

h_立青人韦

博客园

首页

新随笔

联系

订阅

管理

随笔 - 7 文章 - 0 评论 - 5

orb_slam代码解析(2)Tracking线程

在这篇文章里我们主要开始对跟踪线程进行介绍：

在[orb_slam整体编程思路及代码解析\(1\)](#)中我们发现，Tracking线程的入口是TrackMonocular，其中GrabImageMonocular返回位姿。

FUNCTION1:Tracking的构造函数

默认把跟踪状态设为NO_IMAGES_YET，定位跟踪模式，默认的其他参数，诸如字典，图像画布，地图画布，地图，关键帧数据库等都是system.cc类里定义的对象。也从配置文件中传入了相机的内参，图像校正系数，帧率，图像金字塔和角点提取的基本参数，这些参数都是这个类的元素等。

tracking过程都会用到mpORBExtractorLeft作为特征点提取器，在单目初始化的时候，会用mpInORBExtractor来作为特征点提取器，两者的区别是后者比前者最多提出的点数多一倍。

FUNCTION1.1:ORBExtractor的构造函数

是构造函数，传入features_num最多提取的特征点的数量，scale_factor金字塔图像之间的尺度参数，levels_num金字塔的层数，default_fast_threshold默认fast 角点检测的时候的阈值，为了防止用默认阈值fast角点检测检测的特征数过少，添加设置min_fast_threshold最小的fast特征检测阈值，以保证检测的特征数目。每一层都有一些属性参数，比如mvScaleFactor、mvLevelSigma2、mvInvScaleFactor、mvInvLevelSigma2等，以及给每层分配待提取的特征数，具体通过等比数列求和的方式，求出每一层应该提取的特征数，把每一层的特征点数都放在mnFeaturesPerLevel中，值得注意的是第零层的特征点数是 $n\text{features} \times (1 - 1/\text{scaleFactor}) / (1 - (1/\text{scaleFactor})^n\text{levels})$ ，然后下一层是上一层点数的 $1/\text{scaleFactor}$ 倍。以此类推，最后一层兜底。然后复制训练的模板，在计算描述子的时候会用到。最后通过求x坐标对应在半径为HALF_PATCH_SIZE的圆上的y坐标，标出了一个圆形区域用来求特征点方向。相关内容可以参考[Oriented FAST and Rotated BRIEF](#)。

FUNCTION2:GrabImageMonocular

这个函数先把图片转换成了灰度图像，然后根据跟踪的状态构造关键帧，再是进行跟踪得到当前帧的位姿。

FUNCTION2.1:Frame的构造函数

传入图像，时间戳，特征点提取器，字典，内参矩阵等参数来构造关键帧，首先把要构造金字塔的相关参数给Frame类中的跟金字塔相关的元素。然后提取ORB特征，这一步调用了重载了函数调用操作符operator()。

FUNCTION2.1.1:operator()

传入的图像必须是灰度图像，然后构造图像金字塔。相关内容可以参考：[ORB SLAM2 源码阅读 ORB_SLAM2::ORBExtractor](#)。

FUNCTION2.1.1.1:ComputePyramid(image)

这个函数通过传入的图像来构造nlevel层金字塔，level层是level-1层用resize函数得到大小为level-1层大小的scale倍的线性插值后的图像，为了方便做一些卷积计算，所以用copyMakeBorder函数来做边界填充。填充类型是BORDER_REFLECT_101(反射101)，例如：gfedcb|abcdefg|hgfedcba。

FUNCTION2.1.1.2:ComputeKeyPointsOctTree(allKeypoints)

这个函数为了计算金字塔每一层的兴趣点，找到FAST关键点，在操作上是依次针对图像金字塔的每一层图像进行的，首先在图像四周去掉长度为EDGE_THRESHOLD-3个单位的像素点的边界，对去掉边界后的图像网格化，每个窗口的大小为w个像素点的大小，然后依次在划分好的窗口中提取FAST关键点，这样做的目的是为了使得每个网格都有特征，从而使得特征点在图像上的分布相对均匀点。如果存在有的窗口中提取的特征点数为0，则降低阈值继续提取，然后对提取出了的关键点换算出其位于（level层的被裁掉边界图像）的位置，并每个窗口中的关键点存入

公告

昵称：h_立青人韦
园龄：2年2个月
粉丝：4
关注：9
+加关注

2019年1月						
<	日	一	二	三	四	五
	30	31	1	2	3	4
	6	7	8	9	10	11
	13	14	15	16	17	18
	20	21	22	23	24	25
	27	28	29	30	31	1
	3	4	5	6	7	8

搜索

🔍

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔档案

2018年10月 (1)
2018年4月 (4)
2018年1月 (1)
2017年5月 (1)

最新评论

- Re:orb_slam代码解析(2)Tracking线程
@h_立青人韦 代码是orb_slam2中文件 LocalMapping.cc, 函数void LocalMapping::CreateNewMapPoints()中的这句话x3D = x3D.rowRa.....
--Max_诸葛小亮
- Re:orb_slam代码解析(2)Tracking线程
@Max_诸葛小亮我不太清楚你说的是代码中的哪块内容。但如果是4维向量代表空间点，则是齐次坐标的形式，在齐次坐标中坐标的每个分量同乘一个非零常数，仍表示同一个点，所以除以向量的第四个元素只是强制性最后.....

vToDistributeKeys容器中，vToDistributeKeys容器就暂时保存着level层图像的关键点。然后将这些特征点送入DistributeOctTree函数，剔除一些关键点。将剔除后留下的关键点存入allKeypoints[level]容器中。

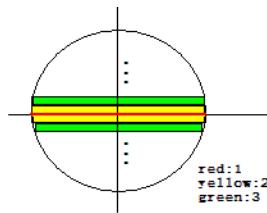
```
FUNCTION2.1.1.2.1:DistributeOctTree(vToDistributeKeys, minBorderX, maxBorderX,
minBorderY, maxBorderY,mnFeaturesPerLevel[level], level);
```

先用(maxX-minX)/(maxY-minY)来确定四叉树有几个初始节点，这里有bug，如果输入的是一张宽高比 小于 0.5 的图像，nIni 计算得到 0，下一步计算 hX 会报错，例如 round(640/480)=1,所以只有一个初始节点，(UL,UR,BL,BR)就会分布到被裁掉边界后的图像的四个角。把所有的关键点分配给属于它的节点，当节点所分配到的关键点的个数为1时就不再进行分裂，当节点没有分配到关键点时就删除此节点。再根据兴趣点分布,利用四叉树方法对图像进行划分区域，当bFinish的值为true时就不再进行区域划分，首先对目前的区域进行划分，把每次划分得到的有关键点的子区域设为新的节点，将nToExpand参数加一，并插入到节点列表的前边，删除掉其父节点。只要新节点中的关键点的个数超过一个，就继续划分，继续插入列表前面，继续删除父节点，直到划分的子区域中的关键点的个数是一个，然后迭代器加以移动到下一个节点，继续划分区域。当划分的区域即节点的个数大于关键点的个数或者分裂过程没有增加节点的个数时就将bFinish的值设为true，不再进行划分。如果以上条件没有满足，但是满足((int)INodes.size()>nToExpand*3)>N, 表示再分一次即将结束，所以开始按照特征点的数量对节点进行排序，特征点数多的节点优先划分，知道节点数量满足。vSizeAndPointerToNode 是前面分裂出来的子节点 (n1, n2, n3, n4) 中可以分裂的节点。按照它们特征点的排序，先从特征点多的开始分裂，分裂的结果继续存储在 INodes 中。每分裂一个节点都会进行一次判断，如果 INodes 中的节点数量大于所需要的特征点数量，退出整个 while(!bFinish) 循环，如果进行了一次分裂，并没有增加节点数量，不玩了，退出整个 while(!bFinish) 循环。取出每一个节点(每个区域)对应的最大响应点，即我们确定的特征点。NOTED:因为经过FAST提取出的关键点有很多，当划分的子区域一旦大于 mnFeaturesPerLevel[level](根据nfeatures算出的每一个level层最多的特征点数)的时候就不再进行区域划分了，所以每个区域内(节点)的关键点数会很多，取出响应值最大的那个就是我们想要的特征点。这个函数的意义就是根据mnFeaturesPerLevel,即该层的兴趣点数,对特征点进行剔除，根据Harris角点的score进行排序，保留正确的。

经过以上步骤，我们提出来level层在无边界图像中的特征点，并给特征点条件边界补偿及尺度信息。

```
FUNCTION2.1.1.2.2:computeOrientation(mvImagePyramid[level], allKeypoints[level],
umax)->IC_Angle(image, keypoint->pt, umax);
```

结合以上的论文我们知道要计算特征点的角度，我们要在一个圆域中算出 m_{10} 和 m_{01} ，计算步骤是先算出中间红线的 m_{10} ，然后在平行于x轴算出 m_{10} 和 m_{01} ，一次计算相当于图像中的同个颜色的两个line。



建立一个MAT容器descriptors，并关联于mDescriptors，其大小为所有层特征点的个数×32，循环计算各尺度图像中的特征点的描述子，依次顺序存放于mDescriptors中。

```
FUNTIION2.1.1.3:computeDescriptors(workingMat, keypoints, desc,
pattern)->computeOrbDescriptor(keypoints[i], image, &pattern[0], descriptors.ptr((int)i));
```

ORB选择了BRIEF作为特征描述方法，但是我们知道BRIEF是没有旋转不变性的，所以我们需要给BRIEF加上旋转不变性，把这种方法称为“Steer BRIEF”。之前我们在计算特征点的时候已经计算出了每个特征点的角度，这个角度对应着一个旋转矩阵：

$$R_\theta = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

bit_pattern_31_是个一维数组，里面放了512个（偏差）点（256个点对）。并把这些点给类的point类型的pattern数组。并把这些点都经过相应的旋转。即Sx=RS。

-h_立青人韦

3. Re:orb_slam代码解析(2)Tracking线程
博主您好，非常感谢您刚才的回答。但我还有一个问题想请教一下。我在ORB_SLAM2的代码中发现，利用SVD分解求出的四维坐标并不是空间点的三维坐标而是需要用前三维除以第四维，归一化得到的结果作为空间点.....

--Max_诸葛小亮

4. Re:orb_slam代码解析(2)Tracking线程
@Max_诸葛小亮因为第三行可以由前两行线性表示，故而省略...

-h_立青人韦

5. Re:orb_slam代码解析(2)Tracking线程
您好，请问在
FUNCTION2.2.1.2.3.1:Triangulate中，为什么在三角化求解阶段把DLT分解右侧的向量第三行给省略了？这是什么原因？

--Max_诸葛小亮

阅读排行榜

1. orb_slam代码解析(2)Tracking线程(2977)
2. 针孔的相机成像模型(1149)
3. orb_slam代码解析(3)LocalMapping线程(415)
4. orb_slam代码解析(4)LocalClosing线程(351)
5. ubuntu14.04+ros_indigo+lsd slam(310)

评论排行榜

1. orb_slam代码解析(2)Tracking线程(5)

推荐排行榜

1. orb_slam代码解析(2)Tracking线程(1)
2. orb_slam代码解析(3)LocalMapping线程(1)
3. Count bits set in parallel (查找32位整形数中置1的个数) (1)

```

const uchar* center = &img.at<uchar>(cvRound(kpt.pt.y), cvRound(kpt.pt.x));
const int step = (int)img.step;

#define GET_VALUE(idx) \
center[cvRound(pattern[idx].x*b + pattern[idx].y*a)*step + \
cvRound(pattern[idx].x*a - pattern[idx].y*b)]

```

center[0]给出了特征点的坐标，偏差点的表示是 $(y,x)^T$ ，乘以旋转矩阵后就是上式中的GET_VALUE获取相应的点。描述子矩阵中的纵坐标的个数是特征点的个数，横坐标表示的是一个特征点的描述子总共有 32×8 (256) 位。

最后把得到的各个尺度下的特征点都换算成在0层图像下的坐标，并存放在mvKeys中。到这就完成了ORB特征点和描述子的提取。

现在回到Frame的构造函数上来，我们调用OpenCV的矫正函数矫正orb提取的特征点，进行畸变矫正，找到关键点实际应该在普通摄像头中的位置，如果矫正矩阵等于0的话，就不进行矫正，否则将进行矫正，并把矫正后的特征点的位置存储在mvKeysUn里。并处置好未用到的立体信息等。当传入第一帧图像或者标定矩阵发生变化时，我们需要计算矫正后的边界就是把未矫正图像的四个角点矫正后找到他们在矫正后图像中的点的位置就可以确认边界，最后把传进来的图片分割成 48×64 的网格，根据特征点的位置分在不同的网格里。每个特征点在mvKeysUn里都有编号，然后用mGrid[nGridPosX][nGridPosY]来存储在各个各自里的特征点的编号。

这样，我们就完成了从图像到信息帧(Frame)的构造，主要是提取图像的特征点和特征点的描述子，把图像初始化mCurrentFrame。

现在回到GrabImageMonocular，进行tracking过程。

FUNCTION2.2:Track()

当系统第一次运行，或者被复位，就进行初始化。

FUNCTION2.2.1:MonocularInitialization()

初始化需要两帧，第一帧为参考帧，这两帧的特征点数都得大于100，跟踪器用mVbPrevMatched来接管参考帧的特征点，并用参考帧、 $\sigma=1.0$ 、iterations:200注册了初始器，如果当前帧特征点太少，重新构造初始器，因此只有连续两帧的特征点个数都大于100时，才能继续进行初始化过程。随后我们注册了一个ORB匹配器，这个匹配器的初始化参数包括最佳得分和第二得分的比例(0.9)，以及是否执行角度检测(true)。首先执行这个匹配器的SearchForInitialization用来寻找参考帧与第二帧之间的匹配点的数量。

FUNCTION2.2.1.1:SearchForInitialization

取出金字塔0层的图像的特征点，亦只对原图像处理。

FUNCTION2.2.1.1.1:GetFeaturesInArea

这个函数是找到以 x,y 为中心，边长为 $2r$ 的方形内且在 $[minLevel, maxLevel]$ 的特征点，返回满足条件的特征点的序号，在具体执行上，找到这个区域占的所有grid，然后把这些grid的特征点都取出来再与 x,y 做差跟边长 r 做比较，如果小的话就证明在边长为 $2r$ 的范围内，返回该特征点的序号，调用是通过帧F2进行的，传入的 x,y 是F1中的特征点的坐标，这样就找到了F1的0层特征点 x,y 在F2的0层中的以 x,y 为中心的 $windowSize \times windowSize$ 范围内的特征点的序号。

再找到F1中特征点的描述子与上面找到的F2中所有的序号对应的特征点的描述子对比，找到最佳匹配，和次佳匹配。然后根据阈值(在注册匹配器时直接被赋值)和角度投票剔除误匹配。最佳匹配点要明显优于次佳匹配才是好的匹配，如果满足上述阈值要求就更行匹配向量(如:tracking里的mVbIniMatches[i1]=bestIdx2；其中i1为F1的特征点的序号，为F2中和特征点i1相匹配特征点的序号)，然后进行角度投票，将F1特征点的角度和F2特征点的角度做差，存入角度列表。最后找到3个得票数最高的角度，将其他的角度对应的匹配关系从匹配向量中剔除。最后将真正有匹配关系的更新到mVbPrevMatched中。并返回匹配对的数量。

如果这两帧之间的匹配对数小于100，那么就得删除初始化器重新初始化，函数返回。否则，通过H模型或F模型进行单目初始化，得到两帧间相对运动、初始MapPoints。

FUNCTION2.2.1.2:Initialize

Initializer里的mVbMatches12是一个元素为Match的向量，Match的数据结构是pair，mVbMatches12只记录Reference到Current匹配上的特征点对，mVbMatched1记录Reference Frame的每个特征点在Current Frame是否有匹配的特征点，先用tracking里的mVbIniMatches更新上述两个容器的值。再新建一个容器vAllIndices，生成0到N-1的数作为特征点的索引，在所有匹配特征点对中随机选择8对匹配特征点为一组，共选择mMaxIterations组，结果放在mVbSets中。用于

FindHomography和FindFundamental求解，在选择的过程中被选择过的索引都会在容器中被删除，确保同一个点不会被重复选到，做一次迭代之后，vAvailableIndices的值会被vAllIndices更新。再调用多线程分别用于计算fundamental matrix和homography。在线程调用成员函数时，需要同时传成员函数的对象，和这个函数的传参。

FUNCTION2.2.1.2.1:FindHomography

假设场景为平面情况下通过前两帧求取Homography矩阵(current frame 2 到 reference frame 1),并得到该模型的评分,将mvKeys1和mvKey2归一化到均值为0, 一阶绝对矩为1, 归一化矩阵分别为T1、T2。即 $[x',y]^T=T1[x,y]^T$ 。

- 特征点归一化，坐标均值为0，一阶绝对矩为1

$$\text{mean_x} = (\sum_{i=0}^N u_i)/N \quad \text{mean_y} = (\sum_{i=0}^N v_i)/N \quad (1)$$

$$\text{mean_devx} = (\sum_{i=0}^N |u_i - \text{mean_x}|)/N \quad \text{mean_devy} = (\sum_{i=0}^N |v_i - \text{mean_y}|)/N \quad (2)$$

$$sX = 1/\text{mean_devx} \quad sY = 1/\text{mean_devy} \quad (3)$$

$$T = \begin{bmatrix} sX & 0 & -\text{mean}_x * sX \\ 0 & sY & -\text{mean}_y * sY \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

然后利用RANSAC算法，找出得分最高的Homography矩阵。

FUNCTION2.2.1.2.1.1:ComputeH21

这里的单应矩阵H_N是利用归一化后的像素坐标进行操作的。在程序中vt是v的转置，所以最小的特征值对应的特征向量是vt的最后一行。

- Homograph矩阵求解（归一化4点算法）

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{right} = \lambda \begin{bmatrix} h1 & h2 & h3 \\ h4 & h5 & h6 \\ h7 & h8 & h9 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{right} \Rightarrow x' = \lambda Hx$$

DLT求解: $x' \times Hx = 0 \Rightarrow Ah = 0 \quad (6)$

$$\text{其中: } A = \begin{bmatrix} 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \\ -x & -y & -1 & 0 & 0 & 0 & xx' & yy' & x' \end{bmatrix} \Rightarrow \begin{array}{l} \text{特征值分解: } A^T A \\ \text{最小特征值对应的特征向量} \\ h = [h1 \ h2 \ h3 \ h4 \ h5 \ h6 \ h7 \ h8 \ h9]' \end{array}$$

恢复到原始的均值和尺度。

$$\begin{aligned} X_N^1 &= T_1 X^1 \Rightarrow X^1 = T_1^{-1} X_N^1 \\ X_N^2 &= T_2 X^2 \Rightarrow X_2 = T_2^{-1} X_N^2 \\ X^2 &= \lambda H X^1 \Rightarrow T_2^{-1} X_N^2 = \lambda H T_1^{-1} X_N^1 \Rightarrow X_N^2 = \lambda T_2 H T_1^{-1} X_N^1 \\ H_N &= T_2 H T_1^{-1} \Rightarrow H = T_2^{-1} H_N T_1 \end{aligned}$$

FUNCTION2.2.1.2.1.2:CheckHomography

利用重投影误差为当次RANSAC的结果评分,把内点保存在vbMatchesInliers，得分所有点是阈值与归一化误差的差的和。很好理解，阈值是固定值，那么归一化误差越小，差值就越大，得分就越高。

Parallel computation of the two models:
Compute in parallel threads a homography \mathbf{H}_{cr} and a fundamental matrix \mathbf{F}_{cr} :

$$\mathbf{x}_c = \mathbf{H}_{cr} \mathbf{x}_r \quad \mathbf{x}_c^T \mathbf{F}_{cr} \mathbf{x}_r = 0 \quad (1)$$

with the normalized DLT and 8-point algorithms respectively as explained in [2] inside a RANSAC scheme. To make homogeneous the procedure for both models, the number of iterations is prefixed and the same for both models, along with the points to be used at each iteration, 8 for the fundamental matrix, and 4 of them for the homography. At each iteration we compute a score S_M for each model M (H for the homography, F for the fundamental matrix):

$$S_M = \sum_i (\rho_M(d_{cr}^2(\mathbf{x}_c^i, \mathbf{x}_r^i, M)) + \rho_M(d_{rc}^2(\mathbf{x}_c^i, \mathbf{x}_r^i, M)))$$

$$\rho_M(d^2) = \begin{cases} \Gamma - d^2 & \text{if } d^2 < T_M \\ 0 & \text{if } d^2 \geq T_M \end{cases} \quad (2)$$

where d_{cr}^2 and d_{rc}^2 are the symmetric transfer errors [2] from one frame to the other. T_M is the outlier rejection threshold based on the χ^2 test at 95% ($T_H = 5.99$, $T_F = 3.84$, assuming a standard deviation of 1 pixel in the measurement error). Γ is defined equal to T_H so that both models score equally for the same d in their inlier region, again to make the process homogeneous.

We keep the homography and fundamental matrix with highest score. If no model could be found (not enough inliers), we restart the process again from step 1.

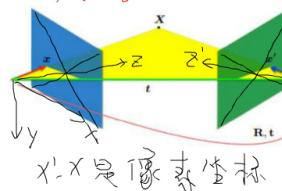
好了，然后循环mMaxIterations迭代找出最高的得分对应的单应矩阵即为所求。

FUNCTION2.2.1.2.2:FindFundamental

假设场景为非平面情况下通过前两帧求取Fundamental矩阵(current frame 2 到 reference frame 1)，并得到该模型的评分，程序步骤与求解单应矩阵一致，这里给出算法：

■ 对极几何模型 (Fundamental Matrix)

$$\begin{aligned} \text{齐次化平面坐标} & \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \\ \text{坐标} & \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \end{aligned}$$



$$\text{刚体旋转 } \hat{x}' = R(\hat{x} - t) \quad (1)$$

$$\text{共平面 } (x - t)^T (t \times x) = 0 \quad (2)$$

$$\Rightarrow (\hat{x}'^T R)(t \times \hat{x}) = 0 \Rightarrow (\hat{x}'^T R)([t_x] \hat{x}) = 0 \quad (3)$$

$$\Rightarrow \hat{x}'^T (R[t_x]) \hat{x} = 0 \quad (3)$$

$$\text{图像坐标系转相机坐标系:}$$

$$\hat{x} = k^{-1}x \quad \hat{x}' = k^{-1}x' \quad (4)$$

$$\Rightarrow x'^T F x = 0 \quad F = K'^{-T} E K^{-1} \quad (5)$$

■ Fundamental 矩阵求解 (归一化8点算法)

$$x'^T F x = 0 \quad F = \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix}$$

$$\Rightarrow Af = 0 \quad (1)$$

$$\text{其中: } A = [x'x \quad x'y \quad x' \quad y'x \quad y'y \quad y' \quad x \quad y \quad 1] \quad \Rightarrow \text{特征值分解分解: } A^T A$$

$$f = [f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7 \quad f_8 \quad f_9]^T \quad \Rightarrow \text{最小特征值对应的特征向量}$$

同样的将F回复到原来的均值和尺度。通过得分求出最佳的基础矩阵。

计算得分比例，选取某个模型， $RH = SH/(SH+SF)$ ，当 $RH > 0.40$ ，从H矩阵中恢复R,t，否则从F矩阵中恢复R,t。

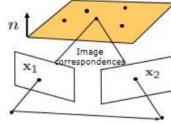
FUNCTION2.2.1.2.3:ReconstructH

■ Homograph 矩阵分解 (Faugeras SVD-based decomposition)

{ X_1, X_2 } 是相机坐标系匹配的特征点

$$aX + bY + cZ = d \text{ 即 } \frac{1}{d}n^T X = 1 \text{ 表示3D点共同所在的平面, } N \text{ 为平面法向量}$$

$$X = \lambda_1 X_1 \text{ 表示 } X_1 \text{ 在平面上对应的3D点, 世界坐标系与第一个相机坐标系重合}$$



$$\lambda_2 X_2 = RX + t \\ = R(\lambda_1 X_1) + t$$

将所有的3D点共平面这个约束引入上式:

$$\lambda_2 X_2 = R(\lambda_1 X_1) + t \cdot \frac{1}{d}n^T(\lambda_1 X_1)$$

$$\Rightarrow X_2 = \lambda \left(R + \frac{1}{d}tn^T \right) X_1$$

G是相机坐标系匹配的H
{ x_1, x_2 } 是图像坐标系匹配特征点, 则:

$$x_2 = \lambda G X_1 \quad G = KHK^{-1}$$

$$\text{令: } A = dR + tn^T$$

$$\text{对 } A \text{ 奇异值分解: } A = UAV^T$$

$$\text{则: } A = U^T AV = dU^T RV + (U^T t)(V^T n)^T$$

$$\text{考虑: } s = \det U \det V \quad s^2 = 1$$

$$\text{则: } A = s^2 d U^T RV + (U^T t)(V^T n)^T \\ = (sd)(sU^T RV) + (U^T t)(V^T n)^T$$

$$\text{令: } R' = sU^T RV, t' = U^T t, n' = V^T n, d' = sd$$

$$\text{则: } A = d'R' + t'n'^T$$

引入s是为了说明 R' 与 R , d' 与 d 存在符号取反的可能

$$A = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} = d'R' + t'n'^T \quad (1)$$

$$\text{取: } e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\text{则: } n' = \begin{bmatrix} x_1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ x_2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ x_3 \end{bmatrix} = x_1 e_1 + x_2 e_2 + x_3 e_3$$

$$x_1^2 + x_2^2 + x_3^2 = 1 \quad (2)$$

(1) 式变为:

$$[d_1 e_1 \quad d_2 e_2 \quad d_3 e_3] = [d'R' e_1 \quad d'R' e_2 \quad d'R' e_3] + [t' x_1 \quad t' x_2 \quad t' x_3] \quad (3)$$

(3) 式也可以拆成3个等式:

$$\begin{cases} d_1 e_1 = d'R' e_1 + t' x_1 \\ d_2 e_2 = d'R' e_2 + t' x_2 \\ d_3 e_3 = d'R' e_3 + t' x_3 \end{cases} \quad (4)$$

$$\begin{cases} d_1 e_1 = d'R' e_1 + t' x_1 \\ d_2 e_2 = d'R' e_2 + t' x_2 \\ d_3 e_3 = d'R' e_3 + t' x_3 \end{cases} \quad (5)$$

$$\begin{cases} d_1 e_1 = d'R' e_1 + t' x_1 \\ d_2 e_2 = d'R' e_2 + t' x_2 \\ d_3 e_3 = d'R' e_3 + t' x_3 \end{cases} \quad (6)$$

(4) (5) (6) 中每两个式子消去 t' 可得:

$$\begin{cases} d'R'(x_2 e_1 - x_1 e_2) = d_1 x_2 e_1 - d_2 x_1 e_2 \\ d'R'(x_3 e_2 - x_2 e_3) = d_2 x_3 e_2 - d_3 x_2 e_3 \\ d'R'(x_1 e_3 - x_3 e_1) = d_3 x_1 e_3 - d_1 x_3 e_1 \end{cases} \quad (7)$$

因为: $\|R'X\| = \|X\|$

对(7)式中三个式子的左右两边同时取范数可得:

$$\begin{cases} (d'^2 - d_2^2)x_1^2 + (d'^2 - d_1^2)x_2^2 = 0 \\ (d'^2 - d_3^2)x_2^2 + (d'^2 - d_2^2)x_3^2 = 0 \\ (d'^2 - d_1^2)x_3^2 + (d'^2 - d_3^2)x_1^2 = 0 \end{cases} \quad (8)$$

对于(8)式如果令:

$$d'^2 - d_1^2 = a \quad d'^2 - d_2^2 = b \quad d'^2 - d_3^2 = c$$

则(8)式简写为:

$$\begin{bmatrix} b & a & 0 \\ 0 & c & b \\ c & 0 & a \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \end{bmatrix} = 0 \text{ 则 } \det \begin{pmatrix} b & a & 0 \\ 0 & c & b \\ c & 0 & a \end{pmatrix} = 0$$

$$\text{则 } abc = 0$$

$$(d'^2 - d_1^2)(d'^2 - d_2^2)(d'^2 - d_3^2) = 0 \quad (9)$$

$$\text{因为: } d_1 \geq d_2 \geq d_3 \quad (10)$$

对于(9)式, 可分成以下三种情况:

$$\begin{cases} d_1 \neq d_2 \neq d_3 \\ d_1 = d_2 \neq d_3 \text{ 或 } d_1 \neq d_2 = d_3 \\ d_1 = d_2 = d_3 \end{cases}$$

这三种情况下均可以得到: $d' = \pm d_2$

现对第一种情况用反证法进行证明:

如果: $d' = \pm d_1$ 或 $d' = \pm d_3$

根据(8)式可得:

$$\begin{cases} x_1 = 0 \\ (d_1^2 - d_3^2)x_2^2 + (d_1^2 - d_2^2)x_3^2 = 0 \\ d_1 > d_2 > d_3 \end{cases}$$

推出: $x_1 = x_2 = x_3 = 0$ 与 $x_1^2 + x_2^2 + x_3^2 = 1$ 矛盾

经过上一页的说明, (3)式的解分以下几种情况:

$$d' = d_2 > 0 \quad \begin{cases} d_1 \neq d_2 \neq d_3 \\ d_1 = d_2 \neq d_3 \text{ 或 } d_1 \neq d_2 = d_3 \\ d_1 = d_2 = d_3 \end{cases} \quad (11) \quad (12) \quad (13)$$

$$d' = -d_2 < 0 \quad \begin{cases} d_1 \neq d_2 \neq d_3 \\ d_1 = d_2 \neq d_3 \text{ 或 } d_1 \neq d_2 = d_3 \\ d_1 = d_2 = d_3 \end{cases} \quad (14) \quad (15) \quad (16)$$

对于 (11) 式这种情况:

根据 (8) 式三个方程可解得:

$$\mathbf{n}' = \begin{cases} x_1 = \varepsilon_1 \frac{\sqrt{d_1^2 - d_2^2}}{\sqrt{d_1^2 - d_3^2}} \\ x_2 = 0 \\ x_3 = \varepsilon_2 \frac{\sqrt{d_2^2 - d_3^2}}{\sqrt{d_1^2 - d_3^2}} \\ \varepsilon_1, \varepsilon_2 = \pm 1 \end{cases} \quad (17)$$

将 \mathbf{n}' 带入 (5) 式可得:

$$R'e_2 = e_2$$

因此可以得到 R' 的形式为:

$$R' = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \quad (18)$$

将 (18) (19) 带入 (3) 可得:

$$t' = (d_1 - d_3) \begin{pmatrix} x_1 \\ 0 \\ x_3 \end{pmatrix} \quad (20)$$

将 (17) 和 (18) 带入 (7) 式第三个可得 (18) 式中的: $\sin\theta \cos\theta$

$$d' \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} -x_3 \\ 0 \\ x_1 \end{bmatrix} = \begin{bmatrix} -d_1 x_3 \\ 0 \\ d_3 x_1 \end{bmatrix} \Rightarrow \begin{cases} \sin\theta = \frac{(d_1 - d_3)}{d_2} x_1 x_3 = \varepsilon_1 \varepsilon_3 \frac{\sqrt{(d_1^2 - d_2^2)(d_2^2 - d_3^2)}}{(d_1 + d_3)d_2} \\ \cos\theta = \frac{d_3 x_1^2 + d_1 x_3^2}{d_2} = \frac{d_1 d_3 + d_2^2}{(d_1 + d_3)d_2} \end{cases} \quad (19)$$

对于 (12) 式这种情况, (11) 情况的特例:

根据 (8) 式三个方程可解得:

$$\mathbf{n}' = \begin{cases} x_1 = 0 \\ x_2 = 0 \\ x_3 = \pm 1 \end{cases}$$

将 \mathbf{n}' 带入 (5) 式可得:

$$R' = I$$

带入 (3) 式可得:

$$t' = (d_3 - d_1)\mathbf{n}'$$

对于 (13) 式这种情况, (11) 情况的特例:

$$R' = I \quad t' = 0$$

\mathbf{n}' 未定义

对于 (14) 式这种情况:

根据 (8) 式三个方程可解得:

$$\mathbf{n}' = \begin{cases} x_1 = \varepsilon_1 \frac{\sqrt{d_1^2 - d_2^2}}{\sqrt{d_1^2 - d_3^2}} \\ x_2 = 0 \\ x_3 = \varepsilon_2 \frac{\sqrt{d_2^2 - d_3^2}}{\sqrt{d_1^2 - d_3^2}} \\ \varepsilon_1, \varepsilon_2 = \pm 1 \end{cases} \quad (17)$$

将 \mathbf{n}' 带入 (5) 式可得:

$$R'e_2 = -e_2$$

因此可以得到 R' 的形式为:

$$R' = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & -1 & 0 \\ \sin\theta & 0 & -\cos\theta \end{bmatrix} \quad (21)$$

将 (18) (19) 带入 (3) 可得:

$$t' = (d_1 + d_3) \begin{pmatrix} x_1 \\ 0 \\ x_3 \end{pmatrix} \quad (23)$$

将 (17) 和 (18) 带入 (7) 式第三个可得 (18) 式中的: $\sin\theta \cos\theta$

$$d' \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} -x_3 \\ 0 \\ x_1 \end{bmatrix} = \begin{bmatrix} -d_1 x_3 \\ 0 \\ d_3 x_1 \end{bmatrix} \Rightarrow \begin{cases} \sin\theta = \frac{(d_1 + d_3)}{d_2} x_1 x_3 = \varepsilon_1 \varepsilon_3 \frac{\sqrt{(d_1^2 - d_2^2)(d_2^2 - d_3^2)}}{(d_1 - d_3)d_2} \\ \cos\theta = \frac{d_3 x_1^2 - d_1 x_3^2}{d_2} = \frac{d_1 d_3 - d_2^2}{(d_1 - d_3)d_2} \end{cases} \quad (22)$$

对于 (15) 式这种情况, (14) 情况的特例:

根据 (8) 式三个方程可解得:

$$\mathbf{n}' = \begin{cases} x_1 = 0 \\ x_2 = 0 \\ x_3 = \pm 1 \end{cases}$$

将 \mathbf{n}' 带入 (5) 式可得:

$$R' = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

带入 (3) 式可得:

$$t' = (d_1 + d_3)\mathbf{n}'$$

对于 (16) 式这种情况, (14) 情况的特例:

根据公式 (1) : $A = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} = d'R' + t'n'^T$

根据 $d' = d_1 = d_2 = d_3$ 可得: $\begin{bmatrix} d' & 0 & 0 \\ 0 & d' & 0 \\ 0 & 0 & d' \end{bmatrix} = d'R' + t'n'^T \Rightarrow Id' = d'R' + t'n'^T$

取垂直于法向量 \mathbf{n}' 的向量 \mathbf{x} , 带入: $Id'\mathbf{x} = d'R'\mathbf{x} + t'n'^T\mathbf{x}$

$$\Rightarrow Rx = -x \quad \text{根据 household 变换: } R' = -I + 2n'n'^T$$

$$t' = -2d'n'$$

$d'=d2$ 和 $d'=-d2$ 分别对应 8 组 (R, t), 现在需要找到最合适的解。

FUNCTION2.2.1.2.3.1:CheckRT

首先要得到投影矩阵(直接从世界坐标系到图像坐标系), 以第一个坐标为世界坐标系, 我们得到的投影矩阵 $P1[k|0]$, 第二个坐标系的投影矩阵为 $P2=k[R|t]$, 第二个相机的光心在世界坐标系下的坐标, $O2=-R^T \times t$. 这个公式在这里解释一下, 在 ORB_SLAM 里, 位移向量 t_{cw} 的方向是从左下标到右下标的, 并且位于左下标坐标系下, R_{cw} 是从世界坐标系到相机坐标系的旋转, R^T 表示 R 的逆旋转, 首先我们把世界 t 变换到世界坐标系下的平移, 然后再加一个符号表示世界坐标系到相机坐标系的平移, 就是相机光心的位置。

FUNCTION2.2.1.2.3.1.1:Triangulate

ReconstructH函数中对t有归一化，这里三角化过程中恢复的3D点深度取决于 t 的尺度，但是这里恢复的3D点并没有决定单目整个SLAM过程的尺度，因为 CreateInitialMapMonocular函数对3D点深度会缩放，然后反过来对 t 有改变。

■ 三角化恢复3D点

(x, x') 为匹配特征点对

(P, P') 分别为它们的投影矩阵

$x = PX \quad x' = P'X$

$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$ 简写: $\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} - & p_0 & - \\ - & p_1 & - \\ - & p_2 & - \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$

DLT求解: $\begin{bmatrix} vp_2 - p_1 \\ p_0 - up_2 \\ up_1 - vp_0 \end{bmatrix} X = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ 一组匹配点: $\begin{bmatrix} vp_2 - p_1 \\ p_0 - up_2 \\ vp'_2 - p'_1 \\ p_0 - up'_2 \end{bmatrix} X = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

再通过3D点的深度不能为负，计算重投影误差不能过大，淘汰掉不符合条件的R, t。在同一个R,t下的各个匹配点之间的视差角会不一样，计算各视差角，最后到一个较大的视差角，并返回成功还原出3D点的数量。

最后，把还原出来的各个特征点对应3D点给mvIniP3D，并把各个特征点是否被成功三角化给vbTriangulated标志位容器。

FUNCTION2.2.1.2.4:ReconstructF

■ Fundamental 矩阵分解

$$E = k'^T F k$$

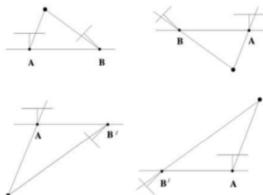
SVD分解: $E = U \Sigma V^T$ 令: $W = R_z \left(\frac{\pi}{2}\right) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$$E = [R|T] \begin{cases} R_1 = UWV^T \quad R_2 = UW^TV^T \\ T_1 = U_3 \quad T_2 = -U_3 \end{cases}$$

(R T) 选择: 3D点出现在两个相机前方最多的模型

统计四个模型中3D点在摄像头前方且投影误差小于阈值的3D点个数，以及每个模型下较大的视差角。

如果其中一个模型的视差角大于阈值，并且满足条件的3D点个数明显大于其它模型，那么这个模型就是最优选择。



设置帧的位姿，将初始化的第一帧作为世界坐标系，因此第一帧变换矩阵为单位矩阵，由Rcw和tcw构造Tcw，并赋值给mTcw，mTcw为世界坐标系到当前帧的变换矩阵。

FUNCTION2.2.1.3:CreateInitialMapMonocular

将三角化得到的3D点包装成MapPoints(Initialize函数会得到mvIniP3D, mvIniP3D是cv::Point3f类型的一个容器，是个存放3D点的临时变量，CreateInitialMapMonocular将3D点包装成MapPoint类型存入KeyFrame和Map中)。

FUNCTION2.2.1.3.1:KeyFrame

首先将初始参考帧和当前帧构成关键帧，该构造过程就是用帧，3D点，关键帧数据库在关键帧里注册。

FUNCTION2.2.1.3.2:ComputeBoW

将初始关键帧的描述子转为

BoW,mpORBvocabulary->transform(vCurrentDesc,mBowVec,mFeatVec,4);根据特征点算出mBowVec, mFeatVec。mBowVec的结构是map<WordId, WordValue>表示的是反向索引，key为wordId, value为tf-idf中的tf, (idf在建立字典的时候计算一次就行)。mFeatVec的结构是map<NodeId, std::vector<unsigned int>>表示的是正向索引，需要指定第m层，每幅图像对应一个正向索引，储存该图像生成BoW向量时曾经到达过的第m层上节点的编号，以及路过这个节点的那些特征的编号，程序中的层数指定的是4。更多关于DBow的内容请参见：DBow2库介绍。

FUNCTION2.2.1.3.3:AddKeyFrame

然后把这两个关键帧插入地图。

将3D点包装成MapPoints

FUNCTION2.2.1.3.4.1:MapPoint

用3D点构造MapPoint

FUNCTION2.2.1.3.4.2:AddMapPoint

Add MapPoint to KeyFrame, 表示该KeyFrame的哪个特征点可以观测到哪个3D点

FUNCTION2.2.1.3.4.3:AddObservation

表示该MapPoint可以被哪个KeyFrame的哪个特征点观测到

FUNCTION2.2.1.3.4.4:ComputeDistinctiveDescriptors

从众多观测到该MapPoint的特征点中挑选区分度最高的描述子,由于一个MapPoint会被许多相机观测到,因此在插入关键帧后,需要判断是否更新当前点的最适合的描述子,先获得当前点的所有描述子,然后计算描述子之间的两两距离,最好的描述子与其他描述子应该具有最小的距离中值,遍历观测到3d点的所有关键帧,获得orb描述子,并插入到vDescriptors中,获得这些描述子两两之间的距离,依次找到各个描述子到其它所有所有描述子之间的距离,每一组都获得中值,寻找最小的中值,最好的描述子,该描述子相对于其他描述子有最小的距离中值,简化来讲,中值代表了这个描述子到其它描述子的平均距离,最好的描述子就是和其它描述子的平均距离最小。最后返回此描述子。

FUNCTION2.2.1.3.4.5:UpdateNormalAndDepth

更新平均观测方向以及观测距离范围,由于一个MapPoint会被许多相机观测到,因此在插入关键帧后,需要更新相应变量,获得观测到该3d点的所有关键帧,对所有关键帧对该点的观测方向归一化为单位向量进行求和,除以所有关键帧数就是获得的平均观测方向。获得观测到该点的参考关键帧和3d点在世界坐标系中的位置,得到该点到参考关键帧相机的距离,预测其在金字塔中的层数,就可以获得其距离范围。

用3D点填补当前帧的结构。

FUNCTION2.2.1.3.4.6:AddMapPoint

在地图中添加该MapPoint。

FUNCTION2.2.1.3.5:UpdateConnections

首先获得该关键帧的所有MapPoint点,统计观测到这些3d点的每个关键帧与其它所有关键帧之间的共视程度,对每一个找到的关键帧,建立一条边,边的权重是该关键帧与当前关键帧公共3d点的个数。在没有执行这个函数前,关键帧只和MapPoints之间有连接关系,这个函数可以更新关键帧之间的连接关系,KFcounter是map<KeyFrame*,int>代表关键帧-权重,权重为其它关键帧与当前关键帧共视3d点的个数,在执行上获得关键帧对应的每一个3D点的能观测到该3D点的所有关键帧,并整理到KFcounter。新建一个容器vPairs, pair<int,KeyFrame*>将关键帧的权重写在前面,关键帧写在后面方便后面排序, vPairs记录与其它关键帧共视帧数大于th的关键帧,设定阈值为15,如果对应权重大于阈值,对这些关键帧建立连接,并更新其它KeyFrame的mConnectedKeyFrameWeights,更新其它关键帧与当前帧的连接权重,如果没有超过阈值的权重,则对权重最大的关键帧建立连接,vPairs里存的都是相互共视程度比较高的关键帧和共视权重,由大到小。更新该KeyFrame的mConnectedKeyFrameWeights,更新当前帧与其它关键帧的连接权重,并更新关键帧和权重。

FUNCTION2.2.1.3.6:GlobalBundleAdjustemnt

3D-2D 最小化重投影误差 $e = (u, v) - \text{project}(T_{cw} * P_w)$, 迭代20次。得到优化后的结果,把优化后的位姿传递给帧参与优化的帧,更新优化后的空间点。

将MapPoints的中值深度归一化到1,并归一化两帧之间变换, $x/z, y/z$, 将z归一化到1。

在把初始关键帧和当前关键帧给局部地图线程。把当前关键帧更新为最新关键帧,初始关键帧和当前关键帧给跟踪器的局部地图关键帧。把所有的地图点给跟踪器的局部关键点,把当前帧设定为参考关键帧,把当前关键帧更新为最新帧,更新Map线程相关信息。

到这,初始化完成,开始进行跟踪。

在viewer中有个开关menuLocalizationMode,有它控制是否ActivateLocalizationMode,并最终管控mbOnlyTracking, mbOnlyTracking等于false表示正常VO模式(有地图更新),mbOnlyTracking等于true表示用户手动选择定位模式。

2.2.2.PART1:得到初始位姿

● 2.2.2.1.mbOnlyTracking=false

如果初始化成功,因为Local Mapping线程可能会将关键帧中某些MapPoints进行替换,而tracking中需要用到mLastFrame,这里检查并更新上一帧中被替换的MapPoints,更新的是Fuse函数和SearchAndFuse函数替换的MapPoints,如果运动模型是空的或刚完成重定位。

FUNCTION2.2.2.1:TrackReferenceKeyFrame

将上一帧的位姿作为当前帧的初始位姿,通过BoW的方式在参考帧中找当前帧特征点的匹配点,优化每个特征点都对应3D点重投影误差即可得到位姿。在执行上:我们先将当前帧的描述子转化

为BOW向量，再注册一个匹配器，通过特征点的BoW加快当前帧与参考帧之间的特征点匹配。

FUNCTION2.2.2.1.1.1:SearchByBoW

这个函数是在同一个node下先从关键帧里面找到一个特征点，在依次遍历当前帧这个节点下的特征点，计算其描述子的距离，如果满足要求，就把关键帧里的特征点对应的3d点赋给当前帧里的特征点里对应的3d点，vpMapPointMatches[i],i表示当前帧特征点的index，值为对应的3d点。其中，函数lower_bound()在first和last中的前闭后开区间进行二分查找，返回大于或等于val的第一个元素位置。

当匹配的点达到15个以上的时候，更新匹配到的3D点，将上一帧的位姿作为当前帧位姿的初始值，用上一次的Tcw设置初值，在PoseOptimization可以收敛快一些。

FUNCTION2.2.2.1.1.2:PoseOptimization

3D-2D 最小化重投影误差 $e = (u,v) - \text{project}(Tcw * Pw)$ ，只优化Frame的Tcw，不优化MapPoints的坐标。开始优化，总共优化四次，每次优化后，将观测分为outlier和inlier，outlier不参与下次优化，由于每次优化后是对所有的观测进行outlier和inlier判别，因此之前被判别为outlier有可能变成inlier，反之亦然，其中外点的误差是需要另行计算的，因为g2o只会计算active edge的误差，除了前两次优化需要RobustKernel以外，其余的优化都不需要。

剔除优化后的outlier匹配点（MapPoints）。如果inlier匹配的点超过10个则为成功，bOK即为true。

FUNCTION2.2.2.1.2:TrackWithMotionModel

根据恒速模型设定当前帧的初始位姿，通过投影的方式在参考帧中找当前帧特征点的匹配点，优化每个特征点所对应3D点的投影误差即可得到位姿。mVelocity，这个两表示的是从参考帧到当前帧的位姿，我们要先乘上参考帧的相对于世界坐标系位姿，表示得到当前帧相对于世界坐标系位姿的初始位姿估计值。初始化当前帧对应的mvpMapPoints为NULL。

FUNCTION2.2.2.1.2.1:SearchByProjection

通过投影，对上一帧的特征点进行跟踪，将上一帧的MapPoints投影到当前帧(根据速度模型可以估计当前帧的Tcw)，在执行上：依次遍历参考帧的pMapPoints，计算出该3D点在当前帧的投影位置，设定一个以该点为中心的正方形区域内的所有特征点，获得该3D点的描述子和这些特征点的描述子之间的距离，找到距离最小的那个特征点，就是该3D点在当前帧匹配到的特征点。返回成功匹配的数量。

如果跟踪的点少，则扩大搜索半径再来一次。

FUNCTION2.2.2.1.2.2:PoseOptimization

剔除优化后的outlier匹配点（MapPoints）。如果inlier匹配的点超过10个则为成功，bOK即为true。

如果恒速模型不成功，那么就利用跟踪参考帧的模式。

如果初始化成功的话，但是mState的状态不是OK的话，那么就需要重定位。

FUNCTION2.2.2.1.3:Relocalization

计算当前帧特征点的Bow映射。

FUNCTION2.2.2.1.3.1:DetectRelocalizationCandidates

在重定位中找到与该帧相似的关键帧，1. 找出和当前帧具有公共单词的所有关键帧，2. 只和具有共同单词较多的关键帧进行相似度计算，3. 将与关键帧相连（权值最高）的前十个关键帧归为一组，计算累计得分，4. 只返回累计得分较高的组中分数最高的关键帧。在执行上：第一步：words是检测图像是否匹配的枢纽，遍历当前帧的每一个word，提取所有包含该word的KeyFrame，如果找到的关键帧还没有标记为当前帧的候选帧，那么就标记上，并且存入候选帧容器，该候选帧与当前帧共有的单词数累加。第二步：统计所有闭环候选帧中与当前帧具有共同单词最多的单词数，并以此决定阈值。第三步：遍历所有闭环候选帧，挑选出共有单词数大于0.8倍共同单词最多的单词数的候选帧且将单词匹配度存入和该候选帧存入IScoreAndMatch。找到与IScoreAndMatch中的候选帧连接的前10个关键帧，计算其组内最高得分和该组的累计得分，将改组里的累计得分和最佳得分的对应帧放入IAccScoreAndMatch。第五步，找到满足组累计得分大于0.75倍的组中的分数最高的关键帧（不一定是之前选出的候选帧），将此存入返回的vpRelocCandidates中。

然后对上式确定的候选帧，依次通过BoW进行匹配，当匹配点超过15时就开始初始化PnP solver，设置每个PnP solver的RANSAC迭代的参数，并记录符合条件的候选帧的帧数。只要候选帧数大于0，就不断得通过RANSAC，找到合适的位姿。

FUNCTION2.2.2.1.3.2:pSolver->iterate

以下内容可参考：[PnP算法简介与代码解析](#)

FUNCTION2.2.2.1.3.2.1:set_maximum_number_of_correspondences(mRansacMinSel)

mRansacMinSel为每次RANSAC需要的特征点数，默认为4组3D-2D对应点，为每次迭代的世界坐标系下的3D点pws，对应的图像坐标us，

匹配的2D点的个数不能小于RANSAC迭代过程中最少的inlier数(mRansacMinInliers=10)

开始RANSAC迭代：

FUNCTION2.2.2.1.3.2.2:add_correspondence

将随机选取的4组对应的3D-2D压入到pws和us

FUNCTION2.2.2.1.3.2.3:compute_pose(mRi, mti)

FUNCTION2.2.2.1.3.2.3.1:choose_control_points

第一个控制点选取为四个世界坐标系下的3D点的质心，然后将这四个点减去质心，做svd变换，另外三个控制点是PCA降维之后的三个主轴上的单位向量加上质心。

FUNCTION2.2.2.1.3.2.3.2:compute_barycentric_coordinates()

求解四个控制点的系数alphas， $(a2\ a3\ a4)' = \text{inverse}(cws2-cws1\ cws3-cws1\ cws4-cws1)'(pws-cws1)$ ， $a1 = 1-a2-a3-a4$ ，每一个3D参考点，都有一组alphas与之对应，cws1 cws2 cws3 cws4为四个控制点的坐标，pws为3D参考点的坐标。

FUNCTION2.2.2.1.3.2.3.3:fill_M

$$\text{根据投影模型: } \lambda_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = P \cdot \sum_{j=1}^4 \alpha_{ij} c_j^c = \begin{bmatrix} f_u & 0 & u_c \\ 0 & f_v & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^4 \alpha_{ij} \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix} \quad (3)$$

展开可得： $\sum_{j=1}^4 \alpha_{ij} f_u x_j^c + \alpha_{ij} (u_c - u_i) z_j^c = 0 \quad \sum_{j=1}^4 \alpha_{ij} f_v y_j^c + \alpha_{ij} (v_c - v_i) z_j^c = 0 \quad (4)$

将 (4) 写成矩阵形式： $Mx = 0$ 其中待求量为四个控制点： $x = [c_1^{ct}, c_2^{ct}, c_3^{ct}, c_4^{ct}]$ (5)

特征值分解M矩阵： $x = \sum_{i=1}^N \beta_i v_i$ (6)

$$\begin{pmatrix} \alpha_{i1}f_x & 0 & \alpha_{i1}(c_x - u_i) & \dots & \alpha_{i4}f_x & 0 & \alpha_{i4}(c_x - u_i) \\ 0 & \alpha_{i1}f_y & \alpha_{i1}(c_y - v_i) & \dots & 0 & \alpha_{i4}f_y & \alpha_{i4}(c_y - v_i) \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} X_1^c \\ Y_1^c \\ Z_1^c \\ \dots \\ X_4^c \\ Y_4^c \\ Z_4^c \end{pmatrix} = 0$$

然后对 $M^T M$ 做SVD分解，得到M的12个解向量，如下图绿色部分表示。下面的限定条件是因为在空间中两个点之间的距离不会因为他们所处的坐标系而改变，所以四个控制点就有6个距离。

Solve β_k :

$$X = \sum_{k=1}^N \beta_k V_k$$

Constraint: $\|c_i^c - c_j^c\|^2 = \|c_i^w - c_j^w\|^2$

\downarrow

N=1 : $\|\beta V^{[i]} - \beta V^{[j]}\|^2 = \|c_i^w - c_j^w\|^2$

N=2 : $\|(\beta_1 V_1^{[i]} + \beta_2 V_2^{[i]}) - (\beta_1 V_1^{[j]} + \beta_2 V_2^{[j]})\|^2 = \|c_i^w - c_j^w\|^2$

N=3 : $\|(\beta_1 V_1^{[i]} + \beta_2 V_2^{[i]} + \beta_3 V_3^{[i]}) - (\dots)\|^2 = \|c_i^w - c_j^w\|^2$

N=4 : $\|(\beta_1 V_1^{[i]} + \beta_2 V_2^{[i]} + \beta_3 V_3^{[i]} + \beta_4 V_4^{[i]}) - (\dots)\|^2 = \|c_i^w - c_j^w\|^2$

对于投影相机模型，N等于1，因为只有一个尺度变量；对于正交相机模型，N等于4，因为每个参考点的深度变化后仍满足约束；因此，当相机焦距比较小时，N为1。当相机焦距更大，相机接近于正交相机时，

将有4个接近于0的特征值。

FUNCTION2.2.2.1.3.2.3.4:compute_L_6x10

当N=4时，化解结果如下，N=1, 2, 3的结果是可以在下面的式子的一部分得到，所以上来直接构造L_{6x10}，后面的其他的L阵都可以从L_{6x10}抽取。

$$\begin{array}{l} \left[\begin{array}{c} \beta_{11} \\ \beta_{12} \\ \beta_{22} \\ \beta_{13} \\ \beta_{23} \\ \beta_{33} \\ \beta_{14} \\ \beta_{24} \\ \beta_{34} \\ \beta_{44} \end{array} \right] = \|c_i^w - c_j^w\|^2 \\ \text{---} \\ \left[\begin{array}{c} (\mathbf{v}_1^{[i]} - \mathbf{v}_2^{[j]})^2 & (\mathbf{v}_1^{[i]} - \mathbf{v}_2^{[j]})(\mathbf{v}_2^{[i]} - \mathbf{v}_2^{[j]}) & (\mathbf{v}_2^{[i]} - \mathbf{v}_2^{[j]})^2 & (\mathbf{v}_1^{[i]} - \mathbf{v}_3^{[j]})(\mathbf{v}_3^{[i]} - \mathbf{v}_3^{[j]}) & (\mathbf{v}_2^{[i]} - \mathbf{v}_3^{[j]})(\mathbf{v}_3^{[i]} - \mathbf{v}_3^{[j]}) \\ (\mathbf{v}_3^{[i]} - \mathbf{v}_3^{[j]})^2 & (\mathbf{v}_1^{[i]} - \mathbf{v}_3^{[j]})(\mathbf{v}_4^{[i]} - \mathbf{v}_4^{[j]}) & (\mathbf{v}_2^{[i]} - \mathbf{v}_3^{[j]})(\mathbf{v}_4^{[i]} - \mathbf{v}_4^{[j]}) & (\mathbf{v}_1^{[i]} - \mathbf{v}_4^{[j]})(\mathbf{v}_4^{[i]} - \mathbf{v}_4^{[j]}) & (\mathbf{v}_2^{[i]} - \mathbf{v}_4^{[j]})(\mathbf{v}_4^{[i]} - \mathbf{v}_4^{[j]}) \\ (\mathbf{v}_1^{[i]} - \mathbf{v}_4^{[j]})^2 & (\mathbf{v}_1^{[i]} - \mathbf{v}_4^{[j]})(\mathbf{v}_1^{[i]} - \mathbf{v}_4^{[j]}) & (\mathbf{v}_2^{[i]} - \mathbf{v}_4^{[j]})(\mathbf{v}_1^{[i]} - \mathbf{v}_4^{[j]}) & (\mathbf{v}_3^{[i]} - \mathbf{v}_4^{[j]})(\mathbf{v}_1^{[i]} - \mathbf{v}_4^{[j]}) & (\mathbf{v}_2^{[i]} - \mathbf{v}_4^{[j]})(\mathbf{v}_3^{[i]} - \mathbf{v}_4^{[j]}) \end{array} \right] \\ \{i, j\} \in [1, 4] \quad L_{6 \times 10} \quad * \quad \beta_{10 \times 1} \quad = \quad \rho_{6 \times 1} \end{array}$$

FUNCTION2.2.2.1.3.2.3.5:compute_rho

计算4个控制点的两两之间的距离。

FUNCTION2.2.2.1.3.2.3.6:find_betas_approx_1

利用L_{6x10}×betas=rho，用子集的近似约束来求解，具体方法可见代码，大概的方法就是用包含部分自项(betas₁₁\betas₂₂\betas₃₃\betas₄₄)和交叉项的betas(betas_{12..})和相应的L以及rho来构成等式并用SVD求出。

FUNCTION2.2.2.1.3.2.3.7:gauss_newton

下面公式中的betas0是上面N=1, 2, 3, 4算出的，在这为高斯牛顿优化提供初始值。

$$\begin{aligned} Error(\beta) &= \sum_{(i,j) \text{ s.t. } i < j} (\|c_i^c - c_j^c\|^2 - \|c_i^w - c_j^w\|^2) \\ Error(\beta) &= \sum_{(i,j) \text{ s.t. } i < j} \left(\sum_{k=1}^4 \left\| \beta_k \mathbf{v}_k^{[i]} - \beta_k \mathbf{v}_k^{[j]} \right\|^2 - \|c_i^w - c_j^w\|^2 \right) \end{aligned}$$

$$Error(\beta_0 + \Delta\beta) = Error(\beta_0) + Error'(\beta)\Delta\beta = 0$$

$$\boxed{Error'(\beta)} \Delta\beta = -Error(\beta_0) = \boxed{\rho - L\beta_0}$$

$$\boxed{\mathbf{A}} \quad \boxed{x} \quad = \quad \boxed{\mathbf{b}}$$

$$\begin{aligned} \boxed{Error'(\beta)} \Delta\beta &= -Error(\beta_0) = \boxed{\rho - L\beta_0} \\ \boxed{\mathbf{A}_{6 \times 4}} \quad \boxed{x_{4 \times 1}} &= \boxed{\mathbf{b}_{6 \times 1}} \\ Error(\beta) &= [L_1 \quad L_2 \quad L_3 \quad L_4 \quad L_5 \quad L_6 \quad L_7 \quad L_8 \quad L_9 \quad L_{10}] \left[\begin{array}{c} \beta_{11} \\ \beta_{12} \\ \beta_{22} \\ \beta_{13} \\ \beta_{23} \\ \beta_{33} \\ \beta_{14} \\ \beta_{24} \\ \beta_{34} \\ \beta_{44} \end{array} \right] - \rho \\ A &= \left[\frac{\partial Error(\beta)}{\partial \beta_1} \quad \frac{\partial Error(\beta)}{\partial \beta_2} \quad \frac{\partial Error(\beta)}{\partial \beta_3} \quad \frac{\partial Error(\beta)}{\partial \beta_4} \right] \\ &= [2L_1\beta_1 + L_2\beta_2 + L_4\beta_3 + L_7\beta_4 \quad L_2\beta_1 + 2L_3\beta_2 + L_5\beta_3 + L_8\beta_4 \quad \dots \quad \dots] \\ A &= QR, \quad x = R^{-1}Q^{-1}b, \quad \beta' = \beta_0 + \Delta\beta \end{aligned}$$

FUNCTION2.2.2.1.3.2.3.7.1:compute_A_and_b_gauss_newton

计算出误差矩阵对betas1, betas2, betas3, betas4的偏导。上式中
betas11=betas1×betas1。

FUNCTION2.2.2.1.3.2.3.7.2:qr_solve

通过QR分解求出x。

根据求出的x值，这个变量加在初始值上，迭代iterations_number次。求得betas。这样我们就可以得到相机坐标系下的控制点的坐标。对于每个3D点，在世界坐标系下，我们可以找到四个algha，使得四个控制点可以表达这个3D点，且这四个algha的和为1。对于同样的algha，在相机左边系下也可以满足四个控制点线性表

示这个3D参考点。这四个控制点可以通过3D点的投影以及在不同坐标系下的两个点之间距离的不变性求出，结果由上面的betas和M的几个特征向量表示，再加上alpha即可以表示3D点在相机坐标系下的参考点。

FUNCTION2.2.2.1.3.2.3.8:compute_R_and_t

Known p_i^c and p_i^w , solve R and t:

- 1). compute center: $p_c^c = \frac{\sum p_c^i}{N}, p_w^c = \frac{\sum p_w^i}{N}$
- 2). remove center: $q_c^i = p_c^i - p_c^c, q_w^i = p_w^i - p_w^c$
- 3). compute $H = \sum_{i=1}^N q_c^i q_w^{i T}$
- 4). SVD: $H = U \Sigma V^T, R = V U^T, t = p_c^c - R p_w^c$
(if $\det(R) < 0, R(2,.) = -R(2,.)$)

到这，我们就求出了不同的N对应的3D点投影的坐标和3D点匹配的2D点的坐标的误差，选出误差最小的那组[R|t]。

FUNCTION2.2.2.1.3.2.4:CheckInliers

在上一步中我们求出了[R|t]，据此我们可以算出检查哪些3D-2D点对属于inliers，这是根据重投影误差和阈值比较来确定是否是内点，并计算出内点的数量mnInliers。

找到内点数量值最大的组对应的位姿作为最佳位姿的候选，同时利用这一组的所有点数再进行EPnP计算位姿，通过位姿再次计算内点数量，如果内点数量还是超过了Ransac要求的最少的内点，那么就返回此位姿，否则到达了最大的Ransac的迭代次数，就置bNoMore为true，返回之前的最佳位姿的候选。

如果迭代达到最大值，我们就把相应的候选关键帧删除。

FUNCTION2.2.2.1.3.3:PoseOptimization

通过重投影优化位姿，得到内点的数量

如果，内点的数量少于10个，就跳过本次循环，如果内点($10 < nGood < 50$)，则通过投影的方式对之前当前帧和候选帧未匹配的点进行匹配(搜索边长20，ORB距离100)找到一些新的匹配点，若此时好的匹配的数量和新找到的匹配点的数量大于50，则进行优化求解，如果找到好的内点的匹配数量还没有超过50，那么就再次通过投影的方式对之前当前帧和候选帧未匹配的点进行匹配(搜索边长6，ORB距离64)找一些新的匹配关系，若此时好的匹配的数量和新找到的匹配点的数量大于50，则进行优化求解，得到当前帧的优化后的位姿。当前帧就是最新的重定位帧。

• 2.2.2.2.mbOnlyTracking=true

只进行跟踪tracking，局部地图不工作，如果系统跟丢了，那么就进行重定位，否则判断变量mbVO，如果mbVO为false表示此帧匹配了很多的MapPoints，跟踪很正常，再根据变量mVelocity是否为空，判断是使用跟踪参考帧模型还是采用运动跟踪模型。如果mbVO为true表明此帧匹配了很少的MapPoints，少于10个，要跪的节奏，这时候既做跟踪又做定位，定位和跟踪的结果分别用bOKMM和bOKReloc表示，只要mVelocity不为空就做基于恒速模型跟踪，结果如果是跟踪成功定位失败，那么结果借用跟踪的结果，但只要是重定位成功，那么整个跟踪过程就正常进行(定位与跟踪，更相信重定位)，最后只要是跟踪和重定位只要一个成功，那么结果就正常。将最新的关键帧作为reference frame。

总结：mbOnlyTracking=false，系统主要做跟踪(恒速模型和参考帧模型)，当系统跟踪失败做重定位。mbOnly Tracking=true，系统一边跟踪(恒速模型)一边定位，但是更相信定位的结果，同样跟踪失败后做重定位。

2.2.3.PART2:在帧间匹配得到初始的姿态后，现在对local map进行跟踪得到更多的匹配，并优化当前位姿（这是回来再看这段程序的一点点补充：这里的局部地图是在跟踪的过程中进行的，他是在现在处理的当前帧的地图点确立地图关键帧和这些地图关键帧确立的地图点构成的局部地图的基础上，完成位姿优化的目的。这些地图会在下一帧进行局部地图更新时被清空，以下一帧为参考再构建一个新的局部地图）

• 2.2.3.1mbOnlyTracking=false

FUNCTION2.2.3.1.1:TrackLocalMap

local map:当前帧、当前帧的MapPoints、当前关键帧与其它关键帧共视关系，1. 更新局部地图，包括局部关键帧和关键点、2. 对局部MapPoints进行投影匹配、3. 根据匹配对估计当前帧的姿态、4. 根据姿态剔除误匹配。

FUNCTION2.2.3.1.1.1:UpdateLocalMap

更新mpMap中的地图点，为了可视化。这行程序放在UpdateLocalPoints函数后面是不是好一些？

FUNCTION2.2.3.1.1.1:UpdateLocalKeyFrames

更新局部关键帧，遍历当前帧的MapPoints，将观测到这些MapPoints的关键帧和相邻的关键帧取出，更新mvpLocalKeyFrames。每个当前帧的优化的地图都是新建的，所以要先清空局部关键帧，依据一下3个策略，所以先建立一个3倍keyframeCounter的大小。步骤1：遍历当前帧的MapPoints，记录所有能观测到当前帧MapPoints的关键帧。步骤2：更新局部关键帧（mvpLocalKeyFrames），添加局部关键帧有三个策略(策略1：能观测到当前帧MapPoints的关键帧作为局部关键帧，策略2：与策略1得到的局部关键帧共视程度很高的关键帧作为局部关键帧(当局部关键帧超过80时就不在添加，策略2.1:最佳共视的10帧，策略2.2:自己的子关键帧，策略2.3:自己的父关键帧)，步骤3：更新当前帧的参考关键帧，与自己共视程度最高的关键帧作为参考关键帧，在策略1的时候计算了与自己共视程度最高的关键帧pKFmax)。在程序中的pKF->mnTrackReferenceForFrame = mCurrentFrame.mnId是为了防止重复添加局部关键帧，表明了我要添加的这一帧已经是mCurrentFrame的局部关键帧了。

FUNCTION2.2.3.1.1.2:UpdateLocalPoints

更新局部关键点，步骤1：清空局部MapPoints，步骤2：遍历局部关键帧mvpLocalKeyFrames，步骤3：将局部关键帧的MapPoints添加到mvpLocalMapPoints。

在局部地图中查找与当前帧匹配的MapPoints

FUNCTION2.2.3.1.1.2:SearchLocalPoints

在局部地图中查找在当前帧视野范围内的点，将视野范围内的点和当前帧的特征点进行投影匹配。步骤1：遍历当前帧的mvpMapPoints，标记这些MapPoints不参与之后的搜索，因为当前的mvpMapPoints一定在当前帧的视野中，取出该MapPoint后，更新能观测到该点的帧数加1，标记该点被当前帧观测到，标记该点将来不被投影，因为已经匹配过。步骤2：将所有局部MapPoints投影到当前帧，判断是否在视野范围内，然后进行投影匹配，已经被当前帧观测到MapPoint不再判断是否能被当前帧观测到，步骤2.1：判断LocalMapPoints中的点是否在视野内。

FUNCTION2.2.3.1.1.2.1:isInFrustum

判断一个点是否在视野内。策略1：将MapPoint投影到当前帧，并判断是否在图像内。策略2：计算MapPoint到相机中心的距离，并判断是否在尺度变化的距离内。策略3：计算当前视角和平均视角夹角(CreateInitialMapMonocular的UpdateNormalAndDepth的函数计算得到)的余弦值，若小于cos(60)，即夹角大于60度则返回。最后根据深度预测尺度（对应特征点在一层），并标记该点将来要被投影(在函数SearchByProjection中被使用)。如果以上条件满足就代表当前的地图点在视野里。

如果当前的地图点在视野里，那么观测到该点的帧数加1，该MapPoint在某些帧的视野范围内，nToMatch计数器+1。步骤2.2：对视野范围内的MapPoints通过投影进行特征点匹配。

FUNCTION2.2.3.1.1.2.2:SearchByProjection(mCurrentFrame,mvpLocalMapPoints,th)

通过投影，对Local MapPoint进行跟踪。将Local MapPoint投影到当前帧中，由此增加当前帧的MapPoints，在SearchLocalPoints()的isInFrustum()中已经将Local MapPoints重投影到当前帧，isInFrustum()还标记了这些点是否在当前帧的视野中，即mbTrackInView，对这些MapPoints，在其投影点附近根据描述子距离选取匹配，以及最终的方向投票机制进行剔除。

更新局部所有MapPoints后对位姿再次优化。

FUNCTION2.2.3.1.1.3:PoseOptimization(&mCurrentFrame)

更新当前帧的MapPoints被观测程度，并统计跟踪局部地图的效果，如果当前帧的MapPoints有被观测到，mnMatchesInliers就+1，过来统计该MapPoint被其它关键帧观测到过，最后得到当前帧有多少MapPoints被能其他帧观测到。如果这个数在刚刚重定位后的情况下还小于30的话认为跟踪失败，或者这个数在跟踪的状态下小于30，则表明跟踪失败。

- 2.2.3.2.mbOnlyTracking=true

在状态跟踪正常(重定位模式下)可进行地图跟踪。

2.2.4.PART3:收尾工作

依据bOK的状态更新mState的状态，把当前帧传递给mpMapDrawer线程。如果bOK的状态是true，表示上面的跟踪是正常的，那么就根据当前帧和上一帧的状态更新运动模型的mVelocity。mVelocity表示的是上一帧到当前帧的位姿。如果bOK为false，则mVelocity为空。把当前帧的位姿传递给mpMapDrawer线程。

FUNCTION2.2.4.1NeedNewKeyFrame

判断当前帧是否为关键帧。步骤1：如果用户在界面上选择重定位，因为不需要地图更新，那么将不插入关键帧。如果局部地图（mpLocalMapper）线程被闭环检测线程使用，则不插入关键帧。如果距离上一次重定位不超过1s不插入或者Map中的关键帧超过了mMaxFrames不插入。步骤3：得到参考关键帧跟踪到的MapPoints数量，在执行上判断如果参考帧的MapPoints点被观测到的次数大于minObs，则认为该点被跟踪到，并递增计数器。步骤4：查询局部地图管理器是否繁忙。决策是否需要插入关键帧，1.很长时间没有插入关键帧(1s)，2. localMapper处于空闲状态，3.与之前参考帧（最近的一个关键帧）重复度不是太高，匹配的内点的数量要小于跟踪到的MapPoints的一个比例值，但也不能太小。当(1||2&&3)时，如果localMapper处于空闲状态就插入，否则中断BA，判断队列里的关键帧是否小于3，如果是就插入关键帧。（tracking插入关键帧不是直接插入，而且先插入到mInNewKeyFrames中，然后localmapper再逐个pop出来插入到mspKeyFrames）。

FUNCTION2.2.4.2CreateNewKeyFrame

步骤1：将当前帧构造成关键帧，将当前关键帧设置为当前帧的参考关键帧，将关键帧插入mInNewKeyFrames中。把当前帧的信息更新到最新关键帧中。

删除那些在bundle adjustment中检测为outlier的3D map点。如果跟踪失败，且地图线程里的关键帧的数量小于5，那么就重置系统。更新当前帧的参考帧，保存上一帧的数据。最后记录位姿信息(记录的是当前帧与其参考帧的相对位姿)，用于轨迹复现。

这里的MAP比较多，我们来稍微区分下，Map* mpMap是用于显示的地图对象；TrackLocalMap我们只是保留离相机当前位置较近的特征点，把远的或者视野外的特征点丢掉，这次特征点是用来和当前帧匹配球相机位置的，所以我们希望他能够做得快一点；mpLocalMapper是全局地图，记录了SLAM运行以来的所有特征点，显然它的规模要大一些，主要用来表达整个环境，但是直接在全局地图上定位对计算机的负担太大了，主要用于回环检测和地图表达。

至此，tracking线程就介绍完了。

好文要顶 关注我 收藏该文

h_立青人韦

关注 - 9
粉丝 - 4

[+加关注](#)

1 0

« 上一篇：[针孔的相机成像模型](#)
 » 下一篇：[orb_slam代码解析\(3\)LocalMapping线程](#)

posted @ 2018-01-09 14:38 h_立青人韦 阅读(2978) 评论(5) 编辑 收藏

评论列表

#1楼 2018-11-12 20:07 Max_诸葛小亮

您好，请问在FUNCTION2.2.1.2.3.1.1:Triangulate中，为什么在三角化求解阶段把DLT分解右侧的向量第三行给省略了？这是什么原因么？

支持(0) 反对(1)

#2楼[楼主] 2018-11-12 20:45 h_立青人韦

@ Max_诸葛小亮

因为第三行可以由前两行线性表示，故而省略

支持(1) 反对(0)

#3楼 2018-11-12 21:23 Max_诸葛小亮

博主您好，非常感谢您刚才的回答。但我还有一个问题想请教一下。我在ORB_SLAM2的代码中发现，利用SVD分解求出的四维坐标并不是空间点的三维坐标而是需要用前三维除以第四维，归一化得到的结果作为空间点的坐标值(x,y,z)。请问两个问题，1.求得的结果第四维有什么含义？2.为什么要除以第四维才是空间点的坐标值？

支持(0) 反对(0)

#4楼[楼主] 2018-11-12 21:35 h_立青人韦

@ Max_诸葛小亮

我不太清楚你说的是代码中的哪块内容。但如果是4维向量代表空间点，则是齐次坐标的形式，在齐次坐标中坐标的每个分量同乘一个非零常数，仍表示同一个点，所以除以向量的第四个元素只是强制性最后一项为1，从而得到一个点唯一的坐标表

示。

支持(0) 反对(0)

#5楼 2018-11-12 21:51 Max_诸葛小亮

@ h_立青人韦
代码是orb_slam2中文件LocalMapping.cc,
函数void LocalMapping::CreateNewMapPoints()中的
这句话x3D = x3D.rowRange(0,3)/x3D.at<float>(3);最终也是将x3D放入地图中，有时间您可以看看。按照您的意思结果的第四项没有具体含义，也不含有尺度，对吧？

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】超50万VC++源码: 大型组态工控、电力仿真CAD与GIS源码库!

【推荐】基于 HTML5 的 WebGL 楼宇自控 3D 可视化监控

【推荐】专业便捷的企业级代码托管服务 - Gitee 码云

相关博文：

- SLAM刚刚开始的未来
- ORB-SLAM2初步（Tracking.cpp）
- ORB-SLAM 代码笔记（四）tracking代码结构
- ORB-SLAM（五）优化
- ORB-SLAM2学习5 Tracking.h

最新新闻：

- 对话顾剑民博士：自动驾驶热度“滑向低谷”，寒冬未真正到来
 - 彭蕾：CEO如何面对“至暗时刻”和无可诉说的孤独感
 - BuzzFeed衰落启示：无限竞争+劣质产品=商业模式失败
 - 阿里没有“失速”
 - FBI抓捕第二位苹果中国工程师 指控窃取无人机密面面临10年监禁
- » [更多新闻...](#)

Copyright ©2019 h_立青人韦