电类工程导论(C 类)实验 12 报告

贾萧松 516030910548

一、实验概述

在 Lab12，我们学习并自己完成了一遍 sift 算法的流程，并用几张图片去测试了 sift 匹配的效果。

二、实验环境

Ubuntu14.04+Pythton2.7+Openv2.4.11+Numpy1.13.3

三、实验内容

1. 关键点定位

由于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，实验中使用另一种较简单的替代方案，使用 opencv 中自带的提取 Harris 角点来作为关键点。
实现如下：

```
# 1 用Harris角点近似关键点
def get_KeyPoints(img):
    tmp = cv2.goodFeaturesToTrack(
        img, maxCorners=1000, qualityLevel=0.01, minDistance=10, blockSize=3, k=0.04)
    KeyPoints = []
    for p in tmp:
        KeyPoints.append([p[0][1], p[0][0]])
    return KeyPoints
```

图中的 cv2.goodFeaturesToTrack，从官方文档了解到：
img 是原图片
maxCorners: 最大数目的角点数
qualityLevel: 该参数指出最低可接受的角点质量，是一个百分数，示例中给出为 0.01。 具体来说，如果最好的角点质量=1500，而 qualityLevel = 0.01，那么角点质量<15 的就都会被拒掉。
minDistance: 角点之间最小的欧拉距离，避免得到相邻特征点。
blockSize: 计算梯度的领域窗口大小
k：Harris 角点检测的阈值比例

2. 舍弃位于边界的点

由于在下面的步骤需要一个半径为 8 的领域窗口，再考虑到需要旋转坐标轴，所以关键点距离边界 x,y 方向均需大于 $8\sqrt{2}\approx 12$

```
# 2 判断该关键点周围是否足够大（不在边界）：
def isValid(height, width, y, x):
    return ((y > 12) and (y < height-12) and (x > 12) and (x < width-12))
```

3. 求关键点主方向

a. 原理：
为了使描述符具有旋转不变性，需要利用图像的局部特征为给每一个关键点分配一个基准方向。使用图像梯度的方法求取局部结构的稳定方向。
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，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)。

b. 代码实现

```
# 3 求关键点主方向
def get_MainDir(img, y, x):
    y = int(y)
    x = int(x)
    dir_hist = [0]*36
    for i in range(-8, 9):
        for j in range(-8, 9):
            dx = img[y+i][x+j+1] - img[y+i][x+j-1]
            dy = img[y+i+1][x+j] - img[y+i-1][x+j]
            # 梯度幅值
            magnitude = sqrt(dx*dx+dy*dy)
            # 梯度方向0~360
            sita = np.arctan2(dy, dx)*180/pi+180
            if(sita >= 360):
                sita -= 360
            dir_hist[int(sita/10)] += magnitude

    max_index = 0
    for i in range(0, 35):
        if(dir_hist[i] > dir_hist[max_index]):
            max_index = i
    return (max_index*10+5)*pi/180
```

在代码中，对每一个特征点（关键点领域中的点 16x16）：求其 x,y 方向的梯度，然后得出梯度幅值和梯度方向，求梯度方向用到了 Numpy 中的 arctan2 其返回范围为(-π， π)，需要换算到 0°~360°方便计算。

最后，统计出来后，找出权重最大的方向区间，取中值返回。

4. 生成描述子

通过以上步骤，对于每一个关键点，拥有了位置和、方向。接下来就是为每个关键点建立一个描述符，用一组向量将这个关键点描述出来，使其不随各种变化而改变，比如光照变化、视角变化等等。这个描述子不但包括关键点，也包含关键点周围对其有贡献的像素点，并且描述符应该有较高的独特性，以便于提高特征点正确匹配的概率。

SIFT 描述子是关键点邻域高斯图像梯度统计结果的一种表示。通过对关键点周围图像区域分块，计算块内梯度直方图，生成具有独特性的向量，这个向量是该

区域图像信息的一种抽象，具有唯一性。

Lowe 建议描述子使用在关键点尺度空间内 4*4 的窗口中计算的 8 个方向的梯度信息，共 4*4*8=128 维向量表征。

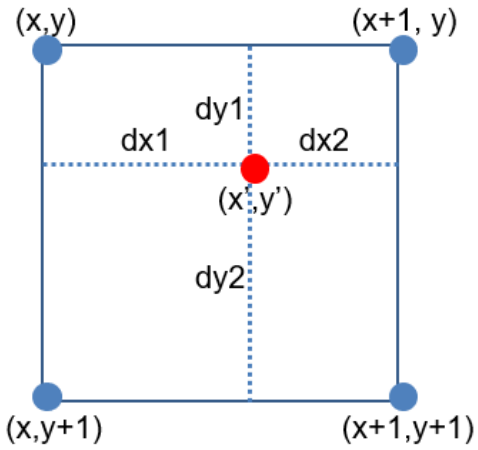
4.1 获得旋转后坐标

对于关键点在物体坐标系中的点，需要转换为图像坐标系中坐标来计算方向
第一步就是计算方向，应用公式：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

4.2 双线性插值

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：



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

实现如下：

```
#得到该点的的灰度梯度方向
def get_dir(img, y, x):
    return np.arctan2(img[y+1][x]-img[y-1][x], img[y][x+1]-img[y][x-1])+pi


# 双线性插值
def insert_value(img, main_dir,y, x):
    x0 = int(x)
    y0 = int(y)
    sita = get_dir(img, y0+1, x0)*(y-y0)*(x0+1-x) + get_dir(img, y0, x0)*(y0+1-y)*(x0+1-x) + \
                get_dir(img, y0, x0+1)*(y0+1-y)*(x-x0) + get_dir(img, y0+1, x0+1)*(y-y0)*(x-x0) - main_dir
    while(sita < 0):
        sita += 2*pi
    return sita
```

就是按照公式实现，算出在图像坐标系的角度后，减去主方向角度即得到在物体坐标系的角度。

4.3 描述子向量归一化

特征向量形成后，为了去除光照变化的影响，需要对它们进行归一化处理，对于图像灰度值整体漂移，图像各点的梯度是邻域像素相减得到，所以也能去除。公式为：

$$l_i = \frac{h_i}{\sqrt{\sum_{j=1}^{128} h_j}}, \quad j = 1,2,3,\ldots$$

4.4 整体实现

把上述步骤合在一起如下：

```
#传入关键点及其主方向，求出128维特征向量
def get_descriptor(img, main_dir, y, x):
    cos_main = cos(main_dir)
    sin_main = sin(main_dir)

    descriptor = []

    for row in range(4):
        for column in range(4):
                subdescriptor = [0]*8
                for i in range(4):
                    for j in range(4):
                        # 计算旋转后采样点坐标
                        xp = cos_main*(column*4-8+j) - sin_main*(row*4-8+i) + x
                        yp = sin_main*(column*4-8+j) + cos_main*(row*4-8+i) + y

                        # 双线性插值
                        sita = insert_value(img, main_dir, yp, xp)
                        sita = sita * 180/pi
                        if(sita>=360):
                            sita -= 360
                        subdescriptor[int(sita/45)] += 1
                descriptor.extend(subdescriptor)
    descriptor = np.array(descriptor, dtype='float32')
    #归一化
    total = np.sum(descriptor**2)**0.5
    if(total == 0):
        return descriptor
    descriptor = descriptor / total
    return descriptor
```

为了统计方便，在遍历的时候我将邻域分为 16 块，这样可以直接统计各块内的 8 个方向上的权重。

5. 图片特征点集合的获得

经过上述步骤，我们可以得到图片的特征点集合：

```
#获得给定图片所有的128维特征向量
def get descriptor set(img):
    # 1 用Harris角点近似关键点
    KeyPoints=get KeyPoints(img)

    height, width=img.shape
    descriptor set=[]
    for KeyPoint in KeyPoints:
        y, x=KeyPoint
        # 2 判断该关键点周围是否足够大（不在边界）：
        if(not isValid(height, width, y, x)):
            continue

        # 3 求关键点主方向
        main dir=get MainDir(img, y, x)

        # 4 生成描述子
        descriptor=get descriptor(img, main dir, y, x)
        descriptor set.append([descriptor, [y,x]])
    return descriptor set
```

为了便于后续使用，集合中的每一个元素都是特征向量+坐标的形式。

6. 特征点匹配

当两幅图像的 SIFT 特征向量生成后，下一步将采用关键点特征向量的欧式距离来作为两幅图像中关键点的相似性判定度量。

然而由于遮挡等原因，匹配可能出现错配的情况，需要采取一些措施降低错配率。为每对匹配特征点的特征向量欧式距离加阈值的方法并不合适，因为特征点可能有较强的独特性，其匹配点对特征向量的欧式距离较大是正常的。

一种比较有效的方法是在匹配时比较与关键点的特征向量最近的和次近的关键点。取图像 1 中的某个关键点，并找出其与图像 2 中欧式距离最近的前两个关键点，在这两个关键点中，如果最近的距离除以次近的距离少于某个比例阈值，则接受这一对匹配点。

实现：

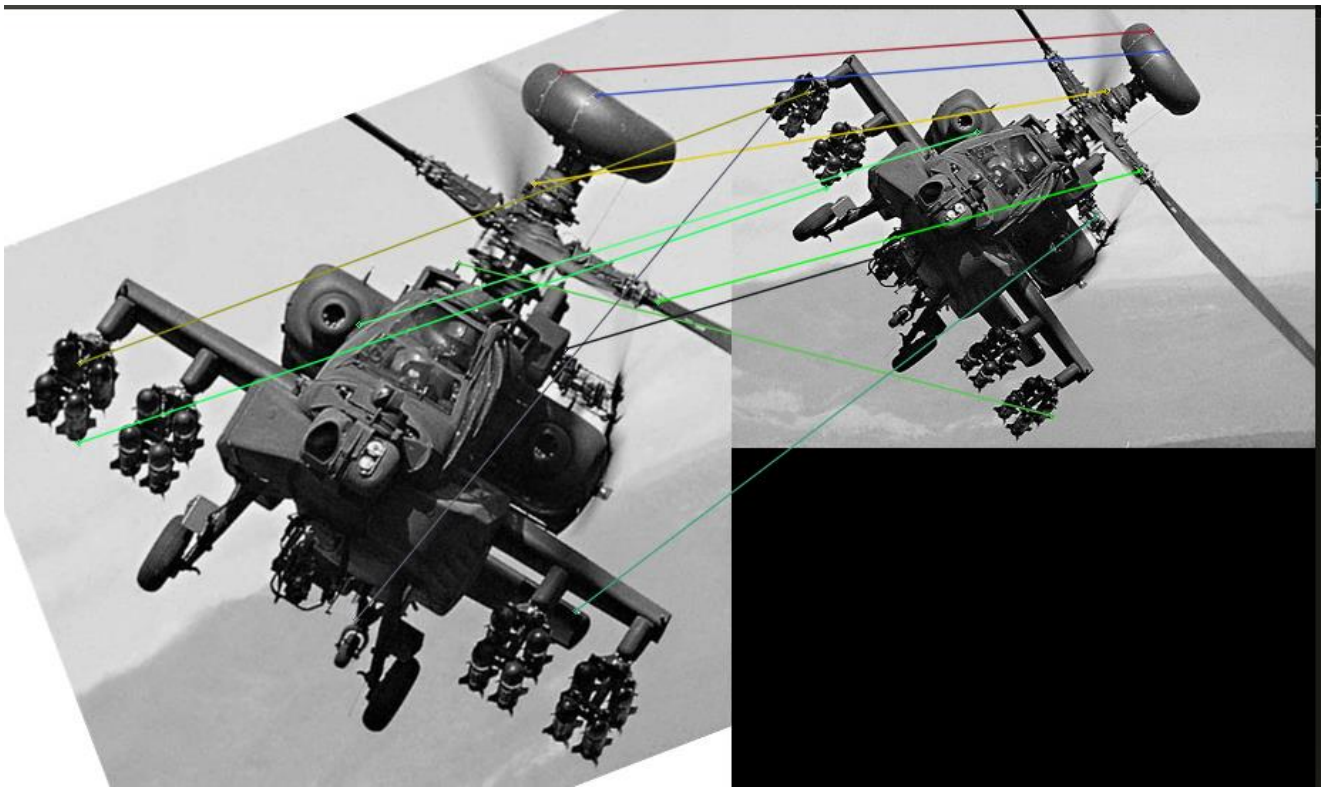
```
#求出每两个特征向量的欧式距离
for i in range(len(des1)):
    minm = 10000
    second_minm = 10000
    minm_index = -1
    #找出与图 1 第 i 个关键点欧氏距离最近和次近的两个图二中关键点
    for j in range(len(des2)):
        distance = np.linalg.norm(des1[i][0] - des2[j][0])
        if(distance < minm):
            second_minm = minm
            minm = distance
            minm_index = j
        if(minm < distance and distance < second_minm):
            second_minm = distance

    #如果最小值和次小值比例小于ratio则认为二者匹配
    if(minm/second_minm < ratio):
```

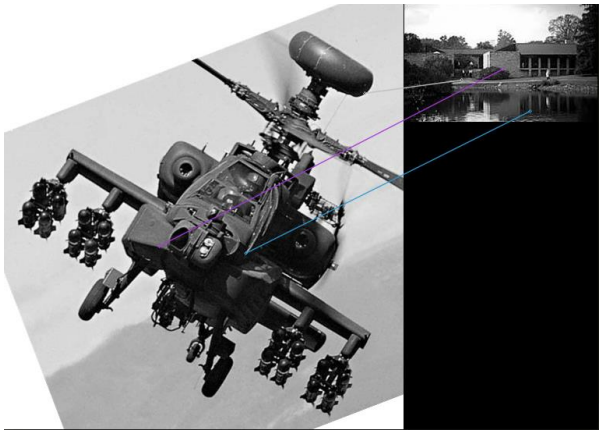
该过程就是找出所有满足条件的点对，接下来我使用 opencv 中的内置绘图将点对连接。

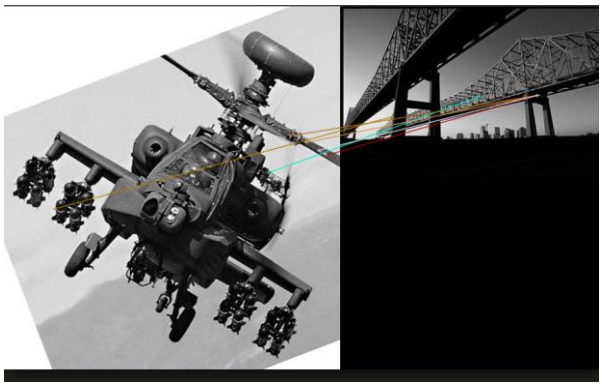
7. 结果

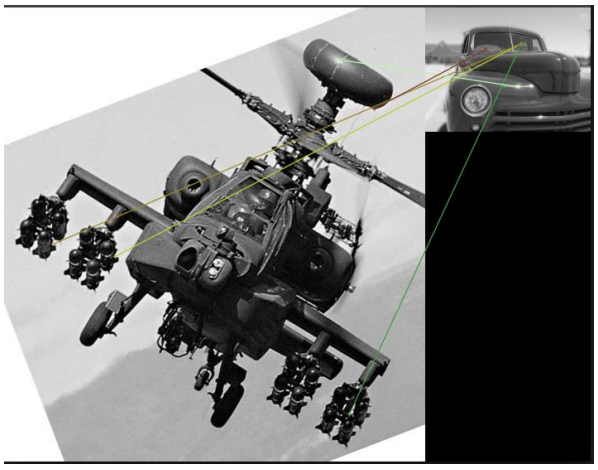
为了便于看清点对，我将线条颜色随机化。

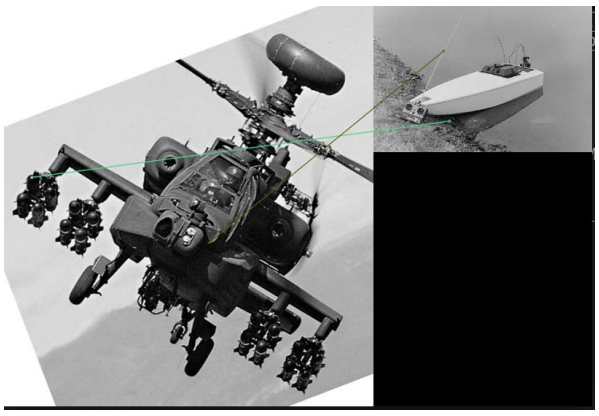


可以看出大部分匹配还是正确的，不过确实存在个别错误匹配。









这里由于我的阈值设置的比较高，在其他图片上还是会有匹配，不过数量相比就比较少了。

四、实验拓展

1. 根据 Lowe 论文的一些优化

在阅读 Lowe 论文之后，我发现其中的很多做法于 PPT 中推荐的不大相同（如只考虑最高峰），于是我尝试在 sift 中使用 Lowe 论文中的内容来优化。

1.1 高斯平滑及描述子向量门限化

Lowe 建议使用高斯函数对采样点梯度大小进行平滑以增强特征点近的邻域点对关键点方向的作用，并减少突变的影响。

```
def get MainDir(img, y, x):
    y = int(y)
    x = int(x)
    dir hist = [0]*36
    for i in range(-8, 9):
        for j in range(-8, 9):
            dx = img[y+i][x+j+1] - img[y+i][x+j-1]
            dy = img[y+i+1][x+j] - img[y+i-1][x+j]
            # 梯度幅值
            magnitude = sqrt(dx*dx+dy*dy)*Gauessian(i, j, 2.6)
            # 梯度方向0~360
            sita = np.arctan2(dy, dx)*180/pi+180
            if(sita >= 360):
                sita -= 360
            dir hist[int(sita/10)] += magnitude
```

上图为计算求取主方向过程中加高斯，其中 sigma = 2.6，计算是根据 Lowe 论文中的关系，计算得 sigma=R/3（R 为采样点区域半径—8）。

非线性光照，相机饱和度变化对造成某些方向的梯度值过大，而对方向的影响微弱。因此设置门限值截断较大的梯度值

```
subdescriptor[int(sita/45)] += min(magnitude*Gauessian(yp-y, xp-x, 2), 0.2)
```

同理，上图为生成描述子过程中加高斯，其中坐标为采样点在图像坐标系中的相对坐标，sigama=2，计算是根据 Lowe 论文中的关系，计算得 sigma=d/2（d 为一个描述子子窗口的大小—4）。同时，进行门限化（Lowe 建议 0.2）。

1.2 直方图平滑

Lowe 论文中提到为了获得更好的稳定性，可以对关键点邻域的梯度大小进行

高斯加权。每相邻三个 bin 采用高斯加权，根据 Lowe 的建议，模板采用[0.25,0.5,0.25]，并连续加权两次。

```
def smooth_hist(hist):
    for i in range(2):
        res = [0]*36
        res[0] = 0.25*hist[35] + 0.5*hist[0] + 0.25*hist[1]
        res[35] = 0.25*hist[34] + 0.5*hist[35] + 0.25*hist[0]
        for j in range(1,35):
            res[j] = 0.25*hist[j-1] + 0.5*hist[j] + 0.25*hist[j+1]
        hist = res
    return hist
```

1.3 获得辅方向

Lowe 在论文中提到为了增强匹配的鲁棒性，只保留峰值大于主方向峰值 80%的方向作为该关键点的辅方向。因此，对于同一梯度值的多个峰值的关键点位置，在相同位置和尺度将会有多个关键点被创建但方向不同，可以明显的提高关键点匹配的稳定性。实际编程实现中，就是把该关键点复制成多份关键点，并将方向值分别赋给这些复制后的关键点。

```
res = [(max index*10+5)*pi/180]
for i in range(0,35):
    if(i == max index or dir hist[i] < dir hist[max index]*0.8):
        continue
    res.append((i*10+5)*pi/180)
return res
```

思路就是在获得主方向后，找统计值大于最高峰 80%的方向，把它们放到一个 list 中返回，之后就相当于有多个位置相同方向不同的关键点处理就可以。

1.4 图像金字塔的建立

Lowe 论文中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。

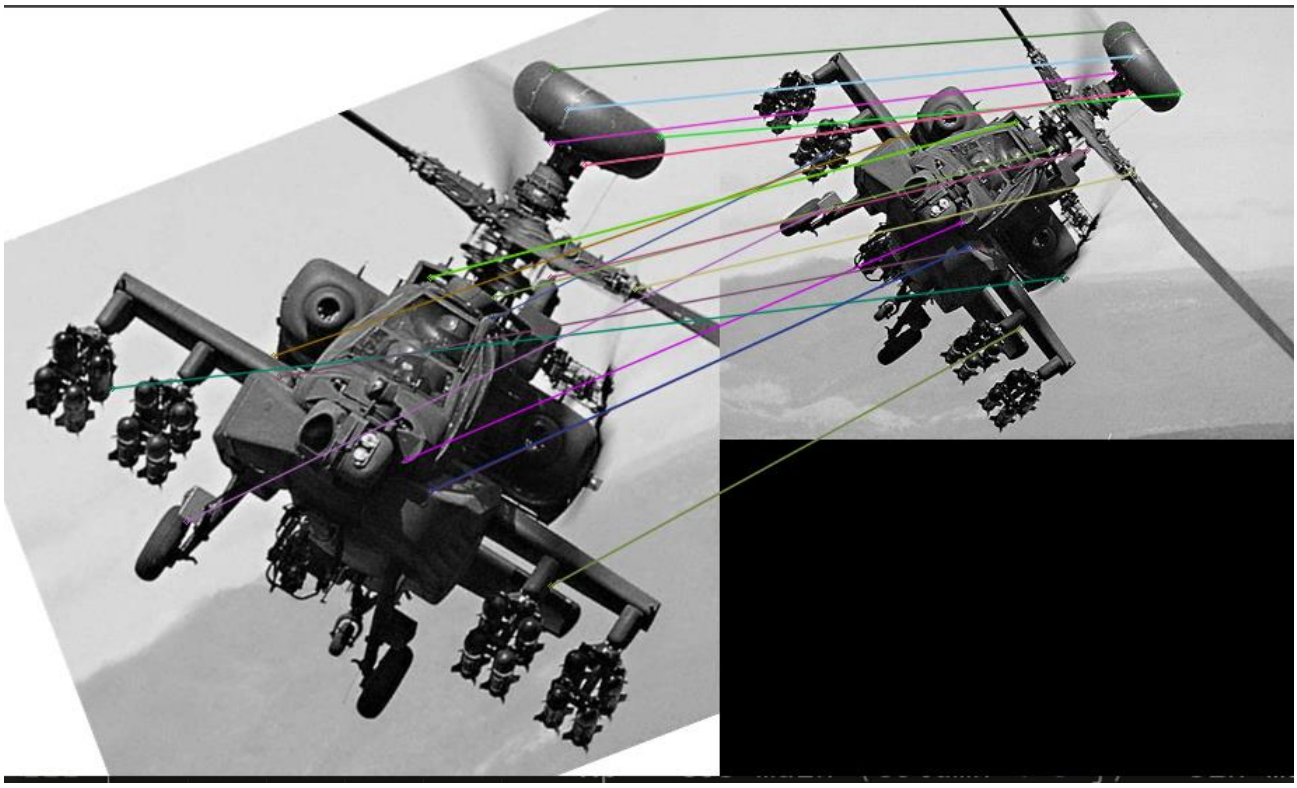
```python
#获得给定图片所有的128维特征向量
def get_descriptor_set(ori_img):
    descriptor_set = []
    scale = 1.0
    height, width = ori_img.shape
    while(height*scale > 100 and width*scale > 100):
        img = cv2.resize(ori_img, (int(scale*width), int(scale*height)), interpolation=cv2.INTER_AREA)
        # 1 用Harris角点近似关键点
        KeyPoints=get_KeyPoints(img)
        for KeyPoint in KeyPoints:
            y, x=KeyPoint
            # 2 判断该关键点周围是否足够大（不在边界）：
            if(not isValid(int(height*scale), int(width*scale), y, x)):
                continue

            # 3 求关键点主方向及辅方向
            main_dir=get_MainDir(img, y, x)

            # 4 生成描述子
            for dire in main_dir:
                descriptor=get_descriptor(img, dire, y, x)
                descriptor_set.append([descriptor, [y/scale,x/scale]])
        scale /= 2.0
    return descriptor_set
```

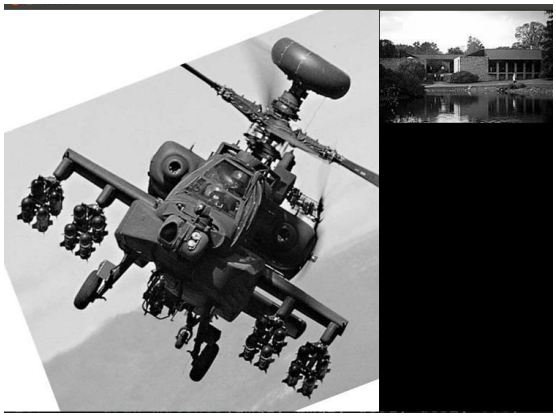
思路就是使用 cv2.resize 函数构建不同大小的图像，在各个图像下都重复上述流程，把各个尺度下的特征向量全部放到一起作为特征向量组。

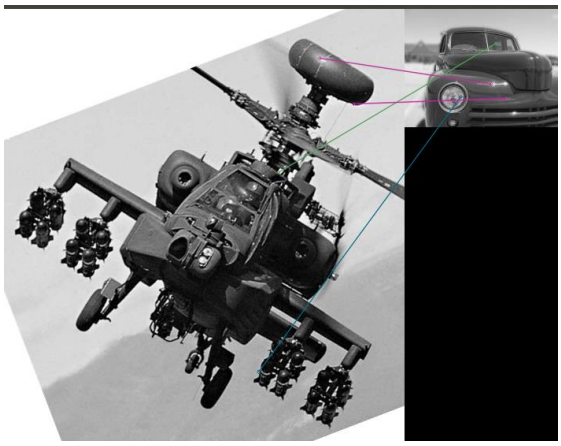
1.5 结果



由图可看出，结果相比较于原本的方法多了不少正确的匹配点，正确率大幅提高(90%)，说明以上的过程还是很有效果的。

下面是其他图片的对比：





发现错误匹配相比原先也好很多，汽车的错点相比原先也好很多

不过，由于考辅方向加上 resize 很多次，相当于多了很多特征向量，计算量也大幅上升，而我在计算方面可能不够优化，加上 Python 本身效率不如 C++，

使得使用这种方法匹配一次大约要 1min，相当于付出了巨大的代价换取了效果的提升。

五、总结

在本实验中，我实现了一遍 sift 算法流程，在这个过程中，我学习到了许多有关解决图片光照，仿射变换和噪音等因素影响的知识和方法。

最后，衷心感谢老师和助教们的精心准备和辛勤付出，你们整理完善的 ppt 和到位的答疑大大提高了我学习 SIFT 的效率，感谢~

贾萧松 516030910548
2017.12.17

六、源代码

下面是我根据 Lowe 论文优化后的源代码：

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-
#实验拓展——Lowe

import sys
reload(sys)
sys.setdefaultencoding('utf-8')

import cv2
import numpy as np
from math import *
from matplotlib import pyplot as plt


# 1 用Harris角点近似关键点
def get_KeyPoints(img):
    tmp = cv2.goodFeaturesToTrack(
        img, maxCorners=1000, qualityLevel=0.01, minDistance=10, blockSize=3, k=0.04)
    KeyPoints = []
    for p in tmp:
        KeyPoints.append([p[0][1], p[0][0]])
    return KeyPoints
```

```
# 2 判断该关键点周围是否足够大（不在边界）：
def isValid(height, width, y, x):
    return ((y > 12) and (y < height-12) and (x > 12) and (x < width-12))


#高斯函数
def Gauessian(y, x, sigma):
    return exp(-(y*y+x*x)/(2*sigma*sigma))

# 3 求关键点主方向
def get MainDir(img, y, x):
    y = int(y)
    x = int(x)
    dir hist = [0]*36
    for i in range(-8, 9):
        for j in range(-8, 9):
            dx = img[y+i][x+j+1] - img[y+i][x+j-1]
            dy = img[y+i+1][x+j] - img[y+i-1][x+j]
            # 梯度幅值
            magnitude = sqrt(dx*dx+dy*dy)*Gauessian(i, j, 2.6)
            # 梯度方向0~360
            sita = np.arctan2(dy, dx)*180/pi+180
            if(sita >= 360):
                sita -= 360
            dir hist[int(sita/10)] += magnitude
```

```
    dir_hist = smooth_hist(dir_hist)
    max_index = 0
    for i in range(0, 35):
        if(dir_hist[i] > dir_hist[max_index]):
            max_index = i

    res = [(max index*10+5)*pi/180]
    for i in range(0,35):
        if(i == max index or dir hist[i] < dir hist[max index]*0.8):
            continue
        res.append((i*10+5)*pi/180)
    return res
```

```
#直方图平滑
def smooth_hist(hist):
    for i in range(2):
        res = [0]*36
        res[0] = 0.25*hist[35] + 0.5*hist[0] + 0.25*hist[1]
        res[35] = 0.25*hist[34] + 0.5*hist[35] + 0.25*hist[0]
        for j in range(1,35):
            res[j] = 0.25*hist[j-1] + 0.5*hist[j] + 0.25*hist[j+1]
        hist = res
    return hist


#得到该点的的灰度梯度方向
def get dir(img, y, x):
    return np.arctan2(img[y+1][x]-img[y-1][x], img[y][x+1]-img[y][x-1])+pi

#得到该点的灰度梯度值
def get mag(img, y, x):
    return sqrt((img[y+1][x]-img[y-1][x])**2+(img[y][x+1]-img[y][x-1])**2)
```

```
# 双线性插值
def insert_value(img, main_dir,y, x):
    x0 = int(x)
    y0 = int(y)
    sita = get_dir(img, y0+1, x0)*(y-y0)*(x0+1-x) + get_dir(img, y0, x0)*(y0+1-y)*(x0+1-x) + \
                get_dir(img, y0, x0+1)*(y0+1-y)*(x-x0) + get_dir(img, y0+1, x0+1)*(y-y0)*(x-x0) - main_dir
    magnitude = get_mag(img, y0+1, x0)*(y-y0)*(x0+1-x) + get_mag(img, y0, x0)*(y0+1-y)*(x0+1-x) + \
                get_mag(img, y0, x0+1)*(y0+1-y)*(x-x0) + get_mag(img, y0+1, x0+1)*(y-y0)*(x-x0)
    while(sita < 0):
        sita += 2*pi
    return sita,magnitude
```

```
def get_descriptor(img, main_dir, y, x):
    cos_main = cos(main_dir)
    sin_main = sin(main_dir)
    descriptor = []
    for row in range(4):
        for column in range(4):
                subdescriptor = [0]*8
                for i in range(4):
                    for j in range(4):
                        # 计算旋转后采样点坐标
                        xp = cos_main*(column*4-8+j) - sin_main*(row*4-8+i) + x
                        yp = sin_main*(column*4-8+j) + cos_main*(row*4-8+i) + y

                        # 双线性插值
                        sita,magnitude = insert_value(img, main_dir, yp, xp)
                        sita = sita * 180/pi
                        if(sita>=360):
                            sita -= 360
                        #加高斯门限值0.2
                        subdescriptor[int(sita/45)] += min(magnitude*Gauessian(yp-y, xp-x, 2), 0.2)
                descriptor.extend(subdescriptor)
    descriptor = np.array(descriptor, dtype='float32')
    #归一化
    total = np.sum(descriptor**2)**0.5
    if(total == 0):
        return descriptor
    descriptor = descriptor / total
    return descriptor
```

```
#获得给定图片所有的128维特征向量
def get_descriptor_set(ori_img):
    descriptor_set = []
    scale = 1.0
    height, width = ori_img.shape
    #获得在高斯金字塔各层下的特征向量
    while(height*scale > 100 and width*scale > 100):
        img = cv2.resize(ori_img, (int(scale*width), int(scale*height)), interpolation=cv2.INTER_AREA)
        # 1 用Harris角点近似关键点
        KeyPoints=get_KeyPoints(img)
        for KeyPoint in KeyPoints:
            y, x=KeyPoint
            # 2 判断该关键点周围是否足够大（不在边界）：
            if(not isValid(int(height*scale), int(width*scale), y, x)):
                continue

            # 3 求关键点主方向及辅方向
            main_dir=get_MainDir(img, y, x)

            # 4 生成描述子
            for dire in main_dir:
                descriptor=get_descriptor(img, dire, y, x)
                descriptor_set.append([descriptor, [y/scale,x/scale]])
        scale /= 2.0
    return descriptor_set
```

```
#得到两组特征向量 进行匹配连线
def draw_matches(img1, img2, des1, des2, ratio):
    rows1 = img1.shape[0]
    cols1 = img1.shape[1]
    rows2 = img2.shape[0]
    cols2 = img2.shape[1]

    res = np.zeros((max([rows1,rows2]),cols1+cols2,3), dtype='uint8')
    res[:rows1,:cols1] = np.dstack([img1, img1, img1])
    res[:rows2,cols1:] = np.dstack([img2, img2, img2])

    #求出每两个特征向量的欧式距离
    for i in range(len(des1)):
        minm = 10000
        second_minm = 10000
        minm_index = -1
        #找出与图 1 第 i 个关键点欧氏距离最近和次近的两个图二中关键点
        for j in range(len(des2)):
            distance = np.linalg.norm(des1[i][0] - des2[j][0])
            if(distance < minm):
                second_minm = minm
                minm = distance
                minm_index = j
            if(minm < distance and distance < second_minm):
                second_minm = distance
```

```
def main():
    img1 = np.float32(cv2.imread('target.jpg', cv2.IMREAD_GRAYSCALE))
    des1 = get_descriptor_set(img1)

    img2 = np.float32(cv2.imread('3.jpg', cv2.IMREAD_GRAYSCALE))
    des2 = get_descriptor_set(img2)

    draw_matches(img1, img2, des1, des2, 0.8)

main()
```