实验1 查找特征点

1、查找特征点

1.1、读入图像

我们采用jupyter notebook作为开发环境。创建一个空白的jupyter文件，引入相应的库。这个项目中会需要使用opencv，numpy和matplotlib库，代码如下

import cv2

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

!pip install --upgrade opencv-contrib-python

其中需要通过%matplotlib inline保证在jupyter文档中的正常显示图片。更新opencv-contrib-python包。通过opencv读取要查找的书本图片和场景图片。

#读取一幅特征

img1 = cv2.imread('book3.jpg', 0)

#读取一幅图片

img2 = cv2.imread('book\_in\_scene.jpg', 0)

1.2、查找特征点

我们在整理使用SIFT特征提取器，首先进行初始化。

#初始化SIFT特征提取器

sift = cv2.SIFT\_create()

然后分别对两个图像提取特征，并返回关键点和特征描述。

# 查找和绘制关键点

kp1, des1 = sift.detectAndCompute(img1, None)

kp2, des2 = sift.detectAndCompute(img2, None)

查看关键点的个数

print(len(kp1))

print(len(kp2))

22418

7614

可以看到每张图片都有相当数量的关键点

1.3、绘制特征点

将特征点绘制如下。

#绘制关键点

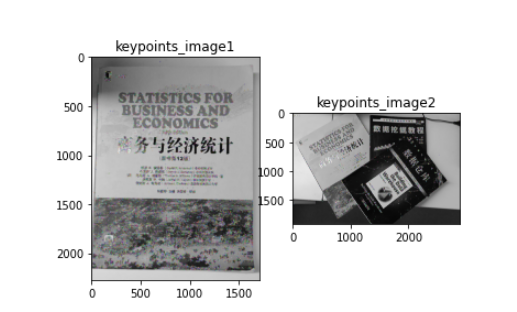
kp\_img1 = cv2.drawKeypoints(img1, kp1, img1)

kp\_img2 = cv2.drawKeypoints(img2, kp2, img2)

plt.subplot(121),plt.imshow(kp\_img1),plt.title("keypoints\_image1")

plt.subplot(122),plt.imshow(kp\_img2),plt.title("keypoints\_image2")

可以看到如下结果：



实验2 完成匹配和校验

1、完成匹配和校验

1.1、完成BF特征匹配

对两幅图完成Brute-Force匹配，首先创建匹配器：

#创建一个BFMatcher匹配器

bf = cv2.BFMatcher(cv2.NORM\_L2, crossCheck=True)

因为我们使用的是SIFT特征提取器，采用NORM\_L2距离测量方式，并使用较为严格的crossCheck匹配条件。

对两幅图像的关键点描述进行匹配

#进行匹配

matches = bf.match(des1, des2)

查看匹配的个数

print(len(matches))

1.2、查找有效的匹配

匹配的距离越短，说明匹配效果越好，首先我们对匹配进行排序，根据距离进行排序。

# 对匹配进行排序

matches = sorted(matches, key=lambda x: x.distance)

查找最大的匹配距离

maxdist = matches[-1].distance

print(maxdist)

365.1109313964844

根据一定的阈值（这个根据实际情况调整），选择最优的匹配。并列入数组。

goodMatches = []

for m in matches:

if m.distance < 0.3 \* maxdist:

goodMatches.append(m)

else:

break

print(len(goodMatches))

403

表示我们找到了最优的403个匹配

1.3、根据匹配完成转换

根据查找到的优质匹配的数量，我们判断目标图像中包含对应的书本。然后我们根据匹配点之间坐标，完成原来的书本和目标中的书本图像的变换。

首先根据优质匹配的数量（这个参数也是自行调节）,判断是否找到对应匹配书本。并获取匹配点在原图和目标图像中的坐标点

MIN\_MATCH\_COUNT = 50

if len(goodMatches) > MIN\_MATCH\_COUNT:

print(len(goodMatches))

#找到本地化对象

obj\_pts = np.float32([kp1[m.queryIdx].pt for m in goodMatches]).reshape(-1,1,2)

scene\_pts = np.float32([kp2[m.trainIdx].pt for m in goodMatches]).reshape(-1,1,2)

这里获取匹配在原图中的坐标点列表，和在目标图中坐标点列表。

计算原图和目标图的变换矩阵M，并获取匹配的mask。

#findHomography函数是计算变换矩阵

M, mask = cv2.findHomography(obj\_pts, scene\_pts, cv2.RANSAC)

matchesMask = mask.ravel().tolist()

获取书本图像的四个顶点坐标。

#获取book的图像尺寸

h,w = img1.shape

#obg\_corners是图像book的四个顶点

obj\_corners = np.float32([[0,0],[w,0],[w,h],[0,h]]).reshape(-1,1,2)

 计算对应的图像的四个顶点坐标位置，并通过多边形绘制出来。

#计算变换后的四个顶点坐标位置

scene\_corners = cv2.perspectiveTransform(obj\_corners, M)

#根据四个顶点坐标位置在img2图像中画出变换后的边框

box\_in\_scene = cv2.polylines(img2, [np.int32(scene\_corners)], True, (0,0,255), 3, cv2.LINE\_AA)

最后将匹配曲线和对应的图像一起绘制展现出来。

# 最后绘制内点(如果成功找到对象)或匹配关键点(如果失败)

draw\_params = dict(matchColor=(0, 255, 0),

singlePointColor=None,

matchesMask=matchesMask,

flags=2)

#绘制匹配

result = cv2.drawMatches(img1, kp1, box\_in\_scene, kp2, goodMatches, None, \*\*draw\_params)

plt.figure(figsize=(10,10))

plt.imshow(result)

获得结果如图：



我们可以自行拍摄图片，并尝试识别效果。

1.4、尝试其它书本图片验证

如果我们用其他书本的图片，查看是否没有找到匹配图像。

这里把完整的代码显示如下：

import cv2

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

plt.figure(figsize=(20,20))

#读取一幅特征

img1 = cv2.imread('book.jpg', 0)

#读取一幅图片

img2 = cv2.imread('book\_in\_scene.jpg', 0)

#初始化SIFT特征提取器

sift = cv2.xfeatures2d.SIFT\_create()

# 查找和绘制关键点

kp1, des1 = sift.detectAndCompute(img1, None)

kp2, des2 = sift.detectAndCompute(img2, None)

#创建一个BFMatcher匹配器

bf = cv2.BFMatcher(cv2.NORM\_L2, crossCheck=True)

#进行匹配

matches = bf.match(des1, des2)

# 对匹配进行排序

matches = sorted(matches, key=lambda x: x.distance)

#发现匹配

maxdist = matches[-1].distance

goodMatches = []

for m in matches:

if m.distance < 0.3 \* maxdist:

goodMatches.append(m)

else:

break

print(len(goodMatches))

MIN\_MATCH\_COUNT = 50

if len(goodMatches) > MIN\_MATCH\_COUNT:

print(len(goodMatches))

#找到本地化对象

obj\_pts = np.float32([kp1[m.queryIdx].pt for m in goodMatches]).reshape(-1,1,2)

scene\_pts = np.float32([kp2[m.trainIdx].pt for m in goodMatches]).reshape(-1,1,2)

#findHomography函数是计算变换矩阵

M, mask = cv2.findHomography(obj\_pts, scene\_pts, cv2.RANSAC)

matchesMask = mask.ravel().tolist()

#获取book的图像尺寸

h,w = img1.shape

#obg\_corners是图像box的四个顶点

obj\_corners = np.float32([[0,0],[w,0],[w,h],[0,h]]).reshape(-1,1,2)

#print("obj\_corners:{}".format(obj\_corners))

#计算变换后的四个顶点坐标位置

scene\_corners = cv2.perspectiveTransform(obj\_corners, M)

#print("scene\_corners:{}".format(scene\_corners))

#根据四个顶点坐标位置在img2图像中画出变换后的边框

box\_in\_scene = cv2.polylines(img2, [np.int32(scene\_corners)], True, (0,0,255), 3, cv2.LINE\_AA)

# 最后绘制内点(如果成功找到对象)或匹配关键点(如果失败)

draw\_params = dict(matchColor=(0, 255, 0),

singlePointColor=None,

matchesMask=matchesMask,

flags=2)

#绘制匹配

result = cv2.drawMatches(img1, kp1, box\_in\_scene, kp2, goodMatches, None, \*\*draw\_params)

else:

print("Not enough matches are found", (len(goodMatches), MIN\_MATCH\_COUNT))

matchesMask = None

# 最后绘制内点(如果成功找到对象)或匹配关键点(如果失败)

draw\_params = dict(matchColor=(0, 255, 0),

singlePointColor=None,

matchesMask=matchesMask,

flags=2)

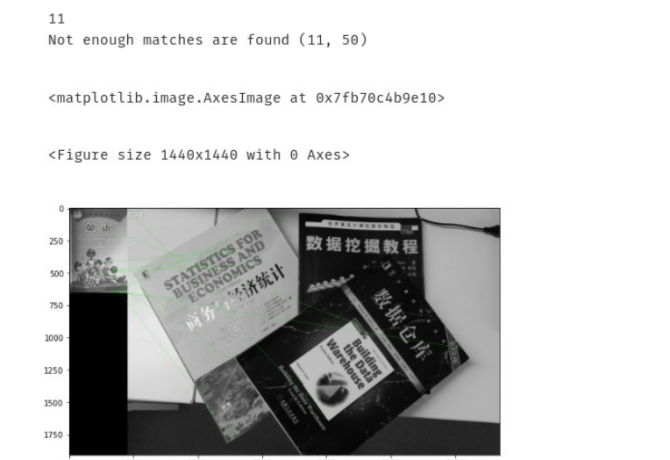
#绘制匹配

result = cv2.drawMatches(img1, kp1, img2, kp2, goodMatches, None, \*\*draw\_params)

plt.figure(figsize=(10,10))

plt.imshow(result)

最后获取的效果如下：



说明没有获取到足够的优质匹配，因此认为没有识别到书本。