

Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

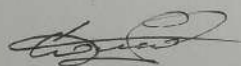
Отчёт

по лабораторной работе №2

Дисциплина: Техническое зрение

Тема: Реализация порогового фильтра с использованием библиотеки OpenCV

Студент гр. 3331506/70401



Сомов А. С.

Преподаватель



Варлашин В. В.

« 3 » 11 2020 г.

Санкт-Петербург

2020

Задача

Требуется реализовать пороговый фильтр с использованием библиотеки OpenCV. Сравнить свой алгоритм с реализованным в библиотеке OpenCV. В варианте находятся 2 пороговых фильтра: бинарный и бинарный инвертированный.

Реализация

Как бинарный, так и бинарный инвертированный пороговый фильтры реализованы в следующем методе класса BinaryThreshold

Mat applyBinaryFilterToImg();

Тело метода выглядит следующим образом:

```
void BinaryThreshold::applyBinaryFilterToImg(Mat imageBase, int threshold, int maxValue, string &filterName)
{
    int minValue;
    if (filterName == "B")
    {
        minValue = 0;
    } else if (filterName == "BI")
    {
        minValue = maxValue;
        maxValue = 0;
    }
    for (int i = 0; i < imageBase.rows; i++)
    {
        for (int j = 0; j < imageBase.cols; j++)
        {
            switch (imageBase.channels())
            {
                case 3:
                    if ((imageBase.at<Vec3b>(i, j)[0] <= threshold) || (imageBase.at<Vec3b>(i, j)[1] <= threshold) ||
                        (imageBase.at<Vec3b>(i, j)[2] <= threshold))
                    {
                        imageBase.at<Vec3b>(i, j)[0] = minValue;
                        imageBase.at<Vec3b>(i, j)[1] = minValue;
                        imageBase.at<Vec3b>(i, j)[2] = minValue;
                    } else
                    {
                        imageBase.at<Vec3b>(i, j)[0] = maxValue;
                        imageBase.at<Vec3b>(i, j)[1] = maxValue;
                        imageBase.at<Vec3b>(i, j)[2] = maxValue;
                    }
                    break;
                case 2:
                    if ((imageBase.at<Vec2b>(i, j)[0] <= threshold) || (imageBase.at<Vec2b>(i, j)[1] <= threshold))
                    {
                        imageBase.at<Vec2b>(i, j)[0] = minValue;
                        imageBase.at<Vec2b>(i, j)[1] = minValue;
                    } else
                    {
                        imageBase.at<Vec2b>(i, j)[0] = maxValue;
                        imageBase.at<Vec2b>(i, j)[1] = maxValue;
                    }
                    break;
                case 1:
                    if ((imageBase.at<uint8_t>(i, j) <= threshold))
                    {
                        imageBase.at<uint8_t>(i, j) = minValue;
                    } else
                    {
                        imageBase.at<uint8_t>(i, j) = maxValue;
                    }
                    break;
            }
        }
    }
}
```

Handwritten notes:

- Under the condition `(imageBase.at<Vec2b>(i, j)[0] <= threshold) || (imageBase.at<Vec2b>(i, j)[1] <= threshold)`, there is a handwritten "false" with an arrow pointing to the condition.
- Under the condition `(imageBase.at<Vec2b>(i, j)[0] <= threshold) || (imageBase.at<Vec2b>(i, j)[1] <= threshold)`, there is a handwritten "true" with an arrow pointing to the condition.
- Under the condition `(imageBase.at<Vec2b>(i, j)[0] <= threshold) || (imageBase.at<Vec2b>(i, j)[1] <= threshold)`, there is a handwritten "}" with a question mark.

Реализация данного фильтра подходит для изображений 1, 2 и 3х канальных изображений.

Так как бинарный фильтр от бинарного инвертированного отличаются только значениями, которые необходимо присваивать пикселям. Выбор значения происходит в зависимости от порогового значения, передаваемого в функцию. Если нижнее значение всегда будет равно 0, то вернее значение может задаваться пользователем.

Перед тем как приступить к применению фильтра к изображениям необходимо подготовить входные данные. Фотографии, подвергаемые обработке, находятся в папке с проектом.

```
void BinaryThreshold::read(string path)
{
    vector<string> imgName;
    glob(path, imgName);
    for (size_t i = 0; i < imgName.size(); i++)
    {
        image.push_back(imread(imgName[i]));
    }
}
```

Метод read () позволяет получить названия файлов из папки с изображениями, затем записать эти изображения в оттенках серого в вектор, передавая ему путь к папке.

```
void BinaryThreshold::print(size_t imgNumber, const string &name)
{
    imshow(name, image[imgNumber]);
    while (waitKey(1) != 'c');
}
```


Метод класса BinaryThreshold

```
void BinaryThreshold::applyFilterToImage(size_t index, FilterTypes filterName)
{
    switch (filterName)
    {
        case BINARY_FILTER:
        {
            int threshold = 125;
            int maxValue = 255;
            string filter = "B";
            return applyBinaryFilterToImg(image[index], threshold, maxValue, filter);
        }
        case BINARY_FILTER_INV:
        {
            int threshold = 125;
            int maxValue = 255;
            string filter = "BI";
            return applyBinaryFilterToImg(image[index], threshold, maxValue, filter);
        }
    }
}
```

реализован для выбора фильтра и его применения к изображению, т. к. нет информации о том, как должна задаваться информация вводимая пользователем, её ввод реализован непосредственно в функции для большего удобства отладки.

Функция сравнения встроенного и реализованного фильтров производится в методе, представленном ниже

```
int BinaryThreshold::getDifference(Mat &original, size_t index)
{
    uint differenceCounter = 0;
    for (int i = 0; i < image[index].rows; i++)
    {
        for (int j = 0; j < image[index].cols; j++)
        {
            if ((abs(image[index].at<uint8_t>(i,j) - original.at<uint8_t>(i,j))) != 0)
            {
                differenceCounter++;
            }
        }
    }
    return (double)differenceCounter / (image[index].rows * image[index].cols);
}
```

из opencv

image[index].channels = ?

Результат сравнения выводится в процентном соотношении отличающихся пикселей. В данном случае отличие результата работы встроенного и реализованного фильтров составляет 0 %.

Все названия фильтров хранятся в заголовочном файле, который выглядит следующим образом:

```
#ifndef CV_LAB2_THRESHOLD_FILTER_H
#define CV_LAB2_THRESHOLD_FILTER_H

#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <opencv2/imgcodecs.hpp>
#include <string>
#include <vector>

using namespace std;
using namespace cv;

class BinaryThreshold
{
public:
    BinaryThreshold() = default;

    ~BinaryThreshold() = default;

    void read(string path);

    void print(size_t imgNumber, const string &name);

    Mat getImage(size_t index);

    enum FilterTypes
    {
        BINARY_FILTER,
        BINARY_FILTER_INV
    };

    BinaryThreshold(BinaryThreshold &base);

    void applyFilterToImage(size_t index, FilterTypes filterName);

    int getDifference(Mat &original, size_t index);

private:
    void applyBinaryFilterToImg(Mat imageBase, int threshold, int maxValue, string &filterName);

    vector<Mat> image;
};

#endif //CV_LAB2_THRESHOLD_FILTER_H
```

Код main-файла представлен ниже

```
#include " threshold_filter.h"
#include <ctime>

int main()
{
    string folderPath = "/home/alex/CLionProjects/CV_Lab2/Images/*.jpeg";
    const string before = "reference sample";
    const string after = "filtered sample";

    BinaryThreshold referenceSample;
    referenceSample.read(folderPath);

    BinaryThreshold workImages_ThrBin(referenceSample);
    BinaryThreshold workImages_ThrBinInv(referenceSample);

    uint32_t startTime = clock();

    //referenceSample.print(0, before);

    workImages_ThrBin.applyFilterToImage(0, BinaryThreshold::BINARY_FILTER);
    //workImages_ThrBin.print(0, after);

    //workImages_ThrBinInv.applyFilterToImage(0, BinaryThreshold::BINARY_FILTER_INV);
    //workImages_ThrBinInv.print(0, after);

    uint32_t endTime = clock();
    cout << double(endTime - startTime) / CLOCKS_PER_SEC << endl;

    uint32_t startTime1 = clock();

    Mat show;
    cv::threshold(referenceSample.getImage(0), show, 125, 255, THRESH_BINARY);
    //imshow("WINDOW", show);

    uint32_t endTime1 = clock();
    cout << double(endTime1 - startTime1) / CLOCKS_PER_SEC << endl;

    std::cout << "Difference " << workImages_ThrBin.getDifference(show, 0);
}
```


В данном файле выполняется сравнение по времени моего алгоритма и встроенного в библиотеку. Замер времени осуществляется функцией `clock()`.

По времени:

Время исполнения моей функции: 0,0013с

Время исполнения встроенной функции: 0,0001с

Результат по времени исполнения отличается на порядок.

Результаты работы фильтра бинарного и бинарного инвертированного.

До обработки фильтром:



После бинарного порогового фильтра:



После бинарного инвертированного порогового фильтра:



Вывод:

Самый простой метод сегментации

Пример применения: выделение области изображения, соответствующей объектам, которые мы хотим проанализировать. Это разделение основано на изменении интенсивности между пикселями объекта и пикселями фона.

Чтобы отличить интересующие нас пиксели от остальных (которые в конечном итоге будут откинута), мы выполняем сравнение значения интенсивности каждого пикселя с пороговым значением (определяемым в соответствии с решаемой проблемой).

После того, как мы правильно разделили важные пиксели, мы можем установить им определенное значение для их идентификации (т.е. мы можем присвоить им значение 0 (черный), 255 (белый) или любое значение, которое соответствует вашим потребностям). Выполнение задачи завершилось успехом, знания усвоены.