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: Фильтр Гаусса

Студент гр. 3331506/70401 Ляпцев И.А.

Преподаватель Варлашин В. В.

« »_____2020 г.

Санкт-Петербург

2020

Задание

Пользуясь средствами языка C++ и библиотеки OpenCV, реализовать фильтр Гаусса- алгоритма, обрабатывающего изображения, чаще всего для снижения уровня шума, с ядром 7 на 7 ячеек с якорем в нижнем левом углу.

Ход работы

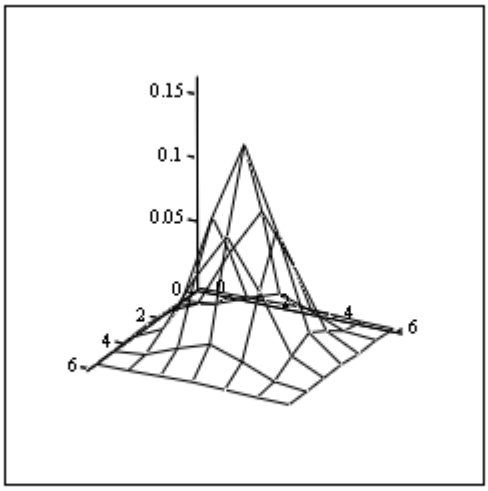
Описание алгоритма

Фильтр Гаусса использует для расчета ядра функцию Гаусса, в результате распределение в ядре удовлетворяет нормальному распределению.

$$G(x,y) := \frac{1}{2\cdot\pi\cdot\sigma^2}\cdot e^{\frac{-\left(x^2+y^2\right)}{2\cdot\sigma^2}}$$

$\sigma := 1$

$$Fu := \begin{pmatrix} G(3,3) & G(3,2) & G(3,1) & G(3,0) & G(3,1) & G(3,2) & G(3,3) \\ G(2,3) & G(2,2) & G(1,2) & G(2,0) & G(1,2) & G(2,2) & G(1,3) \\ G(1,3) & G(1,2) & G(1,1) & G(1,0) & G(1,1) & G(1,2) & G(2,3) \\ G(3,0) & G(2,0) & G(1,0) & G(0,0) & G(1,0) & G(2,0) & G(3,0) \\ G(1,3) & G(1,2) & G(1,1) & G(1,0) & G(1,1) & G(1,2) & G(1,3) \\ G(2,3) & G(2,2) & G(1,2) & G(2,0) & G(1,2) & G(2,2) & G(2,3) \\ G(3,3) & G(3,2) & G(3,1) & G(3,0) & G(3,1) & G(3,2) & G(3,3) \end{pmatrix} = \begin{pmatrix} 1.964\times10^{-5} & 2.393\times10^{-4} & 1.072\times10^{-3} & 1.768\times10^{-3} & 1.072\times10^{-3} & 2.393\times10^{-4} & 1.964\times10^{-5} \\ 2.393\times10^{-4} & 2.915\times10^{-3} & 0.013 & 0.022 & 0.013 & 2.915\times10^{-3} & 1.072\times10^{-3} \\ 1.072\times10^{-3} & 0.013 & 0.059 & 0.097 & 0.059 & 0.013 & 2.393\times10^{-4} \\ 1.768\times10^{-3} & 0.022 & 0.097 & 0.159 & 0.097 & 0.022 & 1.768\times10^{-3} \\ 1.072\times10^{-3} & 0.013 & 0.059 & 0.097 & 0.059 & 0.013 & 1.072\times10^{-3} \\ 2.393\times10^{-4} & 2.915\times10^{-3} & 0.013 & 0.022 & 0.013 & 2.915\times10^{-3} & 2.393\times10^{-4} \\ 1.964\times10^{-5} & 2.393\times10^{-4} & 1.072\times10^{-3} & 1.768\times10^{-3} & 1.072\times10^{-3} & 2.393\times10^{-4} & 1.964\times10^{-5} \end{pmatrix}$$



Fu

Далее происходит проход ядром по всему изображению, в ходе которого пикселю присваивается нормализованная сумма всех значений пикселей вокруг него, умноженная на соответствующее им значение функции Гаусса

Реализация алгоритма

Для описания фильтра Гаусса был создан класс *current_image*, изображенный на рисунке 1.

```
class current_image
{
public:
    current_image() = default;

    current_image(float sigma);

    ~current_image() = default;

    int setImage(string imageName);

    Mat getImage();

    int processImage();

    int expandImage(Mat& image);
    int cropImage(Mat& image);

    int corePass(Mat& image);

private:
    Mat m_processingImage;
    Mat m_defaultImage;
    float m_sigma;
};
```

Рисунок 1 – Класс *current_image*

Публичный метод *processImage()*, код которого представлен на рисунке 2, осуществляет ряд действий для преобразования изображения.

```
int current_image::processImage() // обрабатывает изображение
{
    expandImage([&]m_processingImage);
    corePass([&]m_processingImage);
    return cropImage([&]m_processingImage);
}
```

Рисунок 2 – Метод *processImage()*

Функции, используемые в методе *processImage()*, рассматриваются ниже.

Для расширения границ изображения (константным способом) реализован метод *expandImage(Mat& image)*, код которого представлен на рисунке 3.

```
int current_image::expandImage(Mat& image) //расширение границ, заполнение "пустоты" константным черным цветом
{
    if (image.empty())
    {
        return -1;
    }
    Mat biggerImage = Mat( rows: (image.rows + 6), cols: (image.cols + 6), type: CV_8UC3, s: Scalar(0, 0, 0));
    for (int i = 0; i < image.rows; i++)
    {
        for (int k = 0; k < image.cols; k++)
        {
            for (int g = 0; g < 3; g++)
                biggerImage.at<Vec3b>( row: i + 6, col: k)[g] = image.at<Vec3b>( row: i, col: k)[g];
        }
    }
    image = biggerImage.clone();
    return 0;
}
```

Рисунок 3 – Метод *expandImage(Mat& image)*

Далее выполняется проход ядром по всему изображению, вычисление массива значений функции Гаусса для каждой ячейки ядра представлен на рисунке 4, сам проход ядром с последующим присвоением значения каналу представлен на рисунке 5.

```
double makeGauss[7][7] = {};
double gSum = 0;
for (int x = -3; x <= 3; x++)
{
    for (int y = -3; y <= 3; y++)
    {
        makeGauss[x + 3][y + 3] = (1 / (2 * CV_PI * m_sigma * m_sigma)) * exp( _xx: -(x * x + y * y) / (2 * m_sigma * m_sigma));
        gSum += makeGauss[x + 3][y + 3];
    }
}
```

Рисунок 4 – вычисление значений функции Гаусса

```
for (int i = 6; i < image.rows; i++)
{
    for (int k = 0; k < image.cols - 6; k++)
    {
        for (int g = 0; g < 3; g++)
        {
            bool isAllPixelSame = true;
            const double firstPixel = image.at<Vec3b>(row: i, col: k)[g];
            double pixSum = 0;
            for (int r = 0; r <= 6; r++)
            {
                for (int c = 0; c <= 6; c++)
                {
                    const double currentPixel = image.at<Vec3b>(row: i - r, col: k + c)[g];
                    if (currentPixel != firstPixel)
                        isAllPixelSame = false;
                    pixSum += currentPixel * makeGauss[r][c];
                }
            }
            if (isAllPixelSame)
                continue;

            pixSum /= gSum;

            if (pixSum < 0) pixSum = 0;
            if (pixSum > 255) pixSum = 255;

            image.at<Vec3b>(row: i, col: k)[g] = pixSum;
        }
    }
}
```

Рисунок 5 – Проход ядром по изображению

Далее производится обрезка изображения до исходных размеров методом *cropImage(Mat& image)*, код которого представлен на рисунке 6.

```
int current_image::cropImage(Mat& image) //обрезаем картинку обратно
{
    if (image.empty())
        return -1;

    Mat croppedImage = Mat(rows: (image.rows - 6), cols: (image.cols - 6), type: CV_8UC3, s: Scalar(255, 255, 255));

    for (int i = 6; i < image.rows; i++)
    {
        for (int k = 0; k < image.cols - 6; k++)
        {
            for (int g = 0; g < 3; g++)
                croppedImage.at<Vec3b>(row: i - 6, col: k)[g] = image.at<Vec3b>(row: i, col: k)[g];
        }
    }
    image = croppedImage.clone();
    return 0;
}
```

Рисунок 6 – Метод *cropImage(Mat& image)*

Сравнение с методом из OpenCV

Учитывая, что библиотечная функция имеет другие параметры, а именно якорь в центральной ячейке ядра, алгоритм прохода ядром по изображению был изменен, обновленная версия с якорем в центре ядра представлена на рисунке 7.

```
for (int i = 6; i < image.rows-6; i++)
{
    for (int k = 6; k < image.cols - 6; k++)
    {
        for (int g = 0; g < 3; g++)
        {
            bool isAllPixelSame = true;
            const double firstPixel = image.at<Vec3b>(row: i, col: k)[g];
            double pixSum = 0;
            for (int r = 0; r <= 6; r++)
            {
                for (int c = 0; c <= 6; c++)
                {
                    const double currentPixel = image.at<Vec3b>(row: i + r - 3, col: k + c - 3)[g];
                    if(currentPixel != firstPixel)
                        isAllPixelSame = false;
                    pixSum += currentPixel * makeGauss[r][c];
                }
            }
            if(isAllPixelSame)
                continue;

            pixSum /= gSum;

            if (pixSum < 0) pixSum = 0;
            if (pixSum > 255) pixSum = 255;

            image.at<Vec3b>(row: i, col: k)[g] = pixSum;
        }
    }
}
```

Рисунок 7 – Обновленный проход ядром по изображению

Для сравнения использовалось изображение разрешением 512 на 512 пикселей (см. рисунок 8).

Рисунок 8 – Оригинальное изображение

Изображения, обработанные реализованным фильтром (слева) и фильтром из OpenCV (справа), изображены на рисунке 9.

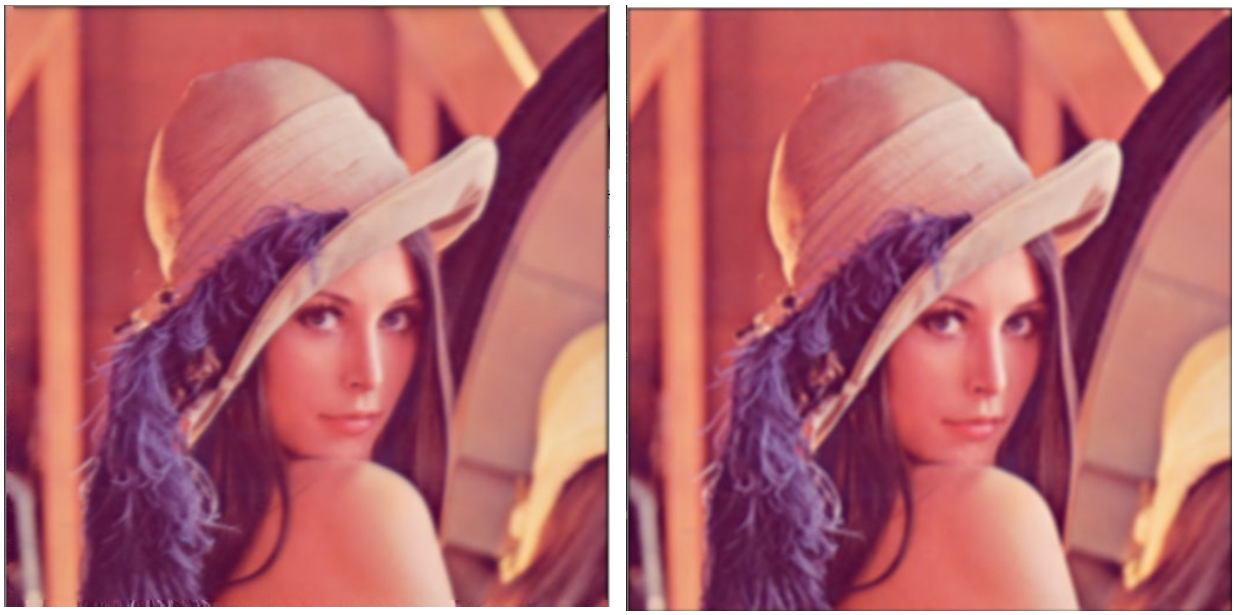


Рисунок 9 – Сравнение результатов

Для точного сравнения изображений произвелось вычитание изображения и последующее инвертирование для удобного представления, результат представлен рисунке 10.

Также было рассчитано среднее отклонение по модулю для каждого канала, которое представлено в таблице 1.

	b	g	r
Среднее отклонение по модулю	2.56041	2.87476	3.56872

Как можно заметить, разница не превышает 1.5%, из чего можно судить о правильной работе метода.

Рисунок 10 – Инвертированная разница двух изображений

При оценке же изображения оно имеет минимум черных точек, из чего можно заключить, что результаты практически совпадают.

Для сравнения времени выполнения обработки изображения использовалась функция *clock()* библиотеки time.h, которая возвращает количество тактов процессора, прошедших с момента запуска программы. Поделив это количество тактов на количество тактов в секунду (CLOCKS_PER_SEC), можно получить время в секундах с момента запуска программы.

Результаты для сборок *release* и *debug* представлены в таблице 2.

Таблица 2. Сравнение времени обработки изображений.

Версия	Время метода, мс	Время метода из OpenCV, мс
debug	2.59	0.111
release	0.076	0.007

Вывод

В результате выполнения лабораторной работы был реализован метод обработки изображения фильтром Гаусса. Результаты методом и методом из OpenCV практически полностью совпали, разница составила не более 1.5%, однако метод из OpenCV более оптимизирован по времени выполнения обработки.