**目标：**

本章节你需要学习以下内容:

* 我们将了解相机的畸变，相机的内部参数和外部参数等。
* 我们将学习如何找到这些参数，和对畸变图像进行修复等。

## api：

### ret, corners = cv2.findChessboardCorners(gray, (7,6), None)

### corners2 = cv2.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)

### cv2.drawChessboardCorners(img, (7,6), corners2, ret)

### ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)

### newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))

### dst = cv2.undistort(img, mtx, dist, None, newcameramtx)

### # undistort

### mapx, mapy = cv.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (w,h), 5)

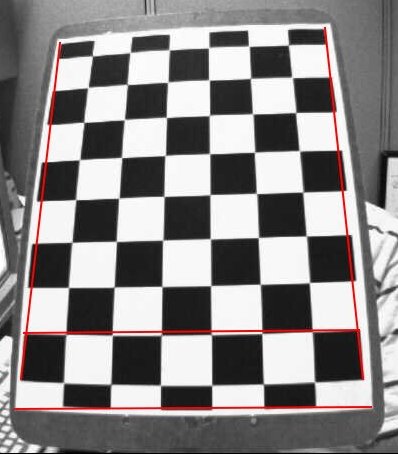
### cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)

### newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))

**基础**

如今市面上便宜的针孔摄像头会给图像带来了很多畸变。两种主要的畸变有两：径向畸变和切向畸变。

由于径向变形，直线会出现弯曲。当我们远离图像中心时，它的效果就会更加明显了。例如，下面显示了一个图像，其中棋盘的两个边缘用红线标记。但是你可以看到图像中边框不是直线而且与红线不重合。所有应该是直线的地方都变成突出的曲线。访问Distortion（光学）了解更多详情。

[](https://camo.githubusercontent.com/7052f8aff6e6d7bb949b36101236aa90376a00d7b342b46e44185413840f5c9d/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f63616c69625f72616469616c2e6a7067)

这种畸变可以通过下面的方程组进行纠正：

𝑥𝑑𝑖𝑠𝑡𝑜𝑟𝑡𝑒𝑑=𝑥(1+𝑘1𝑟2+𝑘2𝑟4+𝑘3𝑟6)

𝑦𝑑𝑖𝑠𝑡𝑜𝑟𝑡𝑒𝑑=𝑦(1+𝑘1𝑟2+𝑘2𝑟4+𝑘3𝑟6)

类似地，另一种失真是发生切向失真，因为图像拍摄镜头未完全平行于成像平面对齐。因此，图像中的某些区域可能看起来比预期的更近。它表示如下：

𝑥𝑑𝑖𝑠𝑡𝑜𝑟𝑡𝑒𝑑=𝑥+[2𝑝1𝑥𝑦+𝑝2(𝑟2+2𝑥2)]

𝑦𝑑𝑖𝑠𝑡𝑜𝑟𝑡𝑒𝑑=𝑦+[𝑝1(𝑟2+2𝑦2)+2𝑝2𝑥𝑦]

简而言之，我们需要找到五个参数，称为失真系数，由下式给出：

𝐷𝑖𝑠𝑡𝑜𝑟𝑡𝑖𝑜𝑛;𝑐𝑜𝑒𝑓𝑓𝑖𝑐𝑖𝑒𝑛𝑡𝑠=(𝑘1𝑘2𝑝1𝑝2𝑘3)

除此之外，我们还需要找到更多信息，比如摄像机的内部参数和外部参数。内部参数特定于相机。它包括焦距(𝑓𝑥,𝑓𝑦)，光学中心(𝑐𝑥,𝑐𝑦)等信息。它也被称为相机矩阵，仅取决于相机，因此可以只计算一次，然后存储以备将来使用。它表示为3x3矩阵：

𝑐𝑎𝑚𝑒𝑟𝑎;𝑚𝑎𝑡𝑟𝑖𝑥=[𝑓𝑥0𝑐𝑥 0𝑓𝑦𝑐𝑦 001]

外部参数对应于旋转向量和平移向量，它可以将3D点的坐标转换到坐标系中。

在 3D 相关应用中，必须要先校正这些畸变。为了找到这些参数，我们必须要提供一些包含明显图案模式的样本图片（比如说棋盘）。我们可以在上面找到一些特殊点（如棋盘的四个角点）。我们起到这些特殊点在图片中的位置以及它们的真是位置。有了这些信息，我们就可以使用数学方法求解畸变系数。这就是整个故事的摘要了。为了得到更好的结果，我们至少需要 10 个这样的图案模式。

**代码实现**

如上所述，我们至少需要 10 图案模式来进行摄像机标定。OpenCV 自带了一些棋盘图像（/sample/cpp/left001.jpg--left14.jpg）, 所以我们可以使用它们。为了便于理解，我们可以认为仅有一张棋盘图像。重要的是在进行摄像机标定时我们要输入一组 3D 真实世界中的点以及与它们对应 2D 图像中的点。2D 图像的点可以在图像中很容易的找到。（这些点在图像中的位置是棋盘上两个黑色方块相互接触的地方）

那么真实世界中的 3D 的点呢？这些图像来源与静态摄像机和棋盘不同的摆放位置和朝向。所以我们需要知道（X，Y，Z）的值。但是为了简单，我们可以说棋盘在 XY 平面是静止的，（所以 Z 总是等于 0）摄像机在围着棋盘移动。这种假设让我们只需要知道 X，Y 的值就可以了。现在为了求 X，Y 的值，我们只需要传入这些点（0,0），（1,0），（2,0）...，它们代表了点的位置。在这个例子中，我们的结果的单位就是棋盘（单个）方块的大小。但是如果我们知道单个方块的大小（加入说 30mm），我们输入的值就可以是（0,0），（30,0），（60,0）...，结果的单位就是 mm。（在本例中我们不知道方块的大小，因为不是我们拍的，所以只能用前一种方法了）。

3D 点被称为 对象点，2D 图像点被称为 图像点。

**设置**

为了找到棋盘的图案，我们要使用函数 cv2.findChessboardCorners()。我们还需要传入图案的类型，比如说 8x8 的格子或 5x5 的格子等。在本例中我们使用的恨死 7x8 的格子。（通常情况下棋盘都是 8x8 或者 7x7）。它会返回角点，如果得到图像的话返回值类型（Retval）就会是 True。这些角点会按顺序排列（从左到右，从上到下）。

其他：这个函数可能不会找出所有图像中应有的图案。所以一个好的方法是编写代码，启动摄像机并在每一帧中检查是否有应有的图案。在我们获得图案之后我们要找到角点并把它们保存成一个列表。在读取下一帧图像之前要设置一定的间隔，这样我们就有足够的时间调整棋盘的方向。继续这个过程直到我们得到足够多好的图案。就算是我们举得这个例子，在所有的 14 幅图像中也不知道有几幅是好的。所以我们要读取每一张图像从其中找到好的能用的。

其他：除了使用棋盘之外，我们还可以使用环形格子，但是要使用函数cv2.findCirclesGrid() 来找图案。据说使用环形格子只需要很少的图像就可以了。

在找到这些角点之后我们可以使用函数 cv2.cornerSubPix() 增加准确度。我们使用函数 cv2.drawChessboardCorners() 绘制图案。

## 实例，对14张图片做矫正

|  |  |
| --- | --- |
|  | **import cv2 import numpy as np import glob  *# termination criteria* criteria = (cv2.TERM\_CRITERIA\_EPS|cv2.TERM\_CRITERIA\_MAX\_ITER,30,0.001) *# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)* objp = np.zeros((6\*7,3),np.float32) objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2) *# Arrays to store object points and image points from all the images.* objpoints = [] *# 3d point in real world space* imagepoints = [] *# 2d points in image plane. # 批量获取当前文件夹下面的jpg文件路径* images = glob.glob('\*.jpg') *# 变量路径列表* for fname in images:  *#读取图片，并且转化为灰度图* img = cv2.imread(fname)  gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)  *# Find the chess board corners* ret,corners = cv2.findChessboardCorners(gray,(7,6),None)  *# If found, add object points, image points (after refining them)* if ret:  objpoints.append(objp)  corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)  imagepoints.append(corners)  *# Draw and display the corners* cv2.drawChessboardCorners(img,(7,6),corners2,ret)  cv2.imshow("pic", img)  cv2.waitKey(0)   cv2.destroyAllWindows()** |

### 效果，会做多次，每一次的效果类似下面的图片

|  |
| --- |
| image2 |

### 上面只是找到了角点

## 标定

在得到了这些对象点和图像点之后，我们已经准备好来做摄像机标定了。

我们要使用的函数是 cv2.calibrateCamera()。它会返回摄像机矩阵，畸变系数，旋转和变换向量等。

### ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)

## 畸变矫正

现在我们找到我们想要的东西了，我们可以找到一幅图像来对他进行校正了。OpenCV 提供了两种方法，我们都学习一下。不过在那之前我们可以使用从函数 cv2.getOptimalNewCameraMatrix() 得到的自由缩放系数对摄像机矩阵进行优化。如果缩放系数 alpha = 0，返回的非畸变图像会带有最少量的不想要的像素。它甚至有可能在图像角点去除一些像素。如果 alpha = 1，所有的像素都会被返回，还有一些黑图像。它还会返回一个 ROI 图像，我们可以用来对结果进行裁剪。

我们读取一个新的图像（left2.ipg）

**img = cv.imread('left12.jpg')**

**h, w = img.shape[:2]**

**newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))**

## 使用 cv2.undistort() 这是最简单的方法。只需使用这个函数和上边得到的 ROI 对结果进行裁剪。

**# undistort**

**dst = cv.undistort(img, mtx, dist, None, newcameramtx)**

**# crop the image**

**x, y, w, h = roi**

**dst = dst[y:y+h, x:x+w]**

**cv.imwrite('calibresult.png', dst)**

## 实例2

|  |  |
| --- | --- |
|  | **import cv2 import numpy as np import glob  *# termination criteria* criteria = (cv2.TERM\_CRITERIA\_EPS|cv2.TERM\_CRITERIA\_MAX\_ITER,30,0.001) *# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)* objp = np.zeros((6\*7,3),np.float32) objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2) *# Arrays to store object points and image points from all the images.* objpoints = [] *# 3d point in real world space* imagepoints = [] *# 2d points in image plane. # 批量获取当前文件夹下面的jpg文件路径* images = glob.glob('\*.jpg') *# 变量路径列表* for fname in images:  *#读取图片，并且转化为灰度图* img = cv2.imread(fname)  gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)  *# Find the chess board corners* ret,corners = cv2.findChessboardCorners(gray,(7,6),None)  *# If found, add object points, image points (after refining them)* if ret:  objpoints.append(objp)  corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)  imagepoints.append(corners)  *# Draw and display the corners* cv2.drawChessboardCorners(img,(7,6),corners2,ret)  *# cv2.imshow("pic", img)  # 做摄像机标定* ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,imagepoints,gray.shape[::-1],None,None) *#我们读取一个新的图像* img = cv2.imread('left12.jpg') h, w = img.shape[:2] newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h)) *# undistort* dst = cv2.undistort(img, mtx, dist, None, newcameramtx)  *# crop the image* x, y, w, h = roi dst = dst[y:y+h, x:x+w] cv2.imwrite('calibresult.png', dst) cv2.waitKey(0)   cv2.destroyAllWindows()** |

## 效果：矫正后的图片如下

|  |
| --- |
|  |

## 使用 remapping 这应该属于“曲线救国”了。首先我们要找到从畸变图像到非畸变图像的映射方程。再使用重映射方程。

**# undistort**

**mapx, mapy = cv.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (w,h), 5)**

**dst = cv.remap(img, mapx, mapy, cv.INTER\_LINEAR)**

**# crop the image**

**x, y, w, h = roi**

**dst = dst[y:y+h, x:x+w]**

**cv.imwrite('calibresult2.png', dst)**

## 实例3

|  |  |
| --- | --- |
|  | **import cv2 import numpy as np import glob  *# termination criteria* criteria = (cv2.TERM\_CRITERIA\_EPS|cv2.TERM\_CRITERIA\_MAX\_ITER,30,0.001) *# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)* objp = np.zeros((6\*7,3),np.float32) objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2) *# Arrays to store object points and image points from all the images.* objpoints = [] *# 3d point in real world space* imagepoints = [] *# 2d points in image plane. # 批量获取当前文件夹下面的jpg文件路径* images = glob.glob('\*.jpg') *# 变量路径列表* for fname in images:  *#读取图片，并且转化为灰度图* img = cv2.imread(fname)  gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)  *# Find the chess board corners* ret,corners = cv2.findChessboardCorners(gray,(7,6),None)  *# If found, add object points, image points (after refining them)* if ret:  objpoints.append(objp)  corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)  imagepoints.append(corners)  *# Draw and display the corners* cv2.drawChessboardCorners(img,(7,6),corners2,ret)  *# cv2.imshow("pic", img)  # 做摄像机标定* ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,imagepoints,gray.shape[::-1],None,None) *#我们读取一个新的图像* img = cv2.imread('left12.jpg') h, w = img.shape[:2] newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h)) *# undistort* mapx, mapy = cv2.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (w,h), 5) dst = cv2.remap(img, mapx, mapy, cv2.INTER\_LINEAR)  *# crop the image* x, y, w, h = roi dst = dst[y:y+h, x:x+w] cv2.imwrite('calibresult2.png', dst) cv2.waitKey(0)   cv2.destroyAllWindows()** |

### 效果和上面是一样的

|  |
| --- |
|  |

你会发现结果图像中所有的边界都变直了。

现在我们可以使用 Numpy 提供写函数（np.savez，np.savetxt 等）将摄像机矩阵和畸变系数保存以便以后使用。

## 把结果保存到一个b2.npz文件中，这里使用的是numpy的savez方法，使用命名保存的方式

|  |
| --- |
| **import** cv2 **import** numpy **as** np **import** glob    *# termination criteria* criteria = (cv2.TERM\_CRITERIA\_EPS|cv2.TERM\_CRITERIA\_MAX\_ITER,30,0.001) *# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)* objp = np.zeros((6\*7,3),np.float32) objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2) *# Arrays to store object points and image points from all the images.* objpoints = [] *# 3d point in real world space* imagepoints = [] *# 2d points in image plane. # 批量获取当前文件夹下面的jpg文件路径* images = glob.glob(**'\*.jpg'**) *# 变量路径列表* **for** fname **in** images:  *#读取图片，并且转化为灰度图* img = cv2.imread(fname)  gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)  *# Find the chess board corners* ret,corners = cv2.findChessboardCorners(gray,(7,6),**None**)  *# If found, add object points, image points (after refining them)* **if** ret:  objpoints.append(objp)  corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)  imagepoints.append(corners)  *# Draw and display the corners* cv2.drawChessboardCorners(img,(7,6),corners2,ret)  *# cv2.imshow("pic", img)  # 做摄像机标定* ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,imagepoints,gray.shape[::-1],**None**,**None**) *#保存上面的矫正的结果，使用命名参数保存法* np.savez(**'b2.npz'**,ret=ret, mtx=mtx, dist=dist, rvecs=rvecs, tvecs=tvecs) *#我们读取一个新的图像* img = cv2.imread(**'left12.jpg'**) h, w = img.shape[:2] newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h)) *# undistort* dst = cv2.undistort(img, mtx, dist, **None**, newcameramtx)  *# crop the image* x, y, w, h = roi dst = dst[y:y+h, x:x+w] cv2.imwrite(**'calibresult.png'**, dst) cv2.waitKey(0)   cv2.destroyAllWindows() |

### 效果：

|  |
| --- |
|  |

### 注意：这个文件需要保存好，下一节需要用到

## 反向投影误差

我们可以利用反向投影误差对我们找到的参数的准确性进行估计。得到的结果越接近 0 越好。有了内部参数，畸变参数和旋转变换矩阵，我们就可以使用 cv2.projectPoints() 将对象点转换到图像点。然后就可以计算变换得到图像与角点检测算法的绝对差了。然后我们计算所有标定图像的误差平均值。

**mean\_error = 0**

**for i in range(len(objpoints)):**

**imgpoints2, \_ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)**

**error = cv2.norm(imgpoints[i], imgpoints2, cv.NORM\_L2)/len(imgpoints2)**

**mean\_error += error**

**print( "total error: {}".format(mean\_error/len(objpoints)) )**