

Computer Vision: Representation and Recognition

Assignment 3

211220075, 孟旷宇, 1665113825@qq.com

2024 年 6 月 12 日

1 Image Mosaics

1.1 Getting correspondences

实现细节见 *get_correspondences.m* 文件。该函数会显示需要拼接的两张图片并唤起 *ginput* 函数，要求用户为两张图片选择至少四对对应点，最后将对应点的坐标返回。

1.2 Computing the homography parameters

实现细节见 *homography_matrix.m* 文件。该函数会对传入的对应点计算单应性矩阵，具体公式推导和数学细节不再赘述。

1.3 Warping between image planes

具体细节见 *Warping.m* 文件。该函数会将上一步计算的单应性矩阵与传入的图像相乘得到源图像的每个像素在目标图像坐标系下的坐标，并对这些坐标取 $\max(x)$ 、 $\max(y)$ 、 $\min(x)$ 和 $\min(y)$ 确定覆盖范围，再反向映射回源图像中以进行 inverse warp 操作得到在目标图像坐标系下源图像的表示。

1.4 Create the output mosaic

具体细节见 *output_mosaic.m* 文件。该函数会根据变形后的源图像在目标坐标下的位置创建画布，并在画布上先后用 **变形后的源图像、目标图像** (顺序不可调整) 进行覆盖，保证目标图像中的像素不受影响，但是目标图像外侧有源图像的部分像素。

1.5 Apply your system to the provided pair of images, and display the output mosaic.

测试该任务请直接调用 *test_for_img_mosaic.m* 脚本文件，并注意更换脚本文件中读入的图像名称。

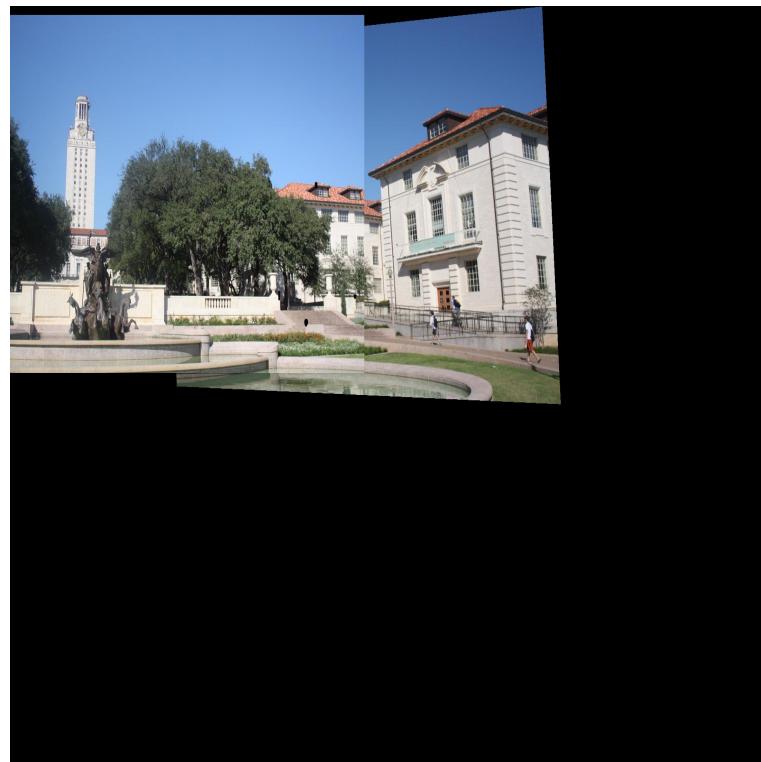


图 1: 拼接结果

对应点选择情况 (只展示一副图像的选择):

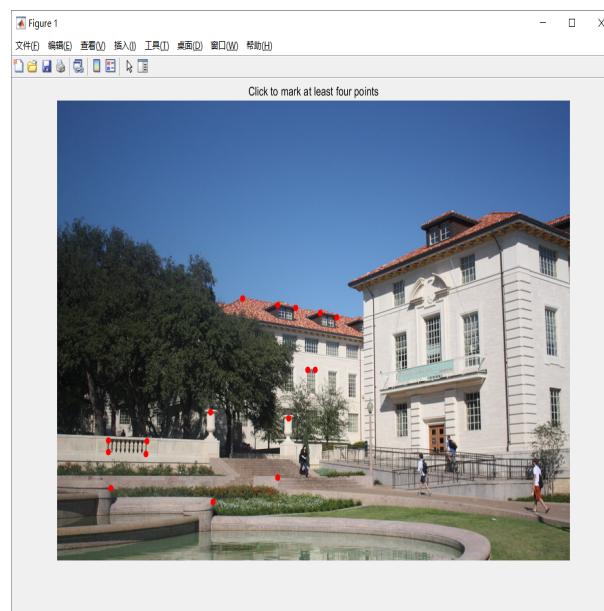


图 2: 对应点选择

1.6 Show one additional example of a mosaic you create using images that you have taken.

测试该任务请直接调用 `test_for_img_mosaic.m` 脚本文件，并注意更换脚本文件中读入的图像名称。

选择的两幅图片：



图 3: 选择的图片之一



图 4: 选择的图片之二

拼接结果：

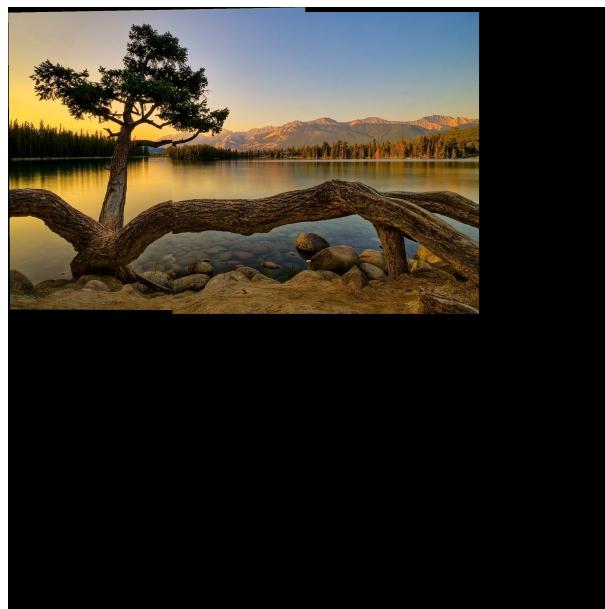


图 5: 拼接结果

对应点选择情况 (只展示一副图像的选择)：

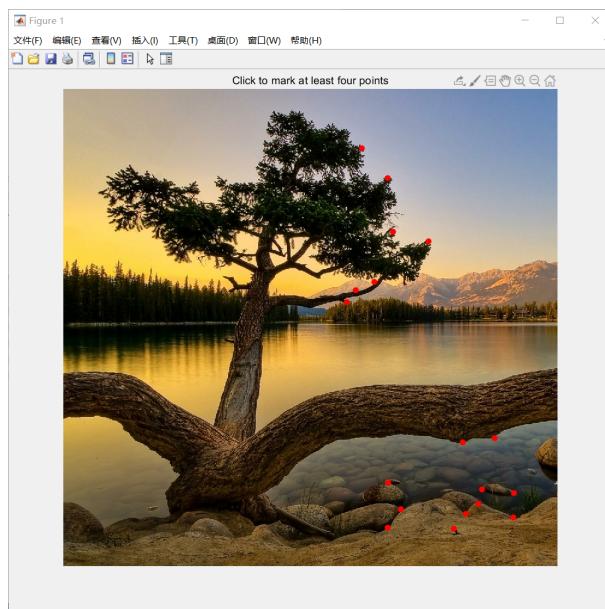


图 6: 对应点选择

1.7 Warp one image into a “frame” region in the second image.

本项任务我认为不太属于图像拼接的任务，反而更偏向于图像嵌入任务，因此我为本项任务写了一个新的脚本文件 `test_for_img_embedding.m`，该脚本文件会让调用者选择要嵌入的

对应点的位置（插入图像为四个顶点；被插入图像为对应矩形区域的四角），并由此计算单应性矩阵将需要嵌入的图像变形后覆盖在目标图像的对应区域中。

准备的图像如下：



图 7: 要嵌入的图像



图 8: 被嵌入的图像

嵌入结果如下：



图 9: 嵌入结果

2 Automatic Image Mosaics

2.1

具体细节请见 `automatic_test_for_img_mosaic.m` 脚本文件，该脚本文件会调用 `vl_sift()` 与 `vl_ubcmatch()` 函数自动获取对应点，并在处理后传给之前写的求单应性矩阵、图像变形与图像拼接函数得到所求结果。

需要注意的是：在调用此脚本前请详细阅读脚本文件中的注释信息配置 `vlfeat` 的函数库，否则会因无法找到相关函数而报错；另外，在调用 `vl_ubcmatch()` 函数时，获取到的对应点的质量受第三个参数 `THRESH` 的影响，`THRESH` 越大速度越快，选择的对应点越少；`THRESH` 越小运行越慢，选择的对应点越多但是错误匹配的点对也会增加，我选择的 `THRESH` 是 10.0，这个值计算出来的拼接图像的质量更好。

拼接结果如下：

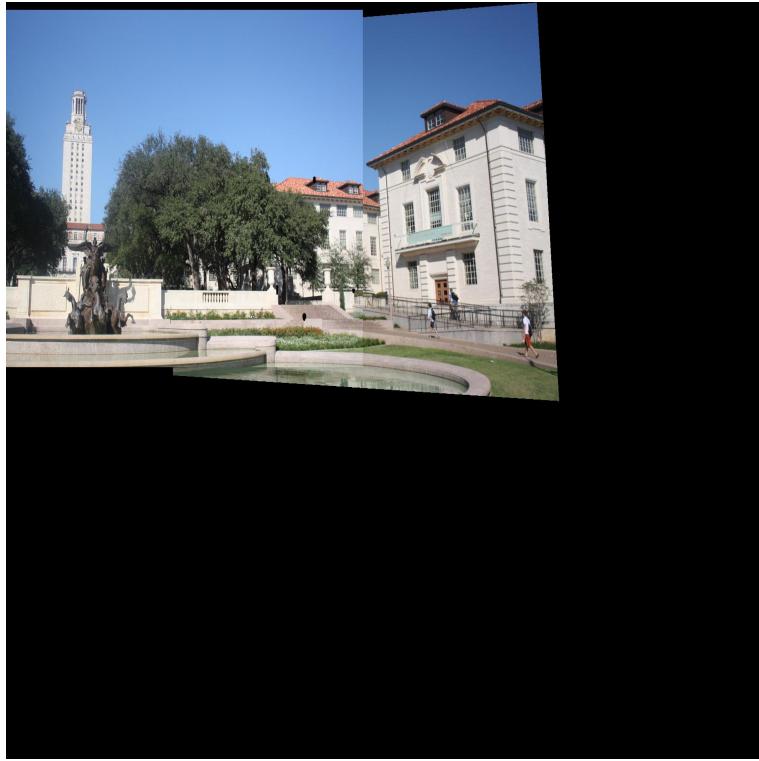


图 10: 拼接结果

2.2

RANSAC 的具体实现见 *RANSAC.m* 函数文件。

该函数会先从输入点中随机取样 25 个点（如果输入点对数不够请修改此处的值）作为初始内点；

接着迭代若干次（次数可以自己选择，我选择的迭代轮数为 10），每次根据目前选出的内点调用之前写过的 *homography_matrix()* 函数来求单应性矩阵，并将这个矩阵与源图像的输入点相乘得到它们在目标图像中的对应位置，接着计算这些位置与实际位置的欧氏距离，取欧氏距离小于 30（这个界值由自己选择，我的选择是 30）的点对作为下一轮内点；

另外，在迭代途中，如果发现当前计算的内点数大于曾经求过的最大内点数，就更新最大内点数并将当前轮次求得的单应性矩阵作为输出结果进行更新。

使用 RANSAC 消除噪声前后的结果对比：

测试脚本仍然是 *automatic_test_for_img_mosaic.m* 脚本文件，但在调用前请阅读其中的注释将计算单应性矩阵的步骤选择为使用 RANSAC 算法的那一种。

另外，为了使对比更加鲜明，我将 *vl_ubcmatch()* 函数的第三个参数调整为 4.0，这样噪声点更多，消除前后反差鲜明。

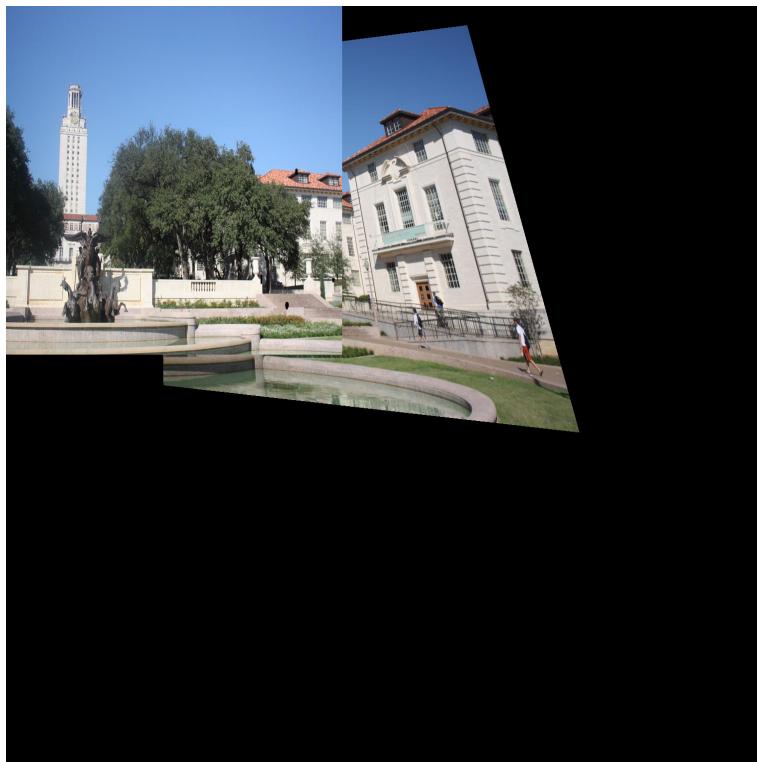


图 11: 不使用 RANSAC 的图像拼接结果

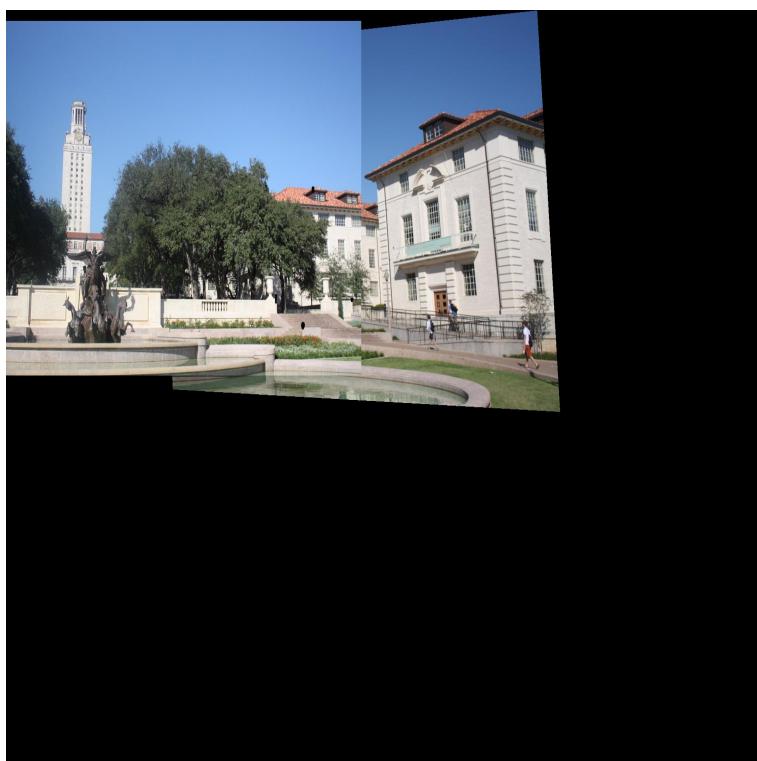


图 12: 使用 RANSAC 的图像拼接结果

reference