# OpenCV学习笔记（十九）之Harris角点检测

**一. Harris角点检测**

**ＡPI cv::cornerHarris**

cv**::**cornerHarris(

InputArray src, *//灰度图像*

OutputArray dst, *//响应输出，可能为小数*

**int** blockSize, *//矩阵大小*

**int** ksize, *//窗口大小*

**double** k, *//计算角度响应时的参数大小，默认在0.04-0.06*

**int** borderType**=**BORDER\_DEFAULT

)

**二. 代码实现：**

*/\*\*\*\*\*Harris角点检测\*\*\*\*\*/*

**#include**<opencv2/opencv.hpp>

**#include**<iostream>

cv**::**Mat src, gray\_src;

**int** thresh **=** 50;

**int** thresh\_max **=** 255;

**void** **Harris\_Demo**(**int**, **void\***);

**using** **namespace** std;

**int** **main**(**int** argc, **char\*\*** argv) {

src **=** cv**::**imread("建筑.jpg");

cv**::**namedWindow("in", cv**::**WINDOW\_NORMAL);

cv**::**imshow("in", src);

cv**::**namedWindow("ou", cv**::**WINDOW\_NORMAL);

*//转灰度图*

cv**::**cvtColor(src, gray\_src, cv**::**COLOR\_BGR2GRAY);

cv**::**createTrackbar("Threshold Value: ", "ou", **&**thresh, thresh\_max,

Harris\_Demo);

Harris\_Demo(0, 0);

cv**::**waitKey(0);

**return** 0;

}

**void** **Harris\_Demo**(**int**, **void\***) {

cv**::**Mat dst, norm\_dst, resultImg, normScaleDst;

dst **=** cv**::**Mat**::**zeros(gray\_src.size(), CV\_32FC1);

*//角点检测参数准备*

**int** blockSize **=** 2;

**int** kSize **=** 3;

**double** k **=** 0.04;

cv**::**cornerHarris(gray\_src, dst, blockSize, kSize, k, cv**::**BORDER\_DEFAULT);

*//灰度图归一化到0-255*

cv**::**normalize(dst, norm\_dst, 0, 255, cv**::**NORM\_MINMAX, CV\_32FC1, cv**::**Mat());

*//绝对值*

cv**::**convertScaleAbs(norm\_dst, normScaleDst);

*//在原图上显示角点的位置*

resultImg **=** src.clone();

**for** (**int** row **=** 0; row **<** resultImg.rows; **++**row) {

*//用行指针*

uchar**\*** currentRow **=** normScaleDst.ptr(row);

**for** (**int** col **=** 0; col **<** resultImg.cols; **++**col) {

*//像素值*

**int** value **=** (**int**)**\***currentRow;

*//像素值小于阈值*

**if** (value **>** thresh) {

*//在角点画个小圆*

circle(resultImg, cv**::**Point(row, col), 2, cv**::**Scalar(0, 0, 255),

2, 8, 0);

}

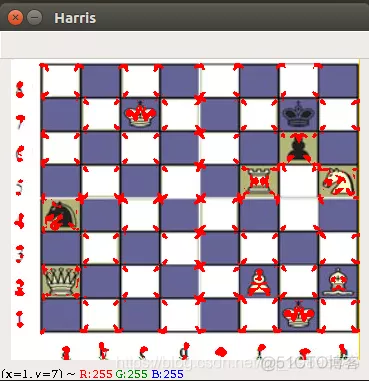
currentRow**++**;

}

}

cv**::**imshow("ou", resultImg);

}



**OpenCv学习笔记--Harris角点检测**

**此opencv系列博客只是为了记录本人对<<opencv3计算机视觉-pyhton语言实现>>的学习笔记,所有代码在我的github主页**[**https://github.com/RenDong3/OpenCV\_Notes**](https://github.com/RenDong3/OpenCV_Notes)**.**

OpenCV可以**检测图像的主要特征，然后提取这些特征、使其成为图像描述符**，这类似于人的眼睛和大脑。这些图像特征可作为图像搜索的数据库。此外，人们可以利用这些关键点将图像拼接起来，组成一个更大的图像，比如将许多图像放在一块，然后形成一个360度全景图像。

这里我们将学习**使用OpenCV来检测图像特征，并利用这些特征进行图像匹配和搜索**。我们会选取一些图像，并通过单应性，检测这些图像是否在另一张图像中。

**一 特征检测算法**

有许多用于特征检测和提取的算法，我们将会对其中大部分进行介绍。OpenCV最常使用的特征检测和提取算法有：

* Harris：该算法用于检测角点；
* SIFT：该算法用于检测斑点；
* SURF：该算法用于检测角点；
* FAST：该算法用于检测角点；
* BRIEF：该算法用于检测斑点；
* ORB：该算法代表带方向的FAST算法与具有旋转不变性的BRIEF算法；

通过以下方法进行特征匹配：

* 暴力(Brute-Force)匹配法；
* 基于FLANN匹配法；

可以采用单应性进行空间验证。

**二 特征定义**

那么，究竟什么是特征呢？为什么一副图像的某个特定区域可以作为一个特征，而其他区域不能呢？粗略的讲，**特征就是有意义的图像区域，该区域具有独特特征和易于识别性**。因此角点及高密度区域都是很好的特征，而大量重复的模式或低密度区域(例如图像中的蓝色天空)则不是很好的特征。边缘可以将图像分为两个区域，因此也可以看做好的特征。斑点是与周围有很大差别的像素区域，也是有意义的特征。

大多数特征检测算法都会涉及图像的角点、边和斑点的识别，也有一些涉及脊向的概念，可以认为脊向是细长物体的对称轴，例如识别图像中的一条路。角点和边都和哦理解，那什么是斑点呢？斑点通常是指与周围有着颜色和灰度差别的区域。在实际地图中，往往存在着大量这样的斑点，如一颗树是一个斑点，一块草地是一个斑点，一栋房子也可以是一个斑点。由于斑点代表的是一个区域，相比单纯的角点，它的稳定性要好，抗噪声能力要强，所以它在图像配准上扮演了很重要的角色。

由于某些算法在识别和提取某类型特征的时候有较好的效果，所以知道输入图像是什么很重要，这样做有利于选择最合适的OpenCV工具包。

**三 Harris检测角点特征**

下面我们从使用cornerHarris()函数讲起。

cv2.cornerHarris(src, blockSize, ksize, k[, dst[, borderType]]) → dst

* 1.

**参数详解:**

* image:输入的单通道8位或者浮点图像；
* blockSize：就是扫描时候窗口的大小。
* ksize：cornerHarris函数会使用Sobel算子，该参数定义了Sobel算子的中孔。简单来说，该函数定义了角点检测的敏感度，其值必须介于3~31之间的奇数。
* k：harris 计算响应公式中的kk值，一般取0.04~0.06；
* borderType：像素插值方法；

函数 cornerHarris 对输入图像进行 Harris 边界检测。输出是一幅浮点值图像，大小与输入图像大小相同，浮点值越高，表明越可能是特征角点(我们可以对图像进行阈值化)。

 完整代码如下:

#-\*- coding:utf-8 -\*-

import os

import cv2

import numpy as np

'''

created on 08:05:10 2018-11-20

@author ren\_dong

使用cornerHarris进行角点检测

cv2.cornerHarris(src, blockSize, ksize, k[, dst[, borderType]]) → dst

'''

img = cv2.imread('chess.jpg')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

gray= np.float32(gray)

#第三个参数定义了角点检测的敏感度,其值必须是介于3和31之间的奇数

##dst为函数返回的浮点值图像,其中包含角点检测结果

#第二个参数值决定了标记角点的记号大小,参数值越小,记号越小

dst = cv2.cornerHarris(gray, 4, 23, 0.04)

print dst.shape

#在原图进行标记 阈值

img[dst > 0.01\*dst.max()] = [0, 0, 255]

while(True):

cv2.imshow('Harris', img)

if cv2.waitKey(100) & 0xff == ord('q'):

break

cv2.destroyAllWindows()

运行结果如下:

本人只是对Harris角点检测code进行了复现,对于其原理并没有深究,所以这里并未列出其具体原理,感兴趣的童鞋可以去到本人一直作为参考的大神的博客去观看浏览,里面有对原理的详细解释.链接放在下面了.

javascript:void(0)

**相关文章**

**[Opencv Harris角点检测](https://blog.51cto.com/u_15127594/4069216" \t "_blank)**

[#include <iostream>#include <opencv2/opencv.hpp> using namespace std;using namespace cv; Mat img1, img2, img3, img4, img5, img6, img\_result, img\_gray1](https://blog.51cto.com/u_15127594/4069216" \t "_blank)

**[OpenCV Harris 角点检测子](https://blog.51cto.com/u_15060467/4737663" \t "_blank)**

[本教程中我们将涉及: 是指 图像特征类型: 本教程涉及 角点 特征。 由于角点代表了图像像素梯](https://blog.51cto.com/u_15060467/4737663" \t "_blank)

**[OpenCV自学笔记14：Harris角点检测实例](https://blog.51cto.com/u_15762852/5613793" \t "_blank)**

[Harris角点检测实例本节使用到的图片：# -\*- coding:utf-8 -\*-import cv2import num](https://blog.51cto.com/u_15762852/5613793" \t "_blank)

**[OpenCV + CPP 系列（卅三）图像特征提取（Harris角点检测、Shi-Tomasi角点检测、自定义角点检测）](https://blog.51cto.com/u_13984132/5622619" \t "_blank)**

[常用图像特征描述：SIFTSURFHOGHaarLBPKAZEAKAZEBRISK图像特征描述子用于:DetectionDescriptionMatchingHarris角点检测Shi-Tomasi角点检测maxCorners 表示返回角点的数目](https://blog.51cto.com/u_13984132/5622619" \t "_blank)

**[OpenCv学习笔记12--FAST角点检测](https://blog.51cto.com/u_15351425/3725458" \t "_blank)**

[此opencv系列博客只是为了记录本人对<<opencv3计算机视觉-pyhton语言实现>>的学习笔记,所有代码在我的github主页https://github.com/RenDong3/OpenCV\_Notes.欢迎star,不定时更新...在前面我们已经陆续介绍了许多特征检测算子，我们可以根据图像局部的自相关函数求得Harris角点，后面又提到了两种十分优秀...](https://blog.51cto.com/u_15351425/3725458" \t "_blank)

**[【OpenCV入门之十九】Harris与Shi-Tomasi角点检测](https://blog.51cto.com/u_14411234/3112973" \t "_blank)**

[角点检测是计算机视觉系统中用来获取图像特征的一种方法。我们都常说，这幅图像很有特点，但是一问他到底有哪些特点，或者这幅图有哪些特征可以让你一下子就识别出该物体，你可能就说不出来了。其实说图像的特征，你可以尝试说一下这幅图有几个矩形啊几个圆形啊，有几条直线啊，当然啦，你也可以说一下有几个角点。](https://blog.51cto.com/u_14411234/3112973" \t "_blank)

**[OpenCV——Harris、Shi Tomas、自定义、亚像素角点检测](https://blog.51cto.com/u_15060515/4027412" \t "_blank)**

[1 #include <opencv2/opencv.hpp> 2 #include <iostream> 3 4 using namespace cv; 5 using namespace std; 6 7 8 9 int main(int argc, char\*\* argv)10 {11 Mat src;12 src = imrea](https://blog.51cto.com/u_15060515/4027412" \t "_blank)

**[OpenCV ORB角点检测](https://blog.51cto.com/u_15669955/5351328" \t "_blank)**

[实例化ORBorb = cv.ORB\_create(nfeatures)参数：nfeatures：特征点的最大数量利用orb.detectAndCo](https://blog.51cto.com/u_15669955/5351328" \t "_blank)

**[OpenCV BRIEF角点检测](https://blog.51cto.com/u_15669955/5351330" \t "_blank)**

[BRIEF是一种特征描述子提取算法，并非特征点的提取算法，一种生成二值化描述子的算法，不提取代价低](https://blog.51cto.com/u_15669955/5351330" \t "_blank)

**[OpenCV Fast角点检测](https://blog.51cto.com/u_15669955/5351332" \t "_blank)**

[import numpy as npimport cv2 as cvfrom matplotlib import pyplot as plt# 1.读取图像img = cv.imread("./1.jpg")# 2.Fast角点检测# 2.1创](https://blog.51cto.com/u_15669955/5351332" \t "_blank)

**[OpenCV 之 角点检测](https://blog.51cto.com/u_15127644/3258101" \t "_blank)**

[角点 (corners) 的定义有两个版本：一是 两条边缘的交点，二是 邻域内具有两个主方向的特征点。 一般而言，角点是边缘曲线上曲率为极大值的点，或者 图像亮度发生剧烈变化的点。例如，从人眼角度来看，下图的 $E$ 和 $F$ 便是典型的角点 1 检测思路 在图像中定义一个局部小窗口，然后沿各个方 ...](https://blog.51cto.com/u_15127644/3258101" \t "_blank)

**[通俗易懂的Harris 角点检测](https://blog.51cto.com/u_14439393/5732385" \t "_blank)**

[Harris 角点检测 认为 特征点具有局部差异性。如何描述“特征点具有局部差异性”：以每一个点为中心，取一个窗口，窗口大小为5\*5或7\*7像素大小。这个窗口描述了这个特征点周围的环境。如果这个特征点具有局部差异性，那么以这个特征点为中心，把窗口向360度任意一个方向移动，窗口的变化比较大，则这个特征点的周围环境变化比较大。数学公式：（x,y）：表示像素的位置。：表示窗口内的每个像素。w](https://blog.51cto.com/u_14439393/5732385" \t "_blank)

# OpenCV入门学习笔记之Harris角点检测与SIFT特征匹配算法

## 1. 写在前面

这篇文章整理两个图像处理中非常重要的算法，一个是[Harris角点检测](https://aitechtogether.com/tag/harris%e8%a7%92%e7%82%b9%e6%a3%80%e6%b5%8b)算法，另一个是[SIFT特征匹配](https://aitechtogether.com/tag/sift%e7%89%b9%e5%be%81%e5%8c%b9%e9%85%8d)算法，这两个算法本质上还是去找图像里面的关键特征点，帮助我们后续更好的理解图像以及做各种各样的分析。 由于这两个算法涉及到的数学原理会比较多，而我刚入门，所以只是从使用的角度，简单的描述到底在做什么事情，至于详细的数学细节或者推导，这里不过多整理，以掉包能完成任务为首要目的啦。

首先，先介绍Harris角点检测算法，角点在图像中是很重要的特征，信息含量很高，那么如何找到一个图像里面的角点呢？ 这个算法就能轻松解决。但是呢？这个算法只考虑了旋转不变性，即一个角点旋转之后还是角点，没有考虑到尺度不变，尺度变化可能会导致角点变成边缘，所以有没有一种方法同时考虑这两个特性去找特征点呢？ 这就是SIFT算法，这个算法也是用来侦测和描述影像中的局部性特征，基于位置，尺度，旋转不变性在空间尺度中寻找特征点。 这两个算法的使用场景非常广泛，比如图像配准，目标识别跟踪，图像对齐，图像拼接(全景图)， 相机标定，三维场景重建等。最后，通过一个[全景图拼接](https://aitechtogether.com/tag/%e5%85%a8%e6%99%af%e5%9b%be%e6%8b%bc%e6%8e%a5)的Demo感受下这两个算法的魅力 😉

大纲如下：

* Harris角点检测算法
* SIFT特征匹配算法
* 特征匹配
* 案例Demo: 全景图拼接

Ok, let’s go!

## 2. Harris角点检测算法

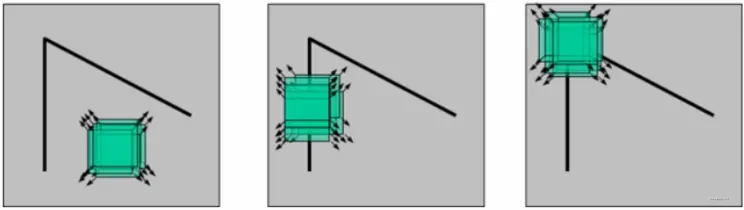
角点特征能在保留图像重要特征的同时，有效减少信息数据量，算是图像中较好的特征，比边缘特征更好的用于定位。 在图像处理中，检测角点特征的算法很多，最常用，最基础的就是Harris角点检测算法。

在说这个算法之前，我们先来感受一下什么是角点：

如果我们用人眼看，很容易理解，也就是图像中物体的一角。然后计算机可以看到，要定位这些角点并不容易。想让电脑看到，就必须明白一件事：这样的角点、边界点，和普通的屏幕点有什么区别，也就是图中红框、蓝框、黑框内的点。上图在数值上会有什么不同？

Harris

角点检测认为：特征点具有局部差异性。如何描述呢？ 每一个点为中心，取一个窗口，比如窗口大小或者， 那么窗口描述了这个特征点周围的环境。

* 如果特定窗口在图像的各个方向移动时，窗口中图像的灰度没有变化，那么窗口中不存在角点，比如蓝框
* 如果窗口向某个方向移动，窗口中图像的灰度变化很大，而其他方向没有变化，那么窗口中的图像可能是一条直线边界，比如黑框，而垂直运动剧烈变化，水平方向不变
* 如果这个特征的小窗口向四面八方移动，窗口内区域的灰度变化很大，则认为窗口存在角点。  
  

说是这么说，但具体应该怎么找到角点呢？ Harris角点检测算法主要是三步：

1. 当窗口同时向x和y两个方向移动时，计算窗口内像素值的变化E(u,v)
2. 对于每个窗口，计算对应的角点响应函数R
3. 对于这个函数的阈值处理，如果R > threshold，表示窗口对应一个角点特征

下面详细展开。

### 2.1 窗口移动，计算像素值变化量

这里的核心问题：如何确定哪些窗口导致灰度值变化较大？

|  |
| --- |
|  |
| 这个公式是我们的目标函数。如果是角点，这个函数的值会比较大，所以我们最大化这个函数，得到图像中的角点。 |
|  |
|  |
|  |
|  |
|  |

### 2.2 窗口的角点响应函数

|  |
| --- |
|  |

### 2.3 非极大值抑制

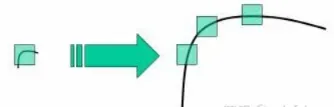
根据 R 的值，将这个窗口所在的区域划分为平面、边缘或角点。为了得到最优的角点，我们还可以使用非极大值抑制。

|  |
| --- |
|  |

一张图片胜过千言万语：

|  |
| --- |
|  |

Harris 角点检测的结果是带有这些分数R的灰度图像，设定一个阈值，分数大于这个阈值的像素就对应角点。

注意：Harris 检测器具有旋转不变性，但不具有尺度不变性，也就是说尺度变化可能会导致角点变为边缘，如下图所示：  


So， 如何找到旋转以及尺度不变的特征点呢？这就是SIFT算法干的事情了，但是介绍之前，先看看OpenCV中角点检测算法咋用。 理论一大推，但是用起来一个函数搞定。

### 2.4 OpenCV中的角点检测

这里的函数是cv2.cornerHarris()：

* img: 数据类型为float32的输入图像
* blockSize: 角点检测中指定区域的大小， 即w窗口大小
* ksize: Sobel求导中使用的窗口大小， 需要用sobel算子求梯度，这里是设置sobel算子求导的窗口
* k：取值参数为[0.04, 0.06]

看一个例子：

img = cv2.imread('img/test\_img.jpg')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# 角点检测

dst = cv2.cornerHarris(gray, 2, 3, 0.04) # 这个是每个像素点的E值，即平移后灰度级变换程度值

img[dst>0.01\*dst.max()] = [0, 0, 255]

Python

Copy

以下是角点检测结果：



## 3. SIFT算法

Scale Invariant Feature Transform(SIFT): 尺度不变特征转换用来侦测与描述影像中的局部性特征， 基于位置，尺度和旋转不变性在空间尺度中寻找极值点。

特征：

* SIFT特征是图像的局部特征，其对旋转、尺度缩放、亮度变化保持不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性
* 唯一性好，信息丰富，适用于数据库中海量特征的快速准确匹配
* 多量性，即使少数几个物体也可以产生大量的SIFT特征向量
* 高速和可扩展性，特征可以很容易地与其他向量结合

解决问题：目标自身状态，场景所处环境和成像器材的成像特性等因素影响图像配准/目标识别跟踪的性能。SIFT算法一定程度可解决：

1. 目标旋转、缩放、平移
2. 图像仿射/投影变换
3. 灯光效果
4. 目标遮挡
5. 杂物场景
6. 噪音

SIFT算法的实质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算法主要下面四步：

1. 尺度空间极值检测：在所有尺度上搜索图像位置，并通过高斯微分函数识别潜在的尺度和旋转不变特征点
2. 关键点定位：在每个候选位置，位置和规模由一个拟合良好的模型确定。关键点是根据它们的稳定性来选择的。
3. 方向确定：根据图像的局部梯度方向为每个关键点位置分配一个或多个方向。对图像数据的所有后续操作都相对于关键点的方向、比例和位置进行转换，从而为这些转换提供不变性。
4. 关键点描述：在每个关键点周围的邻域中，测量所选尺度下图像局部性的梯度。这些梯度被转换成允许相对较大的局部形状变形和光照变化的表示。

由于这个算法理论上稍微复杂一些，下面是一个简单的安排。首先，我们要先了解这个算法是做什么的。

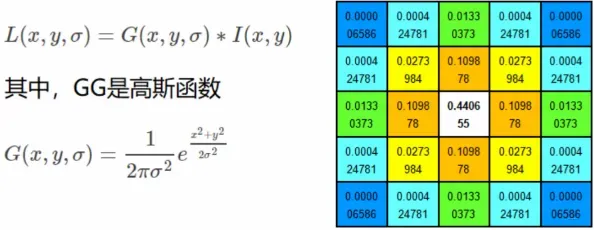
我明白了：找到图像中那些具有旋转、平移和尺度不变性的特征点，最后将它们表示为一个向量。

它究竟是如何完成的？大致内容如下。

### 3.1 图像的尺度空间

在一定范围内，无论物体大小，人眼都能分辨。但是，计算机很难拥有同样的能力。为了让机器对不同尺度的物体有统一的认知，需要考虑不同尺度图像中存在的特征。

尺度空间的获取通常使用高斯模糊来实现。



|  |
| --- |
|  |
|  |

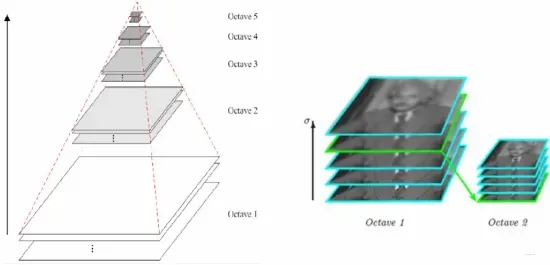
### 3.2 多分辨率金字塔

尺度空间使用高斯金字塔表示， 尺度规范化LoG(Laplacion of Gaussian)算子具有真正尺度不变性，Lowe使用高斯差分金字塔近似LoG算子，在尺度空间检测稳定关键点。

实现尺度空间时，用高斯金字塔表示。高斯金字塔的构建主要分为两部分：

1. 在图像上做不同尺度的高斯模糊
2. 对图像进行下采样以构建金字塔

高斯金字塔应该很熟悉：



|  |
| --- |
|  |

### 3.3 DoG空间极值点检测

在实际计算中，将高斯金字塔中每组图像的相邻上下层相减，得到高斯差分图像。

  
这个就是高斯金字塔中的每一层的图片，进行差分，这样得到的结果中，像素点相差较大的位置，就是不同尺度的图片之间的不同。DoG公式定义如下：

|  |
| --- |
|  |
|  |

当然这样产生的极值点并不全都是稳定的特征点，因为某些极值点响应较弱，而且DoG算子会产生较强的边缘响应。

值得一提的是，选取的高斯差分金字塔极值点只是候选特征点。虽然高斯差分金字塔的极值点已经能够更好地表示图像的特征并且具有尺度不变性，但是在选择过程中没有考虑图像特征点对图像噪声的鲁棒性，确定的图像特征点在这种方式在实际应用中使用。容易出现图像匹配不当等问题。

### 3.4 关键点的精确定位

以上方法检测到的极值点是离散空间的极值点，以下通过拟合三维二次函数来精确确定关键点的位置和尺度，同时去除低对比度的关键点和不稳定的边缘响应点(因为DoG算子会产生较强的边缘响应)，以增强匹配稳定性、提高抗噪声能力。

为了更好的理解一点，我们先来看看如何在一维的离散值拟合一条曲线：

|  |
| --- |
|  |
|  |

### 3.5. 消除边界响应

DoG算子会产生较强的边缘响应，即容易保留边界点，所以下面需要剔除不稳定的边缘响应点。具体做法如下：

|  |
| --- |
|  |
|  |

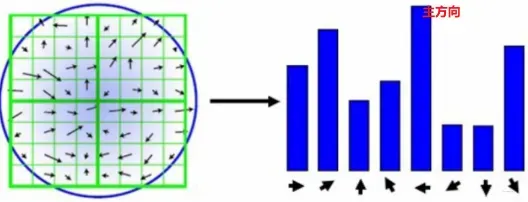
### 3.6 特征点的主方向

根据上述操作，可以找到一些较好的候选关键点，但是在构建高斯差分金字塔选择图像特征点时，算法考虑了关键点的尺度不变性。对于图像特征，旋转不变性与尺度不变性同等重要。

为了使描述旋转不变，需要利用图像的局部特征为每个关键点分配一个主方向。保持与旋转前相同的特征。

|  |
| --- |
|  |

下图直方图为简化版本（只有8个bin，实际操作时算法会统计从0到360°步长为10°的共计36个梯度方向的幅值和，共有36个bin）。梯度方向直方图中最高的bin对应的方向即定义为该关键点的主方向，若存在任一方向的幅值大于主方向幅值的80%，则将其作为辅方向。所以，对于同一个关键点，可能有多个方向，这种情况在相同位置和尺度将会有多个关键点被创建但方向不同。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。

  
直方图的横轴是方向，纵轴是邻域内每个点对应的方向梯度的累积和。

### 3.7 生成特征描述

在得到特征点的二维位置、尺度位置、主方向的具体信息后，算法需要解决的最后一个问题就是生成关键点信息的描述符，即用一个向量来描述图像中的特征点信息。它不随各种变化而变化，如光照变化、视角变化等。该描述符不仅包括关键点，还包括对其有贡献的关键点周围的像素，并且描述符应具有较高的唯一性以提高特征点正确匹配的概率。

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

你可以看到下面的代码。

### 3.8 OpenCV下的SIFT算法

对于SIFT算法， OpenCV中直接也是一个函数搞定。

img = cv2.imread('img/test\_1.jpg')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# opencv版本高于3.4.3， 这个sift算法使用变成cv2.SIFT\_create()， 在这之前的版本是cv2.xfeatures2d.SIFT\_create()

# SIFT检测器

sift = cv2.SIFT\_create()

# 找出图像中的关键点

kp = sift.detect(gray, None)

# 在图中画出关键点

img = cv2.drawKeypoints(gray, kp, img)

# 计算关键点对应的SIFT特征向量

kp, des = sift.compute(gray, kp)

# 上面的两步，也可以用下面的一步

sift = cv2.SIFT\_create()

kp1, des1 = sift.detectAndCompute(gray, None) # 这里还能一步到位，直接算出关键点以及关键向量

Python

看看效果：  


## 4. 特征匹配

得到图像中的关键点后，关键点的特征也可以用向量来描述，给出两张图像后，如何分辨出两张图像中哪些关键点相似？这是看关键点向量之间的差异，也是特征匹配在做的事情。

### 4.1 Brute-Force蛮力匹配

暴力匹配，两个图像中的关键点的特征向量，一个个的计算差异， 即两层for循环了。 这里直接看怎么用：

首先，读入两张图像，然后得到它们各自的关键点和关键点向量。

img1 = cv2.imread('img/box.png', 0)

img2 = cv2.imread('img/box\_in\_scene.png', 0)

# sift算法，得到每张图的关键点以及关键向量

sift = cv2.SIFT\_create()

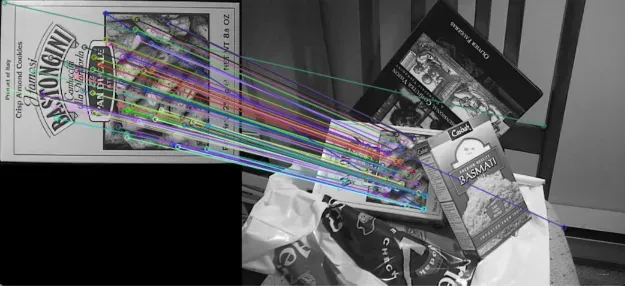
kp1, des1 = sift.detectAndCompute(img1, None) # 这里还能一步到位，直接算出关键点以及关键向量

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

Python

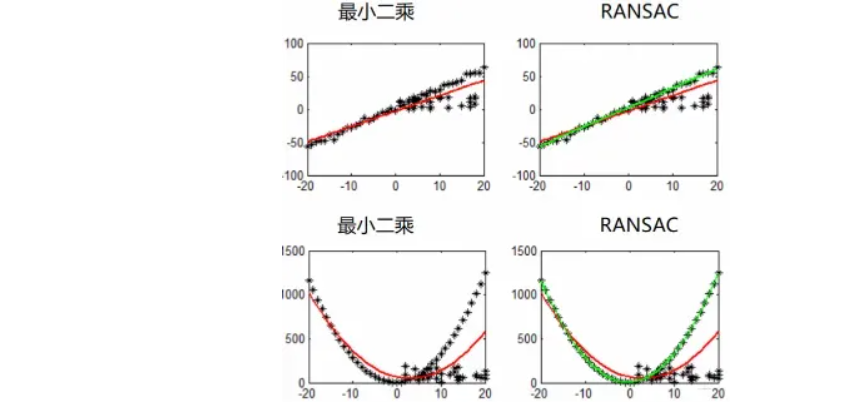
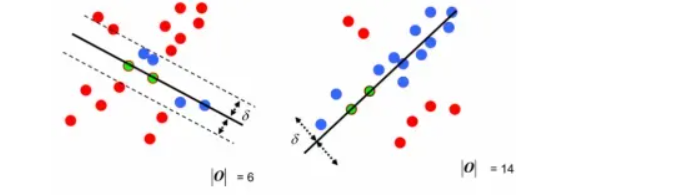
特征匹配执行如下：

1. 1对1的匹配： 即图片A中的一个向量匹配图片B中的一个向量# crossCheck表示两个特征点要互相匹配，例如A中的第i个特征点与B中的第j个特征点最近的，并且B中的第j个特征点到A中的第i个特征点也是最近的# NORM\_L2: 归一化数组的欧几里得距离， 如果其他特征计算方法需要考虑不同的匹配计算方式bf = cv2.BFMatcher(crossCheck=True)# 1对1的匹配matches = bf.match(des1, des2)matches = sorted(matches, key=lambda x: x.distance)  
   如果这里指定属性crossCheck为False， 得到的结果是604，如果为True，得到的结果是206，所以我基于这个，盲猜下暴力匹配以及这个属性的意义。  
   暴力匹配的话，就是对于A图像中的每个观测点的向量， 我遍历一遍B图像中每个观测点的向量，然后求欧几里得距离，拿到最小的。这样对于A图像中每个观测点，就得到了B图像里面的最佳匹配。  
   然后再对于B图像中的每个观测点，也同样用上面的方式走一遍，这样对于B图像中的每个观测点，也得到了A图像中的最佳匹配。  
   此时，如果是：
2. bf.match(des1, des2): 返回的就是A图像中的每个观测点的最佳匹配，个数是A中关键点的个数
3. bf.match(des2, des1): 返回的是B图像中每个观测点的最佳匹配， 个数是B中关键点的个数
4. 如果设置crossCheck为True的话：
5. bf.match(des1, des2)和 bf.match(des2, des1) 都是260， 表示的其实是A和B中，关键点互相匹配的那些，比如A中的第j个观测点，最近点是B中的i，那么B中的i，也要对应A中的j，即 VS 。 img3 = cv2.drawMatches(img1, kp1, img2, kp2, matches[:10], None, flags=2)  
   画10个最近的匹配看看：  
   
6. K对最佳匹配: 对于一个关键点， 找K个最相似的向量。bf = cv2.BFMatcher()matches = bf.knnMatch(des1, des2, k=2) # 相当于对于A中的每个关键点，不是找某一个最相似，而是K相似  
   这里可以根据近邻之间的相似比例对关键点筛选 good = []for m, n in matches: if m.distance < 0.75 \* n.distance: good.append([m])  
   也可以可视化一下： img3 = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good, None, flags=2)  
   结果如下：



这里会发现一个问题，对于上面的匹配结果，大部分结果还可以，但有某些匹配错误的点，这种情况怎么弥补呢？ 可以使用RANSAC(Random sample consensus)随机抽样一致算法。 这个东西可以简单看下原理。

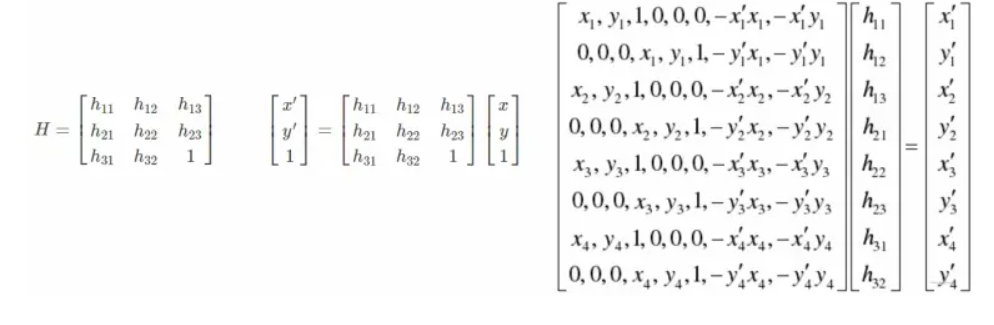
### 4.2 RANSAC(Random sample consensus)算法

该算法也是一种拟合算法，与最小二乘法对比如下：  
  
思路如下：首先选择初始样本点进行拟合，给出一个容差范围，继续迭代。  
  
每次拟合后，在公差范围内都有对应的数据点，找到数据点数最多的案例就是最终的拟合结果。因此，该算法在拟合时，先随机采样初始点，然后在拟合时考虑容差范围（可以拟合的数据点个数）。

那么，这东西干啥用呢？ 下面全景拼接的Demo会用到。

## 5. 全景拼接Demo

全景拼接想必大家都玩过。此功能在相机中可用。其原理大概如下。假设有两张图片要拼接，大概的流程如下：

1. 通过SIFT算法拿到两张图片的关键点以及关键向量
2. 基于关键向量的特征匹配
3. 根据特征匹配点，对图片进行一些仿射变换，比如平移、选择、缩放等，这里是为了保证可以无缝连接。如果不加工，则不得缝合。
4. 既然是仿射变换，本质就是将变换矩阵相乘，所以核心就是求解这个变换矩阵  
   
5. 如果需要做变换，目的就是求变换矩阵， 至少有4对匹配好的特征点才行。
6. 那么思路就是，先用随机采样算法采出四对匹配点，然后解8个方程得到上面的变换矩阵， 然后基于这个变换矩阵算其他的匹配点是否能匹配上，定义一个loss函数为匹配上的点的对数，这样就能基于当前的四对匹配点算出一个loss值。 然后再随机采样四对， 再求变换矩阵，再看看其他匹配点能匹配上的个数，得到损失。依次迭代多次， 取loss函数最小的那个变换矩阵。
7. 找到变换矩阵，先变换一张图片，再拼接。

这么说比较抽象，下面通过一个Demo把这个过程带起来。

### 5.1 读入图像，拿到SIFT特征

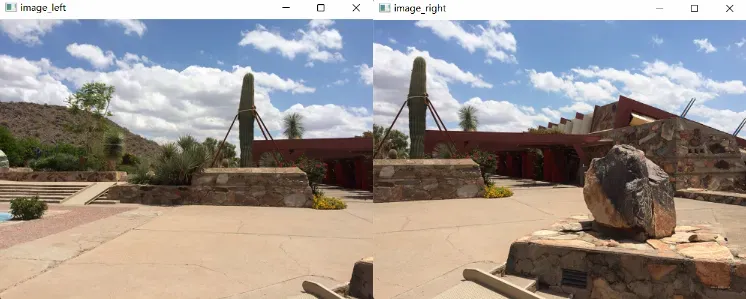
代码显示如下：

image\_left = cv2.imread('img/left\_01.png')

image\_right = cv2.imread('img/right\_01.png')

Python

两张图片如下：

  
显然，这种拼接是不能拼接的。

获取下面的关键点和特征向量

def detectAndDescribe(image):

# 将彩色图片转成灰度图

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# SIFT生成器

destriptor = cv2.SIFT\_create()

(kps, features) = destriptor.detectAndCompute(image, None)

# 结果转成numpy数组

kps = np.float32([kp.pt for kp in kps])

return (kps, features)

# 检测A， B图片的SIFT特征关键点，得到关键点的表示向量

(kps\_left, features\_left) = detectAndDescribe(image\_left) # kpsA (关键点个数， 坐标) features(关键点个数，向量)

(kps\_right, features\_right) = detectAndDescribe(image\_right)

### 5.2 特征匹配

这里还是要写一个函数：

def matchKeyPoints(kpsA, kpsB, featuresA, featuresB, ratio=0.75, reprojThresh=4.0):

# 建立暴力匹配器

matcher = cv2.BFMatcher()

# KNN检测来自两张图片的SIFT特征匹配对

rawMatches = matcher.knnMatch(featuresA, featuresB, 2)

matches = []

for m in rawMatches:

# 当最近距离跟次近距离的比值小于ratio时，保留此配对

# (<DMatch 000001B1D6B605F0>, <DMatch 000001B1D6B60950>) 表示对于featuresA中每个观测点，得到的最近的来自B中的两个关键点向量

if len(m) == 2 and m[0].distance < m[1].distance \* ratio:

# 存储两个点在featuresA, featuresB中的索引值

matches.append([m[0].trainIdx, m[0].queryIdx]) # 这里怎么感觉只用了m[0]也就是最近的那个向量啊，应该没用到次向量

# 这个m[0].trainIdx表示的时该向量在B中的索引位置， m[0].queryIdx表示的时A中的当前关键点的向量索引

# 当筛选后的匹配对大于4时，可以拿来计算视角变换矩阵

if len(matches) > 4:

# 获取匹配对的点坐标

ptsA = np.float32([kpsA[i] for (\_, i) in matches])

ptsB = np.float32([kpsB[i] for (i, \_) in matches])

# 计算视角变换矩阵 这里就是采样，然后解方程得到变换矩阵的过程

(H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC, reprojThresh)

return (matches, H, status)

# 匹配结果小于4时，返回None

return None

主要逻辑是从图片B中给图片A中的关键点拿最近的K个匹配向量，然后基于规则筛选， 保存好匹配好的关键点的两个索引值，通过索引值取到匹配点的坐标值，有了多于4对的坐标值，就能得到透视变换矩阵。 这里返回的主要就是那个变换矩阵。

# 匹配两张图片的所有特征点，返回匹配结果 注意，这里是变换right这张图像，所以应该是从left找与right中匹配的点，然后去计算right的变换矩阵

M = matchKeyPoints(kps\_right, kps\_left, features\_right, features\_left)

if not M:

# 提取匹配结果

(matches, H, status) = M

这里要一定要注意好，到底是对哪张图片做变换，比如给图片B做变换，那么就从A中找与B中特征点匹配的特征向量，求的是让图片B变换的透视矩阵。 这里是对right做变换，所以从left中给right的关键点找匹配点，给right计算透视矩阵，接下来，变换right

# 图片right进行视角变换， result是变换后的图片

result = cv2.warpPerspective(image\_right, H, (image\_left.shape[1] + image\_right.shape[1], image\_right.shape[0]))

cv\_imshow('result', result)

下面看一下改造后的效果：  
  
接下来就可以进行拼接了：

# 将图片A传入result图片最左端

result[0:image\_right.shape[0], 0:image\_right.shape[1]] = image\_left

结果如下：

  
这样就拼接在一起了，除了亮度不一样，效果还是不错的。

### 5.3 画出匹配图像

这里也可以在图像上绘制匹配向量：

def drawMatches(imageA, imageB, kpsA, kpsB, matches, status):

# 初始化可视化图片，将A、B图左右连接到一起

(hA, wA) = imageA.shape[:2]

(hB, wB) = imageB.shape[:2]

vis = np.zeros((max(hA, hB), wA + wB, 3), dtype="uint8")

vis[0:hA, 0:wA] = imageA

vis[0:hB, wA:] = imageB

# 联合遍历，画出匹配对

for ((trainIdx, queryIdx), s) in zip(matches, status):

# 当点对匹配成功时，画到可视化图上

if s == 1:

# 画出匹配对

ptA = (int(kpsA[queryIdx][0]), int(kpsA[queryIdx][1]))

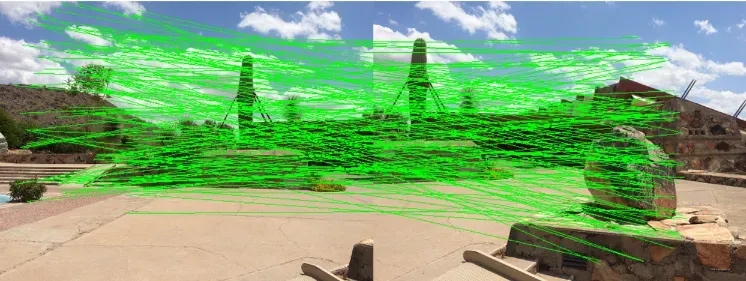
ptB = (int(kpsB[trainIdx][0]) + wA, int(kpsB[trainIdx][1]))

cv2.line(vis, ptA, ptB, (0, 255, 0), 1)

# 返回可视化结果

return vis

vis = drawMatches(image\_left, image\_right, kps\_left, kps\_right, matches, status)

结果如下：  


## 6. 小总

这篇文章主要是整理了在图像处理中重要且常用的找特征点的两个算法Harris和SIFT算法，包括算法的数学原理以及如何使用，然后是整理了下特征匹配与一致性采样算法，这俩东西其实为了后面的透视变换矩阵服务。 最后通过一个全景图拼接的Demo感受了下算法的魅力。

当然，这只是冰山下的一个小角落，因为后面我们会在一个项目中用到这个知识来识别停车场的车位。另外，这些方法还是通用的算法，所以我也想静下心来待会儿复习。

参考：

* OpenCV入门到实战课程
* 角点检测：Harris 与 Shi-Tomasi
* SIFT算法详解
* SIFT算法简述及Python标记SIFT特征检测实践