2022赛季视觉部第二次培训——相机成像原理与相机标定

机械是肉体， 电控是大脑， 视觉是灵魂

摄相机外层结构介绍

摄像机有两部分组成；

- 相机
- 镜头

镜头

镜头是摄像机中的光学元件，直接影响成像质量的优劣。
镜头的分类主要有一下三种：

1. 按照焦距分类
    - 短焦
    - 中焦
    - 长焦

    焦距决定了相机适合观察什么距离的物体，短焦一般适合观察近距离物体，长焦一般适合观察远距离物体。
2. 按照视角大小分类
    - 广角
      特点：视角大，可观测范围广。**但同时会产生较大畸变。**
    - 标准
      特点：视角小，但产生的畸变也较小。

一般来说，在没有特殊需求的情况下，镜头选型选择标准镜头。

3. 光圈焦距调整方式
    - 固定光圈定焦镜头
    - 手动光圈定焦镜头
    - 自动光圈定焦镜头
    - 手动变焦镜头
    - 自动变焦镜头
    - ……

相机

相机是整个摄像机结构的核心部分，也是感光元件的所在部分。他直接决定了摄像机的许多重要特性。
相机选型时主要关注以下的参数：

- 采样分辨率
    - 即相机采样得到的图像的像素点个数
    - 例如我们的相机最大分辨率为1280×1024
- 采样最大帧率
    - 即相机一秒最多能拍摄多少张照片，取决于相机采集，传输图像的速率。
    - 一般来说，相机会因为曝光时间限制无法达到最大帧率。
- 像素深度
    - 即每位像素数据的位数，常见的为$8bit$。
- 触发方式
    - 即使用什么方式触发相机拍摄一张图片。
    - 常见的触发方式有连续触发，软件触发和硬件触发。
- 接口类型
    - 即相机输出图片的接口类型。
    - 常见的有`USB接口`，`以太网接口`等。**队内现役的相机基本为USB3.0接口**
- 曝光方式
    - 线阵相机

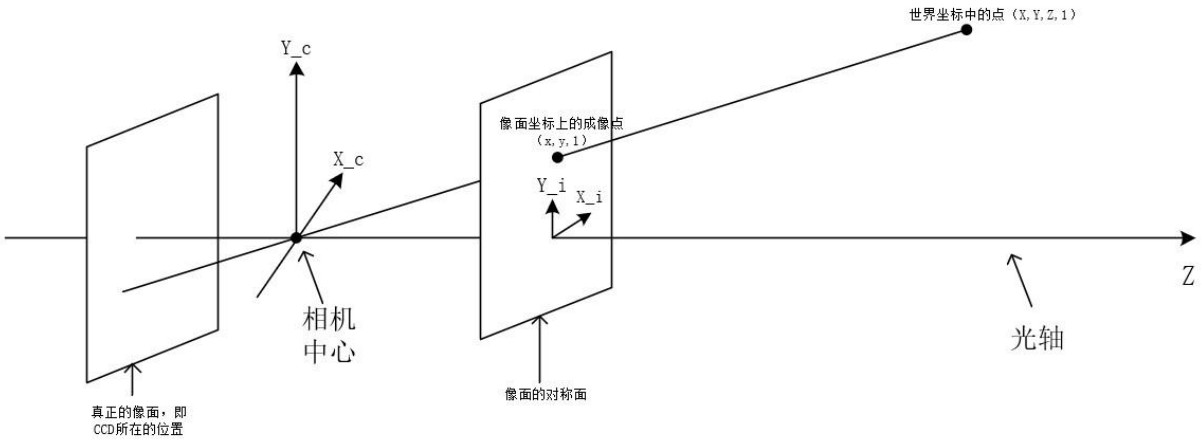
        - 使用线阵图像传感器
        - 一行的数据可以到几K甚至几十K，但是高度只有几个像素，行频很高，可以到每秒几万行，适合做非常高精度、宽画幅的扫描。
    - 面阵相机
        - 使用面阵传感器
        - 一次可以获取整张图像，但一行的分辨率宽度比线阵相机小很多。

线阵相机一般比面阵相机贵很多，现在主要使用的仍为面阵相机。

队内使用的摄像机外层结构介绍

队内现在主要使用的相机型号为`MINDVISION公司`生产的`MV-SUA133GC`。
下面为组装好的摄像机图片。

相机成像模型



相机成像的过程即为将一个三维空间中的点投影到二维空间的过程。
如果用矩阵的形式表示这一过程，那么它可以写成：

$$\frac{1}{Z_w}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

其中X_c, Y_c, Z_c表示点在三维空间内的位置，而x, y表示点在经过相机成像后在二维空间内的位置，而f_x, f_y分别为相机在x轴和y轴上的焦距。

你一定对为什么上式中二维空间中的点被写成了三维坐标的形式，同时三维空间中的点被写成了四维空间的形式表示疑惑。这种表示方法被称为坐标的齐次形式。

齐次表示方法最大的优势在于它保留了常数项，因此可以通过矩阵运算进行点的平移操作。

相机成像的过程其实也是一个点在不同坐标系中转换的过程。在这一过程中，`相机坐标系`中的一个点经过投影变换转换到了`像素坐标系`。
下面我们将介绍在计算机视觉中常用的几个坐标系以及他们之间的关系。

四大坐标系

在相机陀螺仪系统中进行运算时，往往离不开四个坐标系：`世界坐标系`，`陀螺仪参考坐标系`，`相机坐标系`，`像素坐标系`。
由于本讲并不重点涉及陀螺仪，因此有关`陀螺仪参考坐标系`的内容将主要在下一讲中介绍，本讲仅仅做出最简单的介绍。

像素坐标系

即一张图像中描述像素的坐标系。是一个二维空间中的坐标系。

在`OpenCV`中向右为x轴正方向，向下为y轴正方向。

在实际使用中，我们常用齐次形式描述像素坐标系，即一个像素坐标系中的点(x, y)常常被表示为$\begin{bmatrix} x & y & 1 \end{bmatrix}^T$,在这种方法的表示下，你也可以认为像素坐标系位于相机坐标系中平面$z = 1$上。
我们也称这个平面为`归一化平面`。

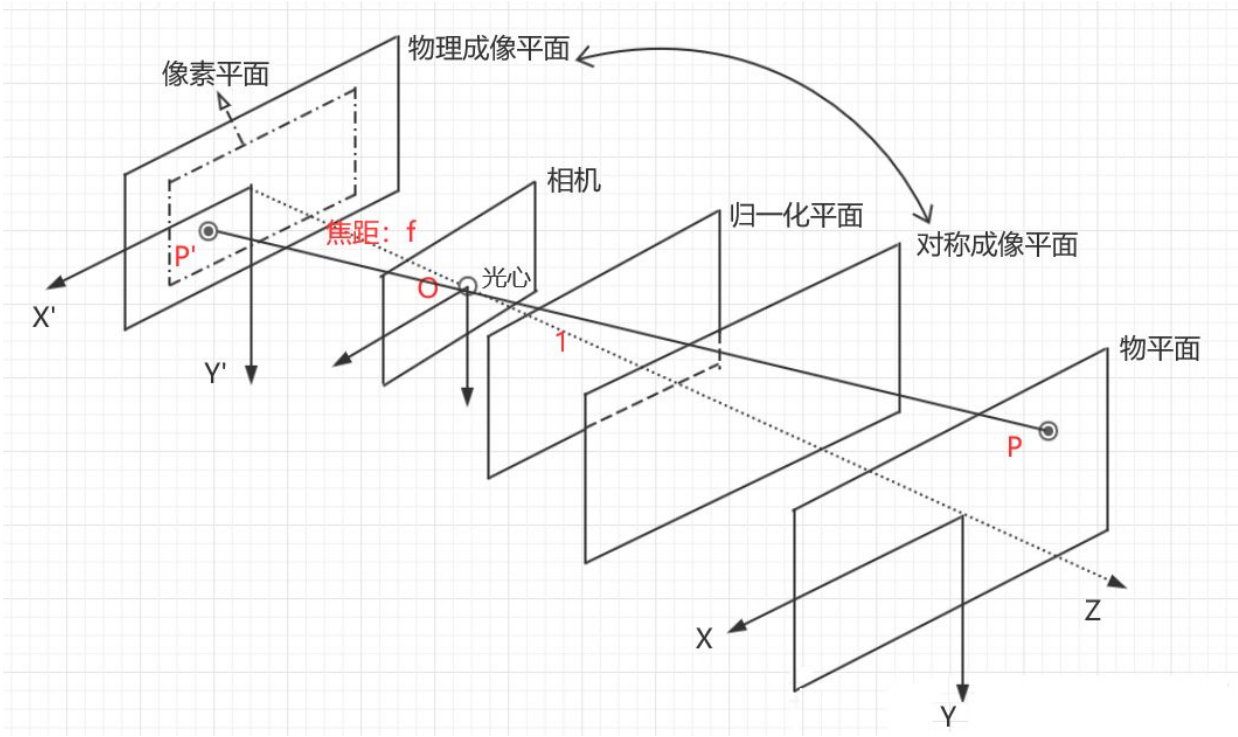
相机坐标系

以相机为原点，z轴朝前的右手系。在相机发生平移和旋转运动时，相机坐标系和相机一起运动。

必须说明的是，相机坐标系一般为右手系，但其各个坐标轴的朝向可以由使用者自行定义，需要注意与像素坐标系保持一致。（这里由于像素坐标系y轴朝下，故相机坐标系中y轴也朝下）

摄像机的成像公式中,$\begin{bmatrix} X_c & Y_c & Z_c \end{bmatrix}^T$即为相机坐标系中的点。

相机的成像过程即为将相机坐标系中的点投影到像素坐标系的过程。

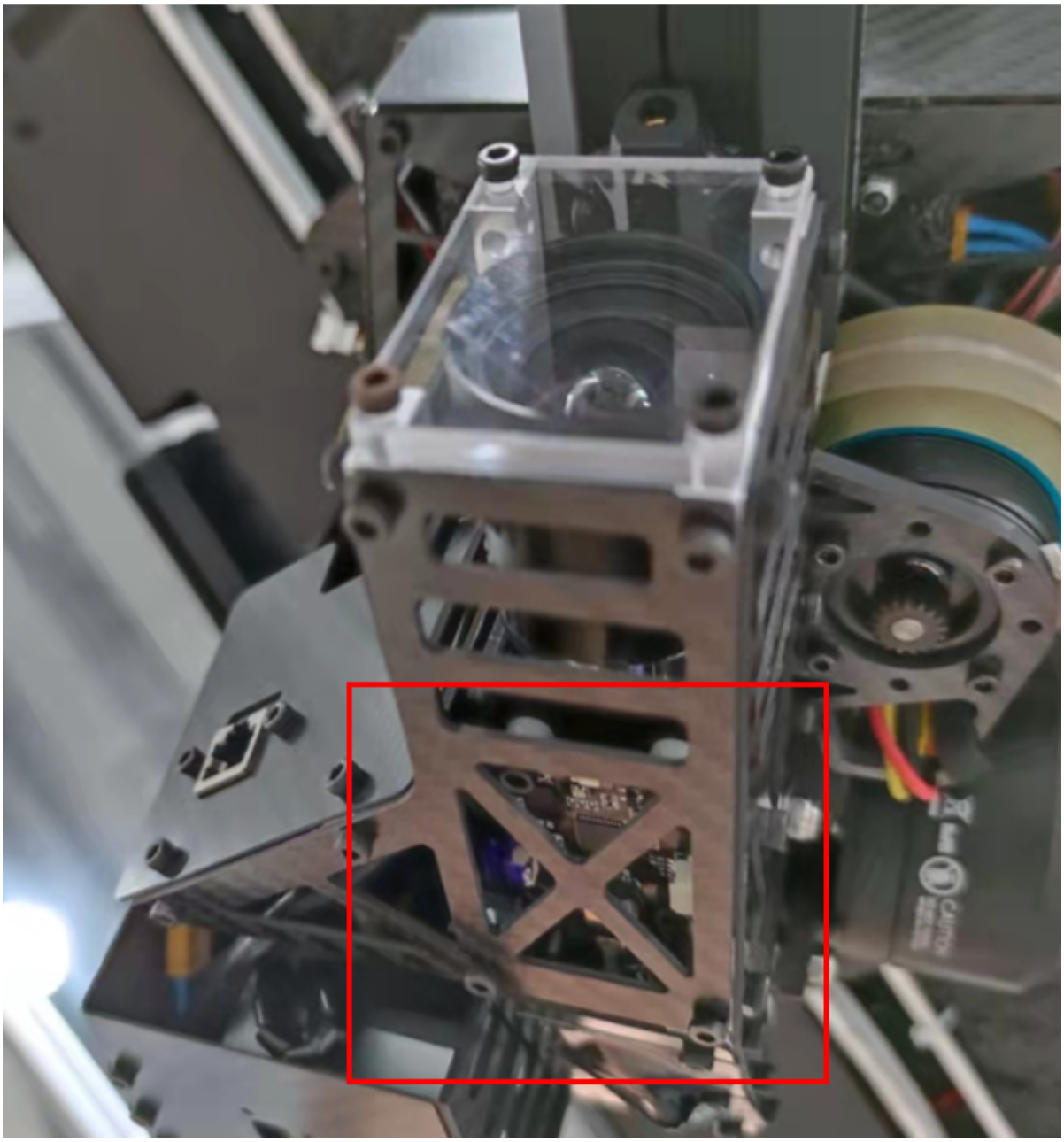


陀螺仪参考坐标系

在相机坐标系中有提到，当相机发生旋转运动时，相机坐标系也会随之一起运动。因此，当相机在发生旋转运动时，想要知道物体和相机的相对位置关系变化将会变得更困难。
对此，我们想要找到一个坐标系，在相机旋转时保持不变，这就是 陀螺仪参考坐标系 。陀螺仪参考系通过固定在相机上的陀螺仪，实时结算相机的位姿，进而得到一个不随相机旋转的坐标系。

由于陀螺仪坐标系和相机坐标系之间的关系为旋转关系，因此陀螺仪参考坐标系中的三个坐标轴的关系与相机坐标系相同。

有关 陀螺仪参考坐标系 的内容将在之后介绍，这里你只需要知道陀螺仪参考坐标系是一个**不随相机旋转，但随相机位移**的坐标系即可。



世界坐标系

即使是陀螺仪参考坐标系也会在相机运动是发生平移运动，而世界坐标系是一个与相机没有任何直接关联，建立在外部世界中的一个**被认为静止的**参考系。

事实上，许多时候视觉定位就是在解算相机坐标系与世界坐标系的相对位置关系。他们两者之间经过了一次平移和一次旋转变换。如果写成矩阵的形式，其过程如下：

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \times \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

世界坐标系的三个坐标轴关系需要与相机坐标系保持一致（同为左手系或者同为右手系），但是世界坐标系的建立可以依据实际问题选择最方便研究的方式建立。

总结一下四个坐标系之间的关系

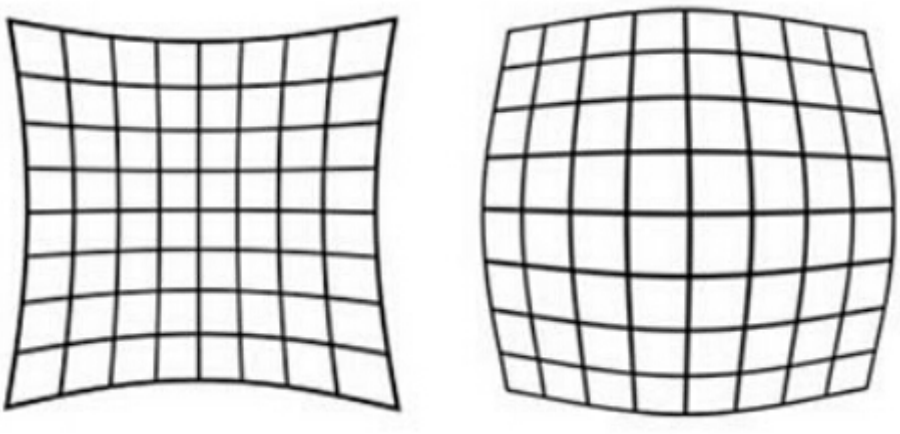
世界坐标系 – [平移] –> **陀螺仪坐标系** – [旋转] –> **相机坐标系** – [投影] –> **像素坐标系**

$$Z_c\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}\frac{1}{dx}&0&u_0\\0&\frac{1}{dy}&v_0\\0&0&1\end{bmatrix}\begin{bmatrix}f&0&0&0\\0&f&0&0\\0&0&1&0\end{bmatrix}\begin{bmatrix}R&T\\\vec{0}&1\end{bmatrix}\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}=\underbrace{\begin{bmatrix}f_x&0&u_0&0\\0&f_y&v_0&0\\0&0&1&0\end{bmatrix}}_{\text{相机内参}}\underbrace{\begin{bmatrix}R&T\\\vec{0}&1\end{bmatrix}}_{\text{相机外参}}\begin{bmatrix}X_w\\Y_w\\Z_w\\1\end{bmatrix}$$

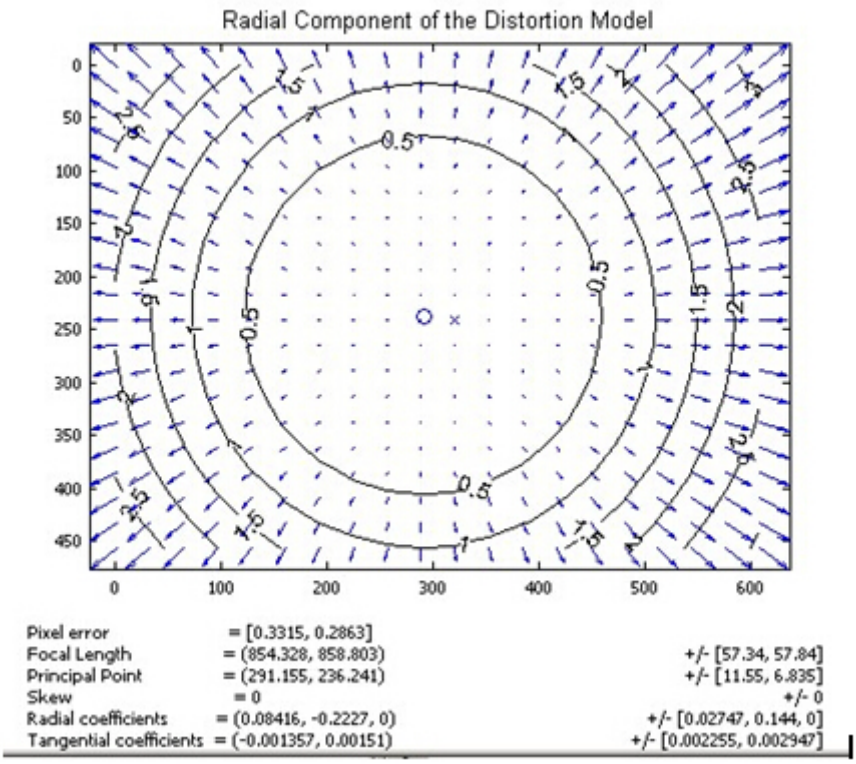
相机畸变的产生

相机畸变主要有两种：

- 径向畸变
  - 径向畸变是沿半径方向分部的畸变
  - 其产生原因是光线在远离透镜中心的地方比靠近中心的地方更加弯曲
  - 径向畸变导致的图像会产生如下图的扭曲

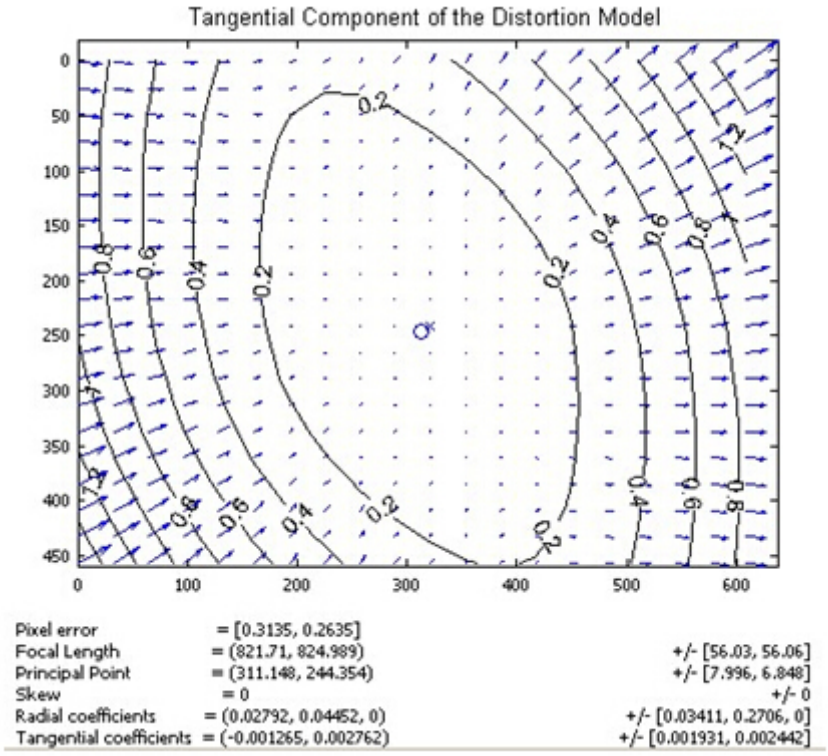


  - 其示意图如图



- 切向畸变
  - 切向畸变是大体上是关于一条中心轴对称的

- 其产生原因是由于透镜本身与传感器平面不平行导致的
- 其示意图如图



相机畸变的矫正

径向畸变的数学模型

径向畸变一般使用三项泰勒级数展开式描述：

$$x_0 = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_0 = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

事实上，对于一般的相机，只需要使用前两项就足够了。对于部分畸变极其严重的鱼眼相机，需要使用最后一项进行修正。

切向畸变的数学模型

$$x_0 = x + [2p_1 y + p_2(r^2 + 2x^2)]$$

$$y_0 = y + [2p_2 x + p_1(r^2 + 2y^2)]$$

在上述的径向畸变和切向畸变的数学模型中，我们一共使用了 5 个参数描述畸变。它们分别是 $[k_1, k_2, k_3, p_1, p_2]$。它们被称为畸变参数。

因此，畸变校正的过程就变成了如何求解畸变参数的过程。

一般来说，我们使用标定板辅助进行相机标定。

相机标定的程序实现

寻找标定板角点

`OpenCV`提供了寻找标定板棋盘格角点的函数`findChessboardCorners()`，他的声明如下：
`bool cv::findChessboardCorners(InputArray image, Size patternSize, OutputArray corners, int flags = CALIB_CB_ADAPTIVE_THRESH+CALIB_CB_NORMALIZE_IMAGE)`
你只需在意前三个参数，他们分别意为：

- image
  输入的图像
- patternSize
  棋盘格的大小，例如这一棋盘格的大小为`(11, 8)`，要注意的是棋盘格的大小只考虑内侧的角点数
- corners
  输出结果，用向量的形式储存输出的角点

对找到的角点亚像素精化

`OpenCV`提供了函数`find4QuadCornerSubpix()`来实现对棋盘格角点的亚像素精化，他的声明如下：
`bool cv::find4QuadCornerSubpix(InputArray img, InputOutputArray corners, Size region_size)`。
它的功能为在给定的点的周围一定范围内以亚像素的精度逼近角点。
下面说明部分参数意义：

- corners
  - 需要逼近的角点的初始值
- region_size
  - 在region_size内寻找角点

利用find4QuadCornerSubpix函数可以更精确的找到角点，提高标定的精度。

相机标定

`OpenCV`提供了函数`calibrateCamera()`来实现相机标定的相关功能，他的声明如下

```
double cv::calibrateCamera(InputArrayOfArrays objectPoints,
                    InputArrayOfArrays imagePoints,
                    Size imageSize,
                    InputOutputArray cameraMatrix,
                    InputOutputArray distCoeffs,
                    OutputArrayOfArrays rvecs,
                    OutputArrayOfArrays tvecs,
                    int flags = 0,
                    TermCriteria criteria = TermCriteria(TermCriteria::CO
UNT+TermCriteria::EPS, 30, DBL_EPSILON)
)
```

下面说明他的参数的意义：

- objectPoints
  - 棋盘格上的角点对应的世界坐标系中的位置
- imagePoints
  - 棋盘格上找到的角点
- imageSize
  - 图片的大小
- cameraMatrix
  - 输出的相机内参矩阵
- distCoeffs
  - 输出的相机畸变矩阵
- rvecs
  - 相机坐标系与世界坐标系的旋转向量
- tvecs
  - 相机坐标系与世界坐标系的平移向量

一下为程序的实现：

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
```

```
using namespace cv;

int main() {

    const int board_w = 11, board_h = 8;
    const int board_n = board_w * board_h;
    Size board_size( 11, 8 );

    Mat gray_img, drawn_img;
    std::vector< Point2f > point_pix_pos_buf;
    std::vector< std::vector<Point2f> > point_pix_pos;

    int found, successes = 0;
    Size img_size;

    int cnt = 0;
    int k = 0, n = 0;
    for (int i = 0; i < 20; i++){
        cv::Mat src0 = cv::imread(std::__cxx11::to_string(i).append(".jp
g"));

        if ( !cnt ) {
            img_size.width = src0.cols;
            img_size.height = src0.rows;
        }
        found = findChessboardCorners( src0, board_size, point_pix_pos_bu
f );
        if ( found && point_pix_pos_buf.size() == board_n ) {
            successes++;
            cvtColor( src0, gray_img, COLOR_BGR2GRAY );
            find4QuadCornerSubpix( gray_img, point_pix_pos_buf, Size( 5,
5 ) );
            point_pix_pos.push_back( point_pix_pos_buf );
            drawn_img = src0.clone();
            drawChessboardCorners( drawn_img, board_size, point_pix_pos_b
uf, found );
            imshow( "corners", drawn_img );
            waitKey( 50 );
        } else
            std::cout << "\tbut failed to found all chess board corners i
n this image" << std::endl;
        point_pix_pos_buf.clear();
        cnt++;

    };
    std::cout << successes << " useful chess boards" << std::endl;

    Size square_size( 10, 10 );
    std::vector< std::vector< Point3f > > point_grid_pos;
    std::vector< Point3f > point_grid_pos_buf;
    std::vector< int > point_count;
```

```
    Mat camera_matrix( 3, 3, CV_32FC1, Scalar::all( 0 ) );
    Mat dist_coeffs( 1, 5, CV_32FC1, Scalar::all( 0 ) );
    std::vector< Mat > rvecs;
    std::vector< Mat > tvecs;

    for (int i = 0; i < successes; i++ ) {
        for (int j = 0; j < board_h; j++ ) {
            for (int k = 0; k < board_w; k++ ){
                Point3f pt;
                pt.x = k * square_size.width;
                pt.y = j * square_size.height;
                pt.z = 0;
                point_grid_pos_buf.push_back( pt );
            }
        }
        point_grid_pos.push_back( point_grid_pos_buf );
        point_grid_pos_buf.clear();
        point_count.push_back( board_h * board_w );
    }

    std::cout << calibrateCamera( point_grid_pos, point_pix_pos, img_siz
e, camera_matrix, dist_coeffs, rvecs, tvecs ) << std::endl;
    std::cout << camera_matrix << std::endl << dist_coeffs << std::endl;
    return 0;
}
```

如果想要将畸变的图像还原为原图，可以使用函数`cvUndistort2( ImageC1, Show1, &intrinsic_matrix, &distortion_coeffs);`来进行图像矫正。