实验 9&10-SIFT 尺度不变特征变换

姓名：王煌基　　学号：519030910100　　班级：F1903004

一、实验原理[1]

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。

总的来说，SIFT 有以下特点：

1. 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**；
2. **独特性**（Distinctiveness）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的 SIFT 特征向量；
4. **高速性**，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**，可以很方便的与其他形式的特征向量进行联合。

并且，利用 SIFT 可以解决例如以下的一些问题：

1. 目标的旋转、缩放、平移（RST）
2. 图像仿射/投影变换（视点 viewpoint）
3. 光照影响（illumination）
4. 目标遮挡（occlusion）
5. 杂物场景（clutter）
6. 噪声

二、实验环境

1. 个人笔记本电脑
2. 操作系统：windows10 专业版
3. 使用软件：Visual Studio Code；Docker Desktop

三、实验算法[2]

1. 尺度空间极值检测：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

[1] 参考自 CSDN: https://blog.csdn.net/zddblog/article/details/7521424

[2] Lowe 将 SIFT 算法分解为四步，即尺度空间极值检测、关键点定位、方向确定、关键点描述
David Lowe. Distinctive Image Features from Scale-Invariant Keypoints. International journal on computer vision. 2004.

于尺度和旋转不变的兴趣点。

2. 关键点定位：在每个候选的位置上，通过一个拟合精细的模型来确定位置和尺度。关键点的选择依据于它们的稳定程度。

3. 方向确定：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，从而提供对于这些变换的不变性。

4. 关键点描述：在每个关键点周围的邻域内，在选定的尺度上测量图像局部的梯度。这些梯度被变换成一种表示，这种表示允许比较大的局部形状的变形和光照变化。

四、实验流程

练习一：请在 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。与 OpenCV 自带的 SIFT 函数比较。

【解】

经过对题意的分析，可以大概清楚，本次实验的流程应该是：

①读入目标匹配图片与当前任务图片并进行灰度处理

②利用 SIFT 算法分别得到两张图片的关键点（特征点）和描述子

③利用临近算法（KNN）进行描述子的匹配，返回符合的匹配

④将符合的匹配删去可能存在的“交叉”现象，得到匹配结果并画图

从而可以利用这个大致的流程来完成本次的练习。同时，为了减小单项任务的任务量，可以将本次练习分为两个部分，第一个部分即利用 OpenCV 自带的 SIFT 函数完成任务，第二个部分即利用自己实现的 SIFT 完成任务，显然，第一个部分是相对比较容易完成的一个方式。

利用 OpenCV 自带的 SIFT 函数实现：

首先先建立 SIFT 的实例化函数

```
sift = cv2.xfeatures2d.SIFT_create()
```

然后读入目标匹配图片与当前任务图片并进行灰度处理，并利用 SIFT 算法分别得到两张图片的关键点（特征点）和描述子，并且为了表示的方便性可以将获得的关键点利用红色圆圈在图上标出（图一图二是类似的）

```
img1 = cv2.imread(imgname1)
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
#灰度处理图像
kp1, des1 = sift.detectAndCompute(img1,None)
#kp1 是关键点(keypoints),des 是描述子
cv2.drawKeypoints(img1,kp1,img1,color=(255,0,255))
#画出特征点，并显示为红色圆圈
```

再利用 BFMatcher 匹配器[3]，对描述子进行最佳匹配，一般而言两张图的匹配应该是结果近似于平行的，因此可以对所有的暴力匹配的结果进行向量距离的限定，即如果两个匹配的距离差距过大，则说明这可能不是一个好的匹配：

```
# BFMatcher 匹配器
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2,k=2)
# 找好的匹配
newmatches = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        newmatches.append((m,n))
```

最后，基于好的匹配结果来进行图像的连线以及绘制（红色圆圈即关键点，有连线说明这个关键点是有效的）：

```
img5 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,newmatches,None,flags=2)
plt.imshow(img5)
plt.savefig(imgoutname)
```

从而，利用 OpenCV 自带的 SIFT 算法任务完成。

自己编写的 SIFT 函数实现：

对于自己编写的 SIFT，可以依葫芦画瓢，按照相同的流程进行，只是需要对关键步骤进行自己的具体实现。为了减轻单个任务的任务量，可以将其分为几个小阶段（按照流程来划分）。

①读入目标匹配图片与当前任务图片并进行灰度处理，并且利用自己实现的 SIFT 算法得到图片的 features、corners、cnt（图一图二是类似的）

```
# 首先进行目标图片以及待匹配图片的读取
target0 = cv2.imread('data3.jpg')
target = cv2.cvtColor(target0, cv2.COLOR_BGR2GRAY)
imgpkp = cv2.cvtColor(imgpkp0, cv2.COLOR_BGR2GRAY)
# sift 函数对图像进行处理，返回相似度的关系 feature，角点 corners，大小 cnt
feature_target, corners_target, cnt_target = sift(target)
```

②合并图像，并且利用自己编写的匹配器得到较好结果，基于该较好结果进行连线

```
# 这个是不同图片之间的位置向量，以此来对两点之间进行连线
w = np.shape(target)[1]
merge_graph = [Merge(target0, imgpkp0)]
# 开始进行匹配并划线
    # 对于匹配不错的两点之间进行连线保存图像，记录适配点的个数，方便与 OpenCV 自带的 SIFT 进行比较
```

[3] 即 opencv 二维特征点匹配常见的办法，BFMatcher 总是尝试所有可能的匹配，从而使得它总能够找到最佳匹配，这也是 Brute Force（暴力法）的原始含义。它通过发现两幅图片分别提取出来 N，M 个特征向量，然后对 N 和 M 的特征向量进行匹配，找到最佳匹配，然后再画出匹配的特征显示出来（这个匹配方式虽然较为暴力但是准确度比较高）

因此，本部分的关键点在于 SIFT 的实现过程。通过对网上的参考资料[4]的查找以及对本次实验教学 PPT 的理解，可以将 SIFT 分解为四个部分来进行完成。

a. 检测尺度空间关键点

这里需要利用到实验 8 中涉及的梯度、方向导数、角度等的计算，且需要随时注意的是，python 的 arctan 函数返回的默认为弧度制，因此如果需要转换成角度等情况的话需要特别注意。

```
def gradient(img):
    Ix, Iy = np.zeros((len(img),len(img[0]))), np.zeros((len(img),len(img[0])))
    M, angle = np.zeros((len(img),len(img[0]))), np.zeros((len(img),len(img[0])))
    for i in range(1,len(img)-1):
        for j in range(1,len(img[0])-1):
            # Ix 是 x 方向的梯度，Iy 即 y 方向的
            Ix[i][j] = int(img[i+1][j])-int(img[i-1][j])
            Iy[i][j] = int(img[i][j+1])-int(img[i][j-1])
            # M 为梯度强度
            M[i][j] = (Ix[i][j]**2+Iy[i][j]**2)**0.5
            angle[i][j] = np.arctan2(Ix[i][j],Iy[i][j])
    return M, angle
```

并且，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（octave）图像，一组图像包括几层（interval）图像。这里可以直接利用 OpenCV 自带的高斯滤波函数 GaussianBlur 而不用费过大的精力去手写高斯滤波函数：

```
img = cv2.GaussianBlur(img, (5, 5), 1)
```

b. 精确定位关键点

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。对某候选关键点做泰勒展开的话，准确的极值点位置便为该泰勒展开对 x 导数为 0 的位置。当然，实验要求中并不要求我们完成这一部分的内容，可以直接利用 OpenCV 中自带的 Harris 角点提取法来进行提取，可以大胆地将这些点就认为是关键点，便可以精确定位：

```
# 计算角点，并且计算其长度
corners, cnt = [], 0
```

---

[4] 本次主要参考了几处地方的教学、教程：
①https://www.bilibili.com/video/BV1Qb411W7cK?from=search&seid=7190328523258436250
标题：《6.SIFT(尺度不变特征变换)》
②https://blog.csdn.net/zddblog/article/details/7521424
标题：《SIFT 算法详解》
③https://www.cnblogs.com/my-love-is-python/p/10414135.html
标题：《机器学习进阶-图像特征 sift-SIFT 特征点 1.cv2.xfeatures2d.SIFT_create(实例化 sift) 2. sift.detect(找出关键点) 3.cv2.drawKeypoints(画出关键点) 4.sift.compute(根据关键点计算 sift 向量)》
④https://zhuanlan.zhihu.com/p/102272392
标题：《opencv 实战：SIFT 的实现》

```
3.  for i in cv2.goodFeaturesToTrack(img,300,0.01,10):
4.      corners.append([int(i[0][0]),int(i[0][1])])
5.      cnt += 1
```

c. 为每个关键点指定方向参数

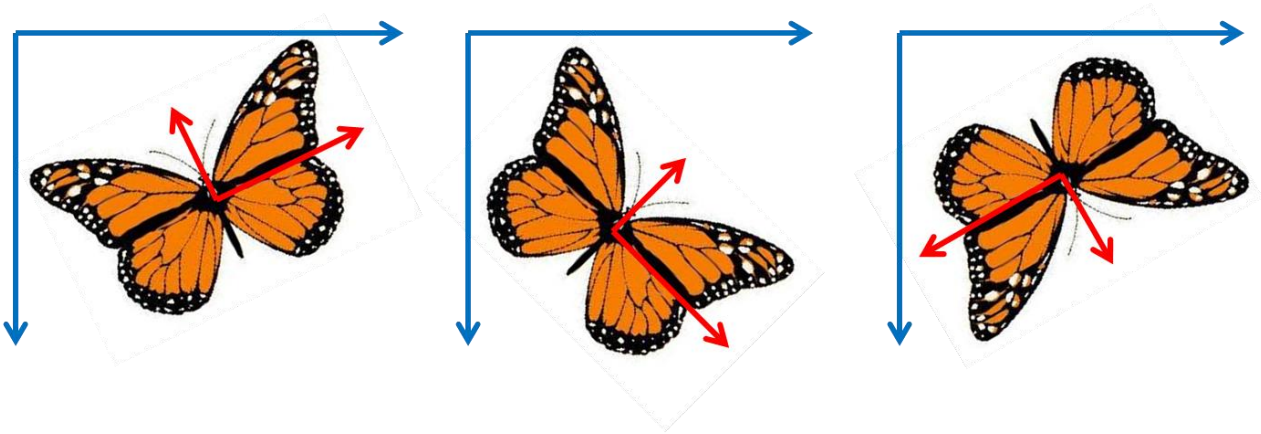
梯度方向为 360 度（注意 atan 函数返回值为 0-180，需要根据 Ix Iy 的符号换算），平均分成 36 个 bins，每个像素以 m(x, y)为权值为其所在的 bin 投票。最终权重最大的方向定位该关键点的主方向(实验中只考虑 highest peak)。

通过对课件内容的理解，可以发现，我们的目的是尽可能地将这 360 度的梯度方向均分为 36 个不同的区域，然后根据权值累加的方式来获取该关键点的最大权值所在方向，并将这个方向定位该关键点的指定方向：

```
1.      ccnt = [0] * 37
2.      # 防止数组越界，通过 max、min 函数进行修饰
3.      for i in range(边界):
4.          for j in range(边界):
5.              # 梯度方向为 360 度
6.              # （注意 atan 函数返回值为 0-180，需要根据 Ix Iy 的符号换算)
7.              weight = 所在区域值
8.              # 超出边界则进行修正，由于限定在 36 以内所以乘上系数 18 再+1
9.              # 分别计算每块区域的加权和，并且记录最大的那个加权和位置
10.             ccnt[weight] += grad[i][j]
11.     # 记录最大的加权和位置，并保存
12.     找最大位置所在处并将其当做主方向
```

d. 关键点描述子的生成

事实上，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。



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。

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$\theta'(x,y)=\theta(x,y)-\theta_0$，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， θ0 是关键点的主方向。

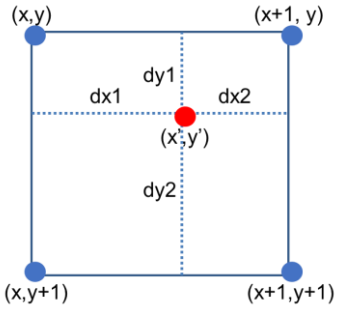
*物体坐标系 16*16 的邻域分成 4*4 个块，每个块 4*4 个像素。*

*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：$\theta(x',y')=\theta\big(Round(x'),Round(y')\big)$，或更精确地，可采用双线性插值：

$$\theta(x',y')=\theta(x,y)dx_2dy_2+\theta(x+1,y)dx_1dy_2+\theta(x,y+1)dx_2dy_1+\theta(x+1,y+1)dx_1dy_1$$

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：$f=f_0*\frac{1}{|f_0|}$，两个 SIFT 描述子 f1 和 f2 之间的相似度可表示为$s(f_1,f_2)=f_1*f_2$



通过上述的描述可以发现，描述子的算法实现还是较为繁琐复杂的，但通过网络上的相关参考资料的详细讲解，总的来说难度下降了不少，步骤大体如下：

1. 确定计算描述子所需的图像区域：

特征描述子与特征点所在的尺度有关，因此，对梯度的求取应在特征点对应的高斯图像上进行。将关键点附近的邻域划分为 d*d(Lowe 建议 d=4)个子区域，每个子区域做为一个种子点，每个种子点有 8 个方向。每个子区域的大小与关键点方向分配时相同

```
bins = (row + column) // 150
# 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。
# 记录 16 次的 8 维分布情况并返回
for xsign in [-1,1]:
    for ysign in [-1,1]:
        vector += cnt(0, bins, 0, bins, xsign, ysign)
        vector += cnt(bins, bins*2, 0, bins, xsign, ysign)
        vector += cnt(bins, bins*2, bins, bins*2, xsign, ysign)
        vector += cnt(0, bins, bins, bins*2, xsign, ysign)
```

2. 将坐标轴旋转为关键点的方向，以确保旋转不变性

旋转后邻域内采样点的新坐标为下面的代码（即利用三角函数去旋转）：

```
Horizon = np.array([np.cos(θ), np.sin(θ)])
Vertical = np.array([-np.sin(θ),np.cos(θ)])
p = Horizon * [x] + Vertical * [y]
```

3. 将邻域内的采样点分配到对应的子区域内，将子区域内的梯度值分配到 8 个方向上，计算其权值

```
weig = int(插值计算某个点所在的区间)
if weig > 8:
    weig = 8
# 满足条件，计数
count[weig] += 1
```

4. 插值计算每个种子点八个方向的梯度

我们是通过插值法计算的权重值，而且尽可能保证确实是从 1~8 一一映射：

```
def Bilinear_Interpolation(x_1, y_1):
    # 首先找到 x_1、y_1 周围的最小整数点 x,y 并且以此为基础进行处理
    x, y = int(x_1), int(y_1)
    dx1, dy1 = x_1-x, y_1-y
    dx2, dy2 = 1-dx1, 1-dy1
    θθ = θ_(x,y)*dx2*dy2 + θ_(x+1,y)*dx1*dy2 + θ_(x,y+1)*dx2*dy1 + θ_(x+1,y+1)*dx1*dy1
    return θθ
```

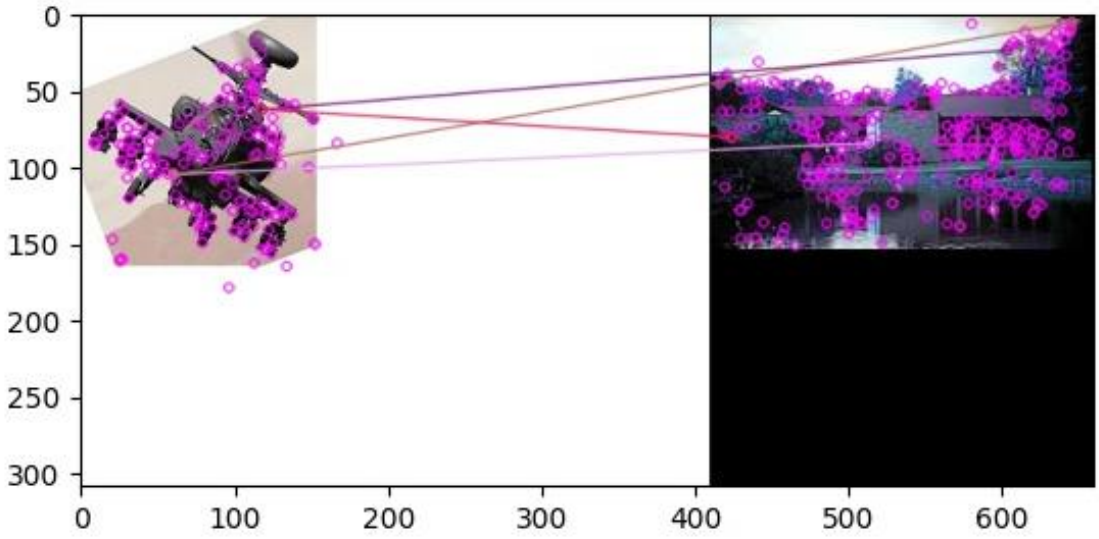
5. 统计特征向量，进行归一化处理，并按特征点尺度进行排序

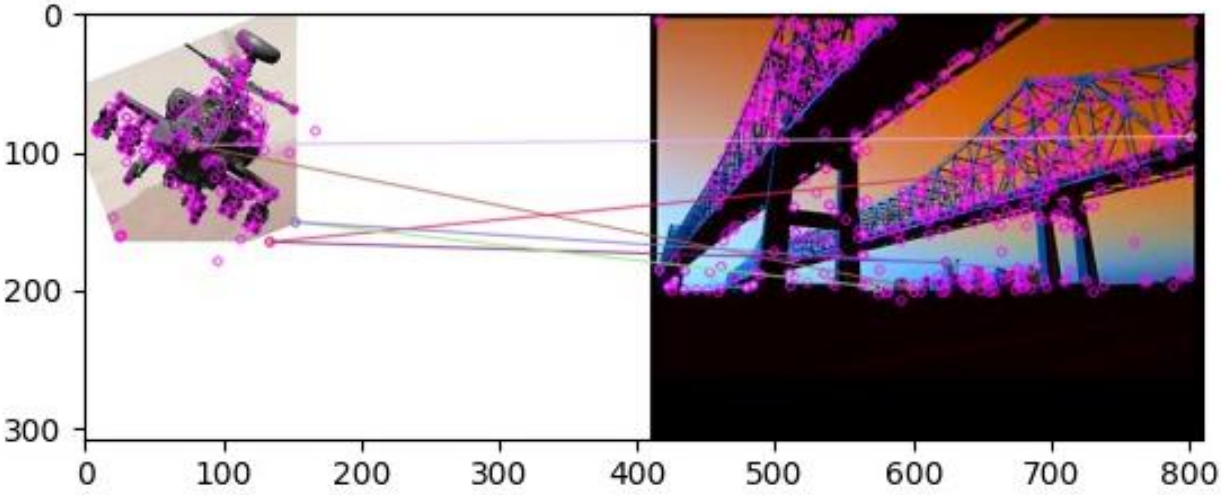
```
# 通过特征提取获取每个角点的主方向
feature = []
# print(cnt)
for i in range(cnt):
    des = Featurelize(corners[i], direction[i])
    # 对该直方图向量进行归一化处理（即描述子）
    norm = sum(k * k for k in des) ** 0.5
    l = [k*1.0 / norm for k in des]
    # 最后对 128 维 SIFT 描述子 f0 归一化得到最终的结果：
    feature.append(l)
```

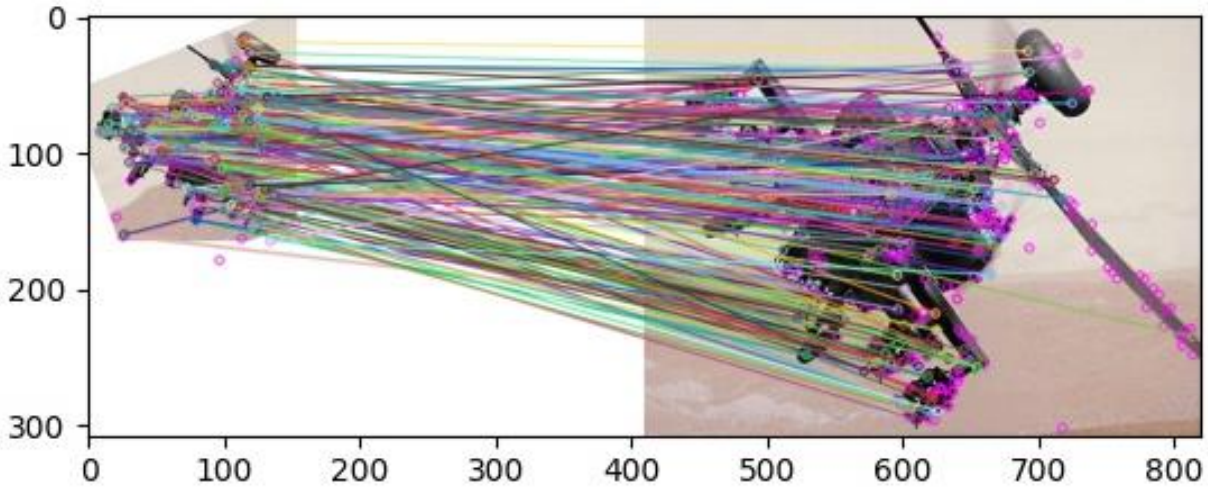
通过这样一步一步的打好基石，实际上我的这一块实验便完成了！

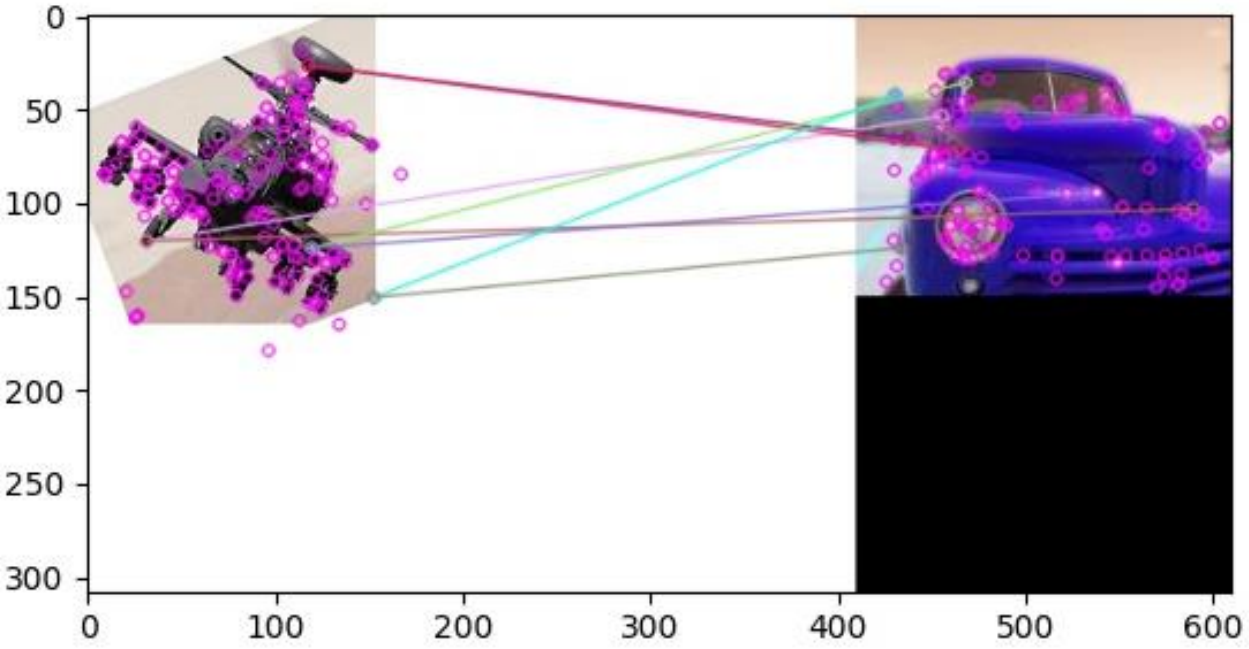
五、实验结果

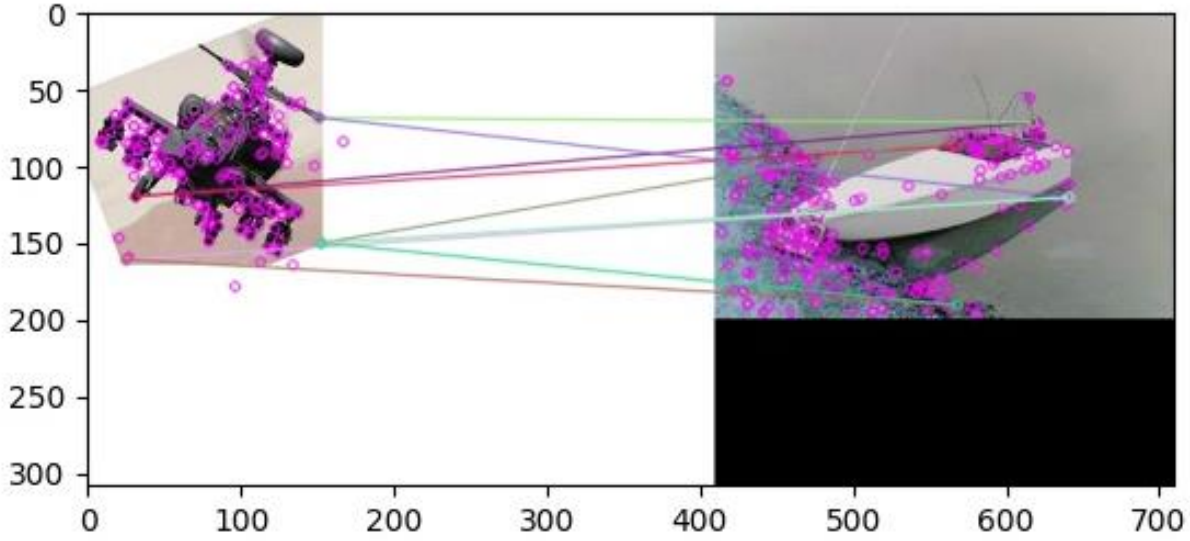
OpenCV 自带的 SIFT 函数实现（有效匹配）（自上往下为图 1、2、3、4、5）：







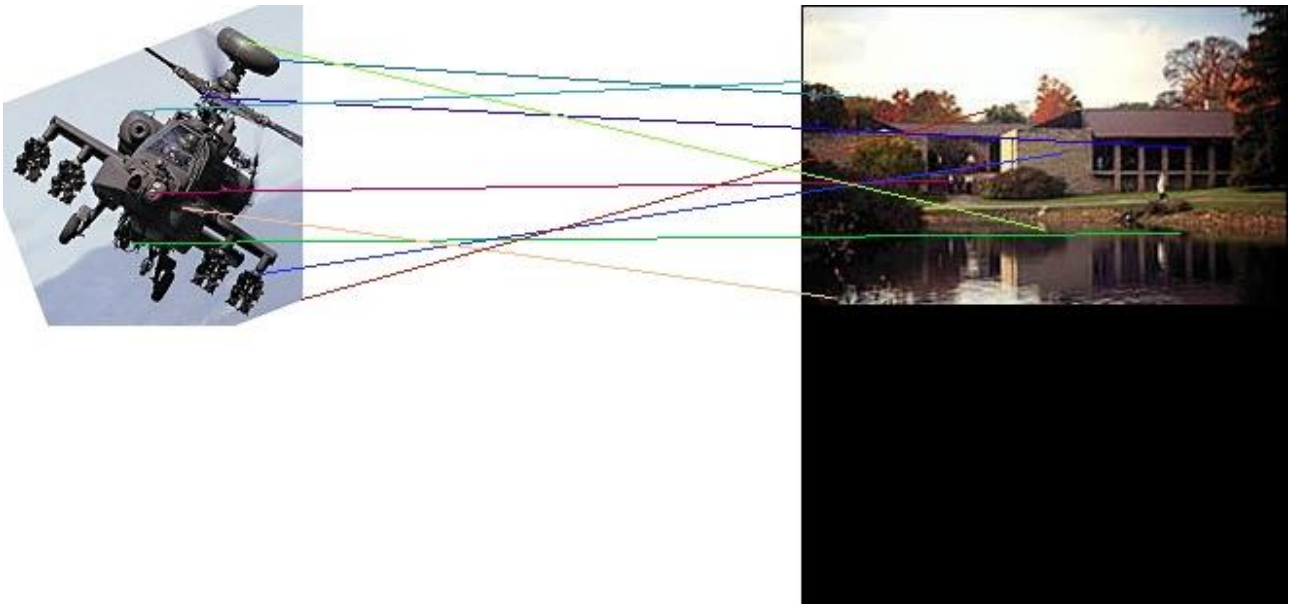


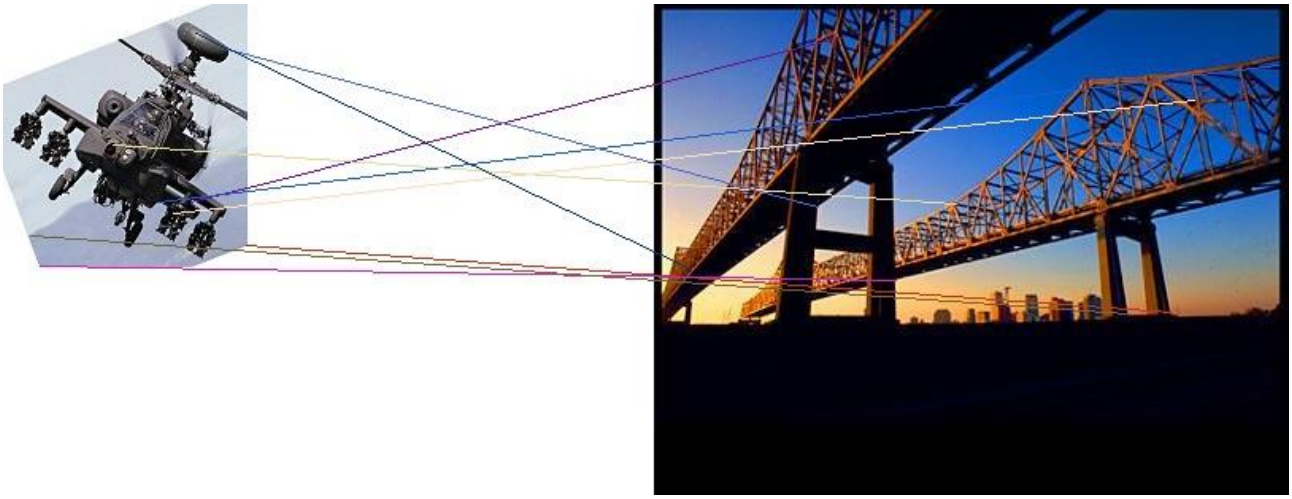


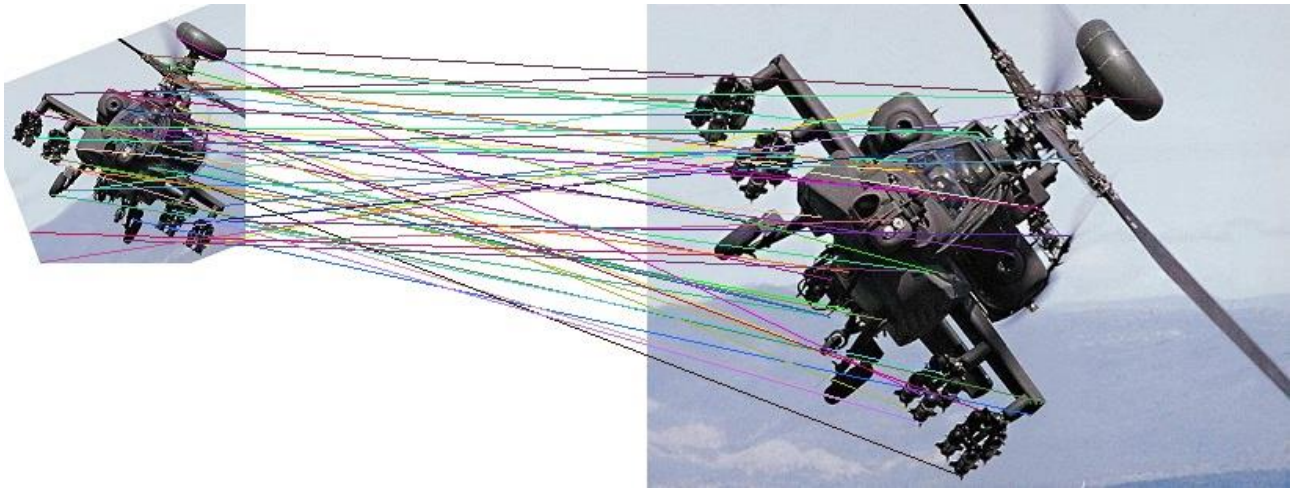
从上面的图像可以发现，虽然两张图新的特征点非常多，但是实际上真正满足匹配的结果并不是很多。

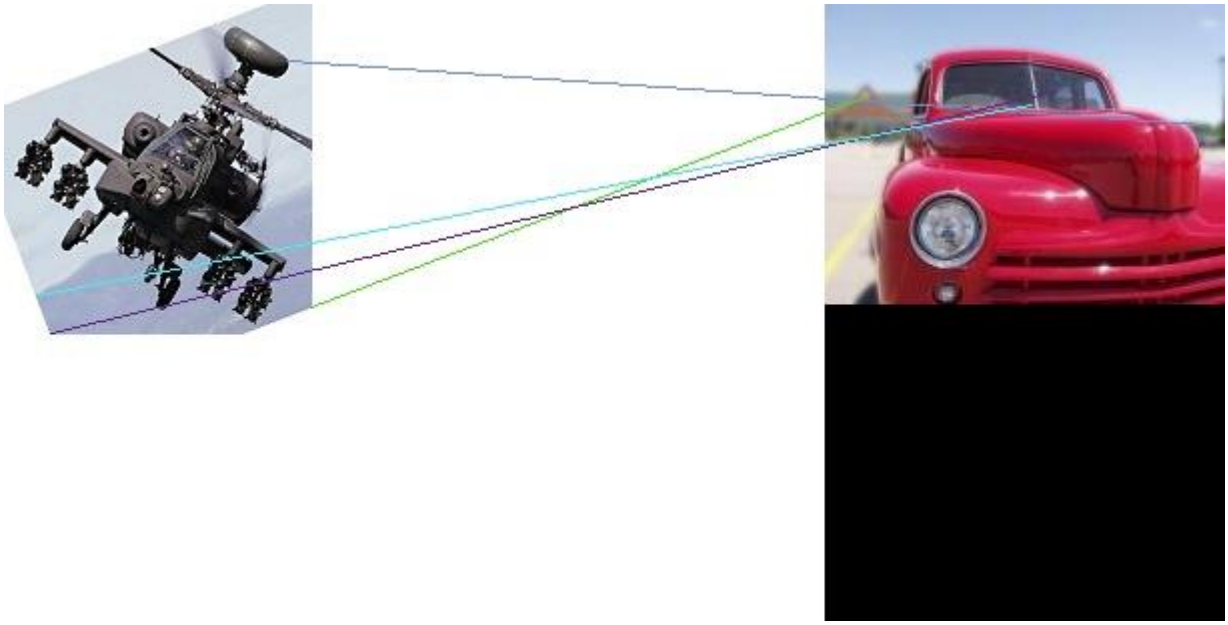
output_with_cv2 文件夹中带有其他的两种图片，均是没有进行“好匹配”筛选的情况下的连线情况，命名分别为：outputofdataX.jpg 和 outputofdataXwith30points.jpg，前者为显示所有匹配点，后者为仅显示最高的 30 个匹配点。通过我们实际的对比可以发现，没有筛选过程的情况下得到的图像有非常多杂糅在一起的线，而这并不是我们想要的，因此需要进行筛选，图片可供参考。

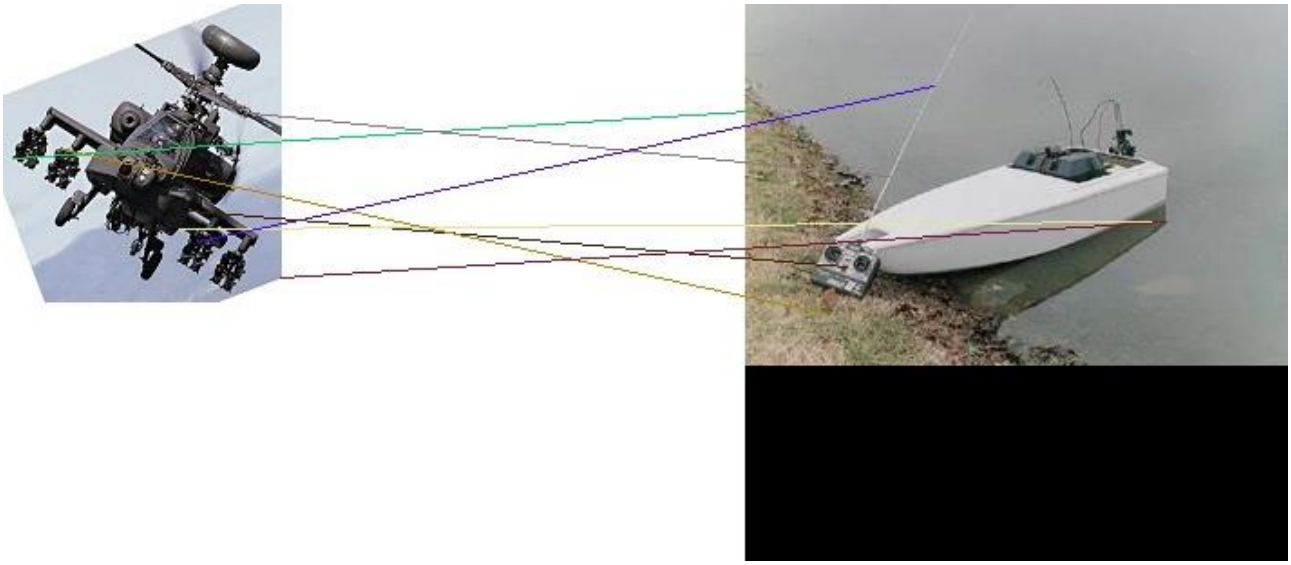
自己编写的 SIFT 函数实现（有效匹配）（自上往下为图 1、2、3、4、5）：











为了方便对比二者的情况，我利用了表格来显示：

连线数	OpenCV 自带的 SIFT	自己实现的 SIFT
1.jpg	4	9
2.jpg	6	9
3.jpg	250	51
4.jpg	8	4
5.jpg	10	7

总体而言，我自己实现的 SIFT 还不够理想，仍旧存在较多的交叉现象和匹配没那么精确的情况，但是由于是自己实现的，因而对此的体会更加丰富。

六、实验体会及拓展思考

本次接触的 SIFT 是一个全新的模块，出自对这个模块的极度缺乏知识以及课件内容的精炼，致使我刚开始对于该实验是完全没有任何想法的，直到后面查找到了如 B 站的相关教程，才对 SIFT 有一个自己的初步认识，得益于此，在本次实验中，我对具体问题的资料查询方向和查询方式都有了很大的提升，并且这次实验进一步加深了我对灰度图、梯度图等基本知识的巩固，对我是大有裨益的！

实验中遇到的比较大的问题均在代码中以注释的形式标出，解决方式主要是：

①将错误原因发至百度、CSDN 等地进行原因查找

②多 print 如 type、size、shape、内容等属性来判断操作是否得当并修改

七、实验代码

1. OpenCV 自带的 SIFT 函数实现（有效匹配）**CV_SIFT.py**：

```python
import numpy as np
import cv2
from matplotlib import pyplot as plt
# imgoutname = 'outputofdata5.jpg'
imgoutname = 'outputofdata3withtestpoints.jpg'
imgname1 = 'data3.jpg'
imgname2 = 'dataset/3.jpg'

sift = cv2.xfeatures2d.SIFT_create()

img1 = cv2.imread(imgname1)
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
# 灰度处理图像
kp1, des1 = sift.detectAndCompute(img1,None)
# kp1 是关键点(keypoints),des 是描述子

img2 = cv2.imread(imgname2)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
# 灰度处理图像
kp2, des2 = sift.detectAndCompute(img2,None)
# kp2 是关键点(keypoints),des 是描述子

# hmerge = np.hstack((gray1, gray2))
# 如果限制点数（由于特征点过多）
# img3 = cv2.drawKeypoints(img1,kp1[:20],img1,color=(255,0,255))
cv2.drawKeypoints(img1,kp1,img1,color=(255,0,255))
# 画出特征点，并显示为红色圆圈
# 如果限制点数（由于特征点过多）
# img4 = cv2.drawKeypoints(img2,kp2[:20],img2,color=(255,0,255))
```

```
cv2.drawKeypoints(img2,kp2,img2,color=(255,0,255))
# 画出特征点，并显示为红色圆圈

# hmerge = np.hstack((img3, img4))

# BFMatcher 匹配器
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2,k=2)
# 找好的匹配
newmatches = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        newmatches.append((m,n))
# img5 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches[:20],None,flags=2)
img5 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,newmatches,None,flags=2)
plt.imshow(img5)
plt.savefig(imgoutname)
```

2. 自己编写的 SIFT 函数实现（有效匹配）SIFT.py：

```
import cv2
import math
from matplotlib import pyplot as plt
import numpy as np
import random
PI = math.pi
# 基本照搬实验 8 的梯度图代码，返回梯度值以及所获得的的角度 angle
def gradient(img):
    Ix, Iy = np.zeros((len(img),len(img[0]))), np.zeros((len(img),len(img[0])))
    M, angle = np.zeros((len(img),len(img[0]))), np.zeros((len(img),len(img[0])))
    for i in range(1,len(img)-1):
        for j in range(1,len(img[0])-1):
            # Ix 是 x 方向的梯度，Iy 即 y 方向的
            Ix[i][j] = int(img[i+1][j])-int(img[i-1][j])
            Iy[i][j] = int(img[i][j+1])-int(img[i][j-1])
            # M 为梯度强度
            M[i][j] = (Ix[i][j]**2+Iy[i][j]**2)**0.5
            angle[i][j] = np.arctan2(Ix[i][j],Iy[i][j])
    return M, angle

def sift(img):
    row, column = np.shape(img)

    # 计算角点，并且计算其长度
    corners, cnt = [], 0
    for i in cv2.goodFeaturesToTrack(img,300,0.01,10):
        corners.append([int(i[0][0]),int(i[0][1])])
        cnt += 1
```

```
29.
30.     # 对其进行高斯模糊化处理
31.     # 高斯矩阵的长宽为 5*5，标准差取 1
32.     img = cv2.GaussianBlur(img, (5, 5), 1)
33.
34.     # 获取梯度图、角图
35.     grad, angle = gradient(img)
36.
37.     # 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。
38.     # 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)。
39.     direction = []
40.     # 确定往不同方向拓展的最大尺度 bins（最好满足能限制于 36 等分的区域内）
41.     # 适当大一些也无伤大雅
42.     bins = (row + column) // 80
43.     for corner in corners:
44.         # 获取角点坐标
45.         y, x = corner
46.         # 对每一个大块进行分块、计数
47.         # 实际上 0 这个维度基本是用不到的，因为我们有+1 的操作
48.         ccnt = [0] * 37
49.         # 防止数组越界，通过 max、min 函数进行修饰
50.         for i in range(max(x-bins,0),min(x+bins+1,row)):
51.             for j in range(max(y-bins,0),min(y+bins+1,column)):
52.                 # 梯度方向为 360 度
53.                 # （注意 atan 函数返回值为 0-180，需要根据 Ix Iy 的符号换算)
54.                 weight = int((angle[i][j]+PI)*1.0/(PI/18)+1)
55.                 # 超出边界则进行修正，由于限定在 36 以内所以乘上系数 18 再+1
56.                 if(weight>36):
57.                     weight = 36
58.                 # 分别计算每块区域的加权和，并且记录最大的那个加权和位置
59.                 ccnt[weight] += grad[i][j]
60.         # 记录最大的加权和位置，并保存
61.         maxn = 1
62.         for i in range(36):
63.             if ccnt[i]>ccnt[maxn]:
64.                 maxn = i
65.         # 注意需要将角度转化为弧度
66.         direction.append((maxn*1.0/18-1-1.0/36)*PI)
67.     # 以此获取最佳方向
68.
69.     # 下面对描述子进行生成
70.     # 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
71.     # 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
72.     # θ’是相对物体坐标系的梯度方向，
73.     # θ 是相对图像坐标系的梯度方向， θ0 是关键点的主方向。
74.     def Featurelize(point, θ):
```

```
        def θ_(x, y):
            if (x < 0 or x >= row) or (y < 0 or y >= column):
                return 0
            tmp = angle[x][y] - θ
            if tmp > 0:
                return tmp
            return tmp + 2 * PI

        # 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：
        # θ_(x',y')=θ_(int(x),int(y))
        # 或更精确地，可采用双线性插值：
        # 从而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。
        def Bilinear_Interpolation(x_1, y_1):
            # 首先找到 x_1、y_1 周围的最小整数点 x,y 并且以此为基础进行处理
            x, y = int(x_1), int(y_1)
            dx1, dy1 = x_1-x, y_1-y
            dx2, dy2 = 1-dx1, 1-dy1
            θθ = θ_(x,y)*dx2*dy2 + θ_(x+1,y)*dx1*dy2 + θ_(x,y+1)*dx2*dy1 + θ_(x+1,y+1)*dx1*dy1
            return θθ

        # 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。
        # 以特征点为中心，取领域内 16*16 大小的区域，
         # 并把这个区域分成 4*4 个大小为 4*4 的小区域，每个小区域内计算加权梯度直方图，
         # 该权值分为 2 部分，其一是该点的梯度大小，
         # 其二是改点离特征点的距离(二维高斯的关系)，每个小区域直方图分为 8 个 bin，
         # 所以一个特征点的维数=4*4*8=128 维。
         # 因此，根据 SIFT 算法详解的文章，应构造旋转变换的邻域
         # 即(x',y')^T=([cosθ,-sinθ],[sinθ,cosθ])(x,y)^T
         y0, x0 = point
         Horizon = np.array([np.cos(θ), np.sin(θ)])
         Vertical = np.array([-np.sin(θ),np.cos(θ)])
         # 并且统计其具体某个方向上（此时为 8 个方向）的梯度，再次计算

         def cnt(x1, x2, y1, y2, signx, signy):
             count = [0] * 9
             for x in range(x1, x2):
                 for y in range(y1, y2):
                     # 找到具体的 x,y（区分正负半轴）
                     dp = [x * signx, y * signy]
                     # 矩阵乘法，获得变换后的坐标
                     p = Horizon * dp[0] + Vertical * dp[1]
                     # 获取这个方向上的点理论上所在的方向（根据 1~8 进行划分）
```

```
                    weig = int((Bilinear_Interpolation(p[0]+x0, p[1]+y0))//(PI/4) + 1)
                    if weig > 8:
                        weig = 8
                    # 满足条件，计数
                    count[weig] += 1
            # 最后返回一个直方图数据，表示 8 个方向各个出现的次数，从而决定主方向
            return count[1:]

        vector = []
        # 确定往不同方向拓展的最大尺度 bins（最好满足能限制于 36 等分的区域内）
        # 适当大一些也无伤大雅
        bins = (row + column) // 150
        # 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。
        # 记录 16 次的 8 维分布情况并返回
        for xsign in [-1,1]:
            for ysign in [-1,1]:
                vector += cnt(0, bins, 0, bins, xsign, ysign)
                vector += cnt(bins, bins*2, 0, bins, xsign, ysign)
                vector += cnt(bins, bins*2, bins, bins*2, xsign, ysign)
                vector += cnt(0, bins, bins, bins*2, xsign, ysign)
        return vector

    # 通过特征提取获取每个角点的主方向
    feature = []
    # print(cnt)
    for i in range(cnt):
        des = Featurelize(corners[i], direction[i])
        # 对该直方图向量进行归一化处理（即描述子）
        norm = sum(k * k for k in des) ** 0.5
        l = [k*1.0 / norm for k in des]
        # 最后对 128 维 SIFT 描述子 f0 归一化得到最终的结果:
        feature.append(l)
    # 因此两个 SIFT 描述子 f1 和 f2 之间的相似度可表示为:s(f1,f2)=f1*f2
    return feature, corners, cnt

def Merge(img1, img2):
    # 为小一点的图像补充空间，为大一点的图像适配小一点的图像的对等大小
    # 获取图片的性质，其中只有 h 是有意义的，其他在本处是无意义的
    h1, w1 ,a= np.shape(img1)
    h2, w2 ,a= np.shape(img2)
    # 图一规模较小则补全图一
    if h1 < h2:
        extra = np.array([[[0,0,0] for i in range(w1)] for ii in range(h2-h1)])
        img1 = np.vstack([img1, extra])
    # 图二规模较小则补全图二
    elif h1 > h2:
```

```
166.         extra = np.array([[[0,0,0] for i in range(w2)] for ii in range(h1-h2)])
167.         img2 = np.vstack([img2, extra])
168.     # 处理完毕之后自然是相同规模，可以直接水平方向拼接
169.     return np.hstack([img1,img2])
170. 
171. if __name__ == "__main__":
172. 
173.     # 首先进行目标图片以及待匹配图片的读取
174.     target0 = cv2.imread('data3.jpg')
175.     imgpkp0 = cv2.imread('dataset/5.jpg')
176.     # print(np.shape(target0))
177.     # 获取图片的大小等参数， 并且将目标图片进行适当的放大
178.     # r0, c0, a0=np.shape(target0)
179.     # times=1.0
180.     # 将目标图片放大至待匹配图片的等大小规模
181.     # resized_target0=cv2.resize(target0,(int(r0*times),int(c0*times)))
182.     # 灰度化处理图像
183.     target = cv2.cvtColor(target0, cv2.COLOR_BGR2GRAY)
184.     imgpkp = cv2.cvtColor(imgpkp0, cv2.COLOR_BGR2GRAY)
185. 
186.     # sift 函数对图像进行处理，返回相似度的关系 feature，角点 corners，大小 cnt
187.     feature_target, corners_target, cnt_target = sift(target)
188.     feature_img, corners_img, cnt_img = sift(imgpkp)
189. 
190.     # 这个是不同图片之间的位置向量，以此来对两点之间进行连线
191.     w = np.shape(target)[1]
192. 
193.     merge_graph = [Merge(target0, imgpkp0)]
194.     # merge_graph = [Merge(target0, imgpkp0)]
195.     # print("All Original Pics Processed!")
196. 
197.     # 开始进行匹配并划线
198.     x = []
199.     # 判断关键点的匹配数目
200.     # 如果超过 6 个则可视为一个合理匹配方式，否则不是
201.     cnt = 0
202.     for i in range(cnt_target):
203.         tmp = []
204.         for j in range(cnt_img):
205.             # 两个 SIFT 描述子 f1 和 f2 之间的相似度可表示为 s(f1,f2)=f1·f2=inner(f1,f2)
206.             similar_rate = np.inner(np.array(feature_target[i]), np.array(feature_img[j]
    ))
207.             tmp.append(similar_rate)
208.         x.append([tmp.index(max(tmp)), max(tmp)])
209.     # Traceback (most recent call last):
210.     #   File "SIFT.py", line 256, in <module>
```

```
    #     cv2.line(merge_graph[0], tuple(ct[a]), tuple([c[b][0] + w,c[b][1]]), color, 1)

    # TypeError: Expected Ptr<cv::UMat> for argument 'img'
    # 为了解决这个报错，需要利用转码改为 uint8
    merge_graph[0] = np.array(merge_graph[0], dtype="uint8")
    for i in range(len(x)):
        j, rate = x[i]
        # 如果匹配率较低，说明这并不是我们想要的点
        if rate < 0.65:
            continue
        cnt += 1
        # 对于划线过程，进行颜色的随机设定
        color = ((random.randint(0, 255)),(random.randint(0, 255)),(random.randint(0, 255)))
        # print("\n----------------")
        # print(corners_target[a])
        # print(corners_img[b])
        # print(merge_graph[0])
        cv2.line(merge_graph[0], tuple(corners_target[i]), tuple([corners_img[j][0] + w, corners_img[j][1]]), color, 1)
    cv2.imwrite("match5with%d.jpg"%cnt, merge_graph[0])
```

八、参考文献及视频

①《6.SIFT(尺度不变特征变换)》——UP 主：会飞的吴克

https://www.bilibili.com/video/BV1Qb411W7cK?from=search&seid=7190328523258436250

②《SIFT 算法详解》——zddhub

https://blog.csdn.net/zddblog/article/details/7521424

③《机器学习进阶-图像特征 sift-SIFT 特征点 1.cv2.xfeatures2d.SIFT_create(实例化 sift) 2. sift.detect(找出关键点) 3.cv2.drawKeypoints(画出关键点) 4.sift.compute(根据关键点计算 sift 向量)》——my-love-is-python

https://www.cnblogs.com/my-love-is-python/p/10414135.html

④《opencv 实战：SIFT 的实现》——Kirei

https://zhuanlan.zhihu.com/p/102272392