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: Морфологические операции. Открывающий фильтр.

Студент гр. 3331506/70401 Соколов Д. А.

Преподаватель Варлашин В. В.

« »_____2020 г.

Санкт-Петербург

2020

Задание

Пользуясь средствами языка C++ и библиотеки OpenCV, реализовать применение открывающего фильтра на бинарном изображении. Ядро:

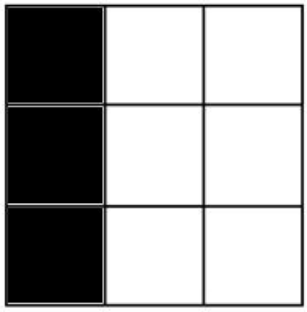

Рисунок 1 — Ядро фильтра

Якорь находится в центре ядра.

Ход работы

Реализация алгоритма

Морфологические операции традиционно применяются именно на бинарных изображениях. Следовательно, перед обработкой был применен библиотечный метод бинаризации изображения.

Для подготовки изображения к применению фильтра необходимо добавить к нему рамку. По заданию, рамка состоит из нулевых интенсивностей. Для этого была написана следующая функция (на рис. 2).

```
void addBorder(cv::Mat& image, size_t borderSize)
{
    if (image.empty())
    {
        std::cerr << "Matrix is empty!" << std::endl;
        return;
    }
    cv::Mat col = cv::Mat::zeros(image.rows, 1, image.type());
    cv::Mat row = cv::Mat::zeros(1, image.cols, image.type());
    for (size_t i = 0; i < borderSize; i++)
    {
        row = cv::Mat::zeros(1, image.cols, image.type());
        cv::vconcat(image, row, image);
        row = cv::Mat::zeros(1, image.cols, image.type());
        cv::vconcat(row, image, image);
        col = cv::Mat::zeros(image.rows, 1, image.type());
        cv::hconcat(image, col, image);
        col = cv::Mat::zeros(image.rows, 1, image.type());
        cv::hconcat(col, image, image);
    }
}
```

Рисунок 2 – Реализация метода добавления рамок к изображению

После подготовки изображения можно приступить к фильтрации. Задачу можно разделить на две части: эрозия и дилатация. Далее будет рассмотрена реализация одного из них, так как различия невелики.

Для доступа к ограниченной части исходного изображения, используется объект под названием “область интереса” (ОИ). ОИ представляет из себя матрицу такого же размера, как ядро. Во время работы алгоритма происходит смещение ОИ по строкам и столбцам изображения, таким образом обрабатываются все пиксели.

Пример кода показан на рисунке 3.

```
void erode (cv::Mat& srcImage, const cv::Mat kernel)
{
    cv::Mat copyImage = srcImage.clone();

    cv::Rect rp(0, 0, kernel.rows, kernel.cols);
    cv::Mat roi = copyImage(rp);
    cv::Mat roiOut = srcImage(rp);

    for (size_t rows = 0; rows <= copyImage.rows - kernel.rows; rows++)
    {
        for (size_t cols = 0; cols <= copyImage.cols - kernel.cols; cols++)
        {
            rp.x = cols;
            rp.y = rows;
            roi = copyImage(rp);
            roiOut = srcImage(rp);
            roiOut.at<uchar>(roi.rows/2, roi.cols/2) = erode_add(kernel, roi);
        }
    }
}
```

Рисунок 3 - Реализация функции эрозии

Расчет интенсивности текущего пикселя проводится по следующей схеме: если хотя бы один белый пиксель ядра наложится на фон (черный пиксель) ОИ, то текущий пиксель заполнится черным цветом (в случае эрозии), в противном случае, значение интенсивности текущего пикселя не изменится. Для дилатации действия в точности те же, только при совпадении с белым пикселем ОИ, текущий пиксель будет закрашен белым. Пример реализации на рисунке 4.

```
uchar erode_add(const cv::Mat& kernel, cv::Mat& roi)
{
    for (size_t cols = 0; cols < kernel.cols; cols++)                    ⚠ compar
    {
        for (size_t rows = 0; rows < kernel.rows; rows++)                ⚠ compar
        {
            if (((kernel.at<uchar>(rows, cols)) == 255) && ((roi.at<uchar>(rows, cols)) == 0))
            {
                return uchar(0);
            }
        }
    }
    return roi.at<uchar>(roi.rows/2, roi.cols/2);
}
```

Рисунок 4 - Реализация вспомогательной функции для эрозии

Сравнение с методом из OpenCV

Таблица 1 – Сравнение времени выполнения

Конфигурация решения	Ручная	OpenCV
Debug	6.312 мс	0.415 мс
Release	2.324 мс	0.375 мс

Время выполнения обработки реализованным методом в конфигурации Release меньше, чем в Debug в 2.7 раза. Такую разницу можно объяснить присутствием оптимизации по времени в сборке Release.

Пример работы фильтра показан на рисунке 5.

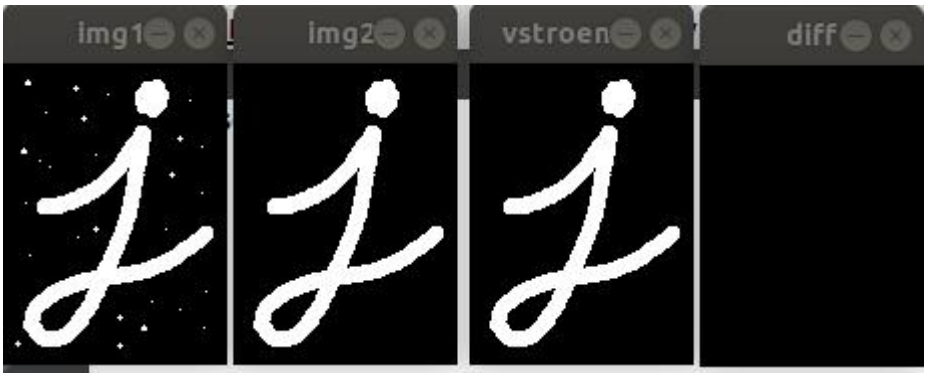

Рисунок 5 - Демонстрация работы фильтра