关键点检测及匹配

1. 哈里斯角点检测

该算法的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。如下图所示，一个局部很小的区域，如果是在图片区域中移动灰度值没有变化，那么窗口内不存在角点。如果在某一个方向上移动，一个发生很大变化而另一侧没有变化，那么说明这个区域是位于该对象的边缘区域，也即检测出角点。

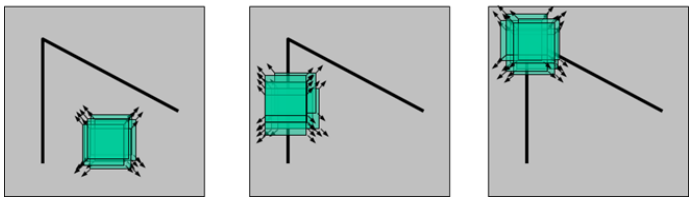

图 1 哈里斯角点检测

哈里斯角点检测算法表达式如下：

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \underbrace{[I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)]^2}_{\text{intensity}}$$

对于图像I(x,y)，在点(x,y)处平移(u,v)后的自相似性。其中w(x,y)是加权函数，它可以是常数，也可以是高斯加权函数。在作业代码中，使用高斯函数作为窗口函数：

```
filtered_ = cv2.filter2D(img, -1, cv2.getGaussianKernel(3, 2))
```

之后，根据泰勒展开和一些数学步骤后可得到如下结果：

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

其中，哈里斯矩阵为，

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

图像的导数使用 Sobel 算子进行计算，

```
dx = cv2.Sobel(filtered_, cv2.CV_64F, 1, 0, ksize=15)
dy = cv2.Sobel(filtered_, cv2.CV_64F, 0, 1, ksize=15)
```

则哈里斯矩阵中各项可计算，

```
Ixy = dx * dy
Ix2 = dx ** 2
Iy2 = dy ** 2
```

泰勒展开忽略余项之后的表达式为一个二项式函数，然而二项式函数的本质上就是一个椭圆函数，椭圆的扁率和尺寸是由M(x,y)的特征值 λ1、λ2 决定的，椭圆的方向是由M(x,y)的特征矢量决定的。椭圆函数特征值与图像中的角点、直线（边缘）和平面之间的关系如下图所示。共可分为三种情况：

a. 图像中的直线。一个特征值大，另一个特征值小，λ1>λ2 或λ2>λ1。自相关函数值在某一方向上大，在其他方向上小。

b. 图像中的平面。两个特征值都小，且近似相等；自相关函数数值在各个方向上都小。

c. 图像中的角点。两个特征值都大，且近似相等，自相关函数在所有方向都增大。

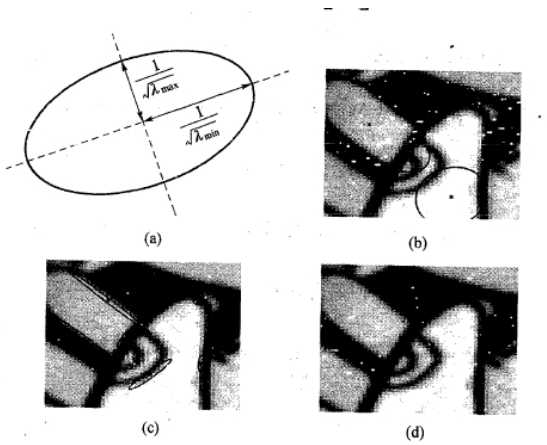

图2. 椭圆函数与图像中的角点、直线和平面之间的关系

由于我们是通过M的两个特征值的大小对图像进行分类，所以，定义角点相应函数 R：

$$R = detM - k(traceM)^2$$

计算所有点的R值后，取R值最大的10000个像素点作为初始角点。再通过非极大值抑制，根据两个角点直接的欧式距离大小来选取其中的1500个角点：

```
N = len(sorted_points)
for i in range(N):
    r_sqaure = 1e10
    point1 = sorted_points[i]
    for j in range(i):
        point2 = sorted_points[j]
        temp = (point1[1] - point2[1]) ** 2 + (point1[2] - point2[2]) ** 2
        r_sqaure = min(temp, r_sqaure)
    radi.append([np.sqrt(r_sqaure), point1[0], point1[1], point1[2]])

points_ = sorted(radi, key=lambda x: x[0], reverse=True)
x = np.array([])
y = np.array([])
for item in points_[:1500]:
    x=np.append(x, [item[2]], axis=0)
for item in points_[:1500]:
    y=np.append(y, [item[3]], axis=0)
```

检测效果如下：

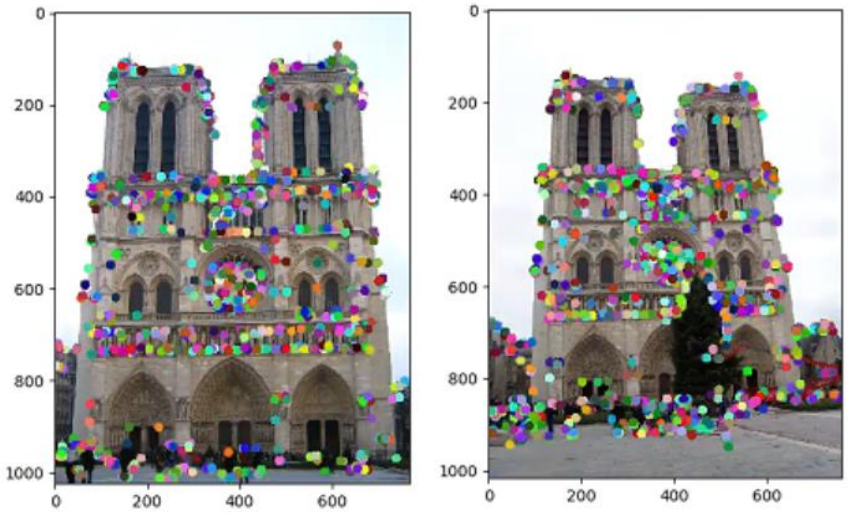

图3. 角点检测效果

2. SIFT 特征提取部分

SIFT特征提取采用简化的方式，其步骤为：

1. 求出上述给出的点的梯度值
2. 在cell内计算梯度直方图
3. 计算相位，按相位将直方图设置为8个区间
4. 统计每个区间内的梯度幅值并作为该cell的SIFT特征
5. 对SIFT特征进行归一化

实现代码如下所示：

```
for k in range(len(x)):
    HG = np.zeros((4, 4, 8))
    for j in range(feature_width):
        for i in range(feature_width):
            bin = np.arctan2(dy[(int)(y[k]) + j][(int)(x[k]) + i], dx[(int)(y[k]) + j][(int)(x[k]) + i])
            mag = np.sqrt(dx[(int)(y[k]) + j][(int)(x[k]) + i] ** 2 + dy[(int)(y[k]) + j][(int)(x[k]) + i] ** 2)
            if bin > 1: bin = 2
            if bin < -1: bin = -1
            if dx[(int)(y[k]) + j][(int)(x[k]) + i] > 0: HG[(int)(j / 4)][(int)(i / 4)][math.ceil(bin + 1)] += mag
            else:HG[(int)(j / 4)][(int)(i / 4)][math.ceil(bin + 5)] += mag
    ft = np.reshape(HG, (1, 128))
    ft = ft / (ft.sum())
    fs.append(ft)
```

3. 特征匹配部分

按作业要求，使用最近邻距离比(Nearest Neighbor Distance Ratio)。匹配点对的位置坐标按照置信度降序排列，默认取前100个置信度最高高的匹配点对进

行评估。代码实现如下：

```
dis = np.zeros((features1.shape[0],features2.shape[0]))
for i in range(features1.shape[0]):
    for j in range(features2.shape[0]):
        dis[i,j] = np.linalg.norm(features1[i] - features2[j])
matches = np.zeros((features1.shape[0], 2))
matches[:,0], matches[:,1]  = np.arange(features1.shape[0]), dis.argmin(axis=1)
dis.sort(axis=1)
confidences = 1 - dis[:,0]/dis[:,1]
sort = np.argsort(1 - confidences)
matches = matches[sort,:]
```

匹配效果如下：

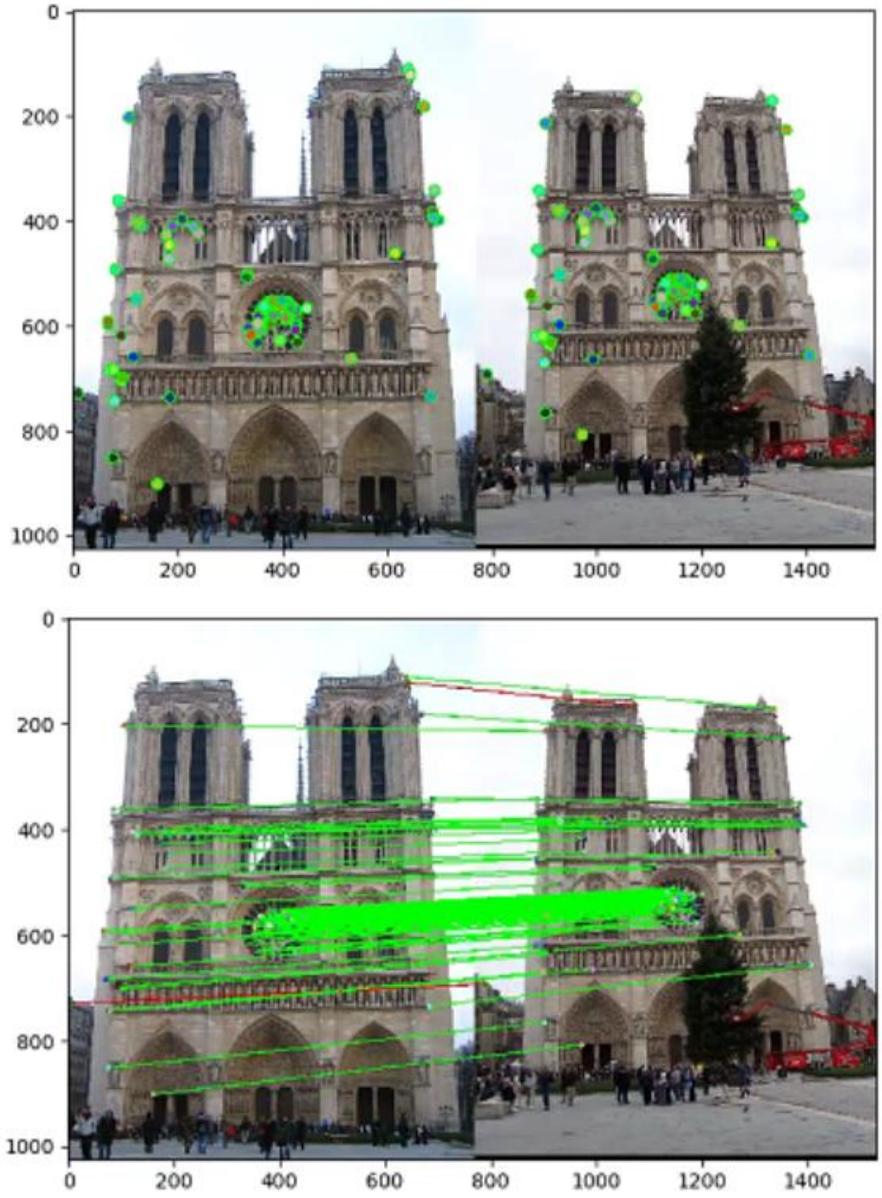

图4. 最终匹配效果

完整程序可以在测试环境中10min以内运行完，算法最终得到的性能指标如下，置信度最高的100个匹配结果为98.0%的准确率。

```
1500 corners in image 1, 1500 corners in image 2
1500 matches from 1500 corners
You found 100/100 required keypoints
one image accuracy = 0.690000
You found 100/100 required keypoints
one image accuracy = 0.630000
------------key point mean acc on  two images: 0.6599999999999999
You found 100/100 required matches
--------Match Accuracy = 0.980000------
```