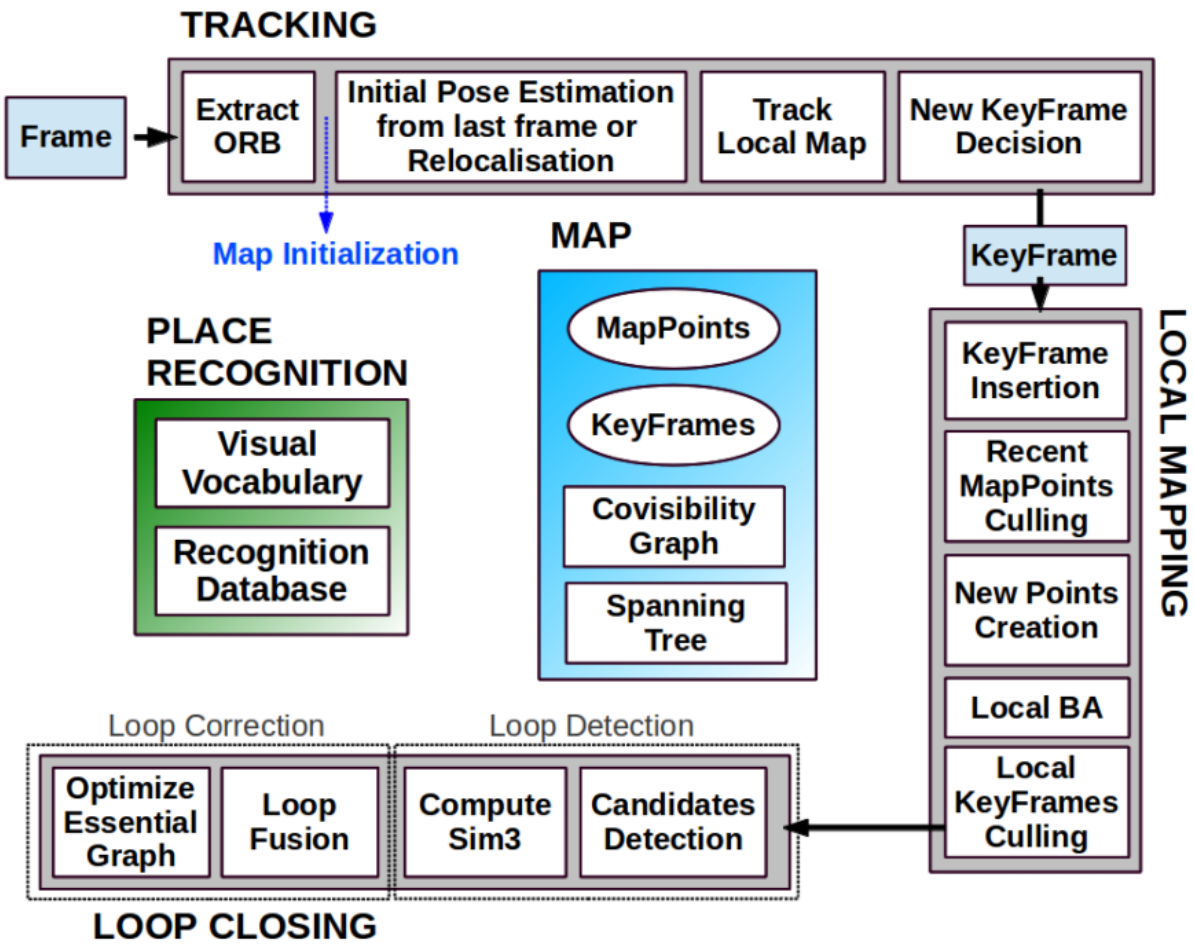


（1）跟踪（Tracking）

这一部分主要工作是从图像中提取 ORB 特征，根据上一帧进行姿态估计，或者进行通过全局重定位初始化位姿，然后跟踪已经重建的局部地图，优化位姿，再根据一些规则确定新的关键帧。

（2）建图（LocalMapping）

这一部分主要完成局部地图构建。包括对关键帧的插入，验证最近生成的地图点并进行筛选，然后生成新的地图点，使用局部捆集调整（Local BA），最后再对插入的关键帧进行筛选，去除多余的关键帧。

（3）闭环检测（LoopClosing）

这一部分主要分为两个过程，分别是闭环探测和闭环校正。闭环检测先使用 WOB 进行探测，然后通过 Sim3 算法计算相似变换。闭环校正，主要是闭环融合和 Essential Graph 的图优化。

一、整体框架

（1）首先使用 LoadImages 读取图片目录和时间戳文件
（2）创建 ORB_SLAM2::System 对象
（3）循环读取数据
　（3.1）读取图片
　（3.2）读取时间戳
　（3.3）将图片传给 SLAM 系统
（4）关闭 SLAM 系统
（5）将相机轨线保存到硬盘中

二、SLAM 系统的创建

```
ORB_SLAM2 System SLAM  //创建ORB_SLAM2系统对象

    mpVocabulary = new ORBVocabulary();  //读取ORB词袋

    mpKeyFrameDatabase = new KeyFrameDatabase(*mpVocabulary);  //创建关键帧数据库

    mpMap = new Map();  //创建地图对象

    //创建两个显示窗口
    mpFrameDrawer = new FrameDrawer(mpMap);
    mpMapDrawer = new MapDrawer(mpMap, strSettingsFile);

    //初始化Tracking对象
    mpTracker = new Tracking(this, mpVocabulary, mpFrameDrawer, mpMapDrawer,  mpMap, mpKeyFrameDatabase, strSettingsFile, mSensor);

    //初始化Local Mapping对象
    mpLocalMapper = new LocalMapping(mpMap, mSensor==MONOCULAR);

    //初始化 Loop Closing对象
    mpLoopCloser = new LoopClosing(mpMap, mpKeyFrameDatabase, mpVocabulary, mSensor!=MONOCULAR);

    // 初始化窗口
    mpViewer = new Viewer(this, mpFrameDrawer,mpMapDrawer,mpTracker,strSettingsFile);
```

（1）创建了 ORB 词袋的对象
（2）创建了关键帧的数据库
（3）创建地图对象
（4）创建两个显示窗口
（5）初始化 Tracking 对象
（6）初始化 Local Mapping 对象，并开启线程运行
（7）初始化 Loop Closing 对象，并开启线程运行
（8）初始化窗口，开启线程显示图像和地图点

三、Tracking 的运行

```
for(int ni=0; ni<nImages; ni++) //循环读取数据

    im = cv::imread(vstrImageFilenames[ni],CV_LOAD_IMAGE_UNCHANGED);  //读取图片

    double tframe = vTimestamps[ni];  //读取时间戳

    SLAM.TrackMonocular(im,tframe);  //将图片传给SLAM系统

        //先完成模式变换的检验，再做一个重置的检验

        return mpTracker->GrabImageMonocular(im,timestamp);  //执行Tracking类的函数

            cvtColor(mImGray,mImGray,CV_RGB2GRAY);  //转灰度图

            //如果没有初始化就没有关键帧

                mCurrentFrame = Frame(mImGray,timestamp,mpIniORBextractor,mpORBVocabulary,mK,mDistCoef,mbf,mThDepth);

            //否则

                mCurrentFrame = Frame(mImGray,timestamp,mpORBextractorLeft,mpORBVocabulary,mK,mDistCoef,mbf,mThDepth);

            Track(); //跟踪
```

自动地图初始化:

系统的第一步是初始化，ORB_SLAM 使用的是一种自动初始化方法。这里同时计算两个模型：用于平面场景的单应性矩阵 H 和用于非平面场景的基础矩阵 F，然后通过一个评分规则来选择合适的模型，恢复相机的旋转矩阵 R 和平移向量 t。

一、找到初始对应点

二、同时计算两个模型

三、模型选择

四、运动恢复（sfm）

（1）从单应性变换矩阵 H 中恢复

(2)从基础矩阵 F 中恢复

五、集束调整

最后使用一个全局集束调整（BA)，优化初始化结果。

跟踪:

这一部分是 ORB_SLAM 系统中最基本的一步，会对每一帧图像进行跟踪计算。Tracking 线程运行在主线程中，主要思路是在当前帧和（局部）地图之间寻找尽可能多的对应关系，来优化当前帧的位姿。

一、ORB 提取

本文做匹配的过程中，是用的都是 ORB 特征描述子。先在 8 层图像金字塔中，提取 FAST 特征点。提取特征点的个数根据图像分辨率不同而不同，高分辨率的图像提取更多的角点。然后对检测到的特征点用 ORB 来描述，用于之后的匹配和识别。跟踪这部分主要用了几种模型：运动模型（Tracking with motion model）、关键帧（Tracking with reference keyframe）和重定位（Relocalization）。

二、从前一帧初始化位姿估计

在成功与前面帧跟踪上之后，为了提高速率，本文使用与之前速率相同的运动模式来预测相机姿态，并搜索上一帧观测到的地图点。这个模型是假设物体处于匀速运动。

三、通过全局重定位来初始化位姿估计

假如使用上面的方法，当前帧与最近邻关键帧的匹配也失败了，那么意味着需要重新定位才能继续跟踪。其次，寻找有足够多的特征点匹配的关键帧；最后，利用 RANSAC 迭代，然后使用 PnP 算法求解位姿。

四、局部地图跟踪

通过之前的计算，已经得到一个对位姿的初始估计，我们就能透过投影，从已经生成的地图点中找到更多的对应关系，来精确结果。为了降低复杂度，这里只是在局部图中做投影。局部地图中与当前帧有相同点的关键帧序列成为 K1，在 covisibility graph 中与 K1 相邻的称为 K2。局部地图有一个参考关键帧 $K_{ref} \in K1$，它与当前帧具有最多共同看到的地图云点。针对 K1, K2 可见的每个地图云点，通过如下步骤，在当前帧中进行搜索:

（1）将地图点投影到当前帧上，如果超出图像范围，就将其舍弃；

（2）计算当前视线方向向量 v 与地图点云平均视线方向向量 n 的夹角，舍弃 $n \cdot v <$

cos(60°)的点云；
（3）计算地图点到相机中心的距离 d，认为[dmin, dmax]是尺度不变的区域，若 d 不在这个区域，就将其舍弃；
（4）计算图像的尺度因子，为 d/dmin；
（5）将地图点的特征描述子 D 与还未匹配上的 ORB 特征进行比较，根据前面的尺度因子，找到最佳匹配。
五、关键帧的判断标准
　　最后一步是确定是否将当前帧定为关键帧，由于在 Local Mapping 中，会剔除冗余关键帧，所以我们要尽快插入新的关键帧，这样才能更鲁棒。
确定关键帧的标准如下：
（1）在上一个全局重定位后，又过了 20 帧；
（2）局部建图闲置，或在上一个关键帧插入后，又过了 20 帧；
(3)当前帧跟踪到大于 50 个点；
（4）当前帧跟踪到的比参考关键帧少 90%。
六、代码架构

```
Track(); //跟踪
    // 如果没有初始化，就进行自动初始化
    MonocularInitialization();
    //跟踪
        //如果当前运动速度为空，使用关键帧模式
        bOK = TrackReferenceKeyFrame();
        //若有运动速度，使用运动模式
        bOK = TrackWithMotionModel();
            //若运动模式跟踪到的特征点数量较少，使用关键帧模式
            bOK = TrackReferenceKeyFrame();
        //否则使用全局重定位
        bOK = Relocalization();
        //跟踪局部地图
        bOK = TrackLocalMap();
        //检测插入新的关键帧
        if(NeedNewKeyFrame())
            CreateNewKeyFrame();
```

局部建图

```
mptLocalMapping = new thread(&ORB_SLAM2::LocalMapping::Run,mpLocalMapper);

if(CheckNewKeyFrames()) //检测是否有关键帧在序列中

    ProcessNewKeyFrame(); //处理新的关键帧，计算其词袋描述子，并且插入地图中

    // 检查当前的地图点，剔除冗余点
    MapPointCulling();

    // 三角化计算新的地图点
    CreateNewMapPoints();

    if(!CheckNewKeyFrames())

        // 通过邻近关键帧优化地图点
        SearchInNeighbors();

    if(!CheckNewKeyFrames() && !stopRequested())

        // Local BA
        if(mpMap->KeyFramesInMap()>2)
        Optimizer::LocalBundleAdjustment(mpCurrentKeyFrame,&mbAbortBA, mpMap);

        // 检查关键帧，剔除冗余关键帧
        KeyFrameCulling();

    //插入关键帧
    mpLoopCloser->InsertKeyFrame(mpCurrentKeyFrame);
```

一、关键帧插入

首先将新的关键帧 Ki 作为新的节点 Ki 加入 Covibility Graph，并且更新与那些能够共享地图点的关键帧节点相连接的边。同时更新关键帧 Ki 的生长树，并计算表示关键帧的词袋 BOW。

二、当前地图点剔除

为了保存地图点，必须在创建该点云的前三帧测试通过约束，才能真正被保存，这样才能保证可跟踪且不容易在三角化时出现较大误差。

一个点要被加入 Map，需要满足下面条件：

（1）这个点要在可预测到能够观察到该点的关键帧中，有超过 25%的关键帧能够跟踪到这个点；

（2）如果一个地图点被构建，它必须被超过三个关键帧观察到（在代码中，可以发现如果是单摄像头，这个阈值被设置为 2）。

一旦地图点被创建了，就只有在少于 3 个关键帧能够观察到该点时才会被剔除。而要剔除关键帧，通常是在局部集束调整剔除外点或者在后面剔除关键帧时才会发生。这样就保证了地图点很少存在外点影响效果。

三、新的地图点创建

通过将检测到的 ORB 特征点，找到 Covisibility Graph 中与之相连的关键帧 Kc，进行特征匹配，然后将匹配到的特征点进行三角化。对于没有匹配上的点，本文又与其他关键帧中未被匹配的特征点进行匹配。匹配方法使用的是之前的方法，并且将不满足对极几何约束的匹配点舍弃。ORB 特征点对三角化后，检查正向景深、视差、反投影误差和尺度一致性，这时才得到地图点。一个地图点是通过两个关键帧观察到的，而它也可以投影到与之相连的其他关键帧中，这个时候可以使用 Tracking 部分的跟踪局部地图来在附近的关键帧中找到匹配。得到更多的地图点。

四、局部集束调整

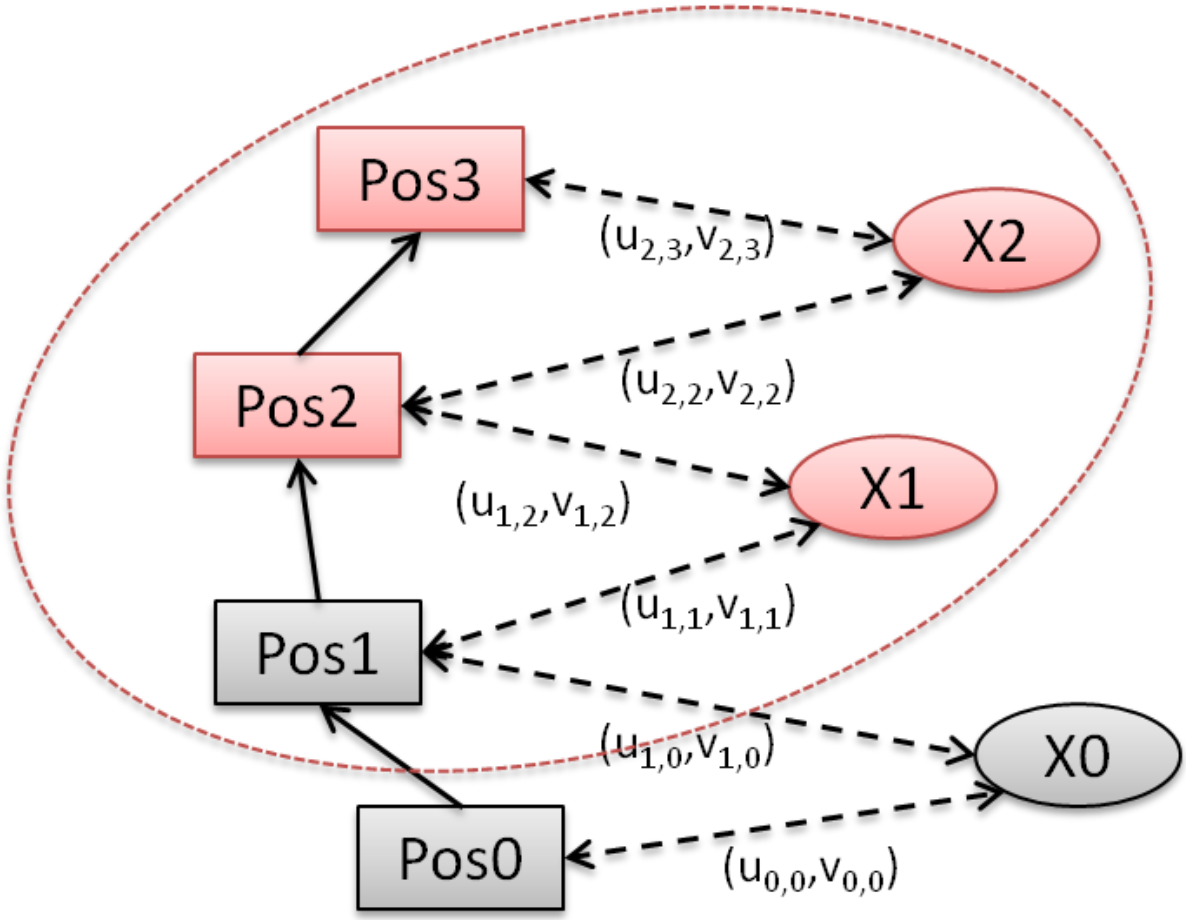
局部集束调整（local BA）会将当前处理的关键帧 Ki 进行优化，优化时如下图所示：现在优化 Pos3 位置的关键帧。同时参与优化的还有：

所有在 Covibility Graph 中与该关键帧相连的关键帧 Kc，即下图中的 Pos2；

所以被这些关键帧观察到的地图点，即 X1 和 X2。

另外还有能观察到地图点的但并未与当前处理的关键帧相连的关键帧，即下图中的 Pos1。

但要注意的是，诸如 Pos1 的关键帧，参与优化中的约束，但不作为变量去改变它们的值。优化时得到的外点会在优化的中期或后期被剔除。



五、局部关键帧剔除

为了控制重建的紧凑度，LocalMapping 会去检测冗余的关键帧，然后删除它们。这样的话会有利于控制，随着关键帧数目增长后，集束调整的复杂度。因为除非视角改变了，否则关键帧的数量在相同的环境中不应该无休止地增长。本文将那些有 90%的点能够被超过三个关键帧观察到的关键帧认为是冗余关键帧，并将其删除。

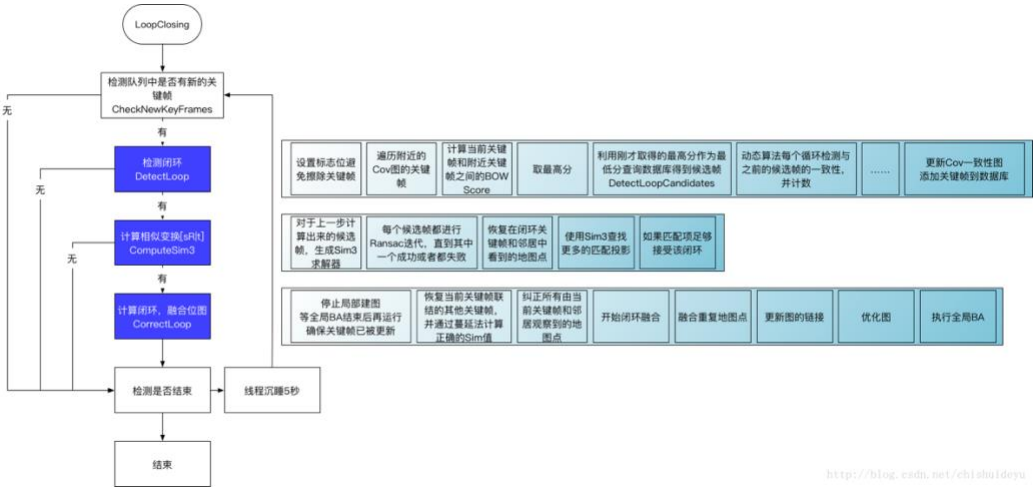
闭环检测:

```
//初始化 Loop Closing对象
mpLoopCloser = new LoopClosing(mpMap, mpKeyFrameDatabase, mpVocabulary, mSensor!=MONOCULAR);

    // 创建线程，并运行
    mptLoopClosing = new thread(&ORB_SLAM2::LoopClosing::Run, mpLoopCloser);

        if(CheckNewKeyFrames()) // 检查队列中存在关键帧

            if(DetectLoop()) //检查满足闭环条件

                if(ComputeSim3()) 计算相似变换

                    // 校正闭环，进行闭环融合和图优化
                    CorrectLoop();
```

一、闭环条件检测

首先我们计算关键帧 Ki 和在 Covisibility Graph 中与其相连的关键帧之间的词袋(BOW)之间的相似度。本文中，作者离线训练了大量的基于 ORB 描述的词袋，在程序运行时加载进去。这里的词袋作为对该关键帧的描述，将闭环检测转变为一个类似于模式识别的问题。当相机再次来到之前到过的场景时，就会因为看到相同的景物，而得到类似的词袋描述，从而检测到闭环。

ORBSlam2 中的 LoopClosing 闭环检测线程主要进行闭环检测，并在检测到闭环的时候计算 Sim3 变换，进行后端优化。



检测闭环 DetectLoop
首先我们会检测当前关键帧在 Covisibility 图中的附近关键帧，并会依次计算当前关键帧和每一个附近关键帧的 BoW 分值，通过我们所得到分数的最低分，到数据库中查询，查找出所有大于该最低分的关键帧作为候选帧，用以检测闭环。
1. 如果地图中的关键帧数小于 10，那么不进行闭环检测
2. 获取共视关键帧，并计算他们和当前关键帧之间的 BoW 分数，求得最低分
3. 通过上一步计算出的最低分数到数据库中查找出候选关键帧，这一步相当于是找到了曾经到过此处的关键帧们
4.在地图中搜索与当前关键帧共享一个 BOW word 的关键帧，并排除上一步搜集到的附近关键帧，得到候选帧，这些候选帧基本上都是曾经来到此处看到的。

计算 Sim3:
该函数主要工作就是在当前关键帧和闭环帧之间找到更多的对应点，并通过这些对应点计算当前关键帧和闭环帧之间的 Sim3 变换，求解出 Rt 和 s。
1.对每一个闭环帧，通过 BoW 的 matcher 方法进行第一次匹配，匹配闭环帧和当前关键帧之间的匹配关系，如果对应关系少于 20 个，则丢弃，否则构造一个 Sim3 求解器并保存起来。
2.对上一步得到的每一个满足条件的闭环帧，通过 RANSAC 迭代，求解 Sim3。
3.通过返回的 Sim3 进行第二次匹配。
4.使用非线性最小二乘法优化 Sim3.
5.恢复闭环关键帧和其邻居关键帧的 MapPoint 地图点，使用投影得到更多的匹配点，如果匹配点数量充足，则接受该闭环。

纠正闭环后端优化:
在上一步求得了 Sim3 和对应点之后，就纠正了当前帧的位姿，但是误差不仅仅在当前帧，此前的每一帧都有累计误差需要消除，函数 CorrectLoop 就是用来消除这个累计误差，进行整体的调节。
1.如果有全局 BA 运算在运行的话，终止之前的 BA 运算。
2.使用传播法计算每一个关键帧正确的 Sim3 变换值
3.使用反向投影的方法，将当前关键帧和邻居观察到的地图点得到三维场景下的位姿，并更新关键帧的位姿
4.将当前关键帧的地图点进行融合，其实融合就是判断如果是同一个点的话，那么将当前的地图点强制换成原本的地图点。
5.使用已纠正的位姿，将在循环关键帧附近观察到的地图点投影到当前的关键帧和邻居，融合重复点。

优化图:
1.使用非线性最小二乘图优化的方法来优化 EssentialGraph。
2.全局 BA 优化