电工导实验报告 8

一、实验目的

1. SIFT 图像特征提取

二、实验内容

1. 了解 SIFT 算法
2. 通过图像特征匹配图像

三、实验环境

1. Python 2.7 + opencv +numpy

四、实验原理

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。

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

1. 图像关键点(keypoint)的提取：

   1.尺度空间极值（较为复杂）

   2.Harris 脚点：

   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

2. SIFT 描述子的计算：

   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。

   *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。

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

五、算法实现及流程简介

SIFT描述子过于复杂，故采用内置SIFT函数。

首先读入target以及要匹配的多张图片，并放到list里方便后期处理。

```
target = cv2.imread('target.jpg')
img1 = cv2.imread('1.jpg')
img2 = cv2.imread('2.jpg')
img3 = cv2.imread('3.jpg')
img4 = cv2.imread('4.jpg')
img5 = cv2.imread('5.jpg')
imgs = [img1,img2,img3,img4,img5]
```

整个程序主要分成两部分，用两个函数实现，一个是 match 匹配函数，另一个是 display 展示结果的函数。

在 match 函数中，先通过内置的 detectAndCompute 函数来获取 target 的 keypoints 和 descriptors 作为基准。

```
#detectAndCompute()->
#keypoints:The input/output vector of keypoints
#descriptors - The output matrix of descriptors
kp_targ, des_targ = sift.detectAndCompute(target,None)
```

然后通过 for 循环对 imgs 中所有 img 进行同样的操作，与 target 得到的 descriptors 进行比较，返回可能匹配的点对。

```
for img in imgs:
    kp, des = sift.detectAndCompute(img,None)
    matched = flann.knnMatch(des,des_targ,k=2) #和target比较
    #print matched
```

但是有许多点是我们并不需要的，通过加以限定来提取相似度高的关键点，并记录下来。

```
accept = []
for m,n in matched:
    if m.distance < 0.1*n.distance: #限定来找相似点 0.08
        accept.append(m)
print len(accept)
```

如果 img 匹配的点对数目比之前记录的多，代替之前的

```
if len(accept)>max_num: #img匹配的数量更多 刷新
    max_num = len(accept)
    best_img = img
    best_kp = kp
    best = accept
```

match 函数最后返回匹配点数最多图像以及它的特征点和描述符。

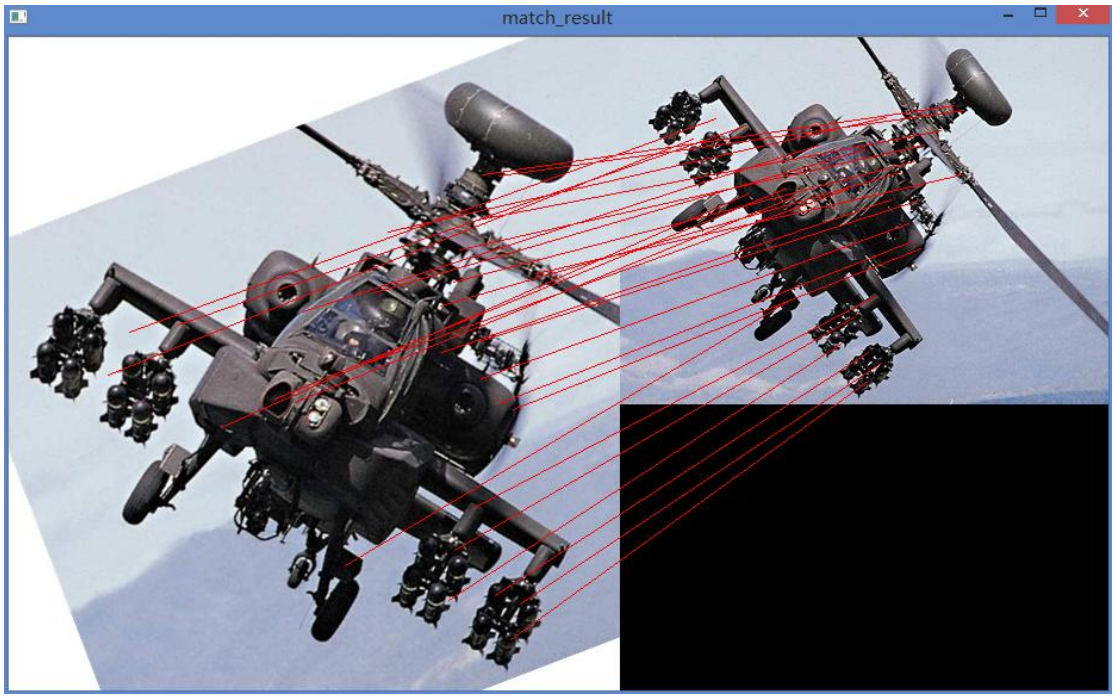
在 display 函数中，首先要将 target 和与之最匹配的图放到同一幅图中，方便后续操作。高取大的，宽取和，并排展示：

```
h1, w1 = target.shape[:2]
h2, w2 = best_img.shape[:2]
result = np.zeros((max(h1, h2), w1 + w2, 3), np.uint8)
result[:h1, :w1] = target[:,:]
result[:h2, w1:] = best_img[:,:]
```

最后将匹配到的关键点用线联结起来：

```
#将匹配到的关键点连起来
for m in best:
    color = (0,0,255)
    cv2.line(result, \
    (int(best_kp[m.queryIdx].pt[0]+w1),int(best_kp[m.queryIdx].pt[1])) , \
    (int(kp_targ[m.trainIdx].pt[0]), int(kp_targ[m.trainIdx].pt[1])), color)
```

六、运行效果

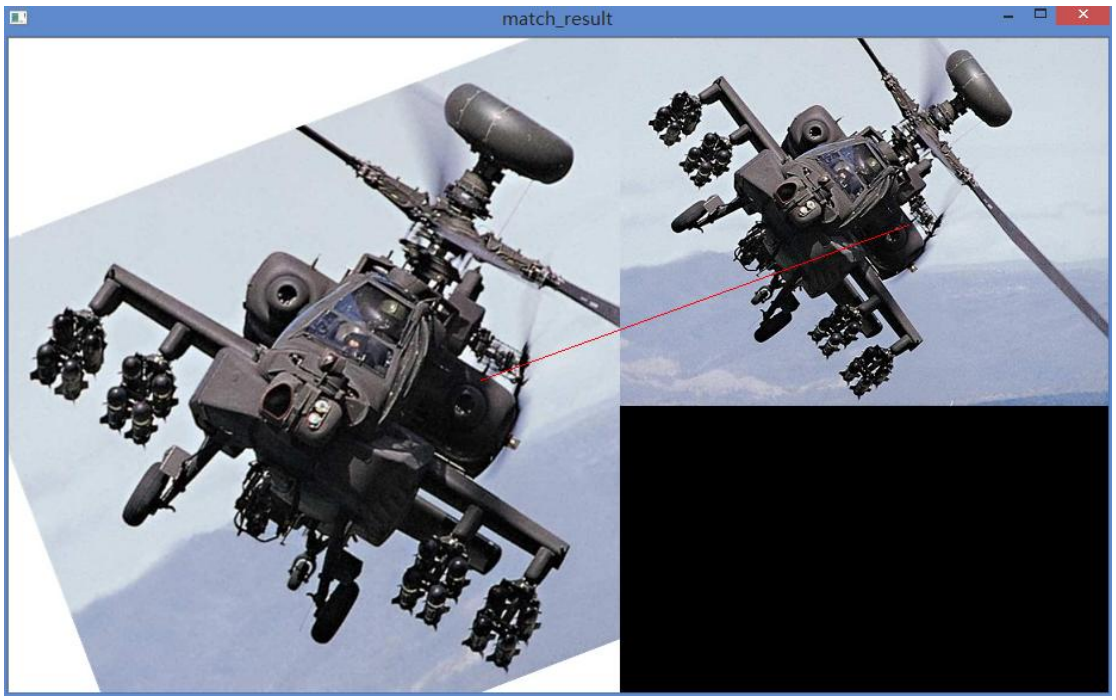


效果很不错，各个点基本都匹配良好，可以通过之前判断来进一步筛选匹配的对应点。

例如，将

```
accept = []
for m,n in matched:
    if m.distance < 0.1*n.distance: #限定来找相似点 0.08
        accept.append(m)
print len(accept)
```

中 0.1 改为 0.05 后结果如下：



最优的匹配图像中也只找到一处匹配点，可见筛选非常严苛，可以通过调节这个参数来进行控制。

七、实验总结

SIFT 检测是一个非常经典的算法，通过提取图像的特征，判断图像间是否匹配。SIFT 非常实用，平移，旋转，仿射变换等都不会影响其检测。SIFT 算法很巧妙，通过简单的数学知识就能教会计算机如何匹配识别相同的图像。加上上次的 canny，我们现在已经能对图像进行许多实用的处理了，机器学习真的是一个很有趣的领域。

F1403023　5140309534　韩坤言

附源码：

```
# -*- coding: cp936 -*-
import cv2
import numpy as np

target = cv2.imread('target.jpg')
img1 = cv2.imread('1.jpg')
img2 = cv2.imread('2.jpg')
img3 = cv2.imread('3.jpg')
img4 = cv2.imread('4.jpg')
img5 = cv2.imread('5.jpg')
imgs = [img1,img2,img3,img4,img5]

def match(target,imgs):
    sift = cv2.SIFT()
    #detectAndCompute()->
    #keypoints:The input/output vector of keypoints
    #descriptors  -  The output matrix of descriptors
    kp_targ, des_targ = sift.detectAndCompute(target,None)

    # define FLANN parameters
    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks=50)
    flann = cv2.FlannBasedMatcher(index_params,search_params)

    max_num = 0
    for img in imgs:
        kp, des = sift.detectAndCompute(img,None)
        matched = flann.knnMatch(des,des_targ,k=2) #和 target 比较
        #print matched
        accept = []
        for m,n in matched:
            if m.distance < 0.05*n.distance: #限定来找相似点 0.08
                accept.append(m)
        print len(accept)
```

```
            if len(accept)>max_num: #img匹配的数量更多 刷新
                max_num = len(accept)
                best_img = img
                best_kp = kp
                best = accept
        #print best
        return kp_targ, des_targ, best, best_img, best_kp


    def display(target, kp_targ, des_targ, best, best_img, best_kp):
        #将两幅图放在一幅图中
        h1, w1 = target.shape[:2]
        h2, w2 = best_img.shape[:2]
        result = np.zeros((max(h1, h2), w1 + w2, 3), np.uint8)
        result[:h1, :w1] = target[:,:]
        result[:h2, w1:] = best_img[:,:]

        #将匹配到的关键点连起来
        for m in best:
            color = (0,0,255)
            cv2.line(result, \
            (int(best_kp[m.queryIdx].pt[0]+w1),int(best_kp[m.queryIdx].pt[1])) ,
\
            (int(kp_targ[m.trainIdx].pt[0]), int(kp_targ[m.trainIdx].pt[1])),
color)

        cv2.imshow('match_result',result)
        cv2.waitKey()
        cv2.destroyAllWindows()


    kp_targ, des_targ, best, best_img, best_kp = match(target,imgs)
    display(target, kp_targ, des_targ, best, best_img, best_kp)
```