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: Медианный фильтр

Студент гр. 3331506/70401 Куликов М.М.

Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург

2020

Задание

С помощью средств *OpenCV* реализовать медианный фильтр с размером ядра 3×3, якорем, расположенным в левом нижнем углу и границей — константа.

Ход работы

Описание

Медианный фильтр используется для уменьшения уровня шума «соли и перца» на изображениях, а также для удаления царапин. Смысл фильтра заключается в прохождении по всем пикселям, расположенным в ядре, упорядочивании их, и выборе медианы из этого массива. Значение медианы записывается в якорь, затем ядро смещается дальше на один пиксель.

Класс *MedianFilter*

Для реализации медианного фильтра был создан класс *MedianFilter*. Данный класс представлен на рисунке 1.

```
10  class MedianFilter
11  {
12  public:
13      MedianFilter();
14      MedianFilter(Mat src);
15
16      ~MedianFilter();
17
18      //Рукописный фильтр (ядро 3x3, якорь в левом нижнем углу)
19      void filteringWithMyAnchor();
20
21      //Рукописный фильтр с ядром 3x3 и якорем по центру (для сравнения изображений)
22      void filteringWithMiddleAnchor();
23
24      //Стандартный фильтр OpenCV
25      void filteringWithOpenCV();
26
27      //Разница между стандартным и рукописным фильтром (в пикселях)
28      void showDifference(Mat src1, Mat src2);
29
30      //Геттеры
31      Mat getImageHW();
32      Mat getImageCV();
33
34  private:
35      Mat m_srcImage;
36      Mat m_dstImageHW;
37      Mat m_dstImageCV;
38      int m_window[9];
39  };
```

Рисунок 1 — Класс *MedianFilter*

При создании объекта класса ему передается исходное изображение (рисунок 3) с флагом IMREAD_GRAYSCALE, которое преобразуется в одноканальное, в оттенках серого. Так же создаются два выходных изображения, которые копируют размер и тип исходного (рисунок 2).

```
MedianFilter::MedianFilter(Mat src):
    m_srcImage(src),
    m_dstImageCV(src.rows, src.cols, src.type()),
    m_dstImageHW(src.rows, src.cols, src.type())
{
    ;
}
```

Рисунок 2 — Конструктор класса *MedianFilter*

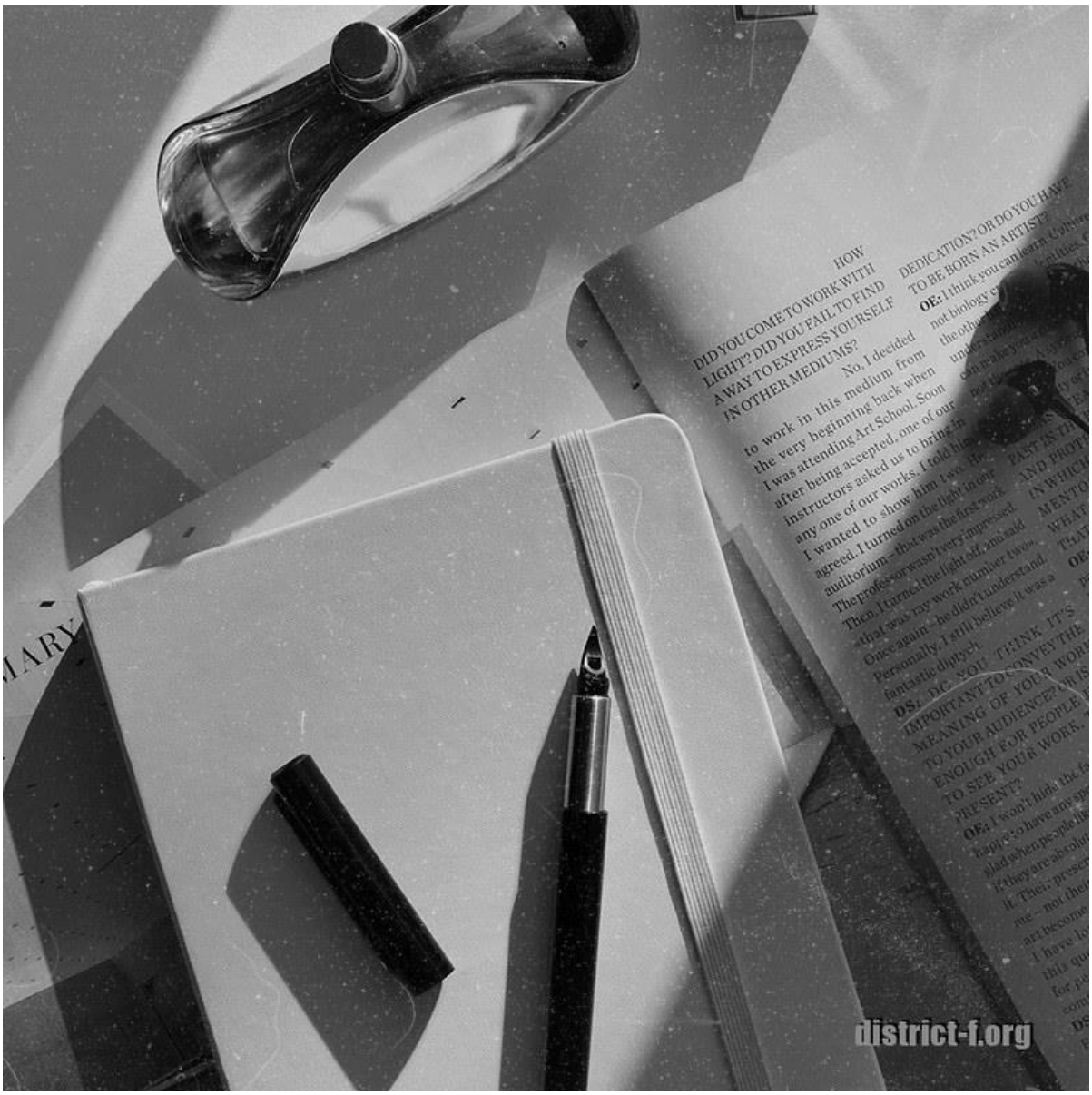

Рисунок 3 — Исходное изображение

Основные методы

1) Метод *filteringWithMyAnchor( )*

Реализация метода представлена на рисунке 4.

```
//Рукописный фильтр (ядро 3x3, якорь в левом нижнем углу)
void MedianFilter::filteringWithMyAnchor()
{
    for (int image_rows = 2; image_rows < m_srcImage.rows; image_rows++)
    {
        for (int image_cols = 0; image_cols < m_srcImage.cols - 2; image_cols++)
        {
            int window[9] = { 0 };

            window[0] = m_srcImage.at<uchar>(image_rows - 2, image_cols);
            window[1] = m_srcImage.at<uchar>(image_rows - 2, image_cols + 1);
            window[2] = m_srcImage.at<uchar>(image_rows - 2, image_cols + 2);
            window[3] = m_srcImage.at<uchar>(image_rows - 1, image_cols);
            window[4] = m_srcImage.at<uchar>(image_rows - 1, image_cols + 1);
            window[5] = m_srcImage.at<uchar>(image_rows - 1, image_cols + 2);
            window[6] = m_srcImage.at<uchar>(image_rows, image_cols);
            window[7] = m_srcImage.at<uchar>(image_rows, image_cols + 1);
            window[8] = m_srcImage.at<uchar>(image_rows, image_cols + 2);

            std::sort(std::begin(window), std::end(window));
            m_dstImageHW.at<uchar>(image_rows, image_cols) = window[4];
        }
    }
}
```

Рисунок 4 — Метод *filteringWithMyAnchor( )*

Данный метод осуществляет медианную фильтрацию изображения. Он имеет ядро размером 3×3 и якорь в левом нижнем углу. Все пиксели, расположенные в ядре, последовательно записываются в массив, который затем сортируется по возрастанию. Медиана данного массива записывается на место якорной точки. Так как якорь в левом нижнем углу, то происходит смещение выходного изображения влево и вниз (рисунок 5). Справа и сверху появляется граница, которая остается черной.

Выходное изображение после использования данного метода представлено на рисунке 5.

Рисунок 5 — Выходное изображение метода *filteringWithMyAnchor( )*

2) Метод *filteringWithMiddleAnchor( )*

Реализация метода представлена на рисунке 6.

```
//Рукописный фильтр с ядром 3x3 и якорем по центру (для сравнения изображений)
void MedianFilter::filteringWithMiddleAnchor()
{
    for (int image_rows = 1; image_rows < m_srcImage.rows - 1; image_rows++)
    {
        for (int image_cols = 1; image_cols < m_srcImage.cols - 1; image_cols++)
        {
            int window[9] = { 0 };

            window[0] = m_srcImage.at<uchar>(image_rows - 1, image_cols - 1);
            window[1] = m_srcImage.at<uchar>(image_rows - 1, image_cols);
            window[2] = m_srcImage.at<uchar>(image_rows - 1, image_cols + 1);
            window[3] = m_srcImage.at<uchar>(image_rows, image_cols - 1);
            window[4] = m_srcImage.at<uchar>(image_rows, image_cols);
            window[5] = m_srcImage.at<uchar>(image_rows, image_cols + 1);
            window[6] = m_srcImage.at<uchar>(image_rows + 1, image_cols - 1);
            window[7] = m_srcImage.at<uchar>(image_rows + 1, image_cols);
            window[8] = m_srcImage.at<uchar>(image_rows + 1, image_cols + 1);

            std::sort(std::begin(window), std::end(window));
            m_dstImageHW.at<uchar>(image_rows, image_cols) = window[4];
        }
    }
}
```

Рисунок 6 — Метод *filteringWithMiddleAnchor( )*

Данный метод отличается от предыдущего лишь расположением якорной точки, она находится в центре. Так как якорь по середине, то изображение обрезается по краям на толщину одного пикселя. По заданию граница должна быть константой, поэтому изображение окружено черной рамкой.

Метод *filteringWithMiddleAnchor( )* является дополнительным и необходим лишь для того, что сравнить выходные изображения рукописного и библиотечного фильтра, так как в стандартном методе *medianBlur( )* нет возможности изменить расположение якорной точки.

Выходное изображение после использования данного метода представлено на рисунке 7.

Рисунок 7 — Выходное изображение метода *filteringWithMiddleAnchor( )*

3) Метод *filteringWithOpenCV ( )*

Реализация метода представлена на рисунке 8.

```
//Стандартный фильтр OpenCV
void MedianFilter::filteringWithOpenCV()
{
    cv::medianBlur(m_srcImage, m_dstImageCV, 3);
}
```

Рисунок 8 — Метод *filteringWithOpenCV( )*

Данный метод является библиотечным. Он производит медианную фильтрацию с размером ядра 3×3, якорем в центре и границей BORDER_REPLICATE. Границу и якорную точку нельзя поменять в данном методе.

Выходное изображение после использования данного метода представлено на рисунке 9.

Рисунок 9 — Выходное изображение метода *filteringWithOpenCV( )*

4) Метод *showDifference( )*

Реализация метода представлена на рисунке 10.

```
//Разница между стандартным и рукописным фильтром (в пикселях)
void MedianFilter::showDifference(Mat src1, Mat src2)
{
    Mat difference(src1.rows, src1.cols, src1.type());
    cv::absdiff(src1, src2, difference);

    int res = 0;
    for (int i = 1; i < difference.rows - 1; i++)
        for (int j = 1; j < difference.cols - 1; j++)
        {
            if (difference.at<uchar>(i, j) != 0) res++;
        }

    imshow("Difference", difference);
    cout << "Different pixels: " << res << endl;
}
```

Рисунок 10 — Метод *showDifference( )*

Данный метод необходим для сравнения двух реализаций медианного фильтра: рукописной и библиотечной. Метод *showDifference( )* использует стандартный метод *absdiff( )*, который попиксельно вычитает из одного изображения другое. Затем суммируются пиксели, разность которых отлична от нуля, и выводятся в консоль. Метод написан с учетом константной границы со всех сторон толщиной в один пиксель. На вход данного метода подается выходное изображение метода *filteringWithMiddleAnchor( )*, так как якорная точка в данном методе расположена посередине как и в *filteringWithOpenCV( )*.

Выходное изображение после использования данного метода представлено на рисунке 11.

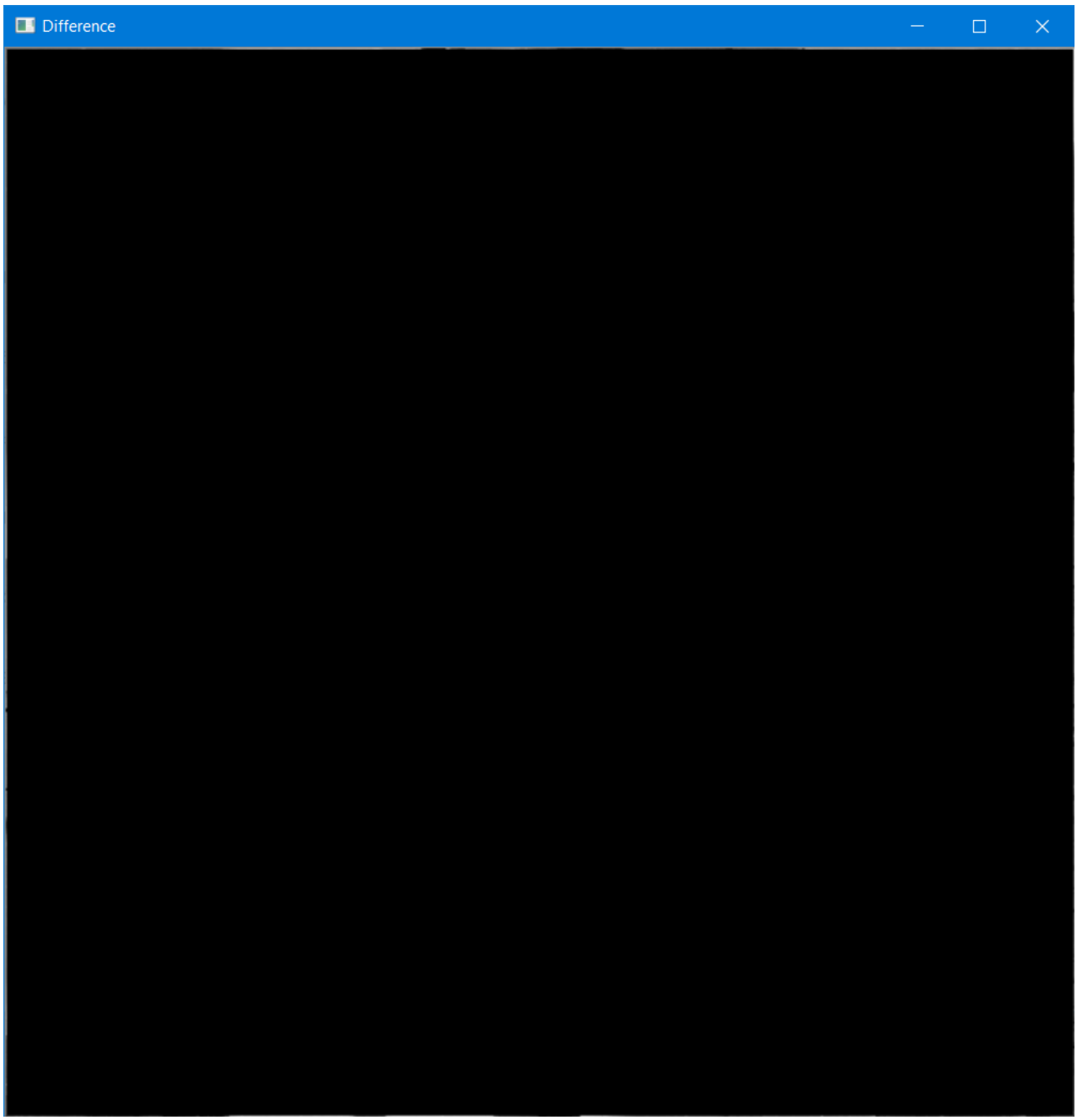

Рисунок 11 — Выходное изображение метода *showDifference( )*

Вывод консоли представлен на рисунке 12.

Рисунок 12 — Вывод консоли метода *showDifference( )*

Класс *Timer*

Для подсчета времени выполнения разных реализаций медианного фильтра был создан класс *Timer*. Для его реализации была использована библиотека *chrono*. Данный класс подсчитывает время в секундах и выводит результат в консоль. Он представлен на рисунке 13.

```
10  class MedianFilter
11  {
12  public:
13      MedianFilter();
14      MedianFilter(Mat src);
15
16      ~MedianFilter();
17
18      //Рукописный фильтр (ядро 3x3, якорь в левом нижнем углу)
19      void filteringWithMyAnchor();
20
21      //Рукописный фильтр с ядром 3x3 и якорем по центру (для сравнения изображений)
22      void filteringWithMiddleAnchor();
23
24      //Стандартный фильтр OpenCV
25      void filteringWithOpenCV();
26
27      //Разница между стандартным и рукописным фильтром (в пикселях)
28      void showDifference(Mat src1, Mat src2);
29
30      //Геттеры
31      Mat getImageHW();
32      Mat getImageCV();
33
34  private:
35      Mat m_srcImage;
36      Mat m_dstImageHW;
37      Mat m_dstImageCV;
38      int m_window[9];
39  };
```

Рисунок 13 — Класс *Timer*

Итоги

При сравнении изображений, метод *showDifference( )* выводил в консоль количество отличающихся пикселей равное 0. Это означает, что изображения полностью совпадают. Результаты времени выполнения реализаций представлены в таблице 1.

Таблица 1 — Время выполнения реализаций

	Время выполнения, мс (конфигурация Release)
Рукописный фильтр	1020
Библиотечный фильтр	14

Вывод

В ходе роботы был успешно реализован медианный фильтр, что подтверждается полным совпадением выходных изображений. Рукописная реализация фильтра оказалась примерно в 100 раз медленнее, чем библиотечная, так как является менее оптимизированной.