Computer Vision: Representation and Recognition
Assignment 3

211220028, 党任飞, 211220028@smail.nju.edu.cn

2024 年 6 月 13 日

1 Image Mosaics

1.1 Getting correspondences

根据手册提示，使用 ginput 函数在两张图片上分别选择四对两两对应的点，并保存为.mat 文件。

具体实现见 *selectPoints.m* 文件。

1.2 Warping between image planes

读取上一步得到的数据（四对点），计算单应性矩阵，并验证该矩阵是否能够成功实现映射。

其中，计算单应性矩阵原理如下。假设有点对 $p_1 = (x_1, y_1)$ 和 $p_2 = (x_2, y_2)$，将其转化为齐次坐标 $p_1' = (x_1, y_1, 1)$ 和 $p_2' = (x_2, y_2, 1)$。现在要找到矩阵 $H \in \mathbf{R}^{3\times3}$，使得 $Hp_1' = p_2'$。记

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{1.1}$$

其中 $h_{33} = 1$，则可以把 H 拉直为一个 8 维向量；且根据 $p_1' = (x_1, y_1, 1)$ 和 $p_2' = (x_2, y_2, 1)$，满足如下关系：

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_2x_1 & -x_2y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_2x_1 & -y_2y_1 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} \tag{1.2}$$

可以方便地用求解线性方程组的方法得到矩阵 H。

测试结果如下：

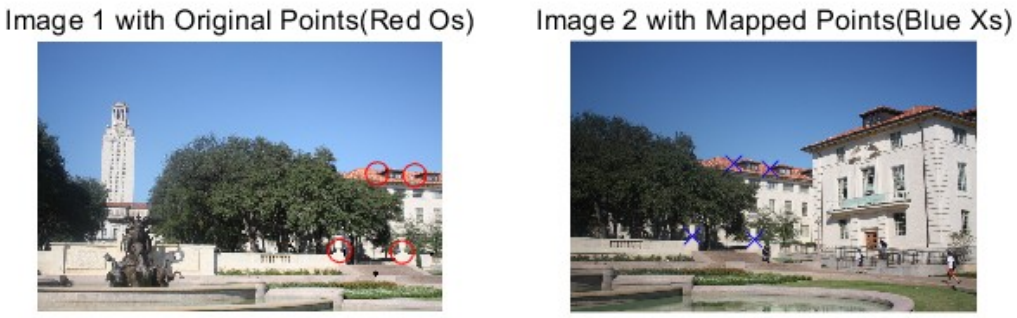

图 1: 矩阵 H 测试结果

其中左图的红圈表示四个点，经过矩阵 H 映射后对应的坐标就是右图蓝叉对应位置。这表示矩阵 H 成功地将左图参考点对齐到右图的坐标系下。

具体实现细节在 $computeH.m$ 文件中。

1.3 Computing the homography parameters

使用之前得到的矩阵 H，对图一进行变换，并拼接到图二上。

具体实现细节在 $warp.m$ 文件中。其中实现了函数 $warpImg(H, img)$，用于将给定的 img 通过 H 映射到另一个参考系下；以及函数 $createMosaic(warped\_img, img2, ...)$ 用于将映射后的图像 $warped\_img$ 与图二 $img2$ 进行拼接，得到最终的输出。

结果如下：

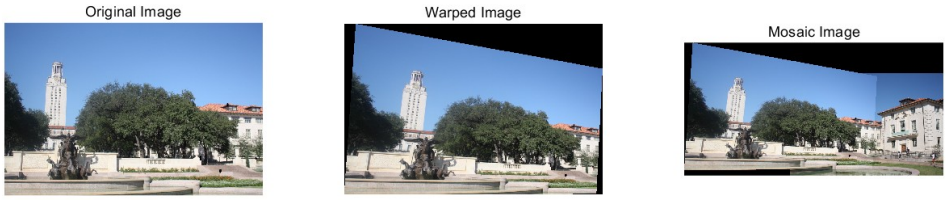

图 2: warp 和 mosaic 结果

1.4 Additional example

在宿舍环境下用我室友作为模特，得到结果如下。从最右边的图片中可以看到不仅同一个室友出现了两次，而且空调水管是成功跨越了图片相连。这是因为我手动地选取了空调水管的一个点作为计算 H 矩阵的参考点之一。

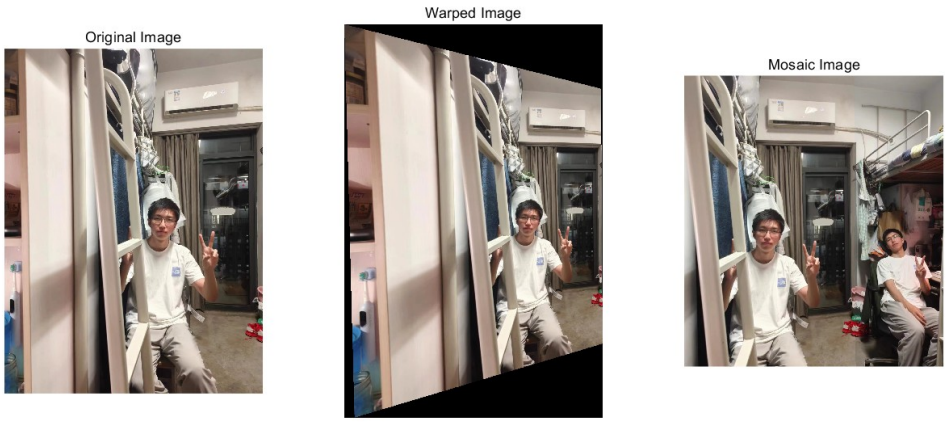

图 3: warp 和 mosaic 结果

1.5 Warp into Frame

以我的宿舍桌面同时为 img1 和 img2，将其递归地映射到我的显示器上，结果如下：

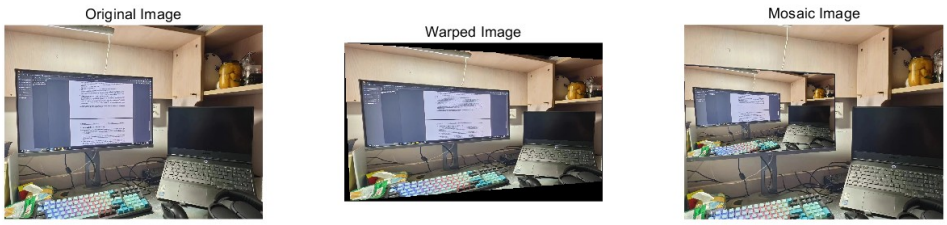

图 4: warp 和 mosaic 结果

2 Automatic Image Mosaics

2.1 VLFeat

修改 *selectPoints.m* 文件，增加 *mode* 参数，可选值为 *manual* 或 *auto*，分别代表手动选择四个点和自动选择 VLFeat 库匹配分数最高的 n 个点。

代码基本逻辑如下，详见 *selectPoints.m* 文件。

```
numPoints = 4;
%mode = "manual";
mode = "auto";

if mode == "manual"
    ......
    ... = ginput(numPoints);
    P1 = [x1, y1];
    ......
end
if mode == "auto"
    run('vlfeat-0.9.21/toolbox/vl_setup');
    ......
```

```
    ... = vl_sift(gray1);
    ......
    ... = sort(...);
end
```

在此之后确实可以随机找到一些 interest points，但是有非常显然的错误。如下图：

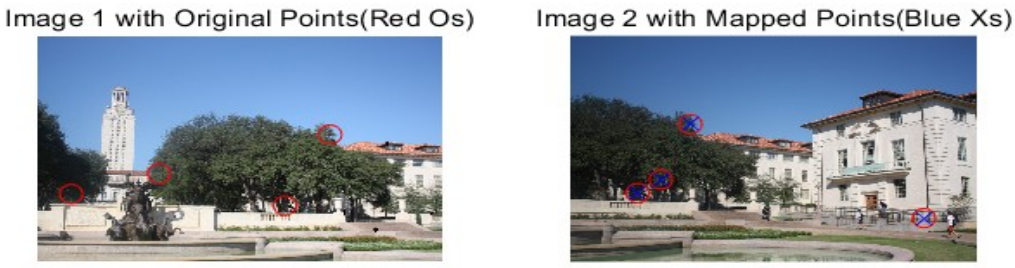

图 5: VLFeat 错误结果

从图中可以看出，虽然我们成功找到了矩阵 H 使点对之间正确映射（右图中红圈为 P2 中选定的点；右图中蓝叉为左图中点经过 H 映射到右图的点），但是显然左图中红圈的四个点与右图中红圈的四个点并不是正确的对应关系。在这样的错误下，直接强行构造 mosaic 会产生报错，因为我的代码逻辑是保留找到被映射到的所有点，如果对应关系找的不好，经过矩阵 H 映射之后可能产生的图片非常巨大，MATLAB 并不能分配这么大的空间。

图 6: 报错信息

2.2 Implement RANSAC

仍然是修改 *selectPoints.m* 文件，使用 $estimateGeometricTransform2D$ 函数来找到正确的点对，丢弃错误的 outliers。基本逻辑是用 VLFeats 找到所有可能的配对点之后，使用 RANSAC 方法找到所有 inliers(这个方法同时会返回 H 矩阵,无需管它,后续会在 *computeH.m* 脚本中自己计算)；然后对所有 inliers 的分数进行排序，选取最高的 n 个作为 interst points。

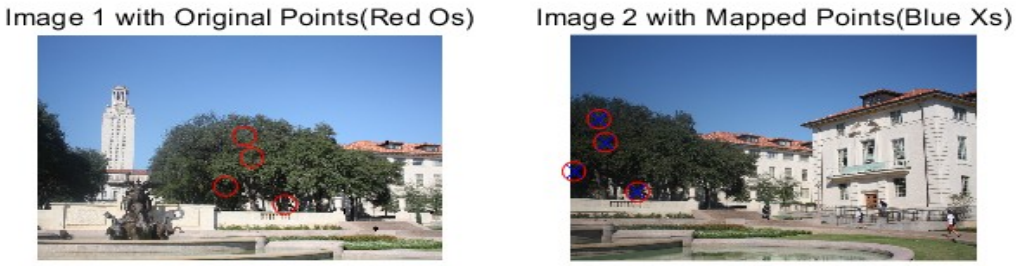

图 7: RANSAC 找到的点对映射

从图 7 中可以看到，经过 RANSAC 之后的高分点对之间有正确的映射关系。

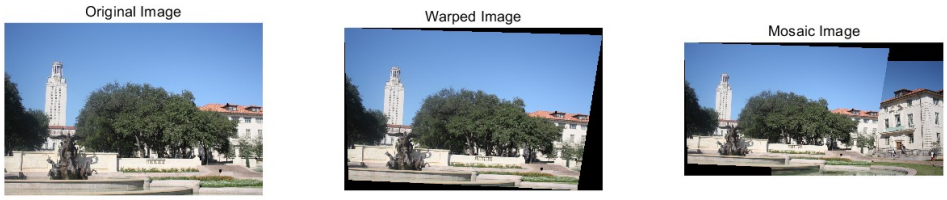

图 8: 最终结果

从图 8 中可以看出，此时找到的映射关系基本上正确，可以输出最终的 Mosaic 图片。但是其边缘的细节处理并不如手动选择的效果好。

下面考察复杂情况下自动模式的能力。使用在宿舍拍摄的我室友的照片进行测试，其结果如下所示。

图 9: 复杂场景下的点对

图 10: 复杂场景结果

可以看出，自动化找到的点对趋向于从同一片邻域内找，本例中也许是因为宿舍地面有一些细节比较相似，四个点中有三个都是从地面找的。相近的点造成计算得到的 H 矩阵映射比较特殊，将原图拉伸得特别长。最终的输出结果很差。作为对比，图 3 手动选择的参考点输出结果，甚至可以做到将空调的水管跨越图片相连，效果很好。

然后联想到让程序自动化多找一些点。之前的所有实验都是基于 4 个点，如果修改为十个，效果如下所示。

图 11: 10 个点对

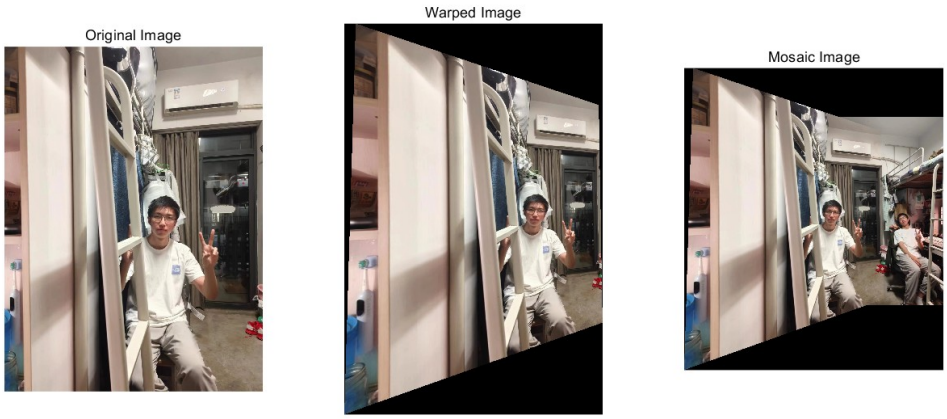

图 12: 10 个点对结果

可以从上图明显看出，虽然自动选点仍然集中几个邻域内，但是可能是由于点多了，效果变得非常好，边缘非常丝滑。然后再测试一下本来就有的两个图片，自动选取十个点，效果依然很好。得出的结论是一定范围内，选点越多效果看起来可能越好。

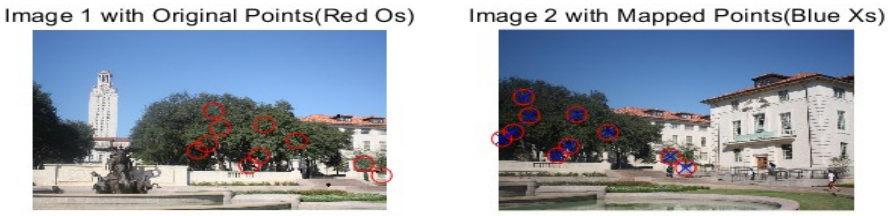

图 13: 10 个点对

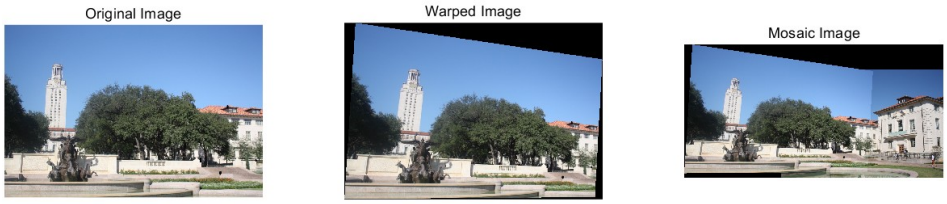

图 14: 10 个点对结果

3 Appendix

程序运行方法：

1. 进入 solutions/目录。
2. 根据需求，修改三个文件中的 img1、img2 路径。选择 mode（手动或自动选择 interst points）。
3. 按顺序分别执行 selectPoints.m、computeH.m、warp.m 三个脚本，即可得到结果。