姓名：刘权祥

学号：2019300414

第二次作业

> 角点检测与图像匹配

任务一：Harris角点检测

内容

数据：棋盘图片
要求：自己写函数实现Harris角点检测子，设置不同参数，比较检测结果

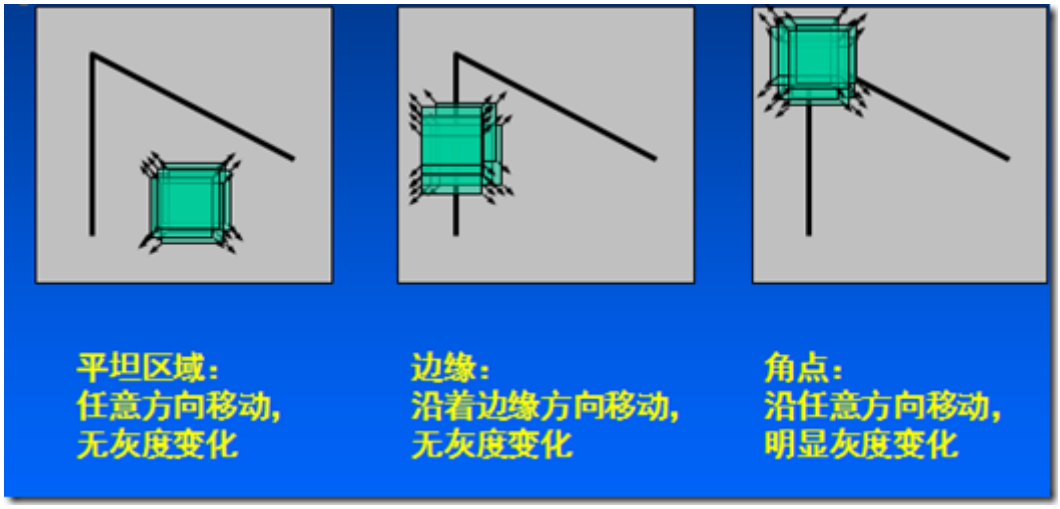
其中 边缘检测子：sobel检测子

参数包括，窗口大小和检测阈值

原理

0、Harris角点检测的基本思想：

算法基本思想是使用一个固定窗口在图像上进行任意方向上的滑动，比较滑动前与滑动后两种情况，窗口中的像素灰度变化程度，如果存在任意方向上的滑动，都有着较大灰度变化，那么我们可以认为该窗口中存在角点。



1、灰度变化描述

当窗口发生$[u,v]$移动时，那么**滑动前与滑动后对应的窗口中的像素点灰度变化**描述如下：

$$E(u,v)=\sum_{(x,y)\epsilon W} w(x,y)[I(x+u,y+v)-I(x,y)]^2$$

其中

- $[u,v]$是窗口W的偏移量；
- (x,y)是窗口W所对应的像素坐标位置，窗口有多大，就有多少个位置；
- $I(x,y)$是像素坐标位置(x,y)的图像灰度值；
- $I(x+u,y+v)$是像素坐标位置$(x+u,y+v)$的图像灰度值；
- $w(x,y)$是窗口函数，最简单情形就是窗口W内的所有像素所对应的w权重系数均为1.但有时候，我们会将$w(x,y)$函数设置为以窗口W中心为原点的二元正太分布。如果窗口W中心点是角点时，移动前与移动后，该点在灰度变化贡献最大；而离窗口W中心(角点)较远的点，这些点的灰度变化几近平缓，这些点的权重系数，可以设定小值，以示该点对灰度变化贡献较小，那么我们自然而然想到使用二元高斯函数来表示窗口函数；

根据上述表达式，当窗口在平坦区域上移动灰度不会发生什么变换。如果窗口处在纹理比较丰富的区域上滑动，那么灰度变化会很大。

算法最终思想就是计算灰度发生较大变化时所对应的位置，当然这个较大是指任意方向上的滑动，并非单指某个方向。

2、$E(u,v)$化简

根据泰勒公式:

$$f(x+u,y+v)\approx f(x,y)+uf_x(x,y)+vf_y(x,y)$$

我们可以将上面的公式化简为:

$$
\begin{aligned}
&\sum_{(x,y)\epsilon W} w(x,y)[I(x+u,y+v)-I(x,y)]^2 \\
\approx &\sum_{(x,y)\epsilon W} w(x,y)[I(x,y)+uI_x+vI_y-I(x,y)]^2 \\
= &\sum_{(x,y)\epsilon W} w(x,y)[u^2I_x^2+2uvI_xI_y+v^2I_y^2] \\
= &\sum_{(x,y)\epsilon W} w(x,y)\begin{bmatrix}u & v\end{bmatrix}\begin{bmatrix}I_x^2 & I_xIy\\ I_xI_y & I_y^2\end{bmatrix}\begin{bmatrix}u\\ v\end{bmatrix} \\
= &\begin{bmatrix}u & v\end{bmatrix}(\sum_{(x,y)\epsilon W} w(x,y)\begin{bmatrix}I_x^2 & I_xIy\\ I_xI_y & I_y^2\end{bmatrix})\begin{bmatrix}u\\ v\end{bmatrix}
\end{aligned}
$$

所以$E(u,v)$表达式可以更新为：

$$
E(u,v)=\begin{bmatrix}u & v\end{bmatrix}M\begin{bmatrix}u\\ v\end{bmatrix}
$$

其中：$M=\sum_{(x,y)\epsilon W} w(x,y)\begin{bmatrix}I_x^2 & I_xIy\\ I_xI_y & I_y^2\end{bmatrix}$,

I_x,I_y分别为窗口内像素点(x,y)在x方向上和y方向上的梯度值。

3、矩阵M的关键性

Harris角点检测并不是直接根据$E(u,v)$来判断角点的，他很好的利用了矩阵M。

我们可以把$E(u,v)$近似为二项函数：$E(u,v)=Au^2+2Cuv+Bv^2$

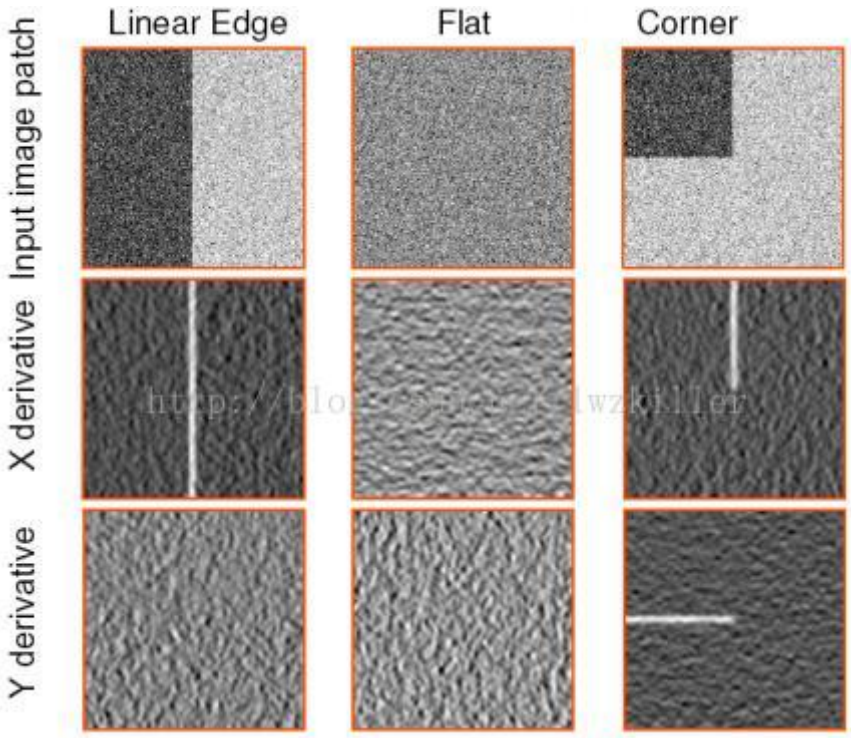
其中：

$$
\begin{aligned}
A&=\sum_{(x,y)\epsilon W} w(x,y)*I_x^2\\
B&=\sum_{(x,y)\epsilon W} w(x,y)*I_y^2\\
C&=\sum_{(x,y)\epsilon W} w(x,y)*I_xI_y
\end{aligned}
$$

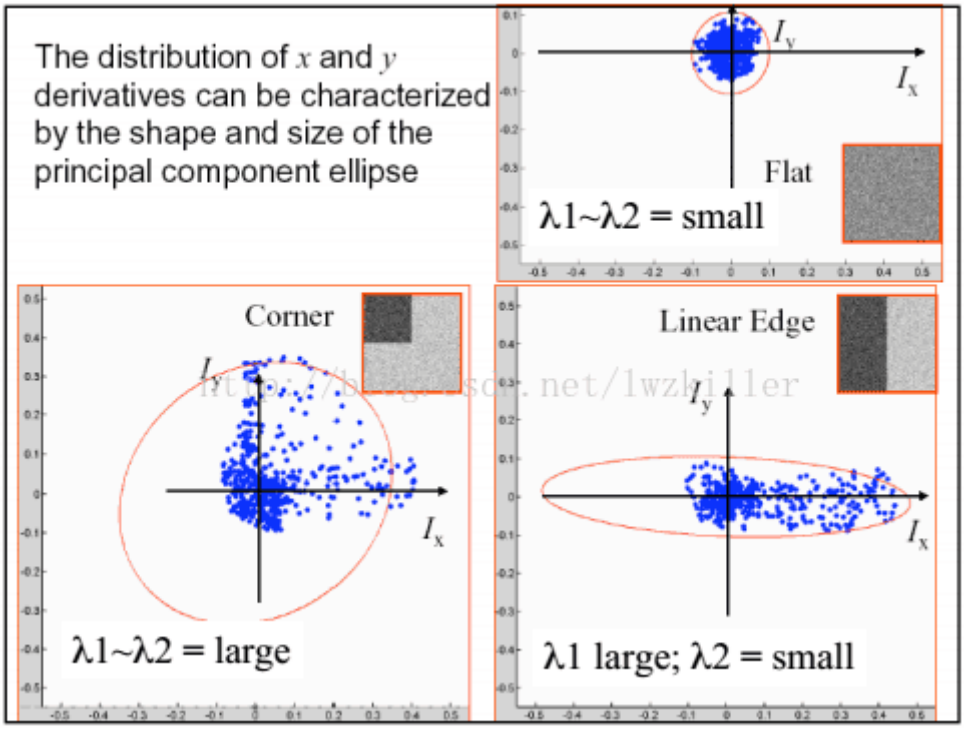
二次项函数本质上就是一个椭圆函数。椭圆的长和宽是由M的特征值$\lambda1,\lambda2$决定的(椭圆的长短轴正是矩阵M特征值平方根的倒数)，椭圆的方向是由M的特征向量决定的，椭圆方程为：

$$
\begin{bmatrix}u & v\end{bmatrix}M\begin{bmatrix}u\\ v\end{bmatrix}=1
$$

针对平坦区域，边缘区域以及角点区域三种情形进行分析：



下图是对这三种情况窗口中的对应像素的梯度分布进行绘制：



从上图中我们可以看出：

- 如果$\lambda 1, \lambda 2$都很大，并且很接近，就可以认为是一个角点
- 如果$\lambda 1, \lambda 2$很接近但是数值比较小，那么这里就是一个平坦区域
- 如果$\lambda 1, \lambda 2$中只有一个值大，那么这只是一个边缘，不是角点

4、角点响应的度量

通常用下面表达式进行度量，对每一个窗口计算得到一个分数R，根据R的大小来判定窗口内是否存在harris特征角。分数R根据下面公式计算得到：

$$R = det(M) - k(trace(M))^2$$
$$det(M) = \lambda_1 \lambda_2$$
$$trace(M) = \lambda_1 + \lambda_2$$

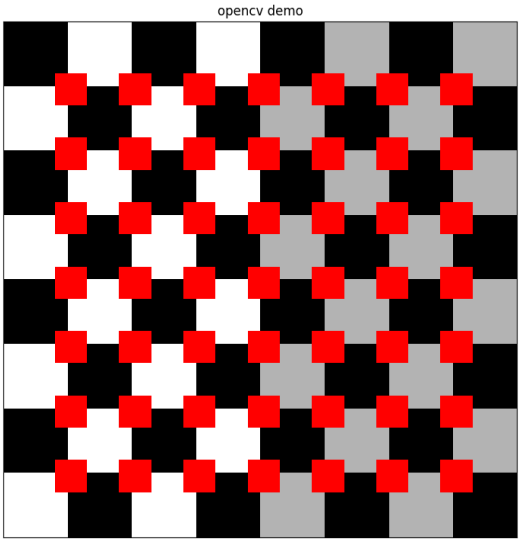
这里$\lambda 1, \lambda 2$是矩阵M的2个特征值，k是一个指定值，这是一个经验参数，需要实验确定它的合适大小，通常它的值在0.04和0.06之间,它的存在只是调节函数的形状而已。

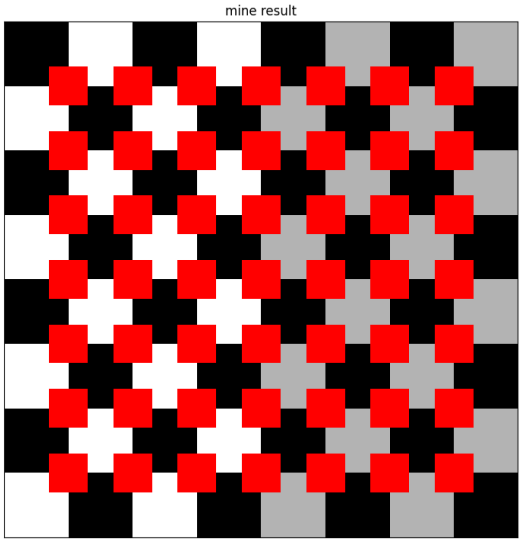
R取决于M的特征值，对于角点$|R|$很大，平坦的区域$|R|$很小，边缘的R为负值;

最后设定R的阈值，进行角点判断即可。

结果

对棋盘格的检测

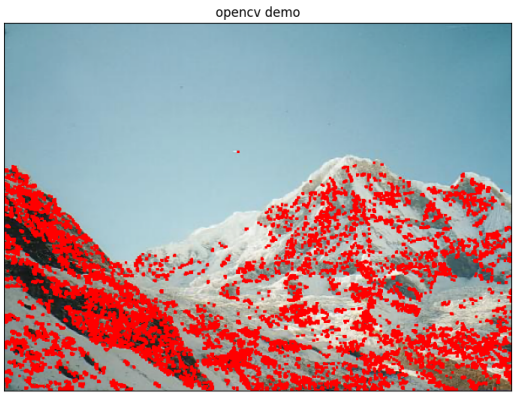


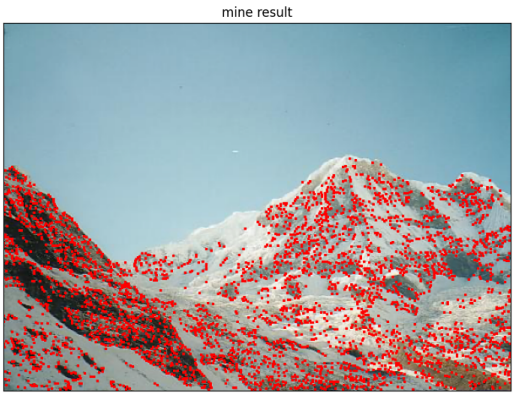


由于棋盘格图片太小，角点计算的结果看不太出来，就不展示了。

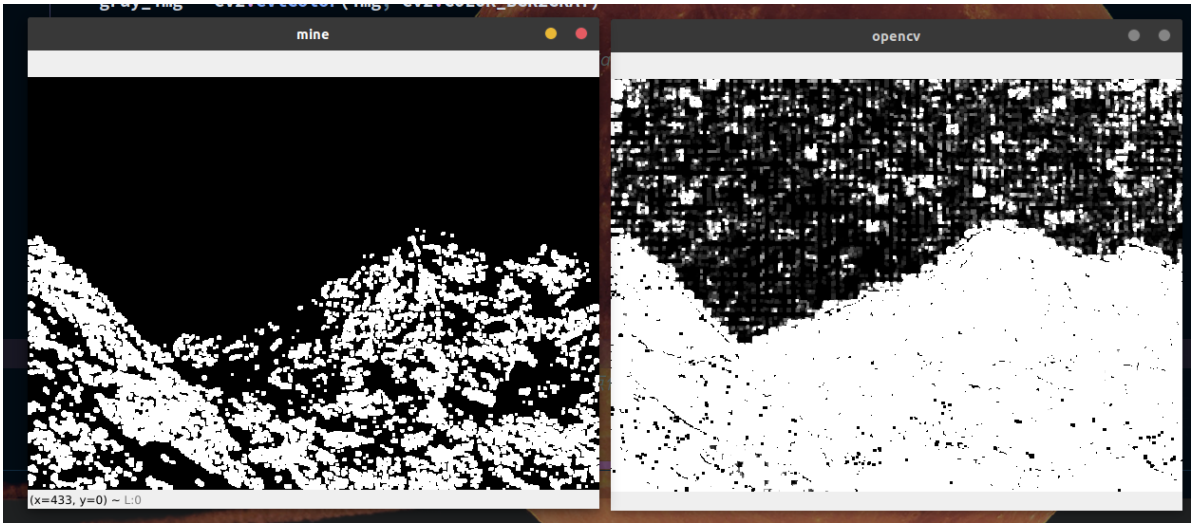
尝试了对山脉图进行角点检测：

对山脉图片的检测





其中，对山脉图的角点计算结果如下图所示：



分析总结

从上面的结果看到，我实现的Harris角点检测基本还是能把角点给检测出来的，并且在棋盘图上面的结果与OepnCV的方法一样，而在山脉图上面，大部分角点也是和OpenCV是一样的。

而在山脉图中直接计算出来的结果就和OpenCV的差的大一些。

任务二：SIFT算法

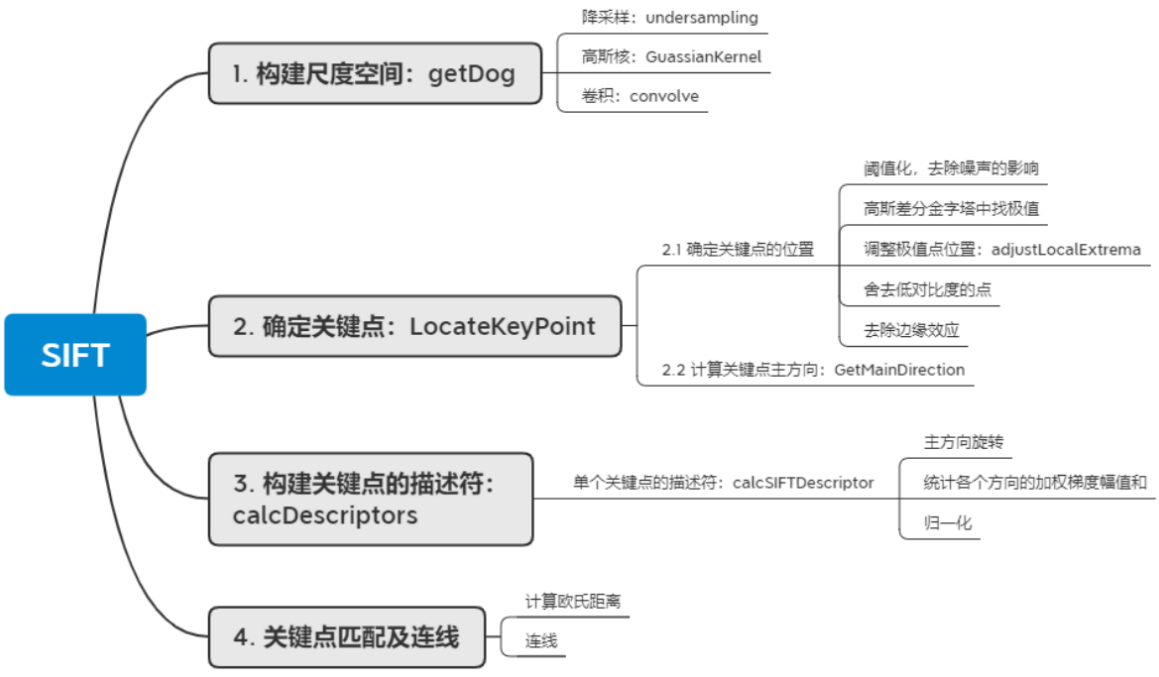
内容

数据：山脉图片1和图片2
要求：使用SIFT算法实现两幅山脉图片的匹配
1.尝试自己写函数实现SIFT算法
2.调节算法参数，进行匹配结果对比
3.与调用库函数结果进行对比

原理

> 这里花了非常多的时间，主要还是参考别人的教程和代码一步一步实现的。

0、SIFT算法的主要流程



1、构建尺度空间

构建尺度空间是初始化操作,后续计算特征都需要在高斯金字塔与差分高斯金字塔上计算。
二维图像的尺度空间定义为:

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

其中 G 是二维高斯函数:

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

I 是灰度图像。将图像经过高斯卷积后，就可以得到高斯滤波后的图像，并且以σ

为第三维（尺度坐标），就构成了二维图像的尺度空间。

σ大小决定图像的平滑程度，大尺度对应图像的概貌特征，小尺度对应图像的细节特征。因此，金字塔越靠上其σ 越大（因为低分辨率需要用大σ 来产生更多信息）。

高斯差分尺度空间（DOG scale-space）的定义如下：

$$\begin{aligned} D(x,y,\sigma) &= (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) \\ &= L(x,y,k\sigma) - L(x,y,\sigma). \end{aligned}$$

即：将每一层（octave）上两个相邻的高斯图像进行差分，就可以得到一个高斯差分组

金字塔每一层的σ 与 k定义如下

$$2^{i-1}(\sigma, k\sigma, k^2\sigma, \cdots k^{n-1}\sigma), \quad k = 2^{1/s}$$

2、确定关键点

1. 确定关键点的位置

- 阈值化

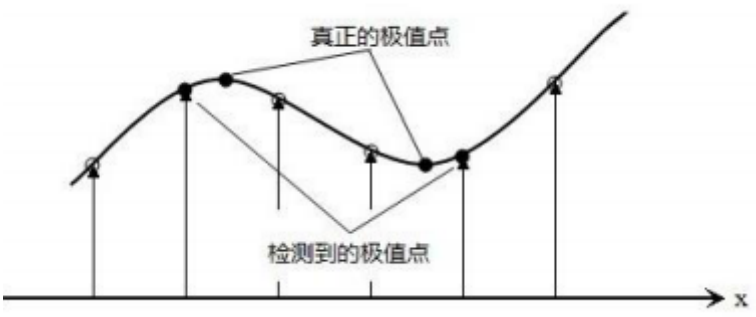
  过小的点易受噪声的干扰而变得不稳定，所以将小于某个经验值(contrastThreshold=0.04)的极值点删除。

- 在高斯差分金字塔中找极值点

  每一个采样点要和它所有的相邻点比较，即与同尺度的8个相邻点和上下相邻尺度对应的9×2个点共26个点比较。如果该点在DOG尺度空间本层以及上下两层的26个领域中是最大或最小值时，则认为该点是图像在该尺度下的一个极值点。

- 调整极值点的位置

  离散空间的极值点并不是真正的极值点，下图显示了二维函数离散空间得到的极值点与连续空间极值点的差别。为了提高关键点的稳定性，需要对尺度空间DoG函数进行曲线拟合。



- 舍去低对比度的点

  当f(X)的绝对值小于T/n时，舍去该点X。

- 去除边缘效应

一个定义不好的高斯差分算子的极值在横跨边缘的地方有较大的主曲率，而在垂直边缘的方向有较小的主曲率。主曲率通过一个2×2 的Hessian矩阵H求出:

$$H(x,y)=\begin{bmatrix} D_{xx}(x,y) & D_{xx}(x,y) \\ D_{xy}(x,y) & D_{yy}(x,y) \end{bmatrix}$$

导数由采样点相邻差估计得到。
D的主曲率和H的特征值成正比，令α为较大特征值，β为较小的特征值，则

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$
$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

令α=γβ，则

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

$(r+1)^2/r$的值在两个特征值相等的时候最小，随着r的增大而增大，因此，为了检测主曲率是否在某域值r（Lowe论文中，取r=10）下，只需检测

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$$

至善

2. 计算关键点的主方向

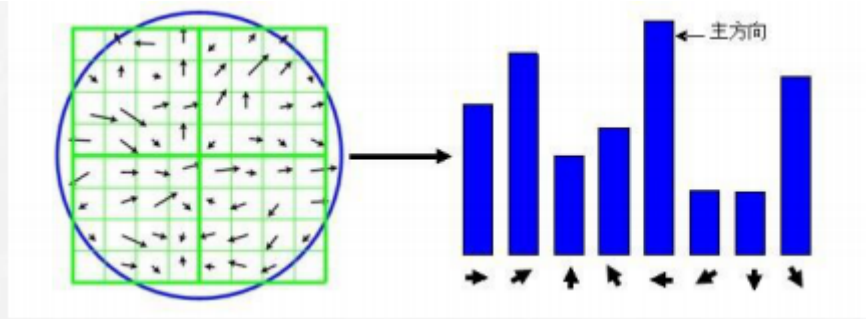
为了使描述符具有旋转不变性，需要利用图像的局部特征为给每一个关键点分配一个方向。

利用关键点邻域像素的梯度及方向分布的特性，可以得到梯度模值和方向如下：

$$m(x,y)=\sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2}$$
$$\theta(x,y)=\tan^{-1}((L(x,y+1)-L(x,y-1))/L(x+1,y)-L(x-1,y)))$$

在以关键点为中心的邻域窗口内采样，并用直方图统计邻域像素的梯度方向。

直方图的峰值则代表了该关键点处邻域梯度的主方向，即作为该关键点的方向。



3、构建关键点的描述符

1. 将坐标轴旋转到关键点的主方向，以确保旋转不变性。

$$\begin{pmatrix} x' \\ y' \end{pmatrix}=\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}$$

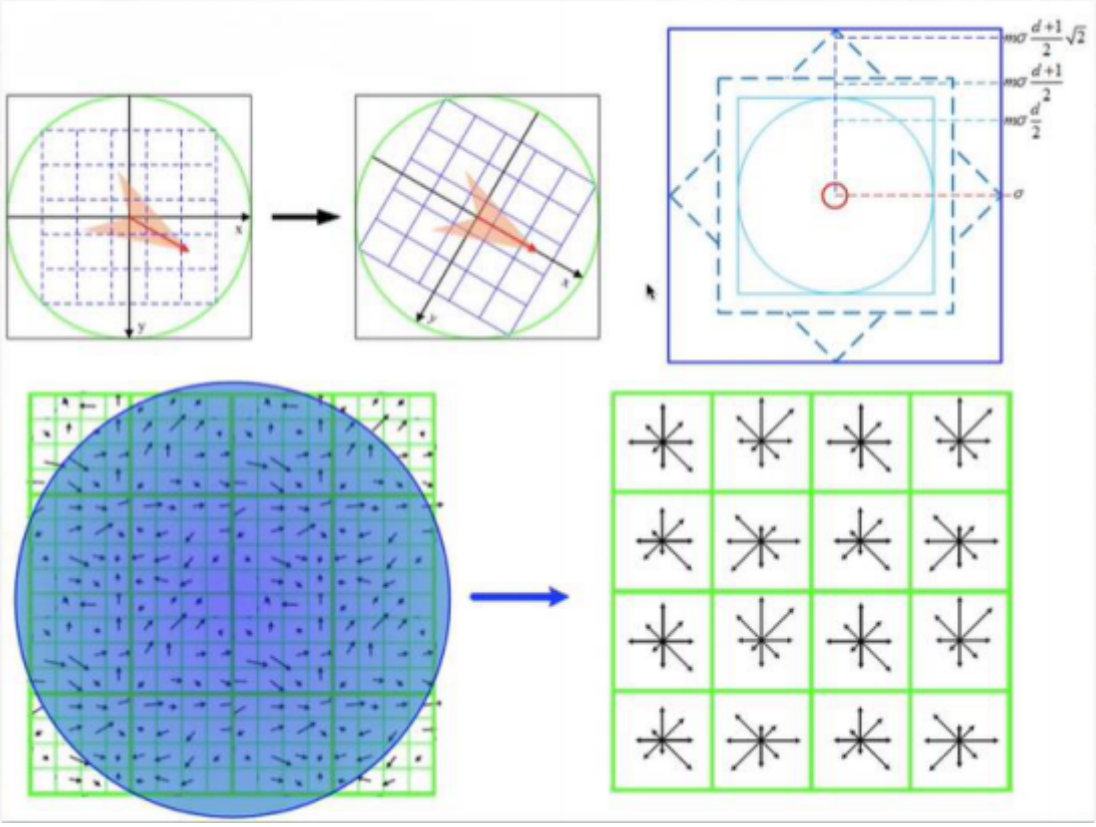
2. 把关键点的周围区域划分为4*4=16个子区域，统计每个子区域里的每个像素的梯度方向和梯度幅值，方法和确定主方向时类似。最终得到每个子区域8个方向上高斯加权的梯度幅值和，所以整个区域，即每个关键点得到一个16*8=128维的描述向量。

区域大小按以下公式确定：

半边长 = 1.414*mσ(d+1)/24

其中，m取3，d为划分每行子区域数量，取4

3. 将特征向量的长度归一化，可以进一步去除光照变化的影响。

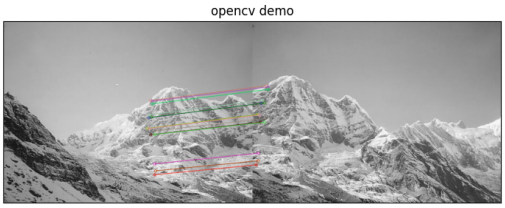


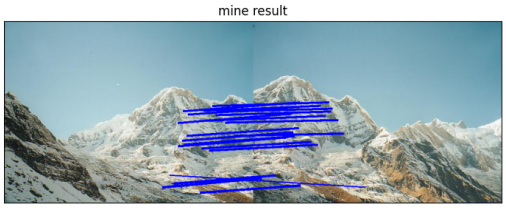
4、关键点匹配及连线

通过前面几步可以计算出图片的关键点和描述符，这里再使用欧氏距离找出匹配的关键点评价匹配程度，最后再进行连线。

结果

最终的结果如下图所示





分析总结

感觉SIFT算法比Harris难实现多了。Harris几乎都是自己写的，但SIFT有一些地方实在是不太会写，所以参考了一些资料。一开始是想参考OpenCV源代码的，但是python的源码跳进去就是一个定义，函数具体内容只是“pass”。后面去找C++的源码，但点进去发现继承了一些东西，跳来跳去也不好找（可能没找对方法）。所以去找了网上的一些实现。

整个过程不敢说把SIFT的原理都搞清楚了，不过这个SIFT的流程还是弄懂了，里面一些关键的步骤也搞清楚了，感觉挺有挑战性的。

从最后的结果看出来，这种算法的精确率还是很高的，基本两张图片之间的相同的物体能成功匹配上。但是OpenCV实现的SIFT跑得很快，不过我实现的SIFT要跑很长时间，应该有挺多地方要优化的。