

机器人视觉中的相机内参标定

程序设计思想与方法



5140309006 欧阳张健

机械与动力工程学院

2016.12.28

目录

一、相机标定简介	3
1.1 背景	3
1.2 相机标定目的	3
1.3 相机标定原理	4
1.4 张氏标定法	4
1.5 库函数 numpy 与 opencv	5
二、实现过程	7
2.1 主程序部分	7
2.2 GUI 部分	9
三、最终结果	11
3.1 chessboard 标定结果	11
3.2 得到的各项参数	12

一、相机标定简介

1.1 背景

我所在的机械与动力工程学院机器人所主要研究方向在于泛在机器人, 移动机器人及医疗机器人。其中大部分机器人的研发都离不开机器视觉。在机器视觉中最基础, 也是最不能离开的部分就是相机标定。而且我最近参加的 Amazon Robotics Challenge 目前正要做机器人的手眼标定。因此我想先在 Python 上试验一下基于 opencv 的相机内参标定。

1.2 相机标定目的

相机标定定义:

在图像测量过程以及机器视觉应用中, 为确定空间物体表面某点的三维几何位置与其在图像中对应点之间的相互关系, 必须建立相机成像的几何模型, 这些几何模型参数就是相机参数。在大多数条件下这些参数必须通过实验与计算才能得到, 这个求解参数的过程就称之为相机标定 (或摄像机标定)。无论是在图像测量或者机器视觉应用中, 相机参数的标定都是非常关键的环节, 其标定结果的精度及算法的稳定性直接影响相机工作产生结果的准确性。因此, 做好相机标定是做好后续工作的前提, 提高标定精度是科研工作的重点所在。

常用的标定方法:

常用的相机标定方法有三种: 传统相机标定法、主动视觉相机标定方法、相机自标定法。

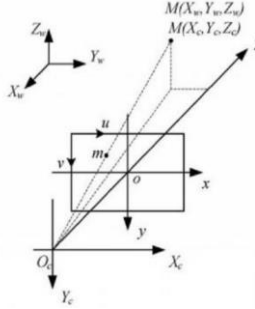
传统相机标定法需要使用尺寸已知的标定物, 通过建立标定物上坐标已知的点与其图像点之间的对应, 利用一定的算法获得相机模型的内外参数。根据标定物的不同可分为三维标定物和平面型标定物。三维标定物可由单幅图像进行标定, 标定精度较高, 但高精密三维标定物的加工和维护较困难。平面型标定物比三维标定物制作简单, 精度易保证, 但标定时必须采用两幅或两幅以上的图像。传统相机标定法在标定过程中始终需要标定物, 且标定物的制作精度会影响标定结果。同时有些场合不适合放置标定物也限制了传统相机标定法的应用。

目前出现的自标定算法中主要是利用相机运动的约束。相机的运动约束条件太强, 因此使得其在实际中并不实用。利用场景约束主要是利用场景中的一些平行或者正交的信息。其中空间平行线在相机图像平面上的交点被称为消失点, 它是射影几何中一个非常重要的特征, 所以很多学者研究了基于消失点的相机自标定方法。自标定方法灵活性强, 可对相机进行在线定标。但由于它是基于绝对二次曲线或曲面的方法, 其算法鲁棒性差。

基于主动视觉的相机标定法是指已知相机的某些运动信息对相机进行标定。该方法不需要标定物, 但需要控制相机做某些特殊运动, 利用这种运动的特殊性可以计算出相机内部参数。基于主动视觉的相机标定法的优点是算法简单, 往往能够获得线性解, 故鲁棒性较高, 缺点是系统的成本高、实验设备昂贵、实验条件要求高, 而且不适合于运动参数未知或无法控制的场合。

1.3 相机标定原理

相机标定实质上就是求解相机坐标系与世界系的转换关系，矩阵方程表达出来关系为



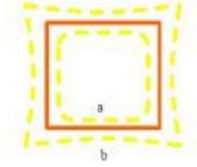
$$Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} {}^wX \\ {}^wY \\ {}^wZ \\ 1 \end{bmatrix} =$$

$$\mathbf{M}_1 \mathbf{M}_2 \mathbf{X}_w = \mathbf{M} \mathbf{X}_w$$

其中 \mathbf{M}_1 代表相机内参， \mathbf{M}_2 代表相机外参。并且传统的相机都是非线性的，在成像边缘区域非线性度更高，因此标定的过程中还需要添加校正系数来校正相机的成像畸变。非线性度可以用下式表示：

$$\bar{x} = x + \delta_x(x, y)$$

$$\bar{y} = y + \delta_y(x, y)$$



(\bar{x}, \bar{y}) 代表线性模型下计算出来的相机上的点， δ_x, δ_y 代表非线性畸变量， (x, y) 代表实际图像上的点。

理论上相机标定只需要进行简单的矩阵计算就可以解决，但是由于畸变量的存在导致标定参数不准，因此在标定过程的难点在于畸变量的确定。

1.4 张氏标定法

本文所采用的方法为张氏标定法。张氏标定法是介于传统标定和自标定之间的一种方法，它只需要摄像机对某个标定板从不同方向拍摄多幅图片，通过标定板上每个特征点和其像平面的像点间的对应关系，即每一幅图像的单应矩阵来进行摄像机的标定，由于该方法模板制作容易，使用方便，成本低，鲁棒性好，准确率高，因此得到了较为广泛的应用。该算法也属于两步法，摄像机和模板可以自由的移动，不需要知道运动参数。

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = A \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

现在世界坐标系平面置于标定模板所在的平面，即 $O_w WZ$ 。则上式可变为如下形式

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{bmatrix} = A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

$$H = A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$$

最后得到摄像机内部参数求解方程

$$B = A^{-T} A^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} = \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2 \beta} & \frac{v_0 \gamma - u_0 \beta}{\alpha^2 \beta} \\ -\frac{\gamma}{\alpha^2 \beta} & \frac{\gamma^2}{\alpha^2 \beta^2} + \frac{1}{\beta^2} & -\frac{\gamma(v_0 \gamma - u_0 \beta)}{\alpha^2 \beta^2} - \frac{v_0}{\beta^2} \\ \frac{v_0 \gamma - u_0 \beta}{\alpha^2 \beta} & -\frac{\gamma(v_0 \gamma - u_0 \beta)}{\alpha^2 \beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0 \gamma - u_0 \beta)^2}{\alpha^2 \beta^2} + \frac{v_0}{\beta^2} + 1 \end{bmatrix}$$

如果有 N 幅图像，则可以得到 5 个参数

$$\begin{cases} v_0 = (B_{12}B_{13} - B_{11}B_{23}) / (B_{11}B_{22} - B_{12}^2) \\ \lambda = B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})] / B_{11} \\ f_u = \sqrt{\lambda / B_{11}} \\ f_v = \sqrt{\lambda B_{11} / (B_{11}B_{22} - B_{12}^2)} \\ s = -B_{12}f_u^2 f_v / \lambda \\ u_0 = sv_0 / f_v - B_{13}f_u^2 / \lambda \end{cases}$$

随后通过最大似然估计法消除噪声。

对于镜头畸变，张氏标定法只考虑了镜头的一阶和二阶径向畸变，且假设摄像机镜头在 x 轴方向和 y 轴方向的畸变系数相同。设径向畸变的畸变模型

$$\begin{cases} \hat{x} = x + x[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \\ \hat{y} = y + y[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \end{cases}$$

当有 n 幅图像时，可以得到

$$\begin{bmatrix} (u^1 - u_0^1)[(x^1)^2 + (y^1)^2] & (u^1 - u_0^1)((x^1)^2 + (y^1)^2)^2 \\ (v^1 - v_0^1)[(x^1)^2 + (y^1)^2] & (v^1 - v_0^1)((x^1)^2 + (y^1)^2)^2 \\ \vdots & \vdots \\ (u^n - u_0^n)[(x^n)^2 + (y^n)^2] & (u^n - u_0^n)((x^n)^2 + (y^n)^2)^2 \\ (v^n - v_0^n)[(x^n)^2 + (y^n)^2] & (v^n - v_0^n)((x^n)^2 + (y^n)^2)^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} \hat{=} \begin{bmatrix} \hat{u}^1 - u^1 \\ \hat{v}^1 - v^1 \\ \vdots \\ \hat{u}^n - u^n \\ \hat{v}^n - v^n \end{bmatrix}$$

随后可以通过最小二乘法得到径向畸变系数

$$k = (D^T D)^{-1} D^T d$$

当摄像机的一级径向畸变系数 $1k$ 和二级径向畸变系数 $2k$ 求出以后,就可以用这一组系数来校正,从式(2-34)和式(2-35)中求出的各内参数和外参数,校正过程是一个非线性优化过程,可以用它们来重新计算 $m(A,R,T,M)$ 的值,然后重新应用最大似然估计,并利用 Levenber-Marquarat 算法迭代进行最小化处理,进一步优化所有的参数。这样,在经过非线性校正及优化的各个参数中,径向畸变的现象将大为削弱。

1.5 库函数 numpy 和 opencv

NumPy 系统是 Python 的一种开源的数值计算扩展。这种工具可用来存储和处理大型矩阵,比 Python 自身的嵌套列表(nested list structure)结构要高效的多(该结构也可以用来表示矩阵(matrix))。据说 NumPy 将 Python 相当于变成一种免费的更强大的 MatLab 系统。一个用 python 实现的科学计算包。包括: 1、一个强大的 N 维数组对象 Array; 2、比较成熟的(广播)函数库; 3、用于整合 C/C++ 和 Fortran 代码的工具包; 4、实用的线性代数、傅里叶变换和随机数生成函数。numpy 和稀疏矩阵运算包 scipy 配合使用更加方便。

NumPy (Numeric Python) 提供了许多高级的数值编程工具,如: 矩阵数据类型、矢量处理,以及精密的运算库。专为进行严格的数字处理而产生。多为很多大型金融公司使用,以及核心的科学计算组织如: Lawrence Livermore, NASA 用其处理一些本来使用 C++, Fortran 或 Matlab 等所做的任务。

OpenCV 的全称是: Open Source Computer Vision Library。OpenCV 是一个基于 BSD 许可(开源)发行的跨平台计算机视觉库,可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成,同时提供了 Python、Ruby、MATLAB 等语言的接口,实现了图像处理 and 计算机视觉方面的很多通用算法。

OpenCV 用 C++ 语言编写,它的主要接口也是 C++ 语言,但是依然保留了大量的 C 语言接口。该库也有大量的 Python, Java and MATLAB/OCTAVE (版本 2.5) 的接口。这些语言的 API 接口函数可以通过在线文档获得。如今也提供对于 C#, Ch, Ruby 的支持。

在 numpy 和 opencv 的基础上我们可以实现 chessboard 标定板的角点提取以及矩阵运算,从而完成转换矩阵和畸变参数的求解。

二、实现过程

在进行实际标定之前需要很多的前期准备工作。

前期工作：

1. 安装 OpenCV。Windows 环境下 OpenCV 的安装过程中牵扯到环境变量的设置，python 库文件导入问题等。
2. 安装 numpy。Numpy 作为一款用来进行矩阵变换运算的工具在三维坐标系变换过程中求解必不可少。Windows 环境下 numpy 需导入 python 函数库。
3. 标定板的制作与拍摄。因为标定板的制作精度在很大程度上影响了标定精度，本次实验采用了 9X9chessboard 标定板。

2.1 主程序部分

```
3  from common import splitfn

# built-in modules
import os

if __name__ == '__main__':
    import sys
    import getopt
    from glob import glob

    args, img_mask = getopt.getopt(sys.argv[1:], '', ['debug=', 'square_size='])
    args = dict(args)
    args.setdefault('--debug', './output/')
    args.setdefault('--square_size', 1.0)
    if not img_mask:
        img_mask = '../data/cc*.jpg' # default
    else:
        img_mask = img_mask[0]

    img_names = glob(img_mask)
    debug_dir = args.get('--debug')
    if not os.path.isdir(debug_dir):
        os.mkdir(debug_dir)
    square_size = float(args.get('--square_size'))

    pattern_size = (9, 9)
    pattern_points = np.zeros((np.prod(pattern_size), 3), np.float32)
    pattern_points[:, :2] = np.indices(pattern_size).T.reshape(-1, 2)
    pattern_points *= square_size
```

```

obj_points = []
img_points = []
h, w = 0, 0
img_names_undistort = []
for fn in img_names:
    print('processing %s...' % fn, end='')
    img = cv2.imread(fn, 0)
    if img is None:
        print("Failed to load", fn)
        continue

    h, w = img.shape[:2]
    found, corners = cv2.findChessboardCorners(img, pattern_size)
    if found:
        term = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_COUNT, 30, 0.1)
        cv2.cornerSubPix(img, corners, (5, 5), (-1, -1), term)

        if debug_dir:
            vis = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
            cv2.drawChessboardCorners(vis, pattern_size, corners, found)
            path, name, ext = splitfn(fn)
            outfile = debug_dir + name + '_chess.png'
            cv2.imwrite(outfile, vis)
            if found:
                img_names_undistort.append(outfile)

    if not found:
        print('chessboard not found')
        continue

    img_points.append(corners.reshape(-1, 2))
    obj_points.append(pattern_points)

print('ok')

# calculate camera distortion
rms, camera_matrix, dist_coefs, rvecs, tvecs = cv2.calibrateCamera(obj_points,
img_points, (w, h), None, None)

print("\nRMS:", rms)
print("camera matrix:\n", camera_matrix)
print("distortion coefficients: ", dist_coefs.ravel())

```



```

# undistort the image with the calibration
print('')
for img_found in img_names_undistort:
    img = cv2.imread(img_found)

    h, w = img.shape[:2]
    newcameramtx, roi = cv2.getOptimalNewCameraMatrix(camera_matrix, dist_coefs, (w,
h), 1, (w, h))

    dst = cv2.undistort(img, camera_matrix, dist_coefs, None, newcameramtx)

    # crop and save the image
    x, y, w, h = roi
    dst = dst[y:y+h, x:x+w]
    outfile = img_found + '_undistorted.png'
    print('Undistorted image written to: %s' % outfile)
    cv2.imwrite(outfile, dst)

cv2.destroyAllWindows()

```

3.2 GUI 部分

```

4. from Tkinter import *
from tkFileDialog import askdirectory

def selectPath(): #select the photos' directory
    path_ = askdirectory()
    path.set(path_)

def helpwindows():
    root = Tk()
    Message(root, text = "what is the camera calibration?\n      Camera Calibration In
the process of image measurement and application of machine vision, in order to
determine the relationship between the 3D geometric position of a certain point at
spatial object surface and the corresponding point in the image, we must establish the
geometric model of camera imaging, these geometric parameters of the model is the
parameters of the camera .Under most conditions, these parameters must be obtained
through experiment and calculation. This process is called camera calibration.").grid()
    root.mainloop()

root = Tk()
path = StringVar()

```

```

root.title("Camera Callibration")
m = Menu(root) #menu part
root.config(menu = m)
filemenu = Menu(m)
m.add_cascade(label="file", menu=filemenu)
filemenu.add_command(label="open...", command = selectPath)
filemenu.add_separator()
filemenu.add_command(label = "exit", command = quit)
helpmenu = Menu(m)
m.add_cascade(label = "help", menu = helpmenu)
helpmenu.add_command(label="about...", command = helpwindows)

# local modules
5. Label(root, text = "file's directory").grid(row = 1, column = 1)
Entry(root, textvariable = path).grid(row = 1, column = 0, sticky=N+S+W+E)
Label(root, text="This program is used to camera calibiration and calculate the nonlinear
parameters.", fg="blue").grid(row=0, column=0, columnspan=2)
Label(root, text = "RMS", fg = "red").grid(row = 2, column = 0, sticky = N+W)
Label(root, text = rms).grid(row = 3, column = 0, sticky = N+W)
Label(root, text = "camera metrix", fg = "red").grid(row = 2, column = 1, sticky = N+W)
Label(root, text = camera_matrix).grid(row = 3, column = 1, sticky = N+W)
Label(root, text = "distortion coefficients", fg="red").grid(row = 4, column=0, sticky =
N+W)
Label(root, text = dist_coefs.ravel()).grid(row = 5, column = 0, sticky =N+W)

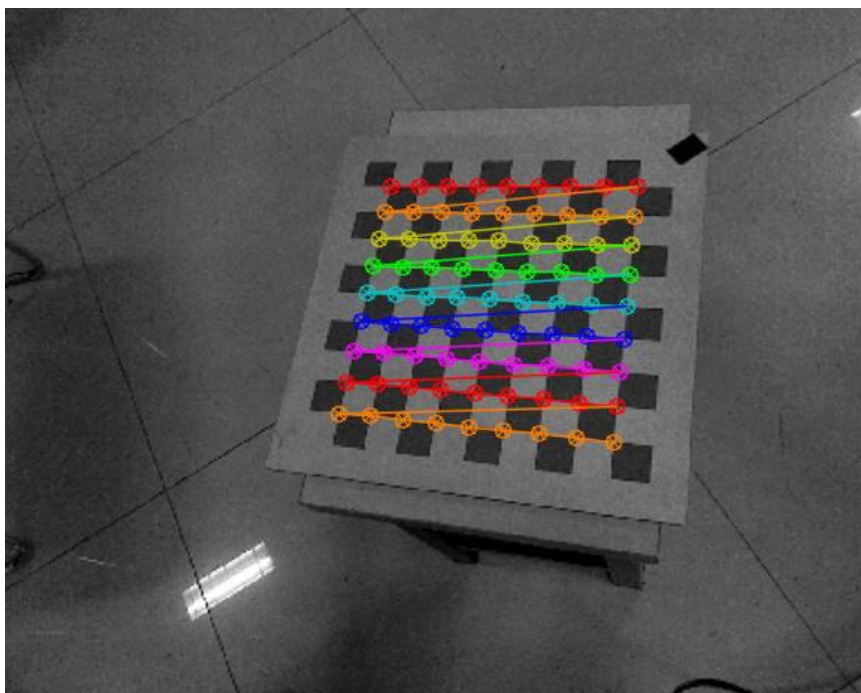
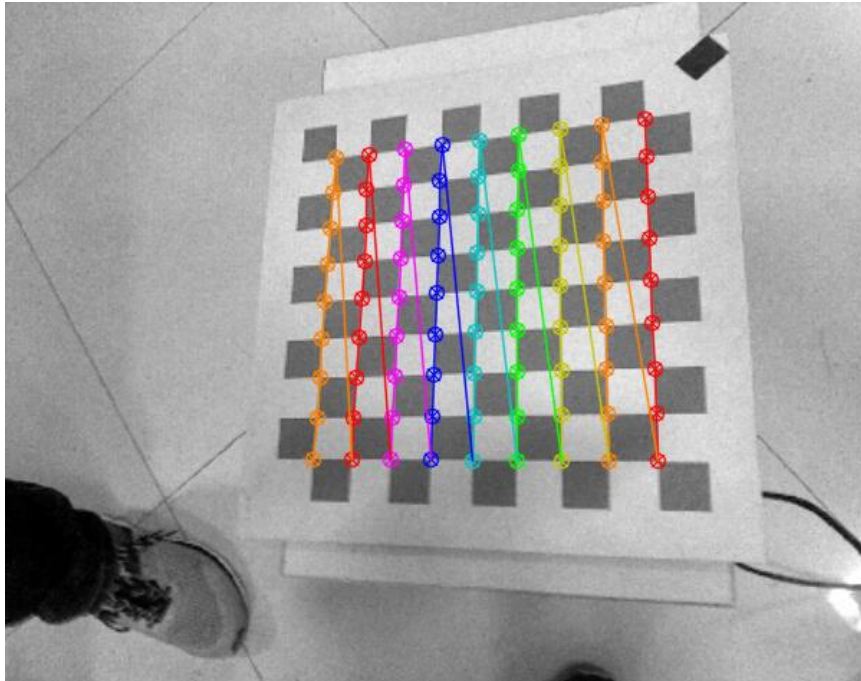
root.mainloop()

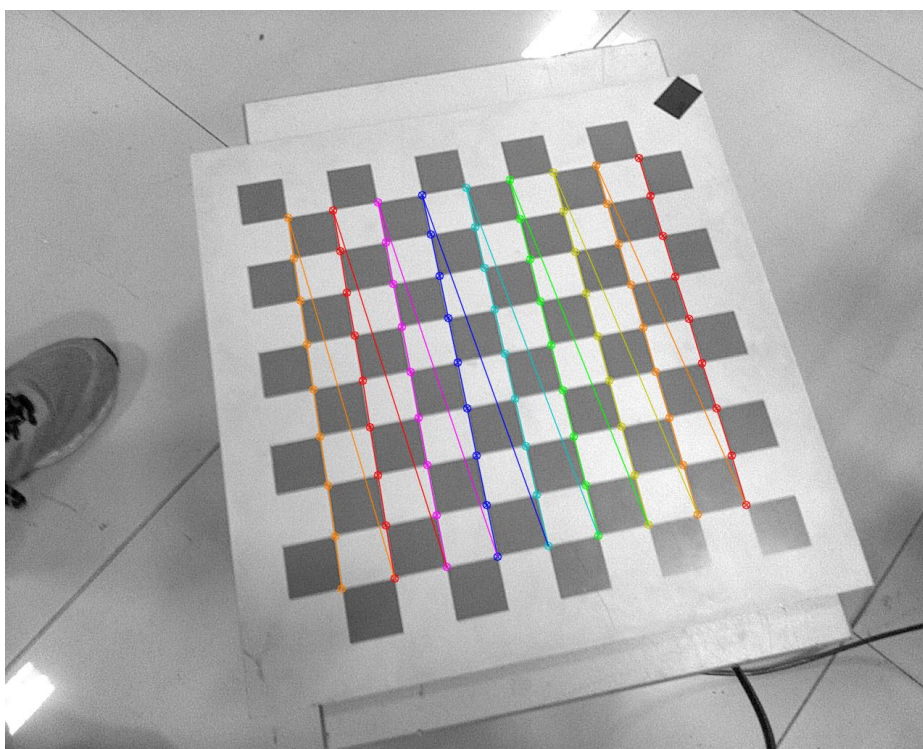
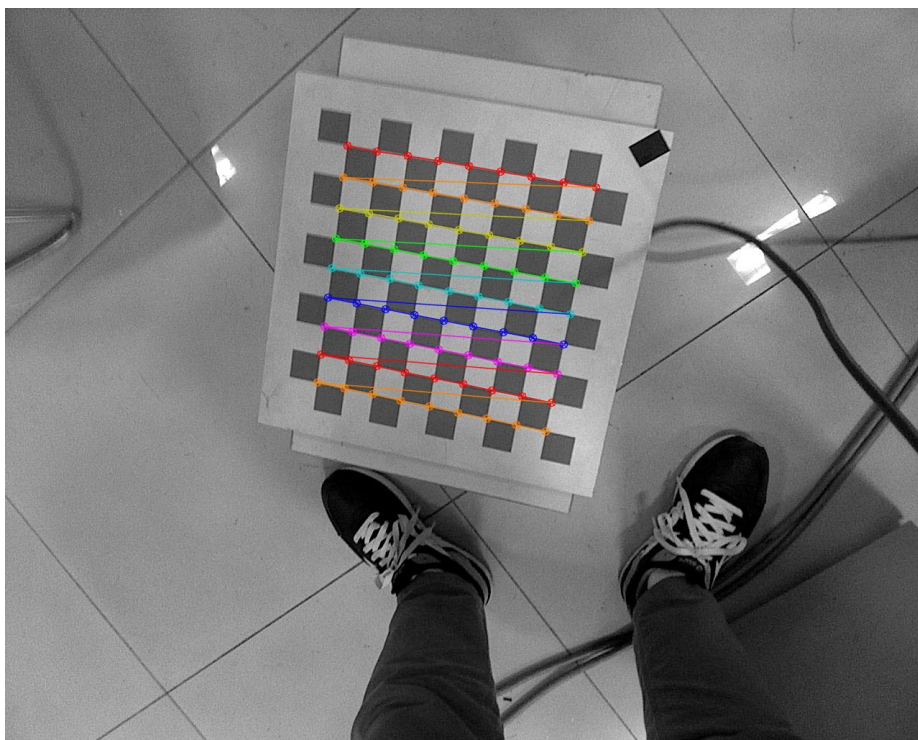
```

三、最终结果

3.1 chessboard 标定结果

(部分结果)





3.2 各项参数

程序运行结果如下图所示

```
76 Camera Callibration
file help
This program is used to camera calibration and calculate the nonlinear parameters.
personal files\programming\Python\camera_calibration\chessboard\before import file's directory
C:\personal files\programming\Python\camera_calibration\chessboard\ output file's directory
RMS camera matrix
1.1601077316905777 [[ 1.07303977e+03 0.00000000e+00 6.31820737e+02]
[ 0.00000000e+00 1.06798875e+03 4.80741494e+02]
[ 0.00000000e+00 0.00000000e+00 1.00000000e+00]]
distortion coefficients
[ 1.57373925e-01 -9.13521853e-01 -4.84016449e-04 -1.02643876e-03
-1.23760333e+00]
```

RMS 的值为 1.16010773169

坐标系转化矩阵为

```
[[ 1.07303977e+03 0.00000000e+00 6.31820737e+02]
 [ 0.00000000e+00 1.06798875e+03 4.80741494e+02]
 [ 0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

畸变校正系数为

```
[ 1.57373925e-01 -9.13521853e-01 -4.84016449e-04 -1.02643876e-03
-1.23760333e+00]
```