

Lab3 实验报告

姓名：陈翎玺 学号：523030910039 班级：电院 2302

1 实验概览

本次实验主要学习了 SIFT 算法和其在图像匹配上的应用

SIFT 算法是 David Lowe 于 1999 年提出的局部特征描述子，并于 2004 年进行了更深入的发展和完善。Sift 特征匹配算法可以处理两幅图像之间发生平移、旋转、仿射变换情况下的匹配问题，具有很强的匹配能力。总体来说，SIFT 算子具有以下特性：

1. SIFT 特征是图像的局部特征，对平移、旋转、尺度缩放、亮度变化、遮挡和噪声等具有良好的不变性，对视觉变化、仿射变换也保持一定程度的稳定性。

2. 独特性好，信息量丰富，适用于在海量特征数据库中进行快速、准确的匹配。

3. 多量性，即使少数的几个物体也可以产生大量 SIFT 特征向量。

4. 速度相对较快，经优化的 Sift 匹配算法甚至可以达到实时的要求。

5. 可扩展性强，可以很方便的与其他形式的特征向量进行联合。

SIFT 算法的实质是在不同的尺度空间上查找关键点（特征点），并计算出关键点的方向。SIFT 所查找到的关键点是一些十分突出，不会因光照、仿射变换和噪音等因素而变化的点，如角点、边缘点、暗区的亮点及亮区的暗点等。

SIFT 算法主要步骤：

- a. 检测尺度空间关键点
- b. 精确定位关键点
- c. 为每个关键点指定方向参数
- d. 关键点描述子的生成

2 练习题的解决思路

2.1 图像关键点提取——高斯金字塔方法

2.1.1 高斯金字塔

尺度空间使用高斯金字塔表示。Tony Lindeberg 指出尺度规范化的 LoG (Laplacian of Gaussian) 算子具有真正的尺度不变性，Lowe 使用高斯差分金字塔近似 LoG 算子，在尺度空间检测稳定的关键点。尺度空间的表示如下：

对原图像 $I(x, y)$ 建立尺度空间 $L(x, y, \sigma)$ 的方法是将图像 $I(x, y)$ 与高斯核 $G(x, y, \sigma)$ 卷积：

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

其中 $G(x, y, \sigma)$ 是高斯卷积核：

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

将两个不同尺度的 L 相减可得到高斯差分图像：

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

构建 $D(x, y, \sigma)$ 是为了检测尺度空间极值点。

为了让尺度体现其连续性，在简单下采样的基础上加上了高斯滤波。一幅图像可以产生几组图像，一组图像包括几层图像。高斯金子塔的构建过程可分为两步：

1. 对图像做高斯平滑；
2. 对图像做降采样

根据论文内容，实际实现与手册/ppt 描述略有不同。对于给定的照片，假定其已进行了 $\sigma = 0.5$ 的高斯模糊，首先将其拉伸至原本的两倍，这一步通过插值填补缺失的像素值。此时其对应的 $\sigma' = 0.5 \times 2 = 1$ 。原文取 $\sigma_0 = 1.6$ ，由高斯模糊的性质（即对一个图像先后做 σ_1, σ_2 的高斯模糊等于做 $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$ 的高斯模糊），需要再进行一次 $\sigma'' = \sqrt{1.6 * 1.6 - 1 * 1}$ 的高斯模糊，得到第一组第一张图像。

随后取 $k = 2^{\frac{1}{3}}$ ，根据上述高斯模糊的性质生成每组的第 2 至 6 层图像。由论文内容，取第四张（也就是每组倒数第三张）降采样后作为下一组的第一张。

一张图像。降采样的操作为提取原图像所有偶数坐标的点，连接后得到原图像分辨率 $\frac{1}{4}$ 的新图像。

与手册/ppt 描述不同的是，由于降采样会导致新图像的高斯模糊等效为原本的一半，这里每组同一层图像的高斯模糊值都应该相等，而非描述中组间差距 2 倍。同样取层间高斯模糊的比例为 k ，重复上述操作，最后得到 6 组 6 层图像。

这里，组数 6 由 $\log_2(\min H, W - 3)$ 得到，其中 H, W 代表初始图像的高和宽。层数选取将在下一节中解释。

An efficient approach to construction of $D(x, y, \sigma)$ is shown in Figure 1. The initial image is incrementally convolved with Gaussians to produce images separated by a constant factor k in scale space, shown stacked in the left column. We choose to divide each octave of scale space (i.e., doubling of σ) into an integer number, s , of intervals, so $k = 2^{1/s}$. We must produce $s + 3$ images in the stack of blurred images for each octave, so that final extrema detection covers a complete octave. Adjacent image scales are subtracted to produce the difference-of-Gaussian images shown on the right. Once a complete octave has been processed, we resample the Gaussian image that has twice the initial value of σ (it will be 2 images from the top of the stack) by taking every second pixel in each row and column. The accuracy of sampling relative to σ is no different than for the start of the previous octave, while computation is greatly reduced.

图 1: 论文原文 1

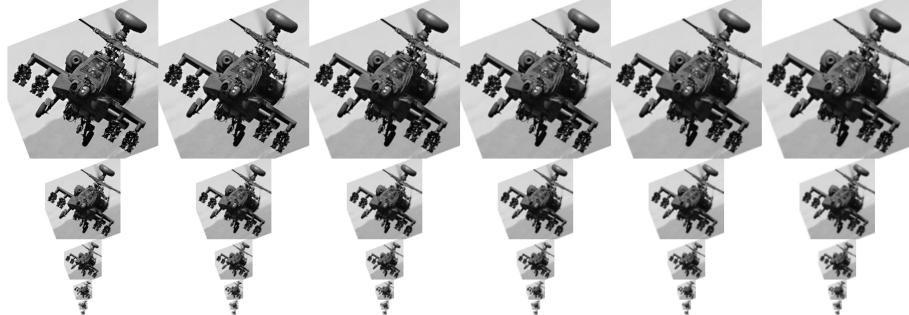


图 2: 高斯金字塔

2.1.2 DoG 金字塔

高斯金字塔中每组的相邻两层相减得到，高斯金字塔有每组 $S + 3$ 层图像。DoG 金字塔每组有 $S + 2$ 层图像。得到的金字塔图像如图所示：由于

后续使用中 DoG 金字塔的头尾两层不直接使用，仅用到中间的 S 层图像。这就解释了为什么开始需要 $S + 3$ 层图像。由于本次实验选取 $S = 3$ ，所以最开始为 6 层。

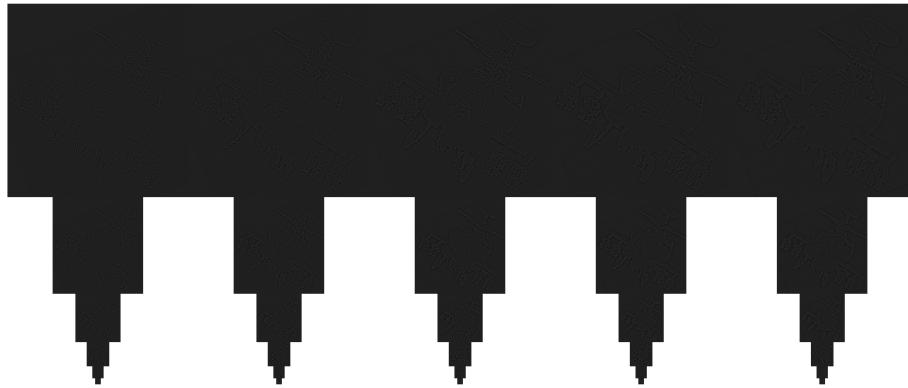


图 3: DoG 金字塔

这里图像极黑是合理的，因为高斯模糊得到的像素值差距并不大，需放大仔细看，能看到相对清晰的边缘。

2.1.3 准确定位关键点

对于 $D(x, y, d)$ 中的某个点，如果它的值大于或小于其周围的 8 个点和它上下相邻尺度的 18 个点，则该点是一个极值点。

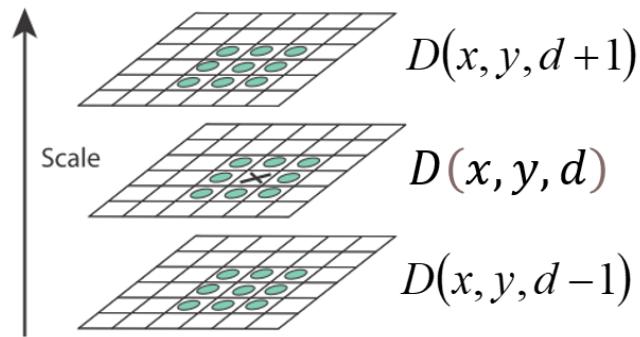


图 4: 原理图 1

对于上述步骤中提取的尺度空间极值点，下一个步骤要对它们进行精选，排除掉一些低对比度的或位于边缘上的点。

对函数 $D(x, y, d)$ 做泰勒展开：

$$D(\vec{x}) = D(\vec{x}_0) + \nabla D(\vec{x}_0)^T (\vec{x} - \vec{x}_0) + \frac{1}{2} (\vec{x} - \vec{x}_0)^T \nabla^2 D(\vec{x}_0) (\vec{x} - \vec{x}_0)$$

求导，令 $\nabla D(\vec{x}) = \vec{0}$

得

$$\hat{x} = - \left[\frac{\partial^2 D}{\partial \vec{x}^2} \right]^{-1} \frac{\partial D}{\partial \vec{x}}$$

$$D(X = \vec{x}_0 + \hat{x}) = D(\vec{x}_0) + \frac{1}{2} \nabla D(\vec{x}_0)^T \hat{x}$$

即经典的牛顿下降法。当 \hat{x} 中的每一个元素均处于 $(-0.5, 0.5)$ 范围内，就视为找到了极值点。根据论文，筛选出 $|D(X)| \geq 0.03$ 的点作为特征点。（此处开始将像素值映射至 $(0, 1)$ 区间处理）

另外，由于图像为离散的点，我们约定用以下方法求解梯度和 Hessian 矩阵

对于梯度 $\nabla D(\vec{x})$ ，我们选择其相邻的像素计算。考虑到间隔，我们将差值除以 2。结果如下：

$$\nabla D(x, y, d) = \frac{1}{2} \begin{bmatrix} D(x, y+1, d) - D(x, y-1, d) \\ D(x+1, y, d) - D(x-1, y, d) \\ D(x, y, d+1) - D(x, y, d-1) \end{bmatrix}$$

这里方向看似有些奇怪，原因在于此处 x, y 按平面直角坐标系选取，而实际存储的 x, y 对应 r, c 。

而对于 Hessian 矩阵，混合系数部分同理梯度算子，再次求导，用对应方向上的相邻像素点作差除 2 替代，最后带有常数 $\frac{1}{4}$ 。而主对角线上的三个元素，用以下方法，以 x 为例：

$$f''(x) = f'(x+1) - f'(x) = f(x+1) - f(x) - (f(x) - f(x-1))$$

$$f''(x) = f(x+1) - 2f(x) + f(x-1)$$

故不需要额外的系数。仿照上式计算即可。

本次实验中采取迭代 6 次作为判断，如迭代 6 次仍不能稳定则舍去。同时对于 Hessian 矩阵不可逆的情况同样舍去。如迭代中遇到边界等情况同样舍去。

论文中还提到要排除边缘处的关键点。具体方法如下：

取 Hessian 矩阵左上角的 2×2 子矩阵 H ，记

$$H = \begin{bmatrix} H_{xx} & H_{xy} \\ H_{yx} & H_{yy} \end{bmatrix}$$

矩阵的迹 $tr(H)$ 和行列式 $det(H)$ ，如果

$$\frac{tr(H)^2}{det(H)} < \frac{(r_0 + 1)^2}{r_0}$$

则舍去这个关键点。论文取 $r_0 = 10$ 。

其原因在于高斯函数的差值沿边缘有很强的响应，甚至会导致无法确定边缘，对少量噪声不稳定。（翻译自论文原文，但解释并不清晰，参考其 4.1 节）

For stability, it is not sufficient to reject keypoints with low contrast. The difference-of-Gaussian function will have a strong response along edges, even if the location along the edge is poorly determined and therefore unstable to small amounts of noise.

A poorly defined peak in the difference-of-Gaussian function will have a large principal curvature across the edge but a small one in the perpendicular direction. The principal curva-

图 5: 论文原文 2

而在实验中发现单纯按上述方法提取关键点存在问题。仅根据 DoG 图中的数值为周围像素点极值提取的像素点这一限制条件，会提取多达 $2e6$ 以上数量的潜在特征点。其远远超出了计算机的处理能力。原因在于多数空间范围由于像素变化不大，整体差值接近 0 导致的。这显然不是我们希望的特征点。故在第一步筛选时增加了 DoG 绝对值大于等于 0.04 和 0.03 两组限制条件，期望得到更好的边缘点并减小后续计算复杂度。在上述限制条件下，潜在特征点的数量分别降至 2405 和 4756，特征点数量降至 866 和 1605，具体结果如下图所示（左图为 0.04，右图为 0.03）：

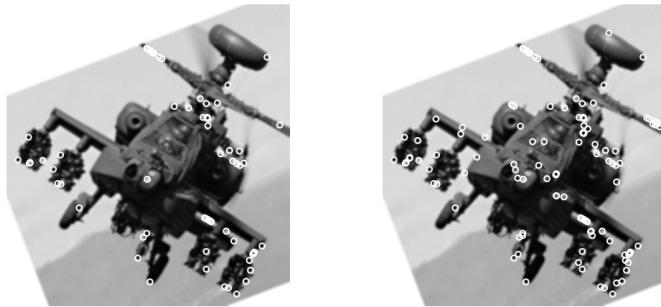


图 6: 特征点提取结果



图 7: 标准库结果

对比标准库结果可以发现，0.03 组效果接近标准库的实现，但仍然存在特征提取不足，部分区域特征提取过密等情况。同时，标准库的运行时间远小于手动实现的特征点提取。因此后续实验采用 opencv 标准库的 goodFeaturesToTrack 函数实现特征提取（即 Harris 方法）

2.2 图像关键点提取——Harris 角点提取函数

与用高斯差分构建的尺度空间不同，图像金字塔通过直接缩放图像的方式构建尺度空间。

通过 `im2 = cv2.resize(im1,(size,size))` 函数可以快速构建一系列分辨率不一的图像，再通过对应函数提取角点。

主要用到 cv2.goodFeaturesToTrack(image, maxCorners, qualityLevel, minDistance [,corners[,mask[,blockSize[,useHarrisDetector[,k]]]]])->corners
image: 输入图像;
maxCorners: 允许返回的最多角点个数, 设为 200;
qualityLevel: 图像角点的最小可接受参数, 质量测量值乘以这个参数就是最小特征值, 小于这个数的会被抛弃, 设为 0.01;
minDistance: 返回的角点之间最小的欧式距离, 设为 10;
blockSize: 邻域大小, 设为 3;
k:Harris 检测的自由参数, 越小返回的角点数越多, 设为 0.04。

检测结果如图所示:



图 8: Harris 角点检测结果

2.3 SIFT 描述子

SIFT 描述子之所以具有旋转不变的性质是因为人们能通过物体的局部细节潜在地建立起物体坐标系, 并建立其和图像坐标系之间的映射关系。而物体之间的相对关系并不随图像的旋转发生变化。

SIFT 描述子把以关键点为中心的邻域内的主要梯度方向作为物体坐标系的 X 方向, 因为该坐标系是由关键点本身的性质定义的, 因此具有旋转不变性。

对应论文第 5 节: 假设某尺度下某关键点为 $L(x, y)$, 对于它 $m*m$ 邻域内 (其所在高斯金字塔图像 3 邻域窗口内) 的每个点, 计算其在该尺度下的梯度强度和梯度方向 (这里无需缩放, 因为比例因子的贡献会抵消):

$$mag(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1} \left[\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \right]$$

随后根据 θ 的值（需根据 L_x, L_y 的正负确定方向，范围为 $[0, 360)$ ）分成 36 等分，将梯度强度乘上该点对应的高斯函数值（作为权重）加入对应方向，以类似于直方图的形式统计。

实验中简化了论文的做法，只选取了最大的方向作为 SIFT 算子的方向，即物体坐标系的 x 轴正方向。逆时针旋转 90° 得 y 轴正方向。

再根据对应的 x,y 轴方向，选取 16×16 个像素，分成 4×4 组，每组 4×4 个像素。对于每组，将梯度强度按梯度方向统计至对应的八个方向，这样一共有 16×8 个方向，归一化 (除以平方和的平方根，L2-norm) 后即 SIFT 特征值。

对于最后的匹配，暴力枚举两张图片中的关键点，选择 SIFT 特征值点积最大且大于 0.7 的组，认为它们是匹配的关键点。

以上为理论分析。在实际代码中做了以下的调整：

1. 考虑到 3σ 范围的像素点选取和图像降采样导致的等效 σ 减半，对每张图像做一次高斯模糊，使其 $\sigma = 1.6$

2. 根据论文内容，这边选择了半径为 $3 \times 1.5 \times \sigma$ 作为实际统计的范围。代码中为避免麻烦统计了正方形区域，理论而言如果选择圆形区域效果会更好。

An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint. The orientation histogram has 36 bins covering the 360 degree range of orientations. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint.

图 9: 论文原文 3

3. 考虑低分辨率图像的范围不足，代码中选择求得梯度方向后映射至原始图像求解 SIFT 特征值。也可以尝试直接在对应图像上进行求解。由于未做对比实验，无法确定两者做法的优劣。

4. 对于 SIFT 特征值的计算：由于像素为离散值，这里选择了中心坐标 ± 7.5 ，间隔为 1 的方式选取了 256 个像素点。根据得到的 θ 值，对相对坐标做变换

$$rotary = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

加上中心点坐标得到对应坐标。首先对四个角进行检测，如存在超出边界的则舍去这个点。随后进行相应的统计即可。如果求得的像素不为整点，使用双线性插值方法，以周围四个整点的梯度值插值得该像素的梯度大小及方向。如图所示：

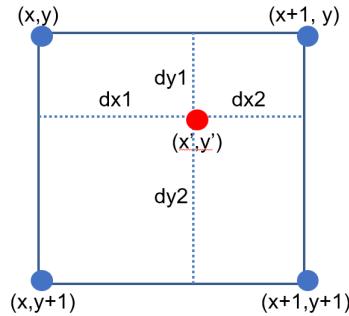


图 10: 双线性插值

3 代码运行结果

3.1 手动实现的结果

通过比较生成的特征点数量，代码能正确找到匹配最多的图像。但由于部分实现的简化和参数设置问题，匹配未能实现理想情况下关键点连线基本平行的结果。

其原因可能在于局部特征的相似性导致相对位置不同的点被误认为匹配，如飞机下端对称的发射器。也有可能是代码中对于高维特征提取不足导致的。

以上依次为图 1、2、4、5 的匹配结果。图 2 中特征点匹配数量较多，可能是由于桥身主体为黑色，接近飞机特征所致。

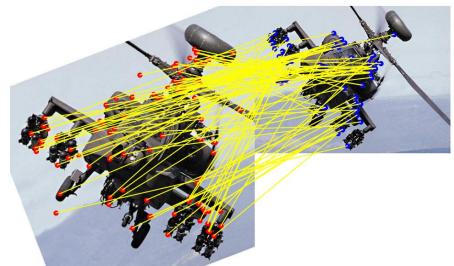


图 11: 匹配结果 1

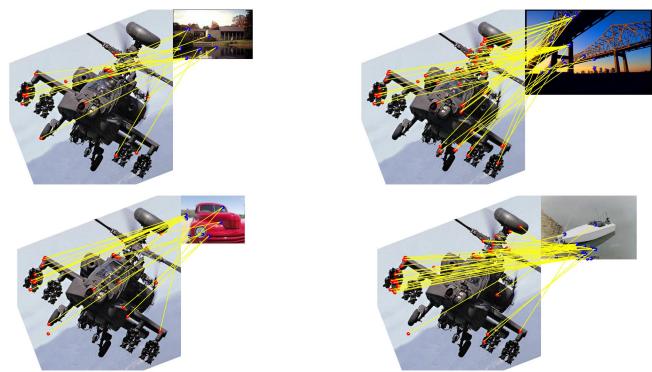


图 12: 匹配结果 2

3.2 系统库的实现结果



图 13: 匹配结果 3

对于正确的匹配（图 3）系统库的关键点匹配效果远好于手动实现，且速度较快（1s，手动实现计算五幅图共用时 30s 左右）总体而言，手动实现的代码仍有很多地方可以改进。

4 实验感想

本次实验学习了 SIFT 算法的实现，对于图像识别算法有了更深入的认识。高斯模糊和高斯差分是有效的图像特征提取的手段，在神经网络尚未发展的过去，这一系列传统方法的实现思路对当今图像识别和计算机视觉的发展仍然有参考价值。

5 相关实现代码

5.1 高斯金字塔方法提取特征点:gauss_method.py

```
1 import cv2
```

```

2 import numpy as np
3 import matplotlib.pyplot as plt
4 import math
5
6 from numpy import dtype
7 from scipy.ndimage import zoom
8
9 image = cv2.imread('./target.jpg', cv2.
    IMREAD_GRAYSCALE)
10 H, W = image.shape
11
12 def gauss_filter(sigma=1.6, k = 3):
13     ax = np.arange(-k, k + 1)
14     x, y = np.meshgrid(ax, ax)
15     gauss_mat = np.exp(-(x ** 2 + y ** 2) / (2 * sigma
16                         ** 2))
17     gauss_mat /= 2 * np.pi * sigma ** 2
18     return gauss_mat / gauss_mat.sum()
19
20 nOctaves = math.floor(math.log2(min(W, H))) - 3
21 nOctaveLayers = 3
22 k = 2 ** (1 / nOctaveLayers)
23 sigma0 = 1.6
24
25 scale = 2
26 image = zoom(image, scale, order=3)
27
28 sigma_init = (sigma0 ** 2 - (2 * 0.5) ** 2) ** 0.5
29 gauss_core = gauss_filter(sigma_init)
30 image = cv2.filter2D(image, -1, gauss_core)
31
32 gauss_pyr_img = [None] * nOctaves
33 DoG_pyr_img = [None] * nOctaves

```

```

33 img_i = image
34
35 for i in range(nOctaves):
36     gauss_oct_i = np.zeros((nOctaveLayers + 3, img_i.
37                             shape[0], img_i.shape[1]), dtype = img_i.dtype)
38     gauss_oct_i[0, :, :] = np.copy(img_i)
39     for j in range(1, nOctaveLayers + 3):
40         sigma_pre = k ** (j - 1) * sigma0
41         sigma_dif = ((k * sigma_pre) ** 2 - sigma_pre
42                       ** 2) ** 0.5
43         G_C = gauss_filter(sigma_dif)
44         img_i = cv2.filter2D(img_i, -1, G_C)
45         gauss_oct_i[j, :, :] = np.copy(img_i)
46         gauss_pyr_img[i] = np.copy(gauss_oct_i)
47         DoG_pyr_img[i] = np.copy(gauss_oct_i.astype(np.
48                                     int32)[1:, :, :] - gauss_oct_i.astype(np.int32)
49                                     [: -1, :, :]).astype(np.int32)
50         img_i = zoom(gauss_oct_i[-3, :, :], 0.5, order =
51                       3)
52
53 # max_width = max([sum(img.shape[1] for img in group)
54 #                   for group in DoG_pyr_img])
55 # total_height = sum(group[0].shape[0] for group in
56 #                      DoG_pyr_img)
57 #
58 # canvas = np.ones((total_height, max_width)) * 255
59
60 # y_offset = 0
61 # for group in DoG_pyr_img:
62 #     group_height = group[0].shape[0]
63 #     x_offset = int(W - group[0].shape[1] / 2)
64 #     for img in group:
65 #         h, w = img.shape

```

```

59 #         canvas[y_offset:y_offset + h, x_offset:
60 #                     x_offset + w] = img
61 #         x_offset += W * 2
62 #         y_offset += group_height
63 #
64 # plt.figure(figsize=(max_width / 100, total_height /
65 #                     100))
66 # plt.imshow(canvas, cmap='gray')
67 # plt.axis('off')
68 # plt.tight_layout()
69 # plt.savefig("./result/DoG_pyr1.png")
70
71 ext = []
72 min_val = 0.03 * 255
73 for i in range(nOctaves):
74     DoG = DoG_pyr_img[i]
75     num, h, w = DoG.shape
76     for j in range(1, num - 1):
77         for r in range(4, h - 3):
78             for c in range(4, w - 3):
79                 neigh = np.copy(DoG[j - 1:j + 2, r -
80                                     1:r + 2, c - 1:c + 2])
81                 val = neigh[1][1][1]
82                 if abs(val) > min_val and (val == np.
83                     max(neigh) or val == np.min(neigh))
84                 :
85                     sigma_i = k ** j * sigma0
86                     ext.append([i, j, r, c, sigma_i])
87
88 keypoint = []
89 for i in range(len(ext)):
90     kpt = ext[i]
91     num_oct = kpt[0]

```

```

87     num_lay = kpt[1]
88     r = kpt[2]
89     c = kpt[3]
90     should_delete = True
91     D_hat = None
92
93     for iter in range(6):
94         DoG = DoG_pyr_img[num_oct][num_lay] / 255
95         DoG_pre = DoG_pyr_img[num_oct][num_lay - 1] /
96                     255
97         DoG_nxt = DoG_pyr_img[num_oct][num_lay + 1] /
98                     255
99
100        dD = np.array([[DoG[r, c + 1] - DoG[r, c -
101                      1]], [
102                          [DoG[r + 1, c] - DoG[r - 1, c
103                            ]], [
104                                [DoG_pre[r, c] - DoG_nxt[r, c]]
105                                  ]) / 2
106        dxx = (DoG[r, c + 1] + DoG[r, c - 1] - 2 * DoG
107          [r, c]) / 1
108        dyy = (DoG[r + 1, c] + DoG[r - 1, c] - 2 * DoG
109          [r, c]) / 1
110        dss = (DoG_pre[r, c] + DoG_nxt[r, c] - 2 * DoG
111          [r, c]) / 1
112
113        dxy = (DoG[r + 1, c + 1] + DoG[r - 1, c - 1] -
114          DoG[r + 1, c - 1] - DoG[r - 1, c + 1]) / 4
115        dxs = (DoG_nxt[r, c + 1] - DoG_nxt[r, c - 1] -
116          DoG_pre[r, c + 1] + DoG_pre[r, c - 1]) / 4
117        dys = (DoG_nxt[r + 1, c] - DoG_nxt[r - 1, c] -
118          DoG_pre[r + 1, c] + DoG_pre[r - 1, c]) / 4

```

```

110     dH = np.array([[dxx, dxy, dxs], [dxy, dyt, dys
111                  ], [dxs, dys, dss]])
112
113     if np.linalg.det(dH) == 0:
114         break
115
116     x_hat = np.linalg.inv(dH) @ dD
117
118
119     if abs(x_hat[0][0]) < 0.5 and abs(x_hat[1][0])
120         < 0.5 and abs(x_hat[2][0]) < 0.5:
121         should_delete = False
122
123         break
124
125         c = c + round(x_hat[0][0])
126         r = r + round(x_hat[1][0])
127         num_lay = num_lay + round(x_hat[2][0])
128
129         if num_lay < 1 or num_lay > nOctaveLayers or \
130             \
131             r <= 0 or r >= DoG_pyr_img[num_oct][
132                 num_lay].shape[0] - 1 or \
133                 c <= 0 or c >= DoG_pyr_img[num_oct][
134                     num_lay].shape[1] - 1:
135
136             break
137
138
139     if should_delete:
140         continue
141
142     D_hat = DoG[r][c] + (dD.T @ x_hat) / 2
143
144     if abs(D_hat) < 0.03:
145         continue
146
147     trH = dxx + dyt
148     detH = dxx * dyt - dxy * dxy

```

```

138     r0 = 10
139
140     if detH <= 0 or trH * trH * r0 >= (r0 + 1) * (r0 +
141         1) * detH:
142         continue
143
144     keypoint.append([num_oct, num_lay, r, c, k**2 *
145         num_lay * sigma0])
146
147 H, W = gauss_pyr_img[1][1].shape
148
149 fig, ax = plt.subplots(figsize=(W / 100, H / 100), dpi
150 =100)
151
152 ax.imshow(gauss_pyr_img[1][1], cmap='gray')
153 ax.axis("off")
154
155 ax.set_xlim(0, W)
156 ax.set_ylim(0, H)
157 ax.invert_yaxis()
158
159 for i in range(len(keypoint)):
160     kpt = keypoint[i]
161     num_oct = kpt[0]
162     num_lay = kpt[1]
163     if num_oct == 1 and num_lay == 1:
164         r = kpt[2]
165         c = kpt[3]
166         if 0 <= r < H and 0 <= c < W:
167             circle = plt.Circle((c, r), radius=5,
168                 color='white', fill=False, linewidth
169                 =1.5)
170             ax.add_patch(circle)

```

```
166  
167 plt.show()  
168 plt.close()
```

5.2 Harris 算法和 SIFT 特征值求解及匹配：main.py

```
1     import cv2  
2     import numpy as np  
3     import matplotlib.pyplot as plt  
4     import math  
5  
6  
7     def gauss_filter(sigma=1.6, k = 3):  
8         ax = np.arange(-k, k + 1)  
9         x, y = np.meshgrid(ax, ax)  
10        gauss_mat = np.exp(-(x ** 2 + y ** 2) / (2 * sigma  
11                           ** 2))  
12        gauss_mat /= 2 * np.pi * sigma ** 2  
13        return gauss_mat / gauss_mat.sum()  
14  
15  
16     def rotary_matrix(theta):  
17         rad = theta / 180 * math.pi  
18         return np.array([[math.cos(rad), -math.sin(rad)],  
19                          [math.sin(rad), math.cos(rad)]])  
20  
21  
22     def sift_opt(path):  
23         image = cv2.imread(path, cv2.IMREAD_COLOR)  
24         image_grey = cv2.cvtColor(image, cv2.  
25                               COLOR_BGR2GRAY)  
26         H, W = image_grey.shape  
27  
28         img = []  
29         sigma = 1.6
```

```

25     init_image = cv2.filter2D(image_grey, -1,
26                               gauss_filter(sigma = (sigma ** 2 - 0.5 * 0.5)
27                                           ** 0.5))
28     img.append(init_image)
29
30     scale = 2
31     level = math.floor(math.log2(min(H, W))) - 3
32     for i in range(level):
33         prev_img = img[-1]
34         new_width = prev_img.shape[1] // scale
35         new_height = prev_img.shape[0] // scale
36         img1 = cv2.resize(prev_img, (new_width,
37                                     new_height))
38         img1 = cv2.filter2D(img1, -1, gauss_filter(
39                               sigma = (sigma ** 2 - (0.5 * sigma) ** 2)
40                               ** 0.5))
41         img.append(img1)
42
43     keypoints = []
44     for i, imgi in enumerate(img):
45         corners = cv2.goodFeaturesToTrack(imgi,
46                                         maxCorners=300, qualityLevel=0.01,
47                                         minDistance=8, blockSize=3, k=0.04)
48         for j in corners:
49             keypoints.append(np.array([j[0][0], j
50                                       [0][1], i]))
51
52     filter = []
53     for kpt in keypoints:
54         x = int(kpt[0])
55         y = int(kpt[1])
56         layer = int(kpt[2])
57         radius = round(3 * 1.5 * sigma)

```

```

50     img_i = img[layer]
51     h, w = img_i.shape
52
53
54     deg = np.zeros(36, dtype = int)
55     for i in range(-radius, radius + 1):
56         r = x + i
57         if r <= 0 or r >= h - 1:
58             continue
59         for j in range(-radius, radius + 1):
60             c = y + j
61             if c <= 0 or c >= w - 1:
62                 continue
63             dx = int(img_i[r][c + 1]) - int(img_i[r][c - 1]) / 2
64             dy = int(img_i[r - 1][c]) - int(img_i[r + 1][c]) / 2
65
66             dx = int(dx)
67             dy = int(dy)
68
69             mag = (dx * dx + dy * dy) ** 0.5
70             if dy >= 0:
71                 ang = math.atan2(dy, dx)
72             else:
73                 ang = math.atan2(dy, dx) + 2 *
74                         math.pi
75             ang = ang * 180 / math.pi
76
77             bin = round(0.1 * ang)
78             if bin >= 36:
79                 bin -= 36
80             elif bin < 0:

```

```

80             bin += 36
81             weight = np.exp(-(i ** 2 + j ** 2) /
82                               (2 * radius ** 2))
83             deg[bin] += mag * weight
84             is_zero = True
85             for i in range(len(deg)):
86                 if deg[i] != 0:
87                     is_zero = False
88             if is_zero:
89                 continue
90             max_deg = max(deg)
91             for i in range(36):
92                 if deg[i] == max_deg:
93                     dir = i
94             filter.append((x * (2 ** (layer)), y * (2 ** (
95                                         layer)), dir))
96
97             sift = []
98             for i in range(len(filter)):
99                 is_edge = False
100                x = filter[i][0]
101                y = filter[i][1]
102                theta = filter[i][2] * 10
103                for j in (-7.5, 7.5):
104                    for k in (-7.5, 7.5):
105                        R_M = rotary_matrix(theta)
106                        pos = R_M @ np.array([[j], [k]])
107                        x_ = pos[0][0] + x
108                        y_ = pos[1][0] + y
109                        if x_ <= 2 or x_ >= H - 3 or y_ <= 2
110                            or y_ >= W - 3:

```

```

110                     is_edge = True
111
112     if is_edge:
113         continue
114
115     cnt = np.zeros((4, 4, 8), dtype = float)
116     for j in range(-8, 8):
117         for k in range(-8, 8):
118             R_M = rotary_matrix(theta)
119             pos = R_M @ np.array([[j + 0.5], [k +
120                                     0.5]])
121             x_ = pos[0][0] + x
122             y_ = pos[1][0] + y
123
124             dx1 = x_ - math.floor(x_)
125             dx2 = math.ceil(x_) - x_
126             dy1 = y_ - math.floor(y_)
127             dy2 = math.ceil(y_) - y_
128
129             _x = math.floor(x_)
130             _y = math.floor(y_)
131
132             gradxy = np.array([int(image_grey[_x] [
133                                         _y + 1]) - int(image_grey[_x] [_y -
134                                         1]),
135                                 int(image_grey[_x -
136                                         1][_y]) - int(
137                                         image_grey[_x +
138                                         1][_y])]) / 2
139             gradx1y1 = np.array([int(image_grey[_x
140                               + 1][_y + 2]) - int(image_grey[_x
141                               + 1][_y]),
142                                 int(image_grey[_x
143                               ][_y + 1]) -

```

```

134                                     int(image_grey[
135                                         _x + 2][_y +
1]]) / 2
gradx1y = np.array([int(image_grey[_x
+ 1][_y + 1]) - int(image_grey[_x +
1][_y]),

136                                     int(image_grey[_x][
_y]) - int(
image_grey[_x +
2][_y])) / 2
gradxy1 = np.array([int(image_grey[_x
][_y + 2]) - int(image_grey[_x][_y
]),

137                                     int(image_grey[_x -
1][_y + 1]) -
int(image_grey[_x + 1][_y + 1])
]) / 2

138
139             dxy = dx2 * dy2 * gradxy + dx1 * dy1 *
gradx1y1 + dx1 * dy2 * gradx1y +
dx2 * dy1 * gradxy1
140             DX = dxy[0]
141             DY = dxy[1]

142
143             mag = (DX * DX + DY * DY) ** 0.5
144             if DY >= 0:
145                 ang = math.atan2(DY, DX)
146             else:
147                 ang = math.atan2(DY, DX) + 2 *
math.pi
148             ang = ang * 180 / math.pi
149

```

```

150         bin = round(ang / 45)
151         if bin >= 8:
152             bin -= 8
153         elif bin <0:
154             bin += 8
155         posi = math.floor((j + 8) / 4)
156         posj = math.floor((k + 8) / 4)
157         cnt[posi][posj][bin] += mag
158         is_empty = True
159         for _ in cnt:
160             for __ in _:
161                 for ___ in __:
162                     if ___ != 0:
163                         is_empty = False
164         if is_empty:
165             continue
166         cnt = cnt / (np.sum(np.square(cnt)) ** 0.5)
167         cnt = cnt.reshape(1, -1)
168         sift.append((x, y, np.copy(cnt)))
169     return sift
170
171 path1 = "./target.jpg"
172
173 sift1 = sift_opt(path1)
174 for idx in range(1, 6):
175     path2 = "./dataset/" + str(idx) + ".jpg"
176     sift2 = sift_opt(path2)
177
178     num = 0
179     match = []
180     for i in range(len(sift1)):
181         maxn = 0.0
182         id = 0

```

```

183     for j in range(len(sift2)):
184         tot = 0.0
185         for k in range(128):
186             tot += sift1[i][2][0][k] * sift2[j]
187                         [2][0][k]
188             if tot > maxn:
189                 maxn = tot
190                 id = j
191             if maxn >= 0.7:
192                 match.append((i, id))
193                         # num += 1
194
195     img1 = cv2.imread(path1, cv2.IMREAD_COLOR)
196     img2 = cv2.imread(path2, cv2.IMREAD_COLOR)
197
198     img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
199     img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
200
201     h1, w1, _ = img1.shape
202     h2, w2, _ = img2.shape
203
204     canvas_height = max(h1, h2)
205     canvas_width = w1 + w2
206     canvas = 255 * np.ones((canvas_height,
207                           canvas_width, 3), dtype=np.uint8)
208
209     canvas[:h1, :w1, :] = img1
210     canvas[:h2, w1:w1 + w2, :] = img2
211
212     plt.figure(figsize=(12, 6))
213     plt.imshow(canvas)
214     for (i, j) in match:

```

```

214     x1 = sift1[i][0]
215     y1 = sift1[i][1]
216
217     x2 = sift2[j][0]
218     y2 = sift2[j][1]
219
220     plt.scatter(x1, y1, color='red', s=50)
221     plt.scatter(x2 + w1, y2, color='blue', s=50)
222
223     plt.plot([x1, x2 + w1], [y1, y2], color='
224             yellow', linewidth=1.5)
225
226     plt.axis('off')
227     plt.tight_layout()
228     plt.savefig("./result/" + str(idx) + ".jpg")
229
230     plt.close()

```

5.3 官方库实现: official.py

```

1   import cv2
2   import matplotlib.pyplot as plt
3   import math
4   import numpy as np
5
6   target_image_path = './target.jpg'
7   dataset_image_paths = ['./dataset/1.jpg', './dataset
8   /2.jpg', './dataset/3.jpg', './dataset/4.jpg', './
9   dataset/5.jpg']
10
11  def show_keypoint1():
12      img = cv2.imread('./target.jpg')

```

```
12     if img is None:
13         raise FileNotFoundError("无法找到指定的图像文
14            件。请检查路径 './target.jpg'")
15
16     cat = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
17
18     sift = cv2.SIFT_create()
19
20     kp, des = sift.detectAndCompute(cat, None)
21
22     img_with_keypoints = cv2.drawKeypoints(
23         img, kp, None, flags=cv2.
24             DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
25     )
26
27     plt.figure(figsize=(img.shape[1] / 100, img.shape
28             [0] / 100), dpi=100)
29     plt.imshow(cv2.cvtColor(img_with_keypoints, cv2.
30             COLOR_BGR2RGB))
31     plt.title('SIFT Keypoints')
32     plt.xticks([]), plt.yticks([])
33     plt.show()
34
35
36     def show_keypoint2():
37         def gauss_filter(sigma=1.6, k=3):
38             ax = np.arange(-k, k + 1)
39             x, y = np.meshgrid(ax, ax)
40             gauss_mat = np.exp(-(x ** 2 + y ** 2) / (2 *
41                 sigma ** 2))
42             gauss_mat /= 2 * np.pi * sigma ** 2
43             return gauss_mat / gauss_mat.sum()
44
45
46         path = "./dataset/3.jpg"
47         image = cv2.imread(path, cv2.IMREAD_COLOR)
```

```
40     if image is None:
41         raise FileNotFoundError("无法找到指定的图像文
42            件，请检查路径 './target.jpg'")
43
44     image_grey = cv2.cvtColor(image, cv2.
45         COLOR_BGR2GRAY)
46     H, W = image_grey.shape
47
48     img = []
49     sigma = 1.6
50     init_image = cv2.filter2D(image_grey, -1,
51         gauss_filter(sigma=(sigma ** 2 - 0.5 * 0.5) **
52             0.5))
53     img.append(init_image)
54
55     scale = 2
56     level = math.floor(math.log2(min(H, W))) - 3
57     for i in range(level):
58         prev_img = img[-1]
59         new_width = prev_img.shape[1] // scale
60         new_height = prev_img.shape[0] // scale
61         img1 = cv2.resize(prev_img, (new_width,
62             new_height))
63         img1 = cv2.filter2D(img1, -1, gauss_filter(
64             sigma=(sigma ** 2 - (0.5 * sigma) ** 2) **
65                 0.5))
66         img.append(img1)
67
68     keypoints = []
69     for i, imgi in enumerate(img):
70         corners = cv2.goodFeaturesToTrack(imgi,
71             maxCorners=200, qualityLevel=0.01,
72             minDistance=8, blockSize=3, k=0.04)
```

```
64     if corners is not None:
65         for j in corners:
66             x, y = j[0]
67             keypoints.append((x * (2 ** i), y * (2
68                           ** i)))
69
70             image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
71             plt.imshow(image)
72             for keypoint in keypoints:
73                 x, y = keypoint
74                 circle = plt.Circle((x, y), radius=5, color='
75                               white', fill=False)
76                 plt.gca().add_patch(circle)
77                 plt.gca().set_aspect('equal', adjustable='datalim'
78 )
79                 plt.axis('off')
80                 plt.show()
81
82             target_image = cv2.imread(target_image_path, cv2.
83             IMREAD_GRAYSCALE)
84             if target_image is None:
85                 print(f"无法加载目标图像 {target_image_path}")
86                 exit()
87
88             sift = cv2.SIFT_create()
89             keypoints_target, descriptors_target = sift.
90                 detectAndCompute(target_image, None)
91             bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
92
93             for i, dataset_image_path in enumerate(
```

```
dataset_image_paths):
92     dataset_image = cv2.imread(dataset_image_path, cv2
93         .IMREAD_GRAYSCALE)
94     if dataset_image is None:
95         print(f"无法加载数据集图像 {dataset_image_path}
96             }")
97         continue
98
99     keypoints_dataset, descriptors_dataset = sift.
100        detectAndCompute(dataset_image, None)
101
102     matches = bf.match(descriptors_target,
103         descriptors_dataset)
104     matches = sorted(matches, key=lambda x: x.distance
105         )
106
107     match_result = cv2.drawMatches(
108         cv2.cvtColor(cv2.imread(target_image_path, cv2
109             .IMREAD_COLOR), cv2.COLOR_BGR2RGB),
110         keypoints_target,
111         cv2.cvtColor(cv2.imread(dataset_image_path,
112             cv2.IMREAD_COLOR), cv2.COLOR_BGR2RGB),
113         keypoints_dataset,
114         matches[:50], None, flags=cv2.
115             DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
116         )
117
118     plt.figure(figsize=(12, 6))
119     plt.imshow(match_result)
120     plt.axis('off')
121     plt.tight_layout()
122     plt.savefig("./official/" + str(i + 1) + ".jpg")
123     plt.close()
```