相机标定opencv python代码

这里只讲使用opencv python api怎么实现相机标定，不涉及理论部分。

相机标定步骤

准备工作

标定至少需要10张不同角度拍摄的测试模板图片，当然可以自己来拍摄，我们这里使用OpenCV提供的棋盘格图片（samples/cpp/left01.jpg —— left14.jpg），可以在下面链接中找到：

链接：https://pan.baidu.com/s/1wXm50Us1DiaroXj3ZGRYtQ
提取码：ogpw

14张棋盘格图片如下所示：

导入必要的工具库

```
import numpy as np
import cv2
import glob
```

获取需要的数据

相机标定需要的重要数据包括：一系列3D物体坐标点和它相对的2D图像坐标点。

2D图片上的点我们可以通过角点检测知道它们的位置。（2D图像坐标点就是棋盘图中两个黑色块相接触的地方）

那么3D物体坐标点怎么得到呢?

图像是固定摄像机拍摄不同位置和方向的棋盘格得到的，为了简便处理，我们假设图片就在XY平面上，那么Z就是0，棋盘图就可以等效为相机自己移动到不同位置和方向拍摄的。

这种简化后，就可以简单用(0,0), (1,0), (2,0), ... 来代表3D物体坐标点的位置，这样我们得到的结果就是与棋盘格等比例的大小，如果我们知道格子的尺寸（比如30mm），我们只需要输入(0,0), (30,0), (60,0), ... 即可。

> 由于我们不知道棋盘格方格的尺寸（不是自己拍摄的，没法测量），所以我们用(0,0), (1,0), (2,0), ... 作为输入来演示。

以下，默认3D物体坐标点称作物体点，2D图像坐标点称作图像点。

- **设置物体点**

```
# 设置终止条件，迭代30次或变动小于0.001
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# 生成42×3的矩阵，用来保存棋盘图中6*7个内角点的3D坐标，也就是物体点坐标
objp = np.zeros((6*7, 3), np.float32)

# 通过np.mgrid生成对象的xy坐标点
# 最终得到的objp为(0,0,0), (1,0,0), (2,0,0) ,..., (6,5,0)
objp[:, :2] = np.mgrid[0:7, 0:6].T.reshape(-1, 2)
```

- **获取图片点**

图片点需要用到角点检测，这里使用**findChessboardCorners**函数先得到棋盘图内角点的近似坐标（因为这个函数的精度不高），然后将近似坐标作为初始值输入cornerSubPix函数，进行亚像素级角点精确检测。

```
obj_points = []  # 用于保存物体点
img_points = []  # 用于保存图像点

# 返回当前目录所有匹配的jpg图片
images = glob.glob('*.jpg')

for fname in images:
    # 读取图片
    img = cv2.imread(fname)
    # 转为灰度图
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # 寻找棋盘图的内角点位置
    ret, corners = cv2.findChessboardCorners(gray, (7,6), None)

    # 如果找到棋盘图的所有内角点
    if ret == True:
        obj_points.append(objp)
        # 亚像素级角点检测，在角点检测中精确化角点位置
        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
        img_points.append(corners2)

        # 在图中标注角点，方便查看结果
        img = cv2.drawChessboardCorners(img, (7,6), corners2, ret)
        cv2.imshow('img', img)
        cv2.waitKey(500)

cv2.destroyAllWindows()
```

执行相机标定程序

下面使用**cv2.calibrateCamera()**进行相机标定，它返回相机矩阵、畸变系数、旋转和平移向量等。

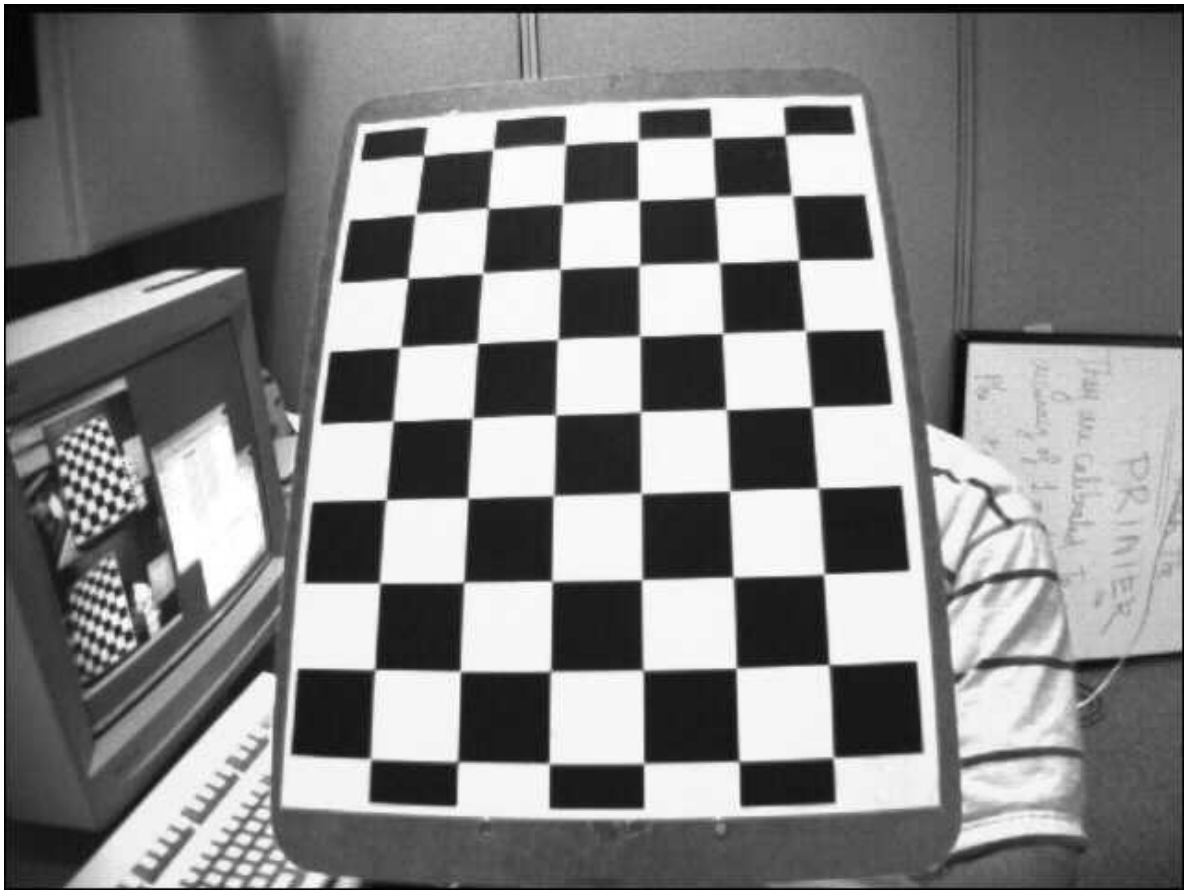
```
# 相机标定
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points,
gray.shape, None, None)
```

消除畸变

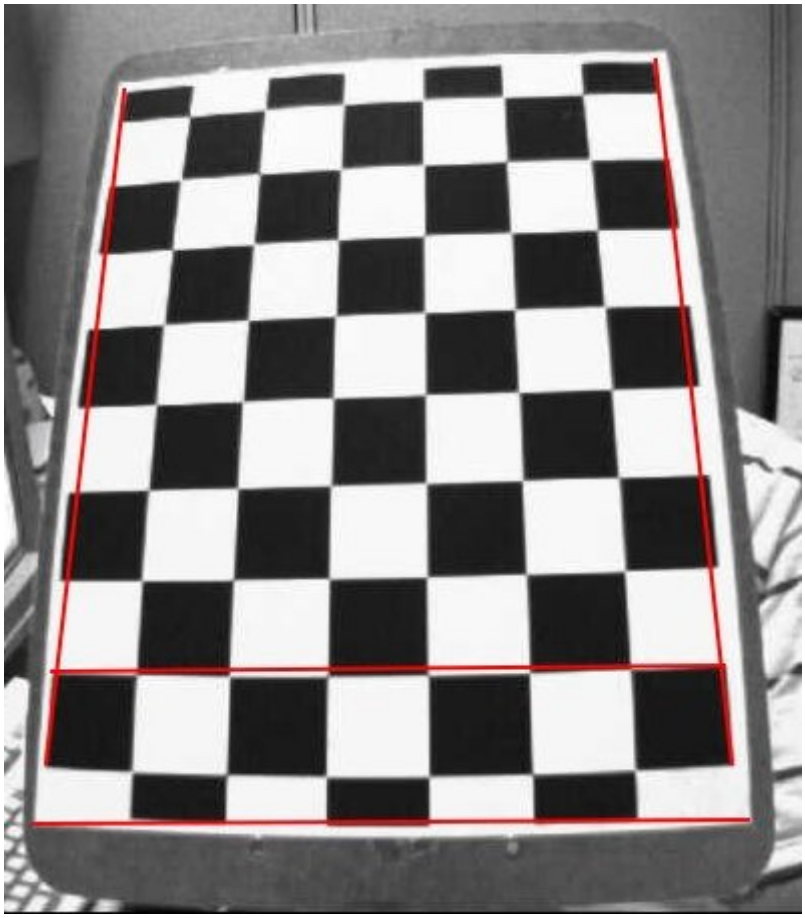
我们已经得到了所有的估计参数，现在可以拿一张图片来看看畸变消除的效果。

```
# 读入图片
img = cv2.imread('left12.jpg')
# 获取图片的长宽
h, w = img.shape[:2]
```

下图就是left12.jpg：



很明显有畸变想象（棋盘图的边缘线不与红色直线重合）。



在这之前，我们需要使用**getOptimalNewCameraMatrix**来重新生成相机矩阵，从而减少原图的有效像素的丢失。

> 为什么要使用getOptimalNewCamera具体可以查看下面的博客：
>
> MatrixOpenCV畸变校正原理以及损失有效像素原理分析
>
> https://www.cnblogs.com/riddick/p/6711263.html

它有一个参数alpha，叫做尺度因子，取值0~1。

如果alpha=0，原图像会损失最多的有效像素；如果alpha=1，原图像中的所有像素都能够得到保留。

getOptimalNewCameraMatrix还返回一个图像ROI，可以用来裁剪结果。

我们选取一张新图片（如left12.jpg）

```
# 根据尺度因子调节相机矩阵
newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))
```

在OpenCV有两种消除畸变方法：

1. 调用undistort函数

这是最简单的办法，通过调用**undistort**，传递ROI参数就可以裁剪图片得到结果。

```
# 校正畸变图片
dst = cv2.undistort(img, mtx, dist, None, newcameramtx)

# 裁剪图片
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv2.imwrite('calibresult.png', dst)
```

2. 调用remap函数

这种方法麻烦一点。首先找到原图片与校正图片之间的映射关系，然后使用重映射函数**remap**，最后依据得到的**roi**裁剪图片即可。

```
# 校正畸变图片
mapx, mapy = cv2.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (w,h),
5)
dst = cv2.remap(img, mapx, mapy, cv2.INTER_LINEAR)

# 裁剪图片
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv2.imwrite('calibresult.png', dst)
```

两种方法都可以得到同样的结果，如下图：

你会发现结果中的所有边都是直的，因此达到了消除畸变效果。

重投影误差

重投影误差是一个判别畸变参数准确度的参考指标，它越接近于0越好。

给定畸变矩阵，旋转矩阵和平移矩阵，首先将物体点坐标变换到图像点坐标，可以使用**projectPoints**函数实现。

然后计算变换后得到的图像点和我们之前检测到的角点坐标的l2范数平均值（即加和开方求平均）。

```
# 计算重投影误差
mean_error = 0
for i in range(len(obj_points)):
    img_points2, _ = cv2.projectPoints(obj_points[i], rvecs[i], tvecs[i], mtx,
dist)
    error = cv2.norm(img_points[i], img_points2, cv2.NORM_L2)/len(img_points2)
    mean_error += error

print("total error: ", tot_error/len(obj_points))
```

注意事项

棋盘图像数目应该取多少对摄像头定标比较适宜?

建议搞个10来幅左右。

单目定标函数cvCalibrateCamera2采用怎样的 flags 比较合适?

一般镜头只需要计算k1,k2,p1,p2四个参数，我们可以设置为CV_CALIB_FIX_K3；

如果所用的摄像头不是高端的、切向畸变系数非常小的，则不要设置CV_CALIB_ZERO_TANGENT_DIST，否则单目校正误差会很大；

如果事先知道摄像头内参的大概数值，并且cvCalibrateCamera2函数的第五个参数intrinsic_matrix非空，则也可设置CV_CALIB_USE_INTRINSIC_GUESS ，以输入的intrinsic_matrix为初始估计值来加快内参的计算；

其它的**flags**一般都不用设置，对单目定标的影响不大。

参考链接：

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html

代码中用到的API说明

glob模块

glob模块可以查找符合特定规则的文件路径名。

查找文件只用到三个匹配符："*"，"?"，"[]"。

- "*"：表示匹配0个或多个字符；
- "?"：匹配单个字符；
- "[]"：匹配指定范围内的字符，如：[0-9]匹配数字。

files = glob.glob(pathname)

功能：按照文件路径匹配规则，找到所有匹配的文件。文件路径匹配规则，可以是绝对路径，也可以是相对路径。

参数：

- **pathname**：文件路径名

返回：

- **files**：所有匹配的文件名，以列表形式输出

例子：

```
import glob

#获取/home/username目录下的所有图片
print(glob.glob(r"/home/username/*/*.png"),"\n")  # 加上r让字符串不转义

#获取上级目录的所有.py文件
print(glob.glob(r'../*.py'))  # 相对路径
```

看opencv-python的api要注意的地方

- 参数在[]中表示可选参数，也就是调用时可以不输入，采用默认。

  如`cv.findChessboardCorners(image, patternSize[, corners[, flags]])`

  `corners`和`flags`都是可选参数。

- opencv的python版本个人猜测是直接把C++的代码给封装了，所以有些输入参数一般不设置，它的存在只是为了符合C++版本的输入参数格式。

  如下面的findChessboardCorners：

  C++版本

  bool cv::findChessboardCorners(InputArray image, Size patternSize, OutputArray corners, int flags=CALIB_CB_ADAPTIVE_THRESH+CALIB_CB_NORMALIZE_IMAGE)

  python版本

  retval, corners = cv.findChessboardCorners(image, patternSize[, corners[, flags]])

  输入的`corners`一般不设置或给个None。

---

mgrid

返回多维结构

```
import numpy as np
x = np.array([1,2,3])
y = np.array([4,5])
x1, y1 = np.mgrid[1:3:3j, 4:5:2j]
```

得到的x1为

```
array([[1., 1.],
       [2., 2.],
       [3., 3.]])
```

得到的x2为

```
array([[4., 5.],
       [4., 5.],
       [4., 5.]])
```

---

findChessboardCorners

retval, corners = cv.findChessboardCorners(image, patternSize[, corners[, flags]])

功能：寻找棋盘图的内角点位置。

参数：

- **image**：输入图片，可以是灰度图或者彩色图
- **patternSize**：棋盘图的内角点行列数
- **corners**：不用设置
- **flags**：操作标志符，可以是0或者下面值的组合，对于opencv3.4.2，默认是 `CALIB_CB_ADAPTIVE_THRESH+CALIB_CB_NORMALIZE_IMAGE`
  - **CALIB_CB_ADAPTIVE_THRESH**：使用自适应阈值（通过平均图像亮度计算得到）将图像转换为黑白图，而不是一个固定的阈值
  - **CALIB_CB_NORMALIZE_IMAGE**：在利用固定阈值或者自适应的阈值进行二值化之前，先使用equalizeHist 来均衡化图像亮度
  - **CALIB_CB_FILTER_QUADS**：使用其他的准则（如轮廓面积，周长，方形形状）来去除在轮廓检测阶段检测到的错误方块
  - **CALIB_CB_FAST_CHECK**：在寻找内角点之前，先快速查看是否有棋盘图，若无则报错

返回：

- **retval**：棋盘图中的所有内角点是否都检测到
- **corners**：检测到的内角点位置

`findChessboardCorners`得到的内角点位置是近似的，后面还需要`cornerSubPix`来得到更为精确的位置。

---

cornerSubPix

corners = cv.cornerSubPix(image, corners, winSize, zeroZone, criteria)

功能：亚像素级角点检测，更加精确化角点位置

参数：

- **image**：输入图片
- **corners**：输入的内角点位置，作为初始值
- **winSize**：搜索窗大小的一半（算法要根据搜索窗内的梯度来精确定位内焦点）。如winSize=(5,5)，搜索窗大小为(5×2+1,5×2+1)=(11,11)，搜索窗大小一般为单数，所以要加1
- **zeroZone**：设置“零区域”，在搜索窗口内，“零区域”内的值不会被累加，权重值为0。如果设置为(-1,-1)，则表示没有这样的区域
- **criteria**：算法终止条件

返回：

- **corners**：输出的内角点位置

---

drawChessboardCorners

image = cv.drawChessboardCorners(image, patternSize, corners, patternWasFound)

功能：在图中标注角点

参数：

- **image**：输入图片
- **patternSize**：内角点的每行和每列的个数
- **corners**：findChessboardCorners检测到的角点
- **patternWasFound**：棋盘图中的所有内角点是否都检测到，可以输入findChessboardCorners返回的retval

返回：

- **corners**：标注了内角点的图片

---

calibrateCamera

retval, cameraMatrix, distCoeffs, rvecs, tvecs = cv.calibrateCamera(objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs[, rvecs[, tvecs[, flags[, criteria]]]])

功能：相机内外参数估计

参数：

- **objectPoints**：物体点坐标
- **imagePoints**：图像点坐标
- **imageSize**：图像大小，用于初始化相机矩阵
- **cameraMatrix**：相机矩阵初始化，flags设置了CV_CALIB_USE_INTRINSIC_GUESS或CALIB_FIX_ASPECT_RATIO才需要
- **distCoeffs**：不用设置
- **rvecs**：不用设置
- **tvecs**：不用设置
- **flags**：操作标志符，可输入为0或以下参数的组合，默认为0，即不使用以下模式。
    - **CV_CALIB_USE_INTRINSIC_GUESS**：使用该参数时，将包含有效的fx,fy,cx,cy的估计值的内参矩阵cameraMatrix，作为初始值输入，然后函数对其做进一步优化。如果不使用该参数，用图像的中心点初始化光轴点坐标(cx, cy)，使用最小二乘估算出fx,fy（这种求法好像和张正友的论文不一样，不知道为何要这样处理）。注意，如果已知内部参数（内参矩阵和畸变系数），就不需要使用这个函数来估计外参，可以使用solvepnp()函数计算外参数矩阵。
    - **CV_CALIB_FIX_PRINCIPAL_POINT**：在进行优化时会固定光轴点，光轴点将保持为图像的中心点。当CV_CALIB_USE_INTRINSIC_GUESS参数被设置，保持为输入的值。
    - **CV_CALIB_FIX_ASPECT_RATIO**：固定fx/fy的比值，只将fy作为可变量，进行优化计算。当CV_CALIB_USE_INTRINSIC_GUESS没有被设置，fx和fy的实际输入值将会被忽略，只有fx/fy的比值被计算和使用。
    - **CV_CALIB_ZERO_TANGENT_DIST**：切向畸变系数(P1, P2)被设置为零并保持为零。
    - **CV_CALIB_FIX_K1,...,CV_CALIB_FIX_K6**：对应的径向畸变系数在优化中保持不变。如果设置了CV_CALIB_USE_INTRINSIC_GUESS参数，就从提供的畸变系数矩阵中得到。否则，设置为0。
    - **CV_CALIB_RATIONAL_MODEL**（理想模型）：启用畸变k4, k5, k6三个畸变参数。使标定函数使用有理模型，返回8个系数。如果没有设置，则只计算其它5个畸变参数。
    - **CALIB_THIN_PRISM_MODEL**（薄棱镜畸变模型）：启用畸变系数S1、S2、S3和S4。使标定函数使用薄棱柱模型并返回12个系数。如果不设置标志，则函数计算并返回只有5个失真系数。
    - **CALIB_FIX_S1_S2_S3_S4**：优化过程中不改变薄棱镜畸变系数S1、S2、S3、S4。如果cv_calib_use_intrinsic_guess设置，使用提供的畸变系数矩阵中的值。否则，设置为0。

    - **CALIB_TILTED_MODEL**（倾斜模型）：启用畸变系数tauX and tauY。标定函数使用倾斜传感器模型并返回14个系数。如果不设置标志，则函数计算并返回只有5个失真系数。
    - **CALIB_FIX_TAUX_TAUY**：在优化过程中，倾斜传感器模型的系数不被改变。如果cv_calib_use_intrinsic_guess设置，从提供的畸变系数矩阵中得到。否则，设置为0。
- **criteria**：算法终止条件

返回：

- **retval**：是否正常得到内外参数结果
- **cameraMatrix**：相机矩阵
- **distCoeffs**：畸变矩阵
- **rvecs**：旋转向量
- **tvecs**：位移向量

---

getOptimalNewCameraMatrix

retval, validPixROI = cv.getOptimalNewCameraMatrix(cameraMatrix, distCoeffs, imageSize, alpha[, newImgSize[, centerPrincipalPoint]])

功能：根据尺度因子调节相机矩阵

参数：

- **cameraMatrix**：输入的相机矩阵
- **distCoeffs**：输入的畸变矩阵
- **imageSize**：原始的图片尺寸
- **alpha**：尺度因子，大小在0~1之间
- **newImgSize**：修正后的图片尺寸，默认与imageSize一样
- **centerPrincipalPoint**：可选的操作符，用于确定输出相机矩阵的主要点是否在图像的中心，默认False

返回：

- **retval**：输出的相机矩阵
- **validPixROI**：输出方框的对角坐标，该方框的范围表示原图中没有畸变的像素范围

---

undistort

dst = cv.undistort(src, cameraMatrix, distCoeffs[, dst[, newCameraMatrix]])

功能：校正畸变图片

参数：

- **src**：输入的畸变图片
- **cameraMatrix**：输入的相机矩阵
- **distCoeffs**：输入的畸变矩阵
- **dst**：不用设置
- **newCameraMatrix**：畸变图片的相机矩阵，默认和cameraMatrix一样，但可以通过这个参数对cameraMatrix做一些缩放平移

返回：

- **dst**：输出的校正图片

---

initUndistortRectifyMap

map1, map2 = cv.initUndistortRectifyMap(cameraMatrix, distCoeffs, R, newCameraMatrix, size, m1type[, map1[, map2]])

功能：计算无畸变和修正转换映射

参数：

- **cameraMatrix**：输入的相机矩阵
- **distCoeffs**：输入的畸变矩阵
- **R**：可选的修正变换矩阵，是个3×3的矩阵
- **newCameraMatrix**：新的相机矩阵
- **size**：畸变校正后的图片尺寸
- **m1type**：map1的类型，可以选择 CV_32FC1, CV_32FC2 或 CV_16SC2

> CV_32FC1, CV_32FC2, CV_16SC2都是矩阵数据类型。
>
> 矩阵数据类型的通用模板如下：
>
> CV_<bit_depth>(S|U|F)C<number_of_channels>
>
> bit_depth = 存储位数
>
> S = 符号整型 U = 无符号整型 F = 浮点型
>
> number_of_channels = 通道数
>
> 例子：CV_8UC3 是指一个8位无符号整型3通道矩阵，可以用来表示RGB彩色图中的一个像素点。

返回：

- **map1**：第一个输出映射
- **map2**：第二个输出映射

---

projectPoints

imagePoints, jacobian = cv.projectPoints(objectPoints, rvec, tvec, cameraMatrix, distCoeffs[, imagePoints[, jacobian[, aspectRatio]]])

功能：根据所给的3D坐标和已知的几何变换来求解投影后的2D坐标

参数：

- **objectPoints**：目标的3D坐标
- **rvec**：旋转向量
- **tvec**：位移向量
- **cameraMatrix**：相机参数矩阵
- **distCoeffs**：畸变矩阵
- **imagePoints**：不设置
- **jacobian**：不设置
- **aspectRatio**：(fx/fy)固定比率

返回：

- **imagePoints**：图片的2D坐标
- **jacobian**：雅可比矩阵，反映图片点的梯度

---

norm

retval = cv.norm(src1, src2[, normType[, mask]])

功能：计算范数

参数：

- **src1**：输入变量1
- **src2**：输入变量2
- **normType**：

  ∞范数：cv.NORM_INF

  L1范数：cv.NORM_L1

  L2范数（最后的结果要做开方）：cv.NORM_L2

  L2范数（最后的结果不做开方）：cv.NORM_L2SQR

返回：

- **retval**：(src1-src2) 的范数计算结果

原理部分较好的资料：

相机标定详细讲解：https://www.jianshu.com/p/7d97fccd79bb

[图像]摄像机标定(2) 张正友标定推导详解：https://blog.csdn.net/humanking7/article/details/44756235