实 验 报 告

实验 12：SIFT 图像特征提取

517030910374
郭嘉宋

目录：

注：在编程过程中我也遇到了很多困难和 Bug，还有很多思考了很久得出的结论与解释，以及一些需要注释的内容，都在下述 “2. 实验过程” 中，用加粗的 NOTE 加以了说明。

正文:

1. 实验准备

(1) 实验环境

本实验主要采用 Ubuntu 系统下的 python2.7 进行，使用 miniconda 环境，Ubuntu 系统安装在 VMware Workstation 提供的虚拟机中，同时使用 OpenCV、Numpy 等库对图像进行分析与后续处理。

(2) 实验目的

实验 12:

了解 SIFT 算法的原理，学习关键点、Harris 角点等操作，进一步深入掌握 OpenCV 和 Numpy 等库对图像进行分析、空间尺度计算，了解获得更多的图像信息。

(3) 实验原理

实验 12:

SIFT 算法是 David Lowe 于 1999 年提出的局部特征描述子，并于 2004 年进行了更深入的发展和完善。Sift 特征匹配算法可以处理两幅图像之间发生平移、旋转、仿射变换情况下的匹配问题，具有很强的匹配能力。总体来说，SIFT 算子具有以下特性:

SIFT 特征是图像的局部特征，对平移、旋转、尺度缩放、亮度变化、遮挡和噪声等具有良好的不变性，对视觉变化、仿射变换也保持一定程度的稳定性。

独特性好，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。

多量性，即使少数的几个物体也可以产生大量 SIFT 特征向量。

速度相对较快，经优化的 Sift 匹配算法甚至可以达到实时的要求。

可扩展性强，可以很方便的与其他形式的特征向量进行联合。

SIFT 算法主要步骤:

a.检测尺度空间极值点

b.精确定位极值点

c.为每个关键点指定方向参数

d.关键点描述子的生成

尺度空间极值点提取

对图像*I(x, y*建立尺度空间的方法是将图像反复与高斯核*G(x, y, d*卷积：

$$L(x,y,d)=G(x,y,d)*I(x,y)$$

其中d是尺度参数。*G(x, y, d)*是高斯卷积核

$$G(x,y,d)=\frac{1}{2\pi d^2}\exp\left\{-\frac{x^2+y^2}{2d^2}\right\}$$

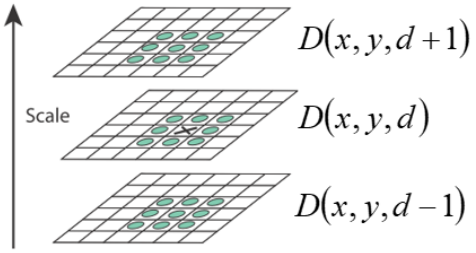
*L*在这里称作尺度空间图像。
将两个不同尺度的*L*相减可得到高斯差分图像：

$$\begin{aligned}D(x,y,d)&=G(x,y,kd)*I(x,y)-G(x,y,d)*I(x,y)\\&=(G(x,y,kd)-G(x,y,d))*I(x,y)\\&=L(x,y,kd)-L(x,y,d)\end{aligned}$$

尺度空间极值点在*D(x, y, d)*上求取

尺度空间极值点提取

对于D(x, y, d)中的某个点，如果它的值大于或小于其周围的8个点和它上下相邻尺度的18个点，则该点是一个极值点。



准确的关键点定位：
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：

$$D(x)=D+\frac{\partial D}{\partial \boldsymbol{x}}\boldsymbol{x}+\frac{1}{2}\boldsymbol{x}^T\frac{\partial^2 D}{\partial \boldsymbol{x}^2}\boldsymbol{x}$$

准确的极值点位置为上式对x导数为0的位置

$$\hat{x}=-\left[\frac{\partial^2 D}{\partial x^2}\right]^{-1}\frac{\partial D}{\partial x}\qquad D(\hat{x})=D+\frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x}$$

论文中，若*D(x)*小于0.03，则排除这个极值点。

排除边缘响应

对于上述得到的关键点，进一步排除位于边缘处的关键点，对于某点D，先计算其Hessian矩阵：

$$H=\begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

假设矩阵的特征值为α，β，则

$$\mathrm{Tr}(H)=D_{xx}+D_{yy}=\alpha+\beta$$

$$\mathrm{Det}(H)=D_{xx}D_{yy}-(D_{xy})^2=\alpha\beta$$

假设α=rβ，

$$\frac{\mathrm{Tr}(H)^2}{\mathrm{Det}(H)}=\frac{(\alpha+\beta)^2}{\alpha\beta}=\frac{(r\beta+\beta)^2}{r\beta^2}=\frac{(r+1)^2}{r}$$

若 $\frac{\mathrm{Tr}(H)^2}{\mathrm{Det}(H)}<\frac{(r_0+1)^2}{r_0}$ ，则排除这个特征点，论文中取r0=10

图像金字塔中提取角点

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。

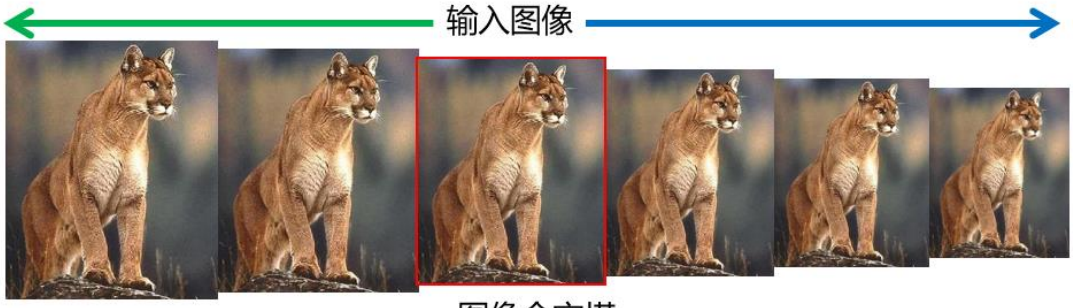
与用高斯差分构建的尺度空间不同，图像金字塔通过直接缩放图像的方式构建尺度空间。

OpenCV中改变图像尺寸：

cvResize(输入图像，输出图像)

输出图像预先申请好内存：

Img=cvCreateImage(cvSize(宽度，高度), 8, 1);



图像金字塔

Harris 角点提取

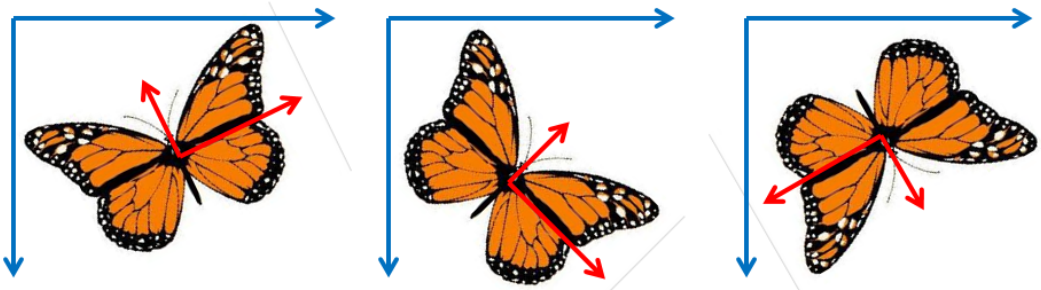
OpenCV中集成了角点检测函数FeaturesToTrack，可直接调用，对应的论文为
Jianbo Shi and C. Toamsi. Good Features to Track. CVPR. 1994.

cv2. cornerHarris(src, blockSize, ksize, k[, dst[, borderType]]) -> dst
-src: 输入的**灰度**图像；
-dst: 用于存放角点检测的输出结果，和输入图像相同大小；
-blockSize：邻域大小（blockSize=2或3）
- apertureSize ： sobel算子孔径大小（3）
-k：角点质量，一般取0.1或0.01，越小返回的角点数目越多；
C++: cornerHarris(srcImage, cornerStrength, 2, 3, 0.01);

计算Harris角点列表，
cv2.goodFeaturesToTrack(image, maxCorners, qualityLevel, minDistance[, corners[, mask[, blockSize[, useHarrisDetector[, k]]]]]) -> corners
Image:输入图像
maxCorners：允许返回的最多角点个数
qualityLevel = 0.01
minDistance = 10
blockSize = 3
k = 0.04

SIFT描述子的计算

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描述子的计算

对应论文第5节：假设某关键点为I(x0, y0)，对于它m*m领域内的每个点，计算其梯度方向和梯度强度：

$$m(x,y)=\sqrt{(I(x+1,y)-I(x-1,y))^2+(I(x,y+1)-I(x,y-1))^2}$$

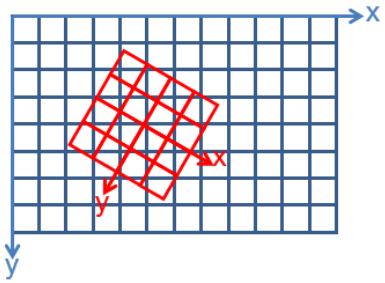
$$\theta(x,y)=\arctan\frac{I(x,y+1)-I(x,y-1)}{I(x+1,y)-I(x-1,y)}$$

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)。

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。



SIFT描述子的计算

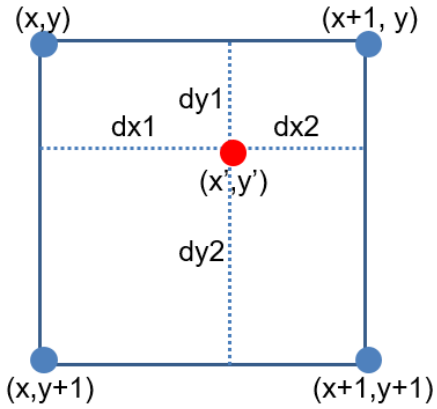
物体坐标系16*16的邻域分成4*4个块，每个块4*4个像素。
在每个块内按照求主方向的方式把360度分成8个bins，统计梯度方向直方图，最终每个块可生成8维的直方图向量，每个关键点可生成4*4*8=128维的SIFT描述子。

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：

$$\theta(x',y')=\theta(\mathrm{Round}(x'),\mathrm{Round}(y'))$$

或更精确地，可采用双线性插值：

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



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 $s(f_1,f_2)=f_1\cdot f_2$

2. 实验过程

1.1 实验步骤

首先用 cv2 读入图片，存入 myimg 变量进行操作，使用 goodFeaturesToTrack 函数提取关键点，再把原图复制两份用作之后求梯度图和角度图，如下图所示：

```
import cv2
import numpy
import math

myimg=cv2.imread("target.jpg",cv2.IMREAD_GRAYSCALE)
corner=cv2.goodFeaturesToTrack(myimg,150,0.01,10,blockSize=3,k=0.04)

num_row=len(myimg)
num_column=len(myimg[0])

gradient_img=myimg.copy()
# gradient_img=gradient_img.astype(float)
angle_img=myimg.copy()
# angle_img=angle_img.astype(float)
```

NOTE：此处注释的含义是将原来图片的 numpy.unit8 类型转换为更为精确的 float 类型，否则处理下来的梯度和角度都是整数，但在之后的调试过程中，发现其实取整操作增加了准确率，其实太精确的浮点数反而无法精确判断图片元素，故此处注释。

```
for t1 in range(num_row):
    for t2 in range(num_column):
        gradient_img[t1][t2] = float(0)
        angle_img[t1][t2] = float(0)

for k1 in range(1,num_row-1):

    for k2 in range(1,num_column-1):
        gradient_data = numpy.sqrt(pow(float(int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2])), 2)
            + pow(float(int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1])), 2))

        if (int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1])) == 0:
            if int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2]) > 0:
                angle_data = 90.0
            else:
                angle_data = 270.0
```

之后对原图的每一个像素求出梯度，如上图所示，保存在 gradient 二维矩阵中，如上图所示。

```
        else:
            if (int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2]) >= 0) \
                    and (int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1]) > 0):
                angle_data = math.atan(float((int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2])))
                                       / (int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1]))) \
                             / math.pi * 180

            if (int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2]) >= 0) \
                    and (int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1]) < 0):
                angle_data = 180 - abs(math.atan(float((int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2])))
                                                 / (int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1])))
                                       / math.pi * 180)

            if (int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2]) <= 0) \
                    and (int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1]) < 0):
                angle_data = abs(math.atan(float((int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2])))
                                           / (int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1])))
                                 / math.pi * 180) + 180
```

```
        if (int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2]) <= 0) \
                and (int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1]) > 0):
            angle_data = 360 - abs(math.atan(float((int(myimg[k1 + 1][k2]) - int(myimg[k1 - 1][k2])))
                                             / (int(myimg[k1][k2 + 1]) - int(myimg[k1][k2 - 1]))) / math.pi * 180)

    gradient_img[k1][k2] = gradient_data

    angle_img[k1][k2] = angle_data
```

NOTE：如上图所示，求角度的时候一定要对 x、y 的大小进行分类讨论，此时的角度在【0,360）范围内，而 arctan 是在（-90,90）内取值，因此需要将角度转换。

```
vector_final=[]
length=len(corner)
for i in range(length):

    y = corner[i][0][0]
    x = corner[i][0][1]

    main_direction_vote=[]
    for o in range(36):
        main_direction_vote.append(0)

    gradient=[]
    angle=[]

    x=int(x)
    y=int(y)
```

```
    for k1 in range(x-3,x+3):
        for k2 in range(y-3,y+3):
            gradient_data=gradient_img[k1][k2]
            angle_data=angle_img[k1][k2]
            gradient.append(gradient_data)
            angle.append(angle_data)
            choice = int(angle_data / 10)
            if choice==36:
                choice=0
            main_direction_vote[choice] += gradient_data
    max=main_direction_vote[0]
    direction=0
    for o in range(36):
        if main_direction_vote[o]>max:
            direction=o
            max=main_direction_vote[o]
    direction *= 10
    direction+=5
```

接下来对每一个关键点进行操作，对 0-360 以 10 度为一档投票，投出主方向，如上图所示，最后我取的是每一档的中点，即最后的加 5，其实每档中取任意哪点差别不大，都可以自由选取，此时我的投票范围是以关键点为中心 5 * 5 的矩阵范围。

```
new_gradient = []
new_angle = []
for k2 in range(- 8, 8):
    new_gradient_per_row=[]
    new_angle_per_row=[]
    for k1 in range(- 8, 8):
        y_new = k1 * math.cos(-direction / 180 * math.pi) + \
                k2 * math.sin(-direction / 180 * math.pi)
        x_new = k2 * math.cos(-direction / 180 * math.pi) - \
                k1 * math.sin(-direction / 180 * math.pi)
        x_new=x_new+x
        y_new=y_new+y

        x_up = int(x_new)
        x_down = x_up + 1
        y_left = int(y_new)
        y_right = y_left + 1
```

之后对坐标进行变换，此时我将新坐标中的 16*16 个点投影会原坐标，此时角度 Φ 应该取为负值，如上图所示，x_new 和 y_new 是原坐标中的位置。

对每个点的梯度和角度使用双线性插值，最近的四个点以一定的权重对此点进行加权计算，如下图所示：

```
try:
    new_gradient_data=gradient_img[x_up][y_left]*(x_down-x_new)*(y_right-y_new)\
                  +gradient_img[x_down][y_left]*(x_new-x_up)*(y_right-y_new)\
                  +gradient_img[x_up][y_right]*(x_down-x_new)*(y_new-y_left)\
                  +gradient_img[x_down][y_right]*(x_new-x_up)*(y_new-y_left)
    new_angle_data=((angle_img[x_up][y_left]-direction+360)%360)*(x_down-x_new)*(y_right-y_new)\
                  +((angle_img[x_down][y_left]-direction+360)%360)*(x_new-x_up)*(y_right-y_new)\
                  +((angle_img[x_up][y_right]-direction+360)%360)*(x_down-x_new)*(y_new-y_left)\
                  +((angle_img[x_down][y_right]-direction+360)%360)*(x_new-x_up)*(y_new-y_left)


except:
    new_gradient_data=0
    new_angle_data=0

new_gradient_per_row.append(new_gradient_data)
new_angle_per_row.append(new_angle_data)
```

NOTE：此时必写 try 进行尝试，因物体坐标系中的点可能在往回旋转的过程中旋转到原图之外，所以一定要做异常处理，取为 0，即不会投票。

之后把 16*16 范围分为 16 个 4*4 范围，将 360 度分为 8 个组，每 45 度一个组，其中对点的梯度角进行投票，如下图所示，此时为第一个方块的投票过程，其余范围内过程相同，详见源码。

```
    vote_per_point=[]

    final_vote=[]
    for o in range(8):
        final_vote.append(0)

    for k1 in range(0,4):
        for k2 in range(0,4):
            final_vote[int(new_angle[k1][k2])/45] += 1
    vote_per_point.extend(final_vote)
```

NOTE：本来我才用每个点的梯度值作为权值进行投票，后来发现结果效果不好，采用相同的权值 1 进行投票，效果大大提高。

```
    len_per_point=len(vote_per_point)
    sum=0
    for p in range(len_per_point):
        sum+=(vote_per_point[p])**2
    sum=math.sqrt(float(sum))+0.000001
    for p in range(len_per_point):
        vote_per_point[p]=float(vote_per_point[p])/sum

    vector_final.append(vote_per_point)
```

使用 extend 函数将每个点的信息向量合成一个 128 维向量, 最后再进行归一化, 添加到 vector_final 这个 list 中，如上图所示。

NOTE：0.00001 是对 sum 为 0 情况的异常处理，有些点旋转后大部分位于图像外，所以向量模是 0，加上小数防止分母为 0 异常。

之后对测试图进行完全一样的操作，详见源码。

比对两组结果我取阈值为 0.76，输出匹配度超过 0.76 的向量组，如下图所示：

```
pair_point=[]
pair_count=0
for i1 in range(len(vector_final)):
    for i2 in range(len(vector_final_search)):
        pair=numpy.vdot(vector_final[i1],vector_final_search[i2])
        if pair>=0.76:
            print pair
            pair_point.append(i1)
            pair_point.append(i2)
            pair_count+=1
```

之后就是可视化部分，使用 numpy.zero 初始化一个 numpy 矩阵，使用 cv2 自带的库 line 和 circle 进行绘图，如下图所示。

```
row1=num_row
column1=num_column
row2=num_row_search
column2=num_column_search
answer_row=numpy.maximum(row1,row2)
result=numpy.zeros(shape=(answer_row,column1+column2,3),dtype=numpy.uint8)
for i1 in range (row1):
    for i2 in range(column1):
        result[i1][i2]=myimg[i1][i2]

for i1 in range(row2):
    for i2 in range(column2):
        result[i1][i2+column1]=myimg_search[i1][i2]

for t in range(pair_count):
    cv2.line(result,(int(corner[pair_point[2*t]][0][0]),int(corner[pair_point[2*t]][0][1])),
    (int(corner_search[pair_point[2*t+1]][0][0])+column1,int(corner_search[pair_point[2*t+1]][0][1]
     ,(0,0,255),1)
```

```
myimg_print = numpy.zeros(shape=(row1, column1, 3), dtype=numpy.uint8)
for i1 in range (row1):
    for i2 in range(column1):
        myimg_print[i1][i2]=myimg[i1][i2]
for t in range(pair_count):

    cv2.circle(myimg_print,(corner[pair_point[2*t]][0][0],corner[pair_point[2*t]][0][1]),
               10,(0,0,255),2)
cv2.imshow('myimg',myimg_print)

myimg_search_print = numpy.zeros(shape=(row2, column2, 3), dtype=numpy.uint8)
for i1 in range (row2):
    for i2 in range(column2):
        myimg_search_print[i1][i2]=myimg_search[i1][i2]
for t in range(pair_count):
    cv2.circle(myimg_search_print,(corner_search[pair_point[2*t+1]][0][0],
                                   corner_search[pair_point[2*t+1]][0][1]),10,(0,0,255),2)
```

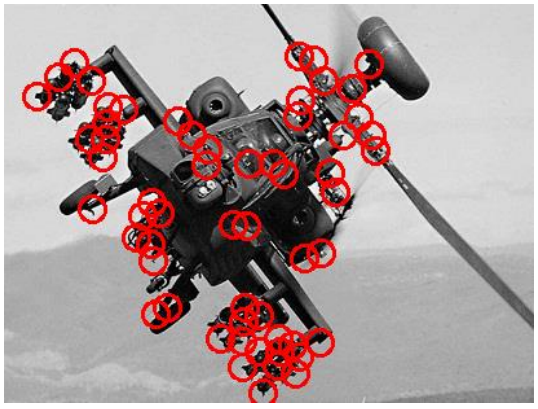
最后再输出几幅图，如下图所示：

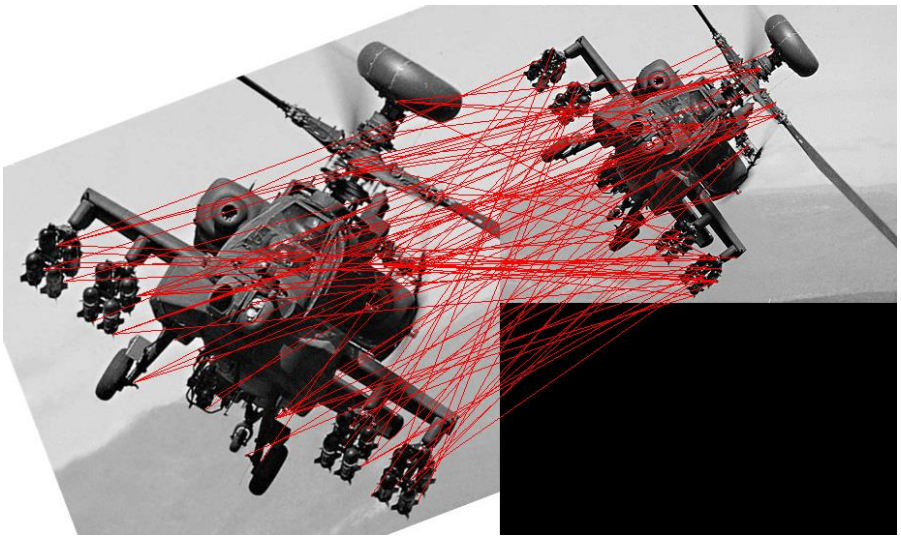
```
cv2.imshow('result',result)

cv2.imshow('myimg_search',myimg_search_print)
# cv2.imwrite('result',result)
```

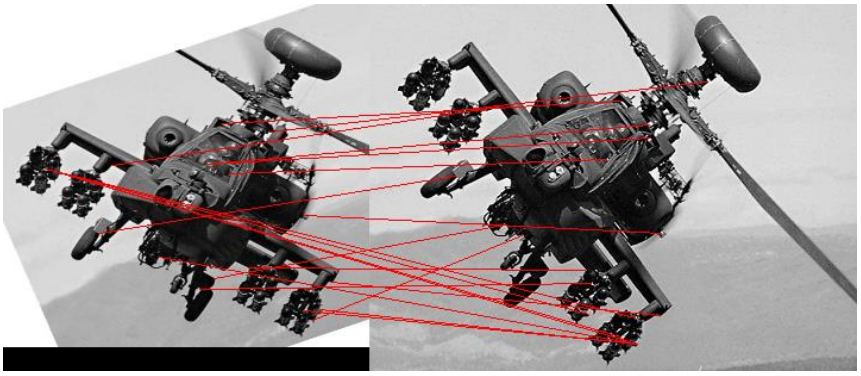
1.2 实验结果：

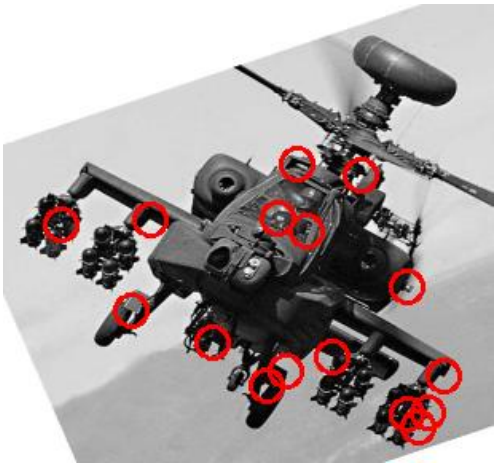
红色圈圈出了匹配的关键点，最后使用直线连接，如图：

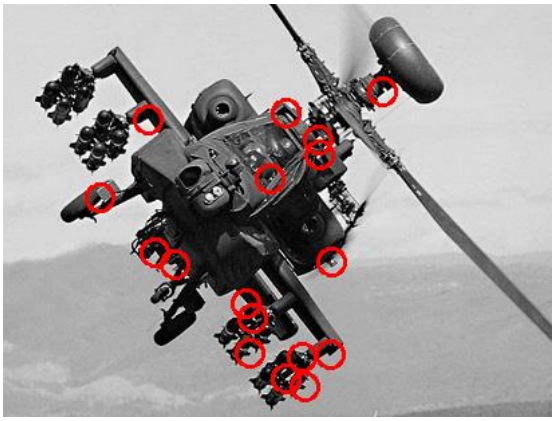




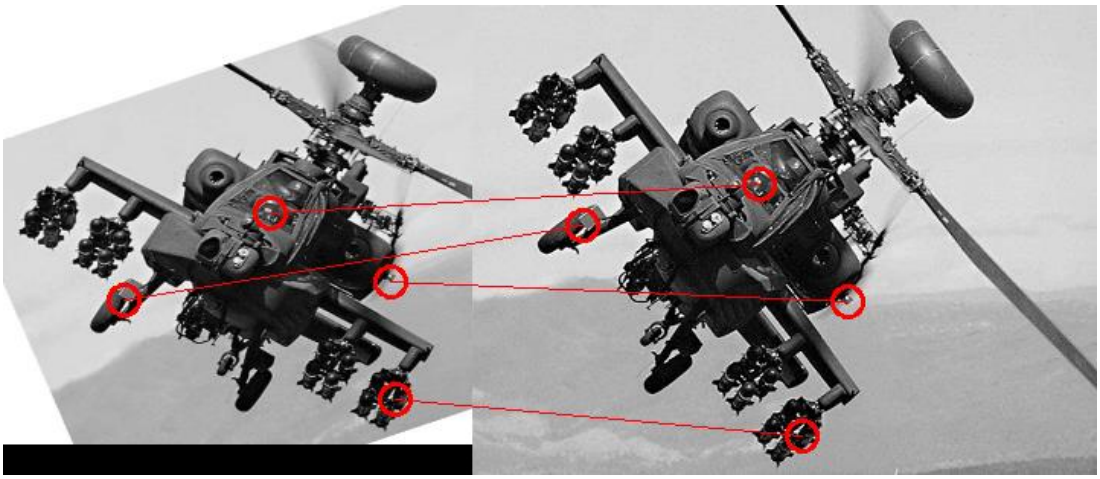
这里我们可以看到大体上是吻合的，但是仍有一部分不匹配的点。
之后我使用 resize 函数对结果进行优化，对相同大小的图进行匹配，结果如下：







可见正确率大大提高，我们再对阈值和关键点个数进行参数调节，最终结果如图:



经过调参后结果非常准确，如上图所示，实验较为成功。但可以看出 SIFT 其实是一个不太稳定的算法，还是需要图像金字塔操作进一步提高准确率。

3. 实验总结

本次实验我的匹配结果调参过后非常准确，实验较为成功，但改进的地方仍有很多。

我先分析了数据不匹配的原因，旋转图片之后，其实很多不匹配的点经放大后查看十分接近，只是在人的直观判断下宏观是不同的点，可在微观其实十分接近。同时直升机这张图对比过于强烈，梯度差距大的点非常多，容易造成不匹配。

之后的优化工作可以加入图像金字塔增加大部分点的准确率，同时这次实验也让我意识到图像处理的不容易，信息这么多，过滤处理之后就连 SIFT 算法仍有很多误差点，图片在旋转翻折变换后确实放大看同一个部位也变得不一样，识别则更为困难，需要进一步学习研究。

此次试验让我对 OpenCV 的了解加深了，上学期计算导论使用过 OpenCV 制作过人脸识别，对 OpenCV 的有一定的认识，这次试验让我学习到了 SIFT 算法的用处，希望在以后的学习过程中继续了解图像处理技术。

注：在编程过程中我也遇到了很多困难和 Bug，还有很多思考了很久得出的结论与解释，以及一些需要注释的内容，都在下述“2. 实验过程”中，用加粗的 NOTE 加以了说明。源码见附带 lab12.py 文件即可。

继续努力！

o(*≧▽≦)ツ！

517030910374

郭嘉宋