实验 2　SIFT

姓名　学号　班级

日期

目录

1 实验概览

1.1 实验内容

SIFT，即 **S**cale-**I**nvariant **F**eature **T**ransform（尺度不变特征变换），由 Lowe 于 1999 发明。本次实验要求实现 Distinctive Image Features from Scale-Invariant Keypoints (Lowe, 2004) [2] 论文中 SIFT 描述子的计算，将 5 张图片分别与目标图片进行匹配，并与 OpenCV 库自带的 SIFT 函数输出结果进行比较。

1.2 实验原理

本次实验要求实现的内容相比原 SIFT 算法有所简化，大致包含如下几步：

1. 选取关键点；
2. 确定关键点主方向；
3. 计算关键点描述子；
4. 关键点匹配。

1.2.1 选取关键点

首先构造图像金字塔，第 $i+1$ 层（octave）图像大小为第 i 层的一半。在每一层上使用 Harris 角点提取函数 `cv2.goodFeaturesToTrack(useHarris=True)`。

1.2.2 确定关键点主方向

将图像转化为灰度图像 $L(x,y)$, 对于每个像素, 可以计算梯度幅值 $m(x,y)$ 及梯度方向 $\theta(x,y)$。

$$m(x,y)=\sqrt{\left(L(x+1,y)-L(x-1,y)\right)^2+\left(L(x,y+1)-L(x,y-1)\right)^2}.$$

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

将 $\theta\in(-\pi,\pi]$ 分为 36 个区间，每个区间 10°。关键点附近的像素按照其 θ 值落在每个区间，权重为 $m(x,y)$ 乘以高斯二维分布函数，即：

$$w_i(x_0,y_0)=\sum_{\theta\in i\text{-th bin}} m(x,y)\cdot\exp\left(-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}\right). \tag{1}$$

取总和最大的区间中点作为该关键点的主方向，即：

$$\theta_0(x_0,y_0)=\text{the middle point of bin No. }\arg\max_{i\in[0,36)} w_i(x_0,y_0).$$

1.2.3 计算关键点描述子

将图像坐标系转换为物体坐标系：

$$\theta'(x,y) = \theta(x,y) - \theta_0(x_0,y_0).$$

在物体坐标系下，整点的 m 和 θ 值会映射到图像坐标系上的非整点，使用双线性插值法进行近似取值。

$$\begin{aligned} f'(x,y) &= f(x',y') \\ &\approx f(\lfloor x' \rfloor, \lfloor y' \rfloor) \cdot (\lfloor x' \rfloor + 1 - x') \cdot (\lfloor y' \rfloor + 1 - y') \\ &+ f(\lfloor x' \rfloor + 1, \lfloor y' \rfloor) \cdot (x' - \lfloor x' \rfloor) \cdot (\lfloor y' \rfloor + 1 - y') \\ &+ f(\lfloor x' \rfloor, \lfloor y' \rfloor + 1) \cdot (\lfloor x' \rfloor + 1 - x') \cdot (y' - \lfloor y' \rfloor) \\ &+ f(\lfloor x' \rfloor + 1, \lfloor y' \rfloor + 1) \cdot (x' - \lfloor x' \rfloor) \cdot (y' - \lfloor y' \rfloor). \end{aligned}$$

对于每个关键点，以其为中心取 4×4 大小的单元，其中每个单元大小为 4×4 像素。对于每个单元，将 θ' 分为 8 个区间，每个区间 $45°$，单元内的每个像素按照其 θ' 值落在每个区间，权重为 $m'(x,y)$ 乘以高斯二维分布函数，即：

$$\begin{aligned} \text{Descriptor}_{(x_0,y_0)}(i,j,k) = &\sum_{(x',y')} m(x',y') \cdot \exp\left(-\frac{(x'-x_0)^2 + (y'-y_0)^2}{2\sigma^2}\right) \\ &\text{s.t. } \theta'(x',y') \in k\text{-th bin} \\ &\text{and } (x',y') \in \text{subregion } (i,j) \text{ around keypoint } (x_0,y_0). \end{aligned} \tag{2}$$

故对于每个关键点，形成了其描述子：大小 4×4 的矩阵，每个元素为 8 维向量，表示每个方向区间上的权重和，共 128 维。

1.2.4 关键点匹配

使用 OpenCV 自带暴力匹配 `cv2.BFMatcher`，按描述子向量距离进行匹配。

1.2.5 总结

一些优化的细节在本章节中略去了，将在 解决思路 章节中予以说明。

2 解决思路

2.1 选取关键点

首先基于原图像构造图像金字塔，并转换为灰度图像，最小层图像大小不小于 32×32。

```
self.octaves.append(Octave(base_img))
num_octaves = int(round(math.log(min(base_img.shape)) / math.log(2) - 4))
for i in range(num_octaves - 1):
    last_img = self.octaves[i].img
    img = cv2.resize(last_img,
        (int(last_img.shape[1] / 2), int(last_img.shape[0] / 2)))
    self.octaves.append(Octave(img))
```

图 1: 灰度图像金字塔[1]

在每层图像上运行 Harris 角点检测，可以得到不同尺度的角点。

图 2: Harris 角点检测逐层运行结果

[1]为保证展示效果，只展示了前三层图像，实际上一共有五层。

2.2 确定关键点主方向

对于每个角点，在其对应的层上，按 (1) 计算各角度区间的权重 $w_i(x_0, y_0)$.

我们取其所有极大值 w_{i^*}。其中 w_{i^*} 为极大值当且仅当（我们可以将 w 的下标视作循环的）：

$$w_{i^*} > w_{i^*-1}, \qquad w_{i^*} > w_{i^*+1}.$$

> The highest peak in the histogram is detected, and then any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation.[2]

根据论文原意，如果 $w_{i^*} > 0.8 \cdot \max w_i$，则保留这个角度区间 i^*。

为获得更精确的主方向，使用二次插值法对 i^* 及其相邻的点进行插值，即：

$$i^\dagger = i^* + \frac{w_- - w_+}{2(w_- - 2w_0 + w_+)}.$$

其中，$w_- = w_{i^*-1}, w_0 = w_{i^*}, w_+ = w_{i^*+1}$。

```
bin_max = max(bins)
peaks = np.where(
    np.logical_and(
        bins > np.roll(bins, 1),
        bins > np.roll(bins, -1))
    )[0]
for peak_index in peaks:
    val = bins[peak_index]
    if val >= 0.8 * bin_max and val < bin_max:
        val_left = bins[(peak_index - 1) % 36]
        val_right = bins[(peak_index + 1) % 36]
        intp_peak_index = (peak_index + \
            0.5 * (val_left - val_right) / (val_left - 2 * val + val_right)) % 36
        more_kps.append(cv2.KeyPoint(*p.pt, p.size, int(intp_peak_index * 10 + 5)))

max_bin_id = np.argmax(bins)
p.angle = int(max_bin_id * 10 + 5)
```

我们可以将所有关键点及其主方向、尺度绘制在一张图上。

[2] Chapter 5, Distinctive Image Features from Scale-Invariant Keypoints (Lowe, 2004).

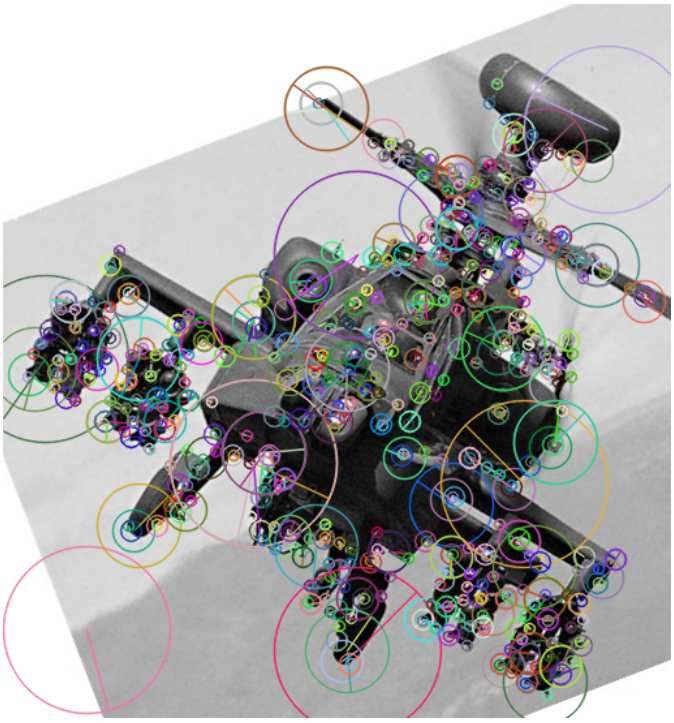


图 3: 关键点结果

2.3 计算关键点描述子

按 (2) 的方式计算关键点描述子。

> Therefore, trilinear interpolation is used to distribute the value of each gradient sample into adjacent histogram bins.[3]

对 $4 \times 4 \times 8$ 维向量采用三线性插值。

最后，对向量进行归一化处理，将 0.2 以上的值调整为 0.2，并再次归一化。这一步骤使描述子对亮度变化、大梯度均不再敏感。

2.4 关键点匹配

> For our object recognition implementation, we reject all matches in which the distance ratio is greater than 0.8, which eliminates 90% of the false matches while discarding less than 5% of the correct matches.[4]

[3] Chapter 6, Distinctive Image Features from Scale-Invariant Keypoints (Lowe, 2004).

[4] Chapter 7, Distinctive Image Features from Scale-Invariant Keypoints (Lowe, 2004).

对于每个描述子，使用比值测试（Ratio Test）进行过滤。我们使用暴力匹配获取距离最近的两个描述子，当第二近的距离与最近距离的比值小于 0.7（原文为 0.8，我们使用 OpenCV 调整后的参数）时才保留这一匹配。

```
kp_target, des_target = sift.detectAndCompute(target, None)
bf = cv2.BFMatcher()
result = []
for img in dataset:
    kp, des = sift.detectAndCompute(img, None)
    matches = bf.knnMatch(des_target, des, k=2)
    good_matches = [m for m, n in matches if m.distance < 0.7 * n.distance]
```

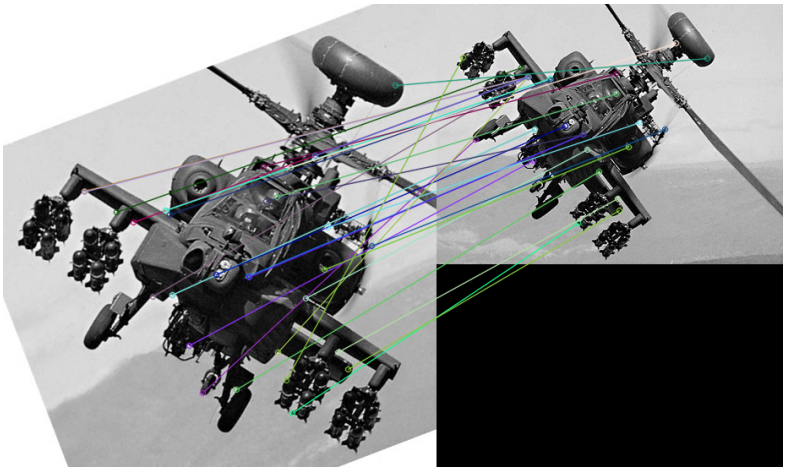
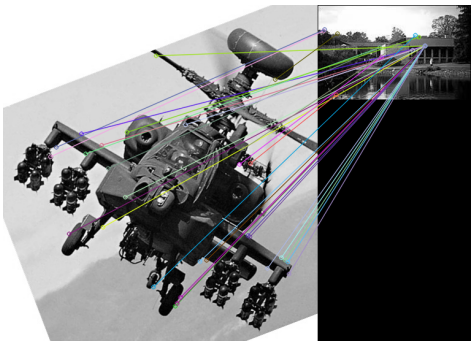


图 4: 匹配结果

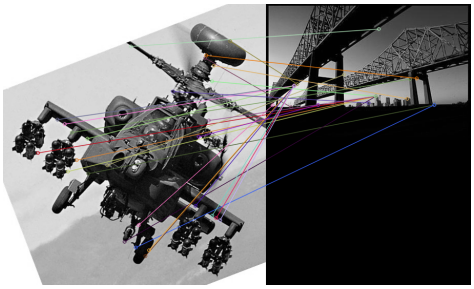
可以看出，我们实现的 Harris 和 SIFT 匹配的关键点数较少，但质量较高。

3 运行结果

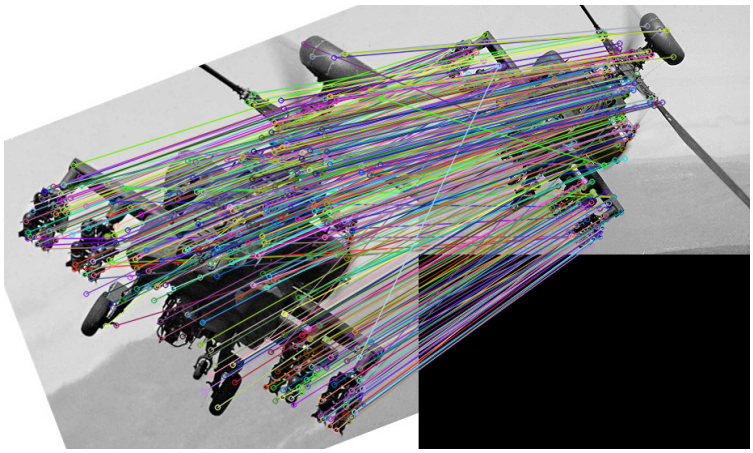
1. OpenCV 自带函数



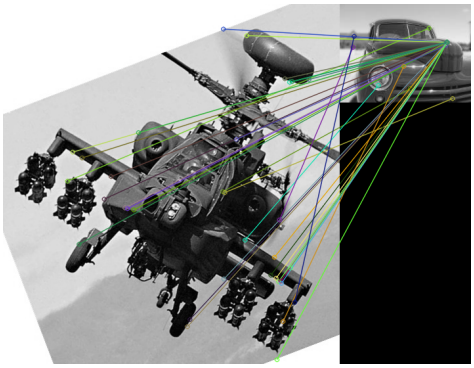
(a) 图 1



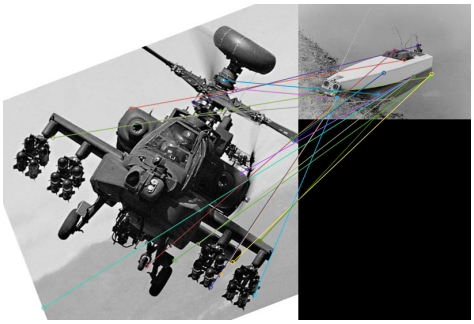
(b) 图 2



(c) 图 3



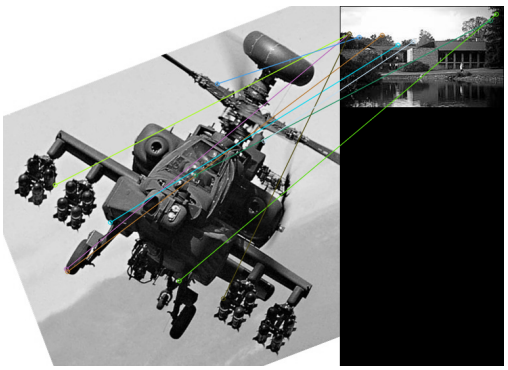
(d) 图 4



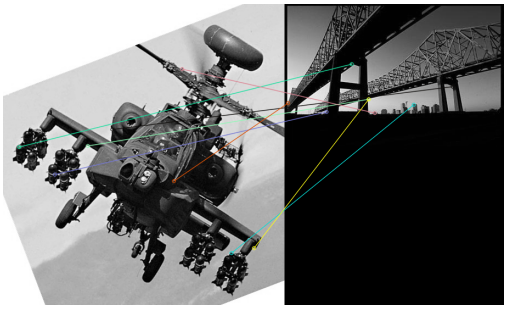
(e) 图 5

图 5: OpenCV 自带函数运行结果

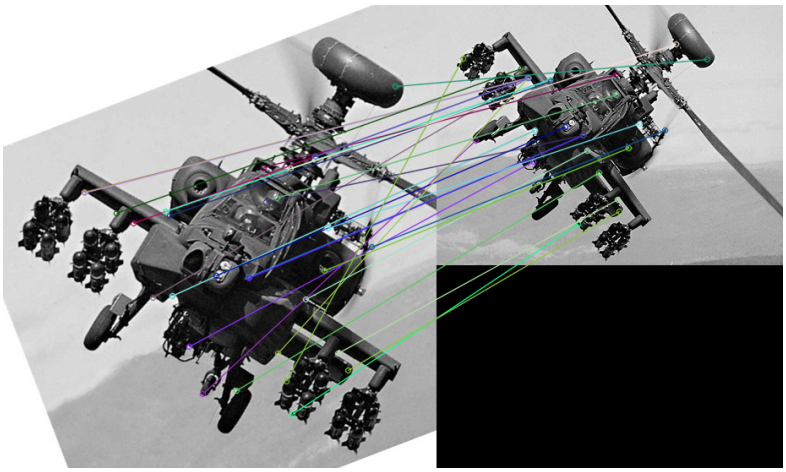
2. 自行实现



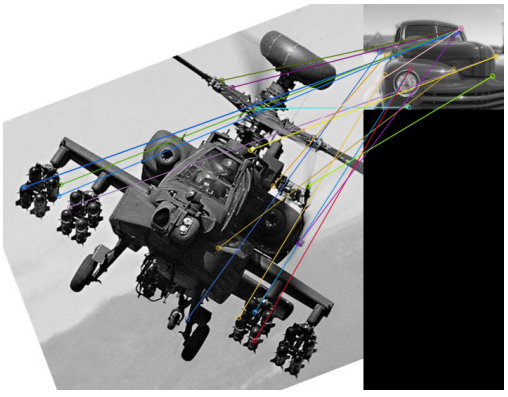
(a) 图 1



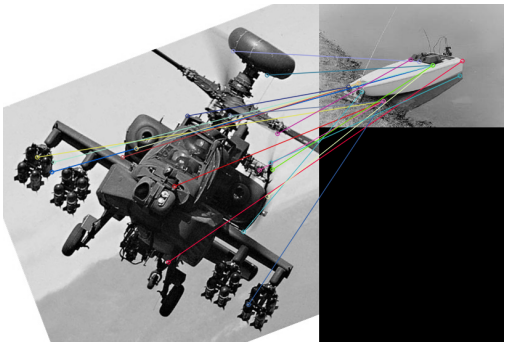
(b) 图 2



(c) 图 3



(d) 图 4



(e) 图 5

图 6: 自行实现运行结果

4 结果分析与思考

从结果来看，我们实现的 Harris 版本的 SIFT 算法相比 OpenCV 自带的 SIFT 算法匹配点数少很多，正确率也较低。我认为可能有以下几点原因：

1. 关键点检测方面，完整版 SIFT 使用更精巧的算法来检测关键点（Interest Points），同时，还对关键点的位置在尺度空间上进行亚像素的调整。而我们实现的 SIFT 只是在图像金字塔各层上运行 Harris 角点提取函数，不仅数量上更少，且特征上不够稳定，还容易因图像压缩导致关键点位置的偏移，进而影响后续计算。

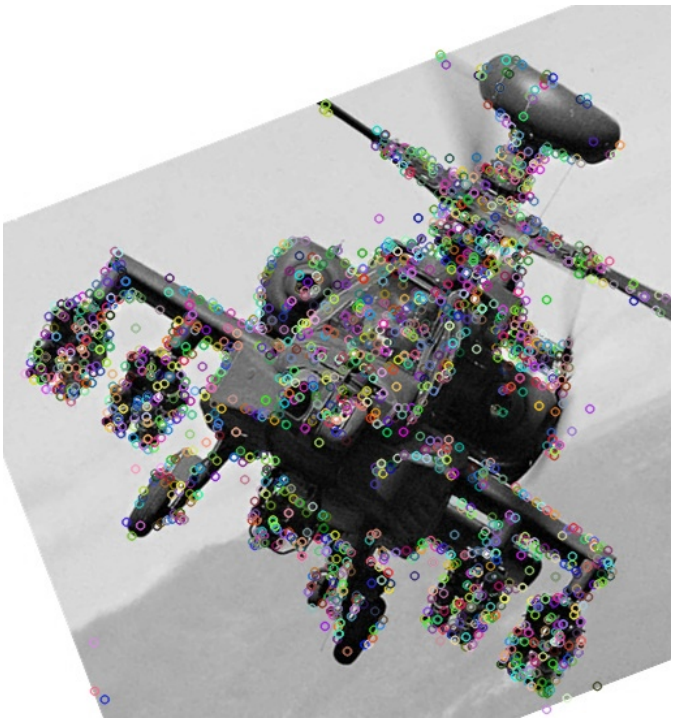


图 7: OpenCV SIFT 关键点检测结果

2. 主方向和描述子计算方面，原版 SIFT 中同一层上每个关键点的大小与梯度计算过程有关，而我们实现的版本同一层上使用统一的大小。

3. 其他优化方面，原版 SIFT 在边界效应处理等方面做的比较完善，而我们实现的版本在此方面较为欠缺。

5 实验感想

SIFT 算法中包含了大量的参数，对匹配结果会产生极为显著的影响。在原论文中一些参数是通过实验确定的，但仍不乏一些经验参数。我在实现 Harris 版 SIFT 的过程中同样遇到了类似的问题，很多参数都会对匹配结果产生很大影响，尤其是：

- Harris 参数 `minDistance`, `blockSize`, `k`
- 主方向及描述子计算半径比例参数

碍于时间精力有限，难以对这些参数进行严谨的取值，深感设计精确、严谨的算法之难度。

在完成本实验的过程中，我对 OpenCV 库、SIFT 算法等都有了更深的了解，收获颇丰。

A 参考资料

[1] Russell Islam. Rmislam/pythonsift: A clean and concise python implementation of sift (scale-invariant feature transform).

[2] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

B 源码

文件名	功能
sift.py	自行实现的 SIFT 描述子计算
lab2.py	主程序
debug.py	调试及报告图片打印
img_io.py	图片读入输出

B.1 sift.py

```python
import cv2
import numpy as np
import math
from typing import List
import logging
import sys
from functools import cmp_to_key
import debug

NUM_REGIONS = 4
REGION_SIZE = 4
NUM_BINS = 8

class Octave:
    def __init__(self, img: cv2.typing.MatLike):
        self.img = img
        self.mag = np.zeros(self.img.shape)
        self.angle = np.zeros(self.img.shape)
        self.compute_mag_and_angle()

    def compute_mag_and_angle(self):
        h, w = self.img.shape
        img = self.img.astype(np.float32)
        for i in range(1, h - 1):
            for j in range(1, w - 1):
                self.mag[i][j] = ((img[i + 1][j] - img[i - 1][j]) ** 2 + (img[i][j + 1] - img[i][j -
                ↪  1]) ** 2) ** 0.5
```

```
                self.angle[i][j] = (360 - math.atan2(img[i, j + 1] - img[i, j - 1], img[i + 1, j] -
                ↪  img[i - 1, j]) * 180 / math.pi) % 360

class SIFT:
    def __init__(self):
        self.octaves: List[Octave] = []

    def get_kp(self, img):
        corners = cv2.goodFeaturesToTrack(
            img,
            maxCorners=1000,
            qualityLevel=0.01,
            minDistance=10,
            blockSize=2,
            useHarrisDetector=True,
            k=0.01
        )
        if corners is None:
            return []
        return [cv2.KeyPoint(x=float(x), y=float(y), size=0) for [[x, y]] in corners]

    def gaussian_mask(self, sigma, dx, dy):
        return math.exp(-(dx * dx + dy * dy) / (2 * sigma * sigma))

    def compute_kp_orient(self, kp: List[cv2.KeyPoint]):
        more_kps = []
        for p in kp:
            octave = self.octaves[p.octave]
            img, mag, angle = octave.img, octave.mag, octave.angle
            h, w = img.shape
            scale = 2 ** -p.octave

            sigma = p.size * 1.5 * scale
            rad = int(sigma * 3)
            px, py = int(p.pt[0] * scale), int(p.pt[1] * scale)

            # fill bins by orientation
            bins = [0] * 36
            for i in range(max(1, py - rad), min(py + rad + 1, h - 1)):
                for j in range(max(1, px - rad), min(px + rad + 1, w - 1)):
                    m = mag[i][j]
                    a = angle[i][j]
                    bin_id = math.floor(a / 10)
                    bins[bin_id] += m * self.gaussian_mask(sigma, j - px, i - py)

            # find peaks and interpolate accurate orientation
            bin_max = max(bins)
            peaks = np.where(np.logical_and(bins > np.roll(bins, 1), bins > np.roll(bins, -1)))[0]
            for peak_index in peaks:
                val = bins[peak_index]
                if val >= 0.8 * bin_max and val < bin_max:
                    val_left = bins[(peak_index - 1) % 36]
                    val_right = bins[(peak_index + 1) % 36]
```

```
                    intp_peak_index = (peak_index + 0.5 * (val_left - val_right) / (val_left - 2 * val
                    ↪  + val_right)) % 36
                    more_kps.append(cv2.KeyPoint(*p.pt, p.size, int(intp_peak_index * 10 + 5)))

            max_bin_id = np.argmax(bins)
            p.angle = int(max_bin_id * 10 + 5)

            logging.debug(f"Keypoint ({p.pt[0]}, {p.pt[1]}) angle {p.angle} determined with radius
            ↪  {rad} and sigma {sigma}.")

        kp.extend(more_kps)

        logging.info(f'Assigned different orientations to {len(more_kps)} ({round(len(more_kps) /
        ↪  (len(kp) - len(more_kps)) * 100, 1)}%) keypoints.')

    def get_val_by_interpolation(self, mat: np.ndarray, x, y): # bilinear interpolation
        h, w = mat.shape
        if x < 0 or x >= w - 1 or y < 0 or y >= h - 1:
            return 0
        x0, y0 = int(x), int(y)
        x1, y1 = x0 + 1, y0 + 1
        dx, dy = x - x0, y - y0
        val = (
            mat[y0][x0] * (1 - dx) * (1 - dy)
            + mat[y0][x1] * dx * (1 - dy)
            + mat[y1][x0] * (1 - dx) * dy
            + mat[y1][x1] * dx * dy
        )
        return val

    def compare_kp(self, a: cv2.KeyPoint, b: cv2.KeyPoint):
        if a.pt[0] != b.pt[0]:
            return a.pt[0] - b.pt[0]
        if a.pt[1] != b.pt[1]:
            return a.pt[1] - b.pt[1]
        return b.size - a.size

    def remove_duplicate_kp(self, kp: List[cv2.KeyPoint]):
        if not len(kp):
            return []
        unique_kps = [kp[0]]
        kp.sort(key=cmp_to_key(self.compare_kp))
        i = 0
        for p in kp[1:]:
            if kp[i].pt == p.pt and kp[i].size > p.size:
                    continue
            i += 1
            unique_kps.append(p)

        logging.info(f'Removed {len(kp) - len(unique_kps)} duplicate keypoints.')

        return unique_kps

```

```
129     def get_des(self, kp: List[cv2.KeyPoint]):
130         des = []
131 
132         for p in kp:
133             desc = np.zeros((NUM_REGIONS + 2, NUM_REGIONS + 2, NUM_BINS)) # 2 was added so that we can
                ↪  deal with the borders more easily
134 
135             octave = self.octaves[p.octave]
136             scale = 2 ** -p.octave
137             img, mag, angle = octave.img, octave.mag, octave.angle
138 
139             sub_width = 3 * 0.5 * p.size * scale
140             radius = np.round(sub_width * (NUM_REGIONS + 1) * math.sqrt(2) * 0.5).astype(int)
141 
142             x, y = int(p.pt[0] * scale), int(p.pt[1] * scale)
143             orient = p.angle
144 
145             h, w = img.shape
146 
147             sin_orient = math.sin(-orient / 180 * math.pi)
148             cos_orient = math.cos(-orient / 180 * math.pi)
149             for yy in range(-radius, radius + 1):
150                 for xx in range(-radius, radius + 1):
151                     y_rot = xx * sin_orient + yy * cos_orient
152                     x_rot = xx * cos_orient - yy * sin_orient
153 
154                     bin_y = y_rot / sub_width + 1.5
155                     bin_x = x_rot / sub_width + 1.5
156 
157                     if bin_x <= -1 or bin_x >= 4 or bin_y <= -1 or bin_y >= 4:
158                         continue
159 
160                     y_img = y + yy
161                     x_img = x + xx
162 
163                     if x_img < 1 or x_img >= w - 1 or y_img < 1 or y_img >= h - 1:
164                         continue
165 
166                     m = self.get_val_by_interpolation(mag, x_img, y_img)
167                     a = self.get_val_by_interpolation(angle, x_img, y_img)
168 
169                     weight = m * self.gaussian_mask(0.75, y_rot / sub_width, x_rot / sub_width) # 3 *
                        ↪  sigma = 2.25
170 
171                     bin_angle = ((a - p.angle) / 360 * NUM_BINS) % NUM_BINS
172 
173                     # Reference: https://github.com/rmislam/PythonSIFT
174                     bin_y_floor, bin_x_floor, bin_angle_floor = np.floor([bin_y, bin_x,
                        ↪  bin_angle]).astype(int)
175                     row_fraction, col_fraction, orientation_fraction = bin_y - bin_y_floor, bin_x -
                        ↪  bin_x_floor, bin_angle - bin_angle_floor
176                     bin_angle_floor = bin_angle_floor % NUM_BINS
177 
```

```
                    c1 = weight * row_fraction
                    c0 = weight * (1 - row_fraction)
                    c11 = c1 * col_fraction
                    c10 = c1 * (1 - col_fraction)
                    c01 = c0 * col_fraction
                    c00 = c0 * (1 - col_fraction)
                    c111 = c11 * orientation_fraction
                    c110 = c11 * (1 - orientation_fraction)
                    c101 = c10 * orientation_fraction
                    c100 = c10 * (1 - orientation_fraction)
                    c011 = c01 * orientation_fraction
                    c010 = c01 * (1 - orientation_fraction)
                    c001 = c00 * orientation_fraction
                    c000 = c00 * (1 - orientation_fraction)

                    desc[bin_y_floor + 1, bin_x_floor + 1, bin_angle_floor] += c000
                    desc[bin_y_floor + 1, bin_x_floor + 1, (bin_angle_floor + 1) % 8] += c001
                    desc[bin_y_floor + 1, bin_x_floor + 2, bin_angle_floor] += c010
                    desc[bin_y_floor + 1, bin_x_floor + 2, (bin_angle_floor + 1) % 8] += c011
                    desc[bin_y_floor + 2, bin_x_floor + 1, bin_angle_floor] += c100
                    desc[bin_y_floor + 2, bin_x_floor + 1, (bin_angle_floor + 1) % 8] += c101
                    desc[bin_y_floor + 2, bin_x_floor + 2, bin_angle_floor] += c110
                    desc[bin_y_floor + 2, bin_x_floor + 2, (bin_angle_floor + 1) % 8] += c111

            desc = desc[1:-1, 1:-1, :]
            desc = desc.flatten()
            threshold = 0.2 * np.linalg.norm(desc)
            desc[desc > threshold] = threshold
            desc /= (np.linalg.norm(desc) + 1e-8)

            des.append(desc)

        return np.array(des, dtype=np.float32)

    # main function
    def detectAndCompute(self, base_img: cv2.typing.MatLike, mask: None = None):
        self.__init__()
        logging.basicConfig(handlers=[logging.StreamHandler(sys.stdout)], level=logging.INFO)

        PRINT = base_img.shape[0] == 548

        self.octaves.append(Octave(base_img))
        logging.info(f'Initialized SIFT class with base image of size {base_img.shape}.')

        num_octaves = int(round(math.log(min(base_img.shape)) / math.log(2) - 4))
        for i in range(num_octaves - 1):
            last_img = self.octaves[i].img
            img = cv2.resize(last_img, (int(last_img.shape[1] / 2), int(last_img.shape[0] / 2))) # add
            ↪  Gaussian blur 1.6?
            self.octaves.append(Octave(img))
            if PRINT:
                debug.save_doc_image(img, f"octave-{i+1}")
            logging.info(f'Added octave No.{i + 1} from image of size {img.shape}.')
```

```
230
231         all_kps = []
232
233         for octave_index, octave in enumerate(self.octaves):
234             img = octave.img
235             kps: List[cv2.KeyPoint] = self.get_kp(img)
236             if PRINT:
237                 debug.save_doc_image(cv2.drawKeypoints(img, kps, img, flags=2),
                    ↪  f"keypoint-octave-{octave_index}")
238             logging.info(f'Detected {len(kps)} keypoints from octave No.{octave_index}.')
239             for kp in kps:
240                 kp.size = 2 ** (octave_index + 1)
241                 kp.pt = tuple(np.array(kp.pt) * (2 ** (octave_index)))
242                 kp.octave = octave_index
243             all_kps.extend(kps)
244
245         self.compute_kp_orient(all_kps)
246         all_kps = self.remove_duplicate_kp(all_kps)
247
248         if PRINT:
249             print_kps = all_kps.copy()
250             for p in print_kps:
251                 p.size *= 4
252             debug.save_doc_image(cv2.drawKeypoints(base_img, print_kps, base_img, flags=4),
                ↪  f"keypoint-all")
253
254         des = self.get_des(all_kps)
255
256         return all_kps, des
```

B.2 lab2.py

```
1   import cv2
2   import img_io as iio
3   from sift import SIFT
4
5   def match(sift, dataset, target):
6       kp_target, des_target = sift.detectAndCompute(target, None)
7       bf = cv2.BFMatcher()
8       result = []
9       for img in dataset:
10          kp, des = sift.detectAndCompute(img, None)
11          matches = bf.knnMatch(des_target, des, k=2)
12          good_matches = [m for m, n in matches if m.distance < 0.7 * n.distance]
13          matched_img = cv2.drawMatches(target, kp_target, img, kp, good_matches, img, flags=2)
14          result.append(matched_img)
15      return result
16
17  def main(useOpenCV: bool):
```

```
    dataset, target = iio.load_dataset_and_target()
    sift = cv2.SIFT_create() if useOpenCV else SIFT()
    result = match(sift, dataset, target)
    iio.save_output(result, suffix='opencv' if useOpenCV else None)

if __name__ == "__main__":
    main(True) # OpenCV
    main(False) # mine
```

B.3 debug.py

```
import cv2
import img_io as iio

def print_keypoint(kp: cv2.KeyPoint):
    print(f"Angle: {kp.angle}, Size: {kp.size}, Octave: {kp.octave}.")

def save_doc_image(img: cv2.typing.MatLike, name: str):
    iio.save_single_image(img, name, path='../doc/images/')
```

B.4 img_io.py

```
import cv2

def load_dataset_and_target(
    dataset_path: str = 'images/dataset/',
    dataset_names: list[str] = ['1.jpg', '2.jpg', '3.jpg', '4.jpg', '5.jpg'],
    target_path: str = 'images/target.jpg',
    color: int = cv2.COLOR_BGR2GRAY) -> tuple[list[cv2.typing.MatLike], cv2.typing.MatLike]:

    dataset = [cv2.imread(dataset_path + name) for name in dataset_names]
    target = cv2.imread(target_path)

    dataset = [cv2.cvtColor(img, color) for img in dataset]
    target = cv2.cvtColor(target, color)

    return dataset, target

def save_output(
    imgs: list[cv2.typing.MatLike],
    output_path: str = 'output/',
    suffix: str = None):

    for _, img in enumerate(imgs):
        path = output_path + f"{_ + 1}"
```

```
        if suffix:
            path += '-' + suffix
        path += '.png'

        cv2.imwrite(path, img)

def save_single_image(
    img: cv2.typing.MatLike,
    name: str,
    path: str
):

    if not name or not path:
        return

    cv2.imwrite(path + name + '.png', img)
```