计算机视觉应用与实践（一）

目录

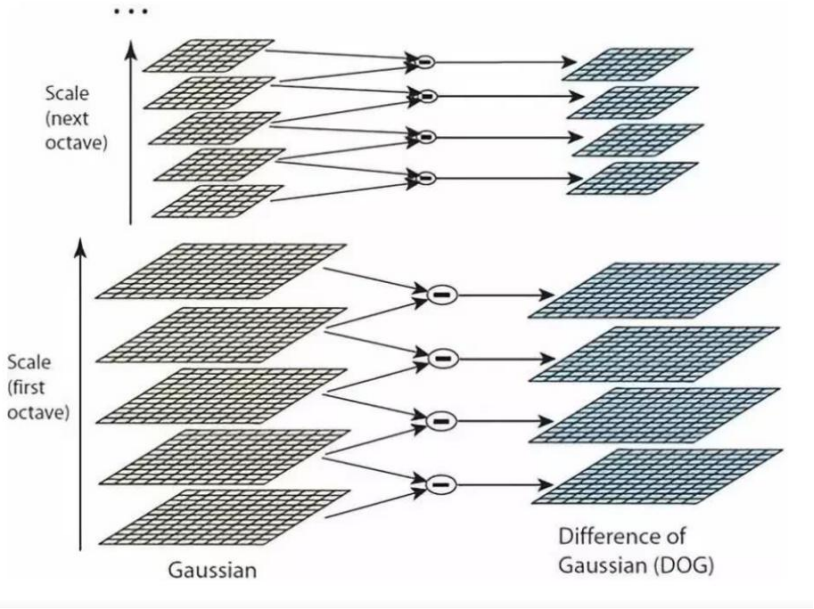
一、 实验目的

1. 理解关键点检测算法 DOG 原理。
2. 理解尺度变化不变特征 SIFT。
3. 采集一系列局部图像，自行设计拼接算法。
4. 使用 Python 实现图像拼接算法。

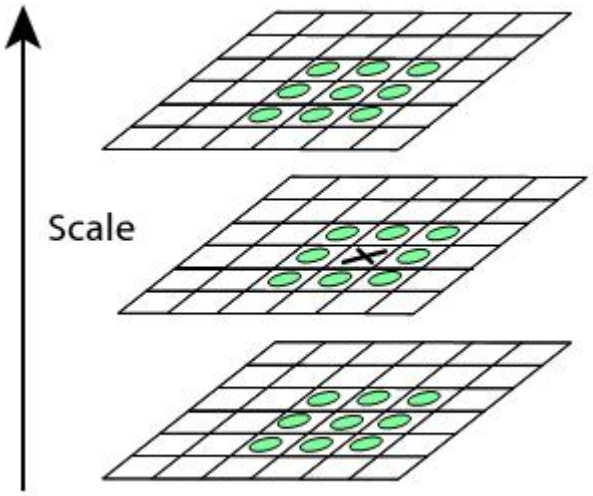
二、 实验原理

DOG 算法

DOG (Difference of Gaussians)算法是一种基于高斯滤波器的图像特征提取算法，其基本原理是对图像进行不同尺度的高斯模糊，并对每个尺度的高斯模糊图像进行相邻两个尺度之间的差分，具体如下图所示。



在得到一系列的差分图像（也就是 DOG）之后，根据图像中每个点与周围 26 个点的大小关系确定极值点，如下所示。

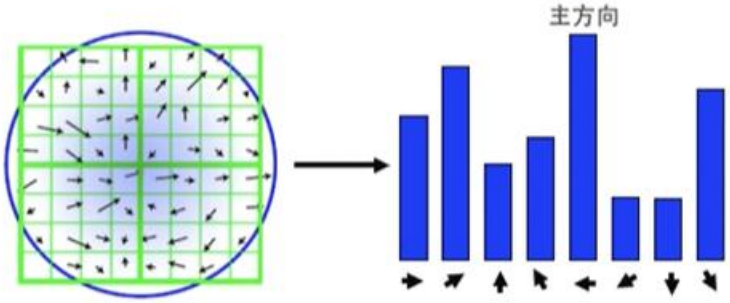


在得到极值点之后，然后在极值点附近通过泰勒展开调整极值的位置。最后再舍去对比度低和边缘上的点，最终得到我们需要的关键点。

SIFT 算法

SIFT (Scale-Invariant Feature Transform)算法是一种用于图像处理和计算机视觉中的特征提取和匹配算法。该算法可以检测和描述图像中的局部特征，并具有尺度不变性和旋转不变性等优点。

在通过 DOG 算法得到关键点之后。SIFT 算法会以每个关键点为中心，用其周围的图像区域内的梯度方向直方图描述关键点的特征向量，如下图所示。



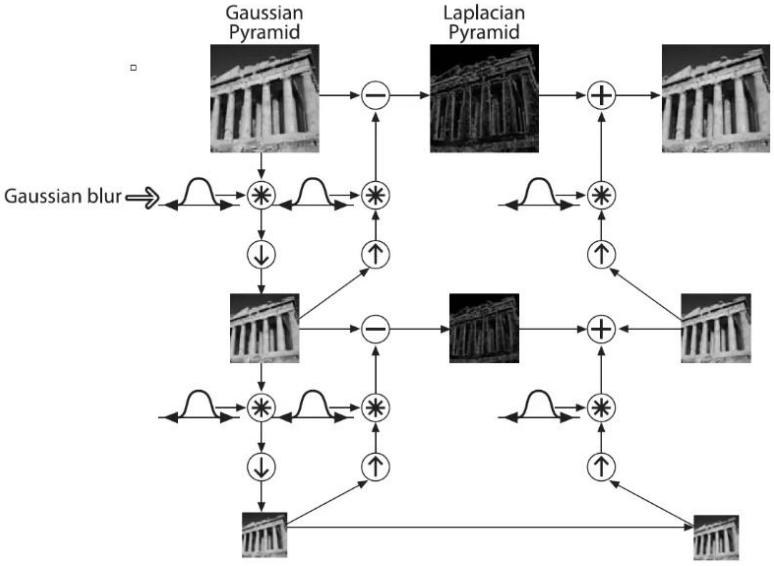
在提取了两张图片的关键点及其特征向量之后，使用 K 近邻算法去匹配，最后实现两个特征点之间的配对。

RANSAC 算法

RANSAC（Random Sample Consensus）是一种迭代方法，用于从一组带有噪声和异常值的数据中估计出最佳模型的参数。它的基本思想是从数据中随机选择一组子集，这些子集被认为是符合模型的数据，然后使用这些子集拟合模型并计算出模型的误差。然后，将所有其他数据与模型进行比较，将其分为内点（符合模型的数据）和外点（不符合模型的数据）。如果内点的数量达到了一定的阈值，则认为当前模型是可接受的，并且使用所有内点来重新拟合模型。这个过程会重复执行多次，每次得到一个新的模型，并将最好的模型作为最终结果返回。

拉普拉斯金字塔图像融合

拉普拉斯金字塔（(Laplacian pyramid)）是一种常用的图像融合方法，其基本原理是通过对原始图像和融合目标图像构建拉普拉斯金字塔，从而得到一系列的高频和低频分量，然后对这些分量进行权重叠加和重构，得到最终的融合图像。构建拉普拉斯金字塔的过程如下所示，每层的图像是由第 n 层高斯金字塔图像的上采样和第 n+1 层高斯金字塔图像做差得到的。在融合时，将高斯金字塔最顶层的图像按照 mask 融合，然后上采样并与对应的按照 mask 融合的拉普拉斯金字塔相加，不断重复直到恢复到图像原本大小。



三、 程序代码

首先是读入图像并调整图像大小。

```
# 读入图像并调整大小
img1 = cv2.imread('data/IMG6.jpg')
img2 = cv2.imread('data/IMG5.jpg')
img1 = cv2.resize(img1, dsize=(1024, 512))
img2 = cv2.resize(img2, dsize=(1024, 512))
```

使用 opencv 构建 SIFT 并完成特征点检测和匹配。

```
# 使用 opencv 创建 SIFT 对象
sift = cv2.xfeatures2d.SIFT_create()
# 检测关键点和计算描述符
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
# 匹配关键点
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)
good_matches = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        good_matches.append(m)
# 绘制匹配结果
img_matches = cv2.drawMatches(img1, kp1, img2, kp2, good_matches, None, flags=2)
```

计算变换矩阵并对 img1 做变换。

```
#    使用 RANSAC 算法筛选特征点并得到变换矩阵
src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
```

```
dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
#   做透视性变换
h, w = img1.shape[:2]
img1_warped = cv2.warpPerspective(img1, M, (int(max(corner[1][0], corner[3][0])), i
mg2.shape[0]))
```

像素加权融合。

```
#   使用加权像素方法处理融合区域，权重取决于像素点与区域左边界的距离
def OptimizeSeam(img2, img1warped, img, corner):
    left = int(min(corner[0][0], corner[2][0]))
    right = img2.shape[1]
    width = right - left
    height = img.shape[0]
    for i in range(height):
        for j in range(left, right):
            if sum(img1warped[i, j, :]) == 0:
                alpha = 1.0
            else:
                alpha = (width - (j - left)) / width
            img[i, j, :] = img2[i, j, :] * alpha + img1warped[i, j, :] * (1 - alpha)

    return img
```

拉普拉斯金字塔融合。

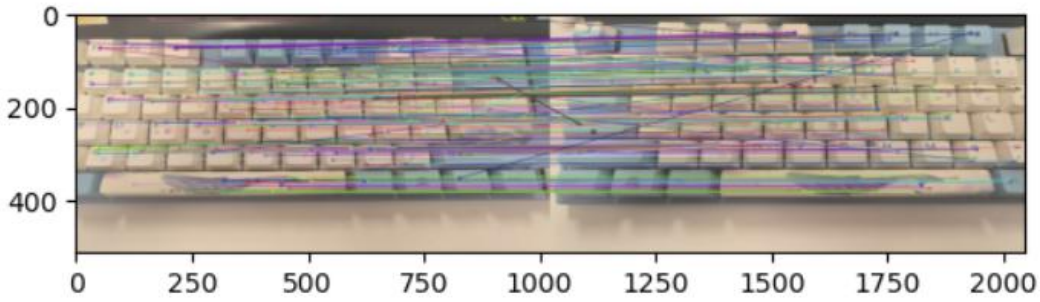
```
def LP(img_left, img_right, level=6, ratio=0.5):
    G_left = img_left.copy()
    G_right = img_right.copy()
    G_mask = np.zeros(img_left.shape)
    left = int(img_left.shape[1] * ratio)
    G_mask[:, :left] = 1.0
    gp_left = [G_left]
    gp_right = [G_right]
    gp_mask = [G_mask]
    # 生成高斯金字塔
    for i in range(level):
        G_left = cv2.pyrDown(G_left)
        gp_left.append(G_left)
        G_right = cv2.pyrDown(G_right)
        gp_right.append(G_right)
        G_mask = cv2.pyrDown(G_mask)
        gp_mask.append(G_mask)
    # 生成左右图像的拉普拉斯金字塔
    lp_left = [gp_left[level-1]]
    lp_right = [gp_right[level-1]]
```

```
    for i in range(level - 1, 0, -1):
        GE_left = cv2.pyrUp(gp_left[i])
        L_left = cv2.subtract(gp_left[i - 1], GE_left)
        lp_left.append(L_left)
        GE_right = cv2.pyrUp(gp_right[i])
        L_right = cv2.subtract(gp_right[i - 1], GE_right)
        lp_right.append(L_right)
    # 融合图像
    gp_mask.pop()
    gp_mask = gp_mask[::-1]
    LS = []
    for l_left, l_mask, l_right in zip(lp_left, gp_mask, lp_right):
        ls = l_mask * l_left + (1 - l_mask) * l_right
        LS.append(ls)
    # 重建图像
    ls_ = LS[0]
    for i in range(1, level):
        ls_ = cv2.pyrUp(ls_)
        ls_ = cv2.add(ls_, LS[i])
    return ls_
```

四、 实验结果

原始图像

特征点匹配结果



对 img1 做变换

直接拼接结果，可以在按键 P 处有明显的缝隙。

使用像素加权融合结果，能够有效的消除存在的裂缝。

使用拉普拉斯金字塔融合结果，有一些效果但是融合区域较为模糊。

五、 实验总结

通过这次实验，我了解了 DOG 和 SIFT 算法的原理，并使用了加权像素融合和拉普拉斯金字塔两种方法实现了两幅图像的融合，实验结果显示，两种方法相较于直接拼接的方法都能很好的去除融合结果中的缝隙，实现较好的融合效果。