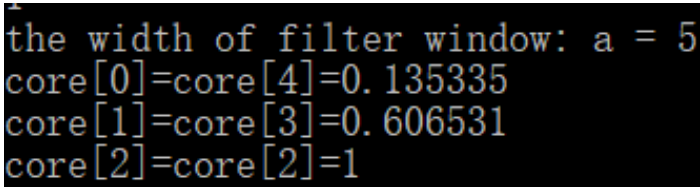


计算机视觉 课程实验报告

学 号 : 201600181073	姓名: 唐超	班级: 16 人工智能
实验题目: 图像滤波		
<p>实验内容:</p> <h3>1. 高斯滤波</h3> <p>a) 通过调整高斯函数的标准差(sigma)来控制平滑程度; <code>void Gaussian(const input, output, double sigma);</code> b) 滤波窗口大小取为$[6 \times \text{sigma} - 1]$, $[.]$表示取整; c) 利用二维高斯函数的行列可分离性进行加速;</p> <h3>2. 快速均值滤波</h3> <p>a) 滤波窗口大小通过参数来指定; b) 采用积分图进行加速, 实现与滤波窗口大小无关的效率;</p>		
<p>实验过程中遇到和解决的问题:</p> <h3>高斯滤波</h3> <h4>1. 计算一维高斯滤波核</h4> <p>为避免图像失真, 一般滤波核不会太大, 这里定义一个长度为 11 的一维数组 <code>core[11]</code> 来存放一维高斯滤波核的值。</p> <p>由高斯分布的对称性知, 滤波核内关于中心点对称的两个点的值相等, 可以一起计算。由于我们只关心高斯滤波核内各点值的比例, 高斯函数前面的系数并不重要, 可以不用计算。</p> <pre>double core[11]; //计算一维高斯滤波核 for (int k = 0; k < (a + 1) / 2; k++) { core[k] = core[a - 1 - k] = exp(-pow((k - r), 2) / (2 * sigma*sigma)); }</pre> <p>当窗口大小为 5 时, core 如下:</p>  <pre>the width of filter window: a = 5 core[0]=core[4]=0.135335 core[1]=core[3]=0.606531 core[2]=core[2]=1</pre> <h4>2. 对行列先后进行高斯滤波</h4> <p>新定义一个 Mat 存储行高斯滤波后的图像, 但需要注意的是此时只是开辟</p>		

一个 Mat 大小的存储空间，里面可能原先存有数值，在后面累加的时候可能会对滤波结果产生影响，因此需要将其每个像素点初始化为 0。

行高斯滤波的循环体如下：

```
dst0.at<Vec3b>(i, j)[0] += src.at<Vec3b>(i, j + p - r)[0] * core[p] / s;  
dst0.at<Vec3b>(i, j)[1] += src.at<Vec3b>(i, j + p - r)[1] * core[p] / s;  
dst0.at<Vec3b>(i, j)[2] += src.at<Vec3b>(i, j + p - r)[2] * core[p] / s;
```

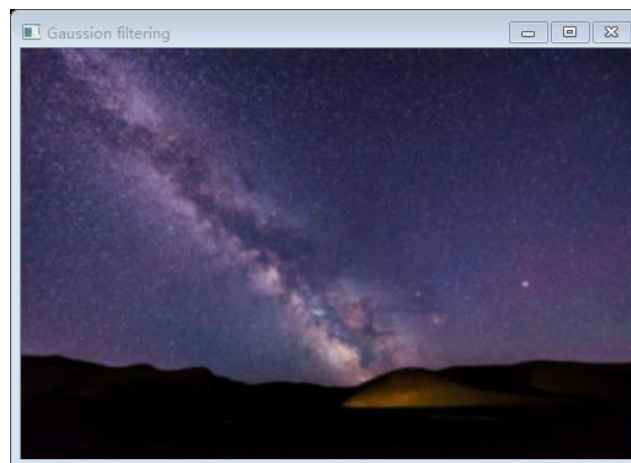
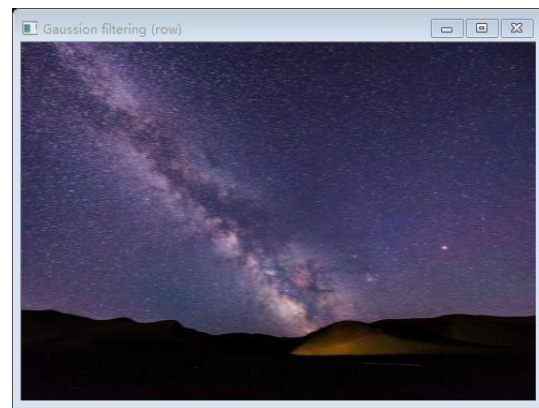
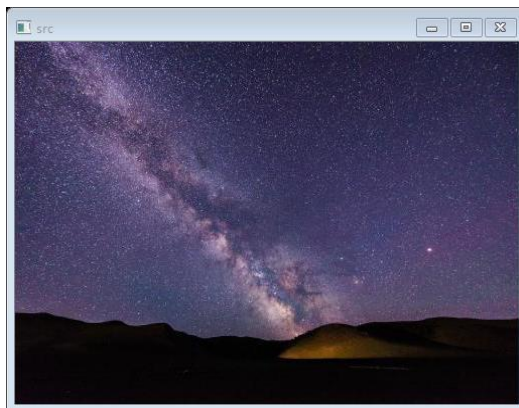
其中 p 为滤波核里点的索引， $r = (a-1)/2$, s 为滤波核的和。

行滤波完成后，再新建一个 Mat，同样的方法再对列方向进行滤波。

3. 边界处理

在行高斯滤波时，对于边界附近的像素点，落在图像外部的滤波核对应的像素点用关于列坐标 j 对称的像素点代替进行计算；列高斯滤波时同理。

Sigma=1 时，高斯滤波结果如下：



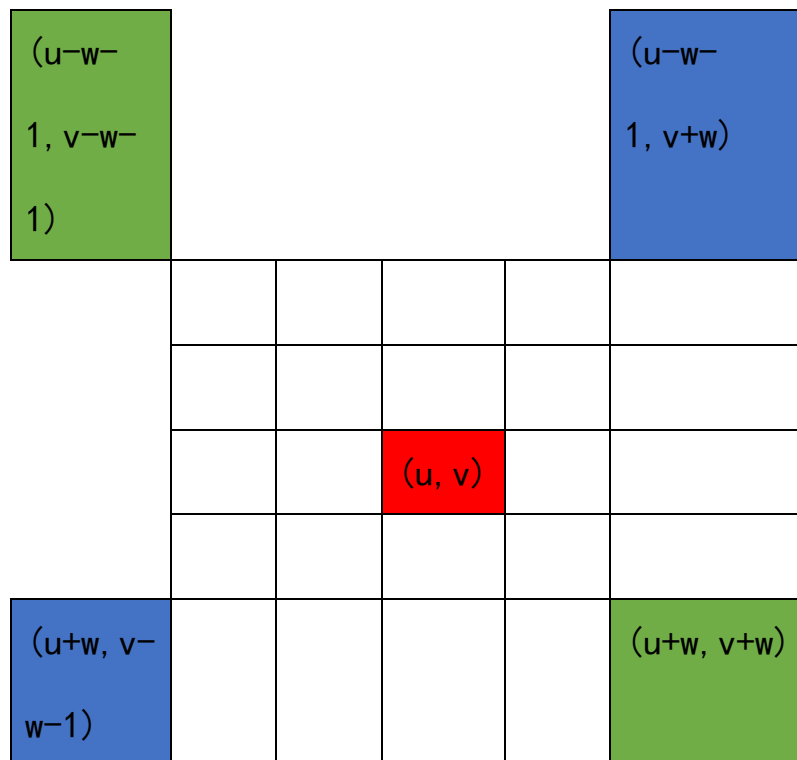
快速均值滤波

1. 边界扩展

在图像的边界附近，某一像素点的滤波核会部分落至图外，这里根据积分图的原理，利用 `copyMakeBorder` 函数

```
void copyMakeBorder(src, dst, int top, int bottom, int left, int right, int borderType)
```

填充类型设为 `IPL_BORDER_REFLECT`，以图像边界为对称轴，反射填充，分别向上、左扩充 $w+1$ ，向下、右扩充 w （假设滤波宽度为 a ， $w=(a-1)/2$ ）；



当滤波窗口大小设为 5 时，图片形状大小变化情况如下：

```
C:\Users\hp\source\repos\x64\Debug\testOpencv.exe
please enter the width(odd number) of filter window:
5
原图大小:[500 x 334]
扩展之后:[505 x 339]
滤波之后最终的图像[500 x 334]
```

2. 积分图

积分图上每个像素点的值是原图像该点左上角（包括自身）所有点像素值的加和，因此用原图的 `CV_8UC3` 势必会造成溢出，将积分图的类型改为 `CV_32FC3`。

```
Mat integrogram(src1.size(), CV_32FC3, Scalar(0.0f, 0.0f, 0.0f));
```

积分图可以增量计算，原本的做法是，从左到右按列扫描，第一列每一点的像素值是原图其上方的加和，之后每一列某点的像素值是其左边点的像素值加上原图该列上方所有点像素值的加和，后者直接用 `for` 循环计算，后来发现这种算法运行效率较低；

新的方法是扫描两遍，方向都是从左到右从上到下，第一遍在列方向上积分，得到的图像每个像素点的值是原图同位置其上方所有像素点值的和，第二

遍在第一遍得到的“列”积分图的基础上，将每个像素点的值加上其左边像素点的值作为该像素点的新值，扫描结束即得到积分图；

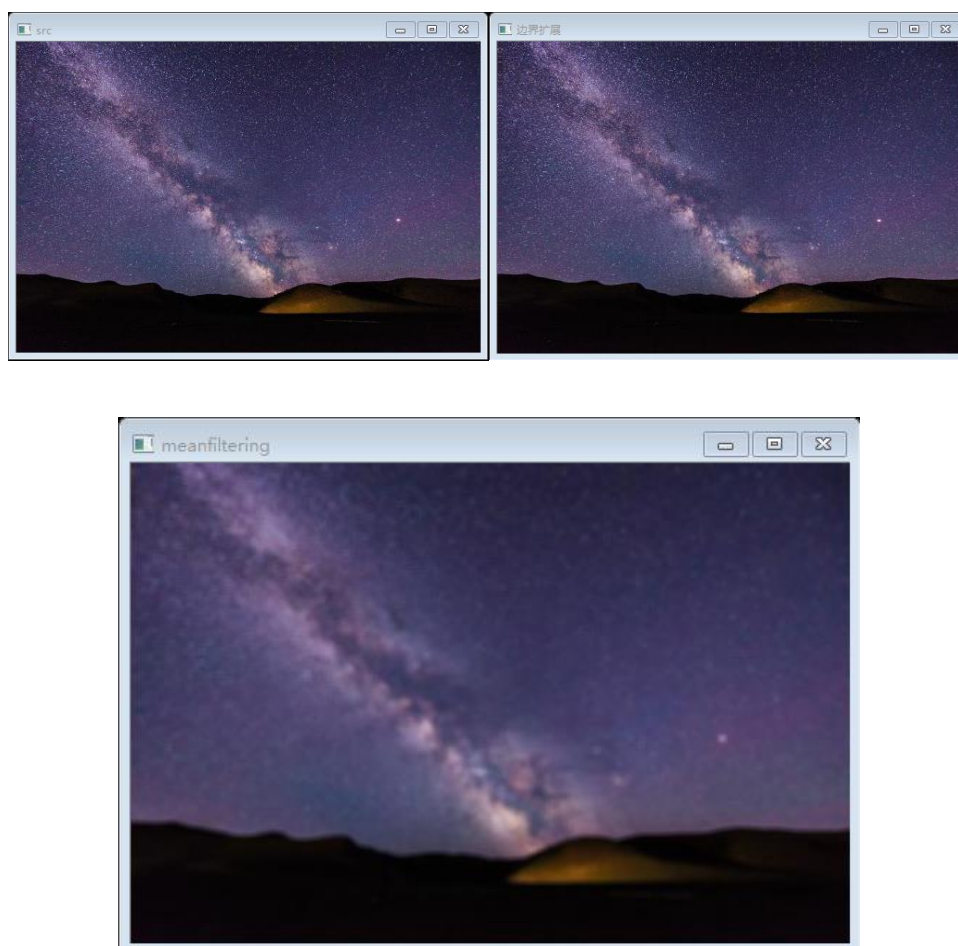
第二种没有在每个像素点再次嵌套循环，运行效率大大提高。（第一种方法运行一次大概 20 多秒，第二种大概 2、3 秒）

3. 应用积分图进行均值滤波

由于积分图(integrogram)相对于原图(src)进行了边界扩展，滤波后图像(dst)的大小不应改变，因此 dst 中点(u, v) 应对应于扩展后图像(src1)的点(u+w+1, v+w+1)。

$$\text{dst.at}\langle\text{Vec3b}\rangle(u, v)[k] = (\text{S.at}\langle\text{Vec3f}\rangle(u + 2*w+1, v + 2*w+1)[k] + \text{S.at}\langle\text{Vec3f}\rangle(u, v)[k] - \text{S.at}\langle\text{Vec3f}\rangle(u + 2*w+1, v)[k] - \text{S.at}\langle\text{Vec3f}\rangle(u, v + 2*w+1)[k]) / z;$$

最终滤波结果如下：



结论分析与体会：

在做写很多图像行列坐标时太大意，运行时经常报像素点超出了图像的范围，还有又把条件判断的等号写成赋值号了，自己应该更加细心认真才是。

在计算积分图时，稍微改变了一下思路，使得程序效率提高了很多，以后也应该对一个问题多想想其他可能的解决办法。