Lab3 实验报告

李青雅 523030910004 电院 2301

2024 年 12 月 15 日

目录

1 实验概览

本次实验主要是对图像特征提取的算法进行实现，主要为 SIFT 特征提取算法的实验与对比。

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 的算法特点如下：

- 保持不变性与稳定性：SIFT 特征是图像的局部特征，其对旋转、尺度缩放、亮度变化保持不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性。
- 独特性：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。
- 多量性：即使少数的几个物体也可以产生大量 SIFT 特征向量。
- 高速性：采用快速的算法，使得特征点的检测与描述子的计算可以实时进行。
- 可扩展性：可以很方便的与其他形式的特征向量进行联合。

并且 SIFT 算法在一定程度上可以解决尺度、旋转、仿射变换、光照影响，目标遮挡，噪声等问题，因此在实际应用中有着广泛的应用。

2 实验环境

Python, OpenCV, Numpy, Matplotlib

3 SIFT 算法原理

1. 检测尺度空间极值：搜索所有尺度上的图像位置。通过高斯微分函数来识别潜在的对于尺度和旋转不变的兴趣点。
2. 关键点的精确定位：通过拟合三维二次函数来精确确定关键点的位置和尺度，同时去除低对比度的关键点和不稳定的边缘响应点（因为 DOG 算子会产生较强的边缘响应），以增强匹配稳定性、提高抗噪声能力。
3. 关键点主方向分配：基于图像局部的梯度方向，分配给每个关键点位置一个或多个方向。所有后面的对图像数据的操作都相对于关键点的方向、尺度和位置进行变换，使得描述符具有旋转不变性。
4. 关键点的特征描述：对于每一个关键点，拥有三个信息：位置、尺度以及方向。接下来就是为每个关键点建立一个描述符，用一组向量将这个关键点描述出来，使其不随各种变化而改变，比如光照变化、视角变化等。这个描述子不但包括关键点，也包含关键点周围对其有贡献的像素点，并且描述符应该有较高的独特性，以便于提高特征点正确匹配的概率。

4 实验流程

4.1 练习一：在 dataset 文件夹中的所有图片中搜索 target.jpg 图片所示物体，并绘制程序认为的好的匹配。

SIFT 算法的实现主要分为以下几个步骤：

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

由于上述在尺度空间中提取极值点再精确定位和消除边缘响应的方法实现起来较为复杂，实验中可用另一种较简单的替代方案，即在图像金字塔中提取角点（使用 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。其中我们需要用到函数 $im2 = cv2.resize(im1, (size, size))$

4.1.1 关键点提取的简化操作

通过构建图像金字塔，我们可以在不同尺度上检测到角点，然后通过 Harris 角点检测算法来检测角点。在检测角点时，如果对阈值不加以限制，会检测到大量的角点，一部分角点并不是我们所需要的关键点。所以我们需要设置合适的阈值来筛选出我们需要的关键点。

我们已将检测出的关键点标记出来，如下图所示。具体实现的代码将在报告末尾给出。

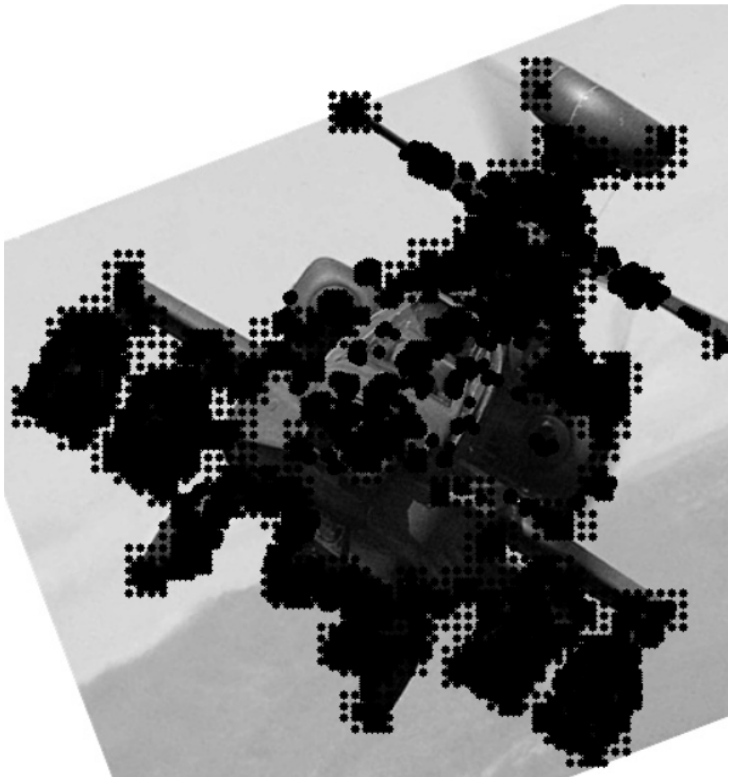


图 1: 关键点检测结果

4.1.2 关键点描述子的生成

SIFT 描述子之所以具有旋转不变 (rotation invariant) 的性质是因为在图像坐标系和物体坐标系之间变换。

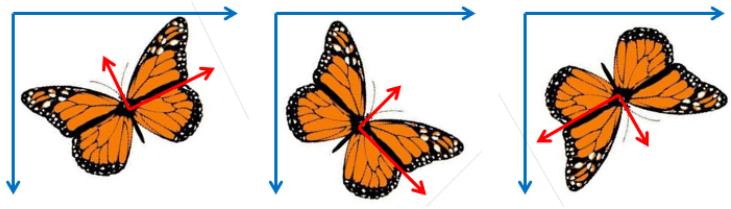


图 2

如上图所示，蓝色箭头表示图像坐标系，红色箭头表示物体坐标系。图像坐标系即观测图像的横向和纵向两个方向，也是计算机处理图像时所参照的坐标系。而物体参照系是人在认知物体时参照的坐标系。人脑之所以能够分辨不同方向的同一物体，是因为能够从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。

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。

接下来介绍如何计算关键点描述子：

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 3σ 邻域窗口内）的每个点，计算其在该尺度下的梯度方向和梯度强度：

$$\text{梯度方向：}\ \theta(x,y) = \arctan(\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)})$$

$$\text{梯度强度：}\ m(x,y) = \sqrt{(L(x+1,y)-L(x-1,y))^2 + (L(x,y+1)-L(x,y-1))^2}$$

梯度方向为 360 度（注意 atan 函数返回值为 0-180，需要根据 lx ly 的符号换算），平均分成 36 个 bins，每个像素以 m(x, y) 为权值为其所在的 bin 投票。最终权重最大的方向定位该关键点的主方向 (实验中只考虑 highest peak)。

SIFT 描述子的统计在相对物体坐标系以关键点为中心的 16×16 的领域内统计，先把之前计算的梯度方向由图像坐标系换算到物体坐标系，即

$$\theta' = \theta(x,y) - \theta_0$$

其中 θ' 是相对物体坐标系的梯度方向，θ 是相对图像坐标系的梯度方向，θ_0 是关键点的主方向。

物体坐标系 16*16 的邻域分成 4*4 个块，每个块 4*4 个像素。在每个块内按照求主方向的方式把 360 度分成 8 个 bins，统计梯度方向直方图，最终每个块可生成 8 维的直方图向量，每个关键点可生成 4*4*8=128 维的 SIFT 描述子。

物体坐标系上的每一个整数点对应的图像坐标系可能不是整数，可采用最邻近插值，即图像坐标系上和它最接近的一个点：

$$\theta(x',y') = \theta(Round(x'), Round(y'))$$

或更精确地，可采用双线性插值：

$$\theta(x',y') = \theta(x,y)*dx2*dy2 + \theta(x+1,y)*dx1*dy2 + \theta(x,y+1)*dx2*dy1 + \theta(x+1,y+1)*dx1*dy1$$

双线性插值的意义在于，周围的四个点的值都对目标点有贡献，贡献大小与距离成正比。

最后对 128 维 SIFT 描述子 f0 归一化得到最终的结果：$f = f_0 * \frac{1}{|f_0|}$

下图为一个 128 维的 SIFT 描述子示例：

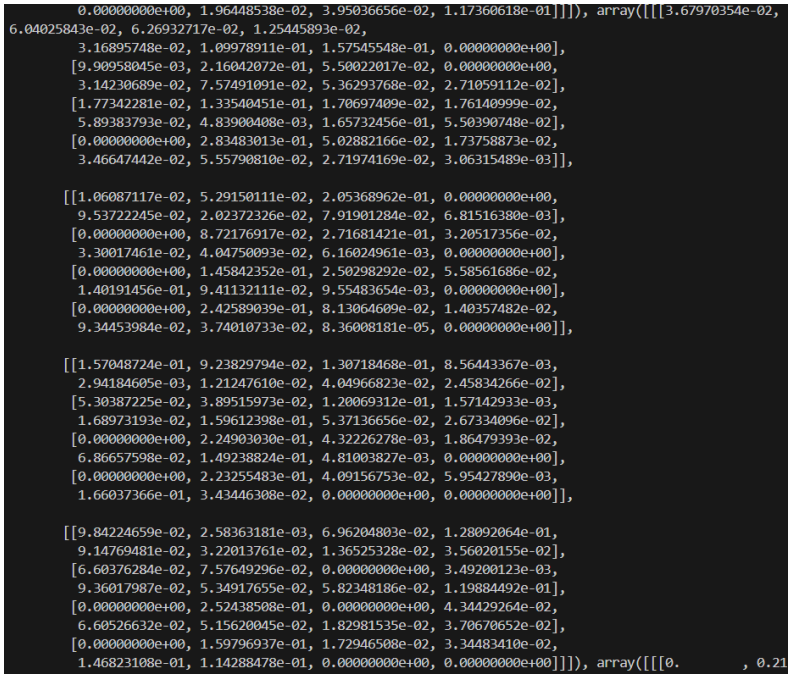
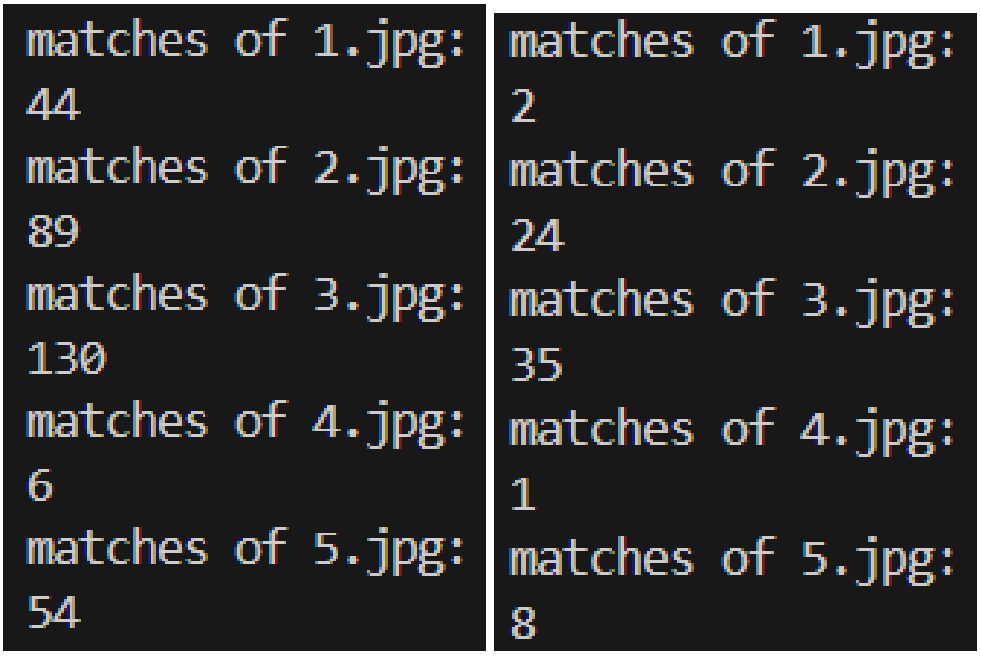


图 3: SIFT 描述子示例

两个 SIFT 描述子 f1 和 f2 之间的相似度可表示为：$s(f_1, f_2) = f_1 \cdot f_2$，即两个描述子的内积。这里我们选取一个阈值 T，当内积大于 T 时，认为两个描述子是匹配的，即两个关键点是匹配的，这对关键点最终会在图上画出匹配的连线。随着不同的阈值 T，能发生匹配的关键点数也会不同。

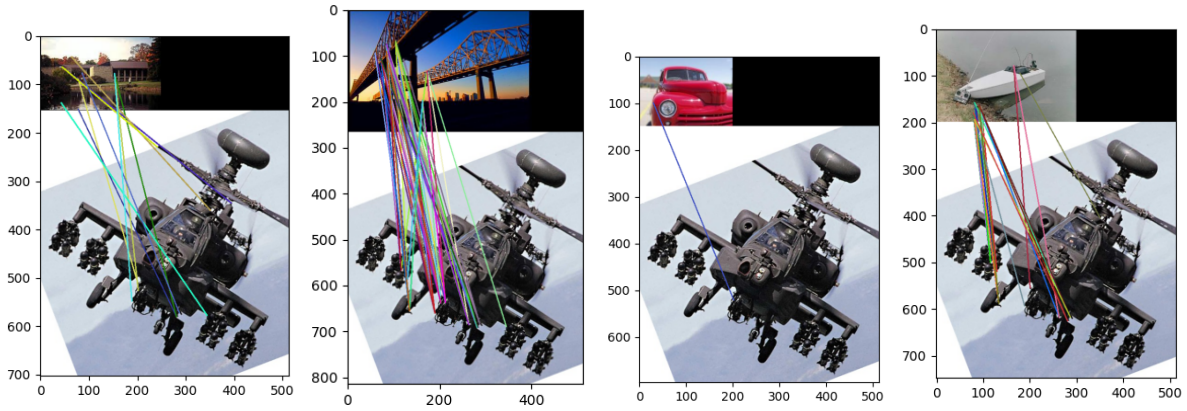
如下，左图中展示的是阈值为 0.5 时的匹配结果，右图中展示的是阈值为 0.6 时的匹配结果：



(a) 阈值为 0.5　　(b) 阈值为 0.6

图 4: 匹配结果

这里为了美观，我们选择了阈值为 0.55 的匹配结果，实际情况的阈值应调整得稍比 0.55 高一些为好。下图为 dataset 文件夹中的所有示例图片与 target 的匹配结果：



(a) 匹配结果 1　(b) 匹配结果 2　(c) 匹配结果 4　(d) 匹配结果 5

图 5: 手动实现的 SIFT 算法与示例图片的匹配结果

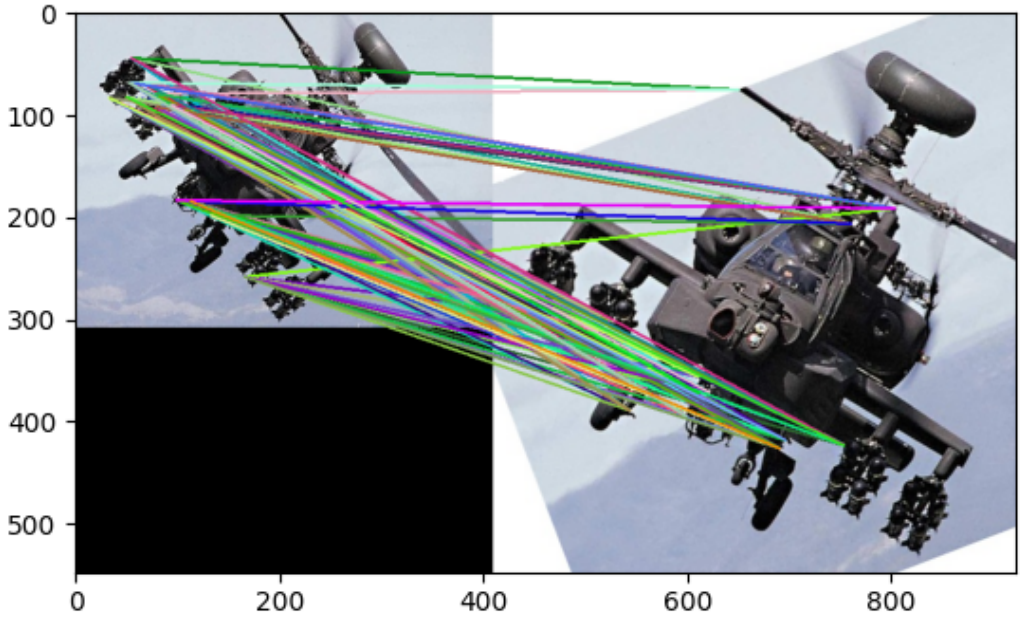


图 6: 与原图的匹配结果

可以看出，我们手动实现的 SIFT 算法在匹配上的效果还是不错的，能够较好地匹配出目标物体。其中示例 2 的匹配程度在四张示例图片中匹配程度较高，笔者猜测可能是因为示例 2 和原图的边缘较为相似，转角较为尖锐。原图的匹配线中大部分是平行线，存在的交叉线即为误匹配，说明我们手动实现的 SIFT 算法还有提升空间。

4.2 练习二：OpenCV 中的 SIFT 算法

为了实现 OpenCV 中的 SIFT 算法，我们主要需要以下几个函数：

1. `cv2.SIFT_create()`：OpenCV 库中的一个函数，用于创建一个 SIFT 检测器对象。调用 `cv2.SIFT_create()` 函数后，会返回一个 SIFT 对象。这个对象可以用来检测图像中的关

键点和计算这些关键点的描述。

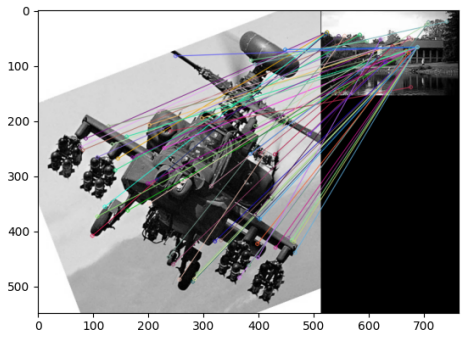
2. `sift.detectAndCompute()`：此函数的第一个参数是输入图像，第二个参数是一个掩码，在这里被设置为 None，表示不使用掩码。掩码通常用于指定图像的某些区域，以便只在这些区域内检测关键点。这个函数返回两个值：关键点和描述子。

3. `cv2.BFMatcher()`：此函数创建一个暴力匹配器。暴力匹配器是一种用于特征匹配的算法，它通过计算每个特征描述符之间的距离找到最佳匹配。`cv2.BFMatcher` 函数可以接受两个可选参数：`normType` 和 `crossCheck`。前者指定用于计算距离的度量标准，默认值为 `cv2.NORM_L2`，表示使用欧氏距离。`crossCheck` 是一个布尔值，默认值为 `False`，如果设置为 `True`，则只有当两个特征互为最佳匹配时才会被认为是匹配的。这里我们选择 `False`。

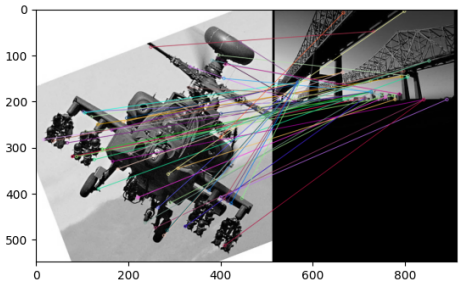
4. `bf.knnMatch()`：它在两个描述符集合之间找到每个描述符的 K 个最佳匹配。`knnMatch` 方法的第一个参数是查询描述符，第二个参数是训练描述符，第三个参数 `k` 指定了要找到的近邻数量。我们将设置为 2，表示每个查询描述符将找到两个最佳匹配。`knnMatch` 方法返回一个包含匹配结果的列表，其中每个元素都是一个包含 K 个匹配项的列表。每个匹配项是一个 `DMatch` 对象，包含匹配描述符的索引和距离等信息。

5. `cv2.drawMatches()`：用于在两幅图像之间绘制特征匹配结果，它返回一个图像，其中包含了两幅输入图像，并在它们之间绘制了匹配的特征点。

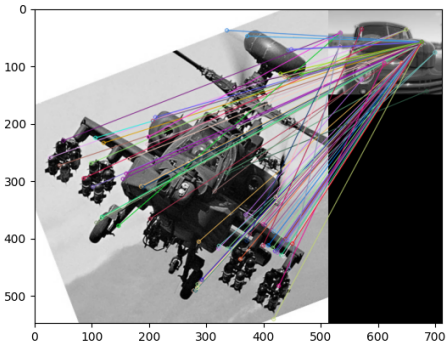
下面是使用 OpenCV 中的 SIFT 算法进行匹配的结果：



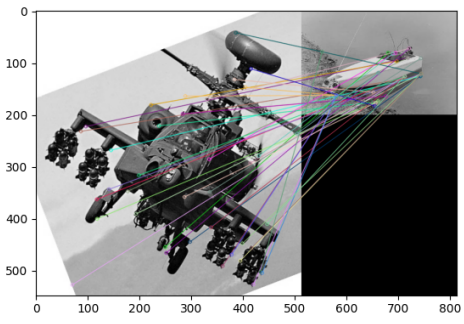
(a) 匹配结果 1



(b) 匹配结果 2



(c) 匹配结果 4



(d) 匹配结果 5

图 7: 与示例图片的匹配结果

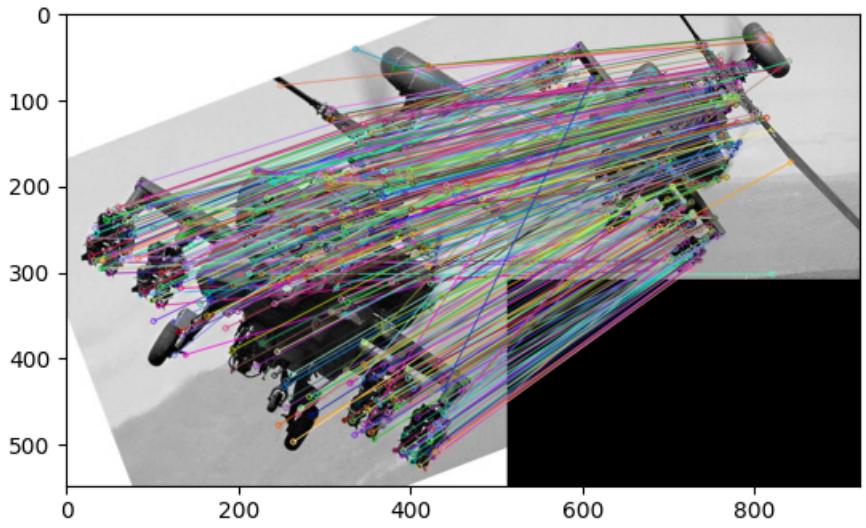


图 8: 与原图的匹配结果

5 实验心得与体会

本次实验主要是对 SIFT 算法的实现进行了学习与实践，通过手动实现 SIFT 算法，对 SIFT 算法的原理有了更深入的理解。

其中手动实现 SIFT 算法的过程中遇到了很多问题，比如将原图通过与高斯核卷积得到高斯金字塔，为了让金字塔的尺度具有连续性，需要对图像做高斯平滑和降采样。然后将高斯金字塔的每两层相减得到 DoG 金字塔，再在 DoG 金字塔中检测极值点，将其作为关键点。这些步骤十分繁琐，并且仅通过周围像素点极值大小提取关键点会导致关键点过多，需要设置阈值来筛选关键点。

在使用 Harris 角点提取时，需要设置合适的阈值来筛选关键点，否则会检测到大量的角点，一部分角点并不是我们所需要的关键点。笔者在未设置阈值时，因为检测到关键点的数量过多，代码跑了十分钟一张图片的匹配都跑不出来，后来将阈值设置为 0.01 后，检测到的关键点数量减少到了合理的数量。阈值若是太大，检测到的关键点又会太少，导致最终能匹配成功的关键点数量过少。

提取了关键点之后，笔者发现很多关键点都堆积在一起，即两个关键点之间的距离过近，于是加入了最小距离参数，使得关键点之间的距离大于一定值时才被认为是两个不同的关键点。

6 源代码

6.1 手动实现 SIFT 算法（exercise1.py）

```
import cv2
from matplotlib import pyplot as plt
import numpy as np
import math
```

```
    from math import pi
    import os

    # 更改工作目录到脚本所在目录
    os.chdir(os.path.dirname(os.path.abspath(__file__)))

    # Harris角点检测
    def detect_harris_corners(image, block_size=2, ksize=3, k=0.04, threshold=0.001):
        gray = np.float32(image)
        dst = cv2.cornerHarris(gray, block_size, ksize, k)
        dst = cv2.dilate(dst, None)
        corners = np.where(dst > threshold * dst.max())
        return list(zip(corners[1], corners[0])), dst

    # 关键点检测
    def get_kp(img, max_corners=100, threshold=0.01, min_distance=10):
        corners, dst = detect_harris_corners(img, threshold=threshold)
        kp = np.zeros((len(corners), 2), dtype=np.int16)
        for i, corner in enumerate(corners):
            kp[i][0] = corner[0]
            kp[i][1] = corner[1]

        # 非极大值抑制
        keypoints = []
        for y, x in corners:
            if 0 <= y < dst.shape[0] and 0 <= x < dst.shape[1]:  # 边界检查
                local_max = np.max(dst[max(0, y-1):min(y+2, dst.shape[0]), max(0, x-1):min(x+2, dst.
                    shape[1])])
                if dst[y, x] == local_max:
                    keypoints.append((x, y))

        # 按响应值排序并限制关键点数量
        keypoints = sorted(keypoints, key=lambda pt: dst[pt[1], pt[0]], reverse=True)
        keypoints = keypoints[:max_corners]

        # 应用最小距离参数
        final_keypoints = []
        for i, (x, y) in enumerate(keypoints):
            if all(np.linalg.norm(np.array([x, y]) - np.array([kx, ky])) >= min_distance for kx, ky
                in final_keypoints):
                final_keypoints.append((x, y))

        kp = np.array(final_keypoints, dtype=np.int16)
        return kp

    # 计算梯度幅值
    def get_amp(img):
        height, wide = img.shape
        img_ca = np.array(img, dtype=float)
        amply = np.zeros((height, wide))
        for i in range(1, height-1):
            for j in range(1, wide-1):
                amply[i][j] = ((img_ca[i][j+1] - img_ca[i][j-1])**2 + (img_ca[i+1][j] - img_ca[i-1][
                    j])**2)**0.5
        return amply

    # 计算梯度方向
    def get_the(img):
        height, wide = img.shape
```

```
        img_ca = np.array(img, dtype=float)
        theta = np.zeros((height, wide))
        for i in range(1, height-1):
            for j in range(1, wide-1):
                theta[i][j] = math.atan2(img_ca[i][j+1] - img_ca[i][j-1], img_ca[i+1][j] - img_ca[i
                    -1][j])
        return theta

    # 计算主方向
    def main_direct(amp, the, img_gray, sigma):
        dir = []
        r = int(4.5 * sigma)
        h, w = img_gray.shape
        kp = get_kp(img_gray)
        for k in kp:
            x, y = k
            vote = [0 for i in range(36)]
            for i in range(max(0, y-r), min(y+r+1, h)):
                for j in range(max(0, x-r), min(x+r+1, w)):
                    index = int((the[i][j] + pi) * 18.0 / pi - 1)
                    vote[index] += amp[i][j]
            t = 0
            for i in range(len(vote)):
                if vote[i] > vote[t]:
                    t = i
            dir.append(t * pi / 18 + pi / 36 - pi)
        return dir

    # 计算描述子
    def get_xy_after_spinning(x, y, th):
        x1 = math.cos(th) * x - math.sin(th) * y
        y1 = math.sin(th) * x + math.cos(th) * y
        return [x1, y1]
    def get_xy2(x1, y1, sigma):
        return [(x1 / (3 * sigma)) + 2, (y1 / (3 * sigma)) + 2]
    def get_descriptors(amp, the, img_gray):
        descriptors = []
        h, w = img_gray.shape
        sigma = (h + w) / 690
        radius = int(sigma * 15 / (2**0.5))
        dir = main_direct(amp, the, img_gray, sigma)
        kp = get_kp(img_gray)
        for k in range(len(kp)):
            x0, y0 = kp[k]
            descriptor = np.zeros((4, 4, 8))
            for i in range(max(0, y0-radius), min(y0+radius+1, h)):
                for j in range(max(0, x0-radius), min(x0+radius+1, w)):
                    th = the[i][j]
                    th = int((th + pi - 0.000001) // (pi / 4))
                    x = j - x0
                    y = i - y0
                    trans1 = get_xy_after_spinning(x, y, dir[k])
                    trans2 = get_xy2(trans1[0], trans1[1], sigma)
                    x2 = trans2[0] - 0.5
                    y2 = trans2[1] - 0.5
                    index_x = math.floor(x2)
                    index_y = math.floor(y2)
                    if index_x >= 0 and index_y >= 0 and index_x < 4 and index_y < 4:
                        descriptor[index_x][index_y][th] += (index_x + 1 - x2) * (index_y + 1 - y2)
```

```
                    if index_x >= 0 and index_y < 3 and index_x < 4 and index_y >= -1:
                        descriptor[index_x][index_y+1][th] += (index_x + 1 - x2) * (y2 - index_y)
                    if index_x < 3 and index_y >= 0 and index_x >= -1 and index_y < 4:
                        descriptor[index_x+1][index_y][th] += (x2 - index_x) * (index_y + 1 - y2)
                    if index_x < 3 and index_y < 3 and index_x >= -1 and index_y >= -1:
                        descriptor[index_x+1][index_y+1][th] += (x2 - index_x) * (y2 - index_y)
            ret = np.linalg.norm(descriptor, axis=None)
            if ret != 0:
                descriptor = descriptor * (1 / ret)
            descriptors.append(descriptor)
        return descriptors

    # 图像匹配
    def match(des1, des2):
        num = 0
        mat = []
        mat_n = []
        mat_index = []
        for i in range(len(des1)):
            for j in range(len(des2)):
                mat_n.append(np.sum(des1[i] * des2[j]))
                mat_index.append([i, j])
                if np.sum(des1[i] * des2[j]) > 0.5:
                    num += 1
        print(num)
        arr = np.array(mat_n)
        idx = arr.argsort()[-num:][::-1]
        for i in range(num):
            mat.append(mat_index[idx[i]])
        return [num > 5, mat]

    # sift算法
    def sift(img):
        kp = get_kp(img)
        amply = get_amp(img)
        theta = get_the(img)
        des = get_descriptors(amply, theta, img)
        return [kp, des]

    target = cv2.imread('../target.jpg')
    tar_gray = cv2.cvtColor(target, cv2.COLOR_BGR2GRAY)
    tar_gray = cv2.GaussianBlur(tar_gray, (3, 3), 0.6)
    keytar, destar = sift(tar_gray)

    # 两张图竖着放
    for i in range(1, 6):
        img = cv2.imread('../dataset/%d.jpg' % i)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        gauss = cv2.GaussianBlur(gray, (3, 3), 0.6)
        key, des = sift(gauss)
        print('matches of %d.jpg:' % i)
        flag, mat = match(des, destar)
        w1, h1 = gauss.shape
        w2, h2 = tar_gray.shape
        newimg = np.zeros((w1 + w2, max(h1, h2), 3), dtype=np.int16)
        for m in range(w1):
            for n in range(h1):
                for s in range(3):
                    newimg[m][n][s] = img[m][n][2-s]
```

```
        for m in range(w2):
            for n in range(h2):
                for s in range(3):
                    newimg[w1 + m][n][s] = target[m][n][2-s]
        for j in range(len(mat)):
            kp0 = key[mat[j][0]]
            kp1 = []
            kp1.append(keytar[mat[j][1]][0])
            kp1.append(keytar[mat[j][1]][1] + w1)
            bgr = np.random.randint(0, 255, 3, dtype=np.int32)
            cv2.line(newimg, kp0, kp1, (int(bgr[0]), int(bgr[1]), int(bgr[2])), 2)

        plt.imshow(newimg)
        plt.show()


    # # 两张图横着放
    # for i in range(1, 6):
    #     img = cv2.imread('E:/ICE2607/lab3-SIFT/dataset/%d.jpg' % i)
    #     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #     gauss = cv2.GaussianBlur(gray, (3, 3), 0.6)
    #     key, des = sift(gauss)
    #     print('matches of %d.jpg:' % i)
    #     flag, mat = match(des, destar)
    #     w1, h1 = gauss.shape
    #     w2, h2 = tar_gray.shape
    #     newimg = np.zeros((max(w1, w2), h1 + h2, 3), dtype=np.int16)
    #     for m in range(w1):
    #         for n in range(h1):
    #             for s in range(3):
    #                 newimg[m][n][s] = img[m][n][2-s]
    #     for m in range(w2):
    #         for n in range(h2):
    #             for s in range(3):
    #                 newimg[m][h1 + n][s] = target[m][n][2-s]
    #     for j in range(len(mat)):
    #         kp0 = key[mat[j][0]]
    #         kp1 = []
    #         kp1.append(keytar[mat[j][1]][0] + h1)
    #         kp1.append(keytar[mat[j][1]][1])
    #         bgr = np.random.randint(0, 255, 3, dtype=np.int32)
    #         cv2.line(newimg, kp0, kp1, (int(bgr[0]), int(bgr[1]), int(bgr[2])), 2)

    #     plt.imshow(newimg)
    #     plt.show()
```

6.2 OpenCV 中的 SIFT 算法（exercise2.py）

```
import cv2
import matplotlib.pyplot as plt

# 读取图像并转换为灰度图
img1 = cv2.imread('../target.jpg')
img2 = cv2.imread('../dataset/1.jpg')

# 转换为灰度图
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
```

```
11
12      # 创建 SIFT 对象
13      sift = cv2.SIFT_create()
14
15      # 检测关键点和计算描述子
16      keypoints_1, descriptors_1 = sift.detectAndCompute(img1, None)
17      keypoints_2, descriptors_2 = sift.detectAndCompute(img2, None)
18
19      # 创建 BFMatcher 对象
20      bf = cv2.BFMatcher()
21
22      # 使用 knnMatch 进行匹配
23      matches = bf.knnMatch(descriptors_1, descriptors_2, k=2)
24
25      # 应用比率测试来过滤错误匹配
26      good_matches = []
27      for m, n in matches:
28          if m.distance < 0.75 * n.distance:
29              good_matches.append(m)
30
31      # 绘制匹配结果
32      img3 = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2, good_matches, None, flags=cv2.
            DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
33
34      # 显示结果
35      plt.imshow(img3)
36      plt.show()
```