相机标定入门详解

Dezeming Family

2022 年 12 月 26 日

DezemingFamily 系列书和小册子因为是电子书，所以可以很方便地进行修改和重新发布。如果您获得了 DezemingFamily 的系列书，可以从我们的网站 [https://dezeming.top/] 找到最新版。对书的内容建议和出现的错误欢迎在网站留言。

目录

一　相机标定的目标

在计算机视觉应用中，尤其是三维重建和三维理解，为了能从一张图像中知道各个物体和目标点之间的位置关系，需要建立相机成像的几何模型（其实就是相机的参数）。

相机参数分为内参、外参合畸变参数。参数标定的精度会影响后面很多操作（一般来说，参数标定是三维重建分析中的第一步）。

外参就是相机在世界空间的位置（包括其朝向角度）。

内参包括相机焦距和光心，在相机出厂时就已经确定了（当然这里的焦距是用小孔成像标定来模拟的焦距，不需要考虑跟真实感相机那种光圈离焦模糊效果有关），当我们不知道时，就可以根据一些标定技术来自己标定。

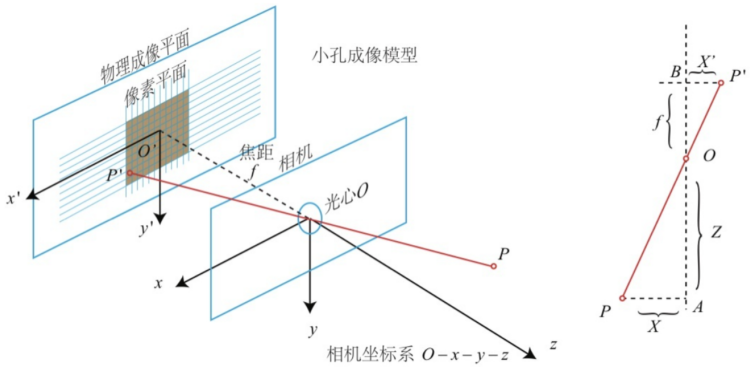
畸变 (distortion) 可以简单理解为是“直线扭曲”，就是本来在场景中是直线，但是相机成像以后在成像面上变为了曲线。

畸变和内参是相机的自身属性，只需要标定一次就可以永久使用。但需要注意的是，由于标定是借助小孔成像原理的，而实际相机成像原理更复杂，可能需要在视野变化较大时重新标定。

二　相机内参和外参

2 1　相机内参

相机坐标系如下图 $O-x-y-z$（图片来自 [3]）：

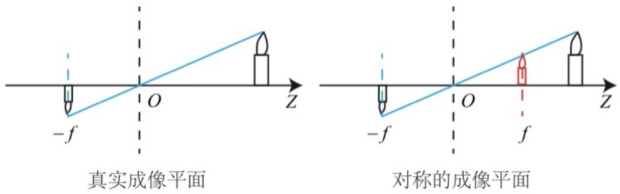


O 是光心，其实就是针孔模型中的孔。f 是焦距，即光心到成像平面的距离。

设真实世界点 $\mathbf{P}=[X,Y,Z]$ 投影到像素屏幕为 $\mathbf{P}'=[X',Y',Z']$，根据上图右的相似三角形关系：

$$\frac{Z}{f}=-\frac{X}{X'}=-\frac{Y}{Y'} \tag{二.1}$$

为了表示更方便，通过对称的方式消除负号：



得到关系式：

$$X'=f\frac{X}{Z}$$
$$Y'=f\frac{Y}{Z}$$

之后还要把成像平面的点映射到像素坐标系上。设 $\mathbf{P}'$ 在像素坐标系的坐标是 $[u,v]$。实际物理世界中距离的基本单位是“米”，而像素空间中基本单位是“像素”。要将“米”转换到“像素”，需要一种放缩关系；而且像素坐标系的原点可能不在像素平面正中心，而是在最左上角位置上，所以需要像素相对于中心的偏

移。

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} \alpha f & 0 & c_x \\ 0 & \beta f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{Z}\mathbf{KP} \tag{二.2}$$

其中，$[c_x, c_y]$ 就是偏移，α, β 是放缩（很多时候两个值是一样的），很多时候人们也习惯表示为 $\alpha = 1/d_x$，$\beta = 1/d_y$，这里 d_x 和 d_y 分别表示像素的长和宽在真实世界中的大小（比如一个像素的长是 0.0001 米，即 0.1 毫米）。$\mathbf{K}$ 就是相机内参，当相机拍摄的屏幕大小固定，且镜头不变焦，则该值就是固定的（注意变焦时，我们一般只能使用聚焦平面处的物体来计算内外参，模糊处的物体不好捕获特征点）。

2 2　相机外参

在前面的表示中，$\mathbf{P}$ 是相机坐标系下描述的点，因此才能用来根据相似三角形求解对应关系。对于世界空间的点 $\mathbf{P}_w$，需要首先变换到相机坐标系下。

如果大家还记得矩阵变换中的基变换，应该就明白应该如何操作了。所谓基变换，就是给定一组基在第二组基下的表示（比如相机坐标系的 $x - y - z$ 基在世界空间坐标系 $x_w - y_w - z_w$ 下的表示），那么用第二组基中描述的坐标点分别与第一组基求内积，就能得到其在第一组基描述的空间中的坐标。

除了基变换，我们也可以将坐标变换分解为旋转和平移：

$$\mathbf{P} = \mathbf{R}\mathbf{P}_w + \mathbf{t} \tag{二.3}$$

$\mathbf{R}$ 表示旋转矩阵，$\mathbf{t}$ 表示平移向量。注意这里的操作一般都是用齐次坐标系来完成的，将旋转和平移变换写到一个矩阵中：

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{二.4}$$

旋转矩阵 $\mathbf{R}$ 虽然是 3×3 的矩阵，但是旋转只需要三个参数就能构建出这个矩阵（本质来说，绕 x 轴转多少度、绕 y 轴转多少度、绕 z 轴转多少度，分解为这三个量即可，尽管实际中会使用 Rodrigues 旋转公式，但也是需要三个参数），我们后面会再详细介绍。

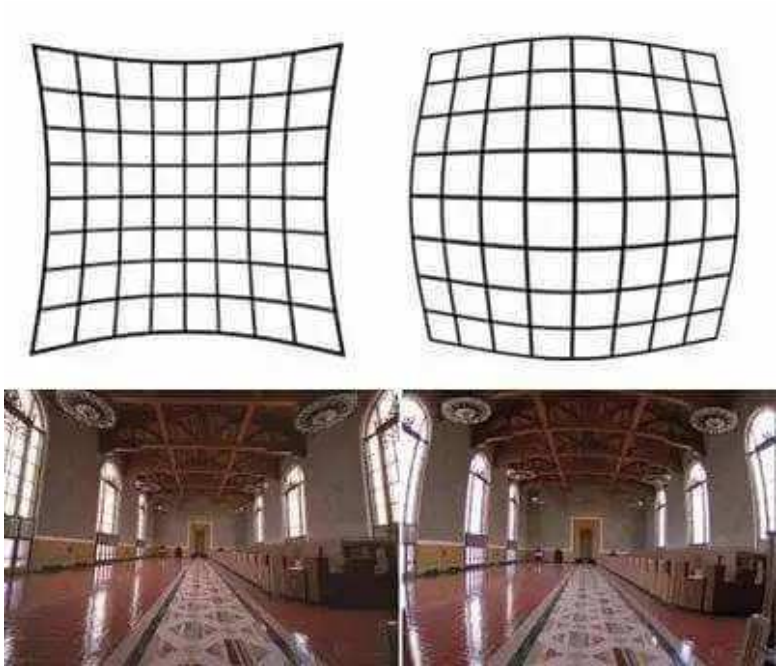
三　相机畸变

畸变一共有两大类，一类是径向畸变 (radial distortion)，另一类是切向畸变 (tangential distortion)。径向畸变是由于透镜形状产生的畸变；切向畸变是在安装中导致的。另外还有薄透镜畸变，会在后面简单介绍一下。

3 1　径向畸变

径向畸变是由于透镜径向曲率的不规则，从而造成越远离透镜中心的区域到成像面以后的畸变越大。径向畸变的效果一般是桶形畸变和枕形畸变：

实际效果如下：



枕形畸变主要是长焦镜头造成的畸变；桶形畸变主要是短焦镜头（又叫广角镜头）造成的畸变。大家可以用相机去试试广角拍摄和长焦拍摄的不同的视觉感受，广角拍出的场面范围更大，而长焦显示的物体看起来会更聚集。

当然，通常意义上描述这两种畸变，一般都指比较小的畸变，和一些特殊相机得到的图像（比如鱼眼相机获得的鱼眼图像，参考 [9]）相比畸变并不大。

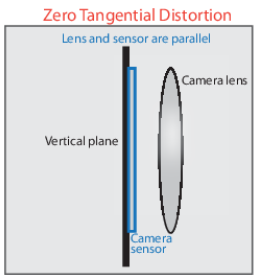
张正友博士的论文 [15] 研究了径向畸变的标定方案（没有考虑切向畸变），根据在 $r=0$ 处的泰勒展开式来近似畸变（x_{dr} 和 y_{dr} 是畸变后的坐标）：

$$\begin{aligned} u_{dr} &= u(1+k_1r^2+k_2r^4+k_3r^6) \\ u_{dr} &= u(1+k_1r^2+k_2r^4+k_3r^6) \end{aligned} \tag{三.1}$$

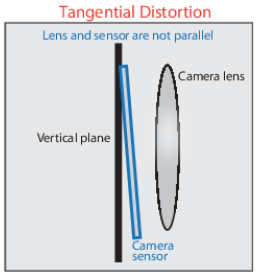
其中 $r^2=x^2+y^2$，r 是距中心像素的距离（经过前面 $[c_x\ c_y]$ 的偏移，图像中心的像素变为原点）。

3 2 切向畸变

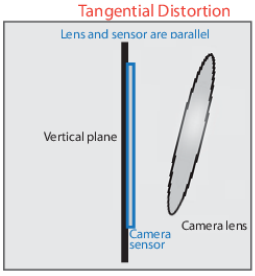
切向畸变是安装中镜头之间以及镜头和成像面之间的位置偏差或不平行导致的，切向畸变又分为不平行畸变和离心畸变等。不平行畸变是因为透镜和成像面之间存在一定的细微倾斜 [10]，这会使得平行于镜头的正方形图像映射为梯形；离心畸变是由于镜头由多个透镜组合而成，而各个透镜的光轴不在同一条中心线上。



无切向畸变



切向畸变



切向畸变

畸变公式为：

$$\begin{aligned} u_{dr} &= u+\left[2p_1v+p_2(r^2+2u^2)\right] \\ v_{dr} &= v+\left[p_1(r^2+2v^2)+2p_2u\right] \end{aligned} \tag{三.2}$$

3 3　薄透镜畸变

薄透镜畸变由镜头设计缺陷和加工误差导致的，畸变公式为（δ_{xt} 和 δ_{yt} 是畸变后的坐标）：

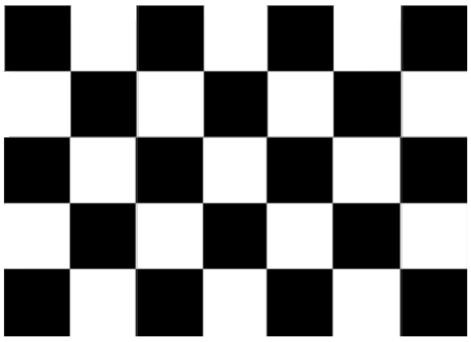
$$\begin{aligned}\delta_{xt} &= s_1(u^2+v^2)+O(u,v)^4 \\ \delta_{yt} &= s_2(u^2+v^2)+O(u,v)^4\end{aligned} \tag{三.3}$$

其中 s_1 和 s_2 是畸变参数。但是由于它的影响实在是比较小，所以常常会忽略。

四　使用标定板的相机标定基本过程

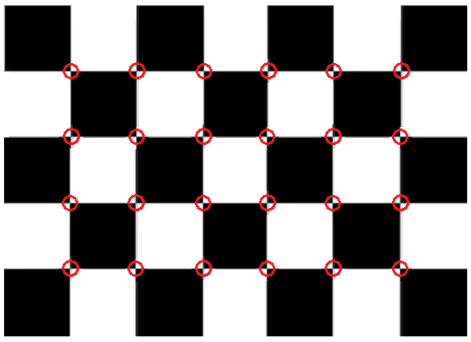
4 1　使用标定板标定

制作一个棋盘格（最好用程序生成），然后打印出来，最好能打印在硬纸板上防止变形：



之后在多个位置进行拍照，要尽量拍出标定板全貌，为了保证拟合参数时更准确，最好多拍一些照片（比如 20 张）：

我们关注的是内角点，下图红圈表示内角点，一排 6 个内角点，一列 4 个内角点：



如果图像没有畸变或者畸变较小，就可以用传统的线性相机标定方法。如果相机有明显畸变，那么就需要使用非线性相机标定方法。在 OpenCV 里，都是用 calibrateCamera 函数标定，如果畸变严重，则可以用 undistort 处理畸变。

直接描述过程会很抽象，我们用 OpenCV 的代码感受一下。网上的代码流程和官方流程都比较相似，这里我参考借鉴的是 [17] 的过程。

4 2　相机标定过程

导入的包

我们主要用到了下面三个包：

```
import cv2
import numpy as np
# glob用于获得文件夹下全部图像文件
import glob
```

棋盘格角点设置

定义棋盘格角点：

```
# 设定阈值
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
#棋盘格模板规格，即内角点个数
w = 6
h = 4
```

棋盘格角点设置

这里首先设置一个世界坐标，为了方便起见，我们直接设板子左上角的点就在世界坐标原点，然后第一排的第二个点在世界坐标 [1. 0. 0.] 位置：

```
# 世界坐标系中的棋盘格点,即[0. 0. 0.], [1. 0. 0.], [2. 0. 0.],....,[5. 3.
    0.]
objp = np.zeros((w*h,3), np.float32)
objp[:,:2] = np.mgrid[0:w,0:h].T.reshape(-1,2)
# 储存棋盘格角点的世界坐标和图像坐标对
objpoints = [] # 在世界坐标系中的三维点
imgpoints = [] # 在图像平面的二维点
```

在标定时，都会默认这种世界坐标形式，并且认为物体没有动，动的只是相机（后面计算得到的每幅图 [i] 的 rvecs[i] 和 tvecs[i] 都是独立的）。

找棋盘格角点并显示

```
# 获取图像序列
images = glob.glob('picture/*.jpg')
# 依次处理每个图像
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    # 找到棋盘图像(8位灰度或彩色图像)中的棋盘格角点
    ret, corners = cv2.findChessboardCorners(gray, (w,h),None)
    # 如果找到足够点对，将其存储起来
    if ret == True:
        # 角点精确检测
```

```
        # 输入图像 角点初始坐标 搜索窗口为2*winsize+1 死区 求角点的迭代终止条件
        cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        objpoints.append(objp)
        imgpoints.append(corners)
        # 将角点在图像上显示
        cv2.drawChessboardCorners(img, (w,h), corners, ret)
        cv2.imshow('findCorners',img)
        cv2.waitKey(1000)
cv2.destroyAllWindows()
```

相机标定

进行相机标定：

```
#标定、去畸变
# 输入：世界坐标系里的位置 像素坐标 图像的像素尺寸大小 3*3矩阵，相机内参数矩阵 畸变矩阵
# 输出：标定结果 相机的内参数矩阵 畸变系数 旋转矩阵 平移向量
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
```

注意 rvecs 和 tvecs 都是数组。

打印输出标定结果：

```
print (("ret:"),ret)
# 内参矩阵
print (("mtx:\n"),mtx)
# dist:畸变系数(k_1,k_2,p_1,p_2,k_3)
print (("dist:\n"),dist)
# rvecs:旋转向量(外参); tvecs:平移向量(外参)
print (("rvecs:\n"),rvecs)
print (("tvecs:\n"),tvecs)
```

去畸变

我们已经得到了相机内参和畸变系数,在将图像去畸变之前,还可以使用 cv.getOptimalNewCameraMatrix() 优化内参和畸变系数，通过设定自由比例因子 alpha。当 alpha 设为 0 时，将会返回一个剪裁过的将去畸变后不想要的像素去掉的内参和畸变系数；当 alpha 设为 1 时，会返回一个包含额外黑色像素点的内参和畸变系数，并返回一个 ROI 用于将其剪裁掉：

```
img2 = cv2.imread('picture/6.jpg')
h,w = img2.shape[:2]
newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),0,(w,h))
```

去畸变：

```
dst = cv2.undistort(img2, mtx, dist, None, newcameramtx)
# 根据前面ROI区域裁剪图片
x,y,w,h = roi
dst = dst[y:y+h, x:x+w]
```

```
5 cv2.imwrite('calibresult.jpg',dst)
```

计算反投影误差

通过反投影误差，我们可以来评估结果的好坏。越接近 0，说明结果越理想。通过之前计算的内参矩阵、畸变系数、旋转矩阵和平移向量，使用 cv2.projectPoints() 计算三维点到二维图像的投影，然后计算反投影得到的点与图像上检测到的点的误差，最后计算一个对于所有标定图像的平均误差，这个值就是反投影误差。

```
1 total_error = 0
2 for i in range(len(objpoints)):
3     imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx,
          dist)
4     error = cv2.norm(imgpoints[i],imgpoints2, cv2.NORM_L2)/len(imgpoints2)
5     total_error += error
6 print(("total error: "), total_error/len(objpoints))
```

参考文献

[1] https://zhuanlan.zhihu.com/p/583883569

[2] https://blog.51cto.com/u_15242250/2870251

[3] https://zhuanlan.zhihu.com/p/389653208

[4] https://blog.csdn.net/weixin_43206570/article/details/84797361

[5] https://blog.csdn.net/m0_49332456/article/details/121011500

[6] https://blog.51cto.com/u_15242250/2870251

[7] https://zhuanlan.zhihu.com/p/552607272

[8] https://blog.csdn.net/qq_42744739/article/details/125911766

[9] https://zhuanlan.zhihu.com/p/511284263

[10] https://zhuanlan.zhihu.com/p/23090593

[11] https://blog.csdn.net/m0_43609475/article/details/112832897

[12] http://www.senlt.cn/article/412446713.html

[13] https://blog.csdn.net/weixin_44010117/article/details/107989223

[14] https://www.jianshu.com/p/6daa8dbbfa30

[15] Zhang Z. Flexible camera calibration by viewing a plane from unknown orientations[C]//Proceedings of the seventh ieee international conference on computer vision. Ieee, 1999, 1: 666-673.

[16] Zhang Z. A flexible new technique for camera calibration[J]. IEEE Transactions on pattern analysis and machine intelligence, 2000, 22(11): 1330-1334.

[17] https://www.jb51.net/article/257634.htm