计算机视觉与应用实践实验报告（一）

目录

一、实验目的

- 理解关键点检测算法 DOG 原理。
- 理解尺度不变特征变换 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

2.1 关键点检测算法 DOG

通过将目标图像与高斯函数进行卷积运算得到一幅目标图像的低通滤波结果，此过程称为去噪。（注：这里的 Gaussian 和高斯低通滤波器的高斯是一个含义，即：正态分布函数）。

$$G_{\sigma_1}(x,y)=\frac{1}{\sqrt{2\pi\sigma_1^2}}exp\left(-\frac{x^2+y^2}{2\sigma_1^2}\right)$$

在某一尺度上的特征检测可以通过两个相邻高斯尺度空间的图像相减，得到 DOG 的响应值图像。

详细过程如下：

首先 ：对一幅图像 f（x，y）进行不同参数的高斯滤波计算，表示如下

$$g_1(x,y)=G_{\sigma_1}(x,y)*f(x,y)$$

$$g_2(x,y)=G_{\sigma_2}(x,y)*f(x,y)$$

其次：将滤波得到的结果 g1（x,y）和 g2（x,y）相减得到：

$$g_1(x,y)-g_2(x,y)=G_{\sigma_1}*f(x,y)-G_{\sigma_2}*f(x,y)=(G_{\sigma_1}-G_{\sigma_2})*f(x,y)=DoG*f(x,y)$$

即：可以将 DOG 表示为：

$$DoG\triangleq G_{\sigma_1}-G_{\sigma_2}=\frac{1}{\sqrt{2\pi}}\left(\frac{1}{\sigma_1}e^{-(x^2+y^2)/2\sigma_1^2}-\frac{1}{\sigma_2}e^{-(x^2+y^2)/2\sigma_2^2}\right)$$

在具体的图像处理中，就是将两幅不同参数下的高斯滤波结果相减。得到 DOG 图。 最后，以此依据上述操作得到不同尺度下的 DOG 图，进而在三维空间中求角点。

2.1 尺度不变特征变化 SIFT

尺度不变特征转换 SIFT 算法的实质是在不同的尺度空间上查找关键点(特征点)，并计算出关键点的方向。SIFT 所查找到的关键点是一些十分突出，不会因光照，仿射变换和噪音等因素而变化的点，如角点、边缘点、暗区的亮点及亮区的暗点等。

SIFT 算法分解为如下四步：

- 尺度空间极值检测：搜索所有尺度上的图像位置。通过高斯差分函数来识别潜在的对于尺度和旋转不变的关键点。
- 关键点定位：在每个候选的位置上，通过一个拟合精细的模型来确定位置和尺度。关键点的选择依据于它们的稳定程度。
- 关键点方向确定：基于图像局部的梯度方向，分配给每个关键点位置一个或多个方向。所有后面的对图像数据的操作都相对于关键点的方向、尺度和位置进行变换，从而保证了对于这些变换的不变性。
- 关键点描述：在每个关键点周围的邻域内，在选定的尺度上测量图像局部的梯度。这些梯度作为关键点的描述符，它允许比较大的局部形状的变形或光照变化。

三、实验步骤

四、关键程序代码

1、获取关键特征点和 sift 的特征向量

```
def sift_keypoints_detect(image):
    # 处理图像一般很少用到彩色信息，通常将图像转换为灰度图
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # 获取图像特征 sift-SIFT 特征点,实例化对象 sift
    sift = cv2.SIFT_create()
    # keypoints:特征点向量,向量内的每一个元素是一个 KeyPoint 对象，包含了特征点的各种属性信息(角度、关键特征点坐标等)
    # features:表示输出的 sift 特征向量，通常是 128 维的
    keypoints, features = sift.detectAndCompute(image, None)
    """
    cv2.drawKeyPoints():在图像的关键特征点部位绘制一个小圆圈。
    如果传递标志 flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS,
    它将绘制一个大小为 keypoint 的圆圈并显示它的方向。
    这种方法同时显示图像的坐标，大小和方向，是最能显示特征的一种绘制方式。
    """
    keypoints_image = cv2.drawKeypoints(
        gray_image, keypoints, None,
     flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
    # 返回带关键特征点的图像、关键特征点和 sift 的特征向量
    return keypoints_image, keypoints, features
```

2、使用 KNN 检测来自左右图像的 SIFT 特征进行匹配

```
def get_feature_point_ensemble(features_right, features_left):

    # 创建 BFMatcher 对象解决匹配
    bf = cv2.BFMatcher()
    # knnMatch()函数：返回每个特征点的最佳匹配 k 个匹配点
    # features_right 为模板图，features_left 为匹配图
    matches = bf.knnMatch(features_right, features_left, k=2)
    """
    利用 sorted()函数对 matches 对象进行升序(默认)操作,x:x[]字母可以随意修改，
    排序方式按照中括号[]里面的维度进行排序，[0]按照第一维排序，[2]按照第三维排序
    """
    matches = sorted(matches, key=lambda x: x[0].distance / x[1].distance)
    # 建立列表 good 用于存储匹配的点集
    good = []
    for m, n in matches:
        # ratio 的值越大，匹配的线条越密集，但错误匹配点也会增多
        ratio = 0.6
        if m.distance < ratio * n.distance:
            good.append(m)
    # 返回匹配的关键特征点集
    return  good
```

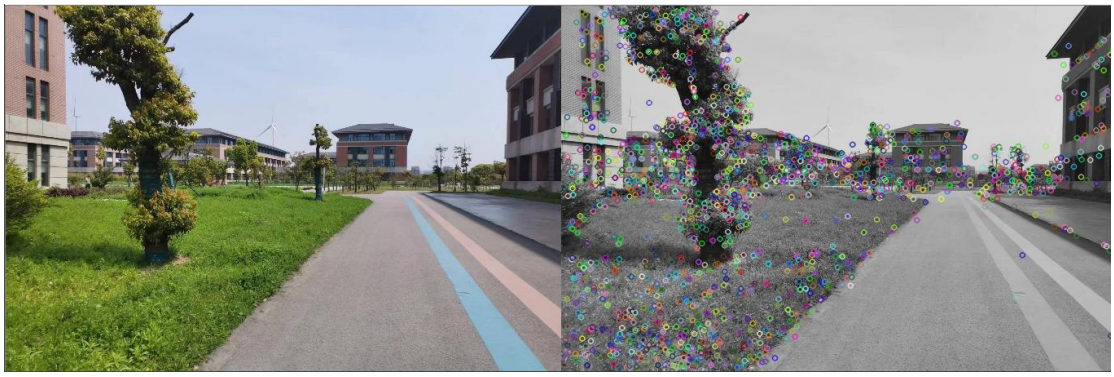
3、计算视角变换矩阵 H，用 H 对右图进行变换并返回全景拼接图像

```
def Panorama_stitching(image_right, image_left):
    _, keypoints_right, features_right = sift_keypoints_detect(image_right)
    _, keypoints_left, features_left = sift_keypoints_detect(image_left)
    goodMatch = get_feature_point_ensemble(features_right, features_left)
    # 当筛选项的匹配对大于 4 对(因为 homography 单应性矩阵的计算需要至少四个点)时,计算视角变
换矩阵
    if len(goodMatch) > 4:
        # 获取匹配对的点坐标
        Point_coordinates_right = np.float32(
            [keypoints_right[m.queryIdx].pt for m in goodMatch]).reshape(-1, 1, 2)
        Point_coordinates_left = np.float32(
            [keypoints_left[m.trainIdx].pt for m in goodMatch]).reshape(-1, 1, 2)
        # ransacReprojThreshold:将点对视为内点的最大允许重投影错误阈值(仅用于 RANSAC 和 RHO
方法时)
        # 若 srcPoints 和 dstPoints 是以像素为单位的，该参数通常设置在 1 到 10 的范围内
        ransacReprojThreshold = 4
        # cv2.findHomography():计算多个二维点对之间的最优单映射变换矩阵 H(3 行 x3 列),使用
最小均方误差或者 RANSAC 方法
        # 作用:利用基于 RANSAC 的鲁棒算法选择最优的四组配对点，再计算转换矩阵 H(3*3)并返回,
以便于反向投影错误率达到最小
```

```
        Homography, status = cv2.findHomography(
            Point_coordinates_right, Point_coordinates_left, cv2.RANSAC,
ransacReprojThreshold)
        # cv2.warpPerspective()：透视变换函数，用于解决 cv22.warpAffine()不能处理视场和图像不平行的问题
        # 作用：就是对图像进行透视变换，可保持直线不变形，但是平行线可能不再平行
        Panorama = cv2.warpPerspective(
            image_right, Homography, (image_right.shape[1] + image_left.shape[1],
image_right.shape[0]))
        # cv2.imshow("扭曲变换后的右图", Panorama)
        plt.rcParams["figure.figsize"] = (20, 10)
        plt.title('transformed right image')
        plt.imshow(Panorama)
        plt.show()
        # 将左图加入到变换后的右图像的左端即获得最终图像
        Panorama[0:image_left.shape[0], 0:image_left.shape[1]] = image_left
        # 返回全景拼接的图像
        return Panorama
```

五、实验结果

（1）左图中的关键点与原图



（2）有图中的关键点与原图

（3）所有匹配的 SIFT 关键特征点连线

（4）全景图结果

六、实验分析与总结

全景拼接的整体思路过程还是比较清晰的，但是从最后生成的全景图结果可以看出拼接后的全景图存在图像边缘过渡不自然的情况，可以看到明显的分割线，可能由于原始图像存在亮度差异，导致拼接缝明显。因为两张原图存在光线差异，后续还需要进一步做平滑处理和优化，理论上可使用 Graph cuts 方法解决。