sift、surf、orb 特征提取及最优特征点匹配

目录

sift

sift特征简介

SIFT(Scale-Invariant Feature Transform)特征，即尺度不变特征变换，是一种计算机视觉的特征提取算法，用来侦测与描述图像中的局部性特征。

实质上，它是在不同的尺度空间上查找关键点(特征点)，并计算出关键点的方向。SIFT所查找到的关键点是一些十分突出、不会因光照、仿射变换和噪音等因素而变化的点，如角点、边缘点、暗区的亮点及亮区的暗点等。

sift特征提取步骤

1. 尺度空间的极值检测： 尺度空间指一个变化尺度 `(σ)` 的二维高斯函数 `G(x,y,σ)` 与原图像 `I(x,y)` 卷积（即高斯模糊）后形成的空间,尺度不变特征应该既是空间域上又是尺度域上的局部极值。极值检测的大致原理是根据不同尺度下的高斯模糊化图像差异（Difference of Gaussians,DoG）寻找局部极值，这些找到的极值所对应的点被称为关键点或特征点。

2. 关键点定位： 在不同尺寸空间下可能找出过多的关键点，有些关键点可能相对不易辨识或易受噪声干扰。该步借由关键点附近像素的信息、关键点的尺寸、关键点的主曲率来定位各个关键点，借此消除位于边上或是

易受噪声干扰的关键点。

3. 方向定位： 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。通过计算关键点局部邻域的方向直方图，寻找直方图中最大值的方向作为关键点的主方向。

4. 关键点描述子： 找到关键点的位置、尺寸并赋予关键点方向后，将可确保其移动、缩放、旋转的不变性。此外还需要为关键点建立一个描述子向量，使其在不同光线与视角下皆能保持其不变性。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，见下图。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在原论文中建议描述子使用在关键点尺度空间内44的窗口中计算的8个方向的梯度信息，共44*8=128维向量表征。(opencv中实现的也是128维)

具体可以参考这篇博客：https://www.cnblogs.com/liuchaogege/p/5155739.html

surf

surf特征简介

SURF(Speeded Up Robust Features, 加速稳健特征) 是一种稳健的图像识别和描述算法。它是SIFT的高效变种，也是提取尺度不变特征，算法步骤与SIFT算法大致相同，但采用的方法不一样，要比SIFT算法更高效（正如其名）。SURF使用海森(Hesseian)矩阵的行列式值作特征点检测并用积分图加速运算；SURF 的描述子基于 2D 离散小波变换响应并且有效地利用了积分图。

surf特征提取步骤

1. 特征点检测： SURF使用Hessian矩阵来检测特征点，该矩阵是x,y方向的二阶导数矩阵，可测量一个函数的局部曲率，其行列式值代表像素点周围的变化量，特征点需取行列式值的极值点。用方型滤波器取代SIFT中的高斯滤波器，利用积分图（计算位于滤波器方型的四个角落值）大幅提高运算速度。

2.特征点定位： 与SIFT类似，通过特征点邻近信息插补来定位特征点。

3. 方向定位： 通过计算特征点周围像素点x,y方向的哈尔小波变换，并将x,y方向的变换值在xy平面某一角度区间内相加组成一个向量，在所有的向量当中最长的(即x、y分量最大的)即为此特征点的方向。

4. 特征描述子： 选定了特征点的方向后，其周围相素点需要以此方向为基准来建立描述子。此时以55个像素点为一个子区域，取特征点周围2020个像素点的范围共16个子区域，计算子区域内的x、y方向(此时以平行特征点方向为x、垂直特征点方向为y)的哈尔小波转换总和Σdx、Σdy Σdx、Σdy与其向量长度总和Σ|dx|、Σ|dy| Σ|dx|、Σ|dy|共四个量值，共可产生一个64维的描述子。

具体可以参考这篇博客：
https://www.cnblogs.com/zyly/p/9531907.html

orb

orb特征简介

ORB（Oriented FAST and Rotated BRIEF）该特征检测算法是在著名的FAST特征检测和BRIEF特征描述子的基础上提出来的，其运行时间远远优于SIFT和SURF，可应用于实时性特征检测。ORB特征检测具有尺度和旋

转不变性，对于噪声及其透视变换也具有不变性，良好的性能是的利用ORB在进行特征描述时的应用场景十分广泛。ORB特征检测主要分为以下两个步骤:(1)方向FAST特征点检测(2)BRIEF特征描述。

orb特征提取算法

1. FAST特征点检测：https://www.cnblogs.com/ronny/p/4078710.html
2. BRIEF特征描述子：https://www.cnblogs.com/ronny/p/4081362.html

代码实现

接下来的代码采用的库如下图所示

Package	Version	Latest version
numpy	1.17.0	1.17.0
opencv-contrib-python	3.4.2.16	▲ 4.1.0.25
opencv-python	3.4.2.16	▲ 4.1.0.25
pip	19.2.1	19.2.1
setuptools	39.1.0	▲ 41.0.1

红色框的那两个库非常重要！版本请使用**3.4.2.16**的，而不是最新的，否则在特征提取的时候会报错。

```
错误提示: sift = cv2.xfeatures2d.SIFT_create()
cv2.error: OpenCV(3.4.3) C:\projects\opencv-
python\opencv_contrib\modules\xfeatures2d\src\sift.cpp:1207: error: (-213:The
function/feature is not implemented) This algorithm is patented and is excluded in this
configuration; Set OPENCV_ENABLE_NONFREE CMake option and rebuild the library in function
'cv::xfeatures2d::SIFT::create'
```

如果你在使用cv2.xfeatures2d.SIFT_create()这个函数的时候出现了上面的错误，就是因为你的库版本太新。把版本退回去就可以了。

特征提取

```
def sift(filename):
    img = cv2.imread(filename) # 读取文件
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # 转化为灰度图
    sift = cv2.xfeatures2d_SIFT.create()
    keyPoint, descriptor = sift.detectAndCompute(img, None) # 特征提取得到关键点以及对应的描述符
(特征向量)
    return img,keyPoint, descriptor
```

```
def surf(filename):
    img = cv2.imread(filename) # 读取文件
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # 转化为灰度图
    sift = cv2.xfeatures2d_SURF.create()
    keyPoint, descriptor = sift.detectAndCompute(img, None) # 特征提取得到关键点以及对应的描述符
(特征向量)
    return img, keyPoint, descriptor
```

```
def orb(filename):
    img = cv2.imread(filename) # 读取文件
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # 转化为灰度图
    sift = cv2.ORB_create()
    keyPoint, descriptor = sift.detectAndCompute(img, None) # 特征提取得到关键点以及对应的描述符（特征向量）
    return img, keyPoint, descriptor
```

这里解释一下为什么要进行转化为灰度图?

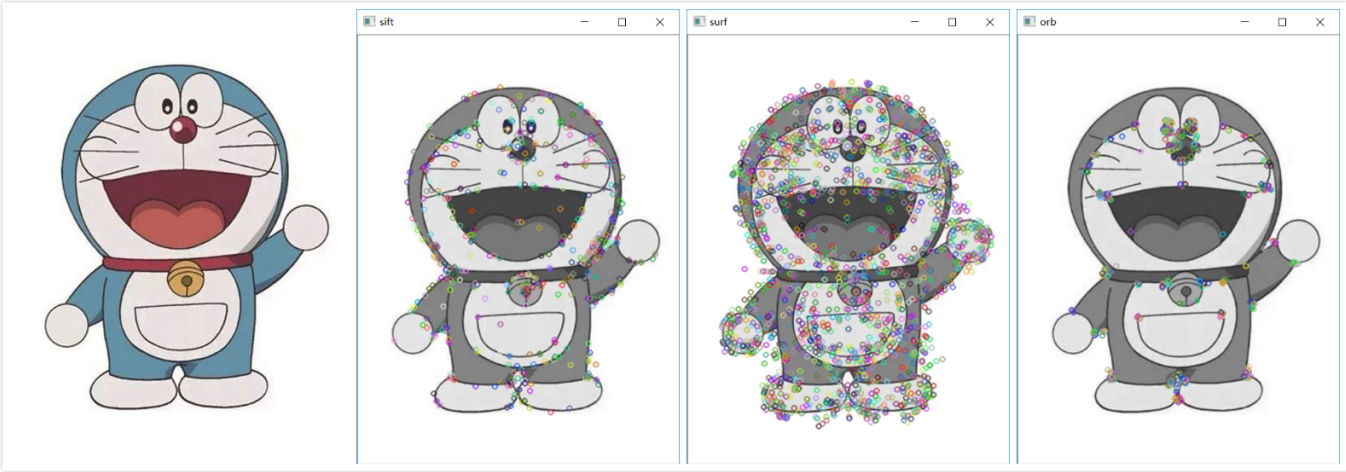
> 1. 识别物体，最关键的因素是梯度（SIFT/HOG），梯度意味着边缘，这是最本质的部分，而计算梯度，自然就用到灰度图像了，可以把灰度理解为图像的强度。
> 2. 颜色，易受光照影响，难以提供关键信息，故将图像进行灰度化，同时也可以加快特征提取的速度。

比较一下提取的结果看看

```
def compare(filename):
    imgs = []
    keyPoint = []
    descriptor = []
    img, keyPoint_temp, descriptor_temp = sift(filename)
    keyPoint.append(keyPoint_temp)
    descriptor.append(descriptor_temp)
    imgs.append(img)
    img, keyPoint_temp, descriptor_temp = surf(filename)
    keyPoint.append(keyPoint_temp)
    descriptor.append(descriptor_temp)
    imgs.append(img)
    img, keyPoint_temp, descriptor_temp = orb(filename)
    keyPoint.append(keyPoint_temp)
    descriptor.append(descriptor_temp)
    imgs.append(img)
    return imgs, keyPoint, descriptor

def main():
    method = ['sift','surf','orb']
    imgs, kp, des = compare('./pic/doraemon1.jpg')
    for i in range(3):
        img = cv2.drawKeypoints(imgs[i], kp[i], None)
        cv2.imshow(method[i], img)
        cv2.waitKey()
        cv2.destroyAllWindows()
    print("sift len of des: %d, size of des: %d" % (len(des[0]), len(des[0][0])))
    print("surf len of des: %d, size of des: %d" % (len(des[1]), len(des[1][0])))
    print("orb len of des: %d, size of des: %d" % (len(des[2]), len(des[2][0])))
```

下图是提取的结果，从左到右分别是原图、sift、surf、orb




```
sift len of des: 458, size of des: 128
surf len of des: 1785, size of des: 64
orb len of des: 500, size of des: 32
```

可以看出：

- sift虽然提取的特征点最少，但是效果最好。
- sift提取的特征点维度是128维，surf是64维，orb是32维。

特征匹配

BruteForce匹配和**FLANN匹配**是opencv二维特征点匹配常见的两种办法，分别对应BFMatcher（BruteForceMatcher）和FlannBasedMatcher。

二者的区别在于BFMatcher总是尝试所有可能的匹配，从而使得它总能够找到最佳匹配，这也是Brute Force（暴力法）的原始含义。而FlannBasedMatcher中FLANN的含义是Fast Library forApproximate Nearest Neighbors，从字面意思可知它是一种近似法，算法更快但是找到的是最近邻近似匹配，所以当我们需要找到一个相对好的匹配但是不需要最佳匹配的时候往往使用FlannBasedMatcher。当然也可以通过调整FlannBasedMatcher的参数来提高匹配的精度或者提高算法速度，但是相应地算法速度或者算法精度会受到影响。

本文是进行最优特征点匹配，因此选用BruteForce Matcher。

```
def match(filename1, filename2, method):
    if(method == 'sift'):
        img1, kp1, des1 = sift(filename1)
        img2, kp2, des2 = sift(filename2)
        bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)  # sift的normType应该使用NORM_L2或者
NORM_L1
        matches = bf.match(des1, des2)
        matches = sorted(matches, key=lambda x: x.distance)
        knnMatches = bf.knnMatch(des1, des2, k=1)  # drawMatchesKnn
    if (method == 'surf'):
        img1, kp1, des1 = surf(filename1)
        img2, kp2, des2 = surf(filename2)
        bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)  # surf的normType应该使用NORM_L2或者
NORM_L1
        matches = bf.match(des1, des2)
```

```
        matches = sorted(matches, key=lambda x: x.distance)
        knnMatches = bf.knnMatch(des1, des2, k=1)  # drawMatchesKnn
    if(method == 'orb'):
        img1, kp1, des1 = orb(filename1)
        img2, kp2, des2 = orb(filename2)
        bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = True) # orb的normType应该使用
NORM_HAMMING
        matches = bf.match(des1, des2)
        matches = sorted(matches, key=lambda x: x.distance)
        knnMatches = bf.knnMatch(des1, des2, k = 1) # drawMatchesKnn
    # 过滤
    for m in matches:
        for n in matches:
            if(m != n and m.distance >= n.distance*0.75):
                matches.remove(m)
                break
    img = cv2.drawMatches(img1, kp1, img2, kp2, matches[:50], img2, flags=2)
    cv2.imshow("matches", img)
    cv2.waitKey()
    cv2.destroyAllWindows()
def main():
    method = ['sift','surf','orb']
    for i in range(3):
        match('./pic/wechat1.jpg', './pic/wechat2.png', method[i])
if __name__ == '__main__':
    main()
```

介绍一下几个关键函数。

首先是 `cv2.BFMatcher(normType, corssCheck)` 函数。它有两个参数。

- 第一个参数是用来指定要使用的距离测试类型。默认值为 cv2.Norm_L2。这很适合 SIFT 和 SURF 等(cv2.NORM_L1 也可以)。对于使用二进制描述符的 ORB,BRIEF,BRISK算法等,要使用 cv2.NORM_HAMMING,这样就会返回两个测试对象之间的汉明距离。
- 第二个参数是布尔变量 crossCheck,默认值为 False。如果设置为True,匹配条件就会更加严格,只有到 A 中的第 i 个特征点与 B 中的第 j 个特征点距离最近,并且 B 中的第 j 个特征点到 A 中的第 i 个特征点也是最近(A 中没有其他点到 j 的距离更近)时才会返回最佳匹配(i,j)。也就是这两个特征点要互相匹配才行。

然后是 `bf.match()` 。它也有两个参数。前面一个是查询用的向量，后面一个是匹配用的向量。

`sorted()` 函数是用来对匹配得到的结果进行排序，按照距离排序。

`knnMatch()` 是BFMatcher 对象的另一个方法, BFMatcher.match()方法会返回最佳匹配。而该方法为每个关键点返回 k 个最佳匹配(降序排列之后取前 k 个),其中 k 是由用户设定的。
（注意：knnMatch()和match()得到的返回并不是一样的结果）

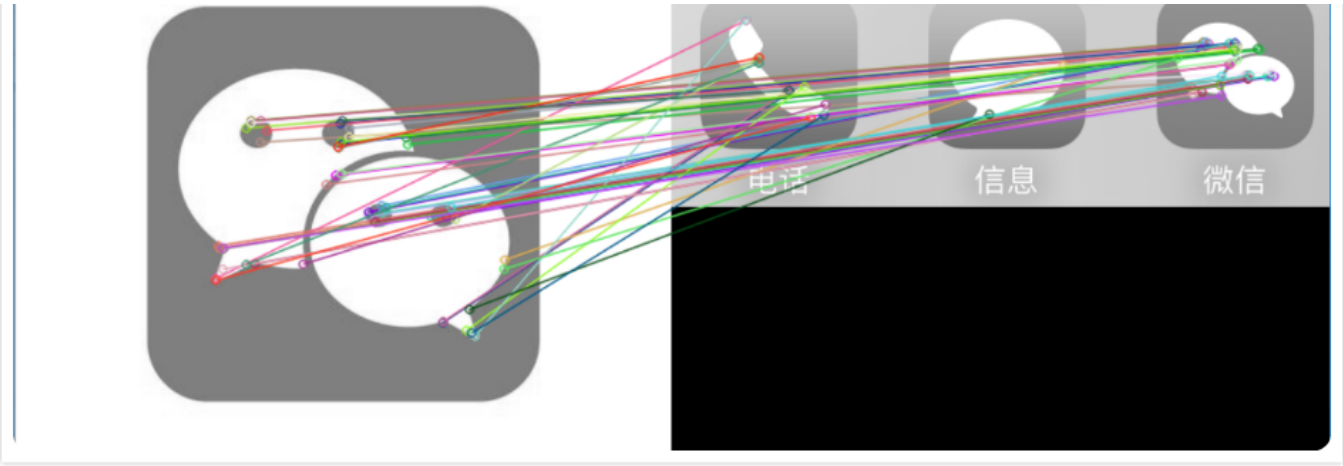
这里还对匹配得到的结果做了过滤，排除一些不好的匹配结果。

`drawMatch()` 函数可以画出两张图的匹配点。参数如下：

- img1 – 源图像1
- keypoints1 –源图像1的特征点.
- img2 – 源图像2.

- keypoints2 – 源图像2的特征点
- matches1to2 – 源图像1的特征点匹配源图像2的特征点
- outImg – 输出图像具体由flags决定.
- matchColor – 匹配的颜色（特征点和连线),若matchColor==Scalar::all(-1)，颜色随机.
- singlePointColor – 单个点的颜色，即未配对的特征点，若matchColor==Scalar::all(-1)，颜色随机.
- matchesMask – Mask决定哪些点将被画出，若为空，则画出所有匹配点.
- flags – Fdefined by DrawMatchesFlags.

接下来看一下上面的代码运行的结果。从上到底依次是原图、sift、surf、orb




```
sift size of kp: 59, after filtering: 20
surf size of kp: 197, after filtering: 35
orb size of kp: 390, after filtering: 47
```

从输出的结果来看，orb的效果最好。感兴趣的话还可以用其他图片看看效果，pic文件夹还提供其他两组比较的图片。

总结

基于特征的匹配分为特征点提取和匹配两个步骤，本篇主要针对特征点提取三种方法进行比较，分别是SIFT，SURF以及ORB三种方法，这三种方法在OpenCV里面都已实现。SURF基本就是SIFT的全面升级版，有 SURF 基本就不用考虑SIFT，而ORB的强点在于计算时间，以下具体比较：

计算速度：ORB>>SURF>>SIFT（各差一个量级）

旋转鲁棒性：SURF>ORB~SIFT（表示差不多）

模糊鲁棒性：SURF>ORB~SIFT

尺度变换鲁棒性：SURF>SIFT>ORB（ORB并不具备尺度变换性）

所以结论就是，如果对计算实时性要求非常高，可选用ORB算法，但基本要保证正对拍摄；如果对实行性要求稍高，可以选择SURF；基本不用SIFT。
参考：https://blog.csdn.net/zilanpotou182/article/details/66478915
不过上面那篇博客的评论提出了不同的看法，正确性有待验证。

附录

GitHub：https://github.com/Professorchen/Computer-Vision/tree/master/feature-extraction