电类工程学导论 C 实验报告 12

518030910406　郑思榕

一、　实验准备

1. 实验环境介绍

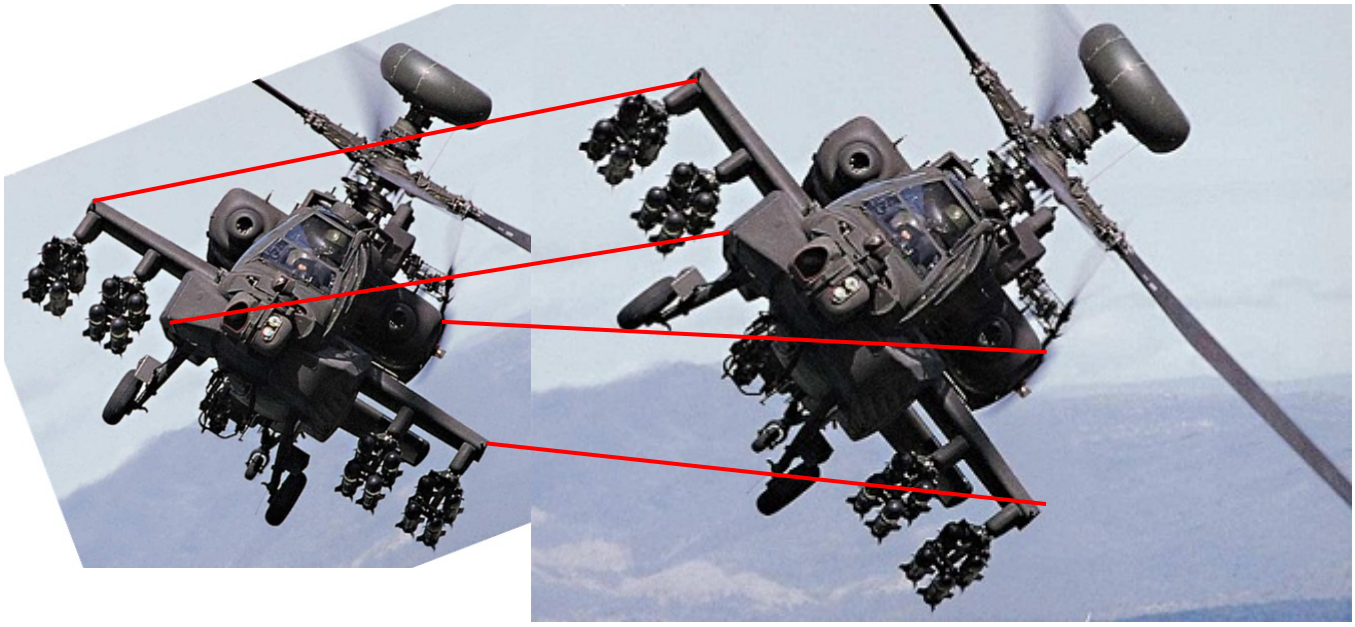
1) 环境：在 windows 系统中进行本次实验

2) 语言：　python 3.7

3) 工具：本实验主要使用了开源计算机视觉库 openCV、处理大型多维矩阵的 python 库 numpy

2. 实验目的

使用自己编写的 SIFT 算法，在 dataset 文件夹中的所有图片中搜索 target.jpg 图片所示物体，并绘制程序认为的好的匹配，如图所示。



3. 实验原理

SIFT 图像特征主要分为三个步骤提取分为：

1) 图像关键点(keypoint)的提取：

使用尺度空间极值的方法或 Harris 角点提取的方法。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，故本实验使用在图像金字塔中提取角点的方法

2) SIFT 描述子的计算：

分为关键点主方向计算和统计 SIFT 描述子两部分。

关键点主方向计算：将某个关键点的四周分成 4*4 个块，每个块中有 16 个像素点。通过统计该关键点周围 4*4*16 个像素点的梯度方向和强度得到梯度直方图，其中梯度方向分为 360 度（注意 atan 函数返回值为 0-180，需要根据 lx ly 的符号换算），平均分成 36 个 bins，每个像素点为其所在的 bin 投票，最终梯度直方图中最大的方向即为该关键点的主方向。

统计 SIFT 描述子：得到关键点主方向后，将该关键点从图像坐标系转换为以主方向为 x 轴的物体坐标系，并将周围 4*4*16 个像素点的梯度转换到物体坐标系，重新得到梯度直方图，因此每个关键点都有一个 16*8=128 维的向量，即为关键点的描述子。

3) 图像特征匹配

将两个图像的所有关键点根据每点的描述子进行配对。本实验由于关键点较少，故基于描述子的欧氏距离通过穷举配对来实现图像特征匹配。

二、 实验过程

1. 图像关键点(keypoint)的提取

由于本实验使用在图像金字塔中提取关键点的方法，故这部分比较简单，主要涉及到两个函数：自定义的改变图像大小的函数 ImgResize 和 cv2

自带的得到图像关键点的函数 goodFeatureToTrack。

goodFeatureToTrack 直接调用即可不再赘述。ImgResize 函数的参数是输入的图片和要放大的倍数 scale，通过改变图片 shape 来改变尺寸。注意使用 round 来进行四舍五入，shape[1]指的是图片行数，shape[0]指的是图片列数。最后使用 cv2.resize 来改变图片，代码如下：

```
5   def ImgResize(pic, scale):  # change the size of the picture
6       rows = int(round(pic.shape[1] * scale, 0))#注意行和列是相反的
7       columns = int(round(pic.shape[0] * scale, 0))
8       return cv2.resize(pic, (rows, columns))
```

2. SIFT 描述子的计算

这部分是本次实验的重点，分为关键点主方向计算和统计 SIFT 描述子两部分。

1) 关键点主方向计算

先定义一个函数 IntensityAndTheta 来得到某像素点的梯度大小和梯度方向，使用 math.hypot 和 math.atan2 函数来计算斜边和角度（以度为单位）。由于对图像矩阵进行梯度计算容易越界，因此顺带定义一个判断是否越界的函数 isValid：

```
10  def IntensityAndTheta(im, x, y):
11      dx = int(im[x + 1, y]) - int(im[x - 1, y])
12      dy = int(im[x, y + 1]) - int(im[x, y - 1])
13      intensity = math.hypot(dx, dy)
14      theta = int(math.atan2(dy, dx) * 180 / math.pi + 180)  # 以度数为单位
15      return intensity, theta
16
17  def isValid(img, x, y):
18      return (x >= 1) and (x <= (img.shape[0]-3)) and (y >= 1) and (y <= (img.shape[1]-3))
```

接下来在 getMainDirection 函数中对关键点周围 16*16 个像素点调用 IntensityAndTheta 函数得到梯度大小和方向。x0,y0 为关键点的坐

标。方向以 10°为划分，共分为 36 个方向，每个点为其所在的梯度方向进行赋分，赋分的值为梯度大小，最终得到梯度直方图 DirectionHist。

```
20  def getMainDirection(img, x0, y0):  # 找到关键点的主方向
21      DirectionHist = [0] * 36
22      radius = 16
23      for i in range(x0 - radius, x0 + radius + 1):
24          for j in range(y0 - radius, y0 + radius + 1):
25              if not isValid(img, i, j):
26                  continue
27              intensity, theta = IntensityAndTheta(img, i, j)
28              theta = theta // 10
29              if (theta == 36): theta = 0
30              DirectionHist[theta] += intensity
31      max = 0
32      mainDirectoin = 0
33      for item in DirectionHist:
34          if (max < item):
35              max = item
36      mainDirectoin = DirectionHist.index(max)*10
37      return mainDirectoin
```

找到 DirectionHist 中权重最大的方向即为关键点主方向，代码如下：

2) 统计 SIFT 描述子

得到关键点主方向后，将该关键点从图像坐标系转换为以主方向为 x 轴的物体坐标系，并将周围 4*4*16 个像素点的梯度转换到物体坐标系，重新得到梯度直方图。由下图的几何关系可知，物体坐标系和图像坐标系存在着 sin(θ)和 cos(θ)的线性关系，因此要得到 4*4=16 个块的 8 维梯度分布向量，可先将坐标转到每个块最左上角的像素点：

```
53  def get128vector(img, cx, cy):  # get 128-dimention vector of each corner point
54      maindirection = getMainDirection(img, cx, cy) * math.pi / 180.0
55      sin = math.sin(maindirection)  # cx,cy:x and y of the corner point
56      cos = math.cos(maindirection)
57
58      # 接下来计算4*4=16个块在物体坐标系下的梯度
59      x0 = cx + 8 * sin - 8 * cos
60      y0 = cy - 8 * cos - 8 * sin
61      # 由几何关系可得上面两行代码，此时坐标转到第一个块的最左上角的像素点
```

然后用 for 循环对每个块中 16 个点计算在物体坐标系下的梯度大

小和方向：

```
    Hist = []
    for i in range(4):  # x1,y1每个块的左上角像素坐标
        for j in range(4):
            x1 = x0 + 4 * i * cos
            y1 = y0 + 4 * j * sin
            # 接下来对每个块里的16个点进行插值(求梯度)
            histtemp = [0] * 8
            for i2 in range(4):
                for j2 in range(4):
                    x2 = x1 + i2*cos
                    y2 = y1 + j2*sin
                    if isValid(img, x2, y2):
                        dir = insertValue(img, x2, y2, maindirection)
                        histtemp[int(dir / 45)] += 1
            Hist += histtemp  # 将8维梯度方向直方图加入到描述子中，由此重复16次，每个角点生成128维描述子
```

我们暂停 get128Vector 函数的介绍，来看插值函数的定义。本实验自定义了插值函数 insertValue 将梯度方向从图像坐标系转到物体坐标系，带入插值的公式容易地就能写出代码。注意梯度方向 360°要转化成 0°，防止绘制梯度直方图时越界。insertValue 函数不再赘述，直接贴出代码如下：

```
def insertValue(img, x, y, maindirection):  # x,y:float
    x0 = int(x)
    y0 = int(y)
    theta = (
                IntensityAndTheta(img, x0, y0)[1] * (x0 + 1 - x) * (y0 + 1 - y) +
                IntensityAndTheta(img, x0 + 1, y0)[1] * (x - x0) * (y0 + 1 - y) +
                IntensityAndTheta(img, x0, y0 + 1)[1] * (x0 + 1 - x) * (y - y0) +
                IntensityAndTheta(img, x0 + 1, y0 + 1)[1] * (x - x0) * (y - y0)
    )
    degree = theta - maindirection
    if degree < 0: degree += 360
    if degree==360.0:degree =0
    return degree
```

我们接着介绍尚未讲完的计算关键点描述子的 get128Vector 函数。在统计完每个块在物体坐标系下的梯度方向和大小后得到每个块的梯度直方图，即 8 维向量 histtemp。将 16 个块的 histtemp 向量加入到属于关键点的 Hist 向量中，即得到关键点的 128 维描述子 Hist。为便

于下面进行描述子匹配，我们将描述子进行归一化，将 Hist 向量除以向量大小：

```
77
78      # 接下来对128维SIFT描述子归一化
79      sum = 0
80      for i in Hist:
81          sum += i
82      sum = math.sqrt(sum)
83      for item in Hist:
84          if sum!=0:
85              item /= sum
86
87      return Hist
```

最后定义函数 getSift 通过 url 读取图片，使用 goodFeatureToTrack 函数得到图片的所有关键点（角点），对每个角点调用 get128Vector 得到角点的描述子。最终函数输出为存放所有角点的列表 corners 和存放所有角点描述子的列表 cornerVector：

```
89  def getSift(url, scale=1):
90      img = cv2.imread(url, cv2.IMREAD_GRAYSCALE)
91      #if scale!=1:
92      #    img = ImgResize(img, scale)
93      #else:
94      #    img = ImgResize(img, scale)
95      img = ImgResize(img, scale)
96
97      cornersTemp = cv2.goodFeaturesToTrack(img,20,0.01,10)
98      corners = []
99      cornerVector = []
100     for corner in cornersTemp:  # [[[ 44. 234.]],[[1. 2.]]] ---> [(44,234),(1,2)]
101         corners.append((int(corner[0][0]),int(corner[0][1])))
102         cornerVector.append(get128vector(img, int(corner[0][0]), int(corner[0][1])))
103
104     return corners,cornerVector
```

3. 图像特征匹配

在主函数中进行角点的匹配。对 target image 的每个角点搜索 input image 中和它最接近的角点，判断依据是角点描述子的欧式距离。如果

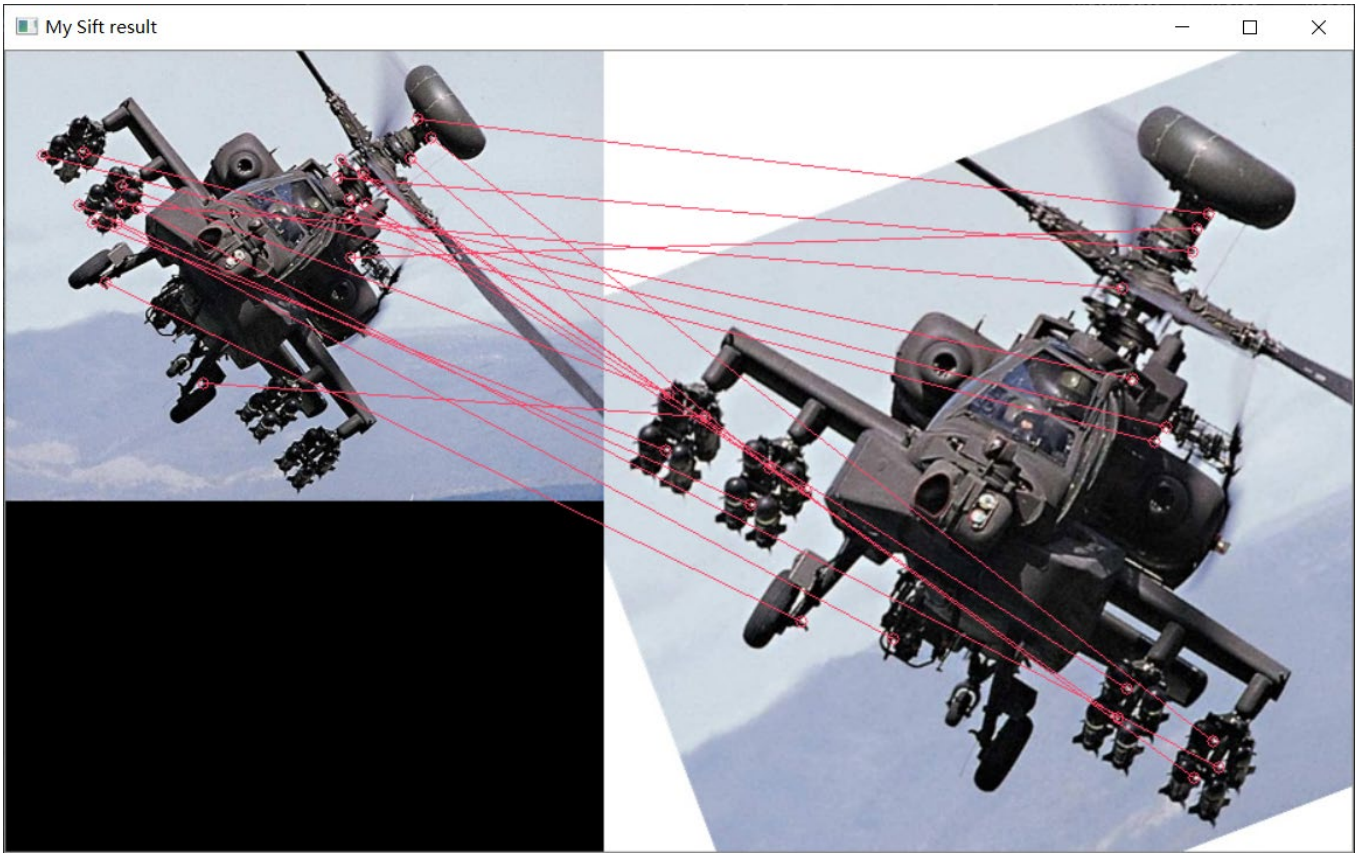
两个描述子的欧氏距离越小，其对应的两个角点就越接近。代码如下：

```
#下面进行角点的匹配,采用欧氏距离遍历
threshold = 1
res_t = []#存放匹配的角点，两个list一一对应
res_i = []

currentIndex_t = 0
for vt in vector_t:
    currentIndex_i = 0
    bestIndex = 0 #最佳匹配角点的下标
    min = 100000 #初始值，用来记录最小欧氏距离，即最佳匹配点的距离
    secondmin = 100000 #用来记录次小欧氏距离
    for vi in vector_i:#遍历一遍input image的所有角点，找到最佳匹配点
        sum = 0
        for i in range(128):
            sum += (vt[i]- vi[i])**2
        sum = math.sqrt(sum)
        if (chosen[currentIndex_i]==0 and sum<min):
            secondmin = min
            min = sum
            bestIndex = currentIndex_i
        currentIndex_i += 1
    if ((min/secondmin)<threshold):
        res_t.append(corner_t[currentIndex_t])
        res_i.append(corner_i[bestIndex])
        chosen[bestIndex] = 1
    currentIndex_t += 1

img_i = cv2.imread(urli)
img_t = cv2.imread(urlt)
img_i = ImgResize(img_i,scale)
draw(img_t,corner_t,img_i,corner_i)
```

然后调用自定义的 draw 函数进行角点和匹配线的绘制。绘制角点主要用到画圈函数 cv2.circle，绘制匹配线主要用到 cv2 中画直线的函数 cv2.line。另外为了将两个图拼在一起，创建了图形矩阵 output，行数为 input image 和 target image 的最大值，列数为两个图像的列数之和，代码如下：

```python
def draw(img_t,corner_t,img_i,corner_i):

    rows_t,columns_t = img_t.shape[0],img_t.shape[1]
    rows_i,columns_i = img_i.shape[0],img_i.shape[1]

    output = np.zeros((max([rows_t,rows_i]),columns_i+columns_t,3),dtype='uint8')
    output[:rows_i,:columns_i,:] = np.dstack([img_i])
    output[:rows_t,columns_i:columns_i+columns_t,:] = np.dstack([img_t])

    for i in range(len(corner_t)):

        xt,yt = corner_t[i]
        xi,yi = corner_i[i]

        color = (101,67,257)
        cv2.circle(output, (int(xi),int(yi)), 4, color, 1)
        cv2.circle(output, (int(xt)+columns_i,int(yt)), 4, color, 1)
        cv2.line(output, (int(xi),int(yi)), (int(xt)+columns_i,int(yt)), color, 1)
    cv2.imshow("My Sift result", output)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

结果展示:



三、 实验总结

1. 实验概述

本实验根据 SIFT 图像特征提取算法的定义，使用 opencv 和 numpy 库函数实现图像关键点提取、SIFT 描述子的计算和图像特征匹配等步骤，绘制了经旋转前后两幅图的匹配图像。

2. 实验心得

本实验我收获了良多，主要有以下几点：

1) 学会了 SIFT 算法的定义，学会了如何自己实现 SIFT 算法。

2) 学会了如何用 cv2 库函数绘制圆圈和直线。

3) 学会了如何在图像坐标系和物体坐标系之间转换

3. 实验创新点

1) 创建图像矩阵 output 来将 target image 和 input image 拼起来显示。output 的行数为 input image 和 target image 的较大值，列数为两个图像的列数之和

```
111     output = np.zeros((max([rows_t,rows_i]),columns_i+columns_t,3),dtype='uint8')
112     output[:rows_i,:columns_i,:] = np.dstack([img_i])
113     output[:rows_t,columns_i:columns_i+columns_t,:] = np.dstack([img_t])
```

2) 使用（最小欧氏距离/次小欧氏距离<threshold）的方式过滤匹配不好的点，提高匹配准确率：

```
160             if ((min/secondmin)<threshold):
161                 res_t.append(corner_t[currentIndex_t])
162                 res_i.append(corner_i[bestIndex])
163                 chosen[bestIndex] = 1
164             currentIndex_t += 1
```

最后，衷心感谢实验中老师和各位助教的帮助!

源代码:

```python
import cv2
import math
import numpy as np

def ImgResize(pic, scale):  # change the size of the picture
    rows = int(round(pic.shape[1] * scale, 0))#注意行和列是相反的
    columns = int(round(pic.shape[0] * scale, 0))
    return cv2.resize(pic, (rows, columns))

def IntensityAndTheta(im, x, y):
    dx = int(im[x + 1, y]) - int(im[x - 1, y])
    dy = int(im[x, y + 1]) - int(im[x, y - 1])
    intensity = math.hypot(dx, dy)
    theta = int(math.atan2(dy, dx) * 180 / math.pi + 180)  # 以度数为单位
    return intensity, theta

def isValid(img, x, y):
    return (x >= 1) and (x <= (img.shape[0]-3)) and (y >= 1) and (y <= (img.shape[1]-3))

def getMainDirection(img, x0, y0):  # 找到关键点的主方向
    DirectionHist = [0] * 36
    radius = 16
    for i in range(x0 - radius, x0 + radius + 1):
        for j in range(y0 - radius, y0 + radius + 1):
            if not isValid(img, i, j):
                continue
            intensity, theta = IntensityAndTheta(img, i, j)
            theta = theta // 10
            if (theta == 36): theta = 0
            DirectionHist[theta] += intensity
    max = 0
    mainDirectoin = 0
    for item in DirectionHist:
        if (max < item):
            max = item
```

```
    mainDirectoin = DirectionHist.index(max)*10
    return mainDirectoin

def insertValue(img, x, y, maindirection):  # x,y:float
    x0 = int(x)
    y0 = int(y)
    theta = (
            IntensityAndTheta(img, x0, y0)[1] * (x0 + 1 - x) * (y0 + 1 - y) +
            IntensityAndTheta(img, x0 + 1, y0)[1] * (x - x0) * (y0 + 1 - y) +
            IntensityAndTheta(img, x0, y0 + 1)[1] * (x0 + 1 - x) * (y - y0) +
            IntensityAndTheta(img, x0 + 1, y0 + 1)[1] * (x - x0) * (y - y0)
    )
    intensity = (
            IntensityAndTheta(img, x0, y0)[0] * (x0 + 1 - x) * (y0 + 1 - y) +
            IntensityAndTheta(img, x0 + 1, y0)[0] * (x - x0) * (y0 + 1 - y) +
            IntensityAndTheta(img, x0, y0 + 1)[0] * (x0 + 1 - x) * (y - y0) +
            IntensityAndTheta(img, x0 + 1, y0 + 1)[0] * (x - x0) * (y - y0)
    )
    degree = theta - maindirection
    if degree < 0: degree += 360
    if degree==360.0:degree =0
    return degree,intensity

def get128vector(img, cx, cy):  # get 128-dimention vector of each corner point
    maindirection = getMainDirection(img, cx, cy) * math.pi / 180.0
    sin = math.sin(maindirection)  # cx,cy:x and y of the corner point
    cos = math.cos(maindirection)

    # 接下来计算4*4=16个块在物体坐标系下的梯度
    x0 = cx + 8 * sin - 8 * cos
    y0 = cy - 8 * cos - 8 * sin
    # 由几何关系可得上面两行代码，此时坐标转到第一个块的最左上角的像素点
    Hist = []
    for i in range(4):  # x1,y1每个块的左上角像素坐标
        for j in range(4):
```

```
70                for j in range(4):
71                    x1 = x0 + 4 * i * cos
72                    y1 = y0 + 4 * j * sin
73                    # 接下来对每个块里的16个点进行插值(求梯度)
74                    histtemp = [0] * 8
75                    for i2 in range(4):
76                        for j2 in range(4):
77                            x2 = x1 + i2*cos
78                            y2 = y1 + j2*sin
79                            if isValid(img, x2, y2):
80                                dir ,intensity = insertValue(img, x2, y2, maindirection)
81                                histtemp[int(dir / 45)] += intensity
82                    Hist += histtemp  # 将8维梯度方向直方图加入到描述子中，由此重复16次，每个角点生成128维描述子
83
84            # 接下来对128维SIFT描述子归一化
85            sum = 0
86            for i in Hist:
87                sum += i
88            sum = math.sqrt(sum)
89            for item in Hist:
90                if sum!=0:
91                    item /= sum
92
93            return Hist
94
95        def getSift(url, scale=1):
96            img = cv2.imread(url, cv2.IMREAD_GRAYSCALE)
97            #if scale!=1:
98            #    img = ImgResize(img, scale)
99            #else:
100           #    img = ImgResize(img, scale)
101           img = ImgResize(img, scale)
102
103           cornersTemp = cv2.goodFeaturesToTrack(img,40,0.01,10)
104           corners = []
```

```
105         cornerVector = []
106         for corner in cornersTemp:  # [[[ 44. 234.]],[[1. 2.]]] ---> [(44,234),(1,2)]
107             corners.append((int(corner[0][0]),int(corner[0][1])))
108             cornerVector.append(get128vector(img, int(corner[0][0]), int(corner[0][1])))
109 
110         return corners,cornerVector
111 
112     def draw(img_t,corner_t,img_i,corner_i):
113 
114         rows_t,columns_t = img_t.shape[0],img_t.shape[1]
115         rows_i,columns_i = img_i.shape[0],img_i.shape[1]
116 
117         output = np.zeros((max([rows_t,rows_i]),columns_i+columns_t,3),dtype='uint8')
118         output[:rows_i,:columns_i,:] = np.dstack([img_i])
119         output[:rows_t,columns_i:columns_i+columns_t,:] = np.dstack([img_t])
120 
121         for i in range(len(corner_t)):
122 
123             xt,yt = corner_t[i]
124             xi,yi = corner_i[i]
125 
126             color = (101,67,257)
127             cv2.circle(output, (int(xi),int(yi)), 4, color, 1)
128             cv2.circle(output, (int(xt)+columns_i,int(yt)), 4, color, 1)
129             cv2.line(output, (int(xi),int(yi)), (int(xt)+columns_i,int(yt)), color, 1)
130         cv2.imshow("My Sift result", output)
131         cv2.waitKey(0)
132         cv2.destroyAllWindows()
133 
134     if __name__=='__main__':
135 
136         urlt = "target.jpg"#target image
137         urli = "dataset/3.jpg"#another input image
138         scale = 1 #输入图片(不是target图片)的放大倍数
139 
```

```
corner_i,vector_i = getSift(urli,scale)

chosen = [0]*len(corner_i)#记录角点是否已被记录

#下面进行角点的匹配,采用欧氏距离遍历
threshold = 1
res_t = []#存放匹配的角点，两个list一一对应
res_i = []

currentIndex_t = 0
for vt in vector_t:
    currentIndex_i = 0
    bestIndex = 0 #最佳匹配角点的下标
    min = 100000 #初始值，用来记录最小欧氏距离，即最佳匹配点的距离
    secondmin = 100000 #用来记录次小欧氏距离
    for vi in vector_i:#遍历一遍input image的所有角点，找到最佳匹配点
        sum = 0
        for i in range(128):
            sum += (vt[i]- vi[i])**2
        sum = math.sqrt(sum)
        if (chosen[currentIndex_i]==0 and sum<min):
            secondmin = min
            min = sum
            bestIndex = currentIndex_i
        currentIndex_i += 1
    if ((min/secondmin)<threshold):
        res_t.append(corner_t[currentIndex_t])
        res_i.append(corner_i[bestIndex])
        chosen[bestIndex] = 1
    currentIndex_t += 1

img_i = cv2.imread(urli)
img_t = cv2.imread(urlt)
img_i = ImgResize(img_i,scale)
draw(img_t,corner_t,img_i,corner_i)
```