

图像裁缝师：图像配准技术在图像拼接中的运用

胡一凡¹, 李一帆² and 陈学添³

¹Stu Num: 19307130010 SIFT 算法框架实现、完成测试实验、部分数据结果可视化、报告书写

²Stu Num: 19307110499 优化重构主程序代码、单应性测试实验设计、GUI 框架搭建、报告书写

³Stu Num: 19307110189 去除 SIFT 的重复特征、搭建主程序框架、优化 GUI 窗口、报告书写

摘要

本文运用图像配准技术，完成图像拼接任务。基于 Python，借鉴 Opencv 源码，实现了 SIFT 算法，并使用 SANSAC 技术拟合最佳投影矩阵。本文还进一步探讨了不同相似性度量、不同拟合点数下，使用 SANSAC 算法进行回归的优劣。使用了 Linear Blending 方法优化拼接效果。最后，我们还探讨了该项目的不足之处，比如拼接顺序的敏感性、重叠区域鬼影问题、自主实现的 SIFT 速度较慢等等，并对未来进一步优化工作做出展望。

Keywords: 图像配准, SIFT, SANSAC, Linear Blending

1 Introduction

全景图像拼接是图像配准任务中的一种，在现实中广泛运用。比如在旅游中，人们往往喜欢运用全景拍摄来记录奇观异景。这项技术就需要运用到图像拼接：通过连续拍摄多张照片，通过图像拼接的技术，把照片一张一张连接起来，进行适当的裁剪，返回用户。

全景图像拼接技术一般分为如下几个步骤：提取两张图像的特征点；在此基础上，匹配两幅图片的相似特征（也就是重合的特征部分）；设计变换函数，使变换后的两张图片在上一步发现的的特征能够尽可能地重合。最后将两张变换后的图像进行拼接，达到以假乱真的地步。

在本次报告中，本小组致力于完成这一项拼接技术。使用 SIFT 进行特征提取，使用穷举方式匹配特征，使用投影变换与图像逆变换的方式，进行图像拼接。最后使用测试图像进行验证展示。同时，我们设计了一款 GUI 进行效果可视化。此外，还对于传统的图像拼接方式进行了一定的改进。

2 问题和数据描述

在本次实验，我们需要的输入数据是两张待左右拼接的图片。图片可以是以不同视角拍摄的，但其中含有重叠的景物，我们需要根据重叠区域的信息特征，计算两张图片的投影角

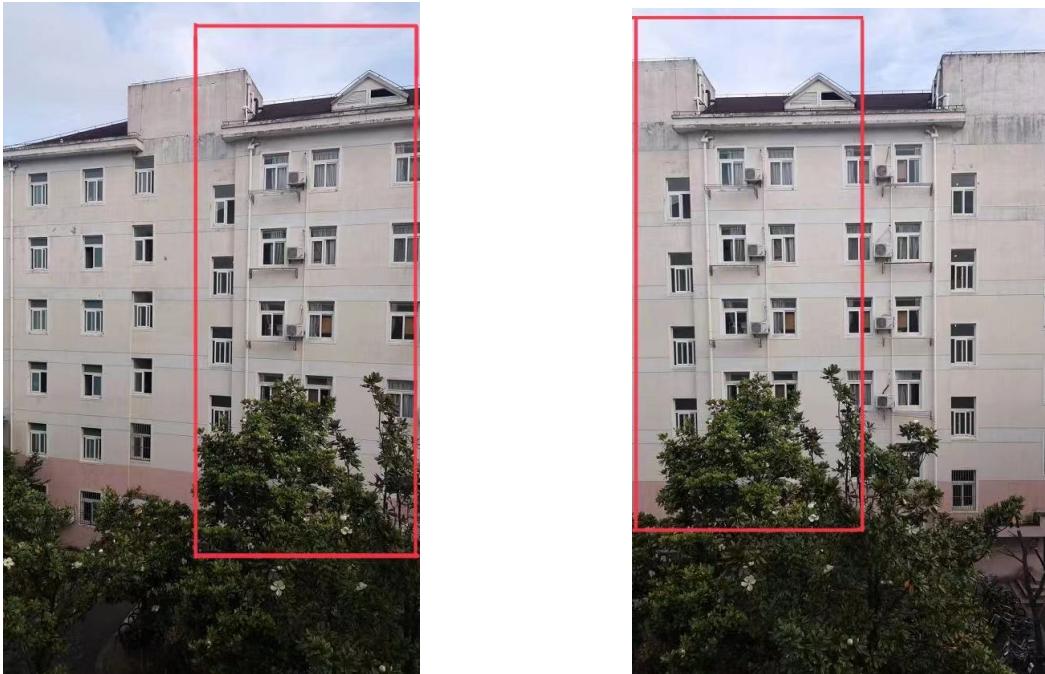


图 1. 输入图像示例，红框内为两张图的重叠区域

度差异，并将右图进行相应的投影变换，和左图重叠景物进行配准，最后输出一张拼接好的图像。

3 开发环境

在 Windows 系统上开发。实验使用的 CPU 型号为 i3-8130U。不使用 GPU 进行图像处理。开发语言为 Python 3.8，需要 Opencv, pillow, 以及 numpy, matplotlib 等基本包支持。最后可视化执行程序为 Python 文件。

4 主要流程

如上所述，本次任务中的全景图像拼接大致分为下列四个步骤：

1. **特征检测和描述** 实现自动检测图像特征点的算法，并生成对每个特征点的描述向量。在本任务中，我们采用 SIFT 算法进行特征点的检测和描述。
2. **特征点匹配** 利用特征点的描述向量的相似性，寻找将两幅图像生成的特征点之间的对应关系。在全景图像拼接任务中，这种对应关系即是两幅图像重合的区域。本任务中，对左图的每一个特征点，我们寻找与其特征向量距离最近的右图特征点作为其对应点。
3. **拟合变换** 利用上步中找出的特征点对作为基准点，尝试拟合右图到左图之间的变换关系。考虑到全景图像常从近处拍摄，视角移动的过程中伴随着非线性变换，我们尝试拟合一个透视变换而非仿射变换。

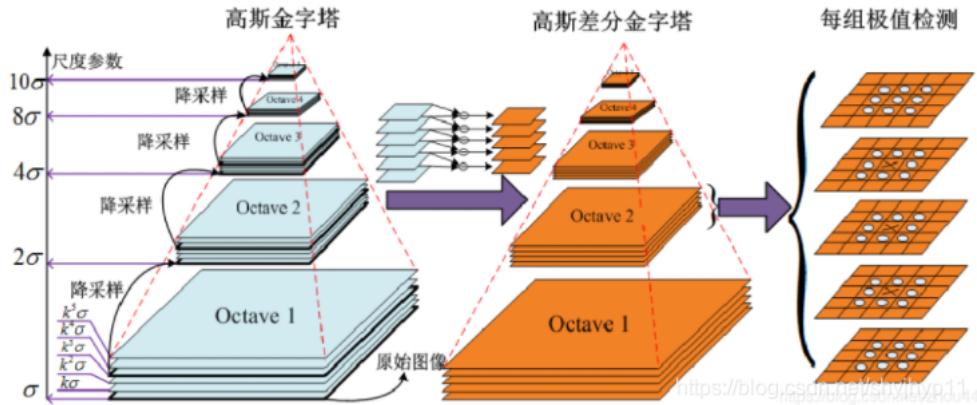


图 2. 左图为高斯金字塔。中间每层经过差分得到高斯差分金字塔。最后进行极值检测，得到特征点。

4. 变换并拼接 将右图的每一个点，经上步拟合的变换映射到左图。

下一节将一一详细描述各个步骤的主要流程和实现方式。

5 实现流程

5.1 特征检测和描述：SIFT (Lowe)

寻找特征点 我们希望图像的特征点对旋转、缩放、亮度变化，这三种变化方式下，保持一定的不变性。同时，这种特征也需要对于噪音有一定的容忍程度。在课堂上，我们学过如何对于图像进行二阶差分。二阶差分得到的图像边界点某种意义上就是一种对于旋转变化有一定容忍度的特征：往往，纹理复杂的比如边缘，点，角之类区域对于二阶导数的地方变化极大。然而，边界信息往往随着图像缩放，其特征点会进行相应的移动。比如下图所示的情况，原本图像的尖角可以作为特征点，而放大以后，图像边界上的尖点变得更加光滑，进而失去了作为特征点的唯一性。

为了解决这一问题，Iijima 于 1962 年引入了尺度空间 (Scale space)。基本思想是：在原来的边界条件基础上，引入不同的模糊系数，将模糊系数和图像尺寸一同作为一个新的参数一同加入到特征的提取过程中。从 2 倍大小的图片，一直到金字塔顶层，各自提取其中的特征，一同作为图像的特征。如此，就能解决在不同的尺度大小下进行特征选择的问题。具体流程为：

1. **构建高斯金字塔** 对于 base_image 以 $(1/n)^j \sigma$ (j 通常为 $n + 3$) 进行高斯模糊；对于该层的第 n 张图片进行下采样，得到上一层金字塔的 base_image 。重复上述操作，直到顶层（设置金字塔层数为 $\lceil \log(\min(M, N)) - 3 \rceil$ ）， M, N 为原始图片边长。
2. **构建高斯差分金字塔** 对于高斯金字塔的每一层，其中 $n + 3$ 张图片，做相邻差分

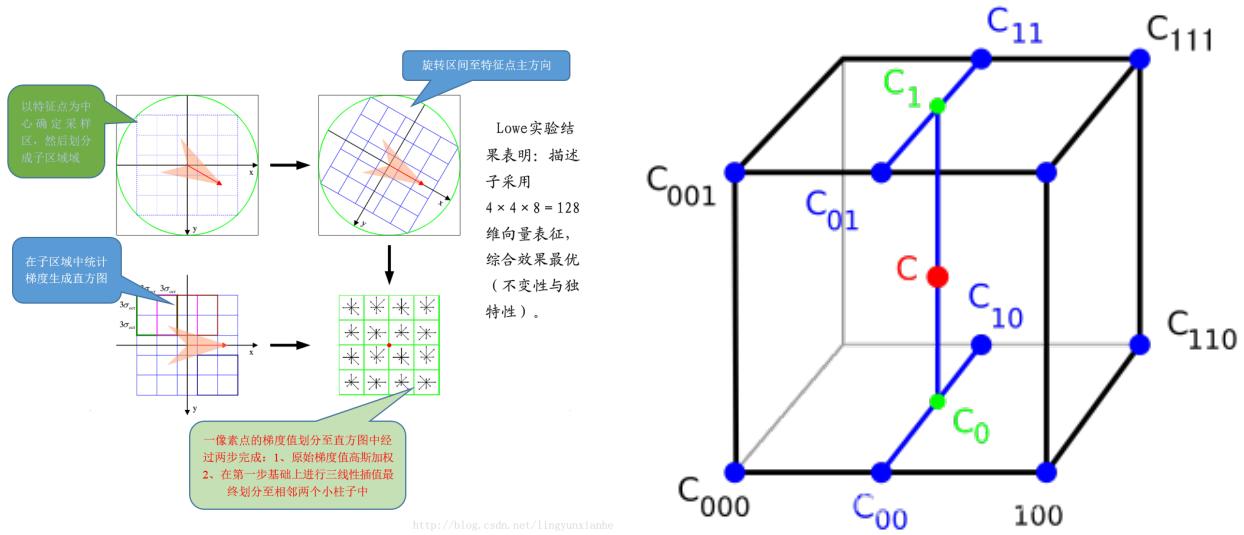


图 3. 左图描述了 SIFT 提取特征向量的过程。右图是三线性插值的示意图。

3. 定位特征点 对于每层 $n + 2$ 的高斯差分金字塔，找到内部的极值点。由于边界点往往不作为极值点考虑，所以去除掉每层金字塔的第一张和最后一张图，在剩下的 n 张图内部找极值点。随后，为了精确定位极值点在原图中的位置，需要使用插值技术，在该区域内部找到精确的极值点，然后乘以缩放倍数，还原到原图。同时，为了保证图像不受噪声干扰，设置阈值超过 $255\alpha/n$ 的极值点作为特征点。其次，需要剔除不稳定的边缘响应点。

相似程度度量 接下来，我们要给这些“特征点”赋予相似程度的度量。这种度量，同样不能收到旋转、光照与缩放的影响。关键的思想在于，一个图像的特征可以由周围的特征进行刻画：一个图像的特征往往是由它和它周围的图像纹理共同决定的。由此，我们就用一个图像周围的梯度直方图情况进行刻画。具体步骤如下：

1. 确定主方向 为了保证旋转不变性，可以把特征点的梯度方向，和其周围的图像一起旋转到统一位置后，进行度量刻画。所以需要确定特征点的主方向。这里选择以特征点为中心的 $3scale$ 的大小区域（ $scale$ 是当前点的模糊参数，刻画了高斯模糊情况），以 $scale$ 为方差高斯核函数，乘以梯度的模长作为权重，按照梯度角度，进行直方图统计。为了去除噪声影响，使用循环插值的方式，对于直方图进行平滑，并以直方图中达到最大值 80% 以上的峰值作为主方向与辅助方向，刻画特征点。
2. 刻画特征 以特征点作为中心，选取周围 4×4 的以 $scale$ 为边长的正方形区域，分别对于每一块小正方形进行梯度直方图统计。具体流程图可以见于上图。值得注意的是，其中梯度统计师，运用了高斯核函数对于梯度进行加权，保证中心梯度权重更大；为了保证梯度计算准确（我们计算的其实是四个角上的梯度，需要通过内插值把权重分配到对应的区域上），运用了三线性插值。

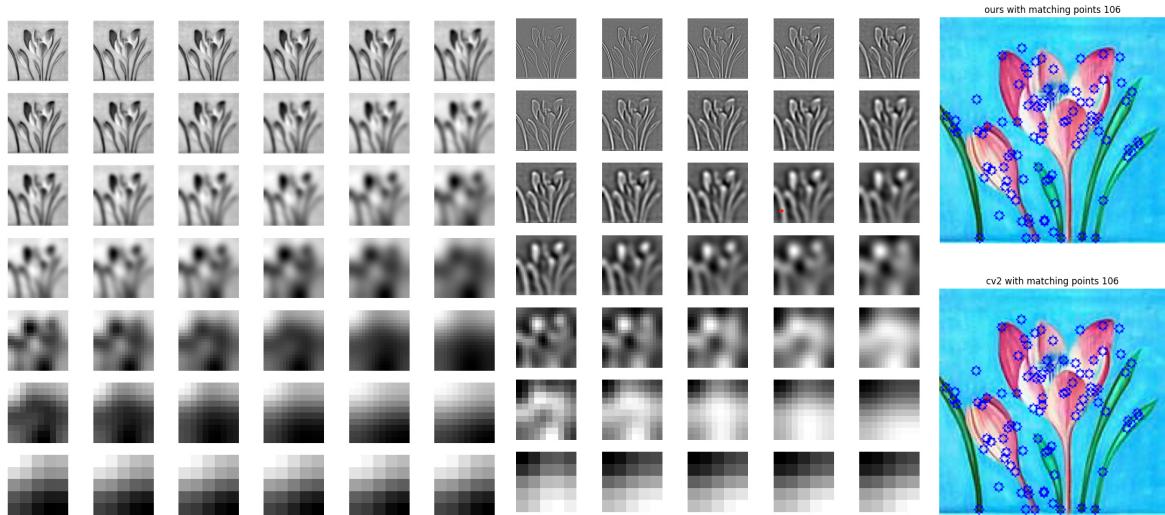


图 4. 左图是高斯金字塔的运算结果。中间是高斯差分金字塔的结构。右图展现了 CV2 和我自己实现的 SIFT 选取特征点的对比

可视化特征选取过程 为了进一步说明特征的选图，我输出了高斯金字塔、高斯差分金字塔以及图像特征点选取结果。可以明显发现，左图从上到小分别是底层金字塔一直到顶层金字塔的处理结果。第一排图片 $300*300$ ，最后一排图片 $5*5$ 。从左至右模糊程度依次提高。高斯金字塔处理结果高斯差分金字塔位于中间，可以发现不同模糊效果、不同图片大小下，图片展示出的特征明显不同：第一排结果关注的更多是边界特征，而底层结果则越来越关注图片的整体特征：比如最后一排左上角明显都是篇黑的，而右下角则是灰度强度更高的地方。接着我运用库函数和我的处理结果进行对比，发现选取的特征点大致相同，说明我的实现应该没有太大问题。

时间开销对比 实验中对于三张不同的图片进行对比。opencv 使用了 C 源码，在速度上有更大的优势，开销时间大致是我们使用时间的 0.1 倍。我们的方法选取的特征更少。这一点需要进一步分析。实际上，在该任务上我们实现的方法在特征选取上更优：事实上，看以下示例图片。门框内部的点不应当作为特征，他们并非边界，也非关键特征。然而从 cv2 库调用的特征提取函数却会选择那些点作为特征进行筛选，可能 cv2 内部有额外的实现机制。不过这点细微的差别不影响说明我们实现的算法的正确性。

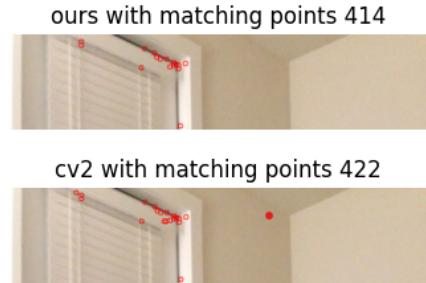
5.2 特征点匹配

上一步中，我们提取到了特征点和特征向量。现在，我们只需要按照特征向量的相似程度，对特征点进行配对即可。相似度由向量间的 L_2 范数定义的距离度量。对于每一个左图的特征点，我们寻找与其距离最近的右图特征点。

然而，简单地由特征向量的距离配对特征存在一个问题，在图片中往往会出现多个相似的特征点，但是特征点的位置并非对应。一个典型的例子是富含重复元素的图片，如图 4 中

表 1. 左表展现不同图片大小与特征数量下，二者算法消耗的时间对比。我们的算法似乎倾向于选择更少的特征。

Image size	cv2 time	ours time	cv2 F	ours F
(150, 150)	0.04	17.34	106	106
(300, 400)	0.14	174.28	1257	1254
(640, 490)	0.36	93.77	422	414



的栅栏，上半图中的特征点似乎与下半图的两个特征点都相似，但真正与其配对（即同属重合区域）的特征点只有一个。若配对了错误的特征点，则下面拟合变换时会出现灾难性的后果。因此我们需要对特征点匹配的结果再做一次测试，扔掉模糊匹配的特征对，保证所有被选为配对的特征点是显著匹配的。

Lowe 的论文给出了一种解决方案：对每一个左图特征点 A ，同时保存最相似与次相似的右图特征 B_1, B_2 ，然后分别计算 A 与 B_1, B_2 的距离 d_1, d_2 ，若二者的比值小于一个预先设定的阈值，即：

$$\frac{\min \text{distance}}{\text{second min distance}} \leq \text{threshold}$$

则认为 A 与 B_1 的配对具有足够的依据，将 A 与 B_1 视为配对的特征点；否则， A 与 B_1, B_2 相似程度相近，不能认为这两个配对有显著的依据，不为 A 配对特征点。

这样处理后，真正完成配对的特征点对数量会小于检测出的特征点数量，以此保证配对的可靠程度。

5.3 拟合变换

如第二节中所描述的，这里我们选择拟合透视变换（也成为单应性变换）而非仿射变换，或 FFD 等其他非线性变换，因为全景图像从左到右的视角变化带来的图像变换，使用透视变换来描述再好不过。不同于仿射变换只在 \mathbb{R}^2 上操作，透视变换需要将在 \mathbb{R}^2 中点映射到 \mathbb{R}^3 再映射回 \mathbb{R}^2 。虽说透视变换是非线性的，但仍可用一个 $3 * 3$ 的矩阵加以描述：

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

因此，我们现有原始点坐标（右图特征点）与最终点坐标（左图特征点），类似最小二乘矩阵的求解过程，可以使用重新整理方程后奇异值分解的方法 (2) 直接拟合最佳的变换矩阵。然而，在我们实际实验过程中，直接使用上步中得到的所有配对特征点拟合会带来非常糟糕的结果！这是因为在匹配特征点的过程中，我们强制要求左图的每一个特征点都对应一个右

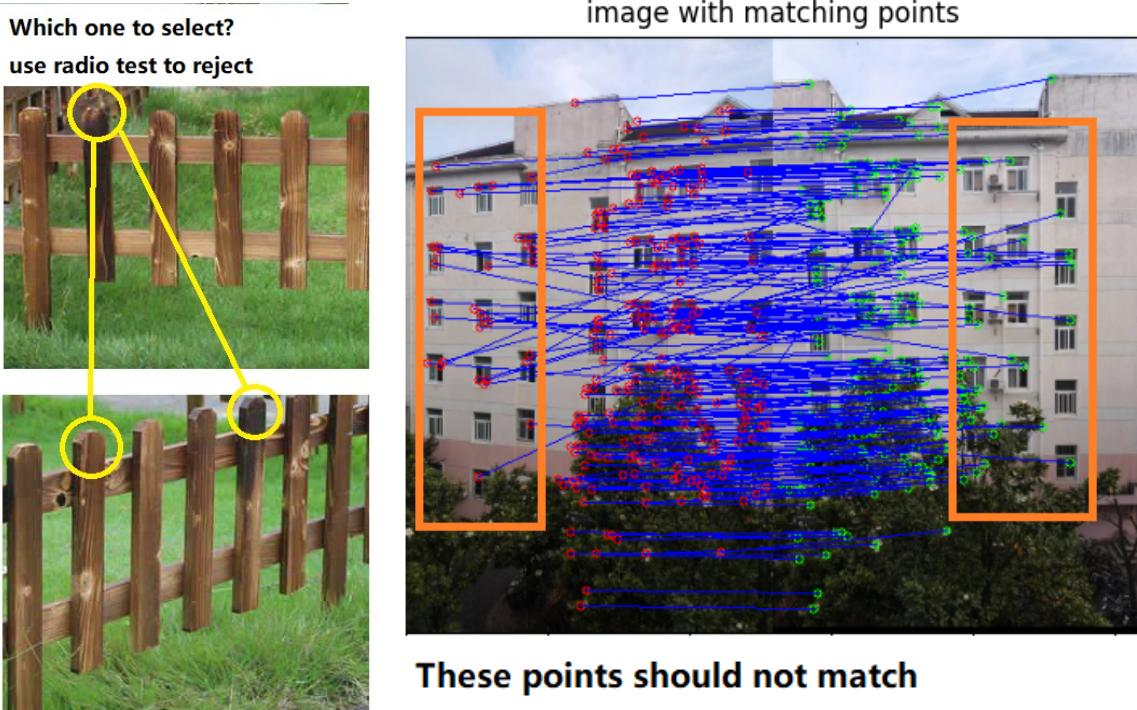


图 5. 特征点匹配中的潜在问题。左图需要去掉相似特征的情况：这些 outliers 会对实验结果造成较大影响。右图展示了去掉相似特征后，仍然出现的 outliers。这需要我们运用对于 outliers 更加鲁棒的算法进行回归。

图的特征点，然而很可能完全不存在对应点（即左右图的特征点存在至少一个不在重合区域内，比如图 5 中右图橙色框内的特征点）。这样的对应会带来 outlier 的影响。与最小二乘回归的弊病类似，极少数的 outliers 却能对整个拟合结果产生很大误差！

为了解决 outlier 带来的问题，我们采用随机抽样一致 (RANdom SAmple Consensus, RANSAC) 算法 (1)。该算法旨在每次只使用一部分点进行回归，计算出多个拟合结果，并将其中拟合程度最好的结果作为最终结果。详细流程为：

1. **取样** 随机抽取一定数量的样本特征点对
2. **拟合** 利用抽取的样本点对拟合透视变换的矩阵
3. **计算误差** 将拟合的变换作用到不参与拟合的特征点对上，计算拟合值（右图特征点变换后位置）与真实值（左图特征点位置）之间的距离，若距离小于给定的阈值，则将该对特征点称为 inlier
4. **重复取优** 多次重复以上三步，将 inlier 累计数量最多的拟合矩阵作为最终结果

该算法有四个超参数：随机取样次数、每次取样选点数、距离度量方式、距离阈值。一般来说，为了得到良好的拟合结果，特征点对数量越多，随机取样次数就应越多。由于透视变换有 8 个未知参数（9 个元素的矩阵，但最后结果需除以 z 坐标，因此损失一个自由度），因此

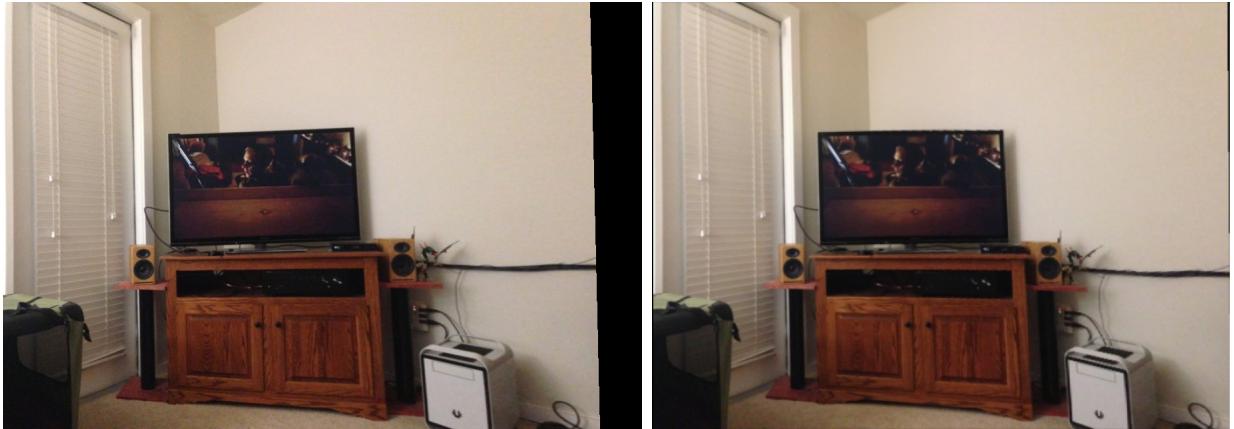


图 6. 左图为边缘存在黑边的图片，右边是去除黑边的结果

至少需要 4 对特征点才能求出。我们后续对取样选点数、距离度量方式这两个超参数做了对比实验。

5.4 变换并拼接

5.4.1 反向变换

上一步中我们求出了右图到左图的透视变换矩阵，因此我们可以将左图扩大一倍后，将右图的每个点映射到左图上。但是正向变换会带来空洞、灰度值重叠等问题，因此这里我们使用反向变换：

1. **取逆** 求出上述变换的逆变换（逆矩阵），即左图坐标到右图坐标的变换
2. **反向变换** 遍历左图坐标，求出其在右图的坐标
3. **采样** 如果上一步求出的坐标在右图范围内，则使用该坐标在右图中进行采样。采样的方法有最邻近方法、向下采样方法、双线性插值方法等，我们在此选择在右图中进行双线性插值

具体地，我们首先创建一张足够大的空图片，将左半部分直接用左图填充，随后再行遍历大图片的所有坐标，若其在右图的坐标在右图范围内（即不超出右图边界），则进行双线性插值，否则不对该点操作。

5.4.2 去除黑边

完成所有坐标点的映射后，图像边缘可能会出现一小段空白（黑边），如图 5 所示。这是由于创建了一张过大的图片，而拼接后两图像部分重合，导致整体尺寸缩小的缘故，因此我们需要去除黑边。与数组删除重复元素类似，反向遍历所有行、所有列，找到拼接图像的右边界和下边界，去除全 0 行、列即可。

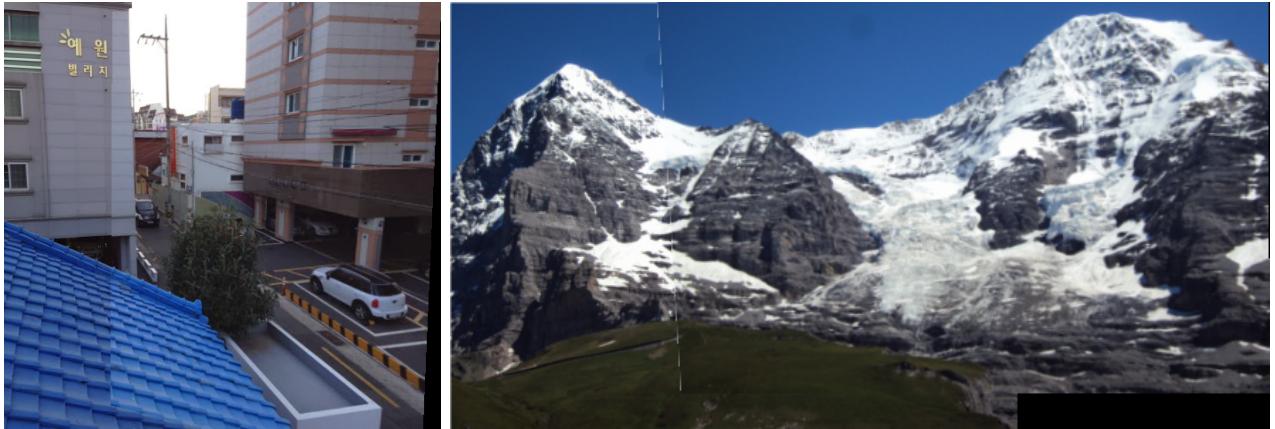


图 7. 拼接处存在色差和白边的图片

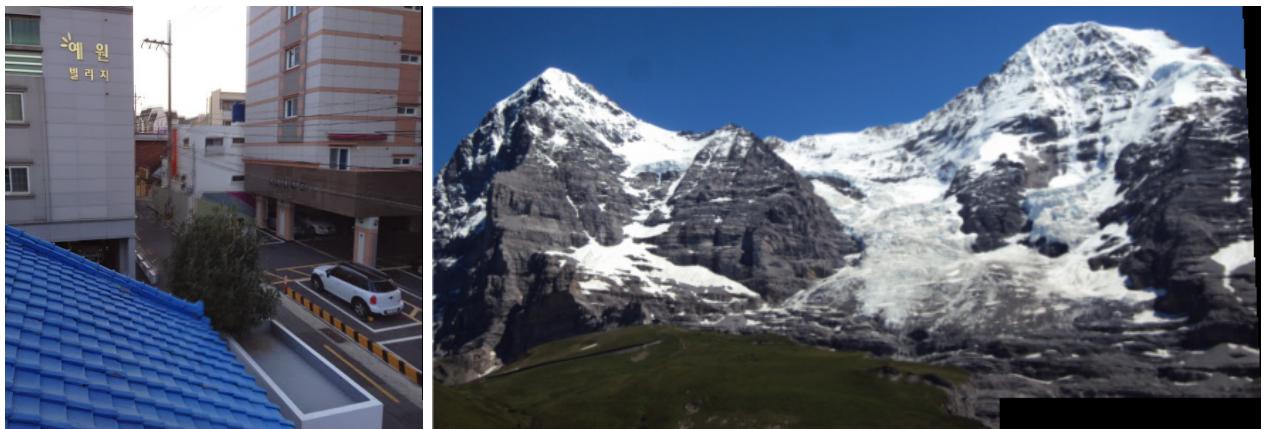


图 8. 最终结果

5.4.3 混合

现在我们可以得到很不错的拼接结果，如图 6 所示。但仔细观察拼接交界处，能看到有明显的色差，甚至出现白边。这是由于右图在拍摄时，光影分布与左图略有不同的问题，要改善这个缺点，可以对重合区域进行混合（blend）操作。

这里我们采用 Linear Blending 方法。我们把两张图重叠区域的每个像素点线性地赋予左右两个权重，权重范围为 0 到 1，越靠近左图的，左权重越大，越靠近右图则右权重越大，同时保持左右两权重的和为 1. 随后我们检测图像重合区域，对其中每个像素点的灰度值进行线性加权。注意到上面赋予的两个权重和为 1，因此实际上我们对重合区域像素点的灰度值进行凸组合，来源分别是左图重合部分和右图重合部分。

混合后，色差和白边的问题得到解决，最终结果如图 7 所示。

5.5 Experiments

对于图片拼接顺序的敏感性 此方法对于图片的顺序有很大依赖。具体而言，我们先验地假设，左边输入图片应该放左边，右边输入图片应该放右边。这样对于有明显角度偏差的图片

表 2. 特征数量和特征匹配数量的增多，算法总时间开销随之提升。

Image size	Features	Paired Features	Fitted Features	Time used
(640, 480)	422	122	85	50
(300, 400)	1257	543	530	131.20
(416, 260)	839	281	218	70

而言，其结果会产生较大变化。如图 8 的两幅建筑物，拍摄的角度有较大差别。将拼接顺序调换可能无法产生我们需要的效果。

现在一些方法可以解决该问题。比如把各个图片的特征匹配情况量化为联通分量，建立图模型（这个图模型应当是双向的，因为拼接位置影响很大）。由于时间关系，未能深入研究。后续工作可以继续在这方面研究。

特征匹配时间的开销 3.2 节特征匹配过程中，我们计算使用双重循环暴力地计算每一个左图特征点与每一个右图特征点间的距离，以此求得最小距离的特征点对，所需时间复杂度是 $O(n^2)$ ，开销较大，实际 200 左右的特征数量可以较快匹配，当待匹配的特征数量增加时，需要花更长的时间进行特征匹配。表 2 展示了不同图像尺寸、不同特征点数量的示例匹配特征点的总用时。特征匹配任务实质上是求解 K-近邻（在我们的任务中 K=2）的任务，因此我们可以使用 kd 树（K-dimensional Tree）数据结构对高维空间进行划分，从而将 K-近邻任务的时间复杂度降到 $O(n \log n)$ 。

投影矩阵的度量 为了进一步比较各种方法的优劣（同时为下文的参数实验提供可靠比较准则），这里设计了一种针对单应性矩阵准确性的实验准则。

1. **构造单应性矩阵** 首先，从同一幅图片里面截取两部分图片，这样可以保证这两部分图片初始投影角度相同。随后，手动生成一个投影矩阵，对于其中需要拼接到右侧的图片进行投影变换。
2. **特征匹配** 将左图和投影变换后的右图输入需要测试的图像拼接程序，输出变换后的图像。同时，需要输出计算得到的相应的变换矩阵。在上一步中我们生成的投影矩阵是得到当前右图的投影矩阵，因此我们需要对这一步得到的投影矩阵进行逆运算才能与上一步的标准投影矩阵进行比较。
3. **差异度量** 在第一步得到的投影矩阵相当于“正确答案”。我们需要比较我们运算得到的投影矩阵和真实的投影矩阵之间的差距。这里我们可以使用矩阵范数的运算来度量二者差距，度量投影矩阵之间的差别。



图 9. 调换顺序后，左右图片显示出的不同特征。Right order 把右图拼接到左图右边。而 Wrong order 把左图拼接到右图右边。虽然某种意义上说，两个图片都完成了图片拼接任务，但 Right order 才是我们需要的

改变 RANSAC 参数 RANSAC 主要为了处理异常值点 (outlier) 的影响。然而，我们知道 L_1 范数较不易受异常值的影响。我们设计了以下的参数实验，对于回归效果进行分析。

实验中先对于输入的图像进行投影变换（使用库函数），得到先验正确的投影矩阵。然后使用我们的匹配算法，对于图像进行匹配。得到逆变换后的矩阵。最后，运用上一段种对投影矩阵度量的方式，进行比较。

明显的， L_1 范数实验效果在大多数情况下，更加接近于真实变化。然而，其匹配的点的数量较 L_2 范数匹配的点较少。这说明 L_1 范数有一定的去除极值点的能力。

并且我们也可以发现，匹配算法的波动性较大。在 24 点以下的波动情况，我们认为不可以说明取点数量与拟合结果之间的联系。因为波动可能是由 RANSAC 算法的随机性导致的。不过，这点也说明了，我们处理过后的图片，其异常值点较少，所以取更多的点对于拟合结果影响不是太大。

同时我们也发现，随着每次用于拟合的点数量增多，inlier 数量变得越来越少。然而拟合差距并没有多大差别。这点可能说明 inlier 点数是一种对于拟合情况更加敏感的指标。

然而，当取点数量接近于特征匹配数量时，损失会突然加大。从图片上我们也可以看出，拟合效果非常差。原因在于异常值点使得拟合结果严重偏离非异常点的最优值。

以上实验也给了我们进一步的思考：使用 RANSAC 选择最优的变换矩阵或许仅仅使用 inlier 数量一个指标是不够的。我们或许需要引入一些其他指标进行加权考虑或者在相同 inlier 数量时做出抉择。原因在于实验中发现，对于 4 个采样点的 L_2 norm 和 L_1 norm 度量方式，其最大 inlier 数量均为 67 个，但最终的 difference 却有差异。并且，投影矩阵细微的变化会导致拟合效果天差地别。后续工作可以在 RANSAC 算法匹配的选择上多加功夫。

自主实现 SIFT 与 OPENCV 对比 上文已经对于时间开销进行过对比。在这里的实验中，对于实际变换矩阵进行分析。在实验中，同样的对于同一张图片进行投影变换，看投影矩阵与



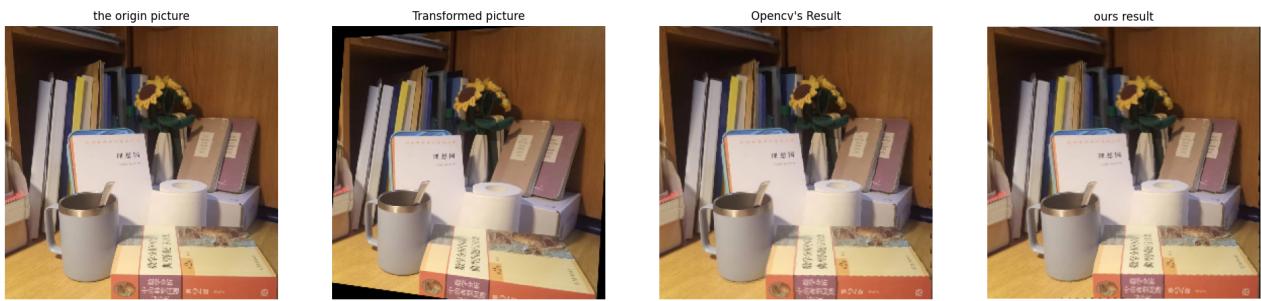
图 10. 不同度量范数和不同拟合点数情况下的实验效果。点数取得极大时，拟合效果会突然变差

表 3. 不同度量范数和不同拟合点数情况下，匹配算法投影矩阵和真实投影矩阵之间的差距表

matching points:		84	4	8	16	24	36	64
L2 norm	Fitted	67	63	55	47	35	2	
	diff	52.82	55.05	54.01	48.11	47.379	23531.33	
L1 norm	Fitted	66	62	55	47	34	2	
	diff	46.92	44.78	49.35	51.08	46.49	44444.993	
$L_{\infty} norm$	Fitted	67	63	55	47	37	2	
	diff	49.44	52.50	50.44	61.41	65.67	25037.09	

表 4. CV2 的投影矩阵和我的投影矩阵范数相差 3.14, 以下是与单应性矩阵对比

	投影矩阵	与单应性矩阵差
Opencv	$\begin{bmatrix} 7.56e - 01 & -6.38e - 02 & 1.34e + 01 \\ -8.02e - 02 & 8.49e - 01 & 1.55e + 01 \\ -7.87e - 04 & -2.48e - 05 & 9.84e - 01 \end{bmatrix}$	51.25
Ours	$\begin{bmatrix} 7.62e - 01 & -7.29e - 02 & 1.55e + 01 \\ -6.42e - 02 & 8.54e - 01 & 1.31e + 01 \\ -7.05e - 04 & -6.61e - 05 & 9.83e - 01 \end{bmatrix}$	51.41
真实矩阵	$\begin{bmatrix} 7.92e - 01 & -6.46e - 02 & 5.00e + 01 \\ -6.14e - 02 & 8.78e - 01 & 5.00e + 01 \\ -1.71e - 04 & -1.13e - 05 & 1.00e + 00 \end{bmatrix}$	

**图 11.** OPENCV 结果和我的结果的最终对比。发现区别不大

实际矩阵之间的范数差距。实验中使用的图像如下。对应计算出的投影矩阵也罗列于表格中。考虑到 RANSAC 算法本身的随机性，再加上二者数据都没有表现出显著的差异，可以认为二者方法是等价的。

6 GUI 界面使用手册

为了展示我们的算法效果，同时方便用户简单且直接地使用，我们使用 tkinter 库设计了一款用户友好性的 GUI。

6.1 环境配置

本次项目开发环境为 Python 3.8。由于程序需要，可能需要安装以下安装包：

```
# 安装 cv2 库
pip install opencv-python
# 安装 PIL 库
```

```
pip install pillow
# 安装numpy包
pip install numpy
```

以上命令可在终端命令行完成。

6.2 使用教程

将我们提交的代码文件夹放到 Python 项目路径下，找到 code 文件夹中的 visual.py 文件，运行之（可在控制台输入‘python visual.py’命令），会出现以下初始界面。只需简单四步，即可实现图像的缝合与结果展示。

6.2.1 选择图像混合模式

首先我们需要在以下位置选择图像的混合方式，

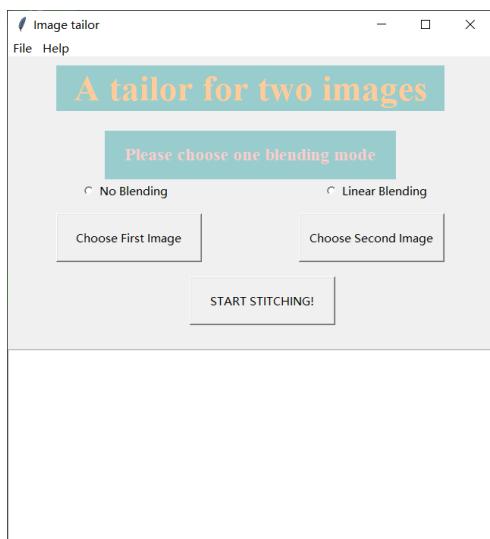


图 12. GUI 初始化界面



图 13. 图像混合方式选择区域

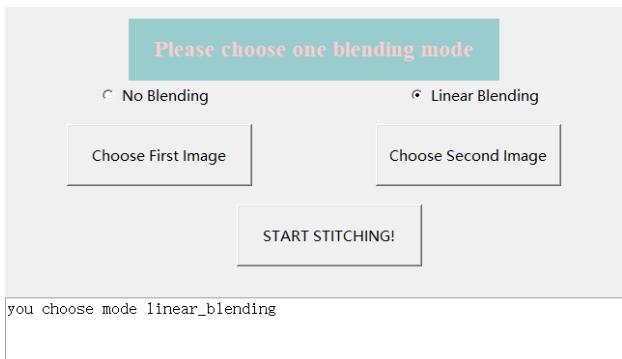


图 14. 图像拟合方式选择后的对话框提示

我们提供了两种混合方式：无拟合（即直接拼接）和线性混合。两种方法的优劣与效果对比已经在上面展示过，可以根据喜好选择。只需点击对应方式前面的小圆圈即可，选择后下面的对话框中也会出现选择的混合方式：

当然，如果不小心点错或者在缝合图像之前临时想要改变图像混合方式，也是完全没问题的，只需要再次点击对应单选框即可，下方的对话框也会再次提示新选择的拟合方式。只需确保在进入第三步之前，最后一次对话框提示的混合方式是你想要的即可。

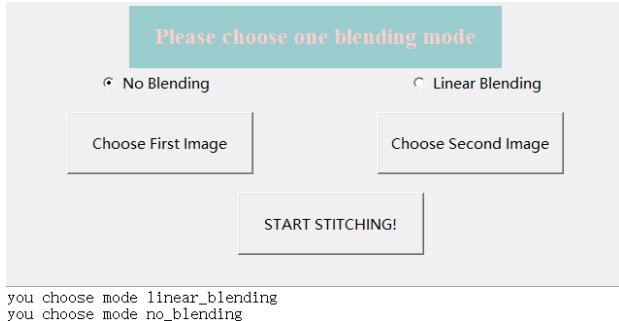


图 15. 重新选择图像混合方式

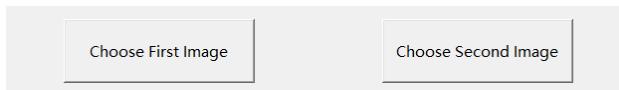


图 16. 导入待缝合图片按钮区域



图 17. 选择图像文件后的对话框提示

Step1 - Use SIFT algorithm extract the key points and features...
Step2 - Match the key points of two images...
Step3 - Find the best homography matrix with RANSAC algorithm...
The number of inliers of the best homograph matrix: 85
Step4 - Warp and stitch images...

图 18. 程序运行过程中的对话框提示

6.2.2 选择待拼接的图像

在这一步，你需要选择要拼接的图像。只需点击以下两个按钮，在弹出的文件选择框选择对应的**图像文件**即可。

同样地，在选择完图像文件后，对话框也会提示你选择的文件地址，可以进行核对。

点击 **Choose first image** 和 **Choose second image** 的顺序并不重要。但重要的是 **Choose first image** 一定要选择缝合图像的左侧图像，**Choose second image** 一定要选择缝合图像的右侧图像。和图像拟合方式一样，图像文件也可以多次重新选择，请在进入第三步之前确认好选择的图像文件地址。

6.2.3 开始拼接图像

在确认完对话框中对图像拟合方式和图像文件地址的提示后，就可以开始缝合图像。请点击 **START STITCHING!** 按钮，程序就可以开始运行。算法的运行需要一定时间，请耐心等待。

在运行过程中，界面的对话框会输出以下信息提示你目前算法处理到哪一步，以及算法一些输出的中间信息。

运行结束后，程序会弹出一个新的界面，对读取图像、图像中间处理结果以及最终结果进行展示。

第一张图像是读取的图像原文件，也即待缝合的两张图片。

点击下方的 **Next image** 可以切换到下一张图片。

下一张展示的图像是两个图片加上对应特征点对的可视化。圆圈代表的是使用 SIFT 算法提取的图像上的特征点位置，连线代表是两张图特征点的对应结果。

最后一张图是最终缝合的图像结果。

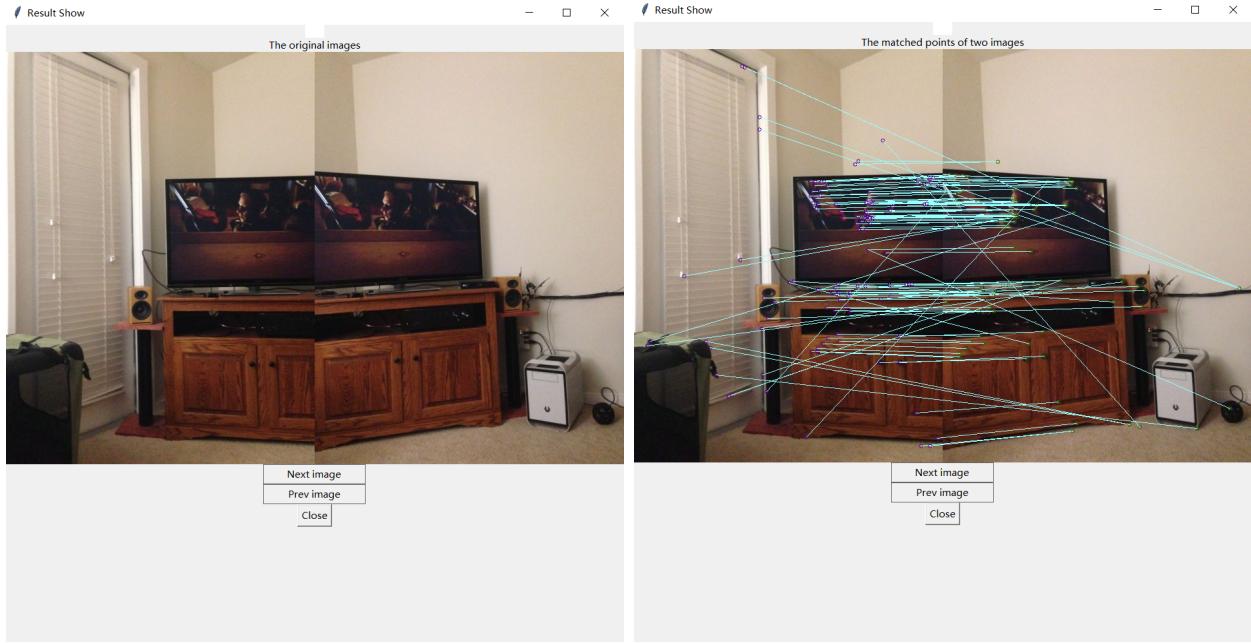


图 19. 运行结果第一张图——读取的原图片 **图 20.** 运行结果第二张图——原图片加上对应的特征点

随意点击 Next image 和 Pre image 可以切换三张图片。

6.2.4 结果图像的保存

退出结果图像展示的页面后，回到可视化主页面。如果对上一步骤展示的结果图像满意的话，可以选择以 PNG 格式保存该结果。点击主页面左上角 File->Save，就会弹出文件选择框，选择目的文件夹，并在文件名处填写需要保存为的文件名，点击保存即可。

同样地，保存后页面的对话框会弹出保存地址的提示，可以再次确认是否地址正确。

6.2.5 其他

需要退出 GUI 页面程序，只需点击主页面左上角 File->Exit，或者直接点击右上角的叉退出即可。程序使用过程中，有任何疑问可点击左上角 Help 寻求帮助。也可以使用右键快速召唤出菜单。

6.3 常见问题与解答

在 GUI 使用过程中，可能会遇到以下常见问题：

1. 图像拼接的结果不理想/很奇怪？

请检查是否读取图片的左右顺序弄反了，或者拼接的方向不符合要求。我们的算法目前仅针对水平方向的缝合，如果遇到竖直方向的缝合或者投影角度较偏的图片，不能进行较好的匹配缝合。

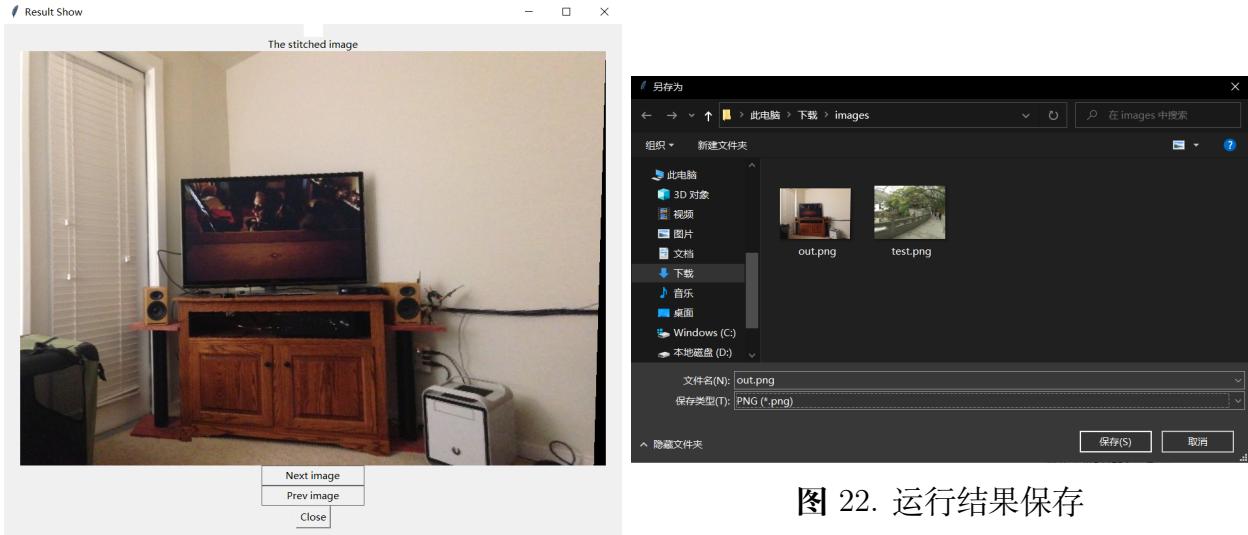


图 21. 运行结果第三张图——缝合后的图像

图 22. 运行结果保存

2. 除了图像拟合方式，是否还有其他的超参数可以选择调整？

目前只开放了图像拟合方式的选择。这是因为其他的超参数选择较为复杂，如果没有较多的实验基础很难选择到合适的超参数取值。在上文中，我们也对几乎所有超参数的取值进行了实验，并在最终将取值调整到了最佳。如果感兴趣，可以阅读仔细阅读该报告的其他部分。当然，如果非常想亲手调试超参数，感受效果，也可以打开 code 文件夹的 `stitch.py` 文件，并在其中修改大部分超参数的取值。

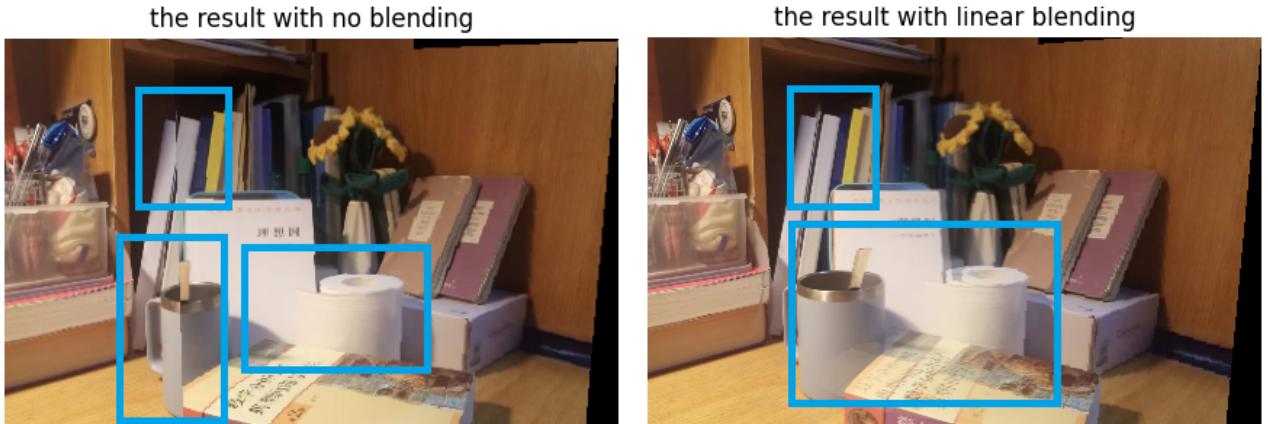
3. 程序运行卡在某一个 step 很久？

请检查输入的图像格式是否过大。因为我们在提取特征一步使用的 SIFT 算法是一个虽然效果不错，但比较耗时的算法。且后续的单应性矩阵计算也是需要一定时间的。因此可以选择裁剪输入的图像大小并耐心等待。

7 Limitations

本实验实现的算法对于拼接顺序有着严格的要求。输入的顺序即为拼接的顺序。目前该问题也有很多比较成熟的解决办法，比如可以基于图论知识，将若干幅无序图像作为图节点构成非连通图，使用一些指标量化图像之间的拼接可信度，并作为图节点之间连接权重。于是无序图像的连接顺序问题就可以转化为遍历非连通图得到若干幅连通子图的问题，可以使用 Dijkstra 等算法求解足有路径。但由于时间考虑，我们并没有来得及进一步实现。

第二个问题是目前我们的程序只适用于两张图片的拼接。现实生活场景中，往往需要拼接多张图像。当然，最简单的方法可以将多张图像两两依次拼接。但这个方法的缺点也显而易见。随着拼接的图像越来越多，每次都进行特征点的提取和匹配是一个非常低效且冗余的操作，且最后的结果会默认以第一张图片作为视觉中心，导致拼接后的图像透视角度会有所偏差。事实上我们知道，图像的拼接就像拼积木一样，每次拼接下一张图时，只需和这张图



Due to the problem of transformation, there is still ghost phenomenon

图 23. 被方框框出的地方是拼接不到位的地方。由此，两幅图片未能完全重合，导致“鬼影”

相邻接的图像进行特征点的匹配即可。因此，一个高效的方法流程如下：

1. 共有 N 张待拼接的图像 I ，目前待拼接的图像即为 I_t ，前 $t - 1$ 张图像已拼接好的结果即为 SI_{t-1} 。
2. 计算第 $t - 1$ 和第 t 张图像的单应性变换矩阵 $H_{t-1,t}$ 。记图像 I_{t-1} 和 SI_{t-2} 的单应性矩阵为 H_{t-1} ，则可以推得第 t 张图像的单应性矩阵为：

$$H_t = H_{t-1} H_{t-1,t}$$

3. 重复上面两个步骤，直到所有图像拼接完成，得到 SI_N

根据上面的算法计算出每一个图像的单应性矩阵后，视觉中心也很好调整了。选出想要的视觉中心（比如说最中间的图像），将该图像的单应性矩阵 H_{visual} 求逆后乘在所有图像的单应性矩阵上即可（等价于将该图像的投影变换矩阵修正为单位矩阵，默认其初始投影角度即为整幅拼接图像的投影角度）。

在本次项目实现中，我们将图像缝合的算法优化为线性拟合，这解决了一定的缝合边界不自然的问题，但也带来一个新的问题，拼接后的图像会出现‘鬼影’现象，如图 22 所示。这有很大的改进空间。一个比较自然的想法是，既然重叠的区域会出现鬼影，那么我们只需要尽可能缩小重叠区域，那么鬼影就会不那么明显。这实际上是直接拼接和线性拟合两种方法的结合，通过在拼接色调不自然和拟合区域有偏差之间进行权衡，找到一个较好的平衡点。但这个方法是个双刃剑，什么问题都缓解也等价于什么问题都没解决。在查阅论文后，我们发现了一种可能更加有效的方法，很多论文都使用到了结合最佳缝合线和多分辨率拟合的改进方法，可以有效消除接缝和鬼影。由于时间因素，我们未能自己实现相关算法。

此外，由于我们拼接的图像是实体景物在不同坐标系下的二维投影，直接对拍摄图像进行拼接其实是无法满足视觉一致性的。所以可以通过将待拼接的图像分别投影到一个标准的

坐标系下，然后再进行图像的拼接。比如可以采用圆柱体、正方体、球体等模型来进行投影。其中由于柱面坐标的变换比较简单并且投影图像与其投影到圆柱表面的位置无关，用其描述的柱面全景图像可在水平方向上满足 360 度环视，具有较好的视觉效果，因此被广泛采用。

除了以上提出的问题展望，我们的算法实际还有很多待改进的问题。比如 SIFT 没有旋转不变性，这意味着当图像出现大的旋转角度时拼接会失败；SIFT 算法和 RANSAC 算法的实现的时间复杂度太高；没有办法识别噪声图像；假如一张和其他待拼接图像无关的图像，算法并不能识别并丢弃掉该图像，会导致拼接结果受到该图像的很大影响等。

参考文献

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

Kriegman, D. (2007). Homography estimation. Lecture computer vision I, CSE a, 252.

Lowe, D. G. Distinctive image features from scale-invariant keypoints. 60(2):91–110.

A 代码结构

本次提交的项目代码主要分为三个文件夹：code、SIFT、images。

code 文件夹主要包含主程序代码、GUI 框架代码以及所有测试实验代码。其中主程序代码文件包括 stitch.py 和 utils.py 两个文件。其中 stitch 文件为算法实现的四步骤，utils 则包含了一些使用到的工具函数。visual.py 为 GUI 可视化程序的代码文件。其余文件均为实验测试。其中 test_homography.py 文件为单应性测试的主文件。

SIFT 文件夹主要包含我们自己实现的 SIFT 算法。sift_detect.py 文件为 SIFT 算法实现的主文件，visualize.py 则为中间结果可视化的文件。

images 文件夹则包含了我们项目过程中使用的所有实验图片，以及输出结果。当然，您也可以选择自己的图片来测试我们的算法。