代码相关：静态成员变量在类的所有实例之间是共享的。

每个.h文件中都有定义一个同名的类。每个类中首先有一个构造函数，再有一些类的成员函数，都在.cc文件中，有些.cc文件中还有一些其他函数或类的定义。

类的构造函数可以在类的声明处定义，也可以在类外进行定义（在类外部定义构造函数时，需要使用类名加上作用域解析运算符 `::` 来指定函数所属的类）。构造函数的参数量不同，根据提供的参数数量和类型，编译器会选择调用匹配的构造函数。

梳理的结构为：

.h文件

构造函数

类的成员函数

变量名前的 `m` 是用来表示该变量为某类的成员变量

变量名前的 `n` 表示 int 类型的变量

变量名前的 `v` 表示向量类型（vector）的变量

变量名前的 `b` 表示布尔类型（boolean）的变量

变量名前的 `p` 表示指针（pointer）类型的变量

变量名前的 `s` 表示std::set类型的变量

变量名前的 `l` 表示std::list类型的变量

`KF` 表示KeyFrame类型

ComputeBoW词袋模型

对一个Frame或KF中每一个描述子都得到了一个BowVector（**包含节点id，weight**）和一个FeatureVector（**节点所属的父节点id**[这里的父节点不是叶子的上一层，它距离叶子深度为levelsup]，**属于该父节点的描述子的索引vector**[这个Frame中的第几个特征点]）。这两个变量都是`std::map`升序容器

如果不同描述子对应到了同一个节点id，则权重累加。

mono_tum.cc

ORB_SLAM2::System SLAM（创建并初始化SLAM system）-->System.h，读取图像的时间戳与图像；SLAM.TrackMonocular(im，tframe)（im为图像，tframe为时间戳）-->System::TrackMonocular。

System.h

System初始化整个系统。

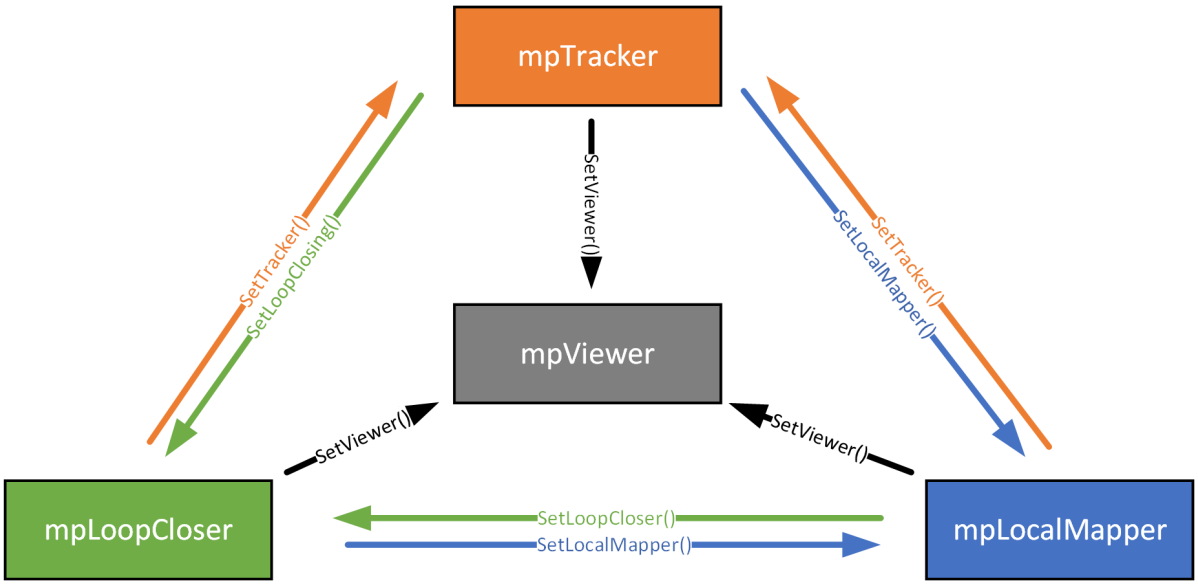
mpTracker = new Tracking(...)，追踪线程 -->Tracking.h；

mpLocalMapper = new LocalMapping(...)， 开启子线程，局部建图线程-->LocalMapping.h；

mpLoopCloser = new LoopClosing(...)， 开启子线程，回环检测线程-->LoopClosing.h

mpViewer = new Viewer(...)， 开启子线程，可视化线程-->Viewer.h；

然后设置了进程间的指针，通过**SetTracker()**、**SetLocalMapper()**、**SetLoopClosing()**三个函数将建立好的三个对象彼此之间互相关联(例如，在Tracking中可以直接通过LocalMapper类的成员变量mpLocalMapper修改LocalMapping线程中的一些变量或者调用其函数)。如果有可视化，则通过**SetViewer()**函数实现关联。整理关系如下图所示。



TrackMonocular单目下的track

首先是模式的改变（可选：关闭局部建图的线程，reset。可视化界面上可勾选或取消）；再获取位姿Tcw -->Tracking.GrabImageMonocular

Tracking.h

Tracking追踪线程

从配置文件加载相机内参、矫正系数，并打屏输出；加载ORB特征点有关的参数，并新建特征点提取器，ORBextractor特征点提取-->ORBextractor.h；STEREO，RGBD传感器初始化远近点阈值，真实深度。

GrabImageMonocular 单目下得到位姿

Step 1： 将彩色图像转为灰度图像；Step 2： 构造Frame(会判断该帧是不是初始化)-->Frame；Step 3： 跟踪-->Tracking::Track

Track

判断当前帧状态，地图更新时加锁；单目初始化、地图初始化-->Tracking::MonocularInitialization；局部建图线程则可能会对原有的地图点进行替换-->Tracking::CheckReplacedInLastFrame；

正常跟踪:
参考关键帧跟踪：如果运动模型为空（说明是刚初始化开始，或者已经跟丢了）或当前帧紧紧地跟着在重定位的帧的后面，使用最近的关键帧来跟踪当前的普通帧-->Tracking::TrackReferenceKeyFrame；
如果有运动模型的话，使用**恒速模型跟踪**：假设短时间内（相邻帧）物体处于匀速运动状态，可以用上一帧的位姿和速度来估计当前帧的位姿-->Tracking::TrackWithMotionModel。

跟踪状态mState：LOST:
那么就只能重定位了，BOW搜索，EPnP求解位姿-->Tracking::Relocalization。

若跟踪成功并得到当前帧初始姿态后，对local map进行跟踪得到更多的匹配，并优化当前位姿-->Tracking::TrackLocalMap。

更新显示线程中的图像、特征点、地图点等信息-->FrameDrawer::Update TODO；

若跟踪成功:
则更新恒速运动模型mVelocity（上一帧到当前帧的变换矩阵T）；否则置为空
更新显示中的位姿-->[MapDrawer::SetCurrentCameraPose] TODO；
清除观测不到的地图点，遍历当前帧的地图点，如果观测到该点的相机数目为0则删除此地图点；清除恒速模型跟踪中 Tracking::UpdateLastFrame中为当前帧临时添加的MapPoints（仅双目和rgbd）；
检测插入关键帧条件-->Tracking::NeedNewKeyFrame；插入关键帧（对于双目或RGB-D会产生新的地图点）-->Tracking::CreateNewKeyFrame；
删除那些在BA中检测为outlier的地图点；

若跟踪失败：只能重新Reset

最后记录位姿信息，用于最后保存所有的轨迹，若跟踪失败就用上一次的值。

MonocularInitialization

如果单目初始器还没有被创建，若特征点大于100则创建当前帧的初始器-->Initializer::Initializer。

如果单目初始化器已经被创建，连续两帧的特征点个数都大于100，才继续进行初始化过程；在mInitialFrame与mCurrentFrame中找匹配的特征点对，先创建一个ORBmatcher类--> ORBmatcher ，特征点匹配-->ORBmatcher::SearchForInitialization，返回匹配特征点的点数；若匹配的特征点太少则初始化失败，重新初始化；若匹配点数量满足，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、并初始化地图点MapPoints-->Initializer::Initialize；删除无法三角化的匹配点；将初始化的第一帧作为世界坐标系，所以第一帧变换矩阵为单位矩阵，记录参考帧和当前帧的位姿-->Frame::SetPose；最后创建初始化地图点MapPoints-->Tracking::CreateInitialMapMonocular。

CreateInitialMapMonocular

创建KeyFrames ，单目初始化时参考帧和当前帧都是关键帧-->KeyFrame；将初始关键帧，当前关键帧的描述子转为BoW-->KeyFrame::ComputeBoW；将关键帧插入到地图-->Map::AddKeyFrame；用初始化得到的3D点来生成地图点MapPoints-->MapPoint，为该MapPoint添加属性-->KeyFrame::AddMapPoint， -->MapPoint::AddObservation，从众多观测到该MapPoint的特征点中挑选最有代表性的描述子-->MapPoint::ComputeDistinctiveDescriptors，更新该MapPoint平均观测方向以及观测距离的范围-->MapPoint::UpdateNormalAndDepth；向地图中插入地图点Map::AddMapPoint；更新关键帧之间的连接权重（该关键帧与其他关键帧观测到同一个3D地图点的个数）-->KeyFrame::UpdateConnections；打屏信息输出地图创建时地图点的数量；全局BA优化，同时优化所有位姿和三维点Optimizer::GlobalBundleAdjustemnt；取场景的中值深度，用于尺度归一化-->KeyFrame::ComputeSceneMedianDepth；平均深度要大于0,当前帧中被观测到的地图点的数目应该

大于100-->KeyFrame::TrackedMapPoints，如果失败则打屏输出初始化失败；将两帧之间的变换归一化到平均深度1的尺度下，变换矩阵中的平移向量除场景中值深度；每个地图点也除这个归一化深度；将关键帧插入局部地图-->LocalMapping::InsertKeyFrame；当前帧位姿修改、mpLastKeyFrame修改为当前帧pKFcur、mnLastKeyFrameId的变换、局部地图点就是现在所有的地图点、mLastFrame修改为当前帧mCurrentFrame（其共视度最高的关键帧为pKFcur）、设置参考地图点（局部地图点）-->Map::SetReferenceMapPoints、设置当前相机的位姿-->MapDrawer::SetCurrentCameraPose。

TrackReferenceKeyFrame

将当前帧的描述子转化为BoW向量-->Frame::ComputeBoW；通过词袋BoW加速当前帧与参考帧之间的特征点匹配-->ORBmatcher::SearchByBoW，得到了mvpMapPoints（特征点索引，3D地图点）；将上一帧的位姿态作为当前帧位姿的初始值，再通过优化3D-2D的重投影误差来获得优化后的当前帧的位姿-->Optimizer::PoseOptimization，会分离出一些外点；剔除优化后的匹配点中的外点，匹配数超10才算成功。

CheckReplacedInLastFrame

TODO

TrackWithMotionModel

更新上一帧的位姿(对于双目或RGB-D相机，还会根据深度值生成临时地图点，提高跟踪鲁棒性)-->Tracking::UpdateLastFrame；根据之前估计的速度，用恒速模型得到当前帧的初始位姿（实际上就是变换矩阵相乘）；用上一帧地图点进行投影匹配，如果匹配点不够，则扩大搜索半径再来一次-->ORBmatcher::SearchByProjection；利用3D-2D投影关系，优化当前帧位姿-->Optimizer::PoseOptimization，会分离出一些外点；剔除优化后的匹配点中的外点，匹配数超10才算成功。

UpdateLastFrame

单目情况：计算了一下上一帧的世界坐标系位姿

双目和rgbd情况：选取有有深度值的并且没有被选为地图点的点生成新的临时地图点，提高跟踪鲁棒性

Relocalization

计算当前帧特征点的词袋向量-->Frame::ComputeBoW；用词袋找到与当前帧相似的候选关键帧-->KeyFrameDatabase::DetectRelocalizationCandidates；遍历所有的候选关键帧，通过词袋进行快速匹配-->ORBmatcher::SearchByBoW，如果匹配的特征点数目够用则用匹配结果初始化EPnPsolver -->PnPsolver.h ，-->[PnPsolver::SetRansacParameters]（TODO）；通过一系列操作,直到找到能够匹配上的关键帧（多重的检验，避免错误的闭环）：通过EPnP算法估计姿态，并得到内点-->[PnPsolver::iterate]；若EPnP 计算出了位姿，BA优化-->Optimizer::PoseOptimization，返回内点的数量；根据内点数量（小于10则舍弃此候选关键帧，大于50则候选关键帧重定位成功，处于中间则进一步操作），如果内点处于中间，则通过投影的方式对之前未匹配的点进行匹配-->ORBmatcher::SearchByProjection，再进行BA优化求解-->Optimizer::PoseOptimization；如果此BA后内点数还是比较少(<50)但是还不至于太少(>30)，用更小窗口、更严格的描述子阈值，重新进行投影搜索匹配-->-->ORBmatcher::SearchByProjection；如果匹配数目>50，最后再BA优化一下-->Optimizer::PoseOptimization，再删除外点。

TrackLocalMap

实际上是一个后处理，前面参考关键帧跟踪or恒速模型跟踪or重定位，只是跟踪一帧得到初始位姿，这里搜索局部关键帧、局部地图点，和当前帧进行投影匹配，得到更多匹配的MapPoints后进行Pose优化。更新局部关键帧和局部地图点-->Tracking::UpdateLocalMap；筛选局部地图中新增的在视野范围内的地图点，投影到当前帧搜索匹配，得到更多的匹配关系Tracking::SearchLocalPoints；新增了更多的匹配关系，BA优化得到更准确的位姿-->Optimizer::PoseOptimization；更新当前帧的地图点被观测程度（被多少帧观测到），并统计跟踪局部地图后匹配数目；根据跟踪匹配数目及重定位情况决定是否跟踪成功（如果最近刚刚发生了重定位,那么至少成功匹配50个点才认为是成功跟踪，否则30个点就行）

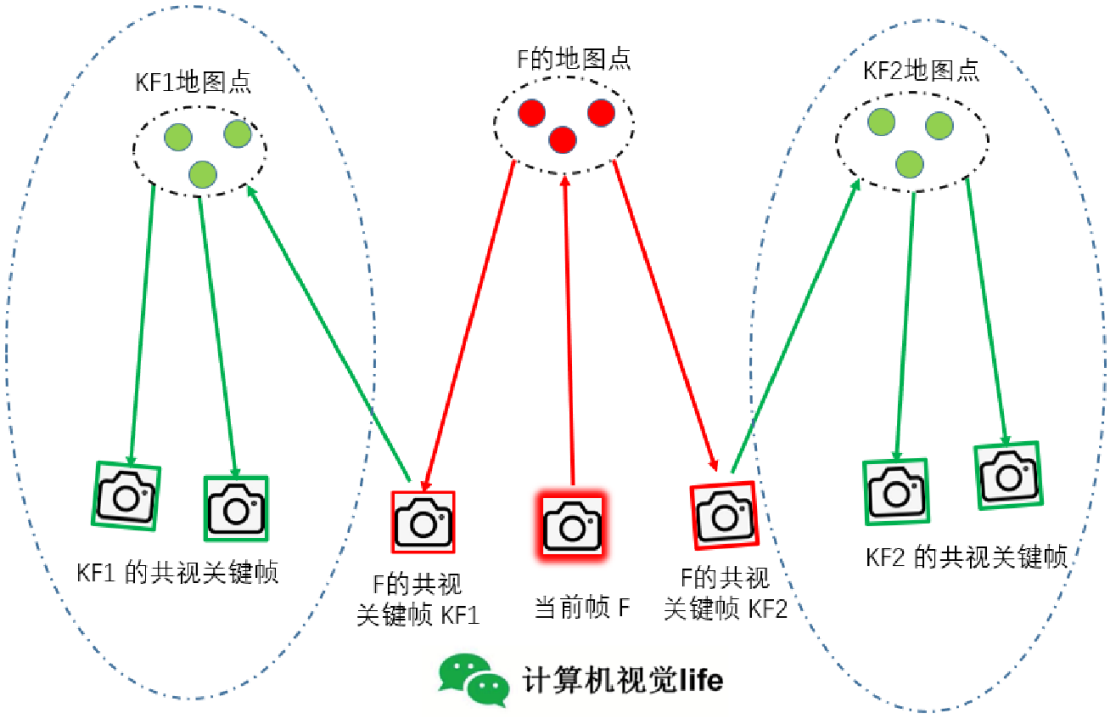
UpdateLocalMap

更新LocalMap，局部地图包括： K1个关键帧、K2个临近关键帧和参考关键帧，由这些关键帧观测到的MapPoints。用共视图来更新局部关键帧-->Tracking::UpdateLocalKeyFrames和局部地图点-->Tracking::UpdateLocalPoints

UpdateLocalKeyFrames

遍历当前帧的地图点，使用到了map容器，巧妙的记录所有能观测到当前帧地图点的关键帧及共视程度（同一个关键帧看到的地图点数量）；更新局部关键帧（mvpLocalKeyFrames），添加局部关键帧有3种类型：
1、能观测到当前帧地图点的关键帧，也称一级共视关键帧（将邻居拉拢入伙）
2、一级共视关键帧的共视（前10个）关键帧，称为二级共视关键帧（将邻居的邻居拉拢入伙）
3、一级共视关键帧的子关键帧（将邻居的孩子们拉拢入伙）；一级共视关键帧的父关键帧（将邻居的父拉拢入伙）

最后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
mCurrentFrame.mpReferenceKF



UpdateLocalPoints

遍历局部关键帧 mvpLocalKeyFrames，将局部关键帧的地图点添加到mvpLocalMapPoints

SearchLocalPoints

遍历当前帧的地图点，标记这些地图点不参与之后的投影搜索匹配；判断所有局部地图点中除当前帧地图点外的点，是否在当前帧视野范围内-->Frame::isInFrustum；投影匹配得到更多的匹配关系-->ORBmatcher::SearchByProjection。

NeedNewKeyFrame

纯VO模式下没有局部建图和回环的过程，不需要关键帧；如果局部地图线程被闭环检测使用，则不插入关键帧；如果距离上一次重定位比较近，且关键帧数目超出最大限制（设置为fps），不插入关键帧；得到参考关键帧mpReferenceKF跟踪到了的地图点数量-->KeyFrame::TrackedMapPoints；（对于双目或RGBD摄像头，统计成功跟踪的近点的数量，如果跟踪到的近点太少(<100)，且没有跟踪到的近点较多(>70)，bNeedToInsertClose为True，可以插入关键帧）；
决策是否需要插入关键帧：设定比例阈值，当前帧和参考关键帧跟踪到了的地图点的比例，比例越大，越倾向于增加关键帧（单目0.9、非单目0.75、若关键帧只有一帧则0.4）
条件c1a：很长时间没有插入关键帧，可以插入；
条件c1b：满足插入关键帧的最小间隔，且局部地图线程localMapper处于空闲状态，可以插入；
条件c1c：（在双目，RGB-D的情况下当前帧跟踪到的点比参考关键帧的0.25倍还少，或者满足bNeedToInsertClose，可以插入）
条件c2：【和参考帧相比当前跟踪到的点太少(根据比例阈值)或者满足bNeedToInsertClose】，且跟踪到的内点还不能太少(>15)；
再判断：(c1a||c1b||c1c)&&c2；成功则localMapper空闲时可以直接插入，不空闲的时候要根据情况插入（tracking插入关键帧不是直接插入，而且先插入到mlNewKeyFrames中，然后localmapper再逐个pop出来插入到mspKeyFrames中。非单目，队列中的关键帧数目不是很多,可以插入；单目直接无法插入，因为单目关键帧相对比较密集）。

CreateNewKeyFrame

局部建图线程运行时才可插入关键帧；将当前帧构造成关键帧-->KeyFrame；将当前关键帧设置为当前帧的参考关键帧；（对于双目或rgbd摄像头，为当前帧生成新的地图点。得到当前帧有深度值的特征点并升序排序（不一定是地图点）；从中找出不是地图点包装为地图点、没有被观测到的地图点清除再重新包装为地图点；包装地图点即将地图点的3D位置及其各种属性添加到地图；点深度超过阈值且不需要创建为地图点的点数超过100就不再包装地图点）；将关键帧插入到列表 mlNewKeyFrames中-->LocalMapping::InsertKeyFrame；更新mnLastKeyFrameId、设置mpLastKeyFrame为当前关键帧。

ORBextractor.h

ORBextractor

图像金字塔（每层提取的特征点的点数按面积来划分，面积越小特征点越少）；fast角点提取（像素点周围像素亮度阈值，使用cv::FAST实现）；brief描述子（使用固定的bit_pattern_31_[256*4]（256个点对直接写到代码中硬编码，这样占用内存，不如写成配置文件读进来），每四个整数值表示一个点对，形式为 (x1， y1， x2， y2)，比较两点的像素值，第二个点像素值大则为1，否则为0，再将nd（nd=128，256和512）个结果排列）；灰度质心法（在一个圆内计算灰度中心，得到像素点角度。想要有旋转不变性，用圆而不用正方形，需要计算每行像素的u坐标边界umax）

operator() 用来得到图像的特征点和描述子

首先得到图像金字塔-->ORBextractor::ComputePyramid；计算图像的特征点，并且将特征点进行均匀化，计算特征点方向信息，八叉树的方式-->ORBextractor::ComputeKeyPointsOctTree；对图像进行高斯模糊，高斯模糊可以去掉杂点，计算高斯模糊后图像的描述子-->ORBextractor.cc中定义的computeDescriptors，描述子大小(nkeypoints，32，CV_8U)；对非第0层图像中的特征点的坐标恢复到第0层图像（原图像）的坐标系下。

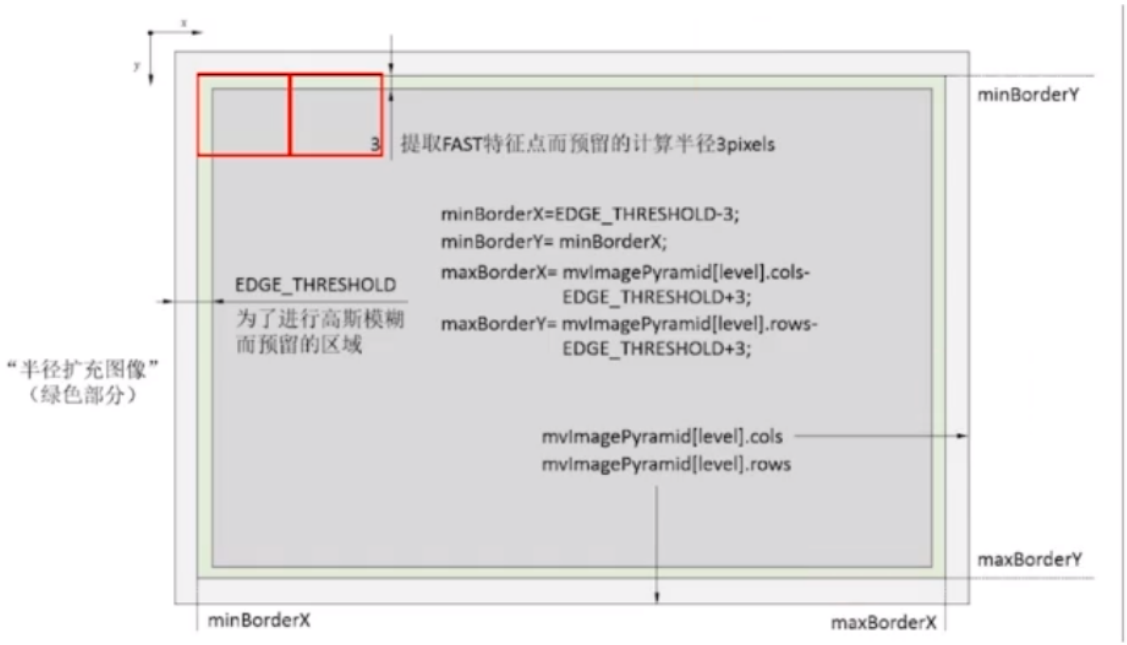
ComputePyramid 构建图像金字塔

为了进行高斯模糊、提取FAST特征点，需要对图像边界进行扩充19像素。得到扩充后金字塔每一层图像

ComputeKeyPointsOctTree 八叉树的方式计算特征点

返回用vector < vector > allKeypoints来存储的每层图层的特征点及其描述子；

图像cell的尺寸为30*30像素，对图像有效边界坐标初始化（之前对图像边界进行了扩充，特征点的提取多3像素），对图像的每一层按框进行遍历FAST提取特征点；



使用八叉树法对一个图像金字塔图层中的特征点进行平均和分发-->ORBextractor::DistributeOctTree，返回当前图层优化过的特征点的坐标；恢复当前图层这些特征点的坐标，并记录每个特征点的计算方向的patch，缩放后对应的大小， 又被称作为特征点半径（处于同一图层的特征点计算方向的patch一样）；计算这些特征点的方向信息（返回每个特征点的角度值），是分图层计算的-->ORBextractor.cc中定义的computeOrientation。

这里第一层vector存储的是某图层里面的所有特征点，第二层存储的是整个图像金字塔中的所有图层里面的所有特征点

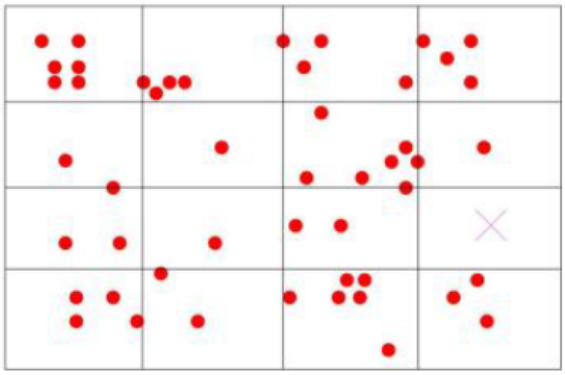
DistributeOctTree

使用八叉树对特征点进行平均和分发，使用类ExtractorNode（在ORBextractor.h中定义的类）来定义每个结点node的各种信息。实现八叉树的过程挺多的，涉及到结点的初始化创建，更新增加子结点，删除母结点，还有马上满足结点数目时，排序结点中的特征点数，取特征点多的结点先分裂。每次运行程序提取的特征点不一样的原因是这个函数里面的排序sort ，建议使用 stable_sort。算法流程：

1. 如果图片的宽度比较宽，就先把分成左右w/h份。一般的640×480的图像开始的时候只有一个node。
2. 如果node里面的点数>1，把每个node分成四个node，如果node里面的特征点为空，就不要了，删掉。
3. 新分的node的点数>1，就再分裂成4个node。如此，一直分裂。
4. 终止条件为：node的总数量> 设定的数，或者无法再进行分裂。
5. 然后从每个node里面选择一个质量最好的FAST点。

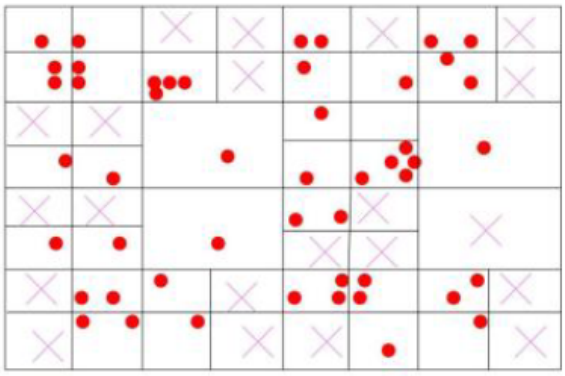
Step 3. 第2次分裂

- 4 node 分裂为15个node。因为有一个node里面没有点，所以不是16个
- node数量 15 < 25,还要接着分裂



Step 4. 第3次分裂

- 15 node 分裂为30个node
- node数量 30 > 25结束分裂



一个结点细分成四个子结点（主要是一些坐标边界的计算，算出这四个结点四个角的坐标），特征点再分配到四个区域-->ExtractorNode::DivideNode。

computeOrientation

遍历每个特征点并计算方向-->ORBextractor.cc中定义的IC_Angle,

IC_Angle

传入要进行操作的某层金字塔图像，当前特征点的坐标，图像块的每一行的坐标边界u_max，返回特征点的角度。

- 第1步：我们定义该区域图像的矩为：

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y), \quad p,q = \{0,1\} \tag{1}$$

式中，p,q取0或者1；$I(x,y)$表示在像素坐标(x,y)处图像的灰度值；m_{pq}表示图像的矩。

在半径为R的圆形图像区域，沿两个坐标轴x,y方向的图像矩分别为：

$$m_{10} = \sum_{x=-R}^{R} \sum_{y=-R}^{R} xI(x,y)$$

$$m_{01} = \sum_{x=-R}^{R} \sum_{y=-R}^{R} yI(x,y)$$

圆形区域内所有像素的灰度值总和为：

$$m_{00} = \sum_{x=-R}^{R} \sum_{y=-R}^{R} I(x,y)$$

- 第2步：图像的质心为：

$$C = (c_x, c_y) = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{2}$$

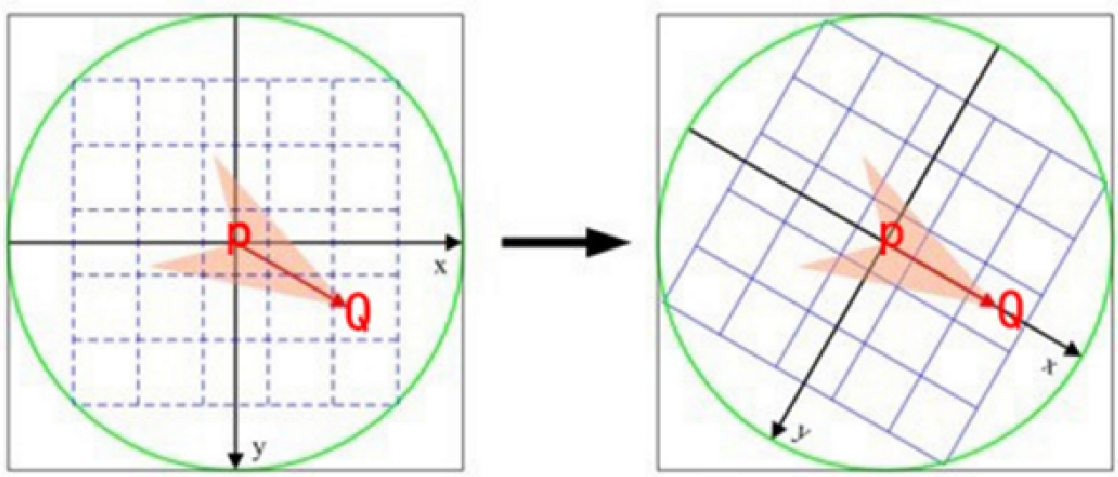
- 第3步：然后关键点的“主方向”就可以表示为从圆形图像形心O指向质心C的方向向量$\overrightarrow{OC}$,于是关键点的旋转角度记为

$$\theta = \arctan 2\left(c_y, c_x\right) = \arctan 2\left(m_{01}, m_{10}\right) \tag{3}$$

以上是灰度质心法求关键点旋转角度的原理。

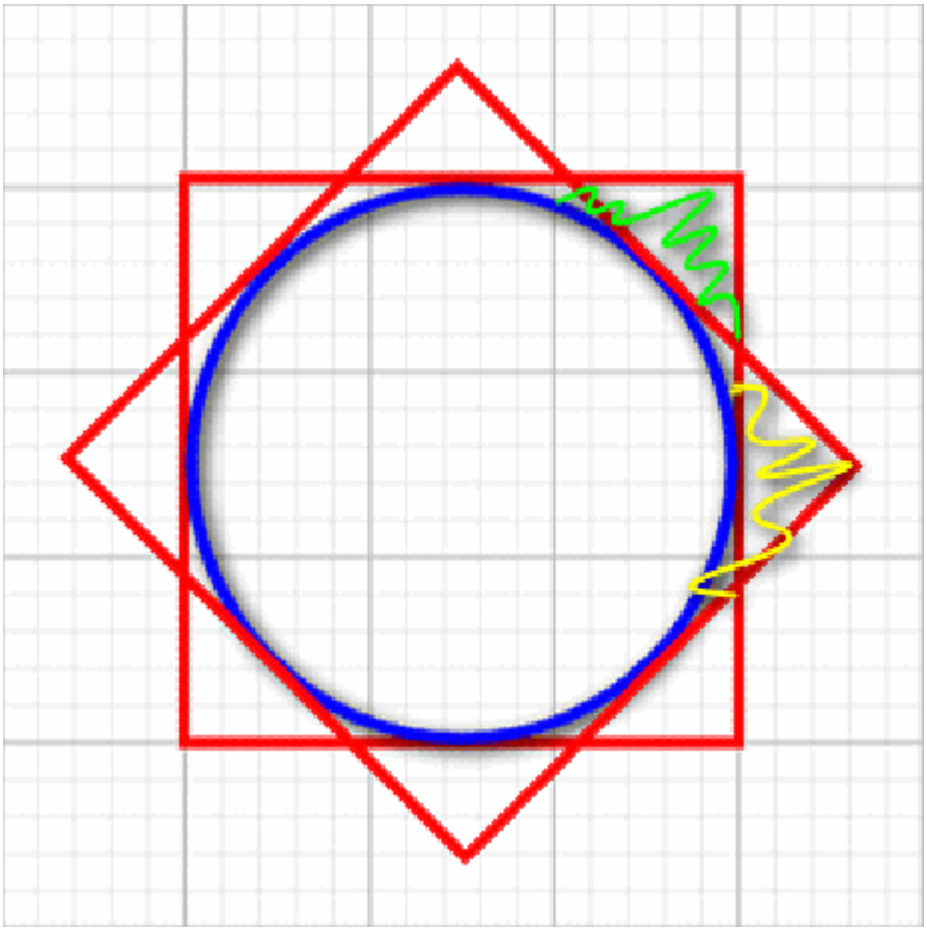
在一个圆内计算灰度质心

下图P为几何中心，Q为灰度质心



思考：为什么是圆？不是正方形？

ORBSLAM里面是先旋转坐标再从图像中采点提取，并不是先取那块图像再旋转，见



computeDescriptors

遍历特征点；计算描述子-->ORBextractor.cc中定义的computeOrbDescriptor

computeOrbDescriptor

输入为计算描述子的特征点、图像、随机点集的首地址、提取出来的描述子的保存位置。

原始的BRIEF描述子没有方向不变性，通过加入关键点的方向来计算描述子，称之为Steer BRIEF，具有较好旋转不变特性。获得pattern中点的灰度值，这里旋转前坐标为(x，y)， 旋转后坐标(x'，y')，他们的变换关系:$x' = xcos(\theta) - ysin(\theta)$，$y' = xsin(\theta) + ycos(\theta)$，并进行比较，得到32个8bit的描述子，每个bit都是通过pattern中的两个点比值大小来得到的。

Frame.h

Frame(Frame)类拷贝

拷贝一个新的Frame类

Frame单目帧构造函数

会传入ORBextractor这个类的指针，先得到ORBextractor中的金字塔层数、缩放因子等参数； 对单目图像进行提取特征点和描述子-->Frame::ExtractORB；用OpenCV的矫正函数、内参对提取到的特征点进行矫正-->Frame::UndistortKeyPoints。计算去畸变后图像边界-->Frame::ComputeImageBounds。将特征点分配到网格中（在后面匹配的时候会用到）-->Frame::AssignFeaturesToGrid。

ExtractORB

得到特征点和描述子-->ORBextractor::operator()(...)，（使用到了仿函数）。

UndistortKeyPoints

为了能够直接调用opencv的函数来去畸变，需要先将保存特征点坐标的矩阵调整为2通道（对应坐标x，y）；存储校正后的特征点，不是直接重新声明一个特征点对象，对其特征点坐标值进行修改，这样能够得到源特征点对象的其他属性。校正后的特征点浅拷贝到当前类的一个容器。

ComputeImageBounds

在去畸变后的图像的外侧加边框作为坐标的边界。

AssignFeaturesToGrid

遍历每个特征点，将每个特征点在mvKeysUn中的索引值放到对应的网格mGrid中。计算某个特征点所在网格的网格坐标-->Frame::PosInGrid

PosInGrid

有可能得到的去畸变特征点落在图像网格坐标外面，则不添加这个点到网格。

GetFeaturesInArea

输入窗口半径，求出左右上下边界所在的网格列和行的id；网格搜索速度快，遍历圆形区域内的所有网格（每个网格中存有特征点的id），寻找满足条件的候选特征点，并将其index放到输出里。

SetPose

简单的保存操作，保存世界坐标系到相机坐标系的变换矩阵、旋转矩阵、平移向量，相机坐标系到世界坐标系的变换矩阵，相机光心在世界坐标系下的坐标。

ComputeBoW

与关键帧中的ComputeBoW相同

isInFrustum

判断地图点是否在视野中。在视野中的条件：地图点变换到当前帧的相机坐标系下，如果深度值为正；地图点投影到当前帧的像素坐标，在图像有效范围内；地图点到相机中心的距离，在有效距离范围内（扩大了20%）；当前相机指向地图点向量和地图点的平均观测方向夹角，余弦值大于0.5（即夹角小于60度）。据地图点到光心的距离来预测处于哪个尺度-->MapPoint::PredictScale（仿照特征点金字塔层级）。

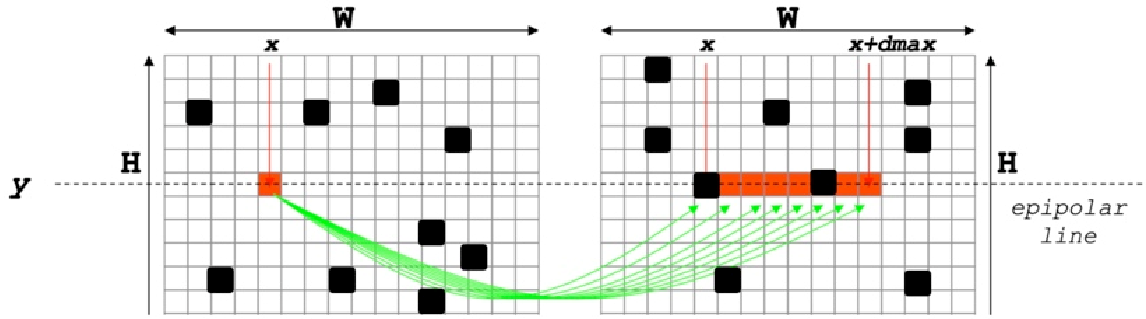
Frame双目帧构造函数

左右两张图片提取特征点双线程。大体与单目帧构造函数类似。在UndistortKeyPoints后多了一个双目间特征点的匹配的过程，只有匹配成功的特征点会计算其深度，深度存放在 mvDepth，-->Frame::ComputeStereoMatches。

ComputeStereoMatches

输入：两帧立体矫正后的图像img_left 和 img_right 对应的orb特征点集
*过程：

```
行特征点统计. 统计img_right每一行上的ORB特征点集，便于使用立体匹配思路(行搜索/极线搜索）进行同名点搜索， 避免逐像素的判断.
粗匹配. 根据步骤1的结果，对img_left第i行的orb特征点pi，在img_right的第i行上的orb特征点集中搜索相似orb特征点， 得到qi
精确匹配. 以点qi为中心，半径为r的范围内，滑动窗口，进行块匹配（归一化SAD），进一步优化匹配结果
亚像素精度优化. 步骤3得到的视差为uchar/int类型精度，并不一定是真实视差，通过亚像素差值（抛物线插值)获取float精度的真实视差
最优视差值/深度选择. 获取最佳匹配点。
删除离群点(outliers). 块匹配相似度阈值判断，取相似度中位数并乘一个系数，相似度大于该数则删除。归一化sad最小，并不代表就一定是正确匹配，比如光照变化、弱纹理等会造成误匹配
输出：稀疏特征点视差图/深度图（亚像素精度）mvDepth 匹配结果 mvuRight
```



ORBmatcher.h

ORBmatcher

只有传入参数并赋值给成员变量的操作，传入参数为（最优和次优评分的比例，是否检查方向）

SearchForInitialization

输入为初始化参考帧F1（类Frame）、当前参考帧F2（类Frame）、初始化参考帧中特征点坐标(经过此函数后更新为匹配好的当前帧的特征点坐标)、搜索窗口半径，返回匹配关系（index保存是F1对应特征点索引，值保存的是匹配好的F2特征点索引）。

遍历F1中的金字塔层数为0图像上的特征点，在半径窗口内搜索当前帧F2中候选匹配特征点-->Frame::GetFeaturesInArea，返回候选匹配点的id；遍历候选匹配点，计算描述子距离-->ORBmatcher::DescriptorDistance，记录匹配关系（通过最优和次优阈值），计算匹配点旋转角度差值，并存入直方图；找出直方图中数量最多的前三个bin-->ORBmatcher::ComputeThreeMaxima，删除其他的匹配点。

DescriptorDistance

计算两个二进制串之间的汉明距离，即不同位数的个数。并行bit位运算

ComputeThreeMaxima

找出前三个bin索引值，如果三个bin数量差距差距太大则只保存主要的。

SearchByBoW

从featurevector分别取出属于同一父node的ORB特征点(只有属于同一node，才有可能是匹配点)；遍历KF中属于该父node的特征点，再遍历F中属于该父node的特征点，寻找与KF中特征点描述子最佳和次佳汉明距离的点；根据阈值（最佳距离小于次佳距离的0.7）和角度投票（用到了角度直方图）剔除误匹配。得到了成功匹配的特征点所对应的地图点vpMapPointMatches。

SearchByProjection

将上一帧跟踪的地图点投影到当前帧，并且搜索匹配点，用于跟踪前一帧，得到了当前帧特征点对应的地图点及其个数。计算当前帧和前一帧的平移向量来判断前进还是后退（双目的时候才判断）；对于前一帧的每一个地图点，通过相机投影模型，得到投影到当前帧的像素坐标 PnP；根据相机的前后前进方向来判断搜索金字塔尺度范围-->Frame::GetFeaturesInArea；遍历候选匹配点，寻找距离最小的最佳匹配点；计算匹配点旋转角度差所在的直方图；进行旋转一致检测，剔除不一致的匹配。

SearchByProjection2

与SearchByProjection类似，但添加了未匹配的地图点才进行PnP投影的判断。

SearchByProjection3

与SearchByProjection类似。多了一项操作：通过当前相机指向地图点向量和地图点的平均观测方向夹角，更改阈值来改变搜索窗口的大小-->ORBmatcher::RadiusByViewingCos（通过投影点以及搜索窗口、地图点距离光心的距离预测的尺度进行搜索, 找出搜索半径内的候选匹配点索引-->Frame::GetFeaturesInArea）。但未使用旋转一致检测。

SearchByProjection4

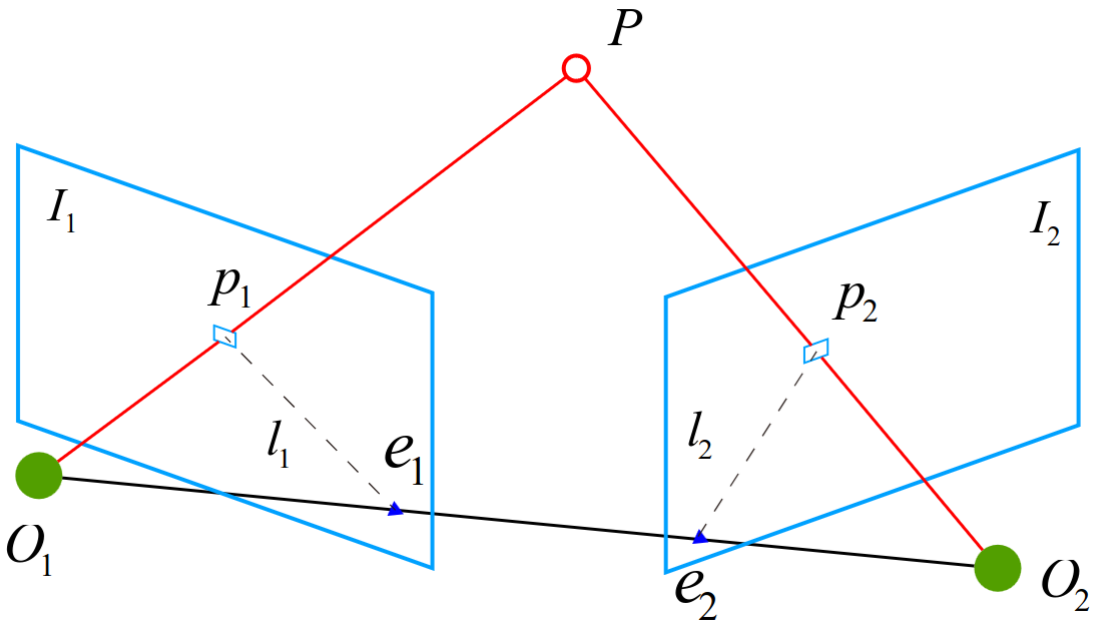
与SearchByProjection3类似

RadiusByViewingCos

当视角相差小于3.6°(对应cos(3.6°)=0.998)，搜索范围是2.5，否则是4。视角大则设置搜索范围大。

SearchForTriangulation

计算KF1的相机中心在KF2图像平面的二维像素坐标（即为图中的e_2极点）；利用BoW加速匹配，与ORBmatcher::SearchByBoW类似，但是其中多了对极几何中极点、极线的判断（若极点e_2到p_2到的像素距离如果小于阈值，认为p_2对应的MapPoint距离O_1相机太近，跳过该匹配点对；计算特征点p_2到p_1对应极线l_2的距离是否小于阈值-->ORBmatcher::CheckDistEpipolarLine）。



CheckDistEpipolarLine

通过特征点p_1和基础矩阵求出p_1在KF2上对应的极线l_2；计算p_2特征点到极线l_2的距离；判断误差是否小于阈值（根据特征点所在金字塔层级，尺度越大，误差允许范围应越大）

$l_2 = p_1^T \cdot F_{12} = [a\ b\ c]$　　则极线l_2：$ax + by + c = 0$
$p_2\ (u, v)$到l_2的距离为：$|au + bv + c| / sqrt(a^2 + b^2)$

Fuse

遍历所有的待投影地图点：跳过该地图点的条件（地图点无效或已经是该帧的地图点、深度值为负）；得到地图点投影到关键帧的图像坐标（投影点坐标需要在图像的有效范围内、地图点到光心距离需在有效范围内[根据地图点成员变量mfMaxDistance、mfMinDistance]、地图点到光心的连线与该地图点的平均观测向量之间夹角要小于60°）；预测投影的点在图像金字塔哪一层-->MapPoint::PredictScale，得到搜索半径，再搜索该区域内的所有候选匹配特征点-->KeyFrame::GetFeaturesInArea；遍历寻找最佳匹配点；找到投影点对应的最佳匹配特征点后，根据是否存在地图点来融合或新增，1.最佳匹配特征点有对应的地图点，那么从这两个地图点选择观测数目多的保留，另外一个进行地图点替换操作-->MapPoint::Replace 2.最佳匹配特征点没有对应的地图点，那么为该点添加该投影地图点。

Fuse2

闭环矫正中使用。将闭环相连关键帧组的地图点投影到当前关键帧，融合地图点。与ORBmatcher::Fuse类似，但是不需要比较两个地图点选择观测数目，直接选择保留闭环相连关键帧组的地图点（因为其在地图中时间比较久经历了多次优化，认为是准确的），而且这个函数里面也没有进行替换的操作，只是记录了一下。

SearchBySim3

计算Sim3的逆 $S = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{s}R^T & -\frac{1}{s}R^T t \\ 0 & 1 \end{bmatrix}$（S相当于添加了尺度的变换矩阵，需要通过S和S的逆来计算地图点在pKF1和pKF2下的坐标）；提前记录了已经匹配过了的特征点（在后面新匹配时就跳过这些特征点），寻找 pKF1 中地图点和 pKF2 中的新的匹配、 pKF2 中地图点和 pKF1 中的新的匹配，与ORBmatcher::SearchByProjection类似（未使用旋转一致性、次佳汉明距

离）；最后进行一致性检查,只有在两次互相匹配中都出现才能够认为是可靠的匹配。

Initializer.h

Initializer

传入了参考帧、测量误差、RANSAC迭代次数。提取了内参

Initialize

重新记录特征点对的匹配关系mvMatches12，只保存有匹配的点对；在所有匹配特征点对中随机选择8对匹配特征点为一组，用于估计H矩阵和F矩阵(ransac次)；计算homography矩阵-->Initializer::FindHomography 和 fundamental 矩阵 -->Initializer::FindFundamental，开了两个线程分别计算； 计算得分比例来判断选取哪个模型来求位姿R，t；从H或F中恢复位姿-->Initializer::ReconstructH，-->Initializer::ReconstructF

FindHomography

- Step 1 将当前帧和参考帧中的特征点坐标进行归一化-->Initializer::Normalize
- Step 2 选择8个归一化之后的点对进行迭代
- Step 3 八点法计算单应矩阵矩阵-->Initializer::ComputeH21，是对归一化点求出的单应矩阵，还要变为实际的单应矩阵
- Step 4 利用重投影误差为当次RANSAC的结果评分-->Initializer::CheckHomography 卡方检验
- Step 5 更新具有最优评分的单应矩阵计算结果，并且保存所对应的特征点对的内点标记

卡方检验相关知识

Normalize

将参考帧和当前帧特征点的x坐标和y坐标分别进行尺度归一化，使得x坐标和y坐标的均值为0，一阶绝对矩为1 。构建出归一化矩阵T。

ComputeH21

八点法的单应矩阵求解。n为平面法向量，d为平面到坐标系原点的距离，P为特征点的世界坐标

$$\boldsymbol{n}^T\boldsymbol{P} + d = 0. \tag{7.16}$$

稍加整理，得：

$$-\frac{\boldsymbol{n}^T\boldsymbol{P}}{d} = 1. \tag{7.17}$$

然后，回顾本开头的式 (7.1)，得：

$$\begin{aligned}
\boldsymbol{p}_2 &= \boldsymbol{K}(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}) \\
&= \boldsymbol{K}\left(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}\cdot(-\frac{\boldsymbol{n}^T\boldsymbol{P}}{d})\right) \\
&= \boldsymbol{K}\left(\boldsymbol{R} - \frac{\boldsymbol{t}\boldsymbol{n}^T}{d}\right)\boldsymbol{P} \\
&= \boldsymbol{K}\left(\boldsymbol{R} - \frac{\boldsymbol{t}\boldsymbol{n}^T}{d}\right)\boldsymbol{K}^{-1}\boldsymbol{p}_1.
\end{aligned}$$

于是，我们得到了一个直接描述图像坐标 $\boldsymbol{p}_1$ 和 $\boldsymbol{p}_2$ 之间的变换，把中间这部分记为 $\boldsymbol{H}$，于是

$$\boldsymbol{p}_2 = \boldsymbol{H}\boldsymbol{p}_1. \tag{7.18}$$

特征点对p_1, p_2，用单应矩阵H_{21} 来描述特征点对之间的变换关系

$$p_2 = H_{21} * p_1$$

我们写成矩阵形式:

$$\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

为了化为齐次方程，左右两边同时叉乘p_2，得到

$$p_2 \times H_{21} * p_1 = 0$$

写成矩阵形式：

$$\begin{bmatrix} 0 & -1 & v_2 \\ 1 & 0 & -u_2 \\ -v_2 & u_2 & 0 \end{bmatrix} \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = 0$$

展开计算得到

$$v_2 = (h_4 * u_1 + h_5 * v_1 + h_6)/(h_7 * u_1 + h_8 * v_1 + h_9)$$
$$u_2 = (h_1 * u_1 + h_2 * v_1 + h_3)/(h_7 * u_1 + h_8 * v_1 + h_9)$$

写成齐次方程

$$-(h_4 * u_1 + h_5 * v_1 + h_6) + (h_7 * u_1 * v_2 + h_8 * v_1 * v_2 + h_9 * v_2) = 0$$
$$h_1 * u_1 + h_2 * v_1 + h_3 - (h_7 * u_1 * u_2 + h_8 * v_1 * u_2 + h_9 * u_2) = 0$$

转化为矩阵形式

$$\begin{bmatrix} 0 & 0 & 0 & -u_1 & -v_1 & -1 & u_1 * v_2 & v_1 * v_2 & v_2 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1 * u_2 & -v_1 * u_2 & -u_2 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0$$

等式左边两项分别用A, X表示，则有

$$AX = 0$$

一对点提供两个约束等式，单应矩阵H总共有9个元素，8个自由度（尺度等价性），所以需要4对点提供8个约束方程就可以求解。

用四对点就可以求出，但是代码中为了统一使用了八对点求最小二乘解；用到了奇异值分解求线性方程的知识。

> 奇异值分解（SVD）：
>
> 设$\boldsymbol{A} \in \mathbb{C}_r^{m \times n}$，则存在正交矩阵$\boldsymbol{U} \in \mathbb{C}^{m \times m}$与正交矩阵$\boldsymbol{V} \in \mathbb{C}^{n \times n}$使得
> $\boldsymbol{A}=\boldsymbol{U}\begin{bmatrix}\boldsymbol{\Sigma}_r & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\boldsymbol{V}^T=\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$，其中$\boldsymbol{\Sigma_r}=diag(\sigma_1,\ldots,\sigma_r)$是矩阵$\boldsymbol{A}$的正奇异值

- 通过奇异值分解将原问题转换为最小二乘问题，可以求解超定线性方程组

1 求解齐次线性方程组（$\boldsymbol{Ax}=\boldsymbol{0}$）

问题等价于$\min_{\boldsymbol{x}}\ ||\boldsymbol{Ax}||^2$，则

$$\begin{aligned}&\min_{\boldsymbol{x}}||\boldsymbol{Ax}||^2\\ =&\min_{\boldsymbol{x}}||\boldsymbol{UDV}^T\boldsymbol{x}||^2\\ =&\min_{\boldsymbol{x}}||\boldsymbol{DV}^T\boldsymbol{x}||^2\end{aligned}\tag{1}$$

令$\boldsymbol{y}=\boldsymbol{V}^T\boldsymbol{x}$，则$(1)$式变为$\min_{\boldsymbol{y}}||\boldsymbol{Dy}||^2=\min_{\boldsymbol{y}}\left(\sigma_1^2y_1^2+\sigma_2^2y_2^2+\ldots+\sigma_r^2y_r^2\right)$，其中$r<=n$

因为$\sigma_1>\sigma_2>\ldots>\sigma_r>0$，所有不管$r$是否等于$n$，在$||\boldsymbol{x}||=1$的前提下的最优解均在$\boldsymbol{y}=[0,0,\ldots,1]^T$时取得（$r=n$时存在最小二乘解，$r<n$时存在数值解），此时$\boldsymbol{x}=\boldsymbol{Vy}$，即$\boldsymbol{V}$的最后一列

CheckHomography

利用H矩阵，进行参考帧和当前帧之间的双向投影，计算投影误差，并计算单应矩阵的得分。双向投影误差都小于卡方检验的阈值则为内点，否则外点。

FindFundamental

与计算H矩阵过程类似，但是两者去归一化的计算不相同（这里用的坐标归一化矩阵的转置）。

- Step 1 将当前帧和参考帧中的特征点坐标进行归一化-->Initializer::Normalize
- Step 2 选择8个归一化之后的点对进行迭代
- Step 3 八点法计算基础矩阵矩阵-->Initializer::ComputeF21
- Step 4 利用重投影误差为当次RANSAC的结果评分-->Initializer::CheckFundamental
- Step 5 更新具有最优评分的基础矩阵计算结果，并且保存所对应的特征点对的内点标记

ComputeF21

注意F矩阵有秩为2的约束，所以需要两次SVD分解。具体推导slam14讲或者手册

CheckFundamental

利用F矩阵，双向投影，得到极线，求点到极线的距离即为误差，并计算基础矩阵的得分。使用了两个卡方阈值。同样判断了内点和外点。

ReconstructH

主要是根据论文代入公式，四种情况进行选择（算了八组R、t，具体怎么算的没看），选择相机前方最多的3D点的解，得到R、t，验证R、t的合法性-->Initializer::CheckRT。会返回可三角化的点的空间坐标。

CheckRT

计算相机的投影矩阵（将空间中的一个点投影到平面上，获得其平面坐标），第一个相机的光心为世界坐标原点；使用特征点对，得到三角化测量之后的3D点坐标-->Initializer::Triangulate；1检查3D点坐标是否合法（非无穷值）、2深度非负值与两相机光心视差角要大于1弧度、3空间点重投影到参考帧和当前帧，计算与特征点坐标误差。都通过了则是good点；

Triangulate

通过两个投影矩阵和特征点坐标来恢复3D点的坐标。中间的为投影矩阵

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

ReconstructF

从基础矩阵F中先得到本质矩阵E；求出4种R、t的组合-->Initializer::DecomposeE；分别检查四组解合法性-->Initializer::CheckRT，得到good点数及坐标；四个结果中如果good点数接近，或good点数小于阈值，则失败。会返回可三角化的点的空间坐标。

DecomposeE

使用SVD分解本质矩阵，得到两个R和一个t

分解本质矩阵 Essential Matrix

$$E = [t]_\times R \tag{3}$$

SVD分解

$$E = Udiag(1,1,0)V^T = (UZU^T)(UXV^T) = [t]_\times R \tag{4}$$

$$Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Z表示反对称矩阵，W表示正交矩阵（第一类正交矩阵就是旋转矩阵），但是这是在90度的情况下，如果是270度的时候 $W_{90} = -W_{270}$ （负号只表示旋转部分），如下：

$$W_{270} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = W_{90}^T \tag{6}$$

因此考虑上面两种情况，就得出X有2种可能：

$$X = WorW^T \tag{7}$$

对于平移矩阵

$$\mathbf{t} = U(0,0,1)^T = \mathbf{u}_3 \tag{8}$$

考虑 $-E$ 的情况，因此最终的平移矩阵也有2种可能

$$\mathbf{t} = -\mathbf{u}_3 or \mathbf{u}_3 \tag{9}$$

KeyFrame.h

KeyFrame

主要是对Frame变量的各种拷贝操作，记录KeyFrame总数目及此KeyFrame对应的Frame，设置当前KeyFrame的位姿-->

KeyFrame::SetPose

SetPose

简单的位姿等信息的保存操作

ComputeBoW

计算词袋表示，从当前帧的描述子中转换得到词袋信息-->Converter::toDescriptorVector；通过调用BoW中的`transform`函数，ORB 词袋模型将特征描述子转换为两个表示：词袋向量（`mBowVec`）和特征向量（`mFeatVec`）-->词袋模型。

AddMapPoint

记录该关键帧可以观测到的地图点的索引

UpdateConnections

获得该关键帧的所有MapPoint点，统计每个地图点能被哪些关键帧观测到；计算边的权重，为该关键帧与观测到地图点的关键帧公共3d点的个数（巧妙的利用map数据结构进行记录）；共视地图点大于一个阈值则建立共视关系，（如果没有超过阈值的权重，则只对权重最大的关键帧建立连接），当然对方关键帧也要添加边权重信息-->KeyFrame::AddConnection；更新生成树的连接，初始化该关键帧的父关键帧为共视程度最高的那个关键帧，建立双向连接关系，将当前关键帧作为其儿子关键帧-->KeyFrame::AddChild

AddConnection

新建或更新连接权重；更新最佳共视，主要是重新进行排序-->KeyFrame::UpdateBestCovisibles

UpdateBestCovisibles

按照权重从大到小对连接（共视）的关键帧进行排序，更新后的变量存储在mvpOrderedConnectedKeyFrames和mvOrderedWeights中

AddChild

添加儿子关键帧到关键帧mspChildrens成员变量

ComputeSceneMedianDepth

计算当前关键帧的所有地图点在当前关键帧坐标系下的深度，并进行从小到大排序,返回位置处于1/q处的深度值作为当前场景的平均深度

TrackedMapPoints

关键帧中，大于等于最少观测数目minObs的MapPoints的数量.这些特征点被认为追踪到了

GetBestCovisibilityKeyFrames

返回前N个最强共视关键帧(已按权值排序)

TrackedMapPoints

返回大于等于最少观测数目minObs（指观测到该地图点的帧的数量）的MapPoints的数量。这些特征点被认为追踪到了

EraseMapPointMatch

根据传入的地图点索引值，将与该地图点从mvpMapPoints中删除（赋值为空指针）。

GetFeaturesInArea

与Frame::GetFeaturesInArea类似

SetBadFlag

第一个关键帧不允许被删除，判断有些关键帧现在还不能进行删除；遍历当前关键帧的共视关键帧，删除他们与当前关键帧的联系-->KeyFrame::EraseConnection；遍历当前关键帧的地图点，删除每一个地图点和当前关键帧的联系-->MapPoint::EraseObservation；更新生成树，主要是处理好父子关键帧
TODO

EraseConnection

从mConnectedKeyFrameWeights中删除当前关键帧，并更新共视关系-->KeyFrame::UpdateBestCovisibles

SetNotErase

mbNotErase = true，这是为了让当前关键帧不要在优化的过程中被删除。由回环检测线程调用

SetErase

如果当前关键帧和其他的关键帧没有形成回环关系（mspLoopEdges为空）,那么就设置mbNotErase = false；若mbToBeErased（删除之前记录的想要删但时机不合适没有删除的帧），则-->KeyFrame::SetBadFlag。

Map.h

Map

构造函数，地图点中最大关键帧id归0

AddKeyFrame

在地图中插入关键帧，同时更新关键帧的最大id

AddMapPoint

向地图中插入地图点

SetReferenceMapPoints

设置参考地图点用于绘图显示局部地图点（红色）

EraseMapPoint

从地图中删除地图点，删除了地图点的指针但是其实这个地图点所占用的内存空间并没有被释放。
mspMapPoints.erase(pMP)

MapPoint.h

MapPoint

初始化地图点的各种信息，计算mfMaxDistance、mfMinDistance（由于特征点在不同金字塔图层提取到，对应着该特征点生成的地图点距离相机光心的远近）

AddObservation

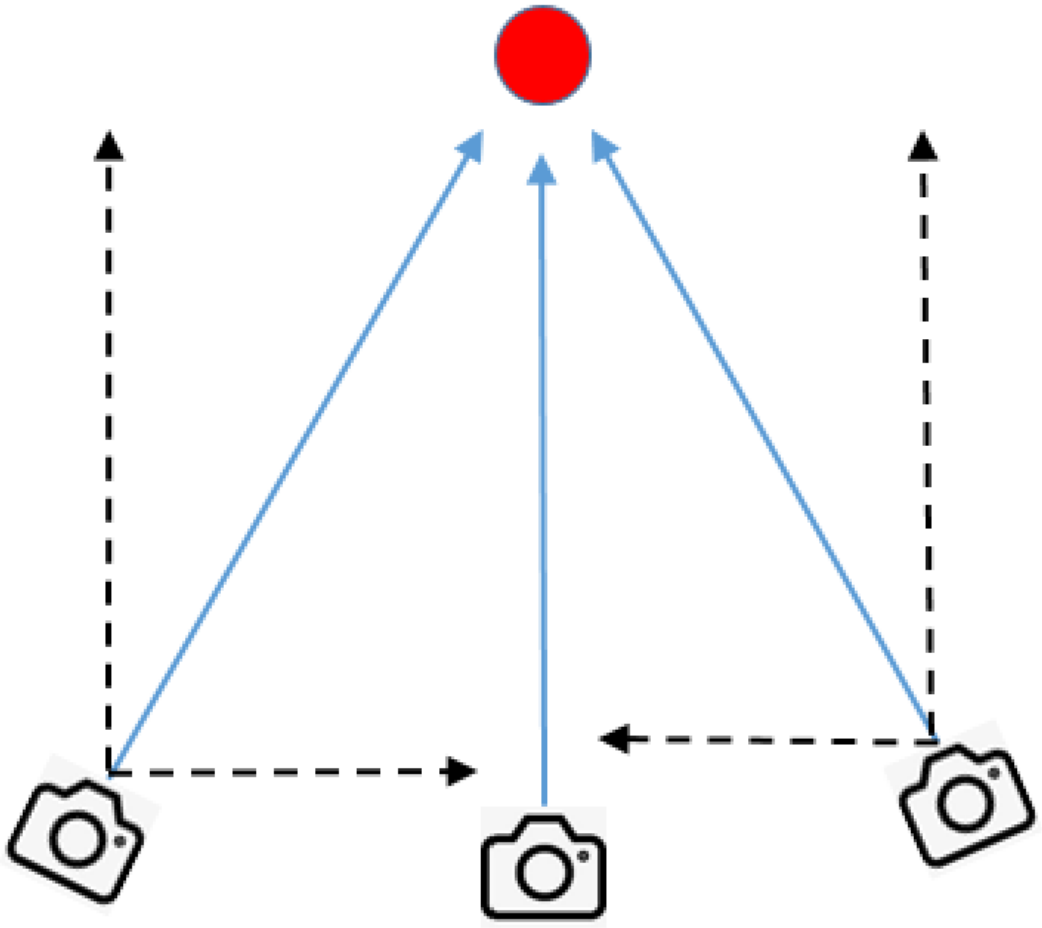
给地图点添加观测mObservations，记录该地图点能被哪些关键帧KF观测到，并记录该地图点在KF中的索引；记录观测到该地图点的KF数目

ComputeDistinctiveDescriptors

遍历观测到该地图点的所有关键帧，对应的orb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（中位数最小）。

UpdateNormalAndDepth

获得观测到该地图点的所有关键帧、坐标等信息；计算该地图点的平均观测方向(地图点与相机光心连线的向量归一化再取均值)；计算mfMaxDistance、mfMinDistance距离光心的上下限。



SetBadFlag

告知可以观测到该MapPoint的Frame，该MapPoint已被删除。遍历该地图点的mObservations，告诉可以观测到该地图点的KeyFrame，该地图点被删了-->KeyFrame::EraseMapPointMatch；

PredictScale

```
// 下图中横线的大小表示不同图层图像上的一个像素表示的真实物理空间中的大小
//                 ____
// Nearer         /____\     level:n-1 --> dmin
//               /______\                       d/dmin = 1.2^(n-1-m)
//              /________\   level:m    --> d
//             /__________\                     dmax/d = 1.2^m
// Farther    /____________\ level:0    --> dmax
//
//              log(dmax/d)
// m = ceil(-------------)
//               log(1.2)
// 这个函数的作用:
// 在进行投影匹配的时候会给定特征点的搜索范围,考虑到处于不同尺度(也就是距离相机远近,位于图像金字塔中不同图层)的特征点受到相机旋转的影响不同,
// 因此会希望距离相机近的点的搜索范围更大一点,距离相机更远的点的搜索范围更小一点,所以要在这里,根据点到关键帧/帧的距离来估计它在当前的关键帧/帧中,
// 会大概处于哪个尺度
```

Replace

这两个地图点为同一个地图点则跳过（根据Id判断）；先清除当前地图点信息，与MapPoint::SetBadFlag类似；将观测到当前地图的的关键帧的信息进行更新（若当前地图点被某关键帧观测到，但是新地图点没被某关键帧观测到，则某关键帧中mvpMapPoints的当前地图点替换为新地图点，并新地图点的mObservations添加某关键帧的观测-->MapPoint::AddObservation；若当前地图点被某关键帧观测到，且新地图点也被某关键帧观测到，则直接删除当前地图点即可-->KeyFrame::EraseMapPointMatch）；将当前地图点的观测数据等其他数据都"叠加"到新的地图点上，描述子更新-->MapPoint::ComputeDistinctiveDescriptors，告知地图删除当前地图点-->Map::EraseMapPoint。

EraseObservation

从地图点的mObservations变量中，删除此关键帧对当前地图点的观测，若此关键帧为当前地图点的参考关键帧（创建当前地图点的帧），则直接将mObservations.begin()->first设置为参考关键帧。如果该地图点被设置bBad，则MapPoint::SetBadFlag

Converter.h

toDescriptorVector

将描述子转换为描述子向量，其实本质上是cv:Mat转为std:vector<cv::Mat>

Optimizer.h

优化器,所有的优化相关的函数都在这个类中; 并且这个类只有成员函数没有成员变量

GlobalBundleAdjustemnt

获取地图中的所有关键帧和地图点，调用GBA-->Optimizer::BundleAdjustment

PoseOptimization

3D-2D 最小化重投影误差,只优化Frame的Tcw，不优化MapPoints的坐标。TODO：g2o的相关知识

g2o相关链接： 简单来说包含以下几步

1、创建线性求解器LinearSolver；

```
g2o::BlockSolver_6_3::LinearSolverType * linearSolver= new
g2o::LinearSolverDense<g2o::BlockSolver_6_3::PoseMatrixType>();
```

2、创建BlockSolver并用线性求解器LinearSolver来初始化；

```
g2o::BlockSolver_6_3 * solver_ptr = new g2o::BlockSolver_6_3(linearSolver);
```

3、创建总求解器solver，并从GN, LM, DogLeg 中选一个，再用上述块求解器BlockSolver初始化；

```
g2o::OptimizationAlgorithmLevenberg* solver = new
g2o::OptimizationAlgorithmLevenberg(solver_ptr);
```

4、创建稀疏优化器（SparseOptimizer），并用已定义总求解器solver作为求解方法；

```
g2o::SparseOptimizer optimizer;
optimizer.setAlgorithm(solver);
```

5、添加关键帧位姿顶点、地图点顶点到优化器optimizer；添加每一对关联的地图点和关键帧构建的边到优化器optimizer

6、最后设置优化参数，开始执行优化 (设置SparseOptimizer的初始化、迭代次数、保存结果等)

```
    optimizer.initializeOptimization();
    optimizer.optimize(5);  //迭代5次
```

LocalBundleAdjustment

用于LocalMapping线程的局部BA优化。

将当前关键帧及其一级共视关键帧存入局部关键帧lLocalKeyFrames；遍历局部关键帧lLocalKeyFrames，将它们观测的地图点加入到局部地图点lLocalMapPoints；遍历局部地图点，将能看到该地图点但不属于局部关键帧lLocalKeyFrames的关键帧存入lFixedCameras（即除了lLocalKeyFrames的二级共视关键帧），lFixedCameras在局部BA优化时不优化，起到添加约束，进行监督的作用；构造g2o优化器，与Optimizer::PoseOptimization类似；添加待优化的局部关键帧lLocalKeyFrames的位姿顶点，和不进行优化的固定关键帧lFixedCameras的位姿顶点（这里的位姿转换为了李代数se3-->Converter::toSE3Quat）；添加待优化的局部地图点lLocalMapPoints的顶点时（使用到了边缘化的操作），对每一对关联的地图点和关键帧构建边（根据金字塔层数设置边的权重，使用到了鲁棒核函数抑制外点）；进行第一次优化，迭代5次；第一段优化后的基础上，根据边的误差或地图点深度判断是否取消该边，进行第二次优化（不使用鲁棒核函数），迭代10次；在优化后重新计算误差，剔除连接误差比较大的关键帧和地图点的观测关系-->KeyFrame::EraseMapPointMatch，MapPoint::EraseObservation；使用优化后的值来更新关键帧位姿以及地图点的位置（Converter::toCvMat）、平均观测方向-->MapPoint::UpdateNormalAndDepth等属性。

OptimizeSim3

初始化g2o优化器，使用BlockSolverX；设置待优化的Sim3变换矩阵作为顶点；设置匹配的地图点作为顶点（当前帧的地图点和与对应的候选帧地图点，添加的坐标是地图点转换到各自相机坐标系下的坐标，并且地图点不进行优化）；候选帧地图点与Sim3变换矩阵构成边——正向投影，当前帧地图点与Sim3变换矩阵构成边——反向投影（边的误差的计算与Sim3Solver::CheckInliers误差计算类似）；进行第一次优化，迭代5次，用卡方检验剔除误差大的边（若有误差大的边，增加第二次的迭代次数）；第二次优化，得到内点及内点数，并用优化后的Sim3顶点来更新Sim3矩阵。

OptimizeEssentialGraph

闭环检测后，EssentialGraph优化，仅优化所有关键帧位姿，不优化地图点。

构造优化器，使用BlockSolver_7_3；将地图中所有关键帧的位姿（优化第一个关键帧，不优化闭环关键帧）作为顶点（尽可能使用经过Sim3调整的位姿g2oCorrectedSiw）；闭环时因为地图点调整而出现的关键帧间的新连接关系LoopConnections（必添加当前帧与闭环帧，其他的任意两帧间共视程度大于100，观测值尽可能使用经过Sim3调整的位姿）——**边1**；遍历所有关键帧，生成树的边（有父关键帧，观测值优先使用未经过Sim3传播调整的位姿g2oSiw）——**边2**，当前帧与闭环匹配帧mspLoopEdges之间的连接关系（观测值优先使用未经过Sim3传播调整的位姿g2oSiw）——**边3**，共视程度超过100的关键帧（优先未经过Sim3传播调整的位姿）——**边4**；开始优化，迭代20次；将优化后的位姿更新到关键帧中；地图点根据其参考帧使用相对位姿变换（先使用旧位姿转换到相机坐标系，再使用优化后的位姿转换回世界坐标系）的方法来更新地图点的3D坐标。

BundleAdjustment

初始化g2o优化器；向优化器添加关键帧位姿作为顶点，地图点作为顶点，地图点和关键帧的投影关系作为边；开始优化，迭代10次；得到优化的结果，如果是初始化调用此BA函数，则直接将优化后的关键帧位姿和地图点位置数据替代旧数据即可，而正常调用此BA函数，则存入mTcwGBA和mPosGBA中备用，并记录由哪一当前帧ID mnBAGlobalForKF调用的GBA

LocalMapping.h

LocalMapping

各种标志位的初始值设置

Run

局部建图的主函数，一个一直进行的循环。设置"允许接受关键帧"的状态标志为False；

若如果等待处理的关键帧列表mlNewKeyFrames不为空：处理列表中的关键帧，包括计算BoW、更新观测、描述子、共视图，插入到地图等-->LocalMapping::ProcessNewKeyFrame；根据地图点的观测情况剔除质量不好的地图点-->LocalMapping::MapPointCulling；当前关键帧与相邻关键帧通过三角化产生新的地图点，使得跟踪更稳-->LocalMapping::CreateNewMapPoints；若处理完关键帧列表mlNewKeyFrames中的最后的一个关键帧，则检查并融合当前关键帧与相邻关键帧帧（两级相邻）中重复的地图点-->LocalMapping::SearchInNeighbors；若处理完队列中的最后的一个关键帧即关键帧列表mlNewKeyFrames为空，且闭环检测没有请求停止LocalMapping时，进行局部地图的BA-->Optimizer::LocalBundleAdjustment (mbAbortBA会被按地址传入这个函数，当mbAbortBA 状态发生变化时，能够及时执行/停止BA)，检测并剔除当前帧相邻的关键帧中冗余的关键帧-->LocalMapping::KeyFrameCulling；将当前帧加入到闭环检测队列中-->LoopClosing::InsertKeyFrame。当关键帧列表mlNewKeyFrames都处理完了，局部建图线程不busy，就SetAcceptKeyFrames，tracking线程就又可以给关键帧列表mlNewKeyFrames传关键帧了。然后根据各种标志位或者外部请求，来选择是否关闭局部建图线程。

InsertKeyFrame

将关键帧插入到关键帧列表mlNewKeyFrames中

ProcessNewKeyFrame

从关键帧缓冲队列中取出列表的第一帧关键帧（该关键帧队列是Tracking线程向LocalMapping中插入的关键帧组成，在Tracking::CreateNewKeyFrame中插入的）；计算该关键帧特征点的词袋向量-->KeyFrame::ComputeBoW。处理当前关键帧中有效的地图点，若该地图点不是来自当前帧的观测，则为当前地图点添加观测，更新平均观测方向和观测距离范围-->MapPoint::UpdateNormalAndDepth，描述子等信息-->MapPoint::ComputeDistinctiveDescriptors；否则将上述地图点放入mlpRecentAddedMapPoints，等待后续LocalMapping::MapPointCulling函数的检验。更新关键帧间的连接关系（共视图）-->KeyFrame::UpdateConnections；将该关键帧插入到地图中-->Map::AddKeyFrame。

MapPointCulling

根据相机类型设置不同的观测阈值cnThObs；遍历检查新添加的地图点mlpRecentAddedMapPoints：已经是坏点的地图点仅从队列中删除；实际跟踪到该地图点的帧数相比预计可观测到该地图点的帧数（通过视角来进行的记录）的比例小于25%，从地图中删除-->MapPoint::SetBadFlag；从该地图点建立开始，到现在已经过了不小于2个关键帧且观测到该点的相机数却不超过阈值cnThObs，从地图中删除-->MapPoint::SetBadFlag；从建立该点开始，已经过了3个关键帧而没有被剔除，则认为是质量高的点，仅从队列中删除；如果前三个条件都不满足，则跳过此地图点，等待下一次进入此函数时再对其进行判断。

CreateNewMapPoints

根据相机类型设置共视关键帧的数目nn；找到共视程度最高的nn帧相邻关键帧；特征点匹配配置ORBmatcher(0.6,false)；取出当前帧的变换矩阵、光心坐标及内参。遍历相邻关键帧：通过两帧之间相机的光心距离判断此相邻关键帧是否可靠（双目则与基线相比，单目则与场景深度中值比-->KeyFrame::ComputeSceneMedianDepth）；根据两个关键帧的位姿计算它们之间的基础矩阵-->LocalMapping::ComputeF12；通过词袋对两关键帧的未匹配的特征点快速匹配，用极线约束抑制离群点，生成新的匹配点对-->ORBmatcher::SearchForTriangulation；遍历匹配的特征点：利用匹配点反投影得到视差角（TODO，视差角计算感觉有点奇怪？）（对于双目，可直接得到左右目对此点的视差角）；根据视差角大小的判断，选择使用用三角法恢复3D点（与Initializer::Triangulate类似，这里把投影矩阵换成了变换矩阵，数学上的变换）or 双目直接恢复3D点；检查深度正负、重投影误差、根据金字塔和深度的尺度连续性；构造成MapPoint、并为该MapPoint添加属性。

ComputeF12

数学公式的应用

ComputeF12

$$T_{1w} = \begin{bmatrix} R_{1w} & t_{1w} \\ 0^T & 1 \end{bmatrix}$$

$$T_{2w} = \begin{bmatrix} R_{2w} & t_{2w} \\ 0^T & 1 \end{bmatrix}$$

$$T_{12} = T_{1w} {T_{2w}}^{-1} = \begin{bmatrix} R_{1w} & t_{1w} \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} R_{2w}^T & -R_{2w}^T t_{2w} \\ 0^T & 1 \end{bmatrix}$$

$$= \begin{bmatrix} R_{1w} R_{2w}^T & -R_{1w} R_{2w}^T t_{2w} + t_{1w} \\ 0^T & 1 \end{bmatrix}$$

再$\boldsymbol{K^{-T}t\hat{}RK^{-1}}$

SearchInNeighbors

获得当前关键帧在共视图中权重排名前nn的邻接关键帧；存储一级相邻关键帧及其二级相邻关键帧向量vpTargetKFs（存储之后标记一下，防止重复添加存储）；遍历存储的关键帧向量vpTargetKFs，将当前帧的地图点分别投影到两级相邻关键帧，寻找匹配点对应的地图点进行融合，称为正向投影融合-->ORBmatcher::Fuse；遍历存储的关键帧向量vpTargetKFs，收集他们的地图点存储到vpFuseCandidates（同样通过标记的方法防止重复），将存储的地图点分别投影到当前关键帧，称为反向投影融合-->ORBmatcher::Fuse；更新当前帧地图点的描述子-->MapPoint::ComputeDistinctiveDescriptors、平均观测方向-->MapPoint::UpdateNormalAndDepth；更新当前帧与其它帧的共视连接关系-->KeyFrame::UpdateConnections。

KeyFrameCulling

冗余关键帧的判定：90%以上的地图点能被其他关键帧（至少3个）观测到。

对当前关键帧的所有一级共视关键帧进行遍历（跳过第一个关键帧）；对一级共视关键帧的地图点遍历；若地图点Observations大于3，则对能观测到该地图点的关键帧（二级共视关键帧）进行遍历，若满足尺度约束（该地图点在二级共视关键帧中所对应特征点的金字塔尺度 < 该地图点在一级共视关键帧中所对应特征点的金字塔尺度+1），则能观测到该地图点的关键帧数加1；若地图点至少被3个关键帧观测到，就记录为冗余点；如果该一级共视关键帧90%以上的有效地图点被判断为冗余的，需要删除该关键帧-->KeyFrame::SetBadFlag。

LoopClosing.h

LoopClosing

传递地图、关键帧数据库、ORB字典参数，其他各值的初始化。

Run

若关键帧列表mlpLoopKeyFrameQueue不为空；闭环检测，寻找闭环候选帧集-->LoopClosing::DetectLoop；闭环成功，则计算当前关键帧和闭环候选帧集的Sim3变换-->LoopClosing::ComputeSim3；闭环矫正-->LoopClosing::CorrectLoop；

InsertKeyFrame

将关键帧插入到回环检测关键帧列表mlpLoopKeyFrameQueue中。

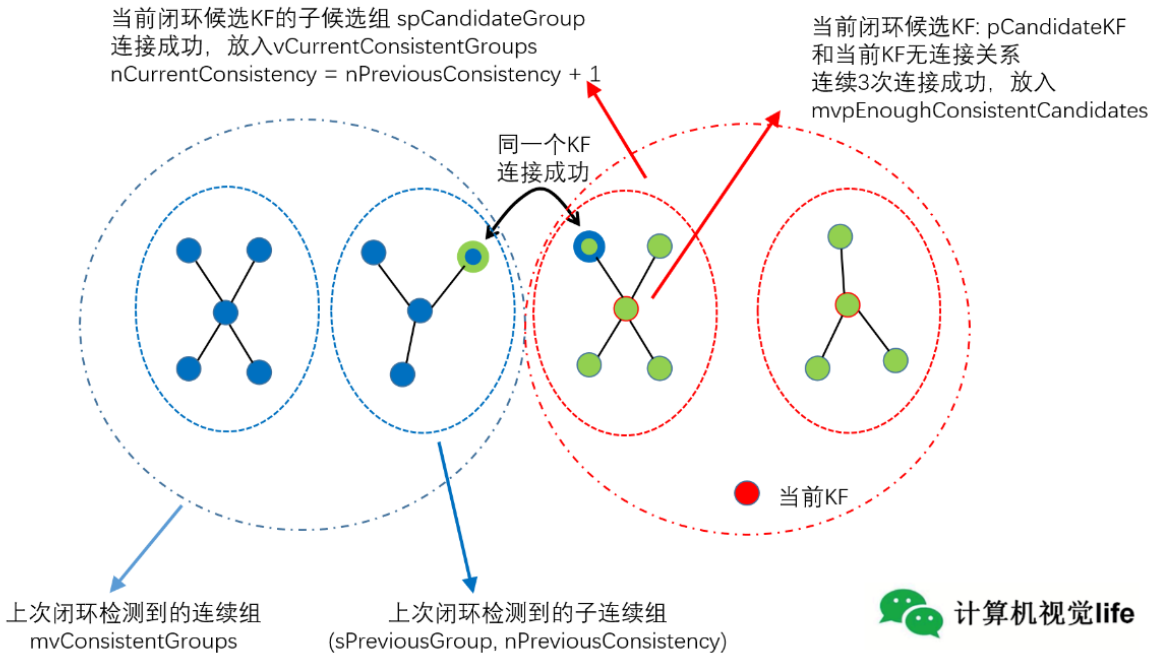
DetectLoop

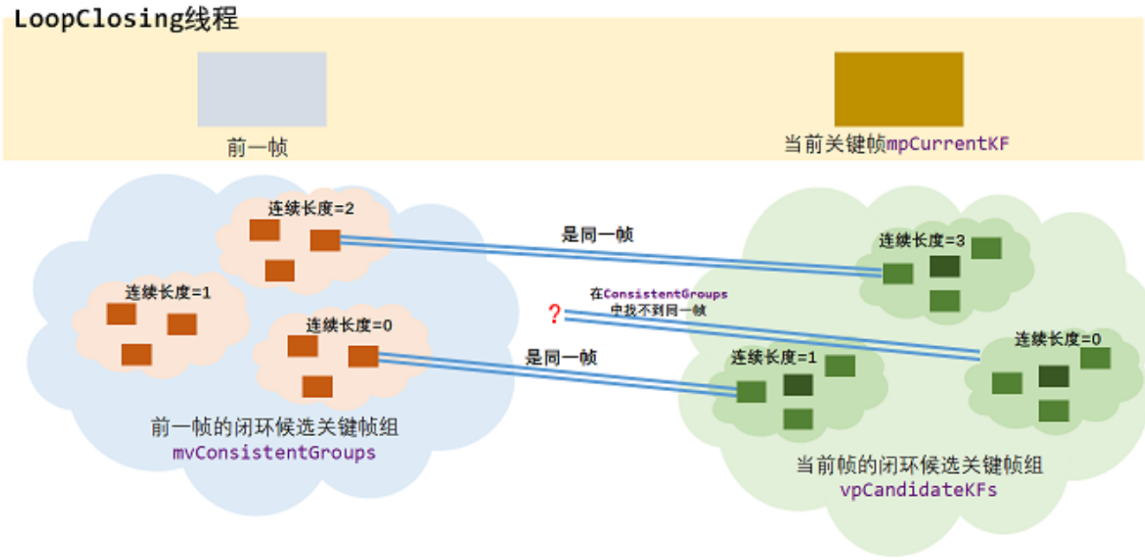
取出关键帧列表mlpLoopKeyFrameQueue中第一个作为当前关键帧，并设置-->KeyFrame::SetNotErase；判断距离上次闭环是否小于10帧，若小于则不进行闭环检测，设置-->KeyFrame::SetErase；遍历当前关键帧所有共视关键帧，计算其与当前关键帧的bow相似度得分，得到最低得分minScore（闭环候选帧需要比minScore高）；在所有关键帧中找出闭环候选帧（注意不和当前关键帧连接）-->KeyFrameDatabase::DetectLoopCandidates；若无闭环候选帧，设置-->KeyFrame::SetErase；在闭环候选帧中**检测具有连续性的候选帧**：

1、每个候选帧将与其共视关键帧组构成一个“子候选组spCandidateGroup”， vpCandidateKFs-->spCandidateGroup
2、遍历“子候选组”，检测组中是否有一个关键帧存在于“连续组mvConsistentGroups”，如果存在则连续程度nCurrentConsistency++，则将该“子候选组”放入“当前连续组vCurrentConsistentGroups”，若果不存在，则连续程度nCurrentConsistency设为0，也将该“子候选组”放入“当前连续组vCurrentConsistentGroups”。得到的vCurrentConsistentGroups将会成为下次进入DetectLoop函数时的“连续组mvConsistentGroups”
3、如果nCurrentConsistency大于等于3，那么该”子候选组“代表的候选帧过关，进入mvpEnoughConsistentCandidates。

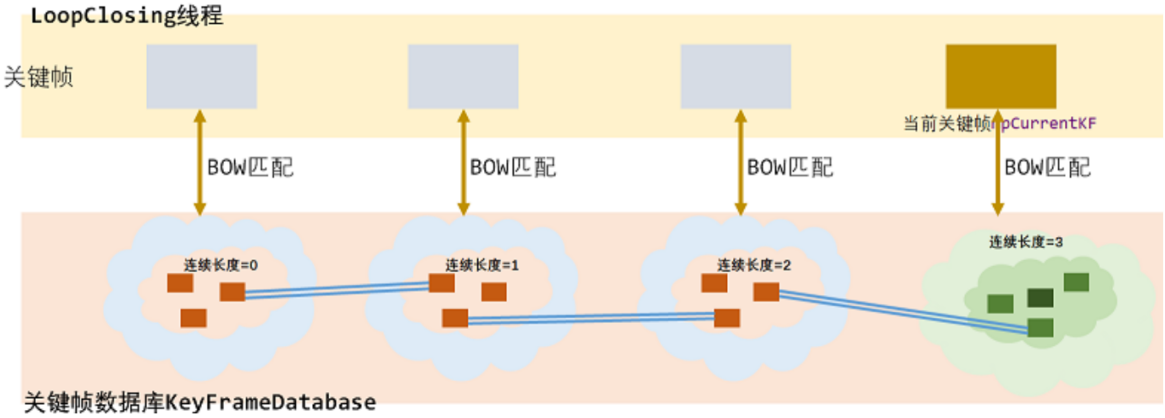
mvpEnoughConsistentCandidates不为空则闭环成功，为空则设置-->KeyFrame::SetErase。

具有连续性的候选帧图解：





若**连续4个**关键帧都能在数据库中找到对应的闭环匹配关键帧组,且这些闭环匹配关键帧组间是连续的,则认为实现闭环:



但是看代码发现，假如"连续组mvConsistentGroups"中每一组都能与"子候选组spCandidateGroup" **A**连续上，那"当前连续组vCurrentConsistentGroups"保存的都会是**A**；然后下一次循环时，"连续组mvConsistentGroups"中存在一组能与"子候选组spCandidateGroup" **B**连续上，因为存在标志位向量vbConsistentGroup，那这个**B**就不会被保存到"当前连续组vCurrentConsistentGroups"中。感觉存在问题 TODO，应该添加连续性程度的判断。

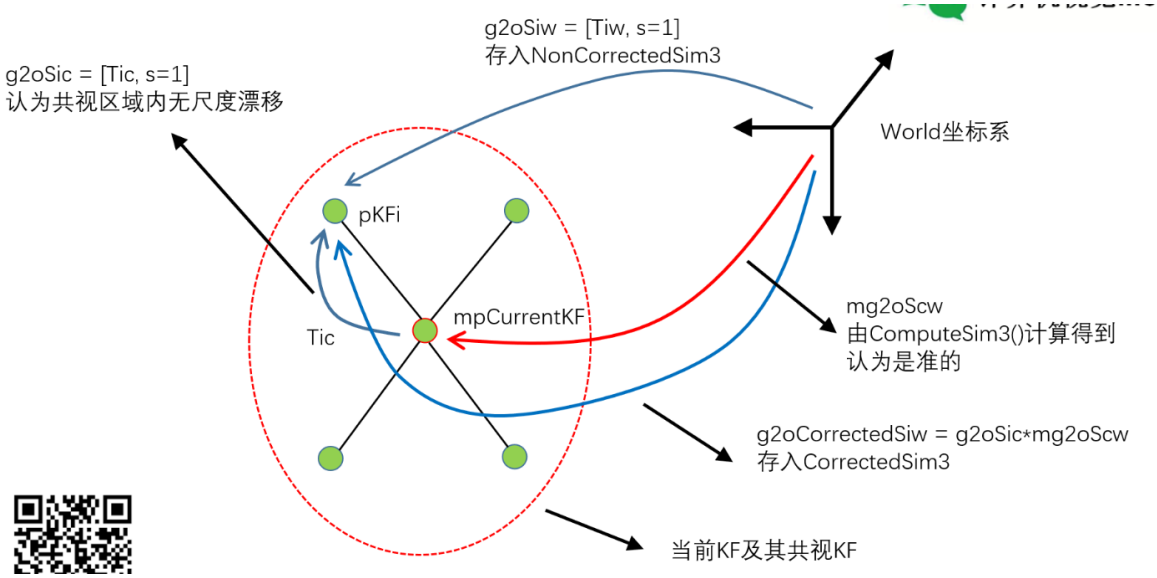
ComputeSim3

Sim3指**使用3对匹配的点计算相似变换**，均摊误差、自然过渡。

遍历闭环候选帧集，设置-->KeyFrame::SetNotErase，初步筛选出与当前关键帧的匹配特征点数-->ORBmatcher::SearchByBoW，特征点数大于20的帧构成候选帧集合，并为每一个候选帧构造一个Sim3Solver-->Sim3Solver.h；Sim3Solver 设置Ransac参数-->Sim3Solver::SetRansacParameters；对每一个候选帧用Sim3Solver 迭代匹配-->Sim3Solver::iterate，得到候选帧到当前帧的Sim3变换，若失败则删除此候选帧，若成功则继续匹配（SearchByBoW 匹配可能会有遗漏）得到更多点并优化-->ORBmatcher::SearchBySim3，优化候选帧到当前帧的Sim3变换矩阵，并得到内点-->Optimizer::OptimizeSim3；如果内点数大于20，候选帧与当前帧形成闭环并记为闭环关键帧mpMatchedKF，通过Sim3矩阵和候选帧位姿得到当前帧的位姿；取出闭环关键帧和其共视关键帧记为闭环相连关键帧组，以及他们的地图点mvpLoopMapPoints，将所有地图点投影到当前关键帧进行投影匹配-->ORBmatcher::SearchByProjection；投影得到的匹配地图点数目，超过40个说明成功闭环，保留此闭环关键帧，其他候选帧-->KeyFrame::SetErase，否则所有候选帧都-->KeyFrame::SetErase。

CorrectLoop

结束当前的局部地图线程、全局BA，为闭环矫正做准备；取出当前关键帧及其共视关键帧记为当前关键帧组；遍历当前关键帧组，计算闭环g2o优化后g2oCorrectedSiw和没有矫正g2oSiw的Sim3变换，用来修正共视关键帧的位姿及其地图点世界坐标g2oCorrectedSwi.map(g2oSiw.map(eigP3Dw)（对位姿和地图点进行修改之后都需要对其进行更新MapPoint::UpdateNormalAndDepth，KeyFrame::UpdateConnections）；



检查当前帧与闭环关键帧的地图点，对冲突的进行替换或缺失的进行添加；将闭环相连关键帧组地图点mvpLoopMapPoints 投影到当前关键帧组中，对地图点进行匹配和融合-->LoopClosing::SearchAndFuse；更新当前关键帧组的两级共视相连关系，得到因闭环时地图点融合而新得到的连接关系LoopConnections；进行本质图优化，优化本质图中所有关键帧的位姿，并通过位姿更新地图点-->Optimizer::OptimizeEssentialGraph；新建一个线程用于全局BA优化，优化所有位姿和地图点-->LoopClosing::RunGlobalBundleAdjustment；释放局部建图线程中的各状态量和关键帧列表。

SearchAndFuse

遍历待矫正的当前关键帧组，将mvpLoopMapPoints投影到此帧，检查地图点冲突-->ORBmatcher::Fuse，替换掉需要替换的地图点-->MapPoint::Replace（不在Fuse 函数中进行地图点融合更新的原因是需要对地图加锁）。

RunGlobalBundleAdjustment

执行全局BA，优化所有的关键帧位姿和地图中地图点-->Optimizer::GlobalBundleAdjustemnt；从第一个关键帧开始矫正关键帧，遍历并更新全局地图中的所有spanning tree中的关键帧，遍历每一个地图点并用更新的关键帧位姿来更新地图点位置（直接参与了GBA则直接更新位置，反之使用其参考关键帧的位姿间接优化）。

MapDrawer.h

MapDrawer

从配置文件中读取Viewer的设置

SetCurrentCameraPose

设置当前帧相机的位姿, 设置这个函数是因为要处理多线程的操作

KeyFrameDatabase.h

KeyFrameDatabase

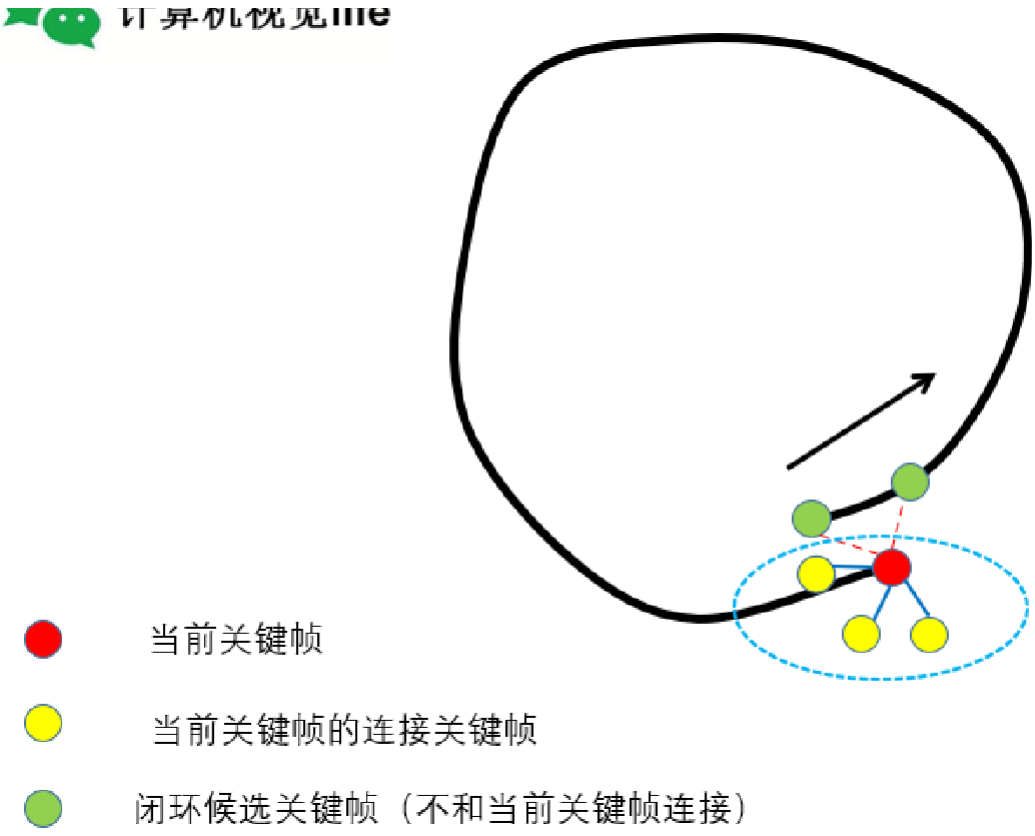
通过词袋模型的字典初始化std::vector<list<KeyFrame*> > mvInvertedFile，mvInvertedFile[i]表示包含了第i个word id的所有关键帧列表，初始化mvInvertedFile的size为词袋模型中词的数量。

DetectRelocalizationCandidates

在重定位中找到与该帧相似的候选关键帧组。利用mvInvertedFile逆索引找出和当前帧具有公共单词的所有关键帧；该帧只与共同单词较多的关键帧进行相似度计算（使用DBow中相似度计算的函数）；取出与关键帧共视度最高的前十个关键帧-->KeyFrame::GetBestCovisibilityKeyFrames，计算累计得分（与当前帧有公共单词的帧才会贡献分数，分数就是公共单词的数量）；只返回累计得分较高的组中分数最高的关键帧。

DetectLoopCandidates

寻找闭环候选帧组的方法与KeyFrameDatabase::DetectRelocalizationCandidates类似，但是多了不和当前关键帧具有共视关系的判断，且闭环候选帧组与当前关键帧bow相似度大于minScore。



Converter.h

实现了 ORB-SLAM2中的一些常用的转换。

toSE3Quat

从变换矩阵中提取旋转矩阵、平移向量，构造g2o::SE3Quat(R,t)类型并返回。

toCvMat

多个同名函数，将不同数据类型的变量转换为cv::Mat。不一一展开

李代数se3转换为变换矩阵：g2o::SE3Quat->cv::Mat
Eigen::Matrix<double,3,1> -> cv::Mat

Sim3Solver.h

Sim3Solver

传入当前关键帧，闭环候选关键帧，ORBmatcher::SearchByBoW得到的匹配地图点vpMatched12（这里的地图点来自于闭环候选关键帧的地图点），尺度标志位。
取出当前关键帧中的所有地图点（通过LocalMapping::CreateNewMapPoints这里得到的地图点）；预分配各种变量的空间；通过vpMatched12得到当前关键帧与闭环候选关键帧相匹配的地图点；分别计算由地图点对应特征点的金字塔尺度得到的卡方分布阈值，地图点在各自相机坐标系下的3D坐标，地图点在各自相机的二维图像坐标；设置默认的RANSAC参数-->Sim3Solver::SetRansacParameters。

SetRansacParameters

TODO

iterate

如果匹配点比要求的最少内点数还少，不满足Sim3 求解条件，返回空；从匹配的地图点点对中随机选择三对，计算P3Dc2i 到 P3Dc1i 的Sim3变换-->Sim3Solver::ComputeSim3，对计算出的Sim3变换通过投影误差进行inlier检测-->Sim3Solver::CheckInliers，记录并更新最多的内点数目及对应的参数。达到了最大迭代次数，内点数目小于Sim3Solver::SetRansacParameters中设置的所需最少内点数，则失败。

ComputeSim3

Sim3计算过程，主要是一些数学公式，TODO

CheckInliers

把2系中的地图点经过Sim3变换到1系中的重投影坐标，与1系地图点直接投影到1系中的像素坐标，计算误差；同理，把1系中的地图点经过Sim3变换到2系中的重投影坐标，与2系地图点直接投影到2系中的像素坐标，计算误差；根据误差来确定其是否为内点。