计算机视觉实践实验报告

目录

一. 实验目的

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

二. 实验原理

主要包括关键点检测算法 DOG，尺度变化不变特征 SIFT 这 2 个部分。

2.1 关键点检测算法 DOG

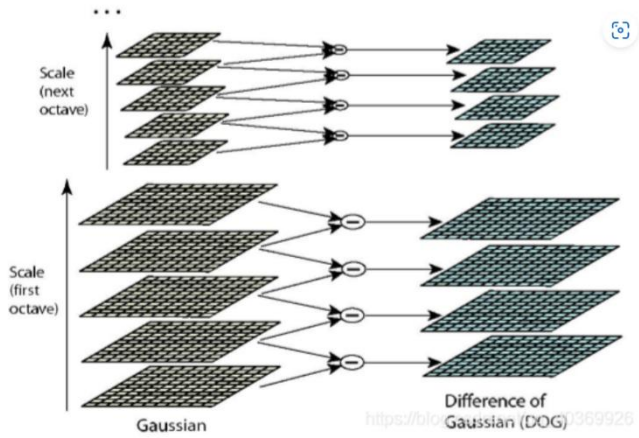
DOG（Difference of Gaussian）是一种关键点检测算法，其基本思想是通过不同尺度下的高斯滤波器之间的差异来检测图像中的关键点。算法首先对图像进行多尺度空间滤波，并计算相邻两个尺度的高斯差分图像，然后通过比较像素点周围邻域的灰度值，来确定局部最大值和最小值，并判断其是否为关键点。

算法的流程如下：

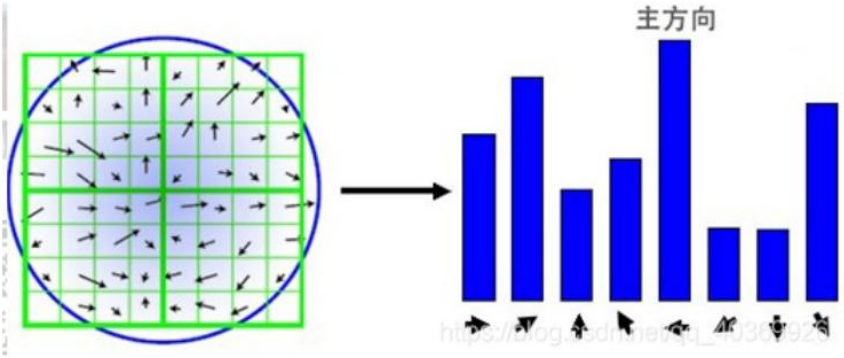
- 对原始图像进行高斯滤波，得到不同尺度下的高斯平滑图像。
- 计算相邻两个尺度的高斯差分图像，得到一组 DOG 图像。
- 对每个像素点的邻域进行比较，找到局部最大值和最小值，判断其是否为关键点。
- 根据关键点的稳定性和可重复性进行筛选，得到最终的关键点集合。

2.2 尺度变化不变特征 SIFT

SIFT（Scale-Invariant Feature Transform）是一种尺度不变的特征提取算法，它可以在不同尺度下提取图像中的关键点，并计算关键点的局部特征描述子。SIFT 算法包括四个主要步骤：尺度空间极值点检测、关键点定位、方向分配和局部特征描述子计算。

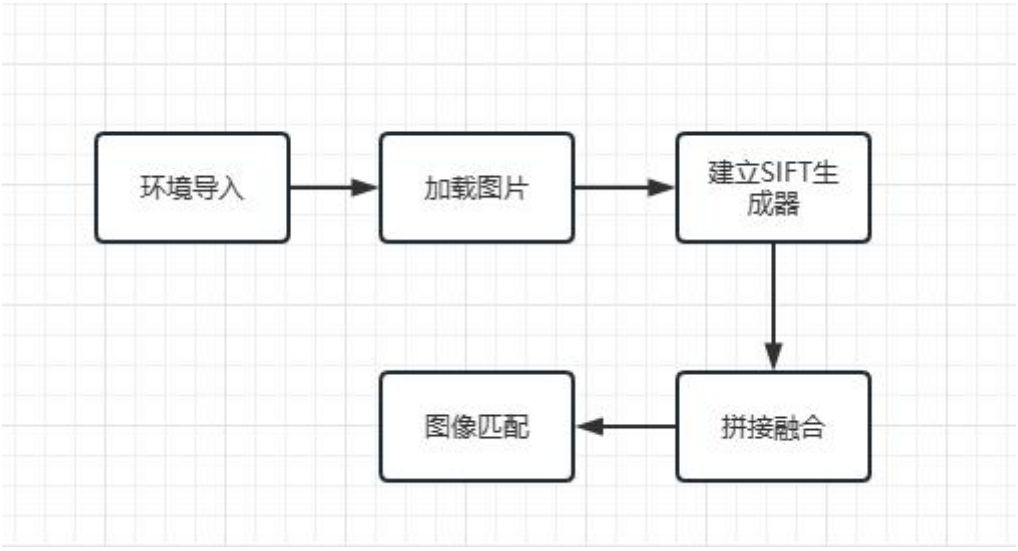


SIFT 算法首先通过构建高斯尺度空间，得到一组图像金字塔。然后在每个尺度上，通过高斯差分来检测尺度空间中的极值点，这些极值点可能代表着关键点。对于检测到的极值点，SIFT 算法通过拟合二维高斯函数来精确定位关键点的位置和尺度，并排除低对比度和边缘响应的点。为了使关键点对旋转不变性，SIFT 算法将关键点的方向分配为其梯度直方图的主方向，以此为基础构建特征描述子。SIFT 算法将关键点周围的像素划分为若干个小区域，并计算每个区域内像素的梯度幅值和方向直方图，将这些直方图组成一个局部特征向量。最终，将所有的局部特征向量组成特征向量集合。



SIFT 算法具有尺度不变性、旋转不变性和光照不变性，可以检测到不同尺度、旋转和光照下的关键点。它可以提取稳定的局部特征描述子，能够区分不同的图像区域。SIFT 算法具有较高的匹配精度和鲁棒性。

三． 实验步骤



四. 程序代码

- 加载图片。使用 opencv 读入图片，并转换为灰度图。

```
# 加载图片
img1 = cv2.imread('imageA.png')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.imread('imageB.png')
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
image1 = gray1.copy()
image2 = gray2.copy()
```

- 建立 SIFT 生成器。检测 SIFT 算子并计算描述子，将结果转换成 Numpy 数组。

```
# 建立SIFT生成器
descriptor = cv2.SIFT_create()
# 检测SIFT特征点，并计算描述子
kps1, features1 = descriptor.detectAndCompute(image1, None)
kps2, features2 = descriptor.detectAndCompute(image2, None)
# 将结果转换成NumPy数组
kps1 = np.float32([kp.pt for kp in kps1])
kps2 = np.float32([kp.pt for kp in kps2])
```

- 建立匹配器，使用 KNN 检测来自 A、B 图的 SIFT 特征匹配对，K=2，当最近距离跟次近距离的比值小于 ratio 值时，保留此匹配对。计算视角变换矩阵。

```
def matchKeypoints(kpsA, kpsB, featuresA, featuresB, ratio=0.75, reprojThresh=4.0):
    # 建立暴力匹配器
    matcher = cv2.BFMatcher()
    # 使用KNN检测来自A、B图的SIFT特征匹配对，K=2
    rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
    matches = []
    for m in rawMatches:
        # 当最近距离跟次近距离的比值小于ratio值时，保留此匹配对
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            # 存储两个点在featuresA, featuresB中的索引值
            matches.append((m[0].trainIdx, m[0].queryIdx))
        # 当筛选后的匹配对大于4时，计算视角变换矩阵
    if len(matches) > 4:
        # 获取匹配对的点坐标
        ptsA = np.float32([kpsA[i] for (_, i) in matches])
        ptsB = np.float32([kpsB[i] for (i, _) in matches])
        # 计算视角变换矩阵
        (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC, reprojThresh)
        # 返回结果
        return (matches, H, status)
    # 如果匹配对小于4时，返回None
    return None
```

- 进行图像拼接融合，将图片 A 进行视角变换，result 是变换后图片，图片 B 传入 result 图片最左端。

```
(matches, H, status) = matchKeypoints(kps1, kps2, features1, features2)
print(H)

if matches is not None:
    # 否则，提取匹配结果
    # 将图片A进行视角变换，result是变换后图片
    result = cv2.warpPerspective(img1, H, (img1.shape[1] + img2.shape[1], img1.shape[0]))
    cv2.imshow('result', result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    # 将图片B传入result图片最左端
    result[0:img2.shape[0], 0:img2.shape[1]] = img2
    cv2.imshow('result', result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

五．实验结果

原图：

变换矩阵：

```
[[ 7.51803604e-01  5.22681687e-03  7.22765904e+01]
 [-6.37029249e-02  9.06643613e-01  9.80276872e+00]
 [-7.34774119e-04  5.07968138e-05  1.00000000e+00]]
```

结果图：

六. 实验分析与总结

- SIFT 算法在图像拼接中有很好的应用效果。通过提取关键点和特征描述符，SIFT 算法可以在两幅图像中寻找匹配点，从而实现图像的配准和拼接。
- 在进行图像拼接时，还需要注意一些细节问题，比如选择适当的匹配算法、确定匹配点的阈值、进行图像融合等。在实验中，我们使用了 OpenCV 库中的函数来实现这些功能，但是需要根据具体的应用情况进行调参和优化，以获得更好的拼接效果。