

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 изображения с использованием библиотеки OpenCV

Студент гр. 3331506/70401

Коновалов В.А.

Преподаватель

Варлашин В.В.

« » _____ 2020 г.

Санкт-Петербург

2020

Задание

Реализовать адаптивный пороговый фильтр с использованием средневзвешенных значений с параметрами: размер ядра 5×5 , якорная точка в правом нижнем углу, обработка границ методом отражения (`border_reflect_101`).

Математическое описание

Согласно заданию дано ядро 5×5 с якорной точкой в правом нижнем углу

$$\begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} \\ C_{51} & C_{52} & C_{53} & C_{54} & \boxed{C_{55}} \end{pmatrix},$$

где C_{ij} – веса, $\boxed{C_{ij}}$ – якорная точка.

По формуле 1 вычисляется порог фильтрации

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

где операцией *mean* мы высчитываем среднее значение интенсивности от пиксела $P1$ до пиксела $P2$, $bsize$ – размер ядра, а C – пороговая константа, задаваемая пользователем.

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

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

Обработка границ осуществляется методом отражения (*border_reflect_101*). Расширение исходного изображения происходит отображением интенсивностей пикселей от края сначала по горизонтали, а потом по вертикали.

Для реализации адаптивного порогового фильтра был создан класс *AdaptiveThreshold*, представленный на листинге 1.

Листинг 1 – Класс *AdaptiveThreshold*

```
#pragma once

#include <iostream>
#include "opencv2\core.hpp"
#include "opencv2\highgui.hpp"

using namespace cv;
using namespace std;

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;
};
```

Переменная *m_positionOfAnchor* отражает позицию якоря (1 – верхний левый угол, ..., 25 – правый нижний угол).

Основные функции

1) Функция *meanValue()*

Реализация функции *meanValue()*, которая возвращает средневзвешенное значение, представлена на листинге 2.

Листинг 2 – Функция *meanValue()*

```
float AdaptiveThreshold::meanValue(Mat& image, int row, int col)
{
    float mean = 0;
    mean = (image.at<uint8_t>(row - 2, col - 2) + image.at<uint8_t>(row - 2, col - 1) +
    image.at<uint8_t>(row - 2, col) + image.at<uint8_t>(row - 2, col + 1) +
    image.at<uint8_t>(row - 2, col + 2) + image.at<uint8_t>(row - 1, col - 2) +
    image.at<uint8_t>(row - 1, col - 1) + image.at<uint8_t>(row - 1, col) +
    image.at<uint8_t>(row - 1, col + 1) + image.at<uint8_t>(row - 1, col + 2) +
    image.at<uint8_t>(row, col - 2) + image.at<uint8_t>(row, col - 1) +
    image.at<uint8_t>(row, col) + image.at<uint8_t>(row, col + 1) +
    image.at<uint8_t>(row, col + 2) + image.at<uint8_t>(row + 1, col - 2) +
    image.at<uint8_t>(row + 1, col - 1) + image.at<uint8_t>(row + 1, col) +
    image.at<uint8_t>(row + 1, col + 1) + image.at<uint8_t>(row + 1, col + 2) +
    image.at<uint8_t>(row + 2, col - 2) + image.at<uint8_t>(row + 2, col - 1) +
    image.at<uint8_t>(row + 2, col) + image.at<uint8_t>(row + 2, col + 1) +
    image.at<uint8_t>(row + 2, col + 2)) / 25 - (double)m_constant;
    return mean;
}
```

2) Функция *adaptiveThreshold()*

В функции *adaptiveThreshold()* осуществляется обработка границ методом *border_reflect_101*, соответственно, происходит расширение изображения. Затем считывается значение переменной *m_positionOfAnchor*, после чего выполняется сравнение значения интенсивности якорной точки с пороговым значением, заполнение выходного изображения и обрезка границ. Реализация функции *adaptiveThreshold()* представлена на рисунка 1, 2 и 3.

```
31 void AdaptiveThreshold::adaptiveThreshold(Mat& inputImage, Mat& outputImage)
32 {
33     Mat tmpImage(Size(inputImage.cols + 4, inputImage.rows + 4), inputImage.type(), Scalar(255));
34
35     //Copy src
36     for (int i = (5 / 2); i < (tmpImage.rows - (5 / 2)); i++)
37     {
38         for (int j = (5 / 2); j < (tmpImage.cols - (5 / 2)); j++)
39         {
40             tmpImage.at<uint8_t>(i, j) = inputImage.at<uint8_t>((i - (5 / 2)), (j - (5 / 2)));
41         }
42     }
43
44     imshow("First", tmpImage);
45     while (waitKey(0) != 27)
46     {
47         ;
48     }
49
50     //border_reflect_101
51     //left -> right -> top -> bottom
52     int count = 1;
53
54     for (int i = (5 / 2); i < (tmpImage.rows - (5 / 2)); i++)
55     {
56         for (int j = ((5 / 2) - 1); j >= 0; j--)
57         {
58             tmpImage.at<uint8_t>(i, j) = tmpImage.at<uint8_t>(i, (j + (2 * count)));
59             count++;
60         }
61         count = 1;
62     }
63
64     for (int i = (5 / 2); i < (tmpImage.rows - (5 / 2)); i++)
65     {
66         for (int j = (tmpImage.cols - (5 / 2)); j < tmpImage.cols; j++)
67     }
```

Рисунок 1 – Фрагмент реализации функции *adaptiveThreshold()*

```

67     {
68         tmpImage.at<uint8_t>(i, j) = tmpImage.at<uint8_t>(i, (j - (2 * count)));
69         count++;
70     }
71     count = 1;
72 }
73
74 for (int j = 0; j < tmpImage.cols; j++)
75 {
76     for (int i = ((5 / 2) - 1); i >= 0; i--)
77     {
78         tmpImage.at<uint8_t>(i, j) = tmpImage.at<uint8_t>((i + (2 * count)), j);
79         count++;
80     }
81     count = 1;
82 }
83
84 for (int j = 0; j < tmpImage.cols; j++)
85 {
86     for (int i = (tmpImage.rows - (5 / 2)); i < tmpImage.rows; i++)
87     {
88         tmpImage.at<uint8_t>(i, j) = tmpImage.at<uint8_t>((i - (2 * count)), j);
89         count++;
90     }
91     count = 1;
92 }
93 imshow("Second", tmpImage);
94 while (waitKey(0) != 27)
95 {
96     ;
97 }
98
99 int rowOfAnchor = 0;
100 int colOfAnchor = 0;
101 switch (m_positionOfAnchor) { ... }
102
204
205 Mat result(Size(outputImage.cols + 4, outputImage.rows + 4), outputImage.type());
206
207 for (int i = 0; i < tmpImage.rows - 4; i++) // строки
208 {

```

Рисунок 2 – Фрагмент реализации функции *adaptiveThreshold()*

```

209     for (int j = 0; j < tmpImage.cols - 4; j++) // столбцы
210     {
211         if ((double)tmpImage.at<uint8_t>(i + rowOfAnchor, j + colOfAnchor) > meanValue(tmpImage, i + 2, j + 2))
212         {
213             result.at<uint8_t>(i + rowOfAnchor, j + colOfAnchor) = m_maxValue;
214         }
215         else
216         {
217             result.at<uint8_t>(i + rowOfAnchor, j + colOfAnchor) = 0;
218         }
219     }
220 }
221
222 Rect rectangle2(colOfAnchor, rowOfAnchor, inputImage.cols, inputImage.rows);
223 result(rectangle2).copyTo(outputImage);
224

```

Рисунок 3 – Фрагмент реализации функции *adaptiveThreshold()*

Результат обработки

Первоначальное изображение и пример обработки изображения с ядром 5×5 и якорем в правом нижнем углу представлен на рисунках 4 и 5 соответственно.



Рисунок 4 – Первоначальное изображение



Рисунок 5 – Пример обработки изображения при значении $maxValue = 125$, константы $C = 10$ и якорем в правом нижнем углу

Сравнение с аналогичной функцией из OpenCV

1) Сравнение качества обработки

В библиотеке OpenCV есть функция *adaptiveThreshold*. При сравнении с ней использовано ядро 5×5 с якорем в центральной точке. Результат обработки функцией, реализованной в ходе работы, представлен на рисунке 6, а функцией из OpenCV – на рисунке 7.

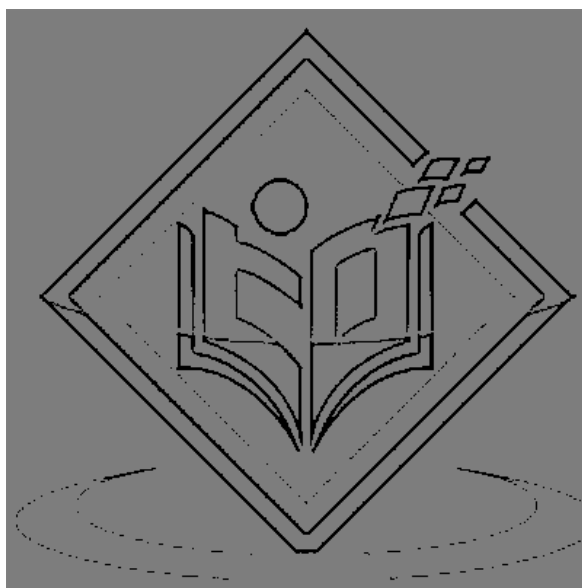


Рисунок 6 – Адаптивный пороговый фильтр, реализованный в ходе работы

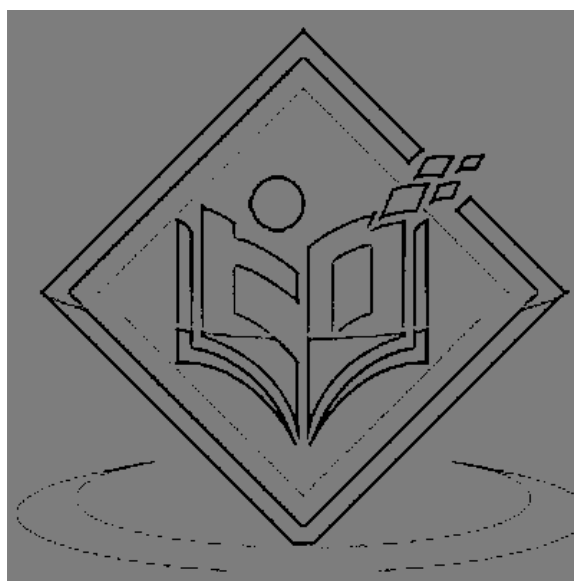


Рисунок 7 – Адаптивный пороговый фильтр из OpenCV

Для оценки найдём среднеквадратичное отклонение при помощи. Ввод входных данных, их обработка и функция вычисления среднеквадратичного отклонения представлены на листинге 3.

Листинг 3 – Ввод входных данных, их обработка и функция вычисления среднеквадратичного отклонения

```
#pragma once

#include <iostream>
#include "opencv2\core.hpp"
#include "opencv2\highgui.hpp"
#include "opencv2\imgproc.hpp"
#include "filter.h"
#include "time.h"

using namespace cv;
using namespace std;

float deviation(Mat& compare);

int main()
{
    clock_t t;
    Mat img = imread("D:/Technical vision/data/thresh_input.jpg", IMREAD_GRAYSCALE);
    imshow("img", img);
    while (waitKey(0) != 27)
    {
        ;
    }
    Mat clone = img.clone();

    Mat result(Size(clone.cols, clone.rows), CV_8UC1, Scalar(255));
    AdaptiveThreshold filter(125, 10, 13);
    t = clock();
    filter.adaptiveThreshold(img, result);
    t = clock() - t;
    cout << (float)t / CLOCKS_PER_SEC << endl;
    imshow("result", result);
    while (waitKey(0) != 27)
    {
        ;
    }

    Mat result2(Size(clone.cols, clone.rows), CV_8UC1, Scalar(255));
    t = clock();
    adaptiveThreshold(clone, result2, 125, ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY, 5, 10);
    t = clock() - t;
    cout << (float)t / CLOCKS_PER_SEC << endl;
    imshow("result2", result2);
    while (waitKey(0) != 27)
    {
        ;
    }

    Mat compare(Size(result.cols, result.rows), CV_8UC1, Scalar(0));
    absdiff(result2, result, compare);
    imshow("compare", compare);
    while (waitKey(0) != 27)
    {

```

```

        ;
    }

    float error = deviation(compare);
    cout << error;

    while (waitKey(33) != 27) {}

    return 0;
}

float deviation(Mat& compare)
{
    int sum = 0;
    int n = compare.rows * compare.cols;

    for (int i = 0; i < compare.rows; i++)
    {
        for (int j = 0; j < compare.cols; j++)
        {
            sum += (int)compare.at<uint8_t>(i, j);
        }
    }

    float middle = (float)sum / n;

    float s = 0;
    float sum_2 = 0;

    for (int i = 0; i < compare.rows; i++)
    {
        for (int j = 0; j < compare.cols; j++)
        {
            sum_2 += ((float)compare.at<uint8_t>(i, j) - middle)
*((float)compare.at<uint8_t>(i, j) - middle);
        }
    }

    s = sqrt((sum_2 / n) / (n - 1));

    return s;
}

```

Среднеквадратичное отклонение составило $S \approx 0,2$.

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 30 раз, что связано с неоптимальным с точки зрения скорости написанием кода в собственной реализации адаптивного порогового фильтра.

Вывод

В ходе работы изучены принципы работы заданного фильтра. Выполнена его реализация и проведено сравнение с аналогичным алгоритмом из библиотеки OpenCV. Отличия в качестве обработки связаны с округлением значений в ходе работы.