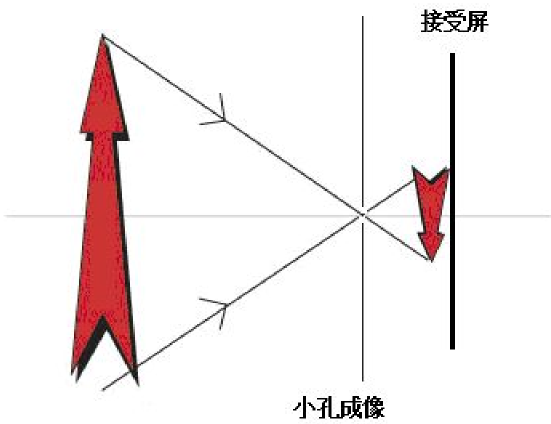
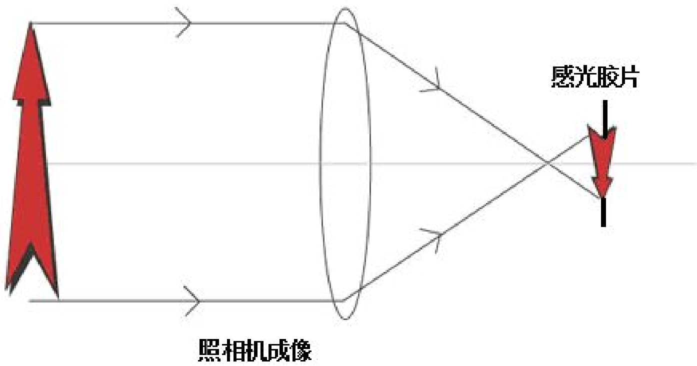
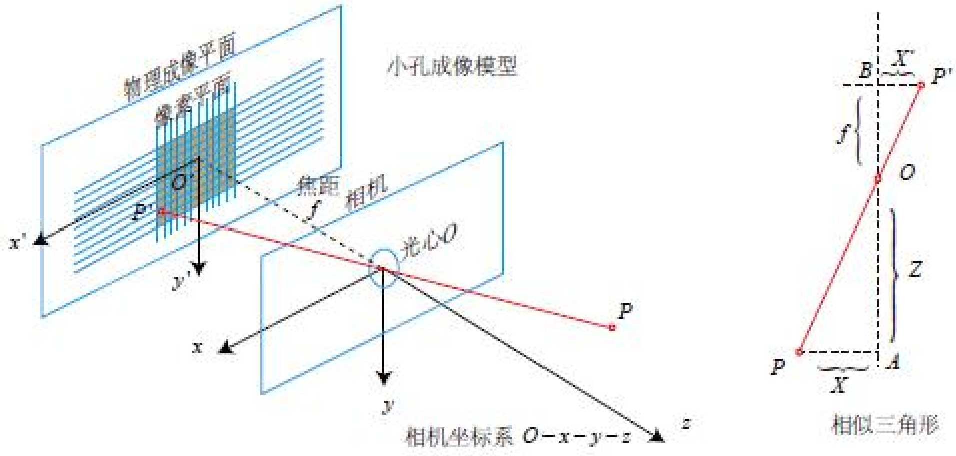
相机模型







$$
\begin{aligned}
X' &= -f\frac{X}{Z} \\
Y' &= -f\frac{Y}{Z}
\end{aligned}
\tag{1}
$$

几个重要坐标系的变换

- **世界坐标系:**

  代表物体在真实世界的三维坐标$(X_w, Y_w，Z_w)$

- **相机坐标系:**

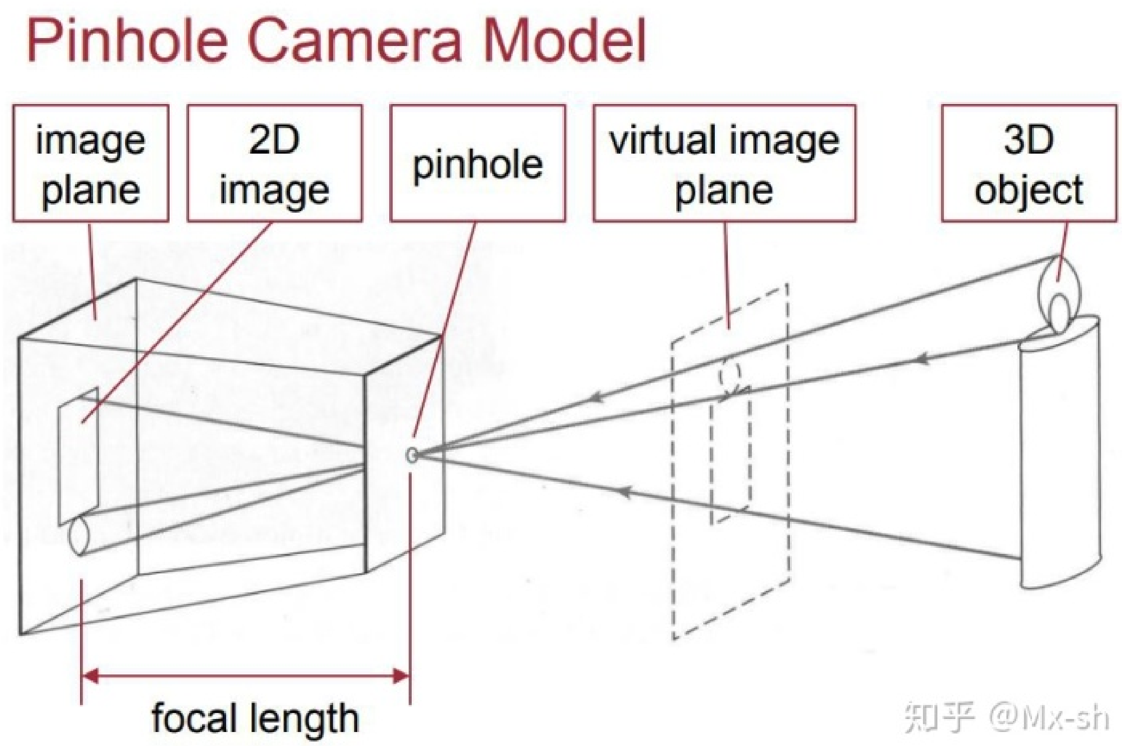
  以相机光学中心为原点的坐标系，光轴与Z轴重合$(X_c, Y_c，Z_c)$

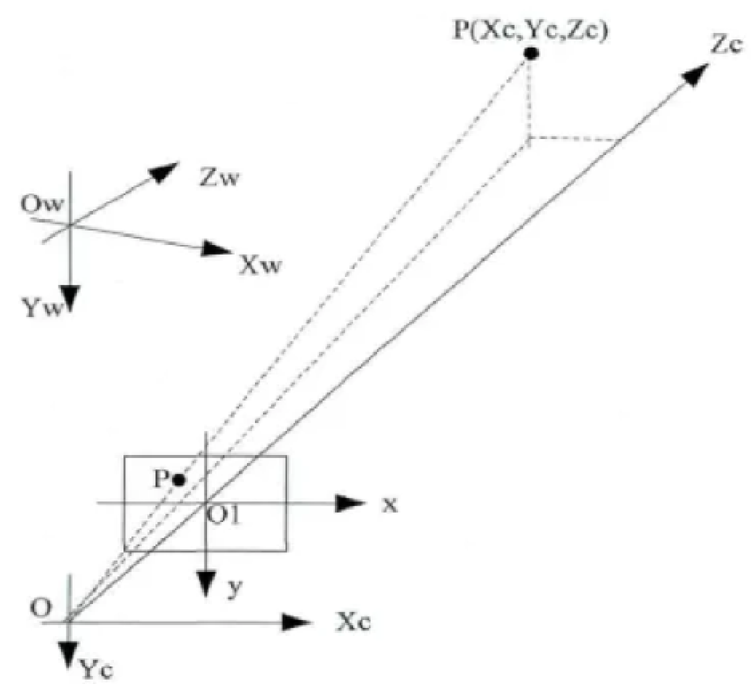
- **图像坐标系**

  代表相机拍摄的图像的坐标系，原点为相机光轴与成像平面的交点(x, y)

- **像素坐标系**

  在图像的平面上，基本单位是像素，原点一般在相片左上角(u, v)





世界坐标系到相机坐标系

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} \Rightarrow TP_w \tag{2}$$

相机坐标系到图像坐标系

$$\begin{cases} x = \frac{f}{Z_c} X_c \\ y = \frac{f}{Z_c} Y_c \end{cases} \tag{3}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{f}{Z_c} & 0 & 0 & 0 \\ 0 & \frac{f}{Z_c} & 0 & 0 \\ 0 & 0 & \frac{1}{Z_c} & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \Rightarrow K'P_c \tag{4}$$

图像坐标系到像素坐标系

设图像x方向每毫米有α个像素，y方向每毫米有β个像素，则有：

$$\begin{cases} u = c_x + x \cdot \alpha \\ v = c_y + y \cdot \beta \end{cases} \tag{5}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow K''P\,xy \tag{6}$$

其中c_x、c_y是图像坐标系原点与像素坐标系原点的偏移

综上，

$$P\,uv = K''K'TP_w \Rightarrow sKTP_w \tag{7}$$

其中，

$$s = \frac{1}{Z_c}; K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$f_x = \alpha f; f_y = \beta f$

矩阵K就是相机的内参数矩阵，K描述了相机焦距、像素精确度等性质。通产认为，相机的内参在出厂后就是固定的，不会在使用中发生变化。确定相机内参的过程就是**标定。**

矩阵T就是相机的外参数矩阵，T描述了相机坐标系相对世界坐标系的姿态和位置。外参会随相机的运动发生改变，也是SLAM估计的目标，描述了机器人的轨迹。

张正友标定法

> Reference: Zhang Z . A Flexible New Technique for Camera Calibration[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000, 22(11):1330-1334.
>
> https://www.bilibili.com/video/BV1eE411c7kr?t=23

引入：

相机标定要标定什么？

内参： $f_x, f_y, c_x, c_y, k_1, k_2, k_3, p_1, p_2$

外参： R, t

目标函数：

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\left\|m_{ij}-\hat{m}(A,R_i,t_i,P_j)\right\|_2$$

i表示拍摄的图片

j表示标定板上的角点数

A表示受$f_x, f_y, c_x, c_y, k_1, k_2, k_3, p_1, p_2$影响的变量

- 直接优化为什么不行？
- 要优化的量太多，初值不好就会陷入局部最优

张正友标定法的假设

1. 标定板的角点在一个平面上
2. 世界坐标系的xy平面在标定板平面上，Z=0
3. 相机模型不考虑畸变

Without loss of generality, we assume the model plane is on $Z = 0$ of the world coordinate system. Let's denote the i^{th} column of the rotation matrix $\mathbf{R}$ by $\mathbf{r}_i$. From (1), we have

$$s\begin{bmatrix}u\\v\\1\end{bmatrix}=\mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t}\end{bmatrix}\begin{bmatrix}X\\Y\\0\\1\end{bmatrix}$$

$$=\mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}\begin{bmatrix}X\\Y\\1\end{bmatrix}.$$

By abuse of notation, we still use $\mathtt{M}$ to denote a point on the model plane, but $\mathtt{M} = [X, Y]^T$ since Z is always equal to 0. In turn, $\widetilde{\mathtt{M}} = [X, Y, 1]^T$. Therefore, a model point $\mathtt{M}$ and its image $\mathbf{m}$ is related by a homography $\mathbf{H}$:

$$s\widetilde{\mathbf{m}}=\mathbf{H}\widetilde{\mathtt{M}} \quad \text{with} \quad \mathbf{H}=\mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}. \tag{2}$$

Given an image of the model plane, an homography can be estimated (see Appendix A). Let's denote it by $\mathbf{H} = \begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix}$. From (2), we have

$$\begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix}=\lambda\mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix},$$

where λ is an arbitrary scalar. Using the knowledge that $\mathbf{r}_1$ and $\mathbf{r}_2$ are orthonormal, we have

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2=0 \tag{3}$$

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_1=\mathbf{h}_2^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2. \tag{4}$$

Let

$$\mathbf{B} = \mathbf{A}^{-T}\mathbf{A}^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}$$
$$= \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2}+\frac{1}{\beta^2} & -\frac{\gamma(v_0\gamma-u_0\beta)}{\alpha^2\beta^2}-\frac{v_0}{\beta^2} \\ \frac{v_0\gamma-u_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma-u_0\beta)}{\alpha^2\beta^2}-\frac{v_0}{\beta^2} & \frac{(v_0\gamma-u_0\beta)^2}{\alpha^2\beta^2}+\frac{v_0^2}{\beta^2}+1 \end{bmatrix} . \tag{5}$$

Note that $\mathbf{B}$ is symmetric, defined by a 6D vector

$$\mathbf{b} = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T . \tag{6}$$

Let the i^{th} column vector of $\mathbf{H}$ be $\mathbf{h}_i = [h_{i1}, h_{i2}, h_{i3}]^T$. Then, we have

$$\mathbf{h}_i^T \mathbf{B} \mathbf{h}_j = \mathbf{v}_{ij}^T \mathbf{b} \tag{7}$$

with

$$\begin{aligned} \mathbf{v}_{ij} = [&h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, \\ &h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}]^T . \end{aligned}$$

Therefore, the two fundamental constraints (3) and (4), from a given homography, can be rewritten as 2 homogeneous equations in $\mathbf{b}$:

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = \mathbf{0} . \tag{8}$$

If n images of the model plane are observed, by stacking n such equations as (8) we have

$$\mathbf{V}\mathbf{b} = \mathbf{0} , \tag{9}$$

畸变模型

为了获得好的成像效果，相机加了透镜。透镜的加入对成像过程中光线的产生了新的影响。

- 径向畸变：透镜的厚薄不一，折射率不同，使得直线在投影后变成曲线。（戴眼镜或者玩过放大镜的同学应该见过）

$$\begin{aligned} x_{distorted} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) \\ y_{distorted} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6) \end{aligned} \tag{9}$$

- 切向畸变：机械组装过程中，透镜和成像平面不可能完全平行，从而导致切向畸变。

$$\begin{aligned} x_{distorted} &= x + 2p_1xy + p_2(r^2 + 2x^2) \\ y_{distorted} &= y + p_1(r^2 + 2y^2) + 2p_2xy \end{aligned} \tag{10}$$

我们一般在图像坐标系下对点计算畸变，然后投影到像素坐标系：

$$\begin{aligned} x_{distorted} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1xy + p_2(r^2 + 2x^2) \\ y_{distorted} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6) + p_1(r^2 + 2y^2) + 2p_2xy \end{aligned} \tag{11}$$

$$\begin{cases} u = c_x + x_{distorted} \cdot f_x \\ v = c_y + y_{distorted} \cdot f_y \end{cases} \tag{12}$$

在实际使用过程中可以灵活使用纠正模型，例如只用k_1, p_1, p_2

Opencv相机标定

OpenCV有现成函数:

```
cv::calibrateCamera(object_points, //三维点坐标
                        image_points_seq, //像素坐标
                        image_size, //图像尺寸
                        cameraMatrix, //输出相机矩阵
                        distCoeffs, //输出畸变矩阵
                        rvecsMat, tvecsMat, //r,t
                        0);
```

```
cv::findChessboardCorners(imageInput,//输入图像
                          board_size,//标定板上角点的行列
                          image_points_buf//输出角点的像素坐标
                         )
```

```
cv::cornerSubPix(view_gray,//输入图像，最好是灰度图
                 image_points_buf,//输入输出角点的像素坐标
                 cv::Size(5,5),//搜索窗口的半径
                 cv::Size(-1,-1),//-1表示忽略
                 cv::TermCriteria(cv::TermCriteria::MAX_ITER +
cv::TermCriteria::EPS,30,//最大迭代次数
                    0.1)//最小精度
                );
```

```
cv::drawChessboardCorners(view_gray, board_size, image_points_buf,
                          true);//如果角点全部找到，返回true
```

```
#include<iostream>
#include<string>
#include<opencv2/core/core.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<fstream>
#include <opencv2/calib3d.hpp>

using namespace std;

int main()
{
    ifstream fin("calibdata.txt");//标定图像文件的路径
    ofstream fout("caliberation_result.txt");//标定结果的文件
    //读取每一幅图像，从中提取角点，然后亚像素精确化
    cout << "开始提取角点。。。。。。\n";
    int image_count = 0;//图像数量
```

```
    cv::Size image_size;//图像的尺寸
    cv::Size board_size(4,6);//标定板上每行、列的角点数
    vector<cv::Point2f> image_points_buf;//保存每幅图上的角点
    vector < vector<cv::Point2f>> image_points_seq;//保存所有角点
    string filename;
    int count = -1;//计算储存角点的数目
    while (getline(fin, filename))//遍历标定图像，提取角点
    {
        image_count++;
        //观察检验输出
        cout << "第 " << image_count << " 张标定图片" << endl;
        cout << "角点个数为 " << count << endl;
        cv::Mat imageInput = cv::imread(filename);
        if (image_count == 1)//第一张图片读入宽高信息
        {
            image_size.width = imageInput.cols;
            image_size.height = imageInput.rows;
            cout << "图像的宽为 " << image_size.width << endl;
            cout << "图像的高为 " << image_size.height << endl;
        }

        //提取角点
        if (0 == cv::findChessboardCorners(imageInput, board_size,
image_points_buf))
        {
            cout << "找不到角点! \n";
            exit(1);
        }
        else//转为灰度图，然后亚像素级精度化，保存
        {
            //转为灰度图
            cv::Mat view_gray;
            cv::cvtColor(imageInput, view_gray, CV_RGB2GRAY);
            //亚像素级精度化
            cv::cornerSubPix(view_gray,image_points_buf,
                cv::Size(5,5),//搜索窗口的半径
                cv::Size(-1,-1),
                cv::TermCriteria(cv::TermCriteria::MAX_ITER +
cv::TermCriteria::EPS,30,//最大迭代次数
                    0.1)//最小精度
                );
            //保存结果
            if (image_points_buf.size() == board_size.area())
            {
                //如果棋盘完整，就保存
                image_points_seq.push_back(image_points_buf);
            }
            //在图像上显示角点的位置
            cv::drawChessboardCorners(view_gray, board_size, image_points_buf,
true);
            cv::imshow("Camera Calibration", view_gray);
            cv::waitKey(1000);
        }
    }


    //相机标定
    cout << "开始标定。。。。。。。\n";
```

```
    //棋盘三维信息
    vector<vector<cv::Point3f>> object_points;/* 保存标定板上角点的三维坐标 */
    //内外参数
    cv::Mat cameraMatrix(3, 3, CV_32FC1, cv::Scalar::all(0));//内参数矩阵
    vector<int> point_counts; // 每幅图像中角点的数量
    cv::Mat distCoeffs(1, 5, CV_32FC1, cv::Scalar::all(0));//5个畸变系数:
k1,k2,p1,p2,k3
    vector<cv::Mat> tvecsMat; /* 每幅图像的旋转向量 */
    vector<cv::Mat> rvecsMat; /* 每幅图像的平移向量 */

    /* 初始化标定板上角点的三维坐标 */
    for (int t = 0; t < image_count; t++)
    {
        vector<cv::Point3f> tempPointSet;
        for (int i = 0; i < board_size.height; i++)
        {
            for (int j = 0; j < board_size.width; j++)
            {
                cv::Point3f realPoint(i,j,0.0f);
                tempPointSet.push_back(realPoint);
            }
        }
        object_points.push_back(tempPointSet);
    }

    /* 初始化每幅图像中的角点数量，假定每幅图像中都可以看到完整的标定板 */
    for (int i = 0; i < image_count; i++)
    {
        point_counts.push_back(board_size.width * board_size.height);
    }

    //开始标定
    cv::calibrateCamera(object_points, //三维点
                        image_points_seq, //图像点
                        image_size, //图像尺寸
                        cameraMatrix, //输出相机矩阵
                        distCoeffs, //输出畸变矩阵
                        rvecsMat, tvecsMat, //r,t
                        0);
    cout << "标定完成! \n";

    //对标定结果进行评价
    cout << "开始评价标定结果.......\n";
    double total_err = 0.0;//总误差
    double err = 0.0;//单张图像的误差
    vector<cv::Point2f> image_points2;//利用内外参数重投影后的点
    cout << "\t每幅图像的标定误差: \n";
    fout << "每幅图像的标定误差: \n";
    for (int i = 0; i < image_count; i++)
    {
        vector<cv::Point3f> tempPointSet = object_points[i];//单张图片的角点的三维坐
标位置
        //重投影
        cv::projectPoints(tempPointSet,
            rvecsMat[i], tvecsMat[i],
            cameraMatrix,
            distCoeffs,
            image_points2//输出图像投影点坐标
```

```
        );
        //计算重投影点与原角点的误差
        vector<cv::Point2f> tempImagePoint = image_points_seq[i];//单张图片角点的像
素坐标
        //将角点point类型转换为矩阵类型
        cv::Mat tempImagePointMat = cv::Mat(1, tempImagePoint.size(), CV_32FC2);
        cv::Mat image_points2Mat = cv::Mat(1, image_points2.size(), CV_32FC2);
        for (int j = 0; j < tempImagePoint.size(); j++)
        {
            image_points2Mat.at<cv::Vec2f>(0, j) = cv::Vec2f(image_points2[j].x,
image_points2[j].y);
            tempImagePointMat.at<cv::Vec2f>(0, j) =
cv::Vec2f(tempImagePoint[j].x, tempImagePoint[j].y);
        }
        err = cv::norm(image_points2Mat, tempImagePointMat, cv::NORM_L2);//求二范
数
        total_err += (err / point_counts[i]);
        cout << "第" << i + 1 << "幅图像的平均误差: " << err << "像素" << endl;
        fout << "第" << i + 1 << "幅图像的平均误差: " << err << "像素" << endl;
    }
    cout << "总体平均误差: " << total_err / image_count << "像素" << endl;
    fout << "总体平均误差: " << total_err / image_count << "像素" << endl << endl;

    //保存定标结果
    std::cout << "开始保存定标结果………………" << endl;
    cv::Mat rotation_matrix = cv::Mat(3, 3, CV_32FC1, cv::Scalar::all(0)); /* 保
存每幅图像的旋转矩阵 */
    fout << "相机内参数矩阵: " << endl;
    fout << cameraMatrix << endl << endl;
    fout << "畸变系数: \n";
    fout << distCoeffs << endl << endl << endl;
    for (int i = 0; i < image_count; i++)
    {
        fout << "第" << i + 1 << "幅图像的旋转向量: " << endl;
        fout << tvecsMat[i] << endl;
        /* 将旋转向量转换为相对应的旋转矩阵 */
        cv::Rodrigues(tvecsMat[i], rotation_matrix);
        fout << "第" << i + 1 << "幅图像的旋转矩阵: " << endl;
        fout << rotation_matrix << endl;
        fout << "第" << i + 1 << "幅图像的平移向量: " << endl;
        fout << rvecsMat[i] << endl << endl;
    }
    std::cout << "完成保存" << endl;
    fout << endl;

    return 0;
}
```

> https://blog.csdn.net/dcrmg/article/details/52929669?ops_request_misc=&request_id=&biz_id=102&utm_term=boardsize%20%20opencv&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-4-.first_rank_v2_pc_rank_v29&spm=1018.2226.3001.4187