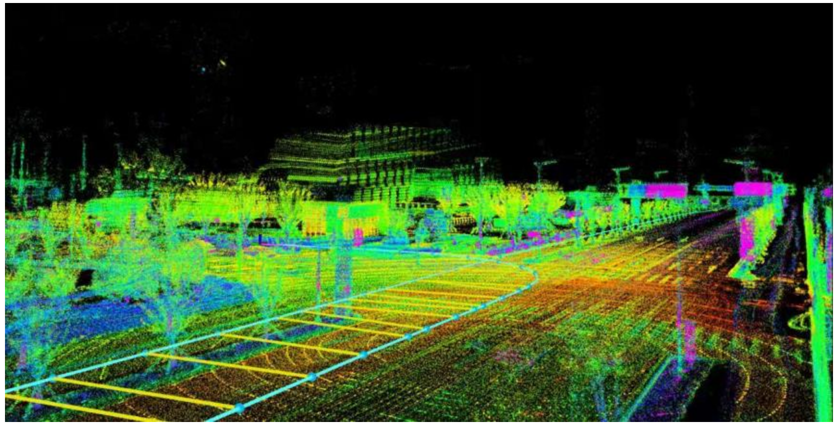
微信扫码加入星球

自动驾驶中实战基础之3D-2D求解方法

Camera + LiDAR + Radar + IMU

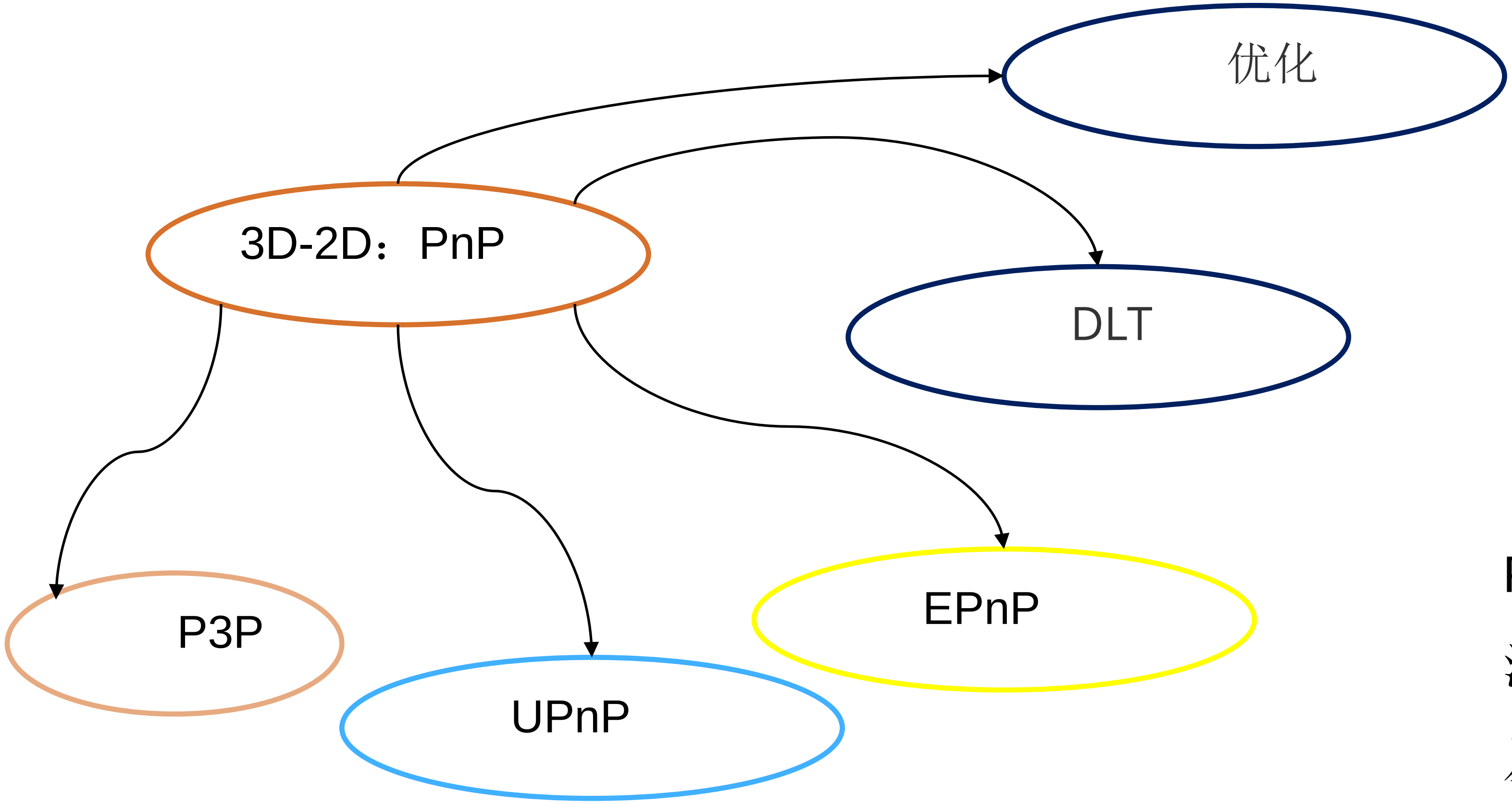


主 讲 人：爱喝苦咖啡的小阿飞

公 众 号：3D 视 觉 工 坊

内容

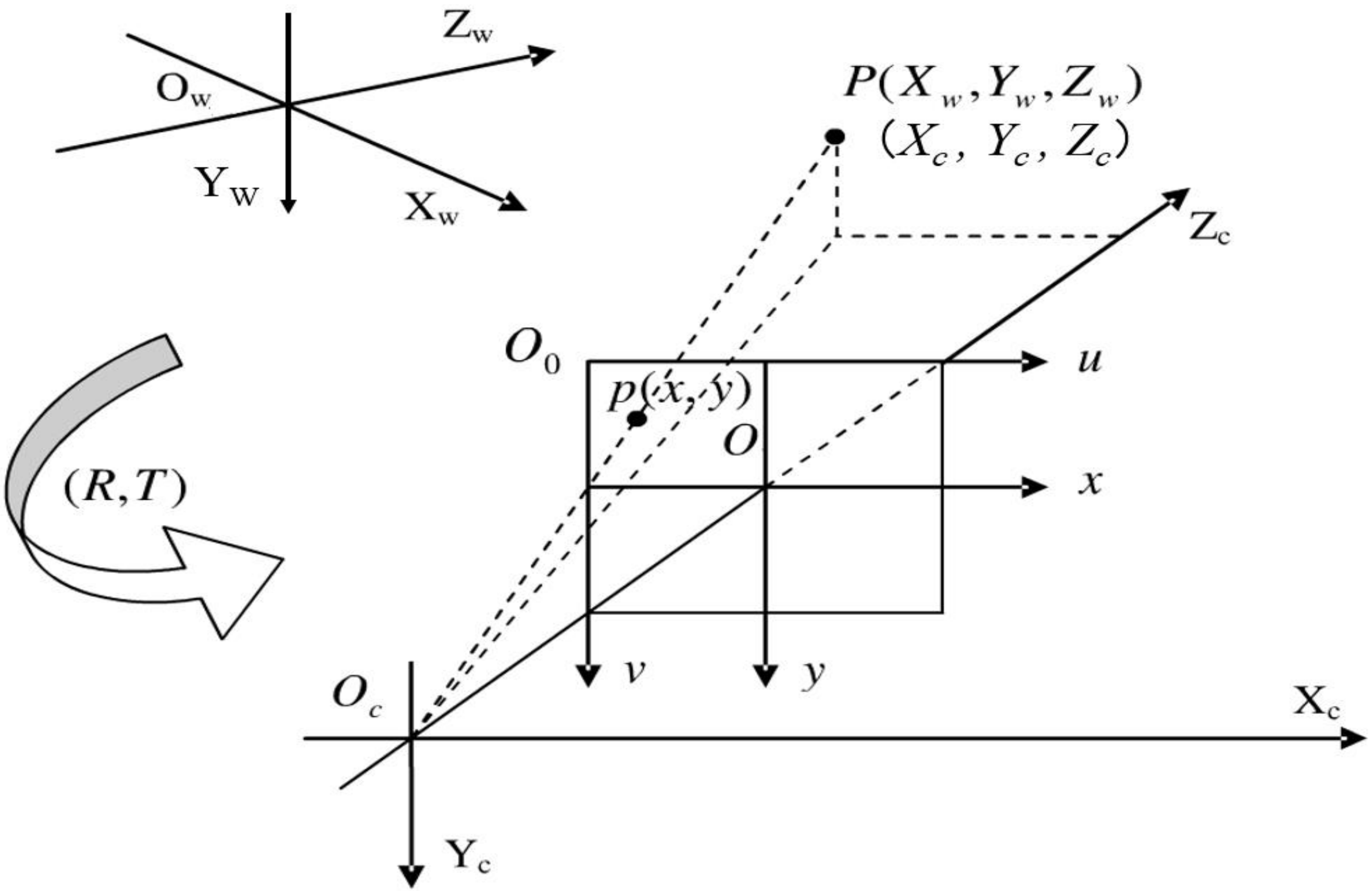
一、3D-2D求解方法



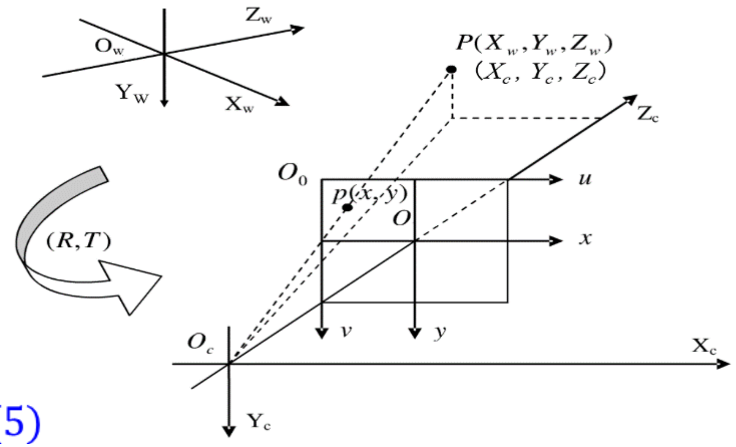
PnP是求解3D到2D点对运动的方法。它描述了当知道n个3D空间点及其投影位置时，如何估计相机的位姿（位置和姿态）。

PnP: perspective-n-Point
EPnP: Efficient PnP

- Image/Camera/World Coordinate



- Camera Projection Model



$$u = \frac{x}{dx} + u_0 \quad (1) \qquad v = \frac{y}{dy} + v_0 \quad (2)$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (3)$$

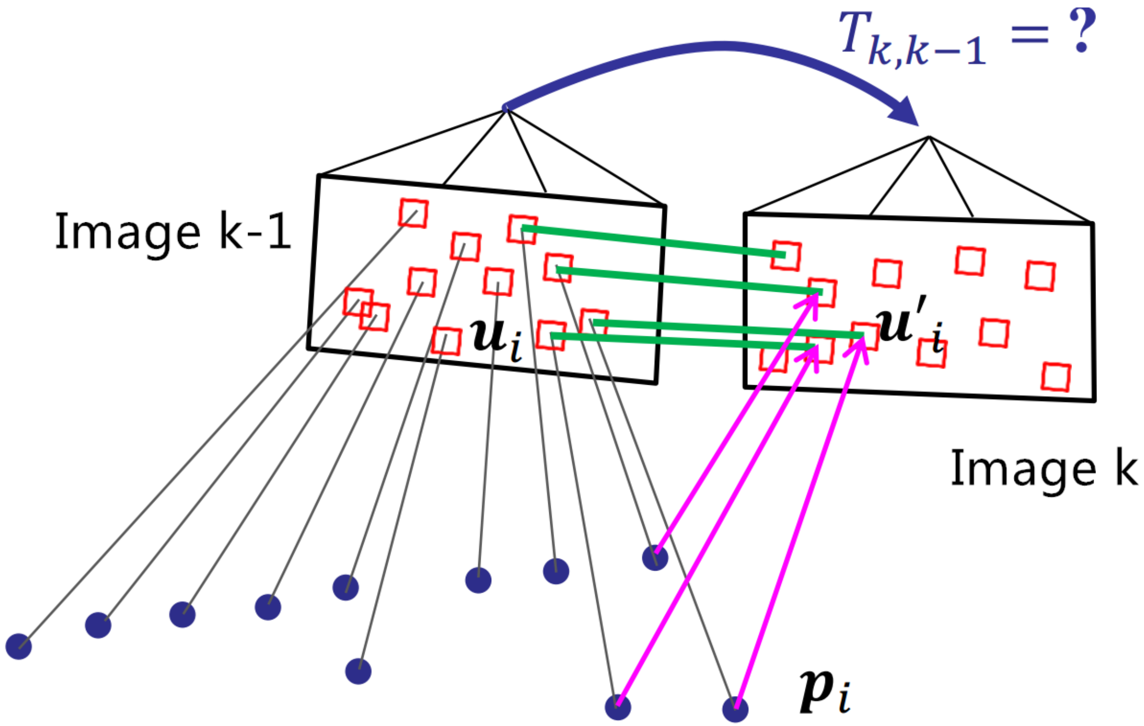
$$x = f\frac{X_c}{Z_c} \quad (4) \qquad y = f\frac{Y_c}{Z_c} \quad (5)$$

$$Z_c \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \quad (6)$$

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = (\boldsymbol{R} \quad \boldsymbol{t}) \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad (7)$$

$$Z_c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \overbrace{\begin{pmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}}^{\boldsymbol{K}} (\boldsymbol{R} \quad \boldsymbol{t}) \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = \boldsymbol{P} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad (8)$$

➢ 3D-2D PnP（DLT,P3P,EPnP,UPnP）



- 图像K中u_i'和世界坐标系下p_i均已知，计算$T_{k,k-1}$；
- 相机内参K已知的情况下至少需要3对对应点，当相机内参K未知时，则需要至少6对对应点。

- Perspective-3-Point (P3P)

已知:

BC = a’

AC = b’

AB = c’

$\angle BPC = \alpha$

$\angle APC = \beta$

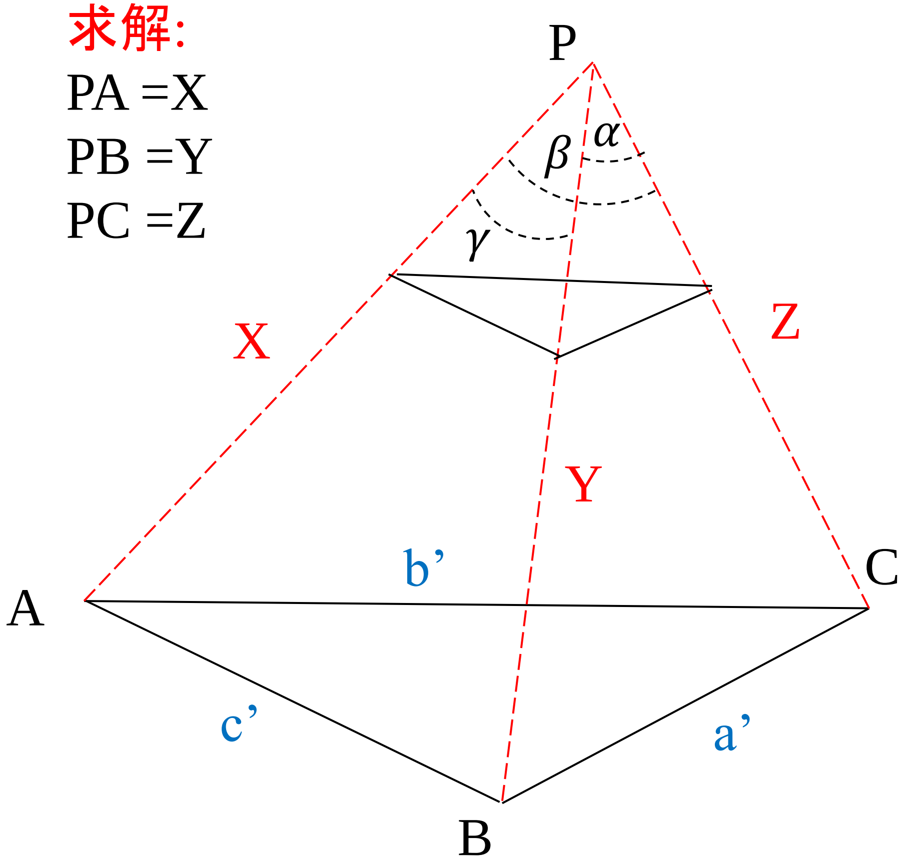
$\angle APB = \gamma$

求解:

PA =X

PB =Y

PC =Z



令:

$$X = xZ\ ;\qquad p = 2\cos\alpha;$$
$$Y = yZ\ ;\qquad q = 2\cos\beta;$$
$$c'^2 = vZ^2\ ;\qquad r = 2\cos\gamma;$$

$$a'^2 = ac'^2 = avZ^2\ ;$$
$$b'^2 = bc'^2 = bvZ^2\ ;$$

$$\begin{aligned} Y^2 + Z^2 - 2YZ\cos\alpha &= a'^2; \\ X^2 + Z^2 - 2XZ\cos\beta &= b'^2; \\ X^2 + Y^2 - 2XY\cos\gamma &= c'^2; \end{aligned} \qquad (10)$$

约束条件：P, A, B, C 不共面条件: $p^2 + q^2 + r^2 - pqr - 1 \neq 0;$ (9)

- Perspective-3-Point (P3P)

$$
\begin{aligned}
Y^2 + Z^2 - 2YZ\cos\alpha &= a'^2; \\
X^2 + Z^2 - 2XZ\cos\beta &= b'^2; \quad (10) \\
X^2 + Y^2 - 2XY\cos\gamma &= c'^2;
\end{aligned}
$$

替代

$$
\begin{aligned}
y^2Z^2 + Z^2 - yZ^2p &= avZ^2; \\
x^2Z^2 + Z^2 - xZ^2q &= bvZ^2; \\
x^2Z^2 + y^2Z^2 - xyZ^2r &= vZ^2;
\end{aligned}
$$

$/ Z^2$

$$
\begin{aligned}
y^2 + 1 - yp - av &= 0; \\
x^2 + 1 - xq - bv &= 0; \\
x^2 + y^2 - xyr - v &= 0;
\end{aligned}
$$

$$v = x^2 + y^2 - xyr \quad (11)$$

$$
\begin{aligned}
(1-a)y^2 - ax^2 + axyr - yp + 1 &= 0; \\
(1-b)x^2 - by^2 + bxyr - xq + 1 &= 0;
\end{aligned}
\quad (12)\ \text{ES}
$$

$Zero(ES/I_0)$ $Zero(TS_1/T_1)$

$$
\begin{aligned}
f&: a_0x^4 + a_1x^3 + a_2x^2 + a_3x + a_4 = 0; \\
g&: b_0y - b_1 = 0;
\end{aligned}
\quad (13)
$$

(14)

$$Z = \frac{c'}{\sqrt{v}} \qquad X = xZ \qquad Y = yZ$$

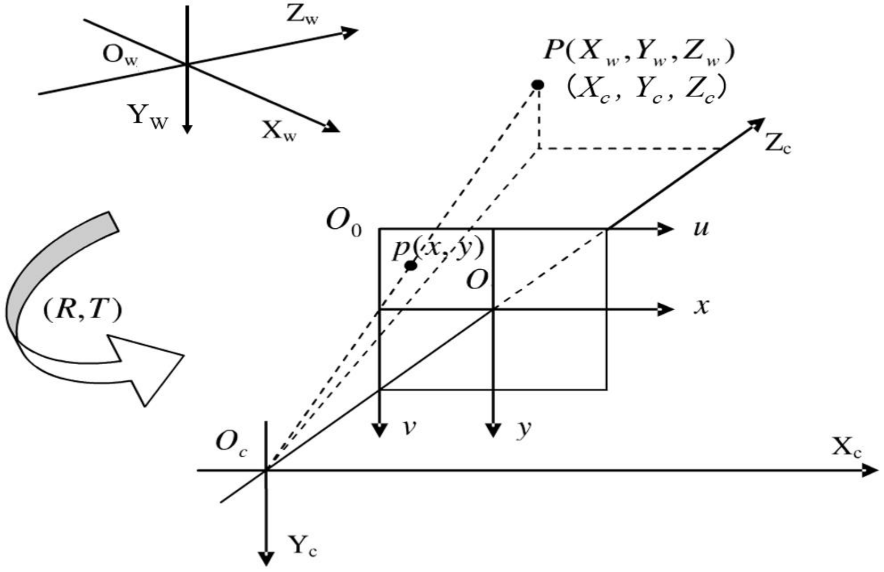
➢ Perspective-3-Point (P3P)

❶
$$\frac{\sqrt{x^2+y^2+f^2}}{OP}=\frac{x}{X_c};$$

$$X_c=\frac{x*OP}{\sqrt{x^2+y^2+f^2}}=\frac{\frac{x}{fx}*OP}{\sqrt{(\frac{x}{fx})^2+(\frac{y}{fy})^2+1}};$$

❷
$$Y_c=\frac{\frac{y}{fy}*OP}{\sqrt{(\frac{x}{fx})^2+(\frac{y}{fy})^2+1}};$$

$$Z_c=\frac{1*OP}{\sqrt{(\frac{x}{fx})^2+(\frac{y}{fy})^2+1}};$$



❸ 已知:
$P_c^1, P_c^2, P_c^3, P_w^1, P_w^2, P_w^3$

求解: **R & t** (Align the camera coordinate and the world coordinate)

- 计算α,β,γ

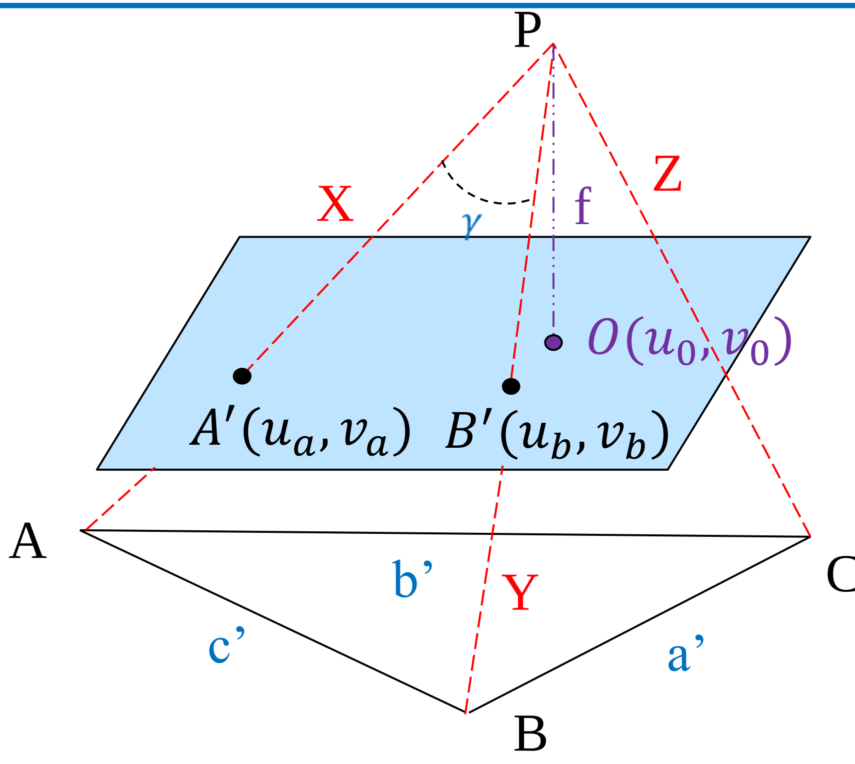
$$\begin{aligned}\cos\gamma &= \cos\angle \mathrm{APB}\\ &= \cos\angle A'\mathrm{P}B'\\ &= \frac{\mathbf{PA'}\cdot\mathbf{PB'}}{|\mathbf{PA'}|*|\mathbf{PB'}|}\end{aligned}$$

$$\mathbf{PA'} = (u_a-u_0, v_a-v_0, f)$$

$$\mathbf{PB'} = (u_b-u_0, v_b-v_0, f)$$

$$\cos\gamma = \frac{(u_a-u_0)(u_b-u_0)+(v_a-v_0)(v_b-v_0)+f^2}{\sqrt{(u_a-u_0)^2+(v_a-v_0)^2+f^2}*\sqrt{(u_b-u_0)^2+(v_b-v_0)^2+f^2}}$$

$$或\ \cos\gamma = \frac{\left(\frac{u_a-u_0}{f_x}\right)\left(\frac{u_b-u_0}{f_x}\right)+\left(\frac{v_a-v_0}{f_y}\right)\left(\frac{v_b-v_0}{f_y}\right)+1}{\sqrt{\left(\frac{u_a-u_0}{f_x}\right)^2+\left(\frac{v_a-v_0}{f_y}\right)^2+1}*\sqrt{\left(\frac{u_b-u_0}{f_x}\right)^2+\left(\frac{v_b-v_0}{f_y}\right)^2+1}}$$



- Perspective-3-Point (P3P)

- 过程:
  1). 已知 |AB|, |BC|, |AC| and ∠BPC, ∠APC, ∠APB;
  2). 求解 |AP|, |BP|, |CP|;
  3). 求解相机坐标系下坐标: A_c, B_c, C_c;
  4). 计算 R 和 t (利用 A_c, B_c, C_c, A_w, B_w, C_w)
  5). 通过D_w and d_i (重投影误差)获得优化后的R 和 t 。

- 优点:
  1). Speed: fast
  2). Coplanar: ok

- 缺点:
  1). Precision: low
  2). Unstable: need RANSAC

```
bool p3p::solve(cv::Mat& R, cv::Mat& tvec, const cv::Mat& opoints, const cv::Mat& ipoints)
{
    double rotation_matrix[3][3], translation[3];
    std::vector<double> points;
    if (opoints.depth() == ipoints.depth())   // 将opoints中的XYZ和ipoints中的uv存入points中
    {
        if (opoints.depth() == CV_32F)
            extract_points<cv::Point3f,cv::Point2f>(opoints, ipoints, points);
        else
            extract_points<cv::Point3d,cv::Point2d>(opoints, ipoints, points);
    }
    else if (opoints.depth() == CV_32F)      // 考虑精度的模版
        extract_points<cv::Point3f,cv::Point2d>(opoints, ipoints, points);
    else
        extract_points<cv::Point3d,cv::Point2f>(opoints, ipoints, points);

    bool result = solve(rotation_matrix, translation, points[0], points[1], points[2], points[3], points[4], points[5],
          points[6], points[7], points[8], points[9], points[10], points[11], points[12], points[13], points[14],
          points[15], points[16], points[17], points[18], points[19]);    // 根据3组点得到最多四个解，然后用最后一组去验证（重投影误差）
    cv::Mat(3, 1, CV_64F, translation).copyTo(tvec);
    cv::Mat(3, 3, CV_64F, rotation_matrix).copyTo(R);
    return result;
}
```

```
bool p3p::solve(double R[3][3], double t[3],
    double mu0, double mv0,   double X0, double Y0, double Z0,
    double mu1, double mv1,   double X1, double Y1, double Z1,
    double mu2, double mv2,   double X2, double Y2, double Z2,
    double mu3, double mv3,   double X3, double Y3, double Z3)
{
    double Rs[4][3][3], ts[4][3];

    int n = solve(Rs, ts, mu0, mv0, X0, Y0, Z0,  mu1, mv1, X1, Y1, Z1, mu2, mv2, X2, Y2, Z2);   // 根据3组点得到最多四个解

    if (n == 0)
        return false;

    int ns = 0;
    double min_reproj = 0;
    for(int i = 0; i < n; i++) {                    // 用最后一组去验证（重投影误差）
        double X3p = Rs[i][0][0] * X3 + Rs[i][0][1] * Y3 + Rs[i][0][2] * Z3 + ts[i][0];
        double Y3p = Rs[i][1][0] * X3 + Rs[i][1][1] * Y3 + Rs[i][1][2] * Z3 + ts[i][1];
        double Z3p = Rs[i][2][0] * X3 + Rs[i][2][1] * Y3 + Rs[i][2][2] * Z3 + ts[i][2];
        double mu3p = cx + fx * X3p / Z3p;
        double mv3p = cy + fy * Y3p / Z3p;
        double reproj = (mu3p - mu3) * (mu3p - mu3) + (mv3p - mv3) * (mv3p - mv3);
        if (i == 0 || min_reproj > reproj) {
            ns = i;
            min_reproj = reproj;
        }
    }

    for(int i = 0; i < 3; i++) {                    // 拥有最小重投影误差的为最终解
        for(int j = 0; j < 3; j++)
            R[i][j] = Rs[ns][i][j];
        t[i] = ts[ns][i];
    }

    return true;
}
```

```
void solvePnP(InputArray objectPoints, //在世界坐标系下的控制点的坐标，如类型std::vector<cv::Point3d>
              InputArray imagePoints,  //在图像坐标系下对应的控制点的坐标，如类型 std::vector<cv::Point2d>
              InputArray cameraMatrix, //相机的内参矩阵，如类型cv::Mat(3, 3, CV_32FC1）
              InputArray distCoeffs,   //相机的畸变系数，如类型cv::Mat(1, 5, CV_32FC1)
              OutputArray rvec,        //输出的旋转向量
              OutputArray tvec,        //输出的平移向量
              bool useExtrinsicGuess=false,
              int flags = SOLVEPNP_ITERATIVE)
```

其中重点flags计算方法选择选择如下所示：

```
enum { SOLVEPNP_ITERATIVE = 0,
     SOLVEPNP_EPNP     = 1, //!< EPnP: Efficient Perspective-n-Point Camera Pose Estimation
     SOLVEPNP_P3P      = 2, //!< Complete Solution Classification for the Perspective-Three-Point Problem
     SOLVEPNP_DLS      = 3, //!< A Direct Least-Squares (DLS) Method for PnP  @cite hesch2011direct
     SOLVEPNP_UPNP     = 4, //!< Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation
     SOLVEPNP_AP3P     = 5, //!< An Efficient Algebraic Solution to the Perspective-Three-Point Problem
     SOLVEPNP_MAX_COUNT      //!< Used for count};
```

1. 课后自学cv::solvepnpRansac()函数，包括其中的每个参数的含义；
2. cv::solvePnP()与cv::solvePnPRansac()区别与联系是啥？

1) 《视觉SLAM十四讲》；

2) 基于PnP问题求解的摄像机标定_鄢琼；

3) PnP问题的线性求解算法_吴福朝；

4) 关于PnP问题多解的分布与解的稳定性的讨论_孙凤梅；

5) P3P: Complete Solution Classification for the Perspective-Three-Point Problem；

购买该课程请扫描二维码

微信扫码加入星球

感谢聆听

Thanks for Listening