线性滤波

by 鱼虾一整碗

要点概览

三种线性滤波：方框滤波、均值滤波、高斯滤波；两种非线性滤波：中值滤波、双边滤波

图像滤波目的

1. 抽出对象特征，作为图像识别的特征模式
2. 消除图像数字化时所混入的噪声

滤波和模糊

高斯滤波是指用高斯函数为滤波函数；高斯模糊是高斯低通滤波

低通即**模糊**，高通即**锐化**

线性滤波核心API函数

方框滤波：boxFilter函数

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
#include <stdio.h>

using namespace cv;
using namespace std;

int main()
{
    //载入原图
    Mat image=imread("E:/Robomaster/视觉组资料/装甲识别/3.jpg");
    //创建窗口
    namedWindow("方框滤波【初始】");
    namedWindow("方框滤波【效果】");
    //显示原图
    imshow("方框滤波【初始】", image);
    //滤波
    Mat out;
    boxFilter(image, out, -1,Size(5, 5));
    //显示效果
    imshow("方框滤波【效果】", out);

    waitKey(0);


}
```

均值滤波：blur函数

高斯滤波：GaussianBlur函数

总而言之，这三个线性函数用法相似，只是在参数上会有些许差异，转到定义里可以查看具体区别

使用范例

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace cv;
using namespace std;

//全局变量声明

Mat g_srcImage, g_dstImage1, g_dstImage2, g_dstImage3;//存储图片的Mat类型
int g_nBoxFilterValue = 3;      //方框滤波参数值
int g_nMeanBlurValue = 3;      //均值滤波参数值
int g_nGaussianBlurValue = 3;  //高斯滤波参数值

//全局函数声明
//轨迹条的回调函数
static void on_BoxFilter(int, void *);
static void on_MeanBlur(int, void *);
static void on_GaussianBlur(int, void *);

//-----------------------------------------main----------------------------------------
int main()
{
    //改变console的颜色
    system("color5E");

    //载入原图
    g_srcImage=imread("E:/Robomaster/视觉组资料/装甲识别/3.jpg");
    if (!g_srcImage.data)
    {
        printf("读取srcImage错误~!\n");
        return false;
    }

    //复制原图到三个Mat类型中
    g_dstImage1 = g_srcImage.clone();
    g_dstImage2 = g_srcImage.clone();
    g_dstImage3 = g_srcImage.clone();

    //显示原图
    namedWindow("【<0>显示原图】",1);
    imshow("【<0>原图窗口】", g_srcImage);


    //方框滤波
    namedWindow("【<1>方框滤波】", 1);
    createTrackbar("内核值：", "【<1>方框滤波】", &g_nBoxFilterValue, 40, on_BoxFilter);
    on_BoxFilter(g_nBoxFilterValue, 0);

    //均值滤波
    namedWindow("【<2>均值滤波】", 1);
    createTrackbar("内核值：", "【<2>均值滤波】", &g_nMeanBlurValue, 40, on_MeanBlur);
    on_MeanBlur(g_nMeanBlurValue, 0);
```

```
    //高斯滤波
    namedWindow("【<3>高斯滤波】", 1);
    createTrackbar("内核值：", "【<3> 高斯滤波】", &g_nGaussianBlurValue, 40, on_GaussianBlur);
    on_GaussianBlur(g_nGaussianBlurValue, 0);

    //------------------------------------help------------------------------------------
    cout << endl << "\t.请调整滚动条观察图像效果\n\n" << "\t.按下 q ，程序退出！\n";
    while (char(waitKey(1)) != 'q') {}
    return 0;

}

//------------------------------回调函数------------------------------------------------

//on_Boxfilter()函数
static void on_BoxFilter(int, void *)
{
    boxFilter(g_srcImage, g_dstImage1, -1, Size(g_nBoxFilterValue + 1, g_nBoxFilterValue + 1));
    imshow("【<1>方框滤波】", g_dstImage1);
}

//on_MeanBlur()函数
static void on_MeanBlur(int, void *)
{
    blur(g_srcImage, g_dstImage2, Size(g_nMeanBlurValue + 1, g_nMeanBlurValue + 1), Point(-1,
-1));
    imshow("【<2>均值滤波】", g_dstImage2);
}

//on_GaussianBlur()函数
static void on_GaussianBlur(int, void *)
{
    GaussianBlur(g_srcImage, g_dstImage3, Size(g_nGaussianBlurValue * 2 + 1,
g_nGaussianBlurValue * 2 + 1), 0, 0);
    imshow("【<3>高斯滤波】", g_dstImage3);
}
```

形态学滤波

腐蚀与膨胀

功能认知

- 消除噪声
- 分割（isolate）出独立的图像元素，在图像中连接（join）相邻的元素
- 寻找图像中的明显的极大值区域或极小值区域
- 求出图像的梯度
- 腐蚀和膨胀都是针对高亮区域而言的：通过与核卷积求出最大（小）值并赋给指定区域

核心API函数

膨胀：dilate函数

```
void dilate(
InputArray src,                         //输入Mat类对象
OutputArray dst,                        //目标图像，需要有和原图像同样的尺寸和类型
InputArray kernel,                      //膨胀操作的核。NULL表示参考点位于中心3X3的核
Point anchor=Point(-1,-1),              //锚的位置，默认值（-1，-1）表示锚位于中心
int iterations=1,                       //迭代使用函数的次数，默认值为1
int borderType=BORDER_CONSTANT,         //边界模式，默认值BORDER_DEFAULT
const Scalar& borderValue=morphlogyDefaultBerderValue );
```

腐蚀：erode函数

```
void erode(
InputArray src,                         //输入Mat类对象
OutputArray dst,                        //目标图像，需要有和原图像同样的尺寸和类型
InputArray kernel,                      //腐蚀操作的核。NULL表示参考点位于中心3X3的核
Point anchor=Point(-1,-1),              //锚的位置，默认值（-1，-1）表示锚位于中心
int iterations=1,                       //迭代使用函数的次数，默认值为1
int borderType=BORDER_CONSTANT,         //边界模式，默认值BORDER_DEFAULT
const Scalar& borderValue=morphlogyDefaultBerderValue );
```

取核：getStructuringElement

该函数返回指定形状和尺寸的结构元素（内核矩阵），配合这个参数 InputArray kernel；

一般在调用erode以及dilate函数之前，先定义一个Mat类型的变量来获得各条getStructuringElement函数的返回值

```
int getStructElementSize=3;     //内核矩阵的尺寸
//获取自定义核
Mat element=getStructuringElement（
MORPH_RECT,       //内核形状：矩形MORPH_RECT;MORPH_CROSS交叉型;MORPH_ELLIPSE椭圆形
Size（2*getStructElementSize+1,2*getStructElementSize+1）,//内核尺寸
Point(getStructElementSize,getStructElementSize)）;      //锚点位置
```

开运算与闭运算、梯度、顶帽、黑帽

功能认知

开运算（Opening Operation）

先腐蚀后膨胀：dsp=open(src,element)=dilate(erode(src,element))

开运算可以用来消除小物体，在纤细处分离物体（放大裂缝或者局部低亮度区域）；在平滑较大物体的边界的同时不改变其面积

闭运算（Closing Operation）

先膨胀后腐蚀：dsp=close(src,element)=erode(dilate(src,element))

闭运算可以排除小型黑洞（黑色区域）

形态学梯度（Morphological Gradient）

形态学梯度是膨胀图与腐蚀图之差，突出边缘，可以保留物体的边缘轮廓。

顶帽（Top Hat）

原图与开运算结果图之差

开运算放大了裂缝或者局部低亮度的区域，所以，从原图中减去开运算后的图，得到的结果突出了比原图轮廓周围的区域更明亮的区域，这个操作与选择的核的大小有关。**TopHat运算一般用来分离比邻近点亮一些的斑块，可以使用这个运算提取背景。**

黑帽运算（Black Hat）

闭运算的结果与原图之差

黑帽运算的结果突出了比原图轮廓周围区域更暗的区域，所以**黑帽运算用来分离比邻近点暗一些的斑块。**

核心API函数

morphologyEx()

```
void morphologyEx(
InputArray src,
OutputArray dst,
int op,              //形态学运算的类型
InputArraykernel,
Pointanchor=Point(-1,-1),
intiterations=1,
intborderType=BORDER_CONSTANT,
constScalar& borderValue=morphologyDefaultBorderValue() );
```

MORPH_OPEN 开运算

MORPH_CLOSE 闭运算

MORPH_GRADIENT 形态学梯度

MORPH_TOPHAT 顶帽

MORPH_BLACKHAT 黑帽

使用范例

```
#include <opencv2/opencv.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

Mat srcIM, dstIM;   //原始图和效果图
int ES = MORPH_RECT;  //元素结构形状

int MaxNm = 10;
int OCNm = 0;
int EDNm = 0;
int TBNm = 0;

static void on_OC(int, void*);
static void on_ED(int, void*);
static void on_TB(int, void*);



int main()
{
    srcIM = imread("1.jpg");
    if (!srcIM.data)
    {
        printf("读取srcIM错误！\n");
        return 0;
    }

    namedWindow("原始图");
    imshow("原始图", srcIM);

    namedWindow("开/闭",1);
    namedWindow("腐蚀/膨胀", 1);
    namedWindow("顶帽/黑帽", 1);


    OCNm = 9;
    EDNm = 9;
    TBNm = 2;

    createTrackbar("迭代值", "开/闭", &OCNm, MaxNm * 2 + 1, on_OC);
    createTrackbar("迭代值", "腐蚀/膨胀", &EDNm, MaxNm * 2 + 1, on_ED);
    createTrackbar("迭代值", "顶帽/黑帽", &TBNm, MaxNm * 2 + 1, on_TB);

    //轮询获取按键信息

    while (1)
    {
        int c;


        //执行回调
```

```
        on_OC(OCNm, 0);
        on_ED(EDNm, 0);
        on_TB(TBNm, 0);

        //获取按键
       c=waitKey(0);

        //按下按键Q或者 ESC，程序退出
        if ((char)c == 'q' || (char)c == 27)
            break;
        //按下按键1，使用 MORPH_ELLIPSE
        else if ((char)c == 49)
            ES = MORPH_ELLIPSE;

        else if ((char)c == 50)
            ES = MORPH_RECT;

        else if ((char)c == 51)
            ES = MORPH_CROSS;
        else if ((char)c == ' ')
            ES = (ES + 1) % 3;




    }



    return 0;
}



static void on_OC(int, void*)
{
    //偏移量定义
    int offset = OCNm - MaxNm;
    int Absolute_offset = offset > 0 ? offset : -offset;

    //自定义核
    Mat element = getStructuringElement(ES, Size(Absolute_offset * 2 + 1, Absolute_offset * 2 +
1), Point(Absolute_offset, Absolute_offset));

    //执行操作
    if (offset < 0)
        morphologyEx(srcIM, dstIM, MORPH_OPEN, element);
    else morphologyEx(srcIM, dstIM, MORPH_CLOSE, element);

    imshow("开/闭", dstIM);

}
```

```
static void on_ED(int, void*)
{
    //偏移量定义
    int offset = EDNm - MaxNm;
    int Absolute_offset = offset > 0 ? offset : -offset;

    //自定义核
    Mat element = getStructuringElement(ES, Size(Absolute_offset * 2 + 1, Absolute_offset * 2 +
1), Point(Absolute_offset, Absolute_offset));

    //执行操作
    if (offset < 0)
        erode(srcIM, dstIM,element);
    else dilate(srcIM, dstIM,element);

    imshow("腐蚀/膨胀", dstIM);
}

static void on_TB(int, void*)
{
    //偏移量定义
    int offset = TBNm - MaxNm;
    int Absolute_offset = offset > 0 ? offset : -offset;

    //自定义核
    Mat element = getStructuringElement(ES, Size(Absolute_offset * 2 + 1, Absolute_offset * 2 +
1), Point(Absolute_offset, Absolute_offset));

    //执行操作
    if (offset < 0)
        morphologyEx(srcIM, dstIM, MORPH_TOPHAT, element);
    else morphologyEx(srcIM, dstIM, MORPH_BLACKHAT, element);

    imshow("顶帽/黑帽", dstIM);
}
```