## 目标：

本章节你需要学习以下内容:

* 我们将了解多视角几何的基础知识
* 我们将看到什么是极点，极线，对极约束等。

**api：**

### sift = cv2.xfeatures2d.SIFT\_create()

### flann = cv2.FlannBasedMatcher(index\_params,search\_params)

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

### F, mask = cv.findFundamentalMat(pts1,pts2,cv.FM\_LMEDS)

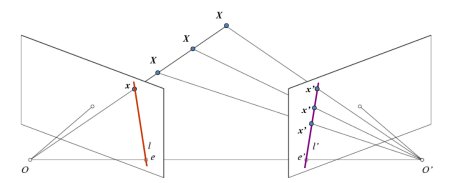
### lines1 = cv.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2,F)

**基础**

当我们使用针孔相机拍摄图像时，我们会丢失一些重要的信息，比如图像的深度。或者是图像中的每个点距离相机有多远，因为它是3D到2D的转换。因此一个重要的问题就产生了，使用这样的摄像机我们能否计算除深度信息呢？答案是我们需要使用多个摄像头。我们的眼睛以类似的方式工作，我们使用两个相机（两只眼睛）来判断物体的距离，称为立体视觉。那么让我们看看OpenCV在这个领域提供了什么。

（《学习 OpenCV》一书有大量相关知识）

在进入深度图像之前，让我们先了解多视图几何中的一些基本概念。在本节中，我们将讨论极线几何。请参见下图，其中显示了使用两个相机拍摄同一场景图像的基本设置。

[](https://camo.githubusercontent.com/f233b389783b238b0072c19678fa74ee5c5ddb0886f9e0dda1f166e637277272/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f657069706f6c61722e6a7067)

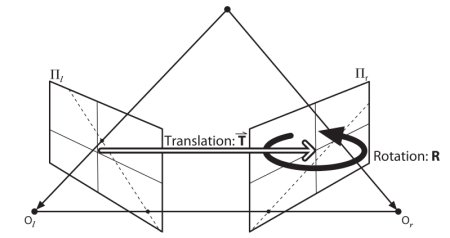
如果我们仅使用左侧相机，则无法找到与图像中的点x对应的3D点，因为线OX上的每个点都投影到图像平面上的相同点。但是如果我们也考虑上右侧图像的话，直线 OX 上的点将投影到右侧图像上的不同位置（x'）。所以根据这两幅图像，我们就可以使用三角测量计算出 3D 空间中的点到摄像机的距离（深度）。这就是整个思路。

OX上不同点的投影在右平面（线l'）上形成一条线。我们称它为对应于点x的极线。也就是说，要在右侧图像上找到点x，需要沿着此极线搜索。它应该在这个一维直线的某个地方（想象一下，要找到其他图像中的匹配点，你不需要搜索整个图像，只需沿着极线搜索。因此它提供了更好的性能和准确性）。这称为极线约束。类似地，所有点将在另一图像中具有其对应的极线。平面XOO'称为极线平面。

O和O'是摄像机中心。从上面给出的设置中，可以看到右侧摄像机O'的投影在该点的左侧图像上看到，例如。它被称为极点。极点是通过摄像机中心和图像平面的线的交叉点。类似地，e'是左相机的极点。在某些情况下，你将无法在图像中找到极点，它们可能位于图像之外（这意味着，一台摄像机看不到另一台摄像机）。

所有的极线都通过它的极点。所以为了找到极点的位置，我们可以先找到多条极线，这些极线的交点就是极点。

所以在本小节中，我们的重点就是寻找极线和极点。但要找到它们，我们需要另外两个元素，本征矩阵（F）和基本矩阵（E）。本征矩阵包含了物理空间中两个摄像机相关的旋转和平移信息。如下图所示（本图来源自：学习 OpenCV）：

[](https://camo.githubusercontent.com/39269fe2d87903a5ffdb70f080a01fcc45a16d4fddf7e019d6d00e998489fc15/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f657373656e7469616c5f6d61747269782e6a7067)

但我们更喜欢用像素坐标进行测量，对吧？基础矩阵 F 除了包含 E 的信息外还包含了两个摄像机的内参数。由于 F包含了这些内参数，因此它可以它在像素坐标系将两台摄像机关联起来。（如果使用是校正之后的图像并通过除以焦距进行了归一化，F=E）。简单来说，基础矩阵 F 将一副图像中的点映射到另一幅图像中的线（极线）上。这是通过匹配两幅图像上的点来实现的。要计算基础矩阵至少需要 8 个点（使用 8 点算法）。点越多越好，可以使用 RANSAC 算法得到更加稳定的结果。

**代码实现**

首先，我们需要在两个图像之间找到尽可能多的匹配，以找到基本矩阵。为此，我们使用SIFT描述符和基于FLANN的匹配器和比率测试。

import numpy as np

import cv2 as cv

from matplotlib import pyplot as plt

img1 = cv.imread('myleft.jpg',0) #queryimage # left image

img2 = cv.imread('myright.jpg',0) #trainimage # right image

sift = cv.SIFT()

# find the keypoints and descriptors with SIFT

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

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

# FLANN parameters

FLANN\_INDEX\_KDTREE = 1

index\_params = dict(algorithm = FLANN\_INDEX\_KDTREE, trees = 5)

search\_params = dict(checks=50)

flann = cv.FlannBasedMatcher(index\_params,search\_params)

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

good = []

pts1 = []

pts2 = []

# ratio test as per Lowe's paper

for i,(m,n) in enumerate(matches):

if m.distance < 0.8\*n.distance:

good.append(m)

pts2.append(kp2[m.trainIdx].pt)

pts1.append(kp1[m.queryIdx].pt)

现在我们有两张图片的最佳匹配列表。让我们找到基本矩阵。

pts1 = np.int32(pts1)

pts2 = np.int32(pts2)

F, mask = cv.findFundamentalMat(pts1,pts2,cv.FM\_LMEDS)

# We select only inlier points

pts1 = pts1[mask.ravel()==1]

pts2 = pts2[mask.ravel()==1]

下一步我们要找到极线。我们会得到一个包含很多线的数组。所以我们要定义一个新的函数将这些线绘制到图像中。

def drawlines(img1,img2,lines,pts1,pts2):

''' img1 - image on which we draw the epilines for the points in img2

lines - corresponding epilines '''

r,c = img1.shape

img1 = cv.cvtColor(img1,cv.COLOR\_GRAY2BGR)

img2 = cv.cvtColor(img2,cv.COLOR\_GRAY2BGR)

for r,pt1,pt2 in zip(lines,pts1,pts2):

color = tuple(np.random.randint(0,255,3).tolist())

x0,y0 = map(int, [0, -r[2]/r[1] ])

x1,y1 = map(int, [c, -(r[2]+r[0]\*c)/r[1] ])

img1 = cv.line(img1, (x0,y0), (x1,y1), color,1)

img1 = cv.circle(img1,tuple(pt1),5,color,-1)

img2 = cv.circle(img2,tuple(pt2),5,color,-1)

return img1,img2

现在我们在两个图像中找到了极线并绘制它们。

# Find epilines corresponding to points in right image (second image) and

# drawing its lines on left image

lines1 = cv.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2,F)

lines1 = lines1.reshape(-1,3)

img5,img6 = drawlines(img1,img2,lines1,pts1,pts2)

# Find epilines corresponding to points in left image (first image) and

# drawing its lines on right image

lines2 = cv.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1,F)

lines2 = lines2.reshape(-1,3)

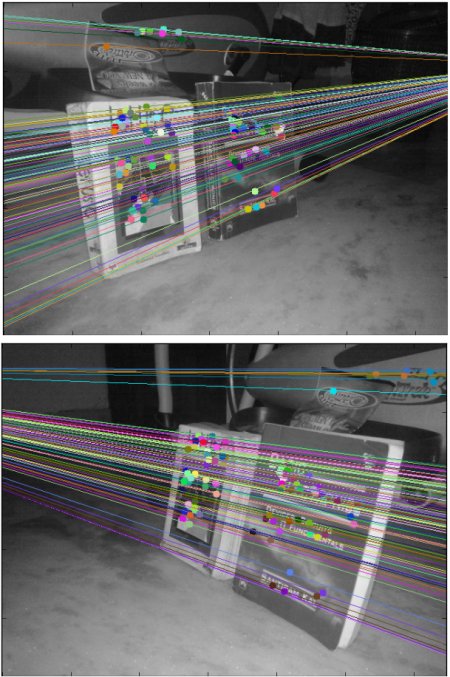
img3,img4 = drawlines(img2,img1,lines2,pts2,pts1)

plt.subplot(121),plt.imshow(img5)

plt.subplot(122),plt.imshow(img3)

plt.show()

结果如下图所示；

[](https://camo.githubusercontent.com/3b896590f266ecc5eaa14efee3df1c80961a80635ba23a5f18a1fc5532e3bfa8/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f657069726573756c742e6a7067)

你可以在左侧图像中看到所有的极线都会聚合在右侧图像外部的一个点上，这个点就是极点。

为了得到更好的结果，我们应该使用分辨率比较高和很多非平面点的图像。

## 完整实例代码：epipolar-geo-demo.py

|  |  |
| --- | --- |
|  | **import cv2 import numpy as np from matplotlib import pyplot as plt  *# 定义一个函数将找到的极线绘制到图像中* def drawlines(img1,img2,lines,pts1,pts2):  *''' img1 - image on which we draw the epilines for the points in img2  lines - corresponding epilines '''* r,c = img1.shape  img1 = cv2.cvtColor(img1,cv2.COLOR\_GRAY2BGR)  img2 = cv2.cvtColor(img2,cv2.COLOR\_GRAY2BGR)  for r,pt1,pt2 in zip(lines,pts1,pts2):  color = tuple(np.random.randint(0,255,3).tolist())  x0,y0 = map(int, [0, -r[2]/r[1] ])  x1,y1 = map(int, [c, -(r[2]+r[0]\*c)/r[1] ])  img1 = cv2.line(img1, (x0,y0), (x1,y1), color,1)  img1 = cv2.circle(img1,tuple(pt1),5,color,-1)  img2 = cv2.circle(img2,tuple(pt2),5,color,-1)  return img1,img2  img1 = cv2.imread('./left.jpg',0) *# queryimage # left image,查询图片，灰度的* img2 = cv2.imread('./right.jpg',0) *#trainimage # right image，训练图片，灰度的  # 创建sift对象* sift = cv2.xfeatures2d.SIFT\_create() *# find the keypoints and descriptors with SIFT，用sift算法寻找关键点* kp1,des1 = sift.detectAndCompute(img1,None) kp2,des2 = sift.detectAndCompute(img2,None) *# 设置FLANN 参数* FLANN\_INDEX\_KDTREE = 1 index\_params = dict(algorithm=FLANN\_INDEX\_KDTREE,trees=5) search\_params = dict(checks=50) *#创建flann对象* flann = cv2.FlannBasedMatcher(index\_params,search\_params) *# 特征匹配* matches = flann.knnMatch(des1,des2,k=2) good=[] pts1=[] pts2=[] *#遍历匹配结果，找到好的特征* for i,(m,n) in enumerate(matches):  if m.distance < 0.8\*n.distance:  good.append(m)  pts1.append(kp1[m.queryIdx].pt) *#保存查询点* pts2.append(kp2[m.trainIdx].pt) *#保存训练点 #找到基本矩阵* pts1=np.int32(pts1) pts2=np.int32(pts2) F,mask = cv2.findFundamentalMat(pts1,pts2,cv2.FM\_LMEDS) *# 我们只选择内点群inlier的点* pts1 = pts1[mask.ravel()==1] pts2 = pts2[mask.ravel()==1]  *# 要找到极线。我们会得到一个包含很多线的数组。所以我们要调用上面的函数将这些线绘制到图像中。 # Find epilines corresponding to points in right image (second image) and # drawing its lines on left image 先做右边的* lines1 = cv2.computeCorrespondEpilines(pts2.reshape(-1,1,2),2,F) lines1 = lines1.reshape(-1,3) img5,img6 = drawlines(img1,img2,lines1,pts1,pts2)  *#再做左边的* lines2 = cv2.computeCorrespondEpilines(pts1.reshape(-1,1,2),1,F) lines2 = lines2.reshape(-1,3) img3,img4 = drawlines(img2,img1,lines2,pts2,pts1) plt.subplot(121),plt.imshow(img5) plt.subplot(122),plt.imshow(img3) plt.show() *# cv2.imshow("pic", img)* cv2.waitKey(0) cv2.destroyAllWindows()** |

### 效果

|  |
| --- |
|  |