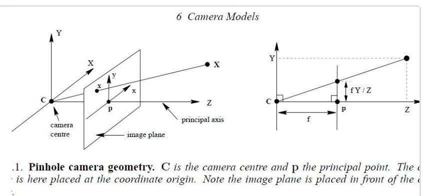
Tutorial_3

1. 相机畸变和 fov cu cv， 内参和外参数矩
   a. 网页与 multiple view geometry： Camera Geometr

相机标定之一：相机模型(读multiple view geometry in computer vision)

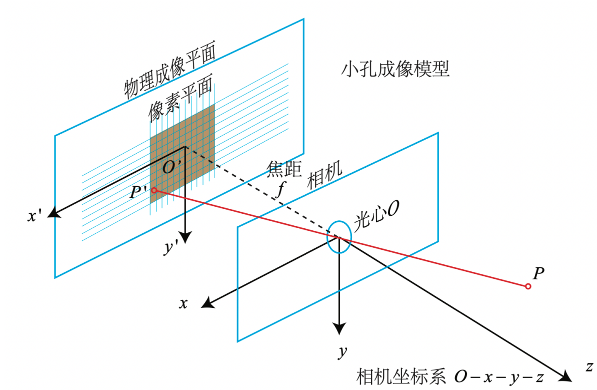
相机是3D世界和2D图像之间的一种映射。 透镜成像原理、小孔成像原理 基本针孔模型 (The Basic Pinhole Model) 以空间点到一张平面的中心投影来说明。令投影中心作为欧几里德坐标系的原点，平面Z=f为图像平面(image plane)或聚焦平面（focal ...

https://blog.csdn.net/pengjc2001/article/details/54237098



Camera and OpenCV Distortion

预先知识

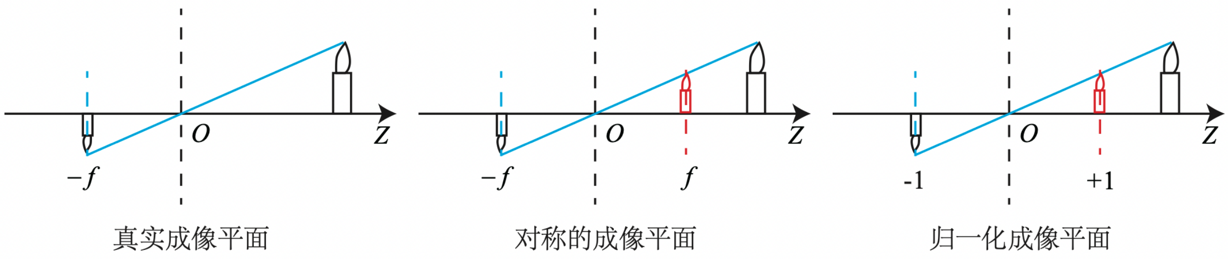


小孔成像

P 点为现实世界的空间点，坐标为$[X,Y,Z]^T$ ；

P' 点为物理成像平面上的成像点，坐标为$[X',Y',Z']^T$;

$\hat{P}$ 点为现实世界的空间点在相机归一化平面上的投影点，坐标为$[\hat{X},\hat{Y},\hat{Z}]^T$ 如下图所示：



- **对于对称的成像平面：**

$$\frac{Z}{f}=\frac{X}{X'}=\frac{Y}{Y'} \tag{1}$$

整理后可得：

$$X'=f\frac{X}{Z},\ Y'=f\frac{Y}{Z} \tag{2}$$

- **对于归一化成像平面：**

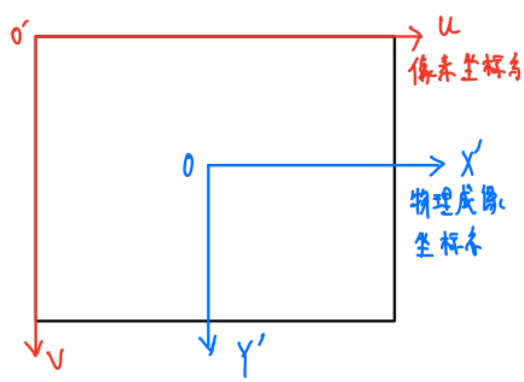
$$\frac{Z}{1}=\frac{X}{\hat{X}}=\frac{Y}{\hat{Y}} \tag{3}$$

整理后可得：

$$\hat{X}=\frac{X}{Z},\ \hat{Y}=\frac{Y}{Z} \tag{4}$$

像素(图像)坐标系定义：原点 o′ 位于图像的左上角，u 轴向右与 x 轴平行，v 轴向下与 y 轴平行。

像素坐标系与成像平面之间，相差了一个缩放和一个原点的平移。我 们设像素坐标在 u 轴上缩放了 α 倍，在 v 上缩放了 β 倍。同时，原点平移了 $[c_x,c_y]^T$ 。那 么，P ′ 的坐标与像素坐标 $[u,v]^T$ 的关系为



$$\begin{cases} u=\alpha X'+c_x \\ v=\beta Y'+c_y \end{cases} \tag{5}$$

代入(2)，把 αf 合并成 f_x ，把 βf合并成 f_y ，得:

$$\left\{\begin{array}{l} u=f_x \frac{X}{Z}+c_x \\ v=f_y \frac{Y}{Z}+c_y \end{array}\right. \tag{6}$$

针孔相机模型

1. 世界坐标系下空间点 P_w → 相机坐标系下空间点 P_c

设世界坐标系下空间点齐次形式 $P_w=[X_w, Y_w, Z_w, 1]^T$

设相机坐标系下空间点齐次形式 $P_c=[X_c, Y_c, Z_c, 1]^T$

$$\left[\begin{array}{c} X_c \\ Y_c \\ Z_c \\ 1 \end{array}\right]=\left[\begin{array}{cc} R & t \\ 0 & 1 \end{array}\right]\left[\begin{array}{c} X_w \\ Y_w \\ Z_w \\ 1 \end{array}\right] \tag{7}$$

2. 相机坐标系下空间点 P_c → 相机归一化平面上的投影点 $\hat{P}_c$

为了能与 3x3 的内参矩阵相乘，取计算得到的齐次形式 P_c 的前三个维度，即 $P_c=[X_c, Y_c, Z_c]^T$为了能凑成 **公式(6)** 的形式，需要将 P_c 投影到归一化平面上，得 $\hat{P}_c$ ：

$$\hat{P}_c=\left[\begin{array}{c} \frac{X_c}{Z_c} \\ \frac{Y_c}{Z_c} \\ 1 \end{array}\right] \tag{8}$$

3. 相机归一化平面上的投影点 $\hat{P}_c$ → 像素坐标点 $[u, v]^T$

将 **公式(6)** 写为矩阵形式：

$$\left[\begin{array}{l} u \\ v \\ 1 \end{array}\right]=\left[\begin{array}{ccc} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{array}\right]\left[\begin{array}{c} \frac{X_c}{Z_c} \\ \frac{Y_c}{Z_c} \\ 1 \end{array}\right] \tag{9}$$

其中：

$$\left\{\begin{array}{l} f_x=\alpha f \\ f_y=\beta f \end{array}\right. \tag{10}$$

去畸变过程以及OpenCV做法

去畸变过程是在 **归一化平面** 上完成的

Code for non-linear optimization

```
// Step 2: Refine the pose estimates using non-linear optimization
  MatrixXd gamma(....);
  MatrixXd JacobianMatrix[.....];
  int iter_threshold = 200;
  for (int count = 0; count < iter_threshold ; count++)
  {
    // step2.1: Get Euler Angles:
    // ?????? euler_angles = R.eulerAngles(.....)
    // double psi = euler_angles(...);
    // double phi = euler_angles(...);
    // double theta = euler_angles(...);

    // step2.2: calculate residue and jacobian
    for (int i = 0;i<num_pts;i++)
    {
      // calculate residue
      ????
      gamma.block<2,1>(0,i) = ........;

      MatrixXd tmp_J(2,6);
      tmp_J(0, 3) =
      tmp_J(0, 4) =
      tmp_J(0, 5) =
      tmp_J(1, 3) =
      tmp_J(1, 4) =
      tmp_J(1, 5) =

      double dR11_dphi =
      double dR12_dphi =
      double dR13_dphi =

      double dR21_dphi =
      double dR22_dphi =
      double dR23_dphi =

      double dR31_dphi =
      double dR32_dphi =
      double dR33_dphi =

      double dR11_dtheta =
      double dR12_dtheta =
      double dR13_dtheta =

      double dR21_dtheta =
      double dR22_dtheta =
      double dR23_dtheta =

      double dR31_dtheta =
      double dR32_dtheta =
      double dR33_dtheta =

      double dR11_dpsi =
      double dR12_dpsi =
      double dR13_dpsi =

      double dR21_dpsi =
      double dR22_dpsi =
      double dR23_dpsi =

      double dR31_dpsi =
      double dR32_dpsi =
      double dR33_dpsi =

      double dtrans_xw_dphi =
      double dtrans_yw_dphi =
      double dtrans_zw_dphi =

      double dtrans_xw_dpsi =
```

```
      double dtrans_yw_dpsi =
      double dtrans_zw_dpsi =

      double dtrans_xw_dtheta =
      double dtrans_yw_dtheta =
      double dtrans_zw_dtheta =

      tmp_J(0, 0) =
      tmp_J(0, 1) =
      tmp_J(0, 2) =

      tmp_J(1, 0) =
      tmp_J(1, 1) =
      tmp_J(1, 2) =

      JacobianMatrix[i] = tmp_J;
    }

    MatrixXd An = MatrixXd::Zero(6,6);
    MatrixXd bn = MatrixXd::Zero(6,1);
    // add up together
    for (int i = 0; i < num_pts;i++)
    {
      An +=
      bn -=
    }
    MatrixXd delta_p =

    // update the optimization variables
    T =
    phi =
    theta =
    psi =
    // std::cout << "update position" << delta_p.block<3,1>(3,0) << endl;
    R = AngleAxisd(psi, Vector3d::UnitZ()) * AngleAxisd(phi, Vector3d::UnitX()) * AngleAxisd(theta, Vector3d::UnitY());
    if (delta_p.cwiseAbs().sum() < 0.0001)
    {
      break;
    }
  }
```