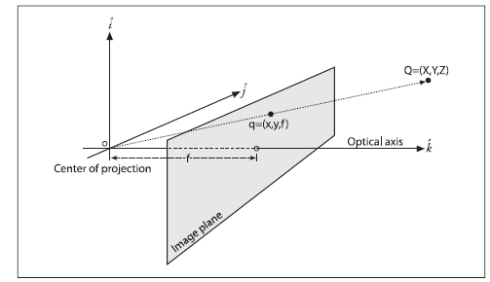
相机标定【Base Theory】

2019年7月24日

9:11

相机标定 = 获取相机内参&&畸变参数（同一相机为固定）、外参（针对不同图片）



（为了计算方便，我们将像平面翻转平移到针孔前）

1. 外参矩阵： 世界坐标系Pw -> 相机坐标系Pc

计算机生成了可选文字:
Pc 
R3x3 × PW + T3x1 

引入齐次坐标，转化为线性变换

计算机生成了可选文字:
R 
0 
t 
1 
YW 
Zw 

假设点云图像是以真实世界中任意一点为原点建立的。

2. 内参矩阵：相机坐标系 -> 图像坐标系【左上角(0,0), 单位像素】

计算机生成了可选文字:
了 / 
0 
1 
丐 
0 
鬱 0 
1 

|  |  |
| --- | --- |
| \gamma | 倾斜参数，理想情况下为0 |
| dx，dy | pixel/mm; 一个像素代表的实际物理值的大小，实现图像物理坐标系与像素坐标系转换的关键(一般二者等值); f / dx = fx; f/ dy = fy.【数量级 = e+02】 |
| u0，v0 | pixel; 图像的中心像素坐标和图像原点像素坐标之间相差的横向和纵向像素数  【近似等于图像宽、高的一半】 |

最终， 建立 世界坐标系的点(X, Y, Z) 与图像坐标系点 (u, v)之间的联系

计算机生成了可选文字:
= KLRItl 

计算机生成了可选文字:
图 像 点 
相 机 内 参 数 
相 机 硬 件 
相 机 旋 转 相 机 平 移 空 间 点 
相 机 位 姿 
场 景 结 构 

t = (Tx,Ty,Tz)，是平移向量，

R = R(α,β,γ)是旋转矩阵

【实际获取：利用opencv 的内置工具】

\*\*\* openCV 除了可以获取内外参，同时还可以得到畸变系数，对图像进行校正；  
圆圈格点校正将会使用更少的图像。

<https://github.com/LongerVision/Examples_OpenCV/blob/master/01_internal_camera_calibration/circle_grid.py>

角点法步骤：

0. 世界坐标系以棋盘格上的点作为标准，单位也是棋盘格的一格；

0+， 元素过程中寻找特征为角点检测，只需要灰度图

首先使用待校正相机，对ChessBoard 进行多角度拍摄；（20-30pic）算法的实质是使用最小二乘法解算上述模型中的未知数;

1. 标定.

输入： 同一相机拍摄，包含棋盘格的多张图像；棋盘格上有效角点的个数

输出： 该相机的K, 畸变系数； 图像Lst中每个图像对应的外参（旋转、平移向量）

|  |  |
| --- | --- |
| ret, corners =  cv2.findChessboardCorners(Gimg, (7,6),None) | 利用Harris获取灰度图像中的所有棋盘格角点，  ret 为 Bool,  corners为np\_Arr,shape = [7\*6, 1, 2], dtype = float |
| CORNER\_CRITERIA = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 30, 0.001)  corners2 = cv2.cornerSubPix(cur\_Gimg,corners,(11,11),(-1,-1),CORNER\_CRITERIA) | 将上一步得到的角点坐标进行亚像素级转化,corners2比corners更精确。 shape, dtype相同    (11,11)是窗口尺寸， 后面两个参数是zeroZone, criteria |
| cv2.drawChessboardCorners(cur\_img, (7,6), corners2,ret)    if ret == True:  # 每找到一次corner2, 将其压入图像坐标 imgpoints\_Lst.append(corners2)  # 同时将常数矩阵压入对应的对象坐标  objpoints\_Lst.append(objp) | 尽管计算过程在灰度图上进行，但是可以在对应的彩图上进行可视化；   存储参与标定的所有有效图像的匹配点对 |
| RMS, \  mtx, dist, \  rvecs\_Lst, tvecs\_Lst = \  cv2.calibrateCamera( objpoints\_Lst, imgpoints\_Lst,  (t\_imgW, t\_imgH),None,None) | 利用匹配束objpoints\_Lst<->imgpoints\_Lst, 计算生成：  # <float>RMS, 平均重新投影误差 RMS【良好校准时应在0.1和1.0像素之间】    # <numpy.ndarray>mtx, 内参矩阵 camera matrix  # <numpy.ndarray>dist , 畸变系数 Distortion coefficients    # <Lst> rvecs\_Lst 有效图像的外参矩阵\_旋转参数 列表， 内部元素为3x1 矩阵  # <Lst>tvecs\_Lst 有效图像的外参矩阵\_平移参数 列表， 内部元素为3x1 矩阵 |
| imgpoints\_i\_check = cv2.projectPoints(objpoints\_Lst[i], rvecs\_Lst[i], tvecs\_Lst[i], mtx, dist)    # RMS即计算所有imgpoints\_i\_check vs. imgpoints\_Lst[i]的均方差 | 自定义重投影过程 原理上可以计算当前视图下，  以棋盘格为空间坐标系（单位为1格）上的任意真实3D点 投影到图像中对应的位置， |
| ret, rvecs, tvecs, inliers = cv2.solvePnPRansac(objp, corners2, mtx, dist) | 姿态估计 根据匹配obj<->img, 估计单张图像中的外参  猜测： cv2.calibrateCamera() 的实现过程应当是先找到相机的公共内参；  之后通过每张图像的cv2.solvePnPRansac() 来计算当前图像的外参（旋转和平移）  ret 是flag inliers 自然数整型？    rvecs, tvecs,即从单张图像求取的外参 |

2. 畸变校正

输入：拍摄相机的内参矩阵mtx、畸变系数dist

|  |  |
| --- | --- |
| newcameramtx, roi = \  cv2.getOptimalNewCameraMatrix(mtx,dist,  (t\_imgW,t\_imgH),1,(t\_imgW,t\_imgH)) | 利用mtx, dist获取新的“内参矩阵”  roi待标在新投影得到的图像中，非黑的区域  （之前的图像投影过来，边缘不一致） |
| 1) dst = cv2.undistort(Gimg\_i, mtx, dist, None, newcameramtx) | 法一。直接利用该函数，获取校正后的图像 |
| 2)  mapx,mapy = \  cv2.initUndistortRectifyMap(mtx, dist, None,newcameramtx,(t\_imgW,t\_imgH),5)  dst = cv2.remap(Gimg\_i,mapx,mapy,cv2.INTER\_LINEAR)  dst = dst[y:y+h, x:x+w] | 法二。利用Remap  首先得到两个map 矩阵 (cameraMatrix, distCoeffs, R, newCameraMatrix, size, m1type, map1, map2)  mapx, mapy 形状[::-1]均为(t\_imgW,t\_imgH)  之后通过(mapx, mapy)描述的重映射关系，将src转化为dst  只保留非黑区域 |