天津理工大学

计算机科学与工程学院

实验报告

2017 至 2018 学年 第 二 学期

实验四 图像的平滑处理

课程名称	数字图像处理				
学号	20152180	学生姓名	王帆	年级	2015
专业	计算机 科学与技术	教学班号	2	实验地点	主 7-212
实验时间	2018 年 4 月 16 日 第 7 节 至 第 8 节				
主讲教师	杨淑莹				

实验成绩

软件运行	效果	算法分析	流程设计	报告成绩	总成绩

实验（四）	实验名称	图像的平滑处理
软件环境	Windows Visual Studio 2017	
硬件环境	PC	

实验目的

实现图像的平滑处理。

实验内容（应包括实验题目、实验要求、实验任务等）

设计并实现两种图像的平滑处理。

要求：了解图像的平滑基本原理，实现图像的平滑。
说明：平滑基本原理
任务：
（1）在左视图中打开一幅位图。
（2）制作两个【图像的平滑】菜单，将消息映射到右视图中，在右视图中实现图像的简单平滑。

实验过程与实验结果

设计并实现两种图像的平滑处理

1) 均值滤波

原理：

均值滤波是将原图中一个像素的灰度值和它周围邻近 8 个像素的灰度值相加，然后将求得的平均值（除以 9）作为新图中该像素的灰度值。它采用模板计算的思想，模板操作实现了一种邻域运算，即某个像素点的结果不仅与本像素灰度有关，而且与其邻域点的像素值有关。邻域平均处理方法是以图像模糊为代价来减小噪声的，且模板尺寸越大，噪声减小的效果越显著。如果 f（i，j）是噪声点，其邻近像素灰度与之相差很大，采用邻域平均法就是用邻近像素的平均值来代替它，这样能明显削弱噪声点，使邻域中灰度接近均匀，起到平滑灰度的作用。因此，邻域平均法具有良好的噪声平滑效果，是最简单的一种平滑方法

实现步骤：

1.获取原图像的 Bitmap 对象 objBitmap，以及其大小参量；
2.获取 N 值；
3.构造一个新 Bitmap 对象 bitmap，迭代实现对 objBitmap 各点像素值的获取；
4.算出以该点为中心的 N×N 屏蔽窗口内平均值；
5.将该点像素值置为平均值，并迭代赋值到 bitmap 内；
6.使用 bitmap 构造全局变量 curBitmap，销毁 bitmap 对象，使用 curBitmap 初始化右侧显示框。

代码：

```
//COMUtil.cs, namespace DIP.Public
//平滑处理相关操作
public static int[] init_templt(int size)
{
    int square = size * size;
    int[] template = new int[square];
    for (int i = 0; i < square; i++)
    {
        template[i] = 1;
    }

    return template;
}
public static int[] Templatable(int x, int y, int[] template, Bitmap bitmap,
int neighborhood_size, bool flag)
{
    int[] m = new int[3];

    int width = bitmap.Width;
    int height = bitmap.Height;
    int px, py;

    //彩色
    if (flag == true)
    {
        int[] t = new int[3] { 0, 0, 0 };
        for (int i = 0; i < neighborhood_size; i++)
        {
            for (int j = 0; j < neighborhood_size; j++)
            {
                py = y - neighborhood_size / 2 + i;
                px = x - neighborhood_size / 2 + j;
                Color color = bitmap.GetPixel(px, py);
                int r = color.R;
                t[0] += r * template[i * neighborhood_size + j];
                int g = color.G;
                t[1] += g * template[i * neighborhood_size + j];
                int b = color.B;
                t[2] += b * template[i * neighborhood_size + j];
            }
        }
        for (int i = 0; i < m.Length; i++)
```

```
        {
            m[i] = t[i];
        }

        return m;
    }
    //黑白
    else if (flag == false)
    {
        int t = 0;

        for (int i = 0; i < neighborhood_size; i++)
        {
            for (int j = 0; j < neighborhood_size; j++)
            {
                py = y - neighborhood_size / 2 + i;
                px = x - neighborhood_size / 2 + j;
                Color color = bitmap.GetPixel(px, py);
                int s = (color.R + color.G + color.B) / 3;
                t += s * template[i * neighborhood_size + j];
            }
        }
        for (int i = 0; i < m.Length; i++)
        {
            m[i] = t;
        }
        return m;
    }
    else
    {
        for (int i = 0; i < m.Length; i++)
        {
            m[i] = 0;
        }
        return m;
    }
}
public static int operation_simple(int r, int size)
{
    r /= (size * size);
    r = r > 255 ? 255 : r;
    r = r < 0 ? 0 : r;
    return r;
}
```

```
//选项：平滑处理-均值滤波
private void ToolStripMenuItem_smooth_ave_Click(object sender, EventArgs e)
{
    try
    {
        aveForm avefrm = new aveForm();
        avefrm.ShowDialog();
        if (avefrm.flag)
        {
            int size = Convert.ToInt32(avefrm.textBox_value.Text);
            int neighborhood_size = (size - 1) / 2;
            int width = objBitmap.Width;
            int height = objBitmap.Height;
            Bitmap bitmap = new Bitmap(width, height);
            int[] map = new int[size];
            int[] template = COMUtil.init_templt(size);

            for (int y = neighborhood_size; y < height - neighborhood_size;
y++)
            {
                for (int x = neighborhood_size; x < width - neighborhood_size;
x++)
                {

                    map = COMUtil.Templatable(x, y, template, objBitmap, size,
ave
frm.color);
                    bitmap.SetPixel(x, y,
Color.FromArgb(COMUtil.operation_simple(map[0], size), COMUtil.operation_si
mple(map[1], size), COMUtil.operation_simple(map[2], size)));
                }
            }
            curBitmap = new Bitmap(bitmap);
            bitmap.Dispose();
            this.pictureBox_new.Image = curBitmap;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBox
Icon.Stop);
    }
}
```

效果图：

图 1-1 均值滤波前-彩色图像-模板基数 3

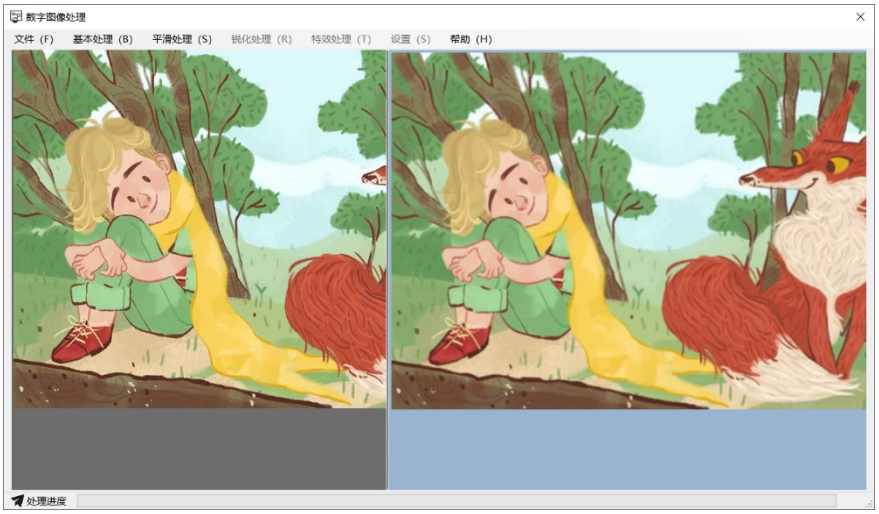

图 1-2 均值滤波后-彩色图像-模板基数 3

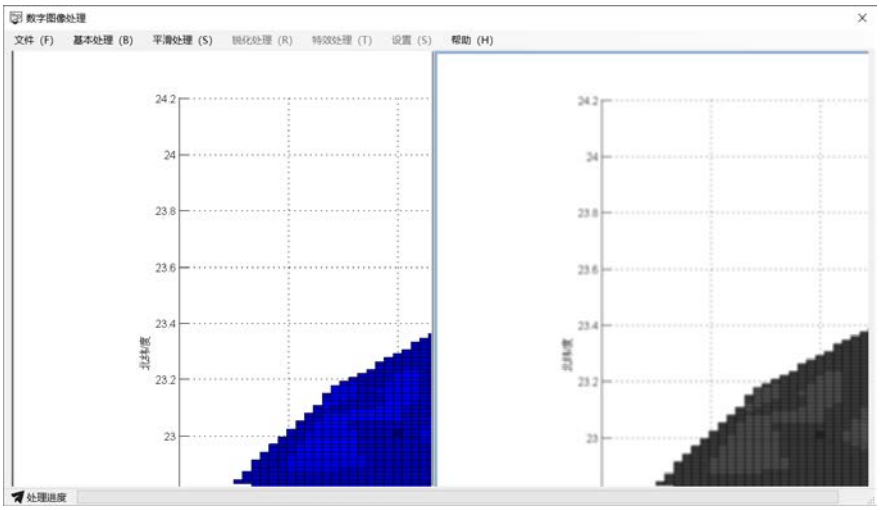

图 1-2 均值滤波后-黑白图像-模板基数 5

2) 高斯滤波

原理：

高斯滤波器是一种线性平滑滤波器，其滤波器的模板是对二维高斯函数离散得到。由于高斯模板的中心值最大，四周逐渐减小，其滤波后的结果相对于均值滤波器来说更好。

高斯滤波器最重要的参数就是高斯分布的标准差 σ，标准差和高斯滤波器的平滑能力有很大的能力，σ 越大，高斯滤波器的频带就较宽，对图像的平滑程度就越好。通过调节 σ 参数，可以平衡对图像的噪声的抑制和对图像的模糊。

在三维计算机视觉领域，对于二维图像的特征抽取是关键的第一步，这主要包括抽取二维图像上的边缘、角点、纹理等。如果能够较好的从二维图像中提取出这些信息，那么对于三维重建，物体定位，空间监控等后期目标能够有很好的支撑作用。而这些含有图像轮廓以及空间位置信息的点、线、面等特征，在图像上都体现为灰度值的不连续或者剧烈变化。也就是说，如果我们能够根据图像灰度矩阵找到灰度不连续的点位置，那么也就是实现了特征抽取。

在二维函数中，梯度向量代表着函数的最大变化率方向，因此对于二维信号，可以采用其梯度向量的模来选取灰度值不连续点。这样在理想情况下，我们就可以用离散化的梯度逼近函数来检测图像灰度矩阵的灰度跃变位置，从而实现特征抽取。

现实中，有摄像机获取的图像，往往都存在噪声，而且信号并不存在理想的阶跃畸变，这样如果依然直接采用拉普拉斯算子进行灰度跃变检测，那样会产生很多的虚假特征点。因此，往往在图像处理之前，需要对灰度图进行滤波处理。

线性平滑就是对每一个像素点的灰度值用他的邻域值来代替，其邻域大小为 N*N, N 一般为奇数。经过线性平滑滤波，相当于图像经过了一个二维的低通滤波器，虽然是降低了噪音，但是同时也模糊了图像的边缘和细节，这是这类滤波器存在的通病。

下面给出高斯平滑函数：

$$h(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

该函数各向同性，其曲线是草帽状的对称图，该曲线对整个覆盖面积求积分为高斯滤波的思路就是：对高斯函数进行离散化，以离散点上的高斯函数值为权值，对我们采集到的灰度矩阵的每个像素点做一定范围邻域内的加权平均，即可有效消除高斯噪声。

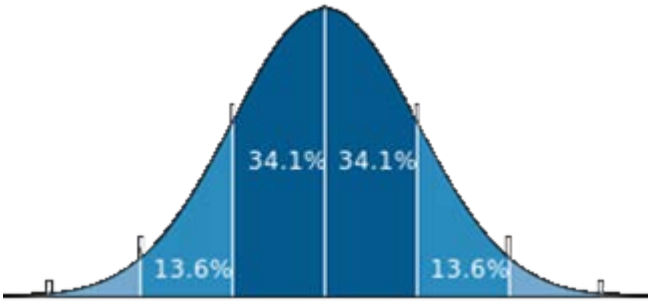

图 2-1 一维高斯分布的概率分布密度图

实现步骤：

1.获取原图像的 Bitmap 对象 objBitmap，以及其大小参量；
2.构造一个新 Bitmap 对象 bitmap，迭代实现对 objBitmap 各点像素值的获取；
3.初始化高斯内核模板（Kernel）；
4.将该点像素值与高斯内核进行卷积运算，并迭代赋值到 bitmap 内；
5.使用 bitmap 构造全局变量 curBitmap，销毁 bitmap 对象，使用 curBitmap 初始化右侧显示框。

代码：

```
//选项：平滑处理-高斯滤波
private void ToolStripMenuItem_smooth_gauss_Click(object sender, EventArgs e)
{
    try
    {
        int width = objBitmap.Width;
        int height = objBitmap.Height;
        int[] m = new int[3];
        int neighborhood_size = 3;
        Bitmap bitmap = new Bitmap(width, height);
        for (int y = neighborhood_size / 2; y < height - neighborhood_size /
2; y++)
        {
            for (int x = neighborhood_size / 2; x < width - neighborhood_size
/ 2; x++)
            {
                m = COMUtil.Templatable(x, y, COMUtil.init_gauss(), objBitmap,
neighborhood_size, true);
                bitmap.SetPixel(x, y,
Color.FromArgb(COMUtil.operation_simple(m[0], 4), COMUtil.operation_simple(m
[1], 4), COMUtil.operation_simple(m[2], 4)));
            }
        }
        curBitmap = new Bitmap(bitmap);
        bitmap.Dispose();
        this.pictureBox_new.Image = curBitmap;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK, Message
BoxIcon.Stop);
    }
}
//高斯卷积核初始化
public static int[] init_gauss()
{
```

```
    int[] template = new int[9] { 1, 2, 1, 2, 4, 2, 1, 2, 1 };
    return template;
}
```

效果图：

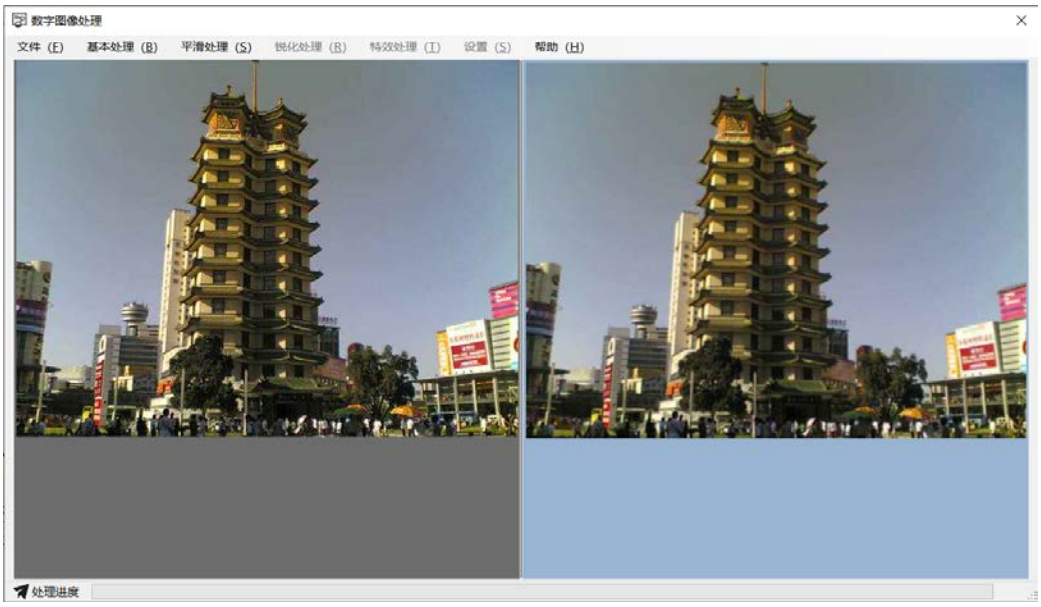

图 2-2 高斯滤波处理

附录

参考资料：

1.《数字图像处理 Visual Studio C++技术实现》杨淑莹，张桦，陈胜勇/著；

2.《C#程序设计》王贤明等 编著；

3. 图像处理之均值滤波介绍及 C 算法实现 - CSDN 博客

https://blog.csdn.net/a8039974/article/details/78069068

4. 关于高斯滤波的一些理解 - CSDN 博客

https://blog.csdn.net/lz0499/article/details/54015150