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 с использованием библиотеки OpenCV

Студент гр. 3331506/70401 Засецкий В.С.

Преподаватель Варлашин В.В.

«21» 10 2020 г.

Санкт-Петербург

2020

Задание

С помощью методов OpenCV реализовать глобальную пороговую фильтрацию (truncate и threshold to zero) изображения.

Ход работы

Описание алгоритма

Фильтр Truncate имеет некоторое пороговое значение интенсивности каждого пикселя *threshold*, при превышении которого значению интенсивности присваивается значение порогового. Фильтр Truncate описывается следующей функцией:

$$dst(x,y) = \begin{cases} threshold, & \text{если } src(x,y) > threshold \\ src(x,y) & \text{иначе} \end{cases},$$

где $dst(x,y)$— матрица выходного изображения,

$src(x,y)$ — матрица входного изображения,

$threshold$ — пороговое значение.

Фильтр Threshold to zero изменяет значение интенсивности пикселя на ноль, если оно меньше порогового значения. Интенсивности значением больше порогового остаются неизменными. Фильтр Threshold to zero описывается следующей функцией:

$$dst(x,y) = \begin{cases} src(x,y), & \text{если } src(x,y) > threshold \\ 0 & \text{иначе} \end{cases},$$

где $dst(x,y)$— матрица выходного изображения,

$src(x,y)$ — матрица входного изображения,

$threshold$ — пороговое значение.

Алгоритм будет заключаться в прохождении по каждому пикселю изображения и сравнении его интенсивности с пороговым значением.

Класс FilteredImage

Для описания алгоритма фильтрации был создан класс FilteredImage. Класс приведён на рисунке 1.

```
class FilteredImage
{
private:
    int m_width;
    int m_height;
    float m_threshold;
    Mat m_image;
public:
    int truncate();
    int thresholdToZero();
    int showImage();
    void setImage(Mat image);
    Mat getImage();

public:
    FilteredImage();
    FilteredImage(float threshold);
    ~FilteredImage();
};
```

Рисунок 1 — Класс FilteredImage

Реализация алгоритма

Для выполнения глобальной пороговой фильтрации Truncate используется публичный метод *truncate*(). Его код представлен на рисунке 2.

```
int FilteredImage::truncate()
{
    if (m_image.empty())
        return -1;
    //для каждого цветового канала сравниваем интенсивность каждого пикселя с пороговой
    for (int channel = 0; channel < 3; channel++)
    {
        for (int i = 0; i < m_height; i++)
        {
            for (int j = 0; j < m_width; j++)
            {
                const float currentPixel = m_image.at<Vec3b>(i, j)[channel];
                if (currentPixel > m_threshold)
                    m_image.at<Vec3b>(i, j)[channel] = m_threshold;
            }
        }
    }
    return 0;
}
```

Рисунок 2 — Код метода *truncate*()

Для выполнения глобальной пороговой фильтрации Threshold to zero используется публичный метод *thresholdToZero*(). Его код представлен на рисунке 3.

```
int FilteredImage::thresholdToZero()
{
    if (m_image.empty())
        return -1;
    //для каждого цветового канала сравниваем интенсивность каждого пикселя с пороговой
    for (int channel = 0; channel < 3; channel++)
    {
        for (int i = 0; i < m_height; i++)
        {
            for (int j = 0; j < m_width; j++)
            {
                const float currentPixel = m_image.at<Vec3b>(i, j)[channel];
                if (currentPixel <= m_threshold)
                    m_image.at<Vec3b>(i, j)[channel] = 0;
            }
        }
    }
    return 0;
}
```

Рисунок 3 — Код метода *thresholdToZero*()

Сравнение методов класса FilteredImage с библиотечными функциями OpenCV

В ходе сравнения вычисляется среднеквадратичная погрешность между значениями интенсивности пикселей на изображениях с применённым фильтром в виде метода класса FilteredImage и в виде библиотечной функции.

Среднеквадратичная погрешность вычисляется кодом, приведённым на рисунке 4.

```
double deviation = 0;
double sum = 0;
for (int channel = 0; channel < 3; channel++)
{
    for (int i = 0; i < inputImage.rows; i++)
        for (int j = 0; j < inputImage.cols; j++)
            sum += pow (outputImageTask.at<Vec3b>(i, j)[channel] - outputImageLib.at<Vec3b>(i, j)[channel], 2);
}
deviation = sqrt(sum / 3 * inputImage.rows * inputImage.cols);
cout << deviation;
```

Рисунок 4 — Код вычисления среднеквадратичной погрешности

Время выполнения алгоритма вычисляется при помощи стандартной функции *clock*().

Для сравнения использовалось изображение градиента от чёрного к белому в формате jpg. Исходное изображение приведено на рисунке 5.

Рисунок 5 — Исходное изображение для сравнения

1. Сравнение для фильтра Truncate

Результат применения фильтра приведён на рисунке 6. Сверху — метод класса FilteredImage, снизу — библиотечная функция. Пороговое значение 127.

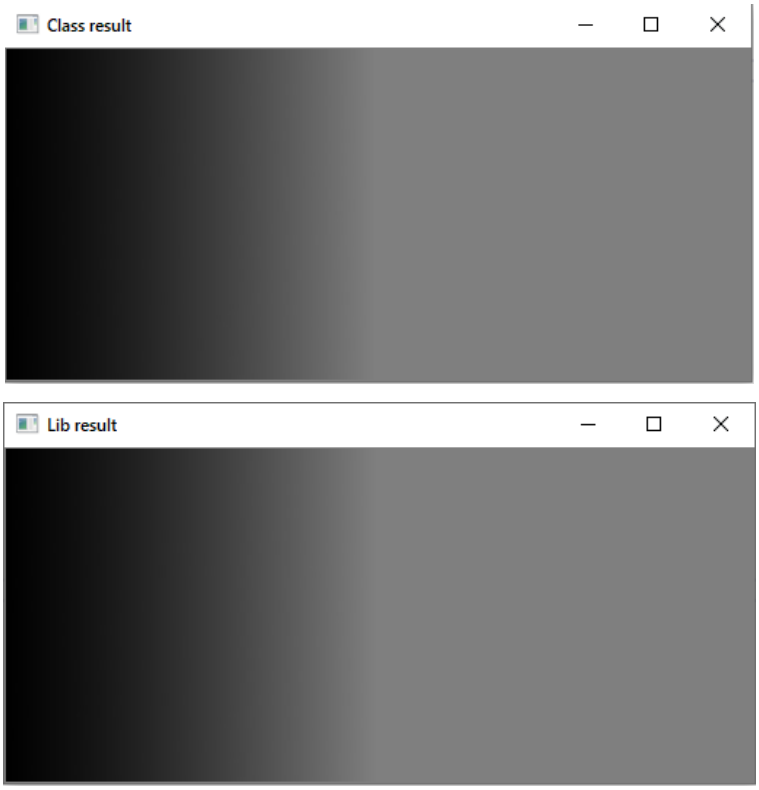

Рисунок 6 — Сравнение для фильтра Truncate

Среднеквадратичная погрешность равна нулю — результаты одинаковы.

Сравнение времени выполнения для версий Debug и Release приведено в таблице 1.

Таблица 1 — Сравнение времени выполнения алгоритма Truncate

	Debug	Release
Метод класса	171 мс	132 мс
Функция OpenCV	100 мс	90 мс

2. Сравнение для фильтра Threshold to zero

Результат применения фильтра приведён на рисунке 7. Сверху — метод класса FilteredImage, снизу — библиотечная функция. Пороговое значение 127.

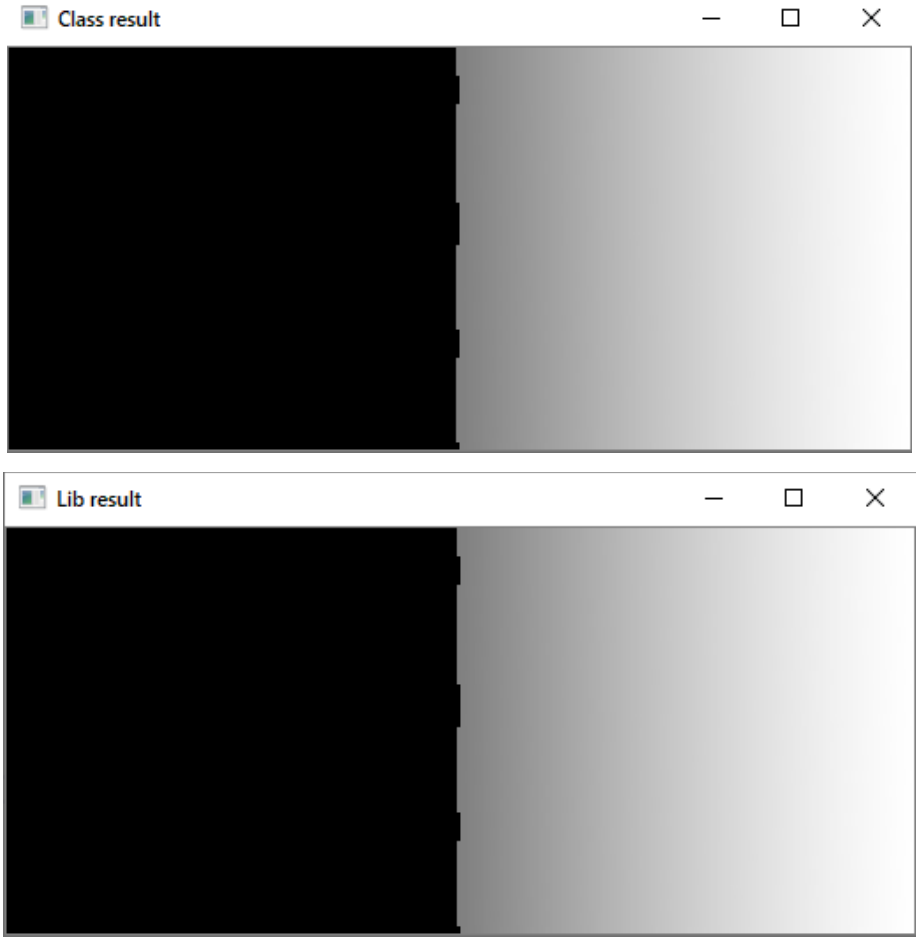

Рисунок 7 — Сравнение для фильтра Threshold to zero

Среднеквадратичная погрешность равна нулю — результаты одинаковы.

Сравнение времени выполнения для версий Debug и Release приведено в таблице 2.

Таблица 2 — Сравнение времени выполнения алгоритма Threshold to zero

	Debug	Release
Метод класса	168 мс	126 мс
Функция OpenCV	93 мс	80 мс

Вывод

В результате выполнения лабораторной работы реализованы глобальные фильтры Truncate и Threshold to zero. Результаты фильтрации реализованным методом и встроенной функцией из OpenCV оказались полностью одинаковы; встроенные функции выполнялись быстрее.