

计算机视觉和模式识别 作业 4

13331231

孙圣

计应 2 班

一、使用说明

MAC OSX 系统：通过 sh execute.sh 直接编译运行即可（需要安装 opencv），默认对 dataset2 第 1-3 张图片进行拼接。要对其他图片进行测试，执行 ./a.out datasetX/X1.jpg datasetX/X2.jpg... 即可。

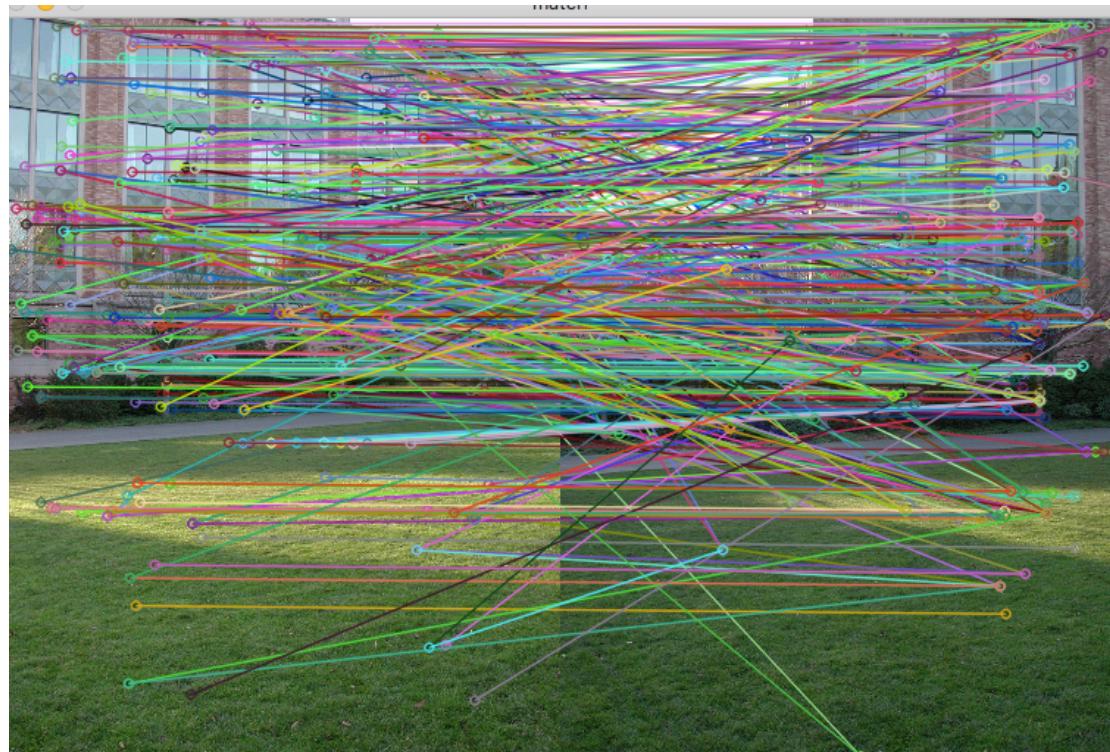
Windows 系统，在 cmd 中输入 execute.bat 对 dataset2 第 1-3 张图片进行拼接。或者执行 execute.exe datasetX/X1.jpg datasetX/X2.jpg...。

部分拼接的结果放在了 stitched 文件夹内。

二、实验过程

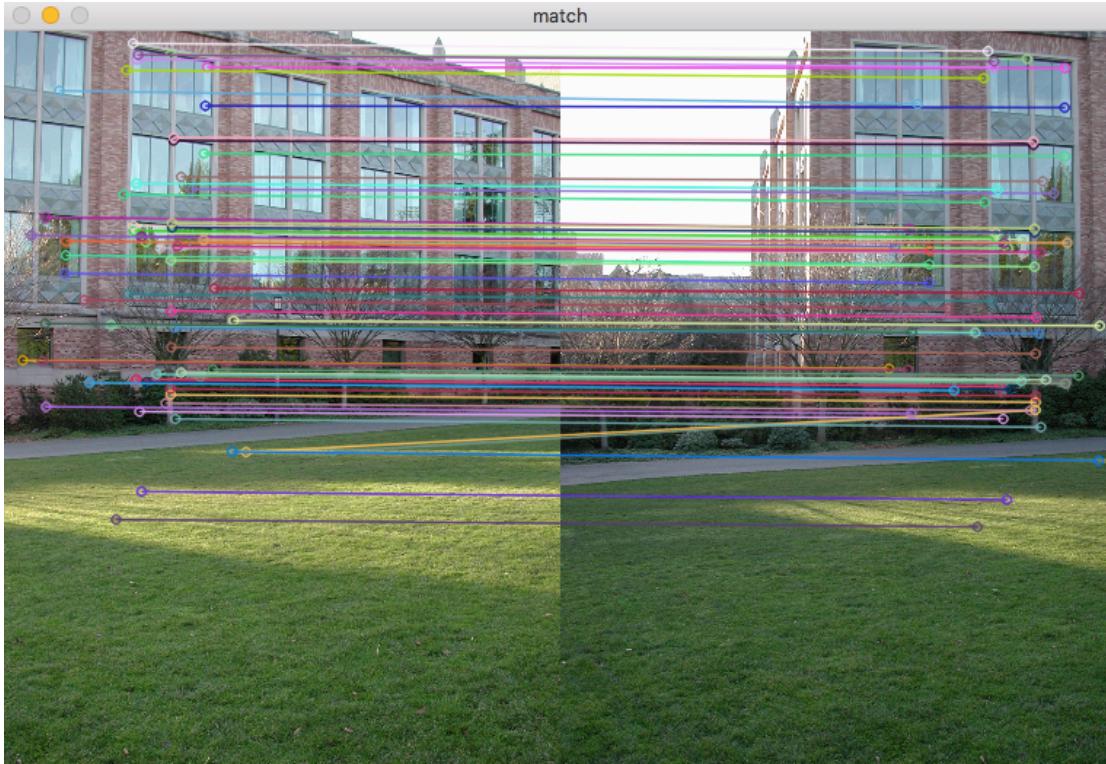
由于是要对多张图片进行拼接，所以考虑先将所有图片读入，之后不断调用 stitch(); 函数进行拼接。

stitch() 函数中，首先将两张图片转换为灰度图，之后利用 SIFT 检测得到相应的关键点。然后将这些关键点匹配起来，如图：



此时，关键点之间的连接关系相当复杂，如果直接使用 RANSAC 求出单应矩阵，需要的迭代次数会相当的大，影响程序效率，而且结果不一定准确。

因此再根据关键点之间的距离筛选出相对较好的关键点：我们先求出关键点之间的最小距离，之后把 3 倍的最小距离作为阈值进行筛选，一般都会剩余 5-80 对的关键点，如图所示：



尽管进行了初步的筛选，还是存在错误的匹配，因此需要用 RANSAC 来求出合理的 Homography。

首先，通过 rand() 随机函数，产生四个不重复的随机数，这里利用了 STL 的 set 来实现：

```
// Randomly choose samples
set<int> index;
for ( int i = 0; i < samples; ) {
    int r = rand() % numberofMatches;
    if ( index.count(r) == 0 ) {
        index.insert(r);
        ++i;
    }
}
```

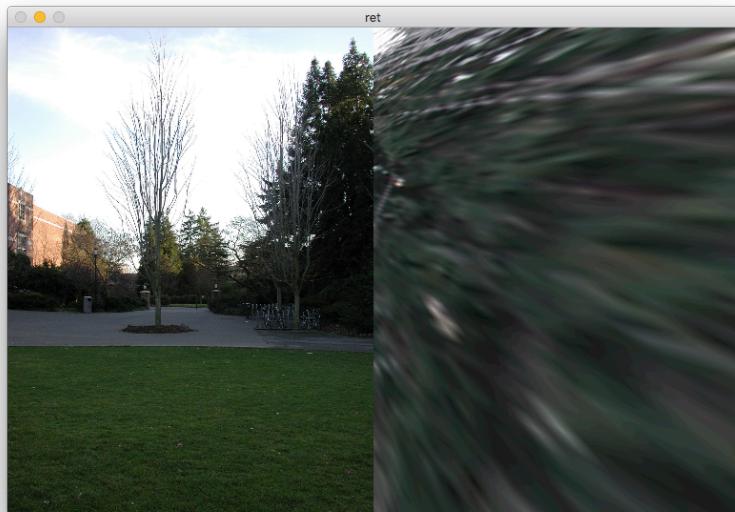
之后根据这四个下标取出相应的点，然后利用 findHomography(); 函数求出 H，这里的第三个参数设为 0，即使用所有点而不是 opencv 库所提供的 RANSAC 方法。然后对左图的所有的点进行 perspectiveTransform，之后统计

```
inliners:
```

```
// Count inliners
const int filterThreshold = 3;
int inliners = 0;
for ( int i = 0; i < number0fMatches; ++i ) {
    if ( norm(transformedMatched[i] - matchedPt1[i]) < filterThreshold ) {
        ++inliners;
        // cout << transformedMatched[i] << " " << matchedPt1[i] << endl;
    }
}
```

这里利用了 3 倍的两点之间的距离作为阈值，进行筛选 inliers。然后判断是否是最大值，如果是，返回相应的匹配的点。

RANSAC 中有一个迭代次数的参数需要调节，对于大部分图片的拼接，迭代 100 次左右就能成功得出 H，但是对于 dataset2 中的第 6 和第 7 幅图，会得出错误的 H，拼接结果如图：



因此尝试较大的迭代次数，例如 1000 次，此时才能成功拼接。



原因分析：可能是两张图中各种树的相似度较大，因此得出的 good matches 的数目较多，有大约 165 个，因此较少次数的迭代不能保证能够求出最佳的 H 使得 inliners 的数目最多。

对 Max inliners 数据的比较：

```
[MacBook-Air-3:Ex4 sunsheng$ sh execute.sh
Stitching image 1:
Good matches ratio: 165 / 500
Max inliner ratio: 11 / 165
Reverse:
^C
[MacBook-Air-3:Ex4 sunsheng$ sh execute.sh
Stitching image 1:
Good matches ratio: 165 / 500
Max inliner ratio: 23 / 165
Reverse:
```

第一个执行输出为迭代 200 次的数据，第二个执行的输出为迭代 1000 次的数据，可见 Max inliners 的数量翻了一倍。

完成了 RANSAC 之后，还有重要的一步就是要判断两张图片的左右位置。我的处理方法是判断对应关键点的 x 的坐标。对于在左边的图片，它的匹配的关键点的应该是聚集在图片的右侧，而对于在右边的图片，情况恰好相反，关键点主要聚集在图片的左侧。因此根据这一特性，来判断两张图像是否需要交换位置：

```

// Decide whether to switch
int reverseCount = 0;
for ( int i = 0; i < filteredPt0.size(); ++i ) {
    if ( filteredPt0[i].x < filteredPt1[i].x ) {
        ++reverseCount;
    }
}

const double reverseThreshold = 0.75;
if ( reverseCount > reverseThreshold * filteredPt0.size() ) {
    cout << "Reverse: " << endl;
    //srcImg0 = merge(matchedPt1, matchedPt0, srcImg1, srcImg0);
    srcImg0 = merge(filteredPt1, filteredPt0, srcImg1, srcImg0);
} else {
    //srcImg0 = merge(matchedPt0, matchedPt1, srcImg0, srcImg1);
    srcImg0 = merge(filteredPt0, filteredPt1, srcImg0, srcImg1);
}

```

对于不同情况，对最后拼接函数 merge(); 的调用也是不一样的。

merge(); 函数所做的仅仅是求出相应的 H，对右侧的图像进行 warpPerspective，然后拼接再一起。

可是，这样简单的拼接在拼接点处会有非常明显的边界痕迹，因此考虑对两附图重叠的部分进行 blending，但是暂时还没有成功。

三、实验结果

1. 对于 dataset1 中的图 1-3 进行拼接：

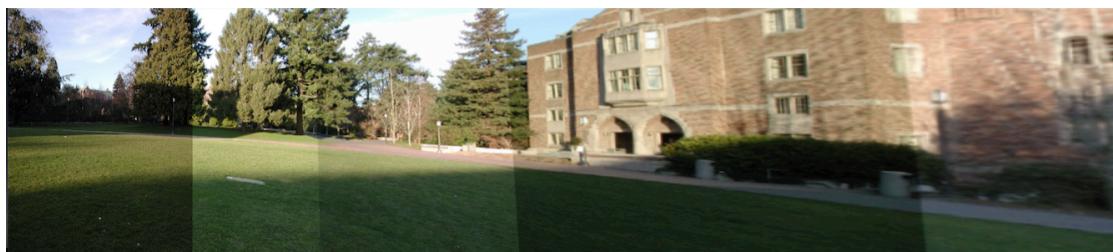


2. 对于 dataset2 中的图 1-4 进行拼接：

由于 dataset2 中的图片的分辨率都相对较大，对这四张图片的拼接大约用了 7 分钟，时间效率相对较低。而且拼出来的图片分辨率约为 6000 X 2000，有 37M 的大小。



3. 对于 dataset2 中的图 1-6 和图 10-15 进行拼接:



可见，当拼接的图片数目增多时，右侧的图片会不完整。原因为：每次进行拼接时，右侧的图片都会进行一次变换，而在变换的过程中，图片会溢出相应的边界，造成只留下了图片的下半部分在拼接好的图片中。

因此考虑将图片拼接的顺序修改，使得图片从左向右拼接，而不是从右向左，结果如下：



发现这样拼接就失败了，原因是：在进行第一个拼接时，右侧的图片进行了一次变换，相应的点也被扭曲了，因此再和下一张图片进行关键点匹配时就会出现问题，造成拼接失败。

因此，尽管程序能够将很多张图片拼接起来，但是效果并不理想，具体的解决办法还不太清楚。

参考资料：

[1] SIFT

http://docs.opencv.org/2.4/doc/tutorials/features2d/feature_homography/feature_homography.html

[2] Pixel coordinates

<http://stackoverflow.com/questions/8436647/opencv-getting-pixel-coordinates-from-feature-matching>

[3] Blending

<http://blog.csdn.net/pi9nc/article/details/9251387>