

第 14 课 图像处理——特征匹配

1.暴力匹配

特征描述符是在关键点计算后，用一组向量将这个关键点描述出来，这个描述不但包括关键点，也包括关键点周围对其有贡献的像素点。用来作为目标匹配的依据，也可使关键点具有更多的不变特性，如光照变化、3D 视点变化等。

将一组特征点中的每一个特征点描述符与另一组的最接近的特征点描述符匹配，然后将所得到的距离进行排序，最后选择距离最短的特征，作为两者的匹配点。

接下来使用暴力匹配，匹配两幅图片的特征。



1.1 实验步骤

注意：

1) 需要先将目录“第 4 章 OpenCV 计算机视觉学习->图像处理进阶篇->第 14 课 图像处理——特征匹配->例程源码”下的例程“bf_demo.py”和示例图片“test.jpg”，“test1.jpg”复制到共享文件夹。

2) 共享文件夹的配置方法可查看目录“第 2 章 Linux 系统简介及使用入门->Linux 基础课程->第 3 课 Linux 系统安装及换源方法”下的文档。

3) 输入指令时需严格区分大小写，且可以使用“Tab”键补齐关键字。

1) 打开虚拟机，启动系统。点击系统任务栏的图标，并点击图标，或使用快捷键“Ctrl+Alt+T”，打开命令行终端。

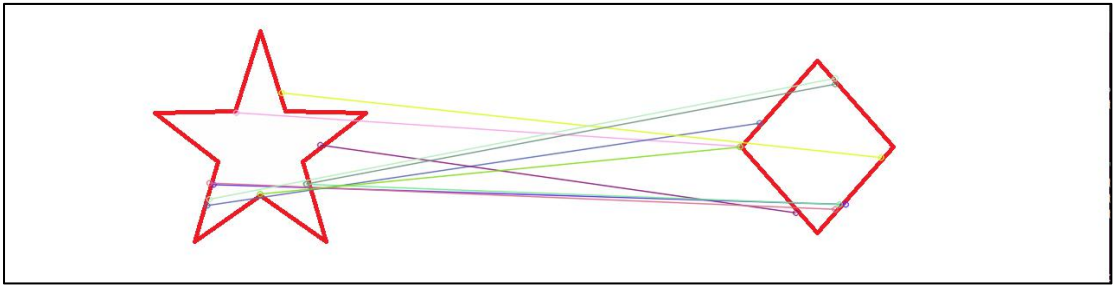
2) 输入指令“cd /mnt/hgfs/share/”，并按下回车，进入共享文件夹。

```
hitwonder@ubuntu:~$ cd /mnt/hgfs/Share/
```

3) 输入指令“python3 bf_demo.py”，并按下回车，运行例程。

```
ubuntu@ubuntu-virtual-machine:/mnt/hgfs/share$ python3 bf_demo.py
```

1.2 实现效果



取出原图像的一部分后，再拿来和原图像匹配。

1.3 代码分析

可在目录“第4章 OpenCV 计算机视觉学习->图像处理进阶篇->第14课 图像处理——特征匹配->例程源码”下查看例程“bf_demo.py”。

```
import cv2

img1 = cv2.imread('test.jpg')
img2 = cv2.imread('test1.jpg')

# 初始化ORB特征点检测器
orb = cv2.ORB_create()

# 检测特征点与描述符
kp1, des1 = orb.detectAndCompute(img1, None)
kp2, des2 = orb.detectAndCompute(img2, None)

# 创建蛮力 (BF) 匹配器
bf = cv2.BFMatcher_create(cv2.NORM_HAMMING, crossCheck=True)

# 匹配描述符
matches = bf.match(des1, des2)

# 画出10个匹配的描述符
img3 = cv2.drawMatches(img1, kp1, img2, kp2, matches[:10], None, flags=2)

cv2.imshow("show", img3)
cv2.waitKey()
cv2.destroyAllWindows()
```

导入模块：导入 cv2 模块。

读取图片：imread 函数读取图片，参数为图片名称。

初始化检测器：使用 ORB_create 构造函数初始化。

```
orb = cv2.ORB_create()
```

检测特征点和描述符：使用函数 detectAndCompute，具体格式和参数如下：

detectAndCompute(src, mask)

- 1) 第一个参数“**src**”，是要阈值处理的图像。
- 2) 第二个参数“**mask**”，是掩模图像，即物体为黑，其余为白的图像，无需则为空。

```
kp1, des1 = orb.detectAndCompute(img1, None)
kp2, des2 = orb.detectAndCompute(img2, None)
```

创建 BFMatcher 对象：BFMatcher 属于 features2d 模块，继承自 DescriptorMatcher，其 create() 函数如下：**static Ptr<BFMatcher> create(int normType , bool crossCheck)。**

1) 第一个参数“**normType**”：NORM_L1，NORM_L2，NORM_HAMMING，NORM_HAMMING2 四种可选。SIFT 和 SURF 的 HOG 描述符，对应欧氏距离 L1 和 L2；ORB 和 BRISK 的 BRIEF 描述符，对应汉明距 HAMMING；HAMMING2 则对应当 WTA_K = 3 或 4 时的 ORB 算法。

- 2) 欧氏距离：最常用的一种距离定义，指的是 n 维空间中，两点之间的实际距离

$$L1 = \sum_I |\text{src1}(I) - \text{src2}|$$

$$L2 = \sqrt{\sum_I (\text{src1}(I) - \text{src2}(I))^2}$$

汉明距离：计算机的异或操作，适用于二进制串描述符，如 BRIEF 描述符，定义如下：

$$\text{Hamming}(a, b) = \sum_{i=0}^{n-1} (a_i \oplus b_i)$$

3) 第二个参数“**crossCheck**”：默认为 FALSE。如果设置为 TRUE，只有当两组中特征点互相匹配时才算匹配成功。也就是说 A 组中 x 点描述符的最佳匹配点是 B 组的 y 点，那么 B 组的 y 点的描述符最佳匹配点也要是 A 组的 x 点才算匹配成功。

```
bf = cv2.BFMatcher_create(cv2.NORM_HAMMING, crossCheck=True)
```

匹配描述符：使用函数 detectAndCompute，具体格式和参数如下：

match(queryDescriptors, trainDescriptors)

- 1) 第一个参数 “**queryDescriptors**” 是需要匹配的图像特征向量。
- 2) 第二个参数 “**trainDescriptors**” 是被匹配的图像特征向量。

```
matches = bf.match(des1,des2)
```

画出匹配结果：使用函数 **drawMatches**，具体格式和参数如下：

drawMatches(src1,kp1,src2,kp2,match,matchesMask,flags)

- 1) 第一个参数 “**src1**”，是匹配图像 1。
- 2) 第二个参数 “**kp1**”，是图像 1 的特征点。
- 3) 第三个参数 “**src2**”，是匹配图像 2。
- 4) 第四个参数 “**kp2**”，是图像 2 的特征点。
- 5) 第五个参数 “**match**”，是需要绘制的匹配点集合。
- 6) 第六个参数 “**matchesMask**”，是决定需要绘制哪些图像，为空则全部绘制。
- 7) 第七个参数 “**flags**”，是指定绘图的标志位，0 为全部绘制，2 为绘制 match 匹配中的，4 为不同的绘制样式。

```
img3 = cv2.drawMatches(img1, kp1, img2, kp2, matches[:10], None, flags=2)
```

显示图像：imshow 函数显示图像，参数为显示窗口的标题和显示的图像。

关闭窗口：waitKey 函数等待按键按下后，执行 destroyAllWindows 函数关闭窗口。

```
cv2.imshow("show",img3)  
cv2.waitKey()  
cv2.destroyAllWindows()
```

2.最近邻匹配

FLANN 是一个开源库，全称 Fast Library for Approximate Nearest Neighbors，它实现了一系列高维向量的近似最近邻搜索算法。

基于 FLANN 库的最近邻匹配算子 `FlannBasedMatcher`，在特征数据集较大或一些实时处理领域，其运行效率要远高于 `BFMatcher`。

接下来使用最近邻匹配，匹配两幅图片的特征。



2.1 实验步骤

注意：

1) 需要先将目录“第四章 OpenCV 计算机视觉学习->图像处理进阶篇->第 14 课 图像处理——特征匹配->例程源码”下的例程“`flann_demo.py`”和示例图片“`test.jpg`”，“`test1.jpg`”复制到共享文件夹。

2) 共享文件夹的配置方法可查看目录“第 2 章 Linux 系统简介及使用入门->Linux 基础课程->第 3 课 Linux 系统安装及换源方法”下的文档。

3) 输入指令时需严格区分大小写，且可以使用“Tab”键补齐关键字。

1) 打开虚拟机，启动系统。点击系统任务栏的图标，并点击图标，或使用快捷键“`Ctrl+Alt+T`”，打开命令行终端。

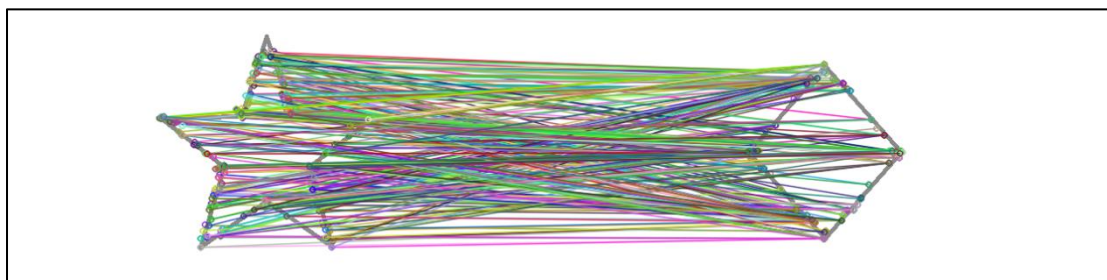
2) 输入指令“`cd /mnt/hgfs/share/`”，并按下回车，进入共享文件夹。

```
hiwonder@ubuntu:~$ cd /mnt/hgfs/Share/
```

3) 输入指令“`python3 flann_demo.py`”，并按下回车，运行例程。

```
ubuntu@ubuntu-virtual-machine:/mnt/hgfs/share$ python3 flann_demo.py
```

2.2 实现效果



2.3 代码分析

可在目录“第4章 OpenCV 计算机视觉学习->图像处理进阶篇->第14课 图像处理——特征匹配->例程源码”下查看例程“flann_demo.py”。

```
import numpy as np
import cv2 as cv

img1 = cv.imread('test.jpg',cv.IMREAD_GRAYSCALE) # 索引图像
img2 = cv.imread('test1.jpg',cv.IMREAD_GRAYSCALE) # 训练图像

# 初始化SIFT描述符
orb = cv.ORB_create()

# 基于SIFT找到关键点和描述符
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# FLANN的参数
FLANN_INDEX_LSH = 6
index_params= dict(algorithm = FLANN_INDEX_LSH,
                    table_number = 6, # 12
                    key_size = 12, # 20
                    multi_probe_level = 1) #2
search_params = dict(checks=50) # 或传递一个空字典
flann = cv.FlannBasedMatcher(index_params,search_params)

matches = flann.knnMatch(des1,des2,k=2)

img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,matches,None)
cv.imshow("show",img3)
cv.waitKey()
cv.destroyAllWindows()
```

设置 FLANN 的参数：使用 FlannBasedMatcher 函数，其格式和参数如下：

FlannBasedMatcher (IndexParams,SearchParams)，两个参数都是字典类型。

- 1) 第一个参数“**IndexParams**”是指定使用的算法。
- 2) 第二个参数“**SearchParams**”是指定索引中的树应递归遍历的次数。较高的值可提供更好的精度，但也需要更多时间。

```
index_params= dict(algorithm = FLANN_INDEX_LSH,
                    table_number = 6, # 12
                    key_size = 12, # 20
                    multi_probe_level = 1) #2
search_params = dict(checks=50) # 或传递一个空字典
flann = cv.FlannBasedMatcher(index_params,search_params)
```

最近邻匹配：使用 knnMathch 函数，其格式和参数如下：

knnMathch(queryDescriptors,trainDescriptors,k)，前两个参数与 match 匹配一致，“k”是需要返回的最佳匹配数量。

```
matches = flann.knnMatch(des1,des2,k=2)
```