全景图拼接

中山大学计算机学院 徐立瀚 22336265

实验目的

1、 熟悉 Harris 角点检测器的原理和基本使用
2、 熟悉 RANSAC 抽样一致方法的使用场景
3、 熟悉 HOG 描述子的基本原理

实验内容

1. 使用 Harris 角点检测器寻找关键点。
2. 构建描述算子来描述图中的每个关键点，比较两幅图像的两组描述子，并进行匹配。
3. 根据一组匹配关键点，使用 RANSAC 进行仿射变换矩阵的计算。
4. 将第二幅图变换过来并覆盖在第一幅图上，拼接形成一个全景图像。
5. 实现不同的描述子，并得到不同的拼接结果。

一，使用 Harris 角点检测器寻找关键点

Harris 角点检测器的主要原理是找到像素局部区域内各个方向都变化较大的像素点，这些像素点被称为角点，并且通常而言这些角点所包含的局部特征较为显著

为了找到这些角点，我们需要以下步骤:

1，计算出各个像素点关于像素中心的水平与垂直方向梯度值

2，用泰勒展开将 1 中式子近似为以下表达式:

$$M=\begin{bmatrix}\sum I_x^2 & \sum I_xI_y\\ \sum I_xI_y & \sum I_y^2\end{bmatrix}=\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}$$

3，使用 sobel 算子计算出水平与垂直梯度 Ix 和 Iy，并计算出 Ix^2，Iy^2 与 Ix*Iy

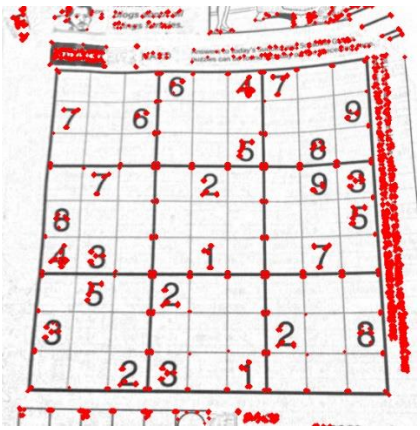
4，使用高斯低通滤波器对上述梯度进行处理，使梯度平滑

5，构造角点表示方程，该方程的特性使得它对角点的响应最大，对边缘的响应次之

$$\theta=\det[M(\sigma_I,\sigma_D)]-\alpha[\mathrm{trace}(M(\sigma_I,\sigma_D))]^2$$
$$=g(I_x^2)g(I_y^2)-[g(I_xI_y)]^2-\alpha[g(I_x^2)+g(I_y^2)]^2$$

6，设置一个阈值，用于筛选出角点方程所得结果较大的点作为目标角点

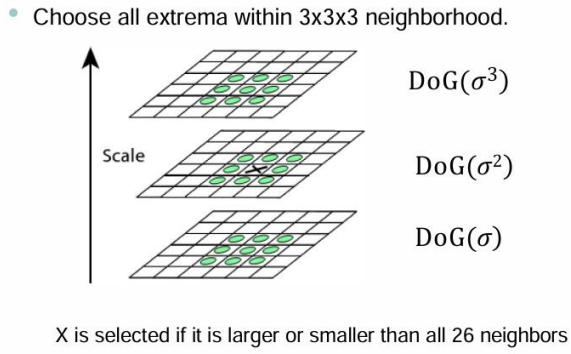
在代码实现中，把角点的坐标作为函数的返回值进行返回。为了方便观察角点的选取而从更好地设置超参数（阈值），我还实现了在图像上将角点绘制出来的函数，具体的表现如下:



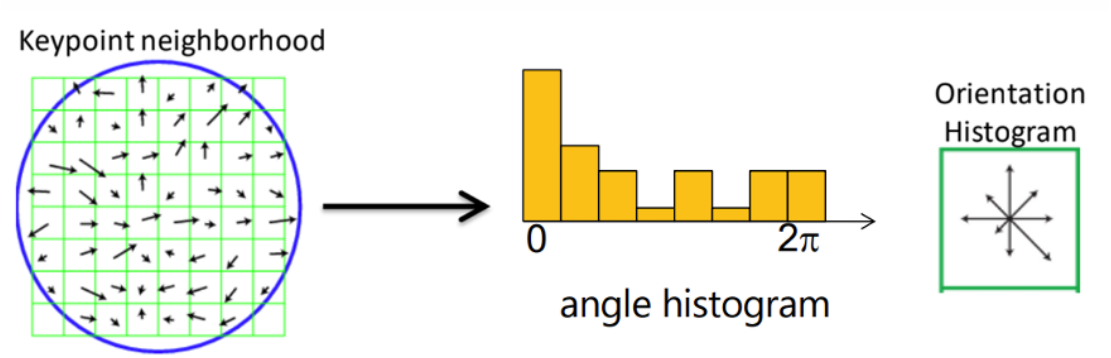
可见，大多数的角点都被合理地标识了出来，包括数独中较大数字的边角，以及周围的小字母单词。至此，Harris 角点检测算法实现完毕。

二，构造 SIFT 描述子对角点特征进行提取：

找到了关键点之后还有几个需要解决的问题，首先 Harris 角点检测对尺度和旋转没有不变性，所以需要实现角点检测尺度和旋转的不变性；其次需要对每个筛选出的关键点计算出一个用于表示关键点的局部特征的向量。



首先解决尺度不变性的问题，对输入图像进行多次高斯模糊，生成不同尺度的图像。然后，计算相邻尺度图像之间的差分，形成 DoG 金字塔。在 DoG 金字塔中，每个像素点与周围 8 个邻域点和上下相邻尺度的 9 个邻域点进行比较，如果该点是局部极值（最大值或最小值），则被初步认定为候选关键点。



其次解决旋转不变性的问题，在关键点周围邻域内，计算每个像素点的梯度幅值和方向。然后，根据梯度方向直方图统计邻域内梯度方向的分布情况。直方图的峰值方向被定义为关键点的主方向。如果存在多个显著的峰值（强度超过主峰值的 80%），则会为关键点分配多个方向，从而生成多个关键点描述子。

最后，根据上述两个方法，以关键点为中心，取一个 4×4 的网格区域，每个网格内计算 8 个方向的梯度直方图，形成一个 4×4×8 的三维数组。将该数组展平为一个 128 维的向量，即为关键点的描述子。这个描述子具有尺度不变性和旋转不变性，能够很好地描述关键点的局部特征。其中每个关键点都会有一个特征向量，将所有向量组合成为一个统一的

数组，并返回该数组。

在具体的代码实现中，我使用了 openCV 函数库中的 SIFT 类来进行调用实现。

三，构造 HOG 描述子对角点特征进行提取：

HOG 描述子的原理十分简单，计算的复杂度也相对较低。首先取关键点的一个局部区域，计算该区域中每个小块的梯度方向和大小，并将所有梯度映射到(0,π)内的 9 个方向上，形成一个方向梯度直方图。该直方图对应的向量就能够对特征进行表示。

在具体的代码实现中，我使用了 openCV 函数库中的 HOGDescriptor 类来进行调用实现。

四，根据欧式距离对特征向量进行匹配

计算两个特征向量之间的欧式距离，从而得到两个向量偏移的差值，向量距离差值越小，匹配越精确。因此，在匹配时设置一个匹配阈值，只有距离差值小于这个阈值才进行匹配，否则放弃该匹配。为了让阈值的设计更加合理，我选择记录匹配中的最大距离，只有当前匹配的距离与最大距离的差值小于最大距离乘阈值百分比才能记录该匹配。这样能使不同大小的特征向量在不进行归一化的前提下都能适配该阈值。

在具体的实现中，我使用了 openCV 函数库中的 BFMatcher 类来进行调用实现，直接实现匹配并输出合适的匹配结果。在匹配完成后，使用 openCV 中的 drawMatches 函数即可将两幅图像之间关键点的匹配情况绘制出来，其结果大致如下：

SIFT 特征提取的匹配：

HOG 特征提取的匹配：

可以看出，SIFT 的匹配相对比较杂乱，说明匹配错误的项较多，而 HOG 的匹配看起来更加整齐且有序，说明正确匹配的频率更高。

五，根据匹配情况使用 RANSAC 算法计算仿射变换矩阵

想要实现全景图拼接实际上使用 3 组匹配的关键点就可以进行实现，3 组关键点匹配正确，那么整体的图像相对位置与旋转变形情况就确定下来了。然而我们在上述步骤中大概率产生了不止一组的匹配，甚至有一些是错误匹配，因此我们使用 RANSAC 算法来处理这些匹配信息。

其基本的思想是随机地取 3 组匹配的关键点，以这 3 组关键点作为匹配的标准，用这个匹配方式对其他的匹配进行验证，倘若大部分的匹配都能在这个匹配标准下成功匹配，那么这 3 组匹配关键点就能够代表图像的匹配情况。这些成功匹配的点称作内点，而在这个匹配标准下无法匹配的点称作外点。

因此，我们的优化目标就变成了寻找内点最多的 3 组匹配。由于匹配的个数其实并不多，所以可以采用随机组合，然后统计内点占比，每次迭代只记录内点占比最大的匹配组合，最后便找出了最合适的 3 组匹配。

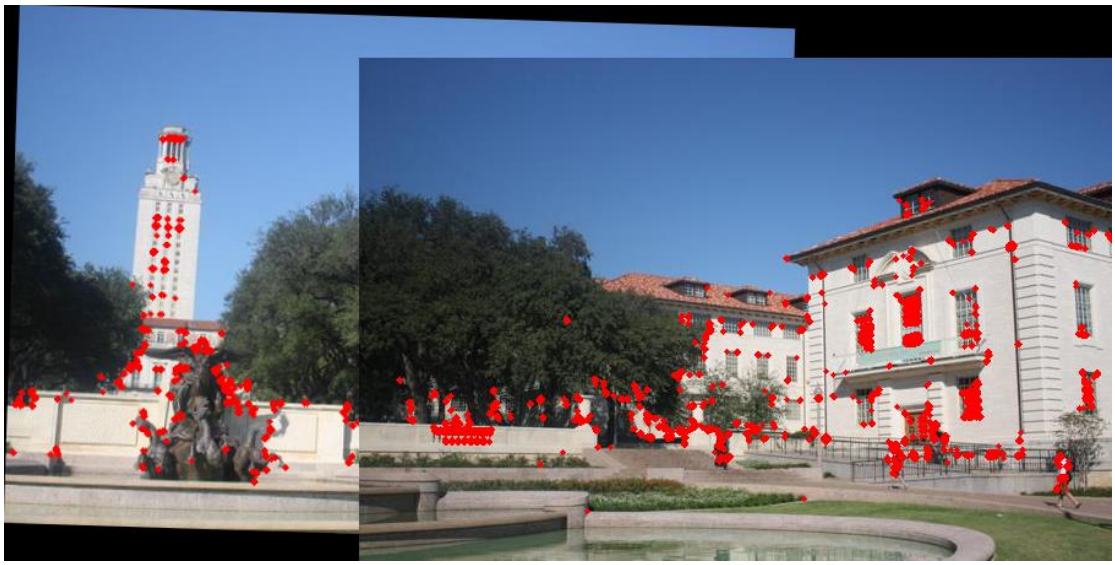
根据这 3 组匹配再使用矩阵变换，求出图像的仿射变换矩阵从而实现算法。

六，根据仿射变换矩阵进行全景图拼接

在有了仿射变换矩阵之后，剩下的工作就只剩下图像矩阵的处理了，这个处理过程大致可以分为以下几个步骤，首先计算变换后的图像边界，从而确定图像的大小情况；其次调整变换矩阵以考虑平移；然后再使用仿射变换将一幅图像变换到另一幅图像的坐标系中；最后将两幅图像合并并裁剪掉黑色边框，得到最终的全景图。

以下即为拼接结果：

SIFT 匹配：



HOG 匹配：

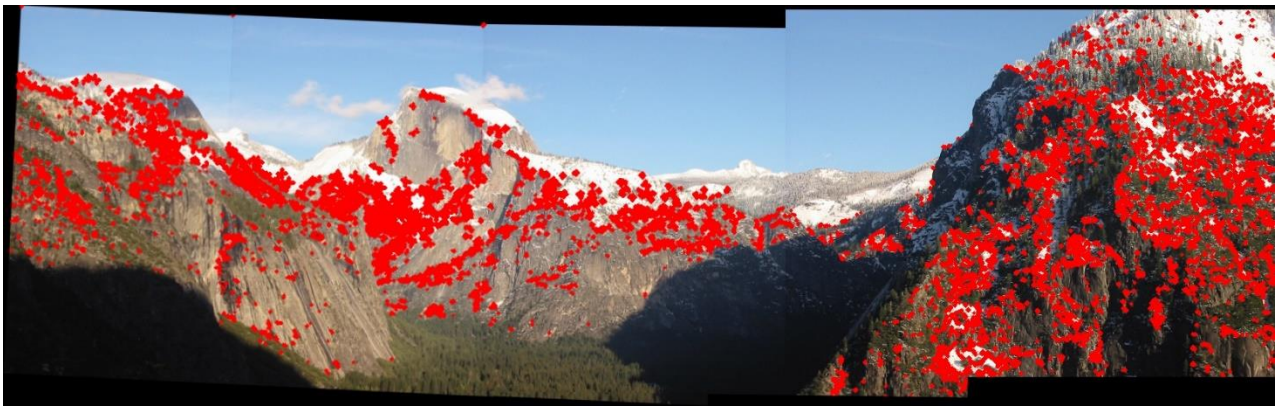
在这个例子中 SIFT 和 HOG 算法对全局图拼接的表现都不错，两幅图像的拼接结果都符合人眼的观感，同时也相差不大。

七，四幅图片的全景图拼接实战

对于多幅图像的全景图拼接，最直接的方法是找到先后顺序，然后依次将图像读入，将上一次已经拼接好的半成品全景图与下一张图片进行拼接，就像火车车厢一样，一节一节地不断拼接。

需要注意的是，拼接的前后两张图片必须要有部分交集，否则可能存在没有任何合适匹配对的情况，会导致算法失效。

最终的拼接成果如下：



拼接非常完好，符合人眼的直觉。