображения:

```
void MyFilter::fillBorder()
{
    for (int i = 0, j = 0; j < m_cols; j++) m_result.at<uint8_t>(i, j)
= 255;
    for (int i = 0, j = 0; i < m_rows; i++) m_result.at<uint8_t>(i, j)
= 255;
    for (int i = m_rows - 1, j = 0; j < m_cols; j++)
m_result.at<uint8_t>(i, j) = 255;
    for (int i = 0, j = m_cols - 1; i < m_rows; i++)
m_result.at<uint8_t>(i, j) = 255;
}
```

Результат работы алгоритма

На рисунке 1 представлен оригинал изображения с помехами.

Рисунок 1 – оригинал изображения

На рисунке 2 представлено изображение после фильтрации.

Рисунок 2 – изображение после фильтрации

Далее убедимся в корректности работы алгоритма, сравнив результат фильтрации нашим алгоритмом с результатом фильтрации функции из библиотеки OpenCV.

Код функции сравнения:

```
void MyFilter::showDifrence()
{
    for (int i = 0; i < m_rows; i++)
    {
        for (int j = 0; j < m_cols; j++)
        {
            if (m_result.at<uint8_t>(i, j) ==
m_CVresult.at<uint8_t>(i, j)) m_difrenceImg.at<uint8_t>(i, j) = 255;
            else m_difrenceImg.at<uint8_t>(i, j) = 0;
        }
    }
    imshow("Difrence image", m_difrenceImg);
    while (waitKey(1) != 27);
}
```

Результат работы функции сравнения представлен на рисунке 3.

Рисунок 3 – результат сравнения

Как мы видим, изображения идентичны.