Lab12-report

SIFT 图像特征提取

杨弘博- 518030910401

2019-12-5

INDEX PAGE

目录

1，实验准备

1.1，实验环境

在本次实验中，我采用的是 Windows10-1903+PyCharm+python3.7+opencv+numpy 环境。（由于 opencv 轻量且适用性强，我们可以在很多系统及语言下使用。比如 Linux，Win10 下的 Java，python，C++等环境）

numpy 库提供了多样的数组处理工具。而以它为基础的 Opencv 库以轻量，功能强大著称。在本次实验中，我们将利用 opencv 库中多样的库函数进行多样的图像操作。同时，为了进行角度计算，我们还需要引入 math 函数库。

```
import cv2
import numpy as np
from math import sin, cos, pi
import random
```

1.2，实验目的

在本次实验中，只有一个任务：根据课件提供的 SIFT 图像特征提取的思路，不直接调用 cv2 库中的 SIFT 函数，对数据集中的 5 张图片进行特征的提取，并与 target.jpg 图像进行特征匹配。输出匹配结果。

下图是本次实验建立的 project 的内容。我先在 base_sift.py 中实现了使用库函数 SIFT 实现的匹配。了解效果后，建立_sift.py，根据 PPT 思路完成了这一过程。下面我将详细介绍_sift.py 中的实现过程。（base_sift.py 文件仅仅完成了目标图像和原图像的匹配，故只上交，不再做详细介绍。）

1.3，实验原理

在本次实验中，我们将利用轻量而强大的 opencv 库和 python3 来实现 SIFT 图像特征提取。实验整体思路如下：

1，利用 cv2.resize 函数对目标图像进行缩放；
2，利用 cv2.goodFeaturesToTrack 函数提取 Harris 角点；
3，计算图像梯度，得到各点梯度强度和方向；
4，计算关键点主方向；
5，统计 SIFT 描述子；
6，对数据集中的图像的特征点和目标图像的特征点对比，画线匹配。

具体代码的实现思想请看下面的实验过程分享。

2，实验过程

Step01：缩放图像

在之前的两次实验中，我们已经对读入图像的方式很熟悉了。在本次实验中，由于涉及到需要将两张图合在一起，以展示效果，我们需要首先以彩色方式读入。其中 tgt 代表目标图像，imgset 代表数据集。

```
### SIFT ###
tgt0 = cv2.imread(r"target.jpg", 1)
imgset0 = [cv2.imread("%d.jpg" % i, 1) for i in range(1, 6)]
```

接着需要记录原图的尺寸，紧接着进行缩放：

```
r0,c0,a0=np.shape(tgt0)
times=1.0
resized_tgt0=cv2.resize(tgt0,(int(r0*times),int(c0*times)))
```

在最后的结果展示阶段，我会取一系列的缩放倍数，对比匹配效果。
第一步完成。

Step02：角点提取

进行后续操作，仍需要使用灰度图像，在此定义转化函数：

```
def _Gray(img):
    x, y, z = np.shape(img)
    gray = np.zeros([x,y], "uint8")# 2 unsigned int 8
    for i in range(x):
        for j in range(y):
            gray[i][j] = np.dot(np.array(img[i][j],
                dtype="float"), [.114, .587, .299])
    return gray
```

调用：

```
# 灰度化
tgt = _Gray(resized_tgt0)
imgset = [_Gray(imgset0[i]) for i in range(len(imgset0))]
```

由于需要多次执行，我将步骤 2-5 全部写进了_Sift 函数中，并在函数中使用内联函数的样式来让代码可以复用，并整洁。

```
def _Sift(img):
```

对于第二步，可以直接调用 cv2 函数库中的 goodFeaturesToTrack 函数提取 Harris 角点，结果存为一个 int 型的二维矩阵：

```
corners = [[int(i[0][0]), int(i[0][1])]
           for i in cv2.goodFeaturesToTrack(img, 233, 0.01, 10)]
```

参数含义：读入 img 图像，返回最多 233 个角点。

角点提取完毕，第二步完成。

Step03：计算梯度幅值和角度

按照之前的步骤，这一步首先需要高斯滤波。这里注意，每进行过一次图像的转换，最好都要做一次类型的声明，以防不必要的麻烦。

```
img = cv2.GaussianBlur(img, (5, 5), 1, 1)
img = np.array(img, dtype="float")
```

计算梯度就需要相应的梯度计算函数，在上次实验中我就已经实现了多种梯度算子。在本次实验中，我先使用 sobel 算子进行了实验，但匹配的结果很糟糕。于是我更换了梯度计算的核：

```
def _Grad(img):
    x, y = r, c

    # sobel效果不好
    # s_X = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype="float")  # X方向
    # s_Y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype="float")

    kernel = np.array([
        [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]],
        [[-1, -1, -1], [0, 0, 0], [1, 1, 1]]], dtype="float") / 6
    gx = cv2.filter2D(img, -1, np.array(kernel[1]))
    gy = cv2.filter2D(img, -1, np.array(kernel[0]))
    gradient = np.zeros([x, y], "float")
    angle = np.zeros([x, y], "float")
    for i in range(x):
        for j in range(y):
            gradient[i][j] = ((gx[i][j]) ** 2 + (gy[i][j]) ** 2) ** 0.5
            angle[i][j] = np.math.atan2(gy[i][j], gx[i][j])
    return gradient, angle
```

由于第四步需要找描述子的主方向，我们加上返角度 angle，以方便后面计算。这个函数的实现完完全全和上次一样，就不多介绍了。

对比上一次的梯度函数，尝试使用了 filter2D 函数来简化代码:

```
def sobel_my(img):
    r, c = img.shape
    new_image = np.zeros((r, c))
    new_imageX = np.zeros(img.shape)
    new_imageY = np.zeros(img.shape)
    s_X = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])      # X方向
    s_Y = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
    for i in range(r-2):
        for j in range(c-2):
            new_imageX[i+1, j+1] = abs(np.sum(img[i:i+3, j:j+3] * s_X))
            new_imageY[i+1, j+1] = abs(np.sum(img[i:i+3, j:j+3] * s_Y))
            new_image[i+1, j+1] = (new_imageX[i+1, j+1]*new_imageX[i+1,j+1]
                                   + new_imageY[i+1, j+1]*new_imageY[i+1,j+1])**0.5
    # return np.uint8(new_imageX)
    # return np.uint8(new_imageY)
    return np.uint8(new_image), np.uint8(new_imageX), np.uint8(new_imageY)  # 无方向
```

调用梯度计算函数:

```
gradient, angle = _Grad(img)
```

第三步完成。

Step04：计算关键点主方向

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，我们来计算关键点的主方向。由上一步得到了梯度幅值和角度。设角点总数为 length，邻域的大小是 bins。

```
gradient, angle = _Grad(img)
bins = (r + c) // 80 #to vote

length = len(corners)
```

定义内联函数_Vote 来计算并存储每个角点的主方向：其中 voting 数组下标是各个 bin，值为各个 bin 的得票数。

```
def _Vote():
    direct = [] # 存储每个角点的主方向
    for corner in corners:
        y, x = corner
        voting = [0 for i in range(37)]
        for i in range(max(x-bins,0), min(x+bins+1,r)):
            for j in range(max(y-bins,0), min(y+bins+1,c)):
                k = int((angle[i][j]+pi) / (pi/18) + 1)
                if k >= 37:
                    k = 36
                voting[k] += gradient[i][j]
        # find max
        p=1
        for i in range(2,37):
            if voting[i]>voting[p]:
                p=i

        direct.append((p/18 - 1- 1/36) * pi)
    return direct

direct = _Vote()
```

对图像内，在角点周围的空间各个像素点进行遍历，计算出代表方向的 bin 值 k，然后以幅值 gradient 为权值为其所在的 bin 投票。最后遍历 voting 数组，找到有最高票数的，来作为该关键点的主方向。最后要记得把 k 换回弧度数的形式。

结果返回一个数组，存储每一个关键点的主方向信息。

第四步完成。

Step05：统计 SIFT 描述子

我们依然选取动态（对于不同图片而言）的邻域大小。在得到了各关键点的主角度后，先将之前计算的梯度方向有图像坐标系换算到物体坐标系，即：

```
def _theta(x, y):
    if (x < 0 or x >= r) or (y < 0 or y >= c):
        return 0
    dif = angle[x][y] - theta
    return dif if dif > 0 else dif + 2 * pi
```

上面的函数中进行了对 x 和 y 的检验，并作为内联函数定义在_Feature 函数里：其中 pos 是存放 x 和 y 的二元数组。

```
def _Feature(pos, theta):
    def _theta(x, y):
```

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，我采用的方法是相对精度更高的双线性插值。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。按照左边的算法实现了右边的程序：

$$\begin{aligned}\theta(x',y') = {} & \theta(x,y)*dx2*dy2 \\ & +\theta(x+1,y)*dx1*dy2 \\ & +\theta(x,y+1)*dx2*dy1 \\ & +\theta(x+1,y+1)*dx1*dy1\end{aligned}$$

```
def _DB_linear(x, y):
    xx, yy = int(x), int(y)
    dy1, dy2 = y-yy, yy+1-y
    dx1, dx2 = x-xx, xx+1-x
    val = _theta(xx,yy)*dx2*dy2 \
          + _theta(xx+1,yy)*dx1*dy2 \
          + _theta(xx,yy+1)*dx2*dy1 \
          + _theta(xx+1,yy+1)*dx1*dy1
    return val
```

接下来，我们要按照求主方向的方式，将 360°分成 8 个 bin 来进行投票。下面定义的是投票的方法，后面还需要对其进行多次调用，故封装。

```
val = []
def cnt(x1, x2, y1, y2, xsign, ysign):
    voting = [0 for i in range(9)]
    for x in range(x1, x2):
        for y in range(y1, y2):
            dp = [x * xsign, y * ysign]
            p = H * dp[0] + V * dp[1]
            bin = int((_DB_linear(p[0]+x0, p[1]+y0))//(pi/4) + 1)
            if bin > 8:
                bin = 8
            voting[bin] += 1
    return voting[1:]
```

其中 H 和 V 定义为将坐标系旋转所乘的算子：为了保证特征矢量具有旋转不变性，需要以特征点为中心，将特征点附近邻域内(mσ(Bp+ 1)√2 x mσ(Bp+ 1)√2)图像梯度的**位置**和**方向**旋转一个方向角 θ，即将原图像 x 轴转到与主方向相同的方向。

```
    y0, x0 = pos
    H = np.array([cos(theta), sin(theta)])
    V = np.array([-sin(theta),cos(theta)])
```

上面函数返回 voting 的切片，即各个 bin 得票的有效部分。接下来调用上面的 cnt 函数，叠加得到最终的一列值。_Feature 函数结束。

```
    bins = (r + c) // 150
    for xsign in [-1,1]:
        for ysign in [-1,1]:
            val += cnt(0, bins, 0, bins, xsign, ysign)
            val += cnt(bins, bins*2, 0, bins, xsign, ysign)
            val += cnt(bins, bins*2, bins, bins*2, xsign, ysign)
            val += cnt(0, bins, bins, bins*2, xsign, ysign)
    return val
```

但是 sift 算法还没有完，我们还需要对数据进行最后一步处理，完成_Sift 函数的接口：

```
feature = []
for i in range(length):
    val = _Feature(corners[i], direct[i])
    m = sum(k * k for k in val) ** 0.5
    l = [k / m for k in val]
    feature.append(l)
return feature, corners, length
```

最终返回每一个特征点的一列 sift 描述子，特征点，和特征点的个数。

第五步完成。到这里，基本的 sift 提取特征的算法就完成了。接下来，需要在图像间建立匹配，并画线表示出来。

Step06：匹配特征点

要想在一张图里画出匹配相应特征点的连线，就必须先要把目标图像和数据集里的图在一个窗口里表示出来。由于数据集比较小，我对所有的 5 张图片都调用了下面的函数，生成 5 个合在一起的图。在函数_Merge 中，读取两张图片的长宽，将两个图片合在一起，用黑色将图片补成矩形：

```
def _Merge(img1, img2):
    h1, w1 ,a= np.shape(img1)
    h2, w2 ,a= np.shape(img2)
    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])
    elif h1 > h2:
        extra = np.array([[[0,0,0] for i in range(w2)] for ii in range(h1-h2)])
        img2 = np.vstack([img2, extra])
    return np.hstack([img1,img2])
```

接下来我们来进行匹配。首先分别对目标图像和数据集中的图像调用_Sift 函数可以得到它们的特征点信息：

```
ff = []
cc = []
ll = []
ft, ct, lt = _Sift(tgt)
for i in range(len(imgset)):
    f, c, l = _Sift(imgset[i])
    ff.append(f)
    cc.append(c)
    ll.append(l)
```

定义函数_Match，完成特征点的匹配。对图片集中的每一个图像，定义一个 cnt 来记录匹配并大于阈值的特征点个数。具体来说，对两张图片的每一个特征点进行 n**2 的遍历，将内积的结果存入 tmp，含义是两个 SIFT 描述子的相似度。再将每一个目标图像描述子的最佳匹配点放进数组 x。

```
def _Match(threshold):
    for id in range(len(imgset)):
        x = []
        cnt = 0
        for i in range(lt):
            tmp = []
            for j in range(ll[id]):
                sc= np.inner(np.array(ft[i]), np.array(ff[id][j]))
                tmp.append(sc)
            x.append([tmp.index(max(tmp)), max(tmp)])
```

紧接着进行阈值检验：

```
        for a in range(len(x)):
            b, s = x[a]
            if s < threshold:
                continue
```

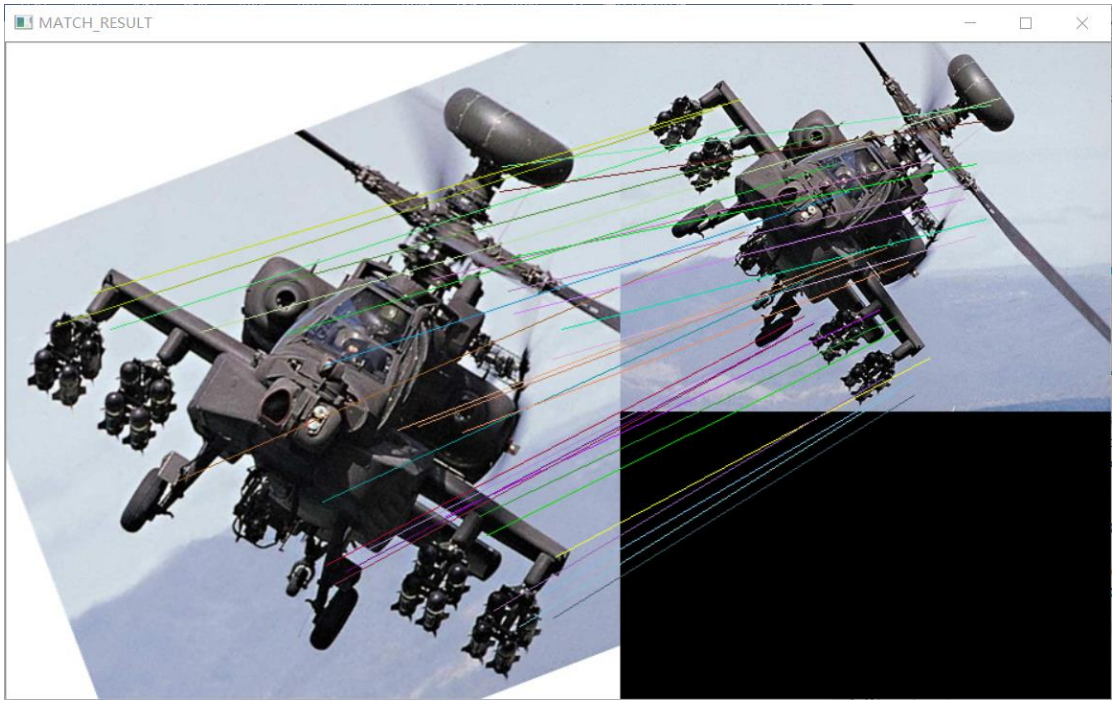
对于满足条件的点，计数：cnt+=1

如果有 6 个以上的点可以匹配（6 是我设置的参数），则可以认为两张图片可以匹配。画线连接融合图中的相应特征点，画出结果图：

```
            cnt += 1
            color = ((random.randint(0, 255)),
                     (random.randint(0, 255)),
                     (random.randint(0, 255)))
            cv2.line(mgimgs[id], tuple(ct[a]),
                     tuple([cc[id][b][0] + w,
                            cc[id][b][1]]), color, 1)
    if cnt > 6:
        cv2.imwrite("match%d.jpg" % id, mgimgs[id])
        print("MATCHED %d" % id)
        img = np.array(mgimgs[id], dtype="uint8")
        cv2.namedWindow("MATCH_RESULT")
        cv2.imshow("MATCH_RESULT", img)
        cv2.waitKey(0)
        cv2.destroyWindow("MATCH_RESULT")

    else:
        print("NOT %d" % id)
```

到这里，实验要求的算法就全部完成了。下面放上设置缩放倍数为一，阈值设为 0.8 时的匹配效果：



可以看到，效果还是很好的。我将在 PLUS 部分调整缩放比例，看一看匹配效果有什么变化。

实验成功!

3，实验总结

PLUS1：问题与解决

1，opencv 成员函数 imshow 报错

这里如果报出错误：

```
size.width>0 && size.height>0 in function 'cv::imshow'
```

就肯定是 imread 函数传入的路径出了问题。先检查文件名，再检查路径是否正确。如果还不行，换成绝对路径试试看。如果还是不行的话，把路径中的‘/’符号换成 windows 下的‘\\’。问题解决。（这里主要是因为在字符串中，\号会结合后面字符，解释为转义字符）

2，参数和返回的数量不符

如果函数书写比较混乱，就相对容易有这种错误。对于多个返回值的函数，一定要有相应数量的变量来接收。举例来说，如果一开始以灰度读入图片，那么调用 np.shape 函数就只能返回二元组，用三个变量接收就显然不对了。

本次实验本身比较难，再加上对 numpy 还不是很了解，还是很有难度的。但会卡住的 bug 并不多，都是我们细心就能解决的问题。

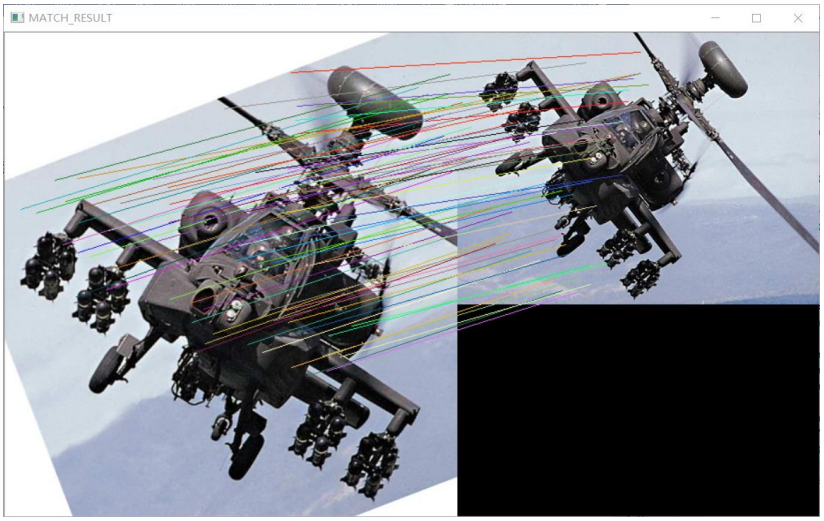
PLUS2：拓展性的工作

探究合适的缩放倍数

这里设置表现较好的 threshold=0.8 不变，更改缩放倍数 times=0.8，0.6，1.2 观察匹配效果及其变化：

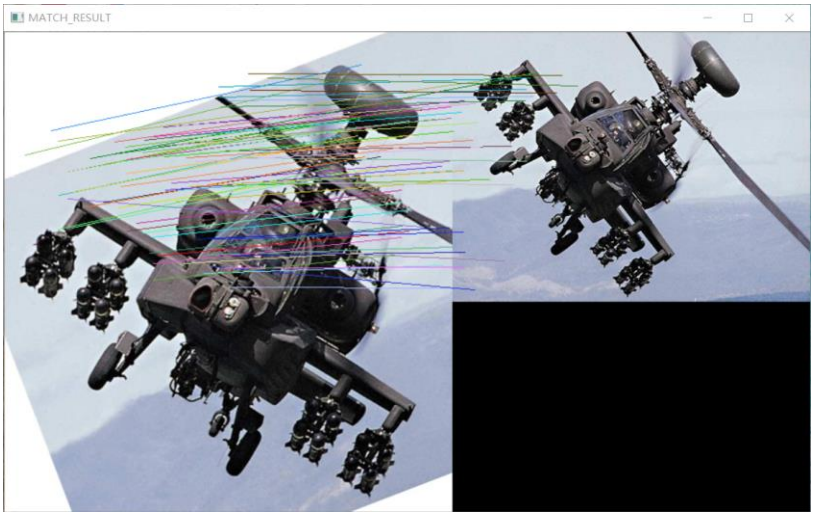
1，设置【times=0.8】

可以明显看到，匹配成功的点数比 threshold=1 时多很多。乍一看上去，齐刷刷的，但是仔细看上去，有一些点都配错了。



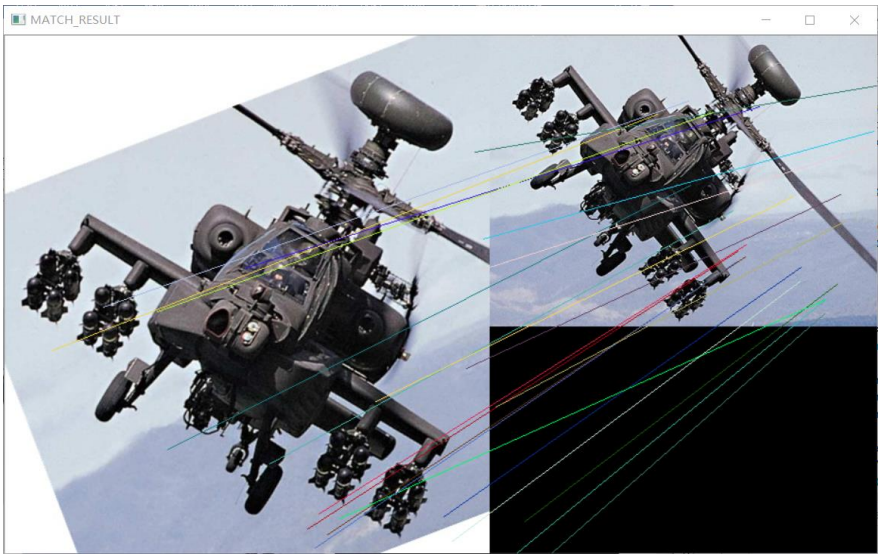
2， 设置【times=0.6】

现在看起来，进一步压缩后，图片中匹配成功的点更多了。但是画出的匹配线却比缩放 0.8 倍时误差更大。



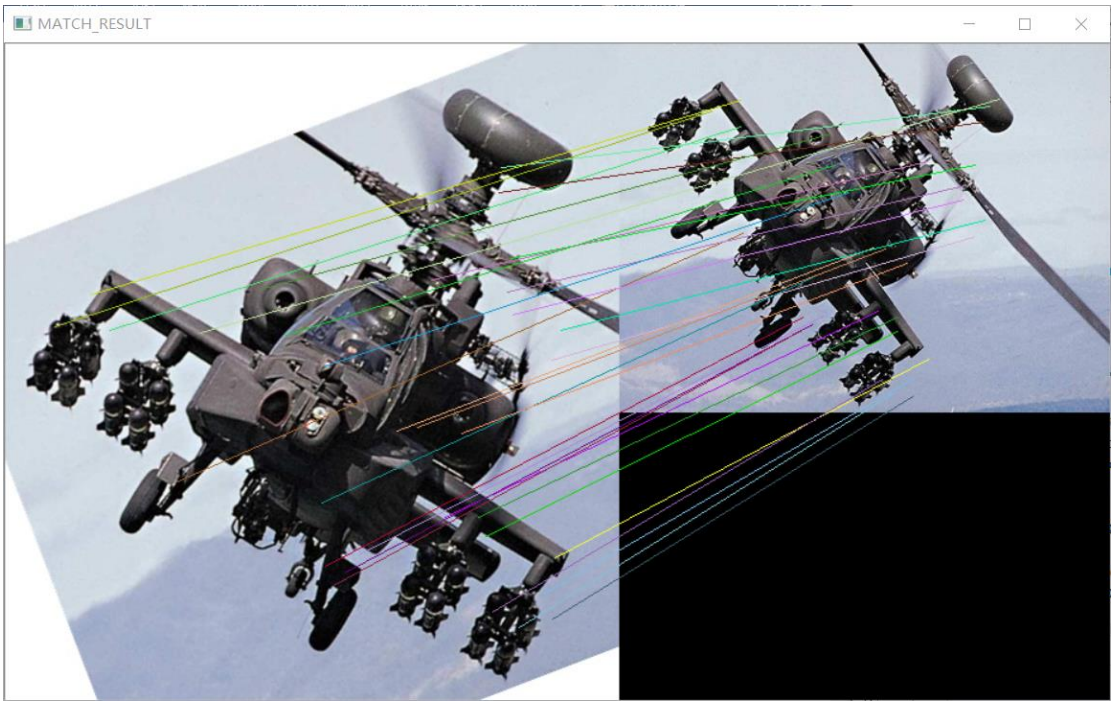
3， 设置【times=1.2】

可以看到，匹配点变少了，而且误差也很大。



根据我多尝试的几组数据。一般来说，times 较小时，匹配点较多，反之则相反。综合来看，反而是 times=1 时匹配效果最好。由于我们前面提取角点用的是近似的方法，有这样的误差应该也是预料之内的情况。

最后，再放一次 times=1 时的匹配结果：



可以看见，这个目标图像在不进行缩放时，有着很好的匹配效果。基本上可以全部做到对应点的正确匹配。

编写调用库函数 SIFT 的程序进行结果对比

由于新版本的 opencv 库里已经没有 SIFT 函数了，要想编写调用 SIFT 的程序，需要重新安装旧版本的 opencv 库：

```
C:\Users\TCfatasy>pip show opencv-python
Name: opencv-python
Version: 3.4.2.16
Summary: Wrapper package for OpenCV python bindings.
Home-page: https://github.com/skvark/opencv-python
Author: None
Author-email: None
License: MIT
Location: c:\tc-prog\tc-python\lib\site-packages
Requires: numpy
Required-by:
```

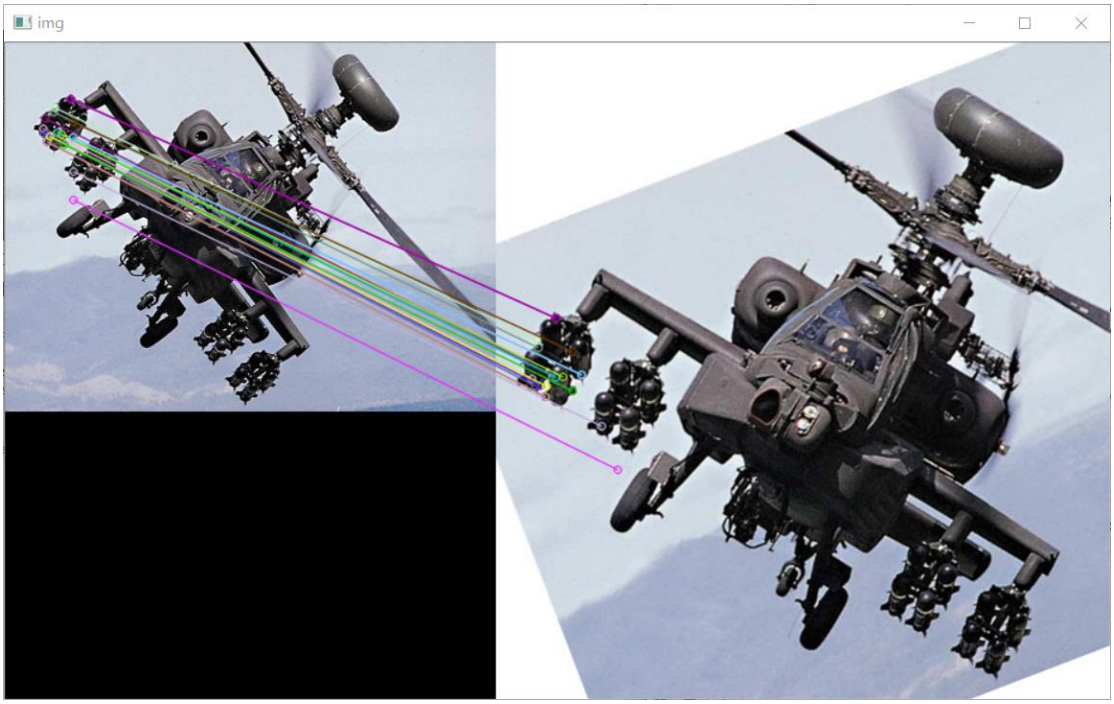
中心程序是这几行，计算和探测图片里的特征点。然后对描述子做匹配。（实际这个过程和我们上面写出来的是一样的）

```
# 这个库需要将opencv版本倒回
sift=cv2.xfeatures2d.SIFT_create()

kp1, des1 = sift.detectAndCompute(gray_orign, None)
kp2, des2 = sift.detectAndCompute(gray, None)

bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)
```

其他的我就不往上贴了，库函数运行结果如下：



可以看到，这才是做到了真正的每个点都对应，我们编写的函数还有很大差距。

实验体会和收获

Lab12 的实验内容十分深刻。在这次实验中，我们练习在算法层面上实现了 SIFT 图像特征提取这一在之前看上去十分高深的课题。而且我实现的效果还算比较理想，很接近库函数的计算结果。我深刻的体会到，需要学习的东西还有很多。在平时的学习中，在 Lab 的完成中，都要多学多看多练，提高自己的水平！相信在之后的学习中，我还会有更多更丰富的收获！

脚踏实地做练习，才可以更深刻的理解。
多尝试新的内容，才可以不断收获。

最后，再次由衷感谢何大治老师和各位助教老师的悉心指导！