基于 SURF 算法的图像拼接

2019.10.14

王泽鹏 2017301510036 网安二班

概要

图像拼接是图像处理中的一个重要内容，利用关键点特征值提取的方法进行的图像特征提取是其中一项关键方法，该项目便利用这一方法实现了对同一场景下具有重叠位置的且具有相似特征的两幅图片的拼接。

目录

1.基本思路

实现图像的拼接，有以下思路：

①利用 SIFT 或 SURF 等算法提取要拼接的两张图片的特征点及特征描述符；

②利用蛮力（Brute-Force）匹配或 FLANN 匹配对关键点的特征进行匹配；

③找到足够多的的匹配点后，利用特征匹配和单应性准确查找到查询对象；

④将两张图片进行拼接，可采用直接拼接或是基于平滑度的拼接方式，最终实现图像拼接。

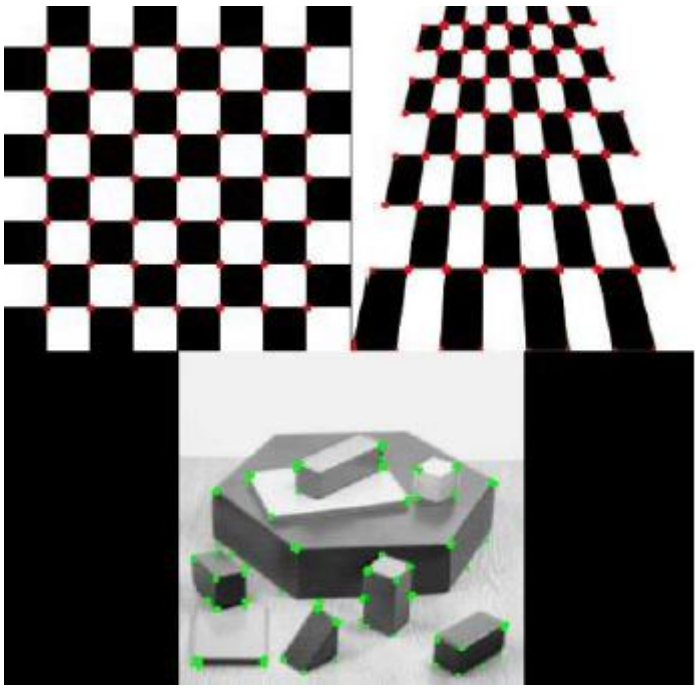
2.关键步骤

（1）关键点特征值提取

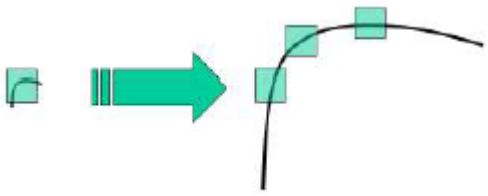
关键点及特征值的提取有很多方法。

①Harris 角点检测

一种方法是 Harris 角点检测。角点有一个特性：向任何方向移动变化都很大。如下图是一个利用 Harris 角点检测提取得到的关键点。



我们可以发现，角点检测技术具有旋转不变特性，即使图片发生了旋转，我们也可以得到相同的角点，因为图像旋转后角点依旧是角点。但这种方法依然有缺陷，以下图为例，如果在一个小图中使用一个小窗口可以检测到一个角点，但如果图像放大后，使用同样的窗口就无法检测到角点了。



为解决图像缩放引起的关键点检测问题，产生了另一种关键点检测算法：SIFT（尺度不变特征变换）算法。

②SIFT 算法

David G.Lowe 在 2004 年正式提出了一种基于尺度空间的、对图像缩放、旋转甚至仿射变换保持不变性的图像局部特征描述符——SIFT 描述符。该算子主要利用关键点邻域的主方向作为该店的方向特征，从而实现描述符对尺度和方向的无关性。

SIFT 特征向量具有如下特性：SIFT 特征是图像的局部特征，其对旋转、尺度缩放、亮度变化保持不变性，对视觉变化、放射变换、噪声也保持一定程度的稳定性；②独特性好，信息量丰富，适用于在海量特征数据库中进行快速、准确的匹配；③多量性，即使少数的几个物体也可以产生大量 SIFT 特征向量；④高速性，经优化的 SIFT 匹配算法甚至可以达到实时的要求；⑤可扩展性，可以很方便地与其他形式的特征向量进行联合。

由于在提取 SIFT 描述符时先对关键点进行了方向统一计算，所以 SIFT 描述符更具备优越性。目前，SIFT 主要用于匹配领域，其匹配能力较强，可以处理两幅图像之间发生平移、旋转、仿射变换情况下的匹配问题，甚至在某种程度上，对任意角度拍摄的图像也具备较为稳定的特征匹配能力。

下图为一幅关键点提取后的图像：

同样，SIFT 算法也有缺点，该算法的执行速度较慢，为满足人们的需求，速度更快的 SURF(加速稳健特征)算法也随之产生,该算法可以说是加速版的 SIFT。

③SURF 算法

与 SIFT 算法类似，SURF 算法的基本流程也可以分为三个主要部分：局部特征点的提取、特征点的描述、特征点的匹配。再具体细分可有以下几个步骤：构建 Hessian 矩阵，生成所有的兴趣点，用于特征的提取；构建尺度空间；特征点定位；特征点主方向分配；生成特征点描述子；特征点匹配。

（2）特征匹配

①Brute-Force 匹配

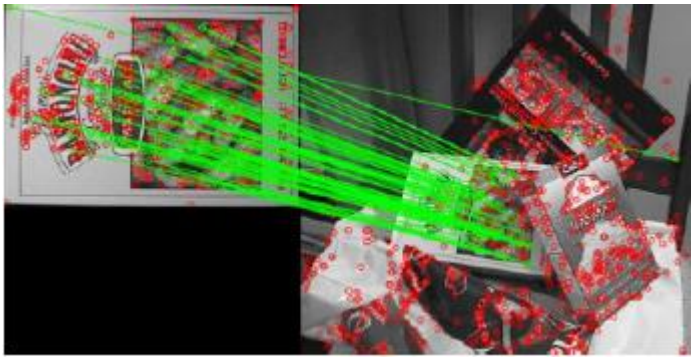
蛮力匹配方法相对较简单。先在第一幅图像中选取一个关键点，再依次与第二幅图像的每个关键点进行距离测试，最后返回距离最近的关键点。

②FLANN 匹配

FLANN 是快速最近邻搜索包的简称,它是一个对大数据集和高维特征进行最近邻搜索的算法的集合，在面对大数据集时它的效果比蛮力匹配方法要好。

它需要传入两个字典作为参数，第一个用于确定要使用的算法和其他相关参数,第二个用于指定递归遍历的次数,值越高结果越准确,但消耗的时间也越长。

下图为使用了 FLANN 匹配后特征匹配的图像:



（3）使用特征匹配和单应性查找对象

上述步骤已经让我们在一张杂乱的图像中找到了所需对象的部分位置信息，这些信息可以让我们在目标图像中准确地找到查询对象。

我们使用 cv2.findHomography()函数，将两个图像中的特征点集传给这个函数，便可得到该对象的透视图变换。进而我们可以使用 cv2.perspectiveTransform()找到这个对象。

（4）图片拼接

透视变换完的图片，大小就是全景图的大小，右侧为透视变换后的图像，左侧没有信息，我们拼接时可以直接将第一张图加到左侧，并覆盖掉重叠部分得到拼接图像；也可以把第一张图片直接放在左边，对重叠部分进行处理，离左图近的部分左图的权重高一些，离右图近的部分右图的权重高一些，这样使得图片重叠部分的过渡是平滑的，但相应所花时间也会更长。

3.核心代码

```
    surf = cv2.xfeatures2d.SURF_create(10000, nOctaves=4, extended=False
, upright=True)
    kp1, descrip1 = surf.detectAndCompute(img1, None)
    kp2, descrip2 = surf.detectAndCompute(img2, None)
```

cv2.xfeatures2d.SURF_create () 函数用于生成一个 SURF 对象，

extended=False，只生成 64 维的描述符而不是 128 维，令 upright=True，不检测关键点的方向。

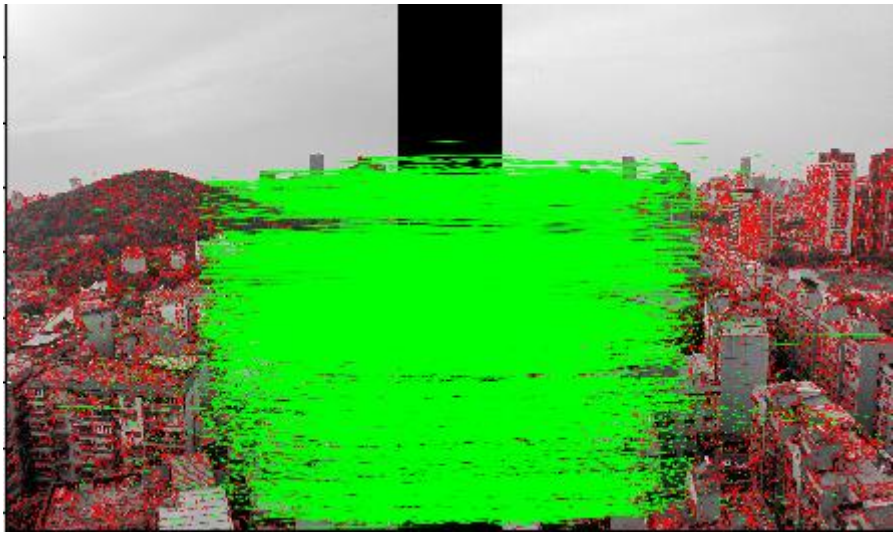
surf.detectAndCompute()函数用于计算图片的关键点和描述符。

```
FLANN_INDEX_KDTREE = 0
indexParams = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
searchParams = dict(checks=50)

flann = cv2.FlannBasedMatcher(indexParams, searchParams)
match = flann.knnMatch(descrip1, descrip2, k=2)
```

前面已介绍，使用 FLANN 匹配需要传入两个字典作为参数，我们使用文献中推荐的参数来设置。创建了匹配器得到匹配数组 match，包括两张图的描述符距离、索引。

我们可以得到初步匹配后的图像，如下图：



```
good = []
for i, (m, n) in enumerate(match):
    if (m.distance < 0.75 * n.distance):
        good.append(m)
```

进行比值测试，对匹配进行过滤。首先获取与 A 距离最近的点 B（最近）和 C（次近）。只有当 B/C 小于阈值时（0.75）才被认为是好的匹配，因为假设匹配是一一对应的，真正的匹配的理想距离为 0。

```
if len(good) > MIN:
```

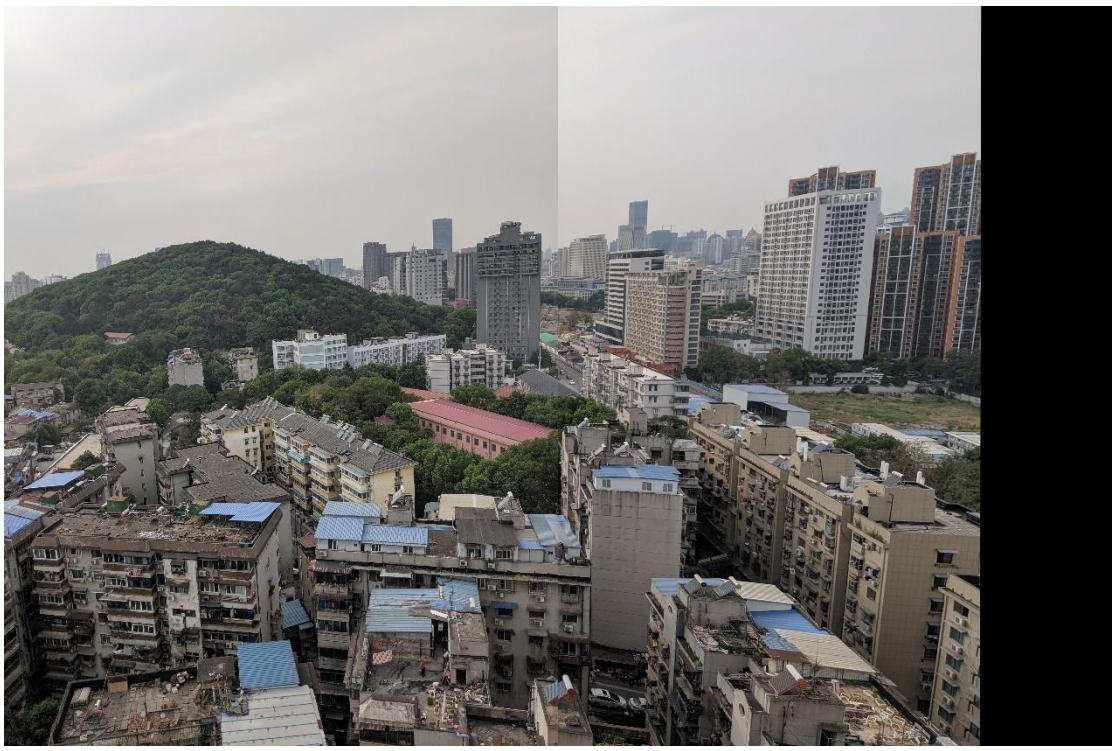
```
    src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-
1, 1, 2)
    ano_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-
1, 1, 2)
    M, mask = cv2.findHomography(src_pts, ano_pts, cv2.RANSAC, 5.0)
    warpImg = cv2.warpPerspective(img2, np.linalg.inv(M), (img1.shape[1
] + img2.shape[1], img2.shape[0]))
```

当找到了足够多的匹配后，我们需要提取两幅图像中匹配点的坐标，将他们传到 cv2.findHomography()函数中计算透视变换，得到变换矩阵，进而可使用cv2.warpPerspective()函数进行透视变换，将其变换视角变换到目标图像中。

进行了透视变换的图像如下图：

```
    direct = warpImg.copy()
    direct[0:img1.shape[0], 0:img1.shape[1]] = img1
```

直接将第一张图拼接到第二张图透视变换后的左边，便可得到直接拼接的结果，如下图：



可以看到，直接拼接后的图片中间有很明显的分割线，因此我们需要对图片进行光滑度处理。

```
    for col in range(0, cols):
        if img1[:, col].any() and warpImg[:, col].any():  # 开始重叠的最左端
            left = col
            break
    for col in range(cols - 1, 0, -1):
        if img1[:, col].any() and warpImg[:, col].any():  # 重叠的最右一列
            right = col
            break

    res = np.zeros([rows, cols, 3], np.uint8)
    for row in range(0, rows):
        for col in range(0, cols):
            if not img1[row, col].any():  # 如果没有原图，用旋转的填充
                res[row, col] = warpImg[row, col]
            elif not warpImg[row, col].any():
                res[row, col] = img1[row, col]
            else:
                srcImgLen = float(abs(col - left))
                testImgLen = float(abs(col - right))
                alpha = srcImgLen / (srcImgLen + testImgLen)
```

```
                res[row, col] = np.clip(img1[row, col] * (1 - alpha) +
warpImg[row, col] * alpha, 0, 255)

    warpImg[0:img1.shape[0], 0:img1.shape[1]] = res
```

这段代码关键就是重新计算重叠区域新的像素值，基本想法是，对于重叠的部分，靠近左边的部分则左边图像内容显示的多一些，靠近右边的部分则右边图像内容显示的多一些。用公式表示，设 alpha 为像素点横坐标到左右重叠边界横坐标距离所占比值，则新的像素值就可以表示为 newpixel=左图像素值×(1-alpha)+右图像素值×alpha，这样就较好的实现了重叠区域与周围区域的融合。这样我们得到了拼接效果更好的图片，如下图：

同时，对图片进行光滑度处理后相比简单拼接处理所花时间要长的多，如下图计算得简单拼接时间及程序总时间可得这一结论：

```
simple stich cost 8.573179

total cost 285.077147
```

至此，图片拼接成功实现。

4.完整代码

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import time

MIN = 10
starttime = time.time()
img1 = cv2.imread('IMG_201901.jpg')  # query
img2 = cv2.imread('IMG_201902.jpg')  # train

surf = cv2.xfeatures2d.SURF_create(10000, nOctaves=4, extended=False, u
pright=True)
# surf=cv2.xfeatures2d.SIFT_create()#可以改为 SIFT
kp1, descrip1 = surf.detectAndCompute(img1, None)
kp2, descrip2 = surf.detectAndCompute(img2, None)

FLANN_INDEX_KDTREE = 0
indexParams = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
searchParams = dict(checks=50)

flann = cv2.FlannBasedMatcher(indexParams, searchParams)
match = flann.knnMatch(descrip1, descrip2, k=2)

good = []
for i, (m, n) in enumerate(match):
    if (m.distance < 0.75 * n.distance):
        good.append(m)

if len(good) > MIN:
    src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-
1, 1, 2)
    ano_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-
1, 1, 2)
    M, mask = cv2.findHomography(src_pts, ano_pts, cv2.RANSAC, 5.0)
    warpImg = cv2.warpPerspective(img2, np.linalg.inv(M), (img1.shape[1
] + img2.shape[1], img2.shape[0]))
    direct = warpImg.copy()
    direct[0:img1.shape[0], 0:img1.shape[1]] = img1
    simple = time.time()

    # cv2.namedWindow("Result", cv2.WINDOW_NORMAL)
    # cv2.imshow("Result",warpImg)
```

```
    rows, cols = img1.shape[:2]

    for col in range(0, cols):
        if img1[:, col].any() and warpImg[:, col].any():  # 开始重叠的最
左端
            left = col
            break
    for col in range(cols - 1, 0, -1):
        if img1[:, col].any() and warpImg[:, col].any():  # 重叠的最右一
列
            right = col
            break

    res = np.zeros([rows, cols, 3], np.uint8)
    for row in range(0, rows):
        for col in range(0, cols):
            if not img1[row, col].any():  # 如果没有原图，用旋转的填充
                res[row, col] = warpImg[row, col]
            elif not warpImg[row, col].any():
                res[row, col] = img1[row, col]
            else:
                srcImgLen = float(abs(col - left))
                testImgLen = float(abs(col - right))
                alpha = srcImgLen / (srcImgLen + testImgLen)
                res[row, col] = np.clip(img1[row, col] * (1 - alpha) +
warpImg[row, col] * alpha, 0, 255)

    warpImg[0:img1.shape[0], 0:img1.shape[1]] = res
    final = time.time()
    img3 = cv2.cvtColor(direct, cv2.COLOR_BGR2RGB)
    plt.imshow(img3, ), plt.show()
    img4 = cv2.cvtColor(warpImg, cv2.COLOR_BGR2RGB)
    plt.imshow(img4, ), plt.show()
    print("simple stich cost %f" % (simple - starttime))
    print("\ntotal cost %f" % (final - starttime))
    cv2.imwrite("simplepanorma.png", direct)
    cv2.imwrite("bestpanorma.png", warpImg)

else:
    print("not enough matches!")
```

5.结果演示

我们以两组图片进行结果演示。

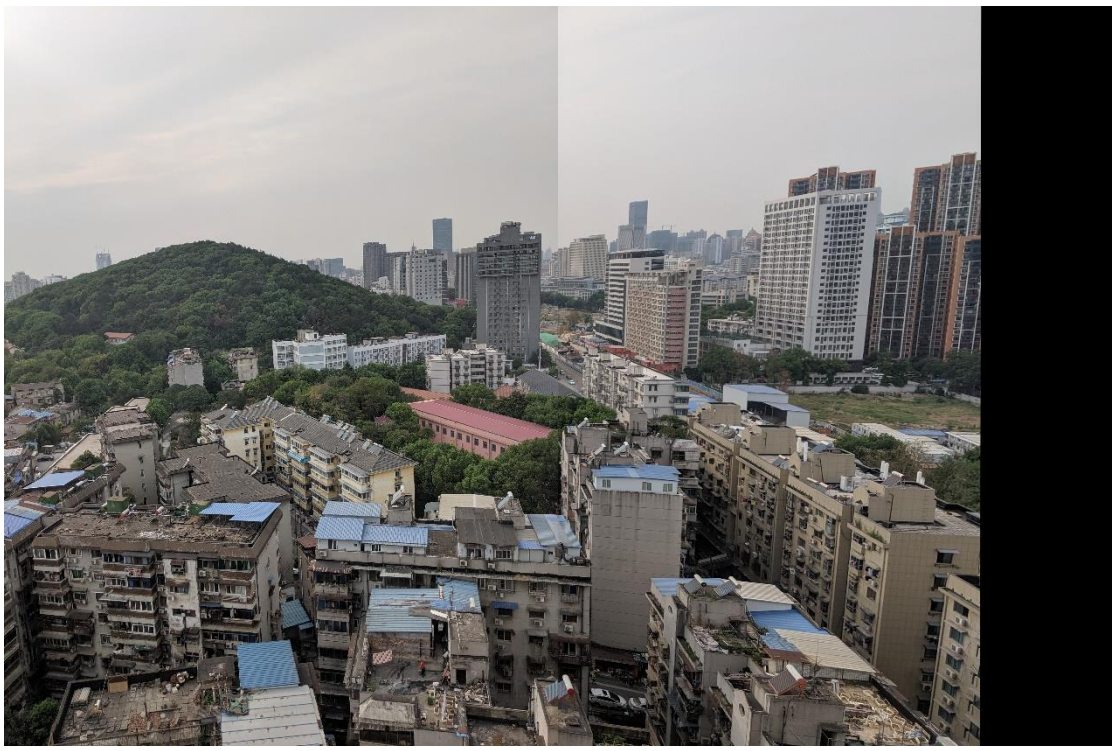
（1）第一组

输入原图：IMG_201901.jpg、IMG_201902.jpg

输出拼接图：simplepanorma.png、bestpanorma.png

IMG_201901.jpg

IMG_201902.jpg



simplepanorma.png

bestpanorma.png

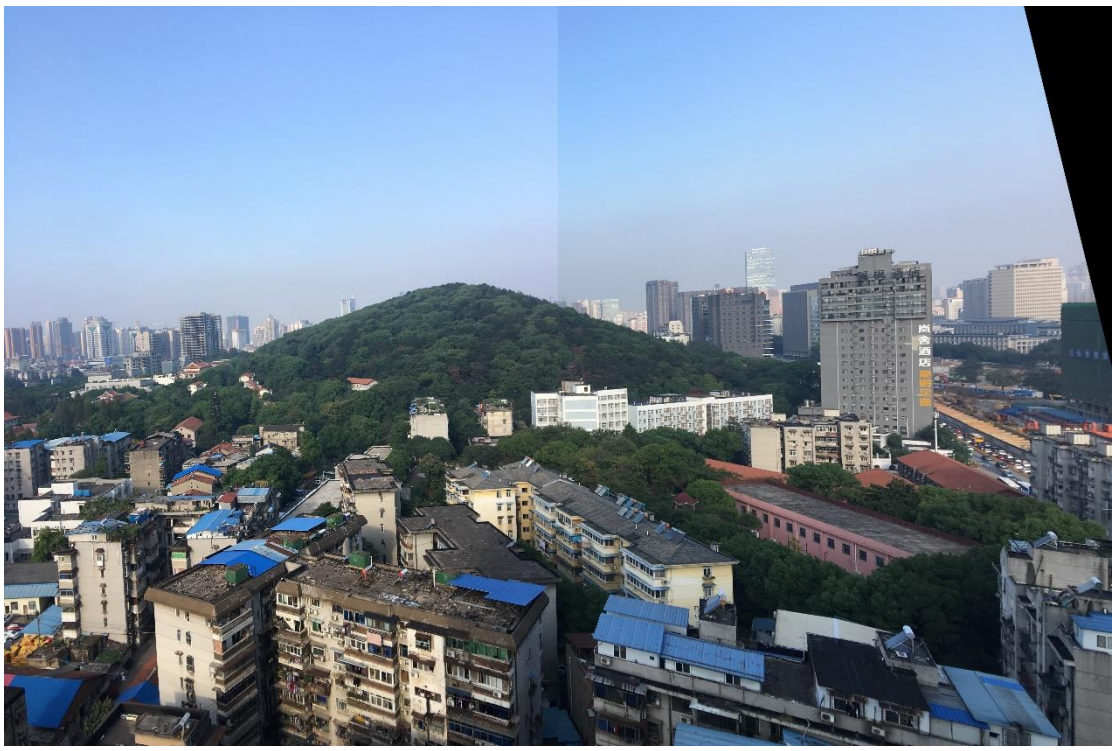
（2）第二组

输入原图：IMG_8293.JPG、IMG_8294.JPG

输出拼接图：simplepanorma1.png、bestpanorma1.png

IMG_8293.JPG

IMG_8294.JPG



simplepanorma1.png

bestpanorma1.png

6.结论

使用基于 SURF 算法的图像拼接技术可以快速地对同一场景下具有重叠位置的且具有相似特征的两幅图片实现拼接，再对重叠区域作平滑度处理后，拼接的效果也十分优良。

本次实验对图像拼接这一任务有了较好的实现。

7.参考文档

OpenCV 官方教程中文版（For Python）