



全景图拼接

24Spring 模式识别课程实验一

姓名 李博

学号 21307278

学院 计算机学院

专业 计算机科学与技术

目录

1 实验目的	2
2 实验要求	2
3 实验内容	2
4 算法原理	3
4.1 Harris 角点检测	3
4.1.1 算法流程简述	3
4.2 关键点描述与匹配	4
4.2.1 SIFT 特征	4
4.2.2 HOG 特征	4
4.3 RANSAC 匹配特征点	5
4.3.1 算法流程简述	5
5 实验过程记录	5
5.1 实现 Harris 角点检测	5
5.2 实现关键点描述与匹配	6
5.2.1 SIFT 特征	7
5.2.2 HOG 特征	9
5.3 实现图像拼接	10
5.4 多图像拼接	12
6 实验结果展示	12

1 实验目的

- 1、熟悉 Harris 角点检测器的原理和基本使用
- 2、熟悉 RANSAC 抽样一致方法的使用场景
- 3、熟悉 HOG 描述子的基本原理

2 实验要求

- 1、提交实验报告，要求有适当步骤说明和结果分析、对比
- 2、将代码和结果打包提交
- 3、实验可以使用现有的特征描述子实现

3 实验内容

1. 使用 Harris 角点检测器寻找关键点。
2. 构建描述算子来描述图中的每个关键点，比较两幅图像的两组描述子，并进行匹配。
3. 根据一组匹配关键点，使用 RANSAC 进行仿射变换矩阵的计算。
4. 将第二幅图变换过来并覆盖在第一幅图上，拼接形成一个全景图像。

5. 实现不同的描述子，并得到不同的拼接结果。

4 算法原理

4.1 Harris 角点检测

通常意义上来说，角点就是极值点，即在某方面属性特别突出的点，是在某些属性上强度最大或者最小的孤立点、线段的终点。对于图像而言，角点是指在两个或多个方向上都有较大灰度变化的像素点。相比于边缘和平坦区域，角点具有更强的特殊性和唯一性，因此在图像特征提取、目标识别、图像匹配等任务中起着重要的作用。

角点检测算法基本思想是使用一个固定窗口（取某个像素的一个邻域窗口）在图像上进行任意方向上的滑动，比较滑动前与滑动后两种情况，窗口中的像素灰度变化程度，如果存在任意方向上的滑动，都有着较大灰度变化，那么我们可以认为该窗口中存在角点。

4.1.1 算法流程简述

1. 定义角点响应函数：

设图像上某点的坐标为 (x, y) ，灰度函数为 $I(x, y)$ 。考虑在该点附近移动一个小的位移向量 (u, v) ，灰度变化可以表示为：

$$E(u, v) = \sum_{x', y'} w(x', y') [I(x' + u, y' + v) - I(x', y')]^2$$

其中， $w(x', y')$ 是一个窗口函数，通常为高斯窗口，用于权衡不同位置的灰度变化。如果窗口中心点是角点时，移动前与移动后，该点在灰度变化贡献最大；而离窗口中心（角点）较远的点，这些点的灰度变化几乎平缓。

2. 计算自相关矩阵：

对于窗口内的灰度变化，我们可以将 $E(u, v)$ 在 $(u, v) = (0, 0)$ 处进行泰勒展开：

$$E(u, v) \approx E(0, 0) + [u, v] M [u, v]^T$$

其中， M 为自相关矩阵：

$$M = \begin{bmatrix} \sum w(x', y') I_x^2 & \sum w(x', y') I_x I_y \\ \sum w(x', y') I_x I_y & \sum w(x', y') I_y^2 \end{bmatrix}$$

其中， I_x 和 I_y 分别为图像在 x 和 y 方向的梯度。

3. 定义角点响应函数：

定义角点响应函数 R 为

$$R = \det(M) - k \cdot \text{trace}^2(M)$$

其中 k 是一个指定的经验参数，需要实验确定它的合适大小，通常它的值在 0.04 和 0.06 之间，它的存在只是调节函数的形状而已。

4. 非极大值抑制:

对角点响应函数进行非极大值抑制，保留局部最大值点作为最终的角点。

4.2 关键点描述与匹配

4.2.1 SIFT 特征

SIFT (Scale-Invariant Feature Transform) 是一种用于图像处理领域的特征提取算法，最初由 David Lowe 在 1999 年提出。SIFT 算法主要用于检测图像中的关键点，并提取这些关键点的特征描述子，以便于后续的图像匹配、目标识别等任务。

SIFT 算法流程:

1. 尺度空间极值检测：对图像在不同尺度下进行高斯模糊处理，使用 DoG 算子检测图像的极值点。

2. 关键点定位：对检测到的极值点进行精确定位，通常使用拟合二维高斯函数来确定关键点的位置和尺度。假设高斯函数为

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

则图像的高斯模糊可以表示为

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

其中， $I(x, y)$ 为原始图像，在不同尺度下进行高斯模糊得到的图像记为 $L(x, y, \sigma)$ 。

3. 关键点方向赋值：为每个关键点分配一个主方向，通过计算关键点周围区域的梯度方向直方图来确定。梯度方向直方图可表示为

$$\theta(x, y) = \arctan2 \left(\sum_p I_x(p) w_\sigma(p), \sum_p I_y(p) w_\sigma(p) \right)$$

其中， I_x 和 I_y 分别为图像在 x 和 y 方向上的梯度， p 表示周围像素， $w_\sigma(p)$ 为高斯加权。确定主方向后，关键点的描述子将相对于该方向进行旋转，以增强描述子的不变性。

4. 关键点描述：在关键点周围的邻域内计算特征向量，构成关键点的描述子。描述子的构建基于关键点周围的图像梯度直方图，通过将邻域分割为子区域，并在每个子区域内计算梯度方向直方图，最后组合成一个向量。

5. 特征匹配：使用描述子对图像中的关键点进行匹配，通常使用欧氏距离等方法来计算相似性。

6. 匹配筛选：对匹配结果进行筛选和优化，通常采用 Lowe 的比率测试等方法来剔除不稳定的匹配。

4.2.2 HOG 特征

HOG (Histogram of Oriented Gradients) 特征是一种用于图像识别和目标检测的特征描述方法，它将图像的局部梯度方向信息编码成直方图特征。HOG 特征广泛应用于行人检测、物体识别等领域，其主要优点是对光照、阴影等影响较小，并且具有一定的旋转不变性。

HOG 算法流程:

1. 图像预处理：包括将彩色图像转换为灰度图像，并对像素值进行归一化处理；Gamma 校正，图像整体亮度提高或降低以标准化亮度。

2. 计算图像梯度：使用 Sobel 算子计算图像在水平和垂直方向上的梯度，得到每个像素点的梯度幅值和梯度方向：

$$\text{水平方向梯度: } G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * I$$

$$\text{垂直方向梯度: } G_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * I$$

$$\text{梯度幅值: } |\nabla I| = \sqrt{G_x^2 + G_y^2}$$

$$\text{梯度方向: } \theta = \arctan2(G_y, G_x)$$

3. 计算图像的梯度直方图: 将图像划分为若干个单元格 (Cell), 对每个单元格内的像素, 根据其梯度方向将梯度幅值累加到对应的梯度方向直方图中。直方图通常分为 9 个方向区间。

4. 块归一化: 将相邻若干个单元格组成一个块 (Block), 对每个块内的梯度直方图进行归一化, 以降低光照变化等因素的影响。一般采用 L2 范数归一化, 即将每个块的特征向量除以其 L2 范数。

5. 生成最终的特征向量: 将所有块的归一化直方图串联起来, 得到最终的特征向量。特征向量的维度取决于单元格和块的数量。

4.3 RANSAC 匹配特征点

在进行图像拼接时, 首先要解决的是找到图像之间的匹配的对应点。在前面的实验中已经实现了关键点的选取和特征点匹配的工作。虽然类似于 SIFT 算法的性能比较强, 能产生很少的错误匹配, 但仍然还是存在错误的对应点。所以需要用一种算法对 SIFT 算法产生的特征描述符进行剔除误匹配点。

RANSAC(RANdom SAMple Consensus, 随机采样一致) 算法是从一组含有“外点”(outliers) 的数据中正确估计数学模型参数的迭代算法。“外点”一般指的数据中的噪声, 比如说匹配中的误匹配和估计曲线中的离群点。RANSAC 算法作为一种经典的消除误匹配的方法, 具有匹配精度高、可靠度强等优点。

4.3.1 算法流程简述

如下的算法结合了通用的 RANSAC 流程以及本实验中的特征点匹配流程进行阐述

1. 随机采样 K 个点, K 是求解模型参数的最少点个数; 本实验中, 从两幅图像的特征点集合中随机选择一定数量的点, 例如选择四对点用于计算仿射变换矩阵。

2. 使用采样的 K 个点估计模型参数; 根据随机选择的点, 计算出一个初步的匹配模型。在特征点匹配中, 这个模型可以是一个仿射变换矩阵或者其他几何变换模型。

3. 计算剩余点到估计模型的距离, 距离小于阈值则为内点, 统计内点的数目; 对于剩余的特征点, 根据当前估计的模型, 计算其到模型的距离。如果距离小于匹配阈值, 则将其标记为内点。

4. 重复以上步骤多次, 选择具有最大内点数量的模型作为最终匹配模型。计算内点数量, 作为当前模型的评价指标。

5 实验过程记录

5.1 实现 Harris 角点检测

1. 由于 Harris 角点检测算法只能处理灰度图像, 因此需要将图像转换为灰度图像。

```
1 img = cv2.imread(image_path)
2 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

2. 使用 Sobel 算子计算图像的梯度，得到图像在 x 和 y 方向上的梯度。

```
dx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
dy = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
```

3. 计算自相关矩阵 M 并计算角点响应函数 R。

```
1   Ixx = dx**2
2   Ixy = dy*dx
3   Iyy = dy**2
4   k = 0.04
5   M = cv2.GaussianBlur(Ixx, (5, 5), 0) * cv2.GaussianBlur(Iyy, (5, 5), 0) -
6           cv2.GaussianBlur(Ixy, (5, 5), 0)**2
7   R = M - k * (Ixx + Iyy)**2
```

4. 非极大值抑制，设置阈值为最大响应值的 2% 并在原图中绘制出角点进行可视化。

```
1   R_max = np.max(R)
2   threshold = 0.02 * R_max
3   corner_img = img.copy()
4   for i in range(img.shape[0]):
5       for j in range(img.shape[1]):
6           if R[i, j] > threshold:
7               cv2.circle(corner_img, (j, i), 1, (255, 0, 255), -1)
```

上述流程手动实现的 Harris 角点检测结果如图 14 所示，图 15 是使用 openCV 库的结果，可以看出结果相对接近，说明手动实现的角点（关键点）检测的效果相对良好。

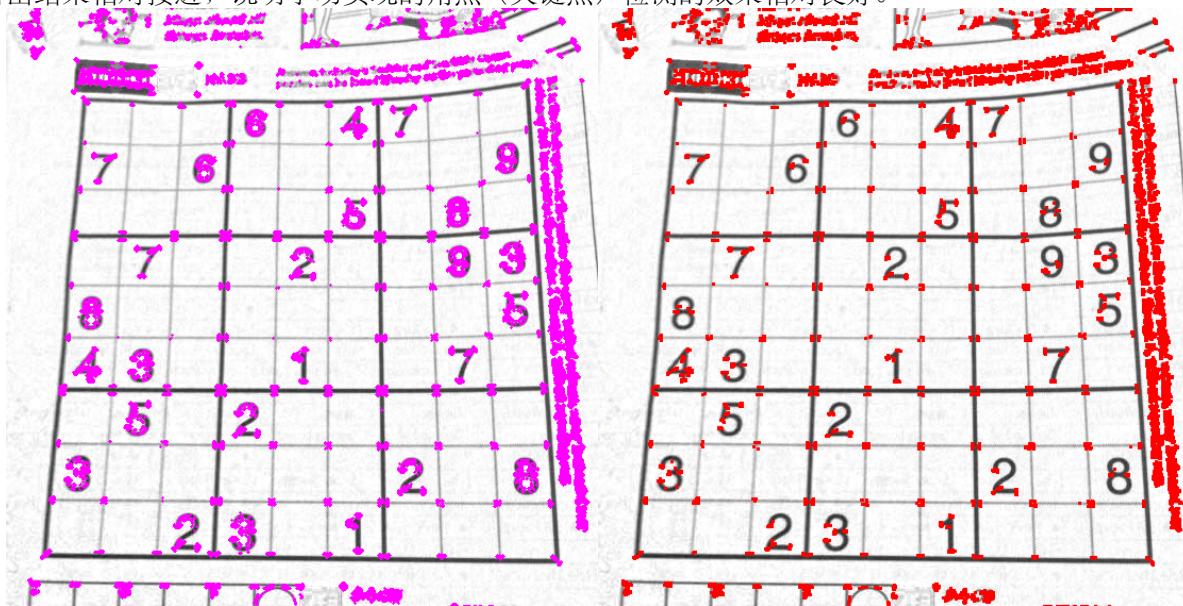


图 1: 手动实现的角点检测结果

图 2: 调库的角点检测结果

5.2 实现关键点描述与匹配

以前述的 Harris 角点检测作为关键点，可以使用 SIFT 和 HOG 特征进行表示和匹配。

首先按照要求，展示 uttower1 和 uttower2 的角点，如图 16 和 17 所示。



图 3: uttower1 角点

图 4: uttower2 角点

5.2.1 SIFT 特征

SIFT 特征算法的原理已经在 4.2.1 一节详细阐述，对 SIFT 特征的计算可以通过 opencv 库的指定方法简单实现。使用 Harris 角点坐标创建 cv2.KeyPoint 对象，把关键点和灰度图传入 sift.compute 方法可以实现 SIFT 描述子的计算。

在邻域大小为 2、Sobel 算子孔径大小为 3、k 为 0.04 的角点参数设置下，以 0.01 倍最大响应值为阈值，可以看出每幅图片计算了约四千到五千个角点。每个角点以半径为 5 的领域计算了 128 维的 SIFT 特征向量。

```
-----Harris corner detection-----
corner_location1_shape: (4340, 2)
corner_location2_shape: (5017, 2)
-----SIFT feature extraction-----
kp_descriptor1_shape: (4340, 128)
kp_descriptor2_shape: (5017, 128)
```

图 5: SIFT 描述子维数

主要的代码流程如下：

```
1 # 根据已经有的角点坐标进行SIFT特征提取
2 def sift_feature_extraction(image_path, corner_location):
3     img = cv2.imread(image_path)
4     gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
5     # corner_location:(N, 2)
6     sift = cv2.SIFT_create()
7     keypoints = [cv2.KeyPoint(float(x[1]), float(x[0]), 5) for x in corner_location
8                  ↪ ]
9     keypoints, descriptors = sift.compute(gray_img, keypoints)
10    # print("sift keypoints shape:", keypoints.shape)
11    # print("sift descriptors shape:", descriptors.shape)
12    return keypoints, descriptors
```

在得到每个角点（关键点）的 128 维 SIFT 特征向量后，可以进行两幅图像的特征向量匹配。流行的向量匹配器包括：Brute-Force Matcher（暴力匹配器）、FlannBasedMatcher（FLANN 匹配器）、KNNMatcher（K-最近邻匹配器）等等。

本实验中，由于数据规模比较小（只有数千个数据点），所以我选择使用了 Brute-Force 匹配器，它会暴力匹配和计算两组特征向量之间的所有对应关系，并通过距离阈值来确定最佳匹配。opencv 有 Brute-Force 匹配器的库方法，直接传入前一步计算的描述子向量即可，主要代码如下：

```

1 # 使用Brute-Force匹配器，对特征点进行匹配
2 bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
3 matches = bf.match(descriptors1, descriptors2)
4 print("bf_matcher_out_len", len(matches))

```

对于匹配结果，我手动选取和截断了匹配最优的 10/100/1000 个匹配点，并在两幅图像中进行匹配和可视化。

```

1 # 根据距离排序，只保留前n个匹配
2 n = 10/100/1000
3 matches = sorted(matches, key = lambda x:x.distance)[:n]
4 print("out_max_distance", matches[n-1].distance)

5 # 绘制匹配结果
6 img1 = cv2.imread(image1_path)
7 img2 = cv2.imread(image2_path)
8 img_matches = cv2.drawMatches(img1, keypoints1, img2, keypoints2, matches, None
9     ↪ , flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

```



图 6: 10 个最优 SIFT 描述子匹配点对



图 7: 100 个最优 SIFT 描述子匹配点对



图 8: 1000 个最优 SIFT 描述子匹配点对

5.2.2 HOG 特征

HOG 特征算法的原理已经在 4.2.2 一节详细阐述，对 HOG 特征的计算也可以通过 opencv 库的指定方法简单实现。

主要的计算依赖于 hog.compute 方法，主要的参数是传入灰度图和角点坐标，该方法会根据默认的窗口大小 winSize 和窗口步长 winStride 来计算 HOG 特征向量。默认的 winSize 为 12864，winStride 为 88，在本实验中，该默认参数的效果良好。

```

1  hog = cv2.HOGDescriptor()
2  des_hog = hog.compute(gray, locations=corner_location_tuple).reshape(
    ↪ corner_location.shape[0], -1)

```

HOG 特征提取的结果如下图所示，可以看出对于每个角点提取了 3780 个特征向量；由于 HOG 特征向量在本实验的精度不高，实验发现有许多向量为纯零向量。

```

-----Harris corner detection-----
corner_location1_shape: (4340, 2)
corner_location2_shape: (5017, 2)
-----HOG feature extraction-----
corner_location_shape: (4340, 2)
corner_location_shape: (5017, 2)
kp_descriptor1_hog_shape: (4340, 3780)
kp_descriptor2_hog_shape: (5017, 3780)

```

图 9: SIFT 描述子维数

在特征匹配时，由于纯零向量的存在，如果进行暴力一对一匹配会出现许多无效数据匹配点；所以在匹配 HOG 特征时考虑使用基于 L2 范数的 knnMatch，它在两组特征描述子之间进行 k 近邻匹配。在给定描述子集合中，对于每个描述子，它会找到另一组中 k 个最接近的描述子。我设置 k 为 2，并设置最优匹配相对于次优匹配至少具有 95% 的距离优势，才认为最优匹配是可靠的。

```

1  bf = cv2.BFMatcher(cv2.NORM_L2)
2  matches = bf.knnMatch(descriptors1, descriptors2, k=2)
# 选择最佳匹配
3  good = []
4  for m, n in matches:
5      if m.distance < 0.95 * n.distance:
6          good.append(m)

```

```
8     matches = good
```

使用如上方法可以对 HOG 特征筛选数十个匹配点对，对于匹配结果可以可视化如下。



图 10: 最优 HOG 描述子匹配点对

5.3 实现图像拼接

要实现同一场景不同视角的图像拼接，可以通过对匹配点对（即同一地点的不同视角下在不同图像上的坐标）计算透视变换矩阵实现。

从前面得到的 match 点对中复原出关键点点对的图像坐标，并依赖于这些坐标计算透视变换矩阵。透视变换矩阵描述了从源图像到目标图像的投影变换关系。通过它可以使图像进行对齐和匹配。

在求解透视变换矩阵使用了 RANSAC 方法，RANSAC 可以接受具有噪声的匹配点对，通过迭代的方法可以终选择具有最大一致性的子集产生的变换矩阵作为最终的估计。对于噪声的处理可以有效提高图像拼接的准确度和效果。

```
1 src_pts = np.float32([keypoints1[m.queryIdx].pt for m in matches]).reshape(-1,
2   ↪ 1, 2)
3 dst_pts = np.float32([keypoints2[m.trainIdx].pt for m in matches]).reshape(-1,
4   ↪ 1, 2)
# 计算透视变换矩阵
5 M, mask = cv2.findHomography(dst_pts, src_pts, cv2.RANSAC, 5.0)
```

计算平移矩阵，来确保拼接的第二幅图像位置合适；并使用 cv2.warpPerspective 方法传递矩阵参数 M 计算透视变换图像并进行拼接。

```
1 translation_matrix = np.array([[1, 0, -x_min], [0, 1, -y_min], [0, 0, 1]])
2 M = np.dot(translation_matrix, M)
3 result = cv2.warpPerspective(image2, M, (x_max - x_min, y_max - y_min))
4 result[-y_min:h1 - y_min, -x_min:w1 - x_min] = image1
```

对于 SIFT 和 HOG 两种不同的特征计算和匹配得到的效果如下图所示，其中 SIFT 为图 21，HOG 为图 23。

可以明显看出使用 SIFT 特征的效果要优于 HOG 特征，推测原因可能有以下两点：

(1) SIFT 描述子具有更多的匹配点对。本实验中，由于 SIFT 特征使用的是一对一匹配，求出并使用了上千个数据点对。但是由于 HOG 特征向量具有稀疏性（部分为全零向量），只能使用 knn 进行匹配，在 95% 的最优阈值下只能找到几十个匹配点对。毫无疑问更多的点对具有更高的精度。

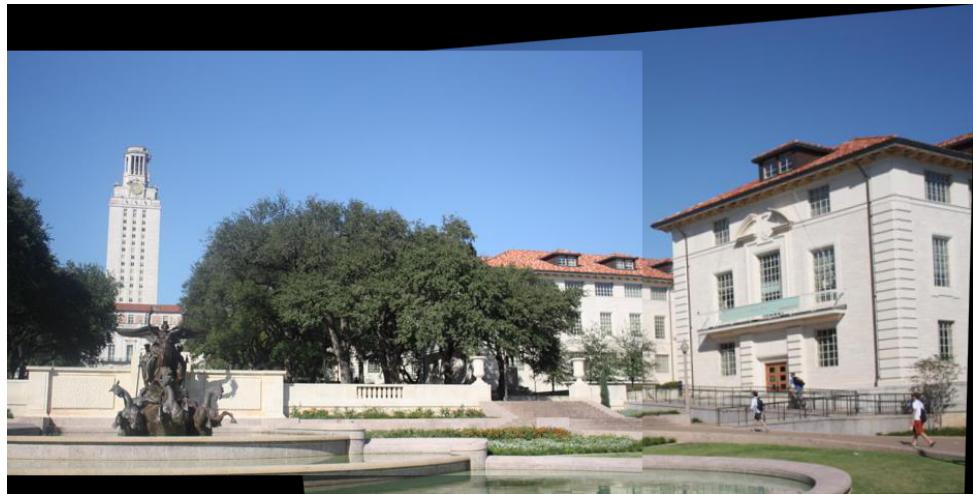


图 11: 基于 SIFT 描述子的图像拼接效果



图 12: 基于 HOG 描述子的图像拼接效果

(2) SIFT 描述子具有很强的几何不变性和光照不变性，能够更好地描述图像的局部特征，对图像旋转、缩放和视角变化具有较好的鲁棒性，因此在匹配过程中更容易找到正确的对应关系；HOG 描述子受到亮度影响较大，并且在本实验中没有进行 Gamma 纠正或者直方图均衡化等预处理操作。

5.4 多图像拼接

对于四幅图像的拼接可以采用递归的方式，即使用图片 $n-1$ 和图片 n 的拼接结果和图片 $n+1$ 进行操作。考虑到图片顺序的问题，可以使用图片之间的 match 点对的数量来判断拼接两张图片是否合理，即在匹配点对数量较少时考虑到图片可能不直接相连来避免此拼接顺序。

下面显示了实验要求中的四幅图像递归拼接的效果：

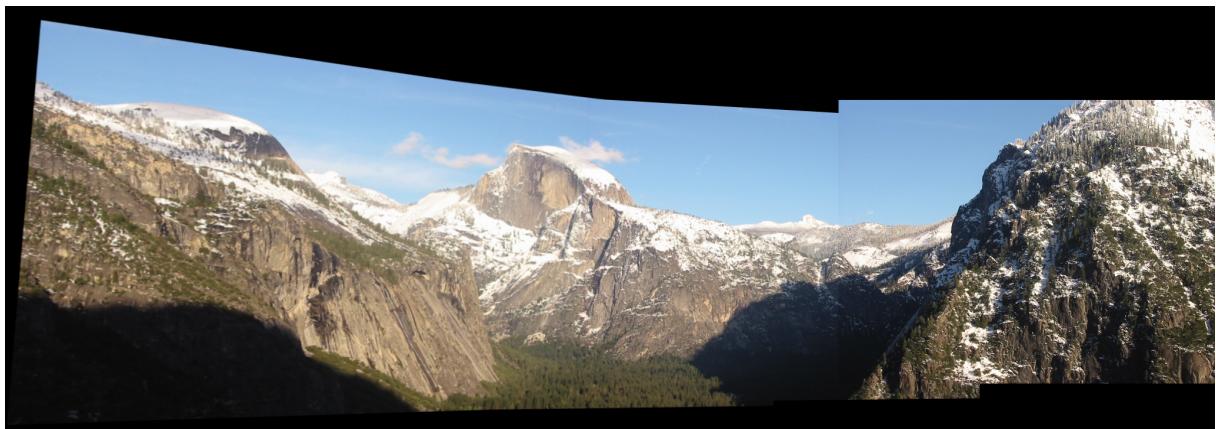


图 13: 多图像拼接效果

6 实验结果展示

在实验过程 5 一节中已经给出了每一步的实验结果，并在这里集中展示实验结果。

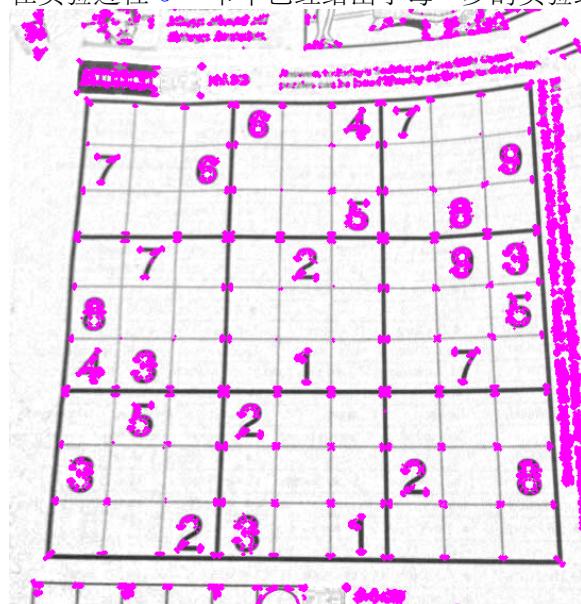


图 14: 手动实现的角点检测结果

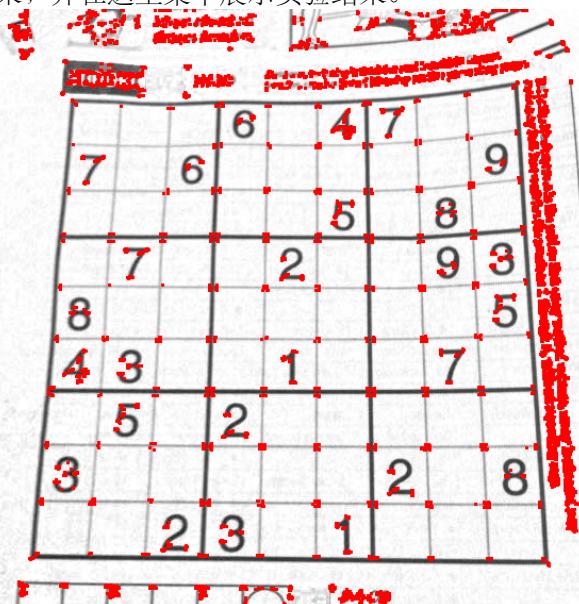


图 15: 调库的角点检测结果



图 16: uttower1 角点

图 17: uttower2 角点



图 18: 10 个最优 SIFT 描述子匹配点对



图 19: 100 个最优 SIFT 描述子匹配点对



图 20: 1000 个最优 SIFT 描述子匹配点对

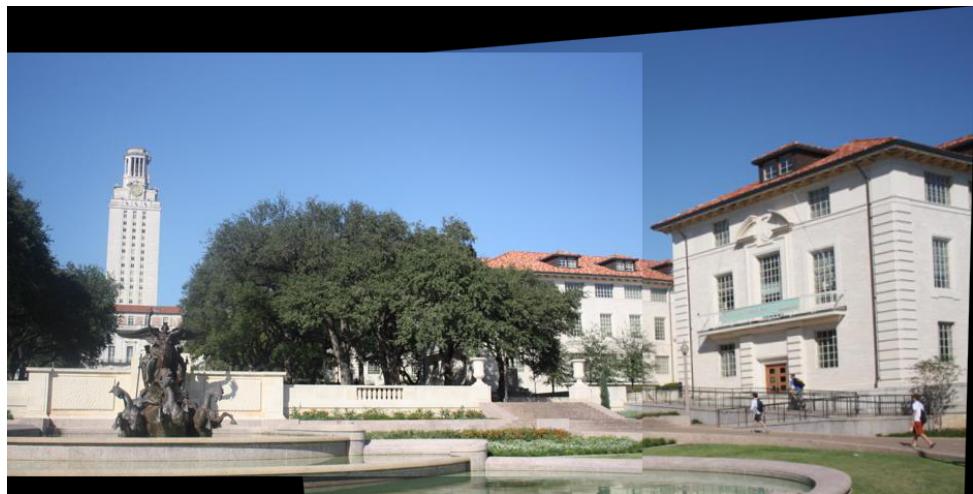


图 21: 基于 SIFT 描述子的图像拼接效果



图 22: 最优 HOG 描述子匹配点对

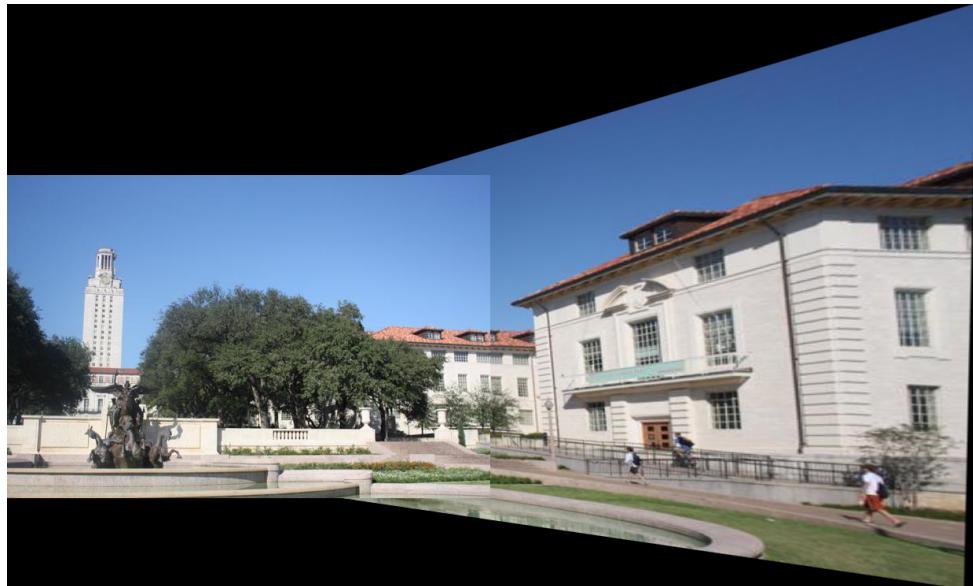


图 23: 基于 HOG 描述子的图像拼接效果

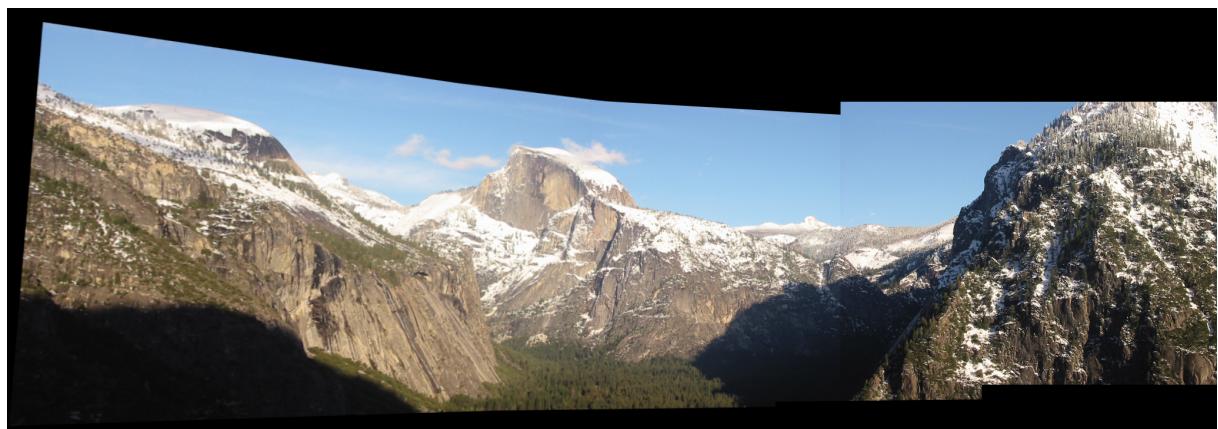


图 24: 基于 SIFT 和 RANSAC 的多图像拼接效果