《多传感器融合感知技术笔记》之
——2.相机标定_Akaxi

任务：使用工具包利用棋盘格对针孔相机进行标定，求相机的内参矩阵，畸变系数，并且还原校正后的相机图像。

1.针孔相机模型

按照镜头的种类可分为针孔相机、鱼眼相机、全景相机。

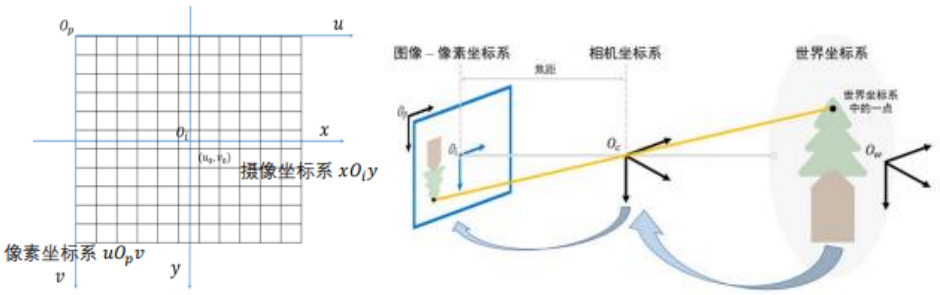


图 1 相机参考系

摄像坐标系是以相机的光轴作为 Z 轴（朝向实物方向），光心***O***i 作为原点。

像素坐标系图像左上角为原点,用(u,v) 表示。

相机内参矩阵：x、y 方向上的焦距和相机中心坐标。

$$\begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix}$$

相机矩阵 K: 由相机内参组成的矩阵。

畸变系数：描述图片畸变的系数。

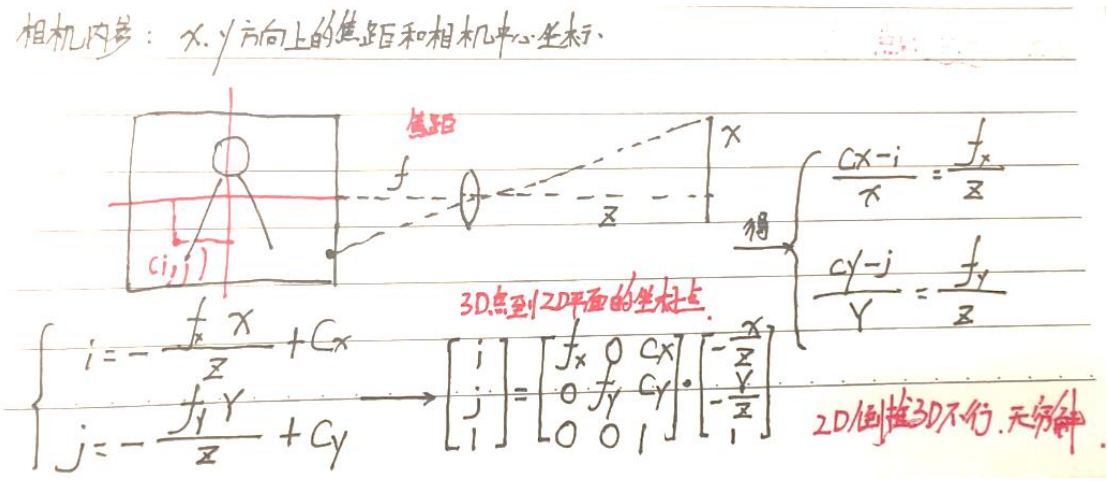


图 2 相机参数推导方程

2.图像畸变及其标定

通常畸变可以分为两种（1）径向畸变；（2）切向畸变

径向畸变：镜头制造工艺不完美，使得镜头形状存在缺陷，通常又分为桶性畸变和枕形畸变

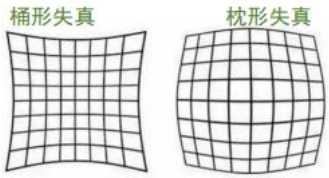


图 3 相机成像失真

切向畸变：安装时候镜头和 CMOS 成像平面不平行。

针孔相机校准：

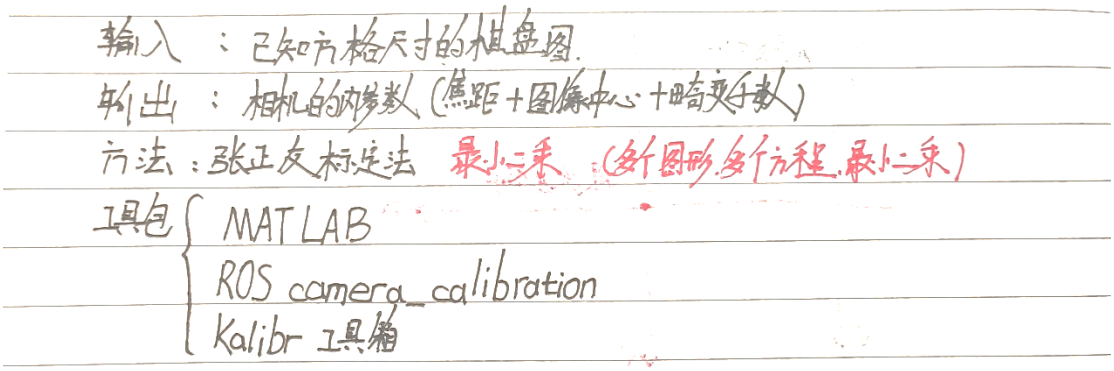


图 4 相机校正示意图

利用工具包结合棋盘图即可实现相机的参数测量以及标定。

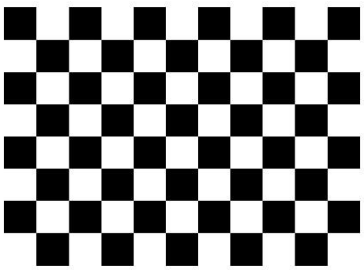


图 5 棋盘格（25mm，11*8，A4）

一、【基于 Python 中 OpenCV 库进行摄像头的标定】

①摄像头、棋盘格的准备

图 6 720P 的 usb 免驱工业摄像头

使用自备的 720P 分辨率的 usb 免驱工业摄像头，接上电脑可以看到其拍摄的棋盘格有明显的畸变，如前文所述的枕形失真。

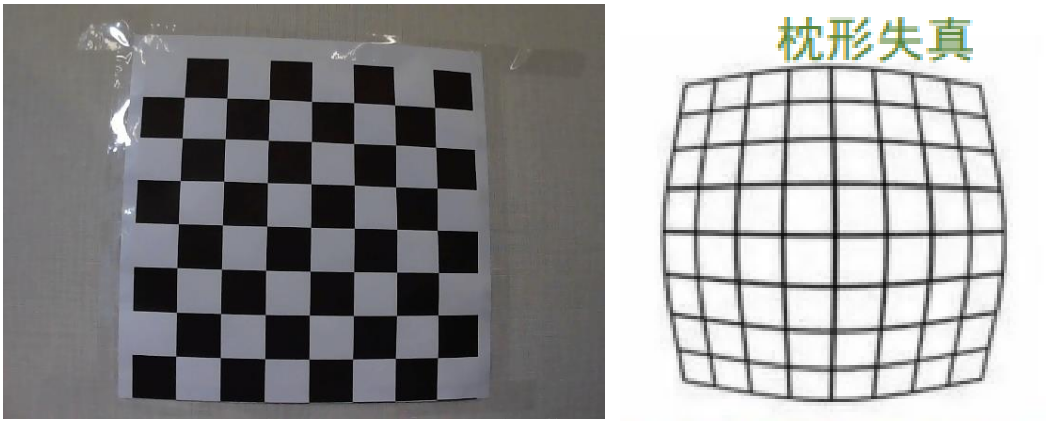


图 7 原相机畸变的棋盘图（左）

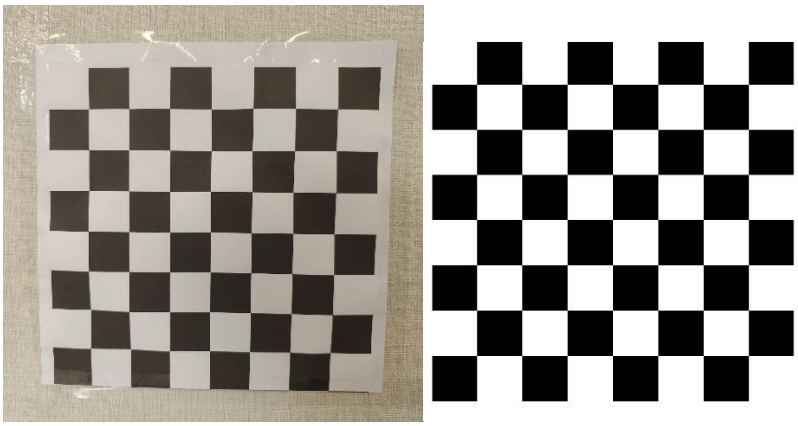


图 8 手机拍摄的棋盘图（左）

观察手机拍摄的图像与原相机畸变的图像我可以很清楚的看到使用的 720P 免驱摄像头有很明显的枕形失真，接下来我利用 Python 中强大的 OpenCV 库对摄像头进行标定，并且求取相机的内参矩阵、畸变系数，最后还原校正后的摄像头图片，打印图片与校正前进行对比，看看校正效果。

②原摄像头图片获取

编写 Python 程序，接上 usb 摄像头后，利用程序自动拍摄 10 张图片保存到指定的目录下。

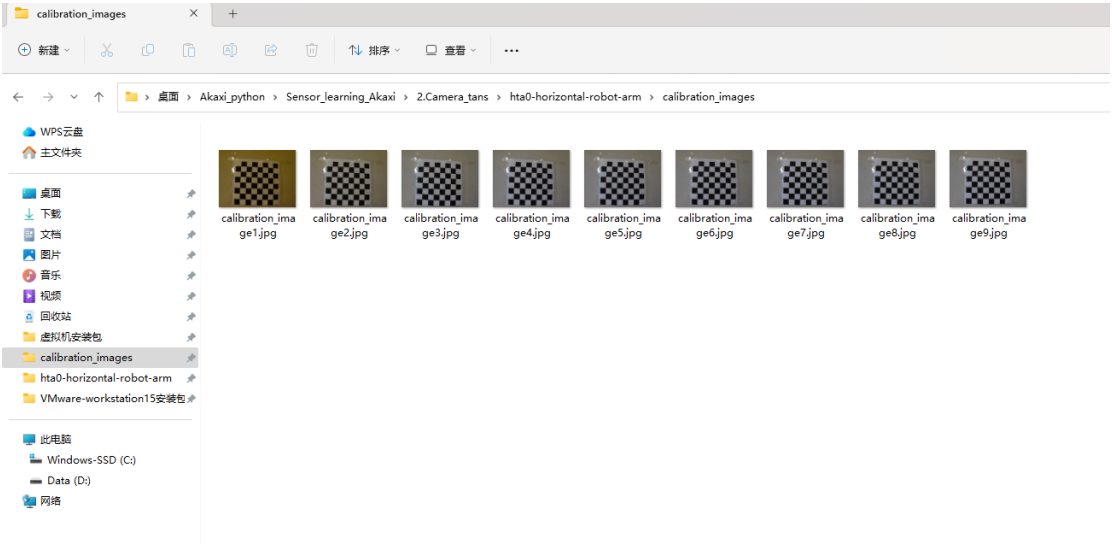


图 9 保存的 10 张校正前图片

图片获取代码如下，当然如果你不用 usb 摄像头，用笔记电脑自带的摄像头的话可以将 `camera = cv2.VideoCapture(1)` 改成 0。

camera_capture.py

```
import cv2
camera = cv2.VideoCapture(1)  # cv2.VideoCapture(1)中的 1 就是外置摄像头
i = 1
while i < 10:  # 拍摄 10 张，可以根据自己需求进行修改
    _, frame = camera.read()
    cv2.imwrite("C:/Users/Akaxi/Desktop/Akaxi_python/Sensor_learning_Akaxi/2.Camera_tans/hta0-horizontal-robot-arm/calibration_images/calibration_image"+str(i)+'.jpg', frame, [int(cv2.IMWRITE_PNG_COMPRESSION), 0])
    cv2.imshow('frame', frame)
    i += 1
    if cv2.waitKey(200) & 0xFF == 27: # 按 ESC 键退出
        break
cv2.destroyAllWindows()
```

注：cv2.VideoCapture 函数用来捕捉视频流、cv2.imwrite 函数用来保存帧图片、cv2.imshow 函数用来显示图片、cv2.waitKey 函数用来等待暂停显示、cv2.destroyAllWindows 函数用来清除。

③初始化参数与变量

这里我的判定校正标准 criteria 参数，初始化全零矩阵 objp 待传入真实世界的坐标，并且对它进行棋盘格大小的变化，生成了真实世界的物体坐标点数组 objpoints[]用来存储图像的真实点，与图像平面的坐标点 imgpoints = []数组，使用 glob.glob()函数对指定目录下的.jpg 图像进行获取，并且创建了一个全屏的窗口便于可视化。

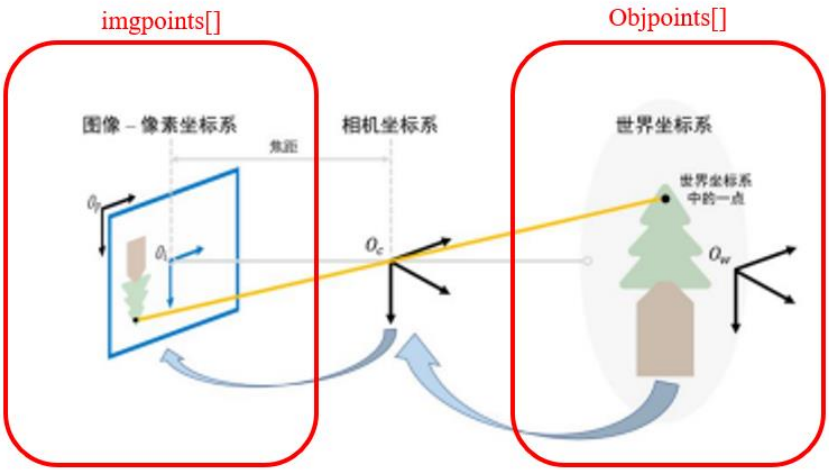


图 10 初始化参数示意图

```
1.      # termination criteria 终止标准
2.
3.      criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# cv2.TERM_CRITERIA_EPS 极小误差 cv2.TERM_CRITERIA_MAX_ITER 最大终止迭代次数
4.      # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
5.      objp = np.zeros((7*7,3), np.float32)  # 生成全 0 的 49 行 3 列数组    【注意这里的
7*7 就是你棋盘格子的行列数】!!!
6.
7.      #add 2.5 to account for 2.5 cm per square in grid  每个格子 2.5cm，根据你打印的
进行设置
8.      objp[:,:2] = np.mgrid[0:7,0:7].T.reshape(-1,2)*2.4  # 原始坐标图   【注意这里的
0:7 会根据你的棋盘格子行列数进行变换】!!后面的*2.5 系数是你实际测量打印的棋盘格单元大小!
9.
10.     # Arrays to store object points and image points from all the images.
11.     objpoints = [] # 3d point in real world space 图像真实点
12.     imgpoints = [] # 2d points in image plane.     图像平面点
13.     images = glob.glob('calibration_images/*.jpg')  # 获取图像
14.
15.     win_name="Verify"  # 创建全屏窗口，可视化
```

```
16.     cv2.namedWindow(win_name, cv2.WND_PROP_FULLSCREEN)  # 创建窗口
名  cv2.WND_PROP_FULLSCREEN 全屏模式
17.     cv2.setWindowProperty(win_name,cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN
)  # 设置窗口
```

④棋盘格格角参数获取与图像展示【未校正】

在读取参数前需要使用 cv2.cvtColor 函数对每一幅图进行灰度化，之后使用 cv2.findChessboardCorners 函数找到棋盘角，如图所示，找到存在棋盘角识别成功的话，就储存初始化的 objpoints[]数组与 imgpoints = []数组，并且使用 cv2.cornerSubPix 函数对棋盘角点进行压细化（粗略估算），最后将识别到的角点参数打印到图片上进行可视化。

图 11 棋盘格角获取图

```
1.      for fname in images:
2.          img = cv2.imread(fname)
3.          print(fname)  # 打印正在标定的图片名字
4.          gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # 灰度化
5.
6.
7.          # Find the chess board corners  找到棋盘格角
8.          ret, corners = cv2.findChessboardCorners(gray, (7,7), None)  # cv2.findC
hessboardCorners 是 OpenCV 中用来自动寻找图像棋盘的函数，【棋盘格 7 行 7 列！！！！  ret 是
布尔类型用来判断有没有识别到棋盘格子 corners 是包含所有角坐标点的 np 数组
9.          # If found, add object points, image points (after refining them)
10.         if ret == True:
11.             objpoints.append(objp)
12.             corners2=cv2.cornerSubPix(gray,corners, (11,11), (-1,-
1), criteria) # 利用 cv2.cornerSubPix 函数对找到的棋盘角点进行压细化
13.             imgpoints.append(corners)  # 图片的角点
```

```
            # Draw and display the corners
            cv2.drawChessboardCorners(img, (7,7), corners2, ret)  # drawChessboardCorners 函数是在棋盘图片上面绘制，(7,7)是你棋盘的行列，corners2 是棋盘格角点坐标
            cv2.imshow(win_name, img)  # 可视化 窗口是我之前创建的全屏窗口
            cv2.waitKey(500)  # 等待 500 毫秒之后结束循环

        img1=img  # 对最后那张图片进行校正

    cv2.destroyAllWindows()
```

⑤相机内参获取【标定】

通过第 4 步的棋盘格角参数的获取（棋盘对应图像），得到真实世界的物体坐标点数组 objpoints[]，与图像平面的坐标点 imgpoints = []数组，接下来使用 OpenCV 自带的相机标定函数 cv2.calibrateCamera 即可获取到相机的内参矩阵、畸变系数，每幅图的旋转向量、平移向量。

图 12 获取到的相机内参

```
    print(">==> Starting calibration")  # 开始标定
    ret, cam_mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)  # cv2.calibrateCamera 函数就是用来计算相机内参的函数，需要这个：objpoints : 3d point in real world space 图像真实点坐标；imgpoints：2d points in image plane.    图像平面点
    # ret 布尔值、cam_mtx 计算出来相机的内参矩阵、dist 相机的畸变系数、rvecs 每幅图的旋转向量、tvecs 每幅图的平移向量

    #print(ret)
    print("Camera Matrix")
    print(cam_mtx)    # 相机的内参矩阵
```

```
    np.save(savedir+'cam_mtx.txt', cam_mtx)  # 保存相机的内参矩阵到 txt 文本

    print("Distortion Coeff")  # 畸变系数
    print(dist)
    np.save(savedir+'dist.txt', dist)

    print("r vecs")  # 每幅图的旋转向量
    print(rvecs[2])

    print("t Vecs")  # 每幅图的平移向量
    print(tvecs[2])
```

⑥相机校正以及显示

使用 cv2.getOptimalNewCameraMatrix 函数就能够很轻松的对原来畸变的相机内参矩阵进行运算，得到校正后的相机内参。补：roi 即计算机感兴趣的区域，在校正后，有形变，削减的边缘区域，这部分没有意义，计算机去掉这部分即为感兴趣的区域（有意义区域），这在计算机的图像处理，机器视觉中有重要的作用。最后将校正后的图像全屏显示，可以观察到图像没有几乎失真，校正效果比较好。

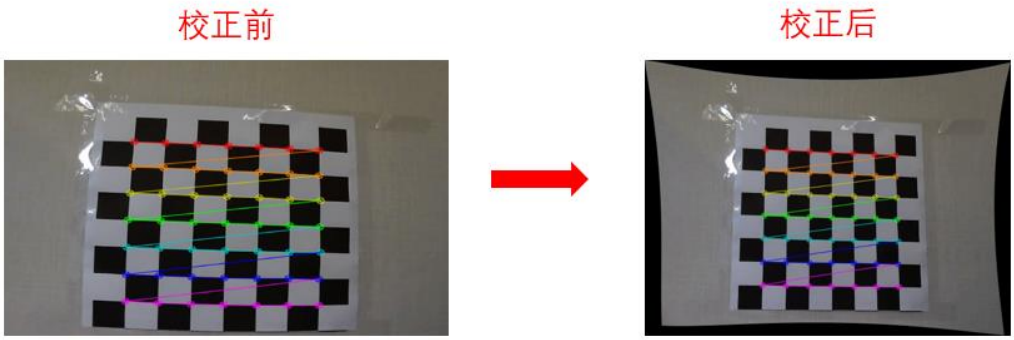


图 13 相机校正图片显示

```
    h,  w = img1.shape[:2]  # 对最后一张图片进行处理
    print("Image Width, Height")
    print(w, h)
    newcam_mtx, roi=cv2.getOptimalNewCameraMatrix(cam_mtx, dist, (w,h), 1, (w,h)
)  # 校正后的相机内参，以及感兴趣的区域

    print("Region of Interest")  # 标定后计算机削减后图像的边缘
    print(roi)
    np.save(savedir+'roi.npy', roi)
```

```
9.
10.     print("New Camera Matrix")  # 校正后的新相机内参矩阵，相机没有失真了就
11.     #print(newcam_mtx)
12.     np.save(savedir+'newcam_mtx.npy', newcam_mtx)
13.     print(np.load(savedir+'newcam_mtx.npy'))
14.
15.     inverse = np.linalg.inv(newcam_mtx)  # 新的相机内参矩阵的逆矩阵，在计算机视觉、图像处理里面常常用其逆矩阵
16.     print("Inverse New Camera Matrix")
17.     print(inverse)
18.
19.
20.     # undistort
21.     undst = cv2.undistort(img1, cam_mtx, dist, None, newcam_mtx)  # 生成校正后的图像 undst
22.     cv2.imwrite("undistorted_image.jpg", undst)
23.
24.     # crop the image
25.     #x, y, w, h = roi
26.     #dst = dst[y:y+h, x:x+w]
27.     #cv2.circle(dst,(308,160),5,(0,255,0),2)
28.     cv2.imshow('img1', img1)
29.     cv2.waitKey(5000)
30.     cv2.destroyAllWindows()
31.     cv2.imshow('img1', undst)
32.     cv2.waitKey(5000)
33.     cv2.destroyAllWindows()
```

⑦基于 Python 的 OpenCV 库进行相机标定源代码

camera_capture.py

```
1.     import cv2
2.     camera = cv2.VideoCapture(1)  # cv2.VideoCapture(1)中的 1 就是外置摄像头
3.     i = 1
4.     while i < 10:
5.         _, frame = camera.read()
6.         cv2.imwrite("C:/Users/Akaxi/Desktop/Akaxi_python/Sensor_learning_Akaxi/2.Camera_tans/hta0-horizontal-robot-arm/calibration_images/calibration_image"+str(i)+'.jpg', frame, [int(cv2.IMWRITE_PNG_COMPRESSION), 0])
```

```
        cv2.imshow('frame', frame)
        i += 1
        if cv2.waitKey(200) & 0xFF == 27: # 按 ESC 键退出
            break
    cv2.destroyAllWindows()
```

initial_camera_calibration.py

```
    #https://docs.opencv.org/3.3.0/dc/dbb/tutorial_py_calibration.html

    import numpy as np
    import cv2
    import glob
    import time

    workingdir="/home/pi/Desktop/Captures/"
    savedir="camera_data/"

    # termination criteria 终止标准

    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# cv2.TERM_CRITERIA_EPS 极小误差 cv2.TERM_CRITERIA_MAX_ITER 最大终止迭代次数
    # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
    objp = np.zeros((7*7,3), np.float32)  # 生成全 0 的 49 行 3 列数组  【注意这里的
7*7 就是你棋盘格子的行列数】!!!

    #add 2.5 to account for 2.5 cm per square in grid  每个格子 2.5cm，根据你打印的
进行设置
    objp[:,:2] = np.mgrid[0:7,0:7].T.reshape(-1,2)*2.4  # 原始坐标图  【注意这里的
0:7 会根据你的棋盘格子行列数进行变换】!!后面的*2.5 系数是你实际测量打印的棋盘格单元大小!

    # Arrays to store object points and image points from all the images.
    objpoints = [] # 3d point in real world space 图像真实点
    imgpoints = [] # 2d points in image plane.    图像平面点
    images = glob.glob('calibration_images/*.jpg')  # 获取图像

    win_name="Verify"  # 创建全屏窗口，可视化
    cv2.namedWindow(win_name, cv2.WND_PROP_FULLSCREEN)  # 创建窗口
名  cv2.WND_PROP_FULLSCREEN 全屏模式
    cv2.setWindowProperty(win_name,cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN
)  # 设置窗口
```

```
28.
29.     print("getting images")
30.     for fname in images:
31.         img = cv2.imread(fname)
32.         print(fname)  # 打印正在标定的图片名字
33.         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # 灰度化
34.
35.
36.         # Find the chess board corners  找到棋盘格角
37.         ret, corners = cv2.findChessboardCorners(gray, (7,7), None)  # cv2.findC
hessboardCorners 是 OpenCV 中用来自动寻找图像棋盘的函数，【棋盘格 7 行 7 列！！！！  ret 是
布尔类型用来判断有没有识别到棋盘格子 corners 是包含所有角坐标点的 np 数组
38.         # If found, add object points, image points (after refining them)
39.         if ret == True:
40.             objpoints.append(objp)
41.             corners2=cv2.cornerSubPix(gray,corners, (11,11), (-1,-
1), criteria) # 利用 cv2.cornerSubPix 函数对找到的棋盘角点进行压细化
42.             imgpoints.append(corners)  # 图片的角点
43.             # Draw and display the corners
44.             cv2.drawChessboardCorners(img, (7,7), corners2, ret)  # drawChessboa
rdCorners 函数是在棋盘图片上面绘制，(7,7)是你棋盘的行列，corners2 是棋盘格角点坐标
45.             cv2.imshow(win_name, img)  # 可视化 窗口是我们之前创建的全屏窗口
46.             cv2.waitKey(500)  # 等待 500 毫秒之后结束循环
47.
48.         img1=img  # 对最后那张图片进行校正
49.
50.     cv2.destroyAllWindows()
51.
52.     print(">==> Starting calibration")  # 开始标定
53.     ret, cam_mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
 gray.shape[::-1], None, None)  # cv2.calibrateCamera 函数就是用来计算相机内参的函数，
需要这个：objpoints : 3d point in real world space 图像真实点坐标；imgpoints：
2d points in image plane.    图像平面点
54.     # ret 布尔值、cam_mtx 计算出来相机的内参矩阵、dist 相机的畸变系数、rvecs 每幅图的旋
转向量、tvecs 每幅图的平移向量
55.
56.     #print(ret)
57.     print("Camera Matrix")
58.     print(cam_mtx)   # 相机的内参矩阵
59.     np.save(savedir+'cam_mtx.txt', cam_mtx)  # 保存相机的内参矩阵到 txt 文本
60.
61.     print("Distortion Coeff")  # 畸变系数
```

```
62.     print(dist)
63.     np.save(savedir+'dist.txt', dist)
64.
65.     print("r vecs")  # 每幅图的旋转向量
66.     print(rvecs[2])
67.
68.     print("t Vecs")  # 每幅图的平移向量
69.     print(tvecs[2])
70.
71.
72.
73.     print(">==> Calibration ended")
74.
75.
76.     h,  w = img1.shape[:2]  # 对最后一张图片进行处理
77.     print("Image Width, Height")
78.     print(w, h)
79.     #if using Alpha 0, so we discard the black pixels from the distortion.  this
 helps make the entire region of interest is the full dimensions of the image (afte
r undistort)
80.     #if using Alpha 1, we retain the black pixels, and obtain the region of inte
rest as the valid pixels for the matrix.
81.     #i will use Apha 1, so that I don't have to run undistort.. and can just cal
culate my real world x,y
82.     newcam_mtx, roi=cv2.getOptimalNewCameraMatrix(cam_mtx, dist, (w,h), 1, (w,h)
) # 校正后的相机内参，以及感兴趣的区域
83.
84.     print("Region of Interest")  # 标定后计算机削减后图像的边缘
85.     print(roi)
86.     np.save(savedir+'roi.npy', roi)
87.
88.     print("New Camera Matrix")  # 校正后的新相机内参矩阵，相机没有失真了就
89.     #print(newcam_mtx)
90.     np.save(savedir+'newcam_mtx.npy', newcam_mtx)
91.     print(np.load(savedir+'newcam_mtx.npy'))
92.
93.     inverse = np.linalg.inv(newcam_mtx)  # 新的相机内参矩阵的逆矩阵，在计算机视觉、
图像处理里面常常用其逆矩阵
94.     print("Inverse New Camera Matrix")
95.     print(inverse)
96.
97.
```

```
    # undistort
    undst = cv2.undistort(img1, cam_mtx, dist, None, newcam_mtx)  # 生成校正后的图像 undst
    cv2.imwrite("undistorted_image.jpg", undst)

    # crop the image
    #x, y, w, h = roi
    #dst = dst[y:y+h, x:x+w]
    #cv2.circle(dst,(308,160),5,(0,255,0),2)
    cv2.imshow('img1', img1)
    cv2.waitKey(5000)
    cv2.destroyAllWindows()
    cv2.imshow('img1', undst)
    cv2.waitKey(5000)
    cv2.destroyAllWindows()
```

参考 **Github**【代码以及一些资料】

https://github.com/pacogarcia3/hta0-horizontal-robot-arm

参考博文

[1] https://blog.csdn.net/LuohenYJ/article/details/104697062

[2] https://blog.csdn.net/qq_29931565/article/details/119395353

[3] https://blog.csdn.net/weixin_42657460/article/details/114886469

二、【基于 Ubuntu 系统 ROS 环境下使用棋盘图进行相机标定】

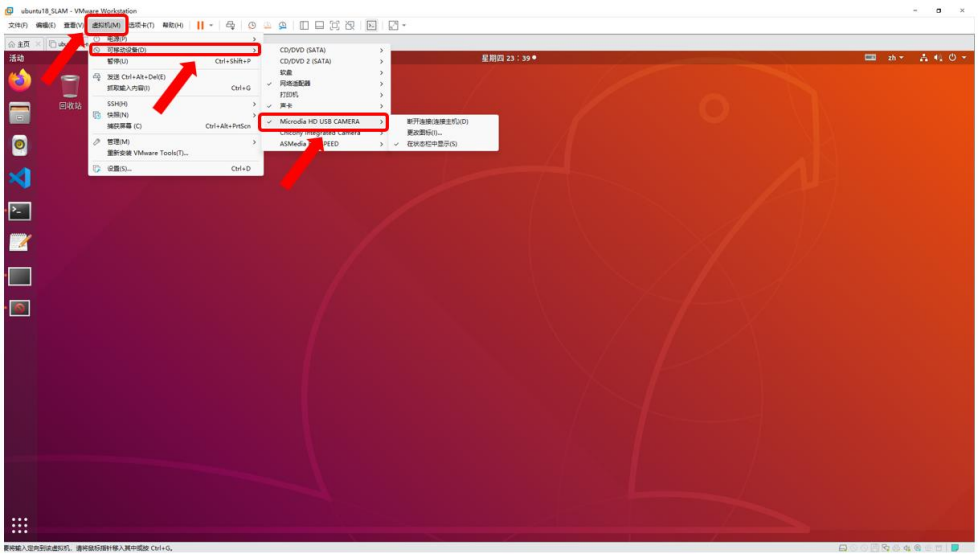
①虚拟机、Ubuntu 的准备

虚拟机使用的 VMware Workstation；Ubuntu 使用的是老师给的。

使用虚拟机搭载 Ubuntu 系统，在 Ros 系统下进行相机的标定，可以学习指令以及 ROS 操作。

②摄像头的准备

插上准备 usb 摄像头，注意要把 usb 摄像头连接到虚拟机上，可以通过虚拟机-可移动设备，检查有没有接上。



③安装相机驱动包

开启终端运行：sudo apt-get install ros-melodic-usb-cam

④启动摄像头

roslaunch usb_cam usb_cam-test.launch

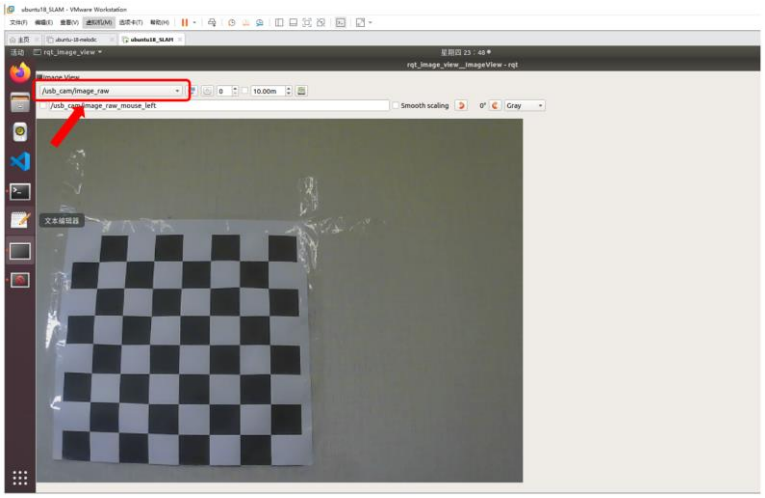
可以观察到摄像头输出图像的棋盘格有明显的畸变

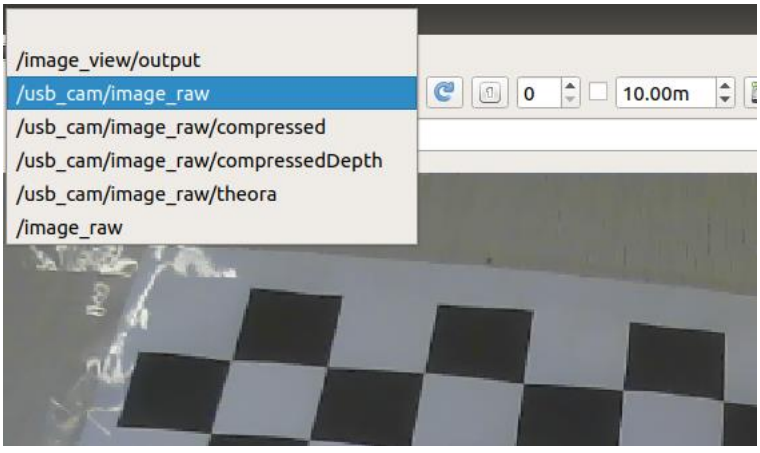
⑤显示图像

新建终端【快捷键 Ctrl+Shift+T】

运行命令：rqt_image_view

选择图像话题：raw 是原画质，compressed 是压缩画质，theora 是高质量压缩，选择原画质就行了。

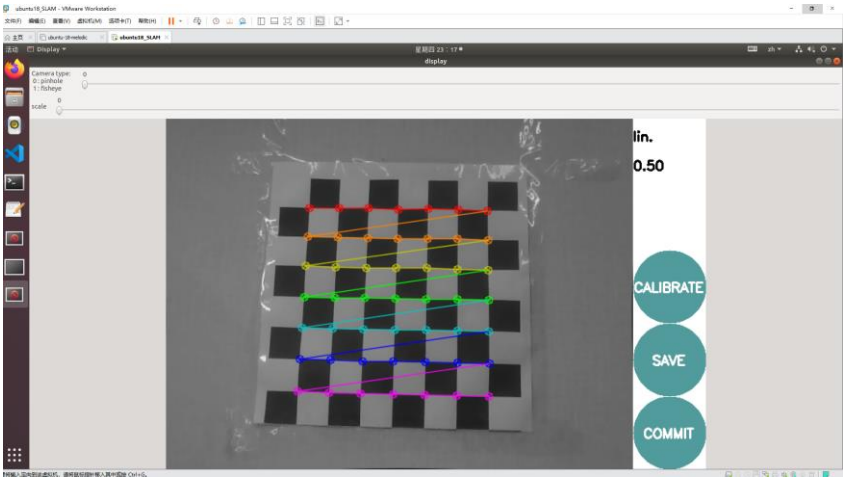




⑥启动标定节点

运行程序：rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.03 image:=/usb_cam/image_raw camera:=/usb_cam 【这里 8x6 是你的棋盘格数量，square 0.03 是你棋盘格的大小】

直到 CALIBRATE 按钮变亮，点击该按钮就可以进行标定。标定过程将持续一两分钟，并且标定界面会变成灰色，无法进行操作。



⑦参数获取与保存

点击 COMMIT 按钮将相机参数保存到默认文件夹，这样我们就成功获取到相机的内参与畸变系数。

```
[image]

width
640

height
480

[narrow_stereo]

camera matrix
780.287751 0.000000 328.343394
0.000000 799.870107 210.421971
0.000000 0.000000 1.000000

distortion
-0.382160 0.210415 0.001052 -0.001239 0.000000

rectification
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000

projection
727.420105 0.000000 328.370755 0.000000
0.000000 769.150330 208.109058 0.000000
0.000000 0.000000 1.000000 0.000000
```

三、【基于 Matlab 应用使用棋盘图进行相机标定】

①摄像头、棋盘格的准备

这进一步跟前面一样，准备摄像头以及棋盘格

②原摄像头图片获取

这里可以选择自己拍照片或者使用**一、【基于 Python 中 OpenCV 库进行摄像头的标定】**中的 camera_capture 程序进行自动拍照获取图像。

③使用 Matlab 软件进行标定与参数计算

打开 Matlab 软件，点击左上角的 APP 软件库，下拉找到 Camera Calibrator 应用，这个应用就是 Matlab 自带的相机标定程序。

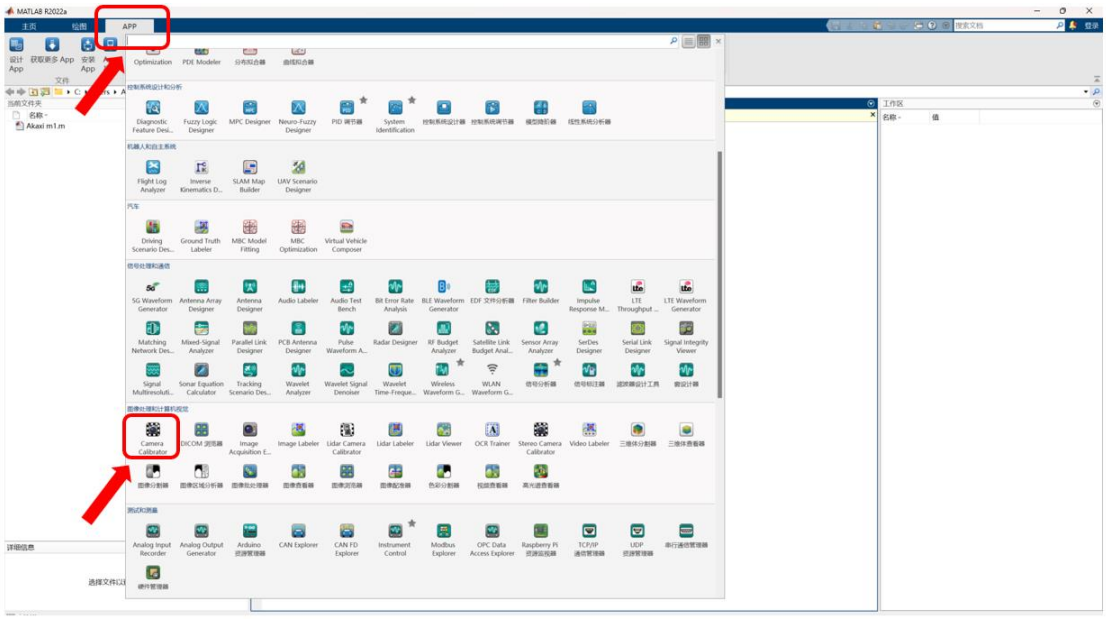


图 14 Matlab 软件相机标定应用

点击添加拍摄好的棋盘图，选择尽量多的棋盘格图片，便于精度的计算与求解，图 15。

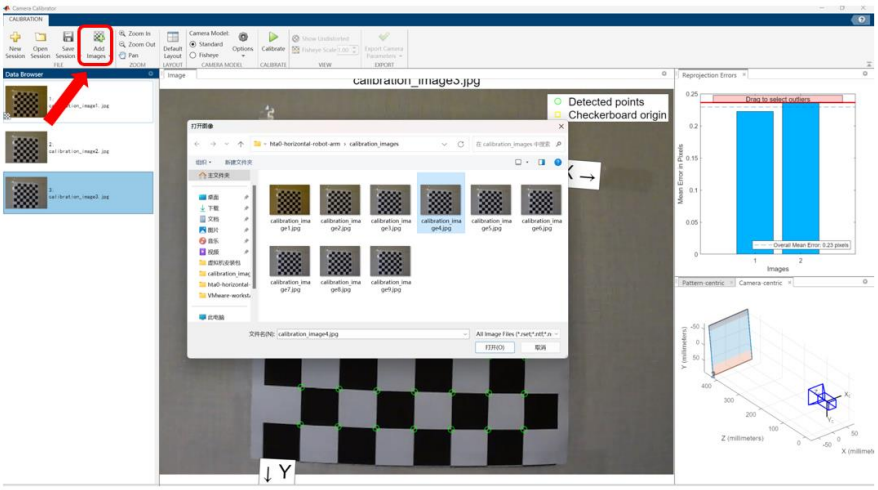


图 15 Matlab 软件相机标定添加图片

选择计算三个畸变参数，图 16。

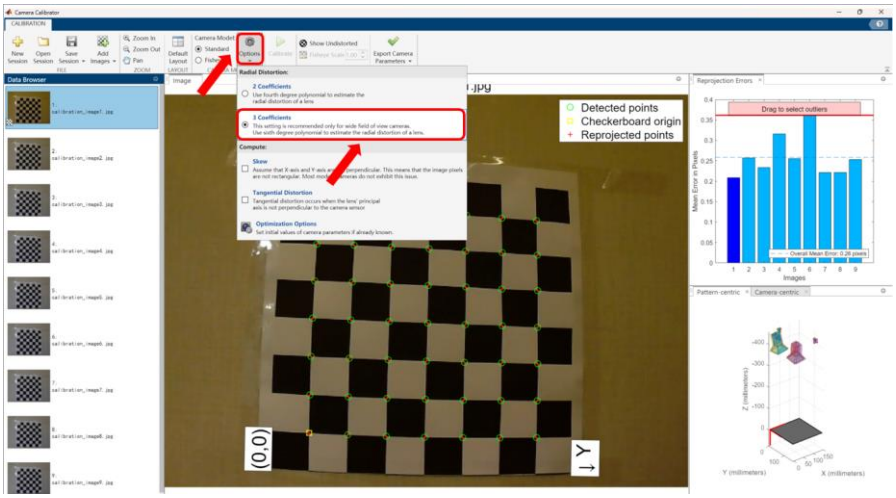


图 16 Matlab 软件相机标定选择畸变参数

点击运行，再保存相机内参矩阵、畸变系数，图 17。

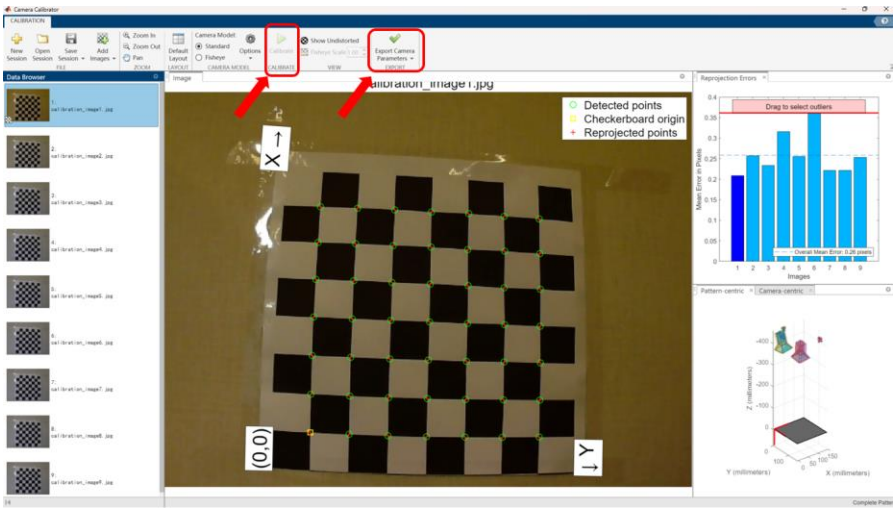


图 17 Matlab 软件相机标定计算与参数保存

之后打开 Matlab 页面就可以看到我们的相机内参结果与畸变系数

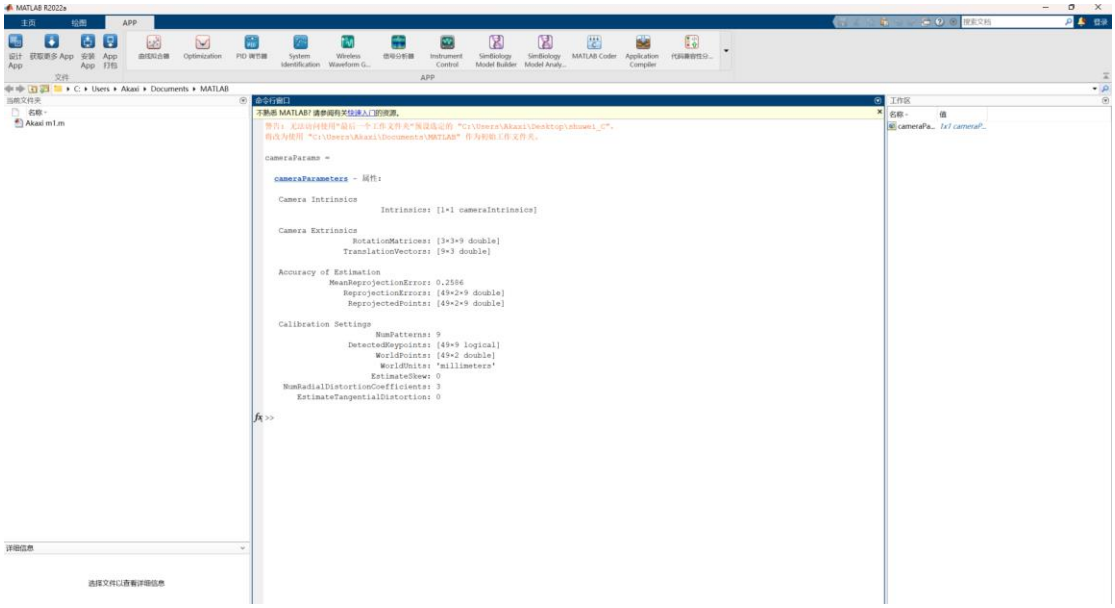


图 18 Matlab 软件相机标定结果

参考博文

[1] https://blog.csdn.net/weixin_45718019/article/details/105823053

总结

学习了相机的基本属性，图像畸变与标定，并且使用工具包利用棋盘格对针孔相机进行标定，求相机的内参矩阵，畸变系数，并且还原校正后的相机图像。学会了 1.基于 Python 中 OpenCV 库进行摄像头的标定；2. 基于 Ubuntu 系统 ROS 环境下使用棋盘图进行相机标定；3. 基于 Matlab 应用使用棋盘图进行相机标定。这在之后图像识别、机器视觉、Ros 机器人学习中有较大的帮助。

2023.9.26

渝北仙桃