第十二次实验报告

学号：518030910308
姓名：刘文轩

一、实验准备

1、 实验环境介绍

操作系统：ubuntu 14.04
语言：Python 2
IDE：Pycharm 2019.2.3

2、 实验目的

2.1 了解 SIFT 算法的概念
2.2 学会检测并定位空间极值点
2.2 学会自己编写 SIFT 算法

3、实验思路

3.1 将获取的图片灰度化
3.2 检测空间极值点
3.3 为每个关键点指定方向参数
3.4 生成关键点描述子
3.5 设计相关辅助函数 code.py，预处理程序 PreProcess.py，匹配程序 Match.py，**运行本实验时，请先运行 PreProcess.py，再运行 Match.py**

二、实验过程

1、 设计相关负责函数、code.py

1.1 getGrad()

我们在此使用上一个实验中的 Canny 算法对图片进行预处理。

首先使用高斯滤波函数对图像处理，接着使用 Canny 核求解图片的相关梯信息，函数返回图片像素梯度的大小和方向。

同时在此使用 scharr 算子，它与 Sobel 的不同点是在平滑部分，这里所用的平滑算子是 1/16*[3,10,3]，相比于 1/4*[1,2,1]，中心元素占的权重更重，这可能是相对于图像这种随机性较强的信号，邻域相关性不大，所以邻域平滑应该使用相对较小的标准差的高斯函数，也就是更瘦高的模板。结果更加精确。

```
1.  def getGrad(img0):
2.      img = cv2.GaussianBlur(img0, ksize=(7, 7), sigmaX=1.03) * 1.0
3.      cannyKerX = np.array([[-3, -10, -3],
4.                            [0, 0, 0],
5.                            [3, 10, 3]])
6.      cannyKerY = np.array([[-3, 0, 3],
7.                            [-10, 0, 10],
8.                            [-3, 0, 3]])
9.      grad_x = cv2.filter2D(img, ddepth=-1, kernel=cannyKerX, anchor=(-1, -
    1)) / 16
10.     grad_y = cv2.filter2D(img, ddepth=-1, kernel=cannyKerY, anchor=(-1, -
    1)) / 16
```

```
    grad_norm = np.power(np.add(np.power(grad_x, 2), np.power(grad_y, 2)), 0.5)
    grad_direct = np.arctan(np.divide(grad_y, np.add(grad_x, 0.000001)))
    M = grad_norm.shape[0]
    N = grad_norm.shape[1]
    for i in range(M):
        for j in range(N):
            if grad_x[i, j] < 0:
                grad_direct[i, j] = grad_direct[i, j] + np.pi
            if grad_direct[i, j] < 0:
                grad_direct[i, j] = grad_direct[i, j] + (np.pi * 2)
    tmpimg = np.array(img0)  # np.array Gouzao
    #cv2.imshow("grayImg", tmpimg)
    #cv2.waitKey(0)
    for i in range(0, M, 10):
        for j in range(0, N, 10):
            endx = i + int(np.cos(grad_direct[i, j]) * 5)
            endy = j + int(np.sin(grad_direct[i, j]) * 5)
            tmpimg = cv2.line(tmpimg, (j, i), (endy, endx), 255)  # Draw Direction
    #cv2.imshow("theGrad", tmpimg)
    #cv2.waitKey(0)
    return grad_norm, grad_direct
```

1.2 subGraph()

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 的领域内统计的。
此函数获取相对应的 16 ×16 的子图图像信息。

```
def subGraph(img, direct, x, y, main_direct):
    N = img.shape[0]
    M = img.shape[1]
    ret = np.zeros((16, 16))
    ret2 = np.zeros((16, 16))
    for i in range(-8, 8):
        for j in range(-8, 8):
            r = np.power((np.power(i, 2) + np.power(j, 2)), 0.5)
            theta = np.arctan(j * 1.0 / (i + 0.000001))
            if i < 0:
                theta += np.pi
            if theta < 0:
                theta += (np.pi * 2)
            theta += main_direct
            if theta > (np.pi * 2):
                theta -= (np.pi * 2)
```

```
            rx = x * 1.0 + r * np.cos(theta)
            ry = y * 1.0 + r * np.sin(theta)
            rx = min(int(np.rint(rx)), N - 1)
            rx = max(rx, 0)
            ry = min(int(np.rint(ry)), M - 1)
            ry = max(ry, 0)
            ret[i + 8, j + 8] = img[rx, ry]
            ret2[i + 8, j + 8] = direct[rx, ry] - main_direct
            if ret2[i + 8, j + 8] < 0:
                ret2[i + 8, j + 8] = ret2[i + 8, j + 8] + (np.pi * 2)
    return ret, ret2
```

1.3 getKPandDES()

SIFT 算法中，相当重要的一部分就是获取关键点并且为其确定方向。此函数就是做此处理。

```
def getKPandDES(img, MaxCorner):
    M = img.shape[0]
    N = img.shape[1]
    kp = cv2.goodFeaturesToTrack(img, maxCorners=MaxCorner,
                                 qualityLevel=0.03, minDistance=10)
    kp = kp.tolist()
    MaxCorner = len(kp)
    des = np.zeros((MaxCorner, 128))
    grad_norm, grad_direct = getGrad(img)
    main_direct_list = [0.0] * MaxCorner
    for ip in range(MaxCorner):
        p = kp[ip]
        y = int(p[0][0])
        x = int(p[0][1])
        if ((x - 8 < 1) or (x + 8 > M - 1)
                or (y - 8 < 1) or (y + 8 > N - 1)):
            continue
        sub_norm = np.array(grad_norm[x - 8:x + 8, y - 8:y + 8])
        sub_direct = np.array(grad_direct[x - 8:x + 8, y - 8:y + 8])
        main_direct_bin = np.zeros((1, 36))
        for i in range(16):
            for j in range(16):
                ind = int(sub_direct[i, j] * 18 / np.pi)
                main_direct_bin[0, ind] += sub_norm[i, j]
        main_direct = np.argmax(main_direct_bin) * (np.pi / 18) + (np.pi / 36)
        sub_norm, sub_direct = subGraph(img, grad_direct, x, y, main_direct)

        main_direct_list[ip] = main_direct
```

```
            deslist = [0.0] * 128
            for i in range(16):
                for j in range(16):
                    ind = (((i // 4) * 4) + (j // 4)) * 8
                    ind += int(sub_direct[i, j] * 4 / np.pi)
                    deslist[ind] += sub_norm[i, j]
            des[ip, :] = np.reshape(np.array(deslist), (1, 128))
        img_tmp = np.array(img)
        for ip in range(MaxCorner):
            begy = int(kp[ip][0][0])
            begx = int(kp[ip][0][1])
            endy = begy + int(30 * np.sin(main_direct_list[ip]))
            endx = begx + int(30 * np.cos(main_direct_list[ip]))
            img_tmp = cv2.line(img_tmp, (begy, begx), (endy, endx), 255, 2)
        #cv2.imshow("Direction", img_tmp)
        #cv2.waitKey(0)
        kplist = []
        for i in range(MaxCorner):
            x = kp[i][0][0]
            y = kp[i][0][1]
            kplist.append(cv2.KeyPoint(x, y, _size=grad_norm[int(y), int(x)]))
        return kplist, np.array(des, dtype=np.float32)
```

1.4 getIMGpyramid()

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。

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。

1. 其中使用小阈值 minMN 和大阈值 maxMN 对图片的的大小进行约束，产生图片金字塔的图片。

```
def getIMGpyramid(img):
    res = [img]
    M = img.shape[0]
    N = img.shape[1]
    minMN = min(M, N)
    maxMN = max(M, N)
    while minMN >= 96:
        nxt = cv2.resize(res[-1], (0, 0), fx=0.8, fy=0.8)
        # cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]]) -> dst

        res.append(nxt)
        M = res[-1].shape[0]
        N = res[-1].shape[1]
        minMN = min(M, N)
```

```
    while maxMN <= 960:
        nxt = cv2.resize(res[0], (0, 0), fx=1.25, fy=1.25)
        res.insert(0, nxt)
        M = res[0].shape[0]
        N = res[0].shape[1]
        maxMN = max(M, N)
    return res
```

1.5 drawKPDES()

这一函数的主要作用就是画出关键点：

```
dedef drawKPDES(img_target, kp1, des1, img_data, kpdes):
    kp2 = kpdes["kp"]
    des2 = kpdes["des"]
    shape2 = kpdes["gsize"]
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1, des2, k=2)
    nice_match = []
    for m, n in matches:
        if m.distance < 0.8 * n.distance:
            nice_match.append([m])
    M = max([img_target.shape[0], img_data.shape[0]])
    N = img_target.shape[1] + img_data.shape[1]
    img_match = np.zeros((M, N))
    kp2New = []
    ratio = shape2[0] / img_data.shape[0]
    for i in range(len(kp2)):
        x = kp2[i][0]
        y = kp2[i][1]
        s = kp2[i][2]
        kp2New.append(cv2.KeyPoint(x / ratio, y / ratio, _size=s))
    kp2 = kp2New
    img_match = cv2.drawMatchesKnn(img_target, kp1, img_data, kp2,
                                   nice_match, img_match, matchColor=[0, 0,
255], singlePointColor=[255, 0, 0])
    cv2.imshow("MyMatch", img_match)
    cv2.waitKey(0)
```

2、PreProcess.oy

2.1 getValidFileName()

此函数用以获取正确呃文件名方便接下来的处理。

```
def getValidFileName(s):
```

```
    l = []
    for c in s:
        if ((c >= '0' and c <= '9') or
                (c >= 'a' and c <= 'z') or (c >= 'A' and c <= 'Z')):
            l.append(c)
    return ''.join(l)
```

2.2 siftOnGraph()

我们把 SIFT 操作需要使用的相关函数都封装在这个函数中，方便接下来的调用。

```
def siftOnGraph(imgName):
    print("Processing {0} ...".format(imgName))
    img_data_color = cv2.imread(imgName, cv2.IMREAD_COLOR)
    img_data = cv2.imread(imgName, cv2.IMREAD_GRAYSCALE)
    dataImgL = getIMGpyramid(img_data)
    KPDESlist = []
    for j in range(len(dataImgL)):
        img_data = dataImgL[j]
        kp2, des2 = getKPandDES(img_data, 200)
        kp2list = []
        for k in range(len(kp2)):
            kp2list.append((kp2[k].pt[0], kp2[k].pt[1], kp2[k].size))
        KPDESlist.append({"kp": kp2list, "des": des2, "gsize": img_data.shape})
    pkname = os.path.join("./dataset", getValidFileName(imgName))
    pkfile = open(pkname + ".pkl", "wb")
    ob = {"img": img_data_color, "filename": imgName, "KPDESlist": KPDESlist}
    pk.dump(ob, pkfile)
    pkfile.close()
```

2.3 main 函数与运行结果

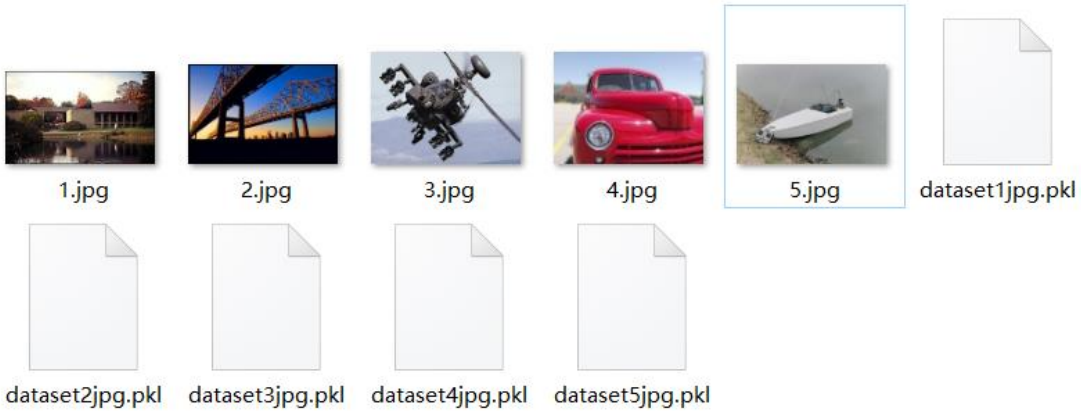
main 函数就是简单的调用，在此不多赘述：

```
if __name__ == "__main__":
    rootPath = "./dataset"
    for dirpath, dirnames, filenames in os.walk(rootPath):
        for filename in filenames:
            filename = os.path.join(dirpath, filename)
            siftOnGraph(filename)
```

运行结果如图所示，生成了 5 个可供接下来匹配的文件：

```
Processing ./dataset\1.jpg ...
Processing ./dataset\2.jpg ...
Processing ./dataset\3.jpg ...
Processing ./dataset\4.jpg ...
Processing ./dataset\5.jpg ...

Process finished with exit code 0
```



3、Match.py

3.1 countMatchNumber()

此函数用于计算匹配数。这部分使用了 cv2.BFMatcher() 中的 bf.knnMatch(des1, des2, k=2) 函数进行快速线性搜索的匹配。

```
1.  def countMatchNumber(kp1, des1, data):
2.      res = 0
3.      KPDESlist = data["KPDESlist"]
4.      for i in range(len(KPDESlist)):
5.          des2 = KPDESlist[i]["des"]
6.          shape2 = KPDESlist[i]["gsize"]
7.          bf = cv2.BFMatcher()
8.          matches = bf.knnMatch(des1, des2, k=2)
9.          nice_match = []
10.         for m, n in matches:
11.             if m.distance < 0.75 * n.distance:
12.                 nice_match.append([m])
13.         if len(nice_match) > res:
14.             res = len(nice_match)
15.     print("Matched Points:", res)
16.     return res
```

3.2 showBestMatch()

此函数可以将目标图片与处理好的所有图片主动比较，并选出最为匹配的一张图片：

```
1.  def showBestMatch(img_target, kp1, des1, dataID):
```

```
    f = open(dataID, "rb")
    data = pk.load(f)
    f.close()
    img_data = data["img"]
    imgName = data["filename"]
    print()
    print("Match Best: '{0}' ".format(imgName))
    KPDESlist = data["KPDESlist"]
    maxNum = 0
    maxID = 0
    bf = cv2.BFMatcher()
    for i in range(len(KPDESlist)):
        des2 = KPDESlist[i]["des"]
        shape2 = KPDESlist[i]["gsize"]
        matches = bf.knnMatch(des1, des2, k=2)
        nice_match = []
        for m, n in matches:
            if m.distance < 0.8 * n.distance:
                nice_match.append([m])
        if len(nice_match) > maxNum:
            maxNum = len(nice_match)
            maxID = i
    drawKPDES(img_target, kp1, des1, img_data, KPDESlist[maxID])
```

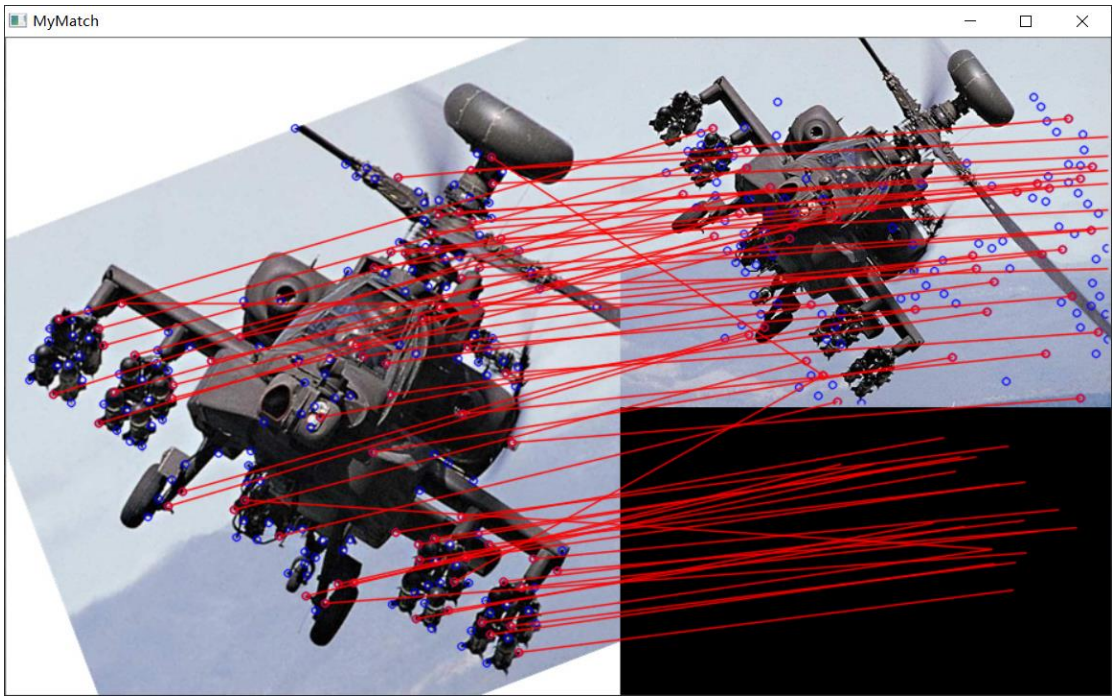
3.3 main 函数与运行结果

主调函数如下:

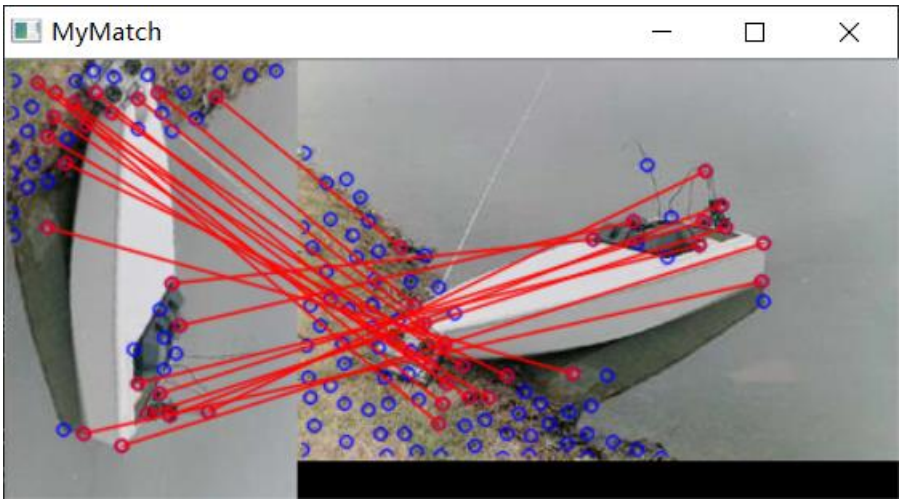
```
if __name__ == "__main__":
    tofind = "target.jpg"
    img_target_color = cv2.imread(tofind, cv2.IMREAD_COLOR)
    img_target = cv2.imread(tofind, cv2.IMREAD_GRAYSCALE)
    kp1, des1 = getKPandDES(img_target, 200)
    dataListName = []
    rootPath = "./dataset"
    print("Reading dataset ... ")
    for dirpath, dirnames, filenames in os.walk(rootPath):
        for filename in filenames:
            filename = os.path.join(dirpath, filename)
            if filename.endswith("pkl"):
                dataListName.append(filename)
    print("Done.")
    maxMatch = 0
    matchID = ""
    for dataname in dataListName:
```

```
        f = open(dataname, "rb")
        data = pk.load(f)
        f.close()
        # cv2.imshow("IMG", data["img"])
        # cv2.waitKey(0)
        imgName = data["filename"]
        print("Comparing with {0} ...".format(imgName))
        count = countMatchNumber(kp1, des1, data)
        if count > maxMatch:
            maxMatch = count
            matchID = dataname
    print()
    print("Number of Best Matched KPoints:", maxMatch)
    showBestMatch(img_target_color, kp1, des1, matchID)
```

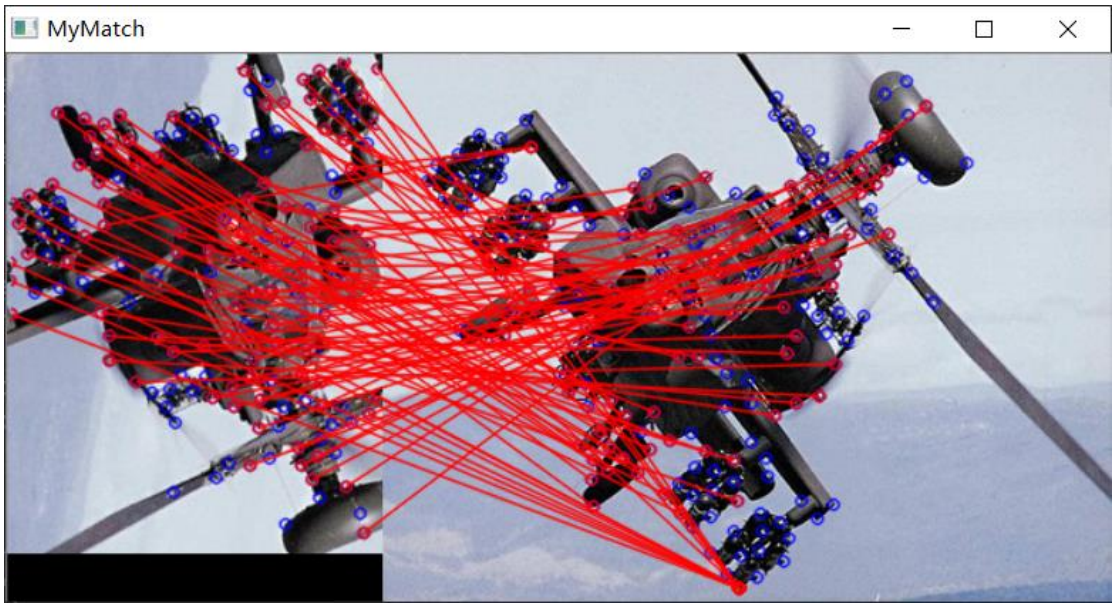
匹配结果如图所示：



能看出匹配的效果，但是结果并不是非常理想，我们另外选择其它经过处理的图片进行匹配：



可见这张图片的处理效果就十分喜人，我们再次进行尝试：



特征点识别既多又准，实验效果是很好的。经检验，大部分情况下，该算法都有较好的匹配能力。

三、实验总结

1、实验概述

本次实验的主要任务，可以总结为自己实现 SIFT 图像特征提取的算法，并观察实验的结果。

2、感想总结

在这次的实验中，学会了的东西有很多，其中最重要的就是提高了自己处理问题、收集相关资料、解决问题的能力，这在我们将来的学习和工作生活中都是很重要的。而具体细化开来，在本次实验中：

2.1 学会了自己动手设计一个 SIFT 图像特征提取算法。

2.2 了解了尺度空间、SIFT 描述子、特征点等的相关概念以及相关应用。

3、创新点

相比于原实验中直接进行两张图片之间的比较，我的代码额外实现了处理好所有图片生成图片库后，再针对用户选择的图片，自主进行分析，并自动找出最匹配的图片这一功能，这样一来，这个程序的可用性与实用性就提高了许多，并且识别效果也很好。