solvePnP——Perspective-n-Point

一、求取位姿的方法

1. 对极约束：2D-2D，通过二维图像点的对应关系，恢复出在两帧之间摄像机的运动。
2. PnP：3D-2D,求解3D到2D点对运动的方法。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。
3. ICP：3D-3D，配对好的3D点,已知世界参考系下的3D点和相机参考系下的3D点

二、PnP概念

如果场景的三维结构已知，利用多个控制点在三维场景中的坐标及其在图像中的透视投影坐标即可求解出相机坐标系与世界坐标系之间的绝对位姿关系，包括绝对平移向量t以及旋转矩阵R，该类求解方法统称为**N点透视位姿求解**（Perspective-N-Point，PNP问题）。这里的控制点是指准确知道三维空间坐标位置，同时也知道对应图像平面坐标的点。

已知条件：

n3D参考点(3D reference points)坐标;
与这n个3D点对应的、投影在图像上的2D参考点(2D reference points)坐标;
相机的的内参K;

求解：

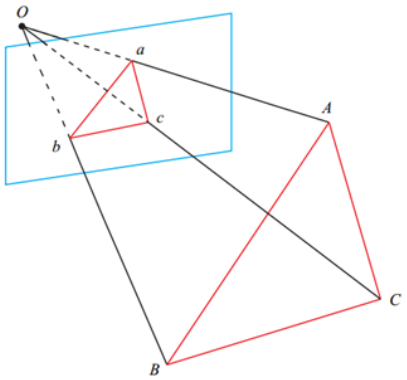
相机的位姿。

三、PnP的使用场景

场景主要有两个

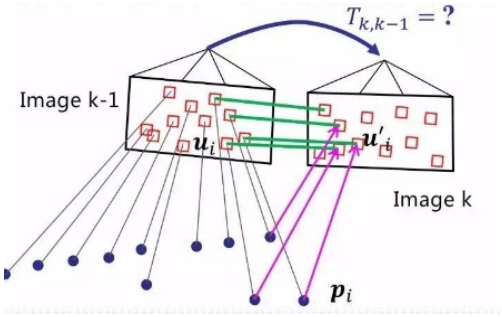
1. 求解相机的位姿，一般应用于AR，人脸跟踪等；

   通常输入的是物体在世界坐标系下的3D点以及这些3D点在图像上投影的2D点，因此求得的是相机（相机坐标系）相对于真实物体（世界坐标系）的位姿



2. 求取前一帧到当前帧的相机位姿变化，一般用于slam中；

通常输入的是上一帧中的3D点（在上一帧的相机坐标系下表示的点）和这些3D点在当前帧中的投影得到的2D点，所以它求得的是当前帧相对于上一帧的位姿变换



四、函数形式

solvePnP

```
bool solvePnP(InputArray _opoints,             //参考点在世界坐标系下的点集
              InputArray _ipoints,             //参考点在相机像平面的坐标
              InputArray _cameraMatrix,        //相机内参
              InputArray _distCoeffs,          //相机畸变系数
              OutputArray _rvec,               //旋转向量
              OutputArray _tvec,               //平移向量
              bool useExtrinsicGuess = false, //是否使用初始的旋转、平移向量，
                                               //只针对SOLVEPNP_ITERATIVE
              int flags = SOLVEPNP_ITERATIVE   //使用的求解方法
              )
```

- `flags` - 默认使用CV_ITERATIV迭代法

SOLVEPNP_ITERATIVE	适合点在同一平面上的情况
SOLVEPNP_EPNP	EPnP: Efficient Perspective-n-Point Camera Pose Estimation [91].点需要在不同平面
SOLVEPNP_P3P	Complete Solution Classification for the Perspective-Three-Point Problem [57].只适用于3点
SOLVEPNP_DLS	A Direct Least-Squares (DLS) Method for PnP [73].至少四点不在同一平面

1. CV_ITERATIVE，默认值，它通过迭代求出重投影误差最小的解作为问题的最优解。该方法实质是迭代求出重投影误差最小的解，这个解显然不一定是正解。

   使用范围：4对共面匹配点，小于4组点不准确

2. CV_P3P，是使用非常经典的Gao的P3P问题求解算法。P3P算法要求输入的控制点个数只能是4对。3对点求出*4*组可能的解，通过对第4对点进行重投影，返回重投影误差最小的，确定最优解。可以使用任意4对匹配点求解，不要共面，特征点数量不为4时报错.

```
xlyuefeng@ubuntu:~/test/pnp/build$ ./pose_estimation_3d2d /home/xlyuefeng/test/pnp/1.png /home/xlyuefeng/test/pnp/2.png /home/xlyuefeng/test/pnp/1_depth.png /home/xlyuefeng/test/pnp/2_depth.png
-- Max dist : 95.000000
-- Min dist : 4.000000
一共找到了79组匹配点
3d-2d pairs: 76
OpenCV Error: Assertion failed (npoints == 4) in solvePnP, file /home/xlyuefeng/package/opencv-3.1.0/modules/calib3d/src/solvepnp.cpp, line 103
terminate called after throwing an instance of 'cv::Exception'
  what():  /home/xlyuefeng/package/opencv-3.1.0/modules/calib3d/src/solvepnp.cpp:103: error: (-215) npoints == 4 in function solvePnP

Aborted (core dumped)
```

使用范围：4对不共面匹配点

3. CV_EPNP，该方法使用*EfficientPNP*方法，求解问题EPnP使用大于等于3组点，是目前最有效的PnP求解方法。

   使用范围：最少需要4对不共面的匹配点（对于共面的情况只需要3对）,点太少能运行，但不准确

注：方法 SOLVEPNP_DLS 和 SOLVEPNP_UPNP 不能使用，因为当前的实现是不稳定的，有时会给出完全错误的结果。如果传递这两个标志之一，则将使用 SOLVEPNP_EPNP 方法。

建议：小于4组匹配点时，用CV_ITERATIVE；在4组匹配点时，用CV_P3P；大于4组匹配点时，用CV_EPNP。

solvePnPRansac

solvePnP的一个缺点是对异常值不够鲁棒，当我们用相机定位真实世界的点，可能存在错配，对误匹配进行Ransac过滤，RANSAC是“Random Sample Consensus（随机抽样一致）”的缩写

```
bool solvePnPRansac(InputArray _opoints,            //参考点在世界坐标系下的点集
                    InputArray _ipoints,            //参考点在相机像平面的坐标
                    InputArray _cameraMatrix,       //相机内参
                    InputArray _distCoeffs,         //相机畸变系数
                    OutputArray _rvec,              //旋转矩阵
                    OutputArray _tvec,              //平移向量
                    bool useExtrinsicGuess = false, //W
                    int iterationsCount = 100,      //算法的迭代次数
                    float reprojectionError = 8.0,  //筛选内点和外点的距离阈值
                    double confidence = 0.99,       //代表N次采样可以使s个点全为内点的概率。
                    OutputArray _inliers,           //返回内点的序列
                    int flags = SOLVEPNP_ITERATIVE  //使用的求解方法
                    )
```

官方文档：
https://docs.opencv.org/4.0.1/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d

五、使用示例

已知一帧图像中特征点的3d位置信息，以及另一帧图像中特征点的2d位置信息，进行相机的位姿变换计算。

- 3d位置信息是指该特征点所对应的真实物体点，在当前相机坐标系下的坐标；

- 2d位置信息则是特征点的像素坐标。这里3d位置信息是由RGB-D相机提供的深度信息进行计算得到的。

详情见代码

六、扩展-PnP问题求解原理

目前主要有直接线性变换（DLT），P3P，EPnP，UPnP以及非线性优化方法。

关于PnP的求解问题

$$Z_c\begin{pmatrix}u\\v\\1\end{pmatrix}=\begin{pmatrix}\frac{1}{dx}&0&u_0\\0&\frac{1}{dy}&v_0\\0&0&1\end{pmatrix}\begin{pmatrix}f&0&0\\0&f&0\\0&0&1\end{pmatrix}\begin{pmatrix}R&T\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}$$

$$=\begin{pmatrix}f_x&0&u_0\\0&f_y&v_0\\0&0&1\end{pmatrix}\begin{pmatrix}R&T\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}=M_1M_2\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}$$

$$Z_c\begin{pmatrix}x\\y\\1\end{pmatrix}=\begin{pmatrix}f_x&0&u_0\\0&f_y&v_0\\0&0&1\end{pmatrix}\begin{pmatrix}R_{11}&R_{12}&R_{13}&T_1\\R_{21}&R_{22}&R_{23}&T_2\\R_{31}&R_{32}&R_{33}&T_3\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}$$

$$\begin{cases}Z_c*x=X_w*(f_x*R_{11}+u_0*R_{31})+Y_w*(f_x*R_{12}+u_0*R_{32})\\\quad+Z_w*(f_x*R_{13}+u_0*R_{33})+f_x*T_1+u_0*T_3 & (1)\\Z_c*y=X_w*(f_y*R_{21}+v_0*R_{31})+Y_w*(f_y*R_{22}+v_0*R_{32})\\\quad+Z_w*(f_y*R_{23}+v_0*R_{33})+f_y*T_2+v_0*T_3 & (2)\\Z_c=X_w*R_{31}+Y_w*R_{32}+Z_w*R_{33}+T_3 & (3)\end{cases}$$

把（3）式带入（1）式和（2）式，整理得：

$$Xw*(fx*R11+u0*R31-x*R31)+Yw*(fx*R12+u0*R32-x*R32)+Zw*(fx*R13+u0*R33-x*R33)=T3*x-fx*T1-u0*T3$$

$$Xw*(fy*R21+v0*R31-y*R31)+Yw*(fy*R22+v0*R32-y*R32)+Zw*(fy*R23+v0*R33-y*R33)=T3*y-fy*T2-v0*T3$$

我们可以看出，fx fy u0 v0是相机内参，上一节中已经求出，Xw Yw x y是一组3D/2D点的坐标，所以未知数有R11 R12 R13 R21 R22 R23 R31 R32 R33 T1 T2 T3一共12个，由于旋转矩阵是正交矩阵，每行每列都是单位向量且两两正交，所以R的自由度为3，秩也是3，比如知道R11 R12 R21就能求出剩下的Rxx。加上平移向量的3个未知数，一共6个未知数，而每一组2D/3D点提供的x y Xw Yw Zw可以确立两个方程，所以3组2D/3D点的坐标能确立6个方程从而解出6个未知数。

故PnP需要知道至少3组2D/3D点。

（一）DLT

通过2D-3D的关系直接构建方程，直接求解：

DLT主要是通过构建一个12维的增广矩阵（R|t），然后通过投影矩阵构建一个方程：

通过最后一行，消去s，最后构建一个12维的线性方程组，通过6对匹配点（一对点两个方程）来求解中间的矩阵。

最后将[R|t]左侧33矩阵块进行QR分解，用一个旋转矩阵去近似（将33矩阵空间投影到SE(3)流形上）。

$$s\begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \begin{pmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}.$$

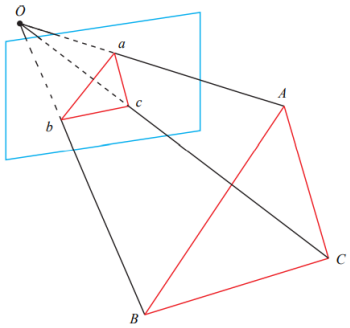
优点：求解简单

缺点：直接将 T 矩阵看成了 12 个未知数，忽略了它们之间的联系。因为旋转矩阵 R ∈ SO(3)，用 DLT 求出的解不一定满足该约束，它是一个一般矩阵。寻找一个最好的旋转矩阵对它进行近似

（二）P3P

采用一种变换的形式，把2D-3D匹配关系，变换成3D-3D点的匹配关系：

将世界坐标系下的ABC三点和图像坐标系下的abc三点匹配，其中AB，BC，AC的长度已知，<a,b>，<b,c>，<a,c>也是已知，通过余弦定理得出方程组



$$OA^2 + OB^2 - 2OA \cdot OB \cdot \cos\langle a,b\rangle = AB^2$$
$$OB^2 + OC^2 - 2OB \cdot OC \cdot \cos\langle b,c\rangle = BC^2$$
$$OA^2 + OC^2 - 2OA \cdot OC \cdot \cos\langle a,c\rangle = AC^2.$$

类似于分解 E 的情况，该方程最多可能得到四个解，但我们可以用验证点来计算最可能的解，得到 A, B, C 在相机坐标系下的 3D 坐标。然后，根据 3D-3D 的点对，使用类似ICP的坐标系对，计算相机的运动 R, t。

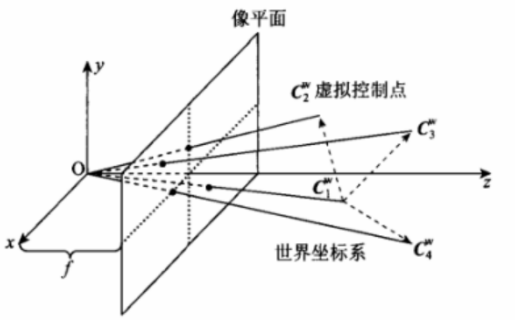
优点：需要的匹配点少

缺点：P3P 只利用三个点的信息。当给定的配对点多于 3 组时，难以利用更多的信息；如果 3D 点或 2D 点受噪声影响，或者存在误匹配，则算法失效。

可参考论文《*CompleteSolution Classification for the Perspective-Three-Point Problem*》

（三）EPnP

将世界坐标系中的3D坐标表示为一组虚拟的控制点的加权和。



可参考论文《EPnP: Efficient Perspective-n-Point Camera Pose Estimation》