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: Фильтры

Студент гр. 3331506/70401 Жернаков А.А.

Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург

2020

Задание

Реализовать адаптивный пороговый фильтр с использованием средневзвешенных значений. Ядро размером 3x3 с якорем в правом верхнем углу.

Задачи

1) Реализовать считывание входного изображения и увеличение его в соответствии с размером ядра и расположением якорной точки;

2) Реализовать обработку изображения и удаление границ;

3) Сравнить время выполнения алгоритма с временем выполнения аналогичного алгоритма из библиотеки OpenCV.

Ход работы

Адаптивный пороговый фильтр с использованием средневзвешенных значений работает следующим образом:

1) Ядро устанавливается в начальное положение на изображении.

2) По формуле 1 высчитывается порог фильтрации.

$$T(x,y) = mean(P1(x - bsize, y + bsize), P2(x + bsize, y - bsize)) - C \quad (1)$$

где - C – константа, задаваемая пользователем.

3) Согласно формуле 2, сравнивается значение интенсивности якорной точки с пороговым значением и определяется интенсивность соответствующей точки выходного изображения.

$$dst(x,y) = \begin{cases} maxValue\ if\ src(x,y) > T(x,y) \\ 0 \qquad\qquad\qquad otherwise \end{cases} \quad (2)$$

где $maxValue$ – переменная, которая задается пользователем.

4) Ядро смещается на один пиксель, пункты 2 и 3 повторяются.

Для описания алгоритма был создан класс *AdaptiveThreshold*, который показан на рисунке 1.

```
class AdaptiveThreshold
{
public:
    AdaptiveThreshold() = default;
    AdaptiveThreshold(int m_maxValue, int m_constant, int m_positionOfAnchor);
    ~AdaptiveThreshold() = default;

    void adaptiveThreshold(Mat& inputImage, Mat& outputImage);
    float meanValue(Mat& image, int row, int col);

private:
    int m_maxValue;
    int m_constant;
    int m_positionOfAnchor;
};
```

Рисунок 1 – Класс *AdaptiveThreshold*

Переменная *positionOfAnchor* отражает позицию якоря (1 – левый верхний угол, … , 9 – правый нижний угол), вводится пользователем.

Основные функции

1) Функция *meanValue*()

Реализация функции *meanValue*, которая возвращает средневзвешенное значение, представлена на рисунке 2.

```
float AdaptiveThreshold::meanValue(Mat& image, int row, int col)
{
    float mean = 0;
    mean = (image.at<uchar>(row - 1, col - 1) + image.at<uchar>(row - 1, col) + image.at<uchar>(row - 1, col + 1) +
            image.at<uchar>(row, col - 1) + image.at<uchar>(row, col) + image.at<uchar>(row, col + 1) +
            image.at<uchar>(row + 1, col - 1) + image.at<uchar>(row + 1, col) + image.at<uchar>(row + 1, col + 1)) / 9 - C;
    return mean;
}
```

Рисунок 2 – Функция *meanValue*

2) Функция *adaptiveThreshold*()

Функция *adaptiveThreshold* считывает значение переменной *positionOfAnchor*, производит увеличение входного изображения в соответствии с размером ядра и расположением якорной точки. Затем выполняется сравнение значения интенсивности якорной точки с пороговым значением, заполнение выходного изображения и обрезка границ. Реализация функции *adaptiveThreshold* представлена на рисунке 3.

```
void AdaptiveThreshold::adaptiveThreshold(Mat& inputImage, Mat& outputImage)
{
    int rowOfAnchor = 0;
    int colOfAnchor = 0;
    switch (m_positionOfAnchor) { ... }

    Mat cloneOfImg = inputImage.clone();

    Mat bigClone(Size(cloneOfImg.cols + 2, cloneOfImg.rows + 2), cloneOfImg.type());
    Rect rectangle(colOfAnchor, rowOfAnchor, cloneOfImg.cols, cloneOfImg.rows);
    cloneOfImg.copyTo(bigClone(rectangle));

    Mat result(Size(outputImage.cols + 2, outputImage.rows + 2), outputImage.type());

    for (int i = 0; i < bigClone.rows - 2; i++) // строки
    {
        for (int j = 0; j < bigClone.cols - 2 ; j++) // столбцы
        {
            if (bigClone.at<uchar>(i + rowOfAnchor, j + colOfAnchor) > meanValue(bigClone, i + 1, j + 1))
            {
                result.at<uchar>(i + rowOfAnchor, j + colOfAnchor) = m_maxValue;
            }
            else
            {
                result.at<uchar>(i + rowOfAnchor, j + colOfAnchor) = 0;
            }
        }
    }

    Rect rectangle2(colOfAnchor, rowOfAnchor, cloneOfImg.cols, cloneOfImg.rows);
    result(rectangle2).copyTo(outputImage);
}
```

Рисунок 3 – Функция *adaptiveThreshold*

Результат обработки

Пример обработки изображения с ядром 3x3 и якорем в правом верхнем углу представлен на рисунке 4.

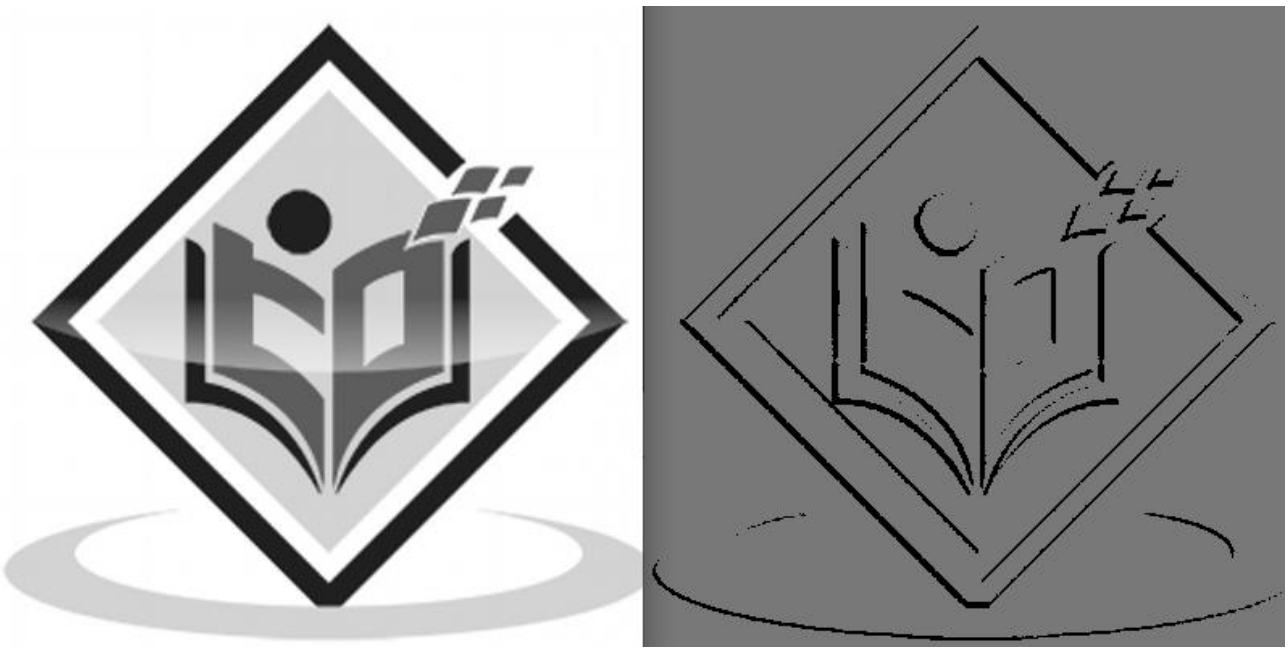

Рисунок 4 – Пример обработки

Сравнение с аналогичной функцией из OpenCV

1) Сравнение качества обработки

В библиотеке OpenCV существует функция *adaptiveThreshold*. При сравнении с ней использовано ядро 3x3 с якорем в центральной точке. Результат обработки двумя функциями показан на рисунке 5 (слева – функция из OpenCV, справа – функция, реализованная в ходе работы).

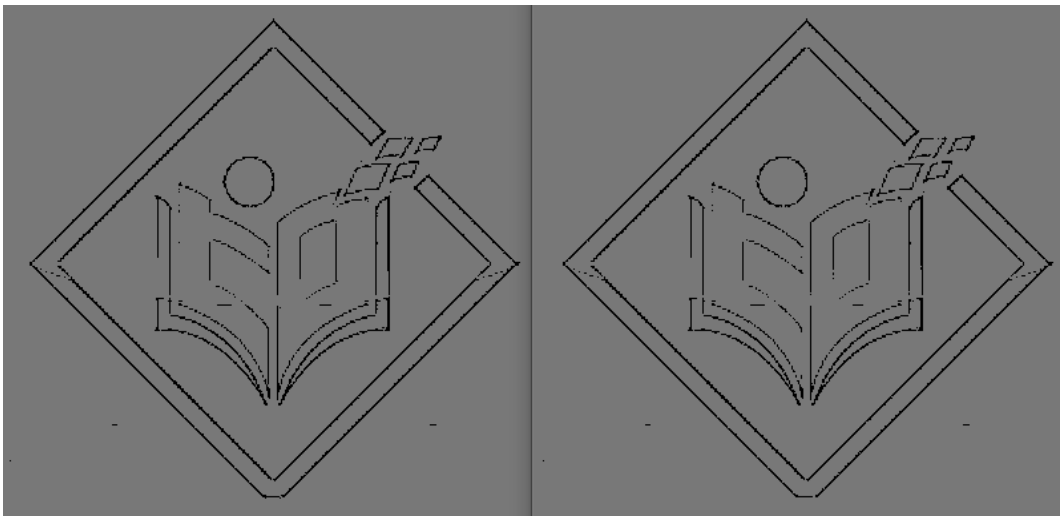

Рисунок 5 – Сравнение с функцией из OpenCV

Для оценки результатов вычтем из одного изображения другое с помощью функции *absdiff* и посчитаем среднюю квадратичную погрешность по формуле 3.

$$S = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \quad (3)$$

где x_i – интенсивность i-го пикселя матрицы, полученной вычитанием одного изображения из другого; $\bar{x}$ – среднее арифметическое значение этой матрицы; n – количество пикселей.

В результате вычислений получаем $S = 3{,}92$.

2) Сравнение времени работы

Для сравнения времени выполнения обработки используем функцию *clock* из библиотеки *time.h*.

В результате сравнения времени выполнения определено, что скорость работы функции из библиотеки OpenCV выше примерно в 15 раз.

Вывод

В ходе работы изучены принципы работы заданного фильтра. Выполнена его реализация и проведено сравнение с аналогичным алгоритмом из библиотеки OpenCV. Незначительные отличия в качестве обработки обусловлены округлениями значений в ходе обработки.