PnPsolver

By薛午阳

**1、功能描述**

这个类就是用来在已知若干三维点的世界坐标和它对应二维图像上的坐标时，计算相机相对世界坐标的姿态的方法类。在orb-slam里使用的模块见图1。

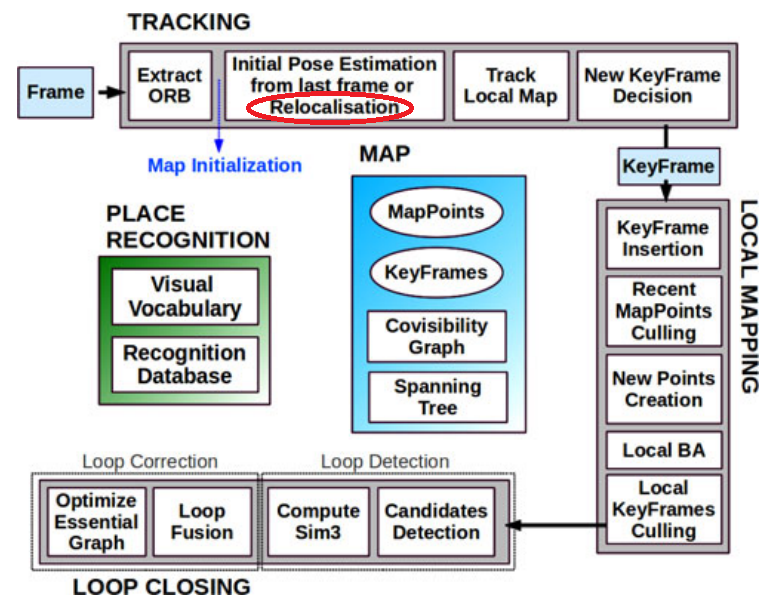


图1 orb-slam总结构

在第一篇orb-slam的paper中，PnP算法唯一地出现在第五章V. TRACKING的第三节C. Initial Pose Estimation via Global Relocalization中[1]。文章描述是：tracking丢失后，我们用BOW把当前帧与所有keyframe的特征匹配得到很多匹配的keyframe和对应的mappoint，对每个关键帧进行RANSAC迭代，并尝试用PnP算法找到相机的姿态。并引用了文章[2]。

在orb-slam程序中，PnP唯一出现在Tracking.cc中：

763行，trackreferencekeyframe函数中注释出现了PnP，实际上连声明变量都没有，此函数没有用PnP，和文章说明的情况符合；

1356~1385行，relocalization函数出现了PnP的调用，和文章说明的情况符合。

本文主要组成部分如下：2和3是类的简单介绍，主要介绍公有函数输入输出，方便直接调用，以及一些变量方便理解；4流程图建议在了解算法核心后阅读；5是算法核心介绍。

**2、成员变量**

pws：参考点世界坐标；

pcs：参考点相机坐标；

us：参考点在图像上的投影坐标；

cws：控制点世界坐标；

ccs：控制点相机坐标；

alphas：式1和2的α；

其他变量都可以直接看名字知道意思。

**3、成员函数**

类名：PnPsolver；

3.1 Public函数：

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 输入变量 | 返回变量 | 功能 |
| PnPsolver | const Frame &F | 无 | 构造函数并给ransac设默认参数 |
| const vector<MapPoint\*> &vpMapPointMatches |
| SetRansacParameters | double probability | 无 | 设定ransac算法的参数，有默认值 |
| int minInliers |
| int maxIterations |
| int minSet |
| float epsilon |
| float th2 |
| find | vector<bool> &vbInliers | cv::Mat | 调用iterate的入口函数 |
| int &nInliers |
| iterate | int nIterations | cv::Mat | 算法的开始，返回空的Mat |
| bool &bNoMore |
| vector<bool> &vbInliers |
| int &nInliers |

输入变量说明：

------------------------------------------------------------------------

PnPsolver

const Frame &F：待处理的一个Frame类对象；

const vector<MapPoint\*> &vpMapPointMatches：该Frame对象里已经匹配上的MapPoint对象

------------------------------------------------------------------------

SetRansacParameters

double probability：找到准确模型希望的概率，默认0.99；

int minInliers：最低内群数目，默认8；

int maxIterations：最大迭代次数，默认300；

int minSet：每次迭代使用最小参考点集，默认4；

float epsilon：预期的内群数目/总数目，默认0.4；

float th2：默认5.991，和允许误差最大值有关；

------------------------------------------------------------------------

find

vector<bool> &vbInliers：记录哪些点正确匹配了；

int &nInliers：正确匹配的点数目；

------------------------------------------------------------------------

iterate

int nIterations：最大迭代次数；

bool &bNoMore：1-已经所有参考点都判断过了；0-没把所有参考点都判断过（但可能拟合结果已经足够准确了）；

vector<bool> &vbInliers：记录哪些点正确匹配了；

int &nInliers：正确匹配的点数目。

------------------------------------------------------------------------

3.2 私有函数

当你阅读明白其算法原理后，从iterate()函数进入按调用顺序一步步看注释就可以看明白了。

**4、重要算法和流程**

算法程序流程图如下（初始化部分省略）：

主流程：

Compute\_pose()：

**5、PnP算法介绍**

PnP全称Perspective-n-Point problem，n点透视问题，目的是利用给出的三维点的世界坐标和这些三维点在二维图像的投影确定相机的位置和方向。引用的文章核心思想是把n个三维点写成四个非共面虚拟点的加权和，这可以降低估算相机坐标下控制点的坐标，方法是用12\*12矩阵的特征向量加权和表示这些坐标并解出二次规划问题获得权重。

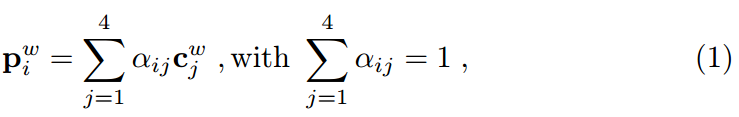
文献里，n个已知三维世界坐标的参考点（reference point）为：



4个控制点为（当需要区分时，上标w表示世界坐标，c表示相机坐标）：



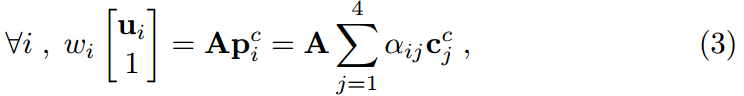
那么n个点可表示为：



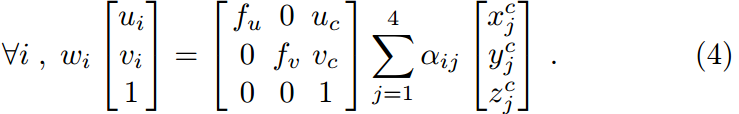


α是齐次重心坐标（homogeneous barycentric coordinates，是本算法的核心思想，文后有简单说明）。为提高稳定性，选择参考点的图心为一个控制点，剩下三个点和这个点一起组成一个向着数据主要方向的基。

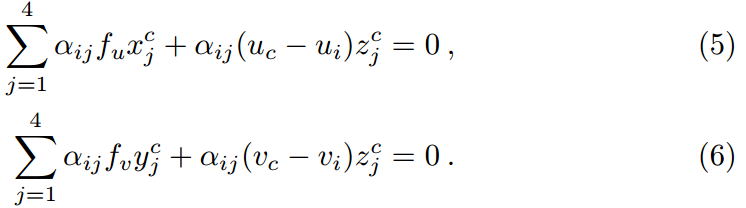
相机坐标三维点投影到二维上为：



其中，A为相机内参校准矩阵，u为投影到二维的图像上坐标，w为标量投影参数。（3）可写为：



这里未知参数是控制点的12个坐标（四个点每个点xyz坐标）和n个投影参数wi（编者问：为何这个未知参数不是αij和wi？四个点不是自己确定的吗？编者答：在确定四个点时是在世界坐标下，此时α可以全计算出来，而且从世界坐标变到相机坐标α不变，这里求的是四个控制点的相机坐标）。把第三行代入前两行消去wi可得两个方程（注意求和包含了等号左边所有部分）：



式(5)和(6)可以总结为：

C:\Users\XUEWUY~1\AppData\Local\Temp\1494226674(1).png

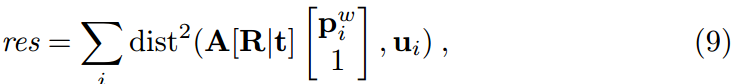
这里C:\Users\XUEWUY~1\AppData\Local\Temp\1494226717(1).png，M是2n×12矩阵，系数从式5和6可推出。式7的解或者属于空空间，或者属于核，并表示成：



vi是M的右奇异矩阵的列向量，可通过求MTM的特征向量（null eigenvector）求得。求MTM的计算量是整个算法里最大的，但是在整个过程只要求一次。

（然后是一大段说明为啥N可能是1到4，我看不明白什么意思）用透视相机模型（perspective camera model），至少6个参考点的情况，MTM的空空间（null space）维数N是1；如果用仿射相机模型（affine camera model），维数变成4，因为四个控制点的深度不清楚；由于焦距较大的透视相机会被仿射模型近似，因此维数N是不能预知的，但是基本在1到4范围内。

不预先猜想N是多少，而是直接计算N为1到4的重投影误差保留其最小值：

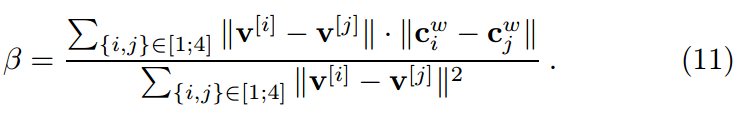


N=1：

此时式8只有x=βv，令是v对应 的子向量。保持控制点的距离：

C:\Users\XUEWUY~1\AppData\Local\Temp\1494234732(1).png

由于控制点的距离已知，则β可这样求（编者注：为啥要上下乘以左边分子左边的部分？）：



N=2：

此时式8有x=β1v1+β2v2。距离限制变为：

C:\Users\XUEWUY~1\AppData\Local\Temp\1494235098(1).png

两个β只出现在二次项，我们可以用密码学里的线性化技术求解。即求解 （其中 ）：

C:\Users\XUEWUY~1\AppData\Local\Temp\1494235462(1).png

其中L是v1和v2组成的6×3矩阵，ρ是6维距离向量||ciw-cjw||2，用L的伪逆矩阵求解并对βa选择符号使得pic的z值为正。产生的β值可被式11改善使得 。

N=3：

和N=2类似，这时L式一个6×6矩阵，用v1v2v3组成，β变成6维向量。

N=4：

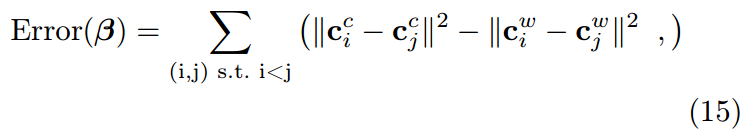
此时有四个β，但是方程会认为βab=βaβb有十个变量，而点数仍然6个，所以需要一些技巧，用重线性化技术解决。βab的解是一次齐次线性系统的解，正确的系数通过引入新二次等式后线性化解得的，新二次等式是通过交换律得到的：



Abcd与a‘b’c‘d‘为排序不同的四个相同的数值。即添加一些二次方程的条件。

当参考点的矩阵（什么矩阵？）有很小的特征值，我们需要三个控制点，M的维数为2n×9和vi是9维。解算方法和上面的基本一致，不一样的地方是前面N=4用到的方法需要用在N≥3上。

我们再对β=[β1β2β3β4]T进行优化。使用高斯牛顿法最小化：



世界坐标下控制点的距离已知，而相机坐标下控制点表示为：



由于这个优化只对四个β参数，所以其计算消耗独立于三维到二维的参考点数，一般十次迭代内足够。

Ransac算法（Random Sample Consensus）：

在文献里出现在实验章。用ransac选择出较好的7个匹配点的子集上，再用PnP解算姿态。Ransac算法思想就是在较大的候选点集合中随机选择一定数目的点拟合出一个模型，判断总集合中符合这个模型的点有多少，若符合的点数足够多，则认为这个模型足够好，否则重新选择点拟合模型直到符合的点数达标。如图2所示点，从肉眼观察可以猜想到用直线拟合最合理的线时绿色的，但是由于不匹配的点数目较多，直接用最小二乘法得到的直线是红色的，而通过ransac算法则得到更符合正确模型的绿色直线。

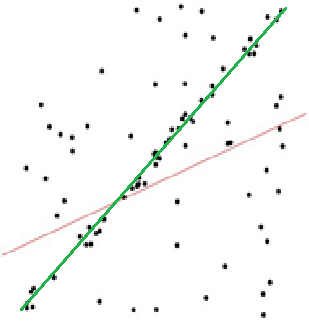


图2 ransac简单例子

关于重心坐标的简单说明：

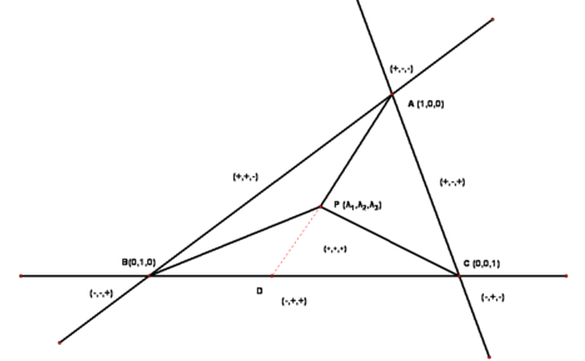
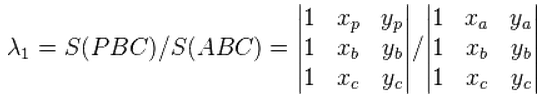


图3 重心坐标（二维）

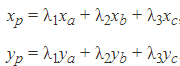
图3的二维三角形，对任意点p仅有唯一λ：

（1-1）

Abc为顶点在直角坐标系的坐标，λ为重心坐标。已知P点的直角坐标求λ的求法：

（1-2）

S为面积，但是要注意符号，若所有点都在基点包含的区域里那全为正，其他位置的符号参考图2。反过来已知λ求p的直角坐标：

（1-3）

其中重心坐标有仿射不变性。三维用四面体也有类似结论。推导方式也很简单，把其中一点a和p连线与bc相交于p’，过p’作ab的平行线，用相似三角形可证三个系数和为1。唯一注意二维上的点不能共线，三维的点不能共面。

而orb-slam所用的方法更直接，用下式写成关于α的矩阵方程形式求α：

 （1-4）

参考文献：

[1] Mur-Artal R, Montiel J M M, Tardos J D. ORB-SLAM: a versatile and accurate monocular SLAM system[J]. IEEE Transactions on Robotics, 2015, 31(5): 1147-1163.

[2] Lepetit V, Moreno-Noguer F, Fua P. Epnp: An accurate o (n) solution to the pnp problem[J]. International journal of computer vision, 2009, 81(2): 155-166.