《多媒体技术》第 2 次作业

姓名：张半佛　　完成日期：2019 年 10 月

题目要求：

将两幅图片拼接到一起。（提交完整的实验报告，具体要求有：核心算法原理、处理步骤、每一步骤的结果分析，还包括使用不同参数的性能分析。不能直接调用拼图的库，必须一步一步走，核心算法可以调库。）
解：

目录

一：核心算法原理

SIFT 算法：

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 特征检测主要包括以下 4 个基本步骤：

1. 尺度空间极值检测：搜索所有尺度上的图像位置。通过高斯微分函数来识别潜在的对于尺度和旋转不变的兴趣点。

2. 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。

3. 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，从而提供对于这些变换的不变性。

4. 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。这些梯度被变换成一种表示，这种表示允许比较大的局部形状的变形和

光照变化。

SIFT 特征匹配主要包括 2 个阶段：

第一阶段：SIFT 特征的生成，即从多幅图像中提取对尺度缩放、旋转、亮度变化无关的特征向量。

第二阶段：SIFT 特征向量的匹配。

SIFT 特征的生成一般包括以下几个步骤：

1. 构建尺度空间，检测极值点，获得尺度不变性。
2. 特征点过滤并进行精确定位。
3. 为特征点分配方向值。
4. 生成特征描述子。

RANSAC 算法：

RANSAC 是“RANdom SAmple Consensus（随机抽样一致）”的缩写。它可以从一组包含“局外点”的观测数据集中，通过迭代方式估计数学模型的参数。它是一种不确定的算法——它有一定的概率得出一个合理的结果；为了提高概率必须提高迭代次数。

RANSAC 的基本假设是：

（1）数据由“局内点”组成，例如：数据的分布可以用一些模型参数来解释；

（2）“局外点”是不能适应该模型的数据；

（3）除此之外的数据属于噪声。

局外点产生的原因有：噪声的极值；错误的测量方法；对数据的错误假设。

RANSAC 也做了以下假设：给定一组（通常很小的）局内点，存在一个可以估计模型参数的过程；而该模型能够解释或者适用于局内点。

RANSAC 算法的输入是一组观测数据，一个可以解释或者适应于观测数据的参数化模型，一些可信的参数。

RANSAC 通过反复选择数据中的一组随机子集来达成目标。被选取的子集被假设为局内点，并用下述方法进行验证：

1.首先我们先随机假设一小组局内点为初始值。然后用此局内点拟合一个模型，此模型适应于假设的局内点，所有的未知参数都能从假设的局内点计算得出。

2.用 1 中得到的模型去测试所有的其它数据，如果某个点适用于估计的模型，认为它也是局内点，将局内点扩充。

3.如果有足够多的点被归类为假设的局内点，那么估计的模型就足够合理。

4.然后，用所有假设的局内点去重新估计模型，因为此模型仅仅是在初始的假设的局内点估计的，后续有扩充后，需要更新。

5.最后，通过估计局内点与模型的错误率来评估模型。

整个这个过程为迭代一次，此过程被重复执行固定的次数，每次产生的模型有两个结局：

1、要么因为局内点太少，还不如上一次的模型，而被舍弃，

2、要么因为比现有的模型更好而被选用。

二：详细处理步骤解析

1. 首先定义了 Stitcher 类，可以检测我们是否使用了 OpenCV3。由于在 opencv 2.4 和 OpenCV 3 处理关键点检测和局部不变特征的有明显的差异，OpenCV 的版本对我们的使用是很重要的。

```
class Stitcher:
    def __init__(self):
        # determine if we are using OpenCV v3.X
        self.isv3 = imutils.is_cv3()
```

2. 接下来是定义方法 stitch，stitch 方法只需要一个单一的参数：images。这是传入图片的列表，后面是要缝合在一起形成全景图。还可以提供 ratio ，用于特征匹配时 David Lowe 比率测试，reprojthresh 是 RANSAC 算法中最大像素“回旋的余地”，最后的 showMatches，是一个布尔类型的值，用于表明是否应可以可视化关键点匹配。
   我们拆包图片列表后，调用 detectAndDescribe 方法这个方法可以检测到两张图片里关键点、提取局部不变特征。
   有了关键点和特征，我们可以用 matchKeypoints 方法来匹配两张图片里的特征。
   如果返回匹配的 M 为 None，就是因为现有的关键点不足以匹配生成全景图。
   接下来就是准备应用透视变换：
   假设 M 不返回 None，我们在拆包这个元组，是一个包含关键点匹配、从 RANSAC 算法中得到的单应矩阵 H 以及最后的 status，用来表明那些已经成功匹配的关键点。
   有了单应矩阵 H 后，就可将两张图片“缝合起来”。首选调用 cv2.warpPerspective，需要三个参数：想要“缝合”上来的照片（本程序里的右边的图片）；还有 3*3 的转换矩阵 H；最后就是塑造出要输出的照片。我们得到输出图像的宽是两图片之和，高即为第二张图像的高度。
   之后检查看是否应该将关键点匹配，如果是的话就调用 drawMatches 函数，然后返回一个包含全图和可视化的图的元组。
   这样，就简单的返回一个拼接的图片。

```
def stitch(self, images, ratio=0.75, reprojThresh=4.0,
    showMatches=False):
    (imageB, imageA) = images
    (kpsA, featuresA) = self.detectAndDescribe(imageA)
    (kpsB, featuresB) = self.detectAndDescribe(imageB)
    M = self.matchKeypoints(kpsA, kpsB,
        featuresA, featuresB, ratio, reprojThresh)
    if M is None:
        return None
    (matches, H, status) = M
    result = cv2.warpPerspective(imageA, H,
        (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))
    result[0:imageB.shape[0], 0:imageB.shape[1]] = imageB
    if showMatches:
        vis = self.drawMatches(imageA, imageB, kpsA, kpsB, matches,
            status)
        return (result, vis)
    return result
```

3. detectAndDescribe 方法用来接收照片，检测关键点和提取局部不变特征。在实现中用到了高斯差分（Difference of Gaussian (DoG)）关键点检测，和 SIFT 特征提取。之后检测是否用了 OpenCV 3.X，如果是，就用 cv2.xfeatures2d.SIFT_create 方法来实现 DoG 关键点检测和 SIFT 特征提取。
detectAndCompute 方法用来处理提取关键点和特征。
cv2.FeatureDetector_create 方法来实现关键点的检测（DoG）。detect 方法返回一系列的关键点。
之后需要用 SIFT 关键字来初始化 cv2.DescriptorExtractor_create，设置 SIFT 特征提取。调用 extractor 的 compute 方法返回一组关键点周围量化检测的特征向量。
最后，关键点从 KeyPoint 对象转换为 NumPy 数列后返回给调用函数。

```
def detectAndDescribe(self, image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    if self.isv3:
        descriptor = cv2.xfeatures2d.SURF_create()
        (kps, features) = descriptor.detectAndCompute(image, None)
    else:
        detector = cv2.FeatureDetector_create("SURF")
        kps = detector.detect(gray)
        extractor = cv2.DescriptorExtractor_create("SURF")
        (kps, features) = extractor.compute(gray, kps)
    kps = np.float32([kp.pt for kp in kps])
    return (kps, features)
```

4. matchKeypoints 方法需要四个参数，第一张图片的关键点和特征向量，第二张图片的关键点特征向量。David Lowe's ratio 测试变量和 RANSAC 重投影门限也应该被提供。
   匹配的特征实际上是一个相当简单的过程。循环每张图片的描述子，计算距离，最后找到每对描述子的最小距离。因为这是计算机视觉中的一个非常普遍的做法，OpenCV 已经内置了一个 cv2.DescriptorMatcher_create 方法，用来匹配特征。BruteForce 的值表示能够更详尽计算两张图片直接的欧式距离，以此来寻找每对描述子的最短距离。knnMatch 方法是在 K=2 的两个特征向量的 k-NN 匹配（k-nearest neighbors algorithm，K 近邻算法），表明每个匹配的前两名作为特征向量返回。
   之所以要的是匹配的前两个而不是只有第一个，是因为我们需要用 David Lowe's ratio 来测试假匹配然后做修剪。
   之后， rawMatches 来计算每对描述子，但是这些描述子可能是错误的，也就是这是图片不是真正的匹配。去修剪这些有误的匹配，我们可以运用 Lowe's ratio 测试特别的来循环 rawMatches，这是用来确定高质量的特征匹配。正常的 Lowe's ratio 值在[0.7,0.8].
   用 Lowe's ratio 测试得到 matche 的值后，就可以计算这两串关键点之间的单应性。
   计算两串关键点的单应性需要至少四个匹配。为了获得更为可信的单应性，我们至少需要超过四个匹配点。

```
def matchKeypoints(self, kpsA, kpsB, featuresA, featuresB,
    ratio, reprojThresh):
    matcher = cv2.DescriptorMatcher_create("BruteForce")
    rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
    matches = []
    for m in rawMatches:
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            matches.append((m[0].trainIdx, m[0].queryIdx))
    if len(matches) > 4:
        ptsA = np.float32([kpsA[i] for (_, i) in matches])
        ptsB = np.float32([kpsB[i] for (i, _) in matches])
        (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,
            reprojThresh)
        return (matches, H, status)
    return None
```

5. 最后，Stitcher 里的最后一个方法 drawMatches–用来将两张图片关键点的联系可视化。
   这种方法需要通过两张原始图像来对每个图像的关键点进行设置，应用 Lowe's ratio 试验后的初始匹配，和最后由单应计算提供的状态列表。
   运用这些变量，可以通过将两张图片“里面”的关键点 N 和关键点 M 画直线来可视化。

```
def drawMatches(self, imageA, imageB, kpsA, kpsB, matches, status):
    (hA, wA) = imageA.shape[:2]
    (hB, wB) = imageB.shape[:2]
    vis = np.zeros((max(hA, hB), wA + wB, 3), dtype="uint8")
    vis[0:hA, 0:wA] = imageA
    vis[0:hB, wA:] = imageB
    for ((trainIdx, queryIdx), s) in zip(matches, status):
        if s == 1:
            ptA = (int(kpsA[queryIdx][0]), int(kpsA[queryIdx][1]))
            ptB = (int(kpsB[trainIdx][0]) + wA, int(kpsB[trainIdx][1]))
            cv2.line(vis, ptA, ptB, (0, 255, 0), 1)
    return vis
```

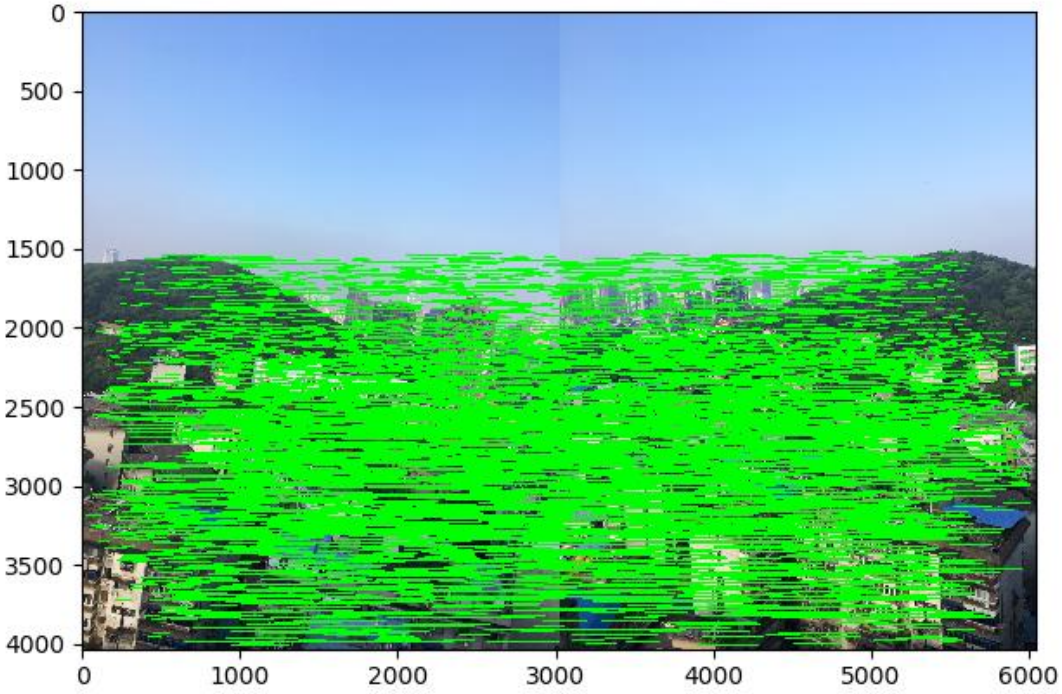
6. 最后主函数加载图片，由于 opencv 是 bgr，matplotlib 是 rgb 形式，所以需要转换一下，不然会出现颜色和原图有差别。

```
if __name__ == '__main__':
    imageA = cv2.imread("ori_a.jpg")
    imageB = cv2.imread("ori_b.jpg")

    stitcher = Stitcher()
    (result, vis) = stitcher.stitch([imageA, imageB], showMatches=True)
    # opencv is bgr, matplotlib is rgb
    result = cv2.cvtColor(result, cv2.COLOR_BGR2RGB)
    vis = cv2.cvtColor(vis, cv2.COLOR_BGR2RGB)
    plt.figure()
    plt.imshow(vis)
    plt.show()
    plt.figure()
    plt.imshow(result)
    plt.show()
```

三：实验结果分析

经过加载运行，可得出可视化的关键点:


0
500
1000
1500
2000
2500
3000
3500
4000
0
1000
2000
3000
4000
5000
6000

以及经过合成后的图片：


0
500
1000
1500
2000
2500
3000
3500
4000
0
1000
2000
3000
4000
5000
6000

需要注意的是，由于 SIFT 算法存在专利限制，所以在安装 opencv 库的时候要选择 3.4.3.X 之前的版本才能使用 SIFT 算法。
SURF 是 SIFT 算法的改进版本，计算量更小，更有优势。

四：附录（python 完整代码）

```
import numpy as np
import imutils
import cv2
from matplotlib import pyplot as plt
class Stitcher:
    def __init__(self):
        # determine if we are using OpenCV v3.X
        self.isv3 = imutils.is_cv3()
    def stitch(self, images, ratio=0.75, reprojThresh=4.0,
        showMatches=False):
        (imageB, imageA) = images
        (kpsA, featuresA) = self.detectAndDescribe(imageA)
        (kpsB, featuresB) = self.detectAndDescribe(imageB)
        M = self.matchKeypoints(kpsA, kpsB,
            featuresA, featuresB, ratio, reprojThresh)
        if M is None:
            return None
        (matches, H, status) = M
        result = cv2.warpPerspective(imageA, H,
            (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))
        result[0:imageB.shape[0], 0:imageB.shape[1]] = imageB
        if showMatches:
            vis = self.drawMatches(imageA, imageB, kpsA, kpsB, matches,
                status)
            return (result, vis)
        return result


    def detectAndDescribe(self, image):
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        if self.isv3:
            descriptor = cv2.xfeatures2d.SURF_create()
            (kps, features) = descriptor.detectAndCompute(image, None)
        else:
            detector = cv2.FeatureDetector_create("SURF")
            kps = detector.detect(gray)
            extractor = cv2.DescriptorExtractor_create("SURF")
```

```
            (kps, features) = extractor.compute(gray, kps)
        kps = np.float32([kp.pt for kp in kps])
        return (kps, features)


    def matchKeypoints(self, kpsA, kpsB, featuresA, featuresB,
        ratio, reprojThresh):
        matcher = cv2.DescriptorMatcher_create("BruteForce")
        rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
        matches = []
        for m in rawMatches:
            if len(m) == 2 and m[0].distance < m[1].distance * ratio:
                matches.append((m[0].trainIdx, m[0].queryIdx))
        if len(matches) > 4:
            ptsA = np.float32([kpsA[i] for (_, i) in matches])
            ptsB = np.float32([kpsB[i] for (i, _) in matches])
            (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,
                reprojThresh)
            return (matches, H, status)
        return None

    def drawMatches(self, imageA, imageB, kpsA, kpsB, matches, status):
        (hA, wA) = imageA.shape[:2]
        (hB, wB) = imageB.shape[:2]
        vis = np.zeros((max(hA, hB), wA + wB, 3), dtype="uint8")
        vis[0:hA, 0:wA] = imageA
        vis[0:hB, wA:] = imageB
        for ((trainIdx, queryIdx), s) in zip(matches, status):
            if s == 1:
                ptA = (int(kpsA[queryIdx][0]), int(kpsA[queryIdx][1]))
                ptB = (int(kpsB[trainIdx][0]) + wA,
int(kpsB[trainIdx][1]))
                cv2.line(vis, ptA, ptB, (0, 255, 0), 1)
        return vis


if __name__ == '__main__':
    imageA = cv2.imread("ori_a.jpg")
    imageB = cv2.imread("ori_b.jpg")

    stitcher = Stitcher()
    (result, vis) = stitcher.stitch([imageA, imageB], showMatches=True)
    # opencv is bgr, matplotlib is rgb
    result = cv2.cvtColor(result, cv2.COLOR_BGR2RGB)
```

```
    vis = cv2.cvtColor(vis, cv2.COLOR_BGR2RGB)
    plt.figure()
    plt.imshow(vis)
    plt.show()
    plt.figure()
    plt.imshow(result)
    plt.show()
```