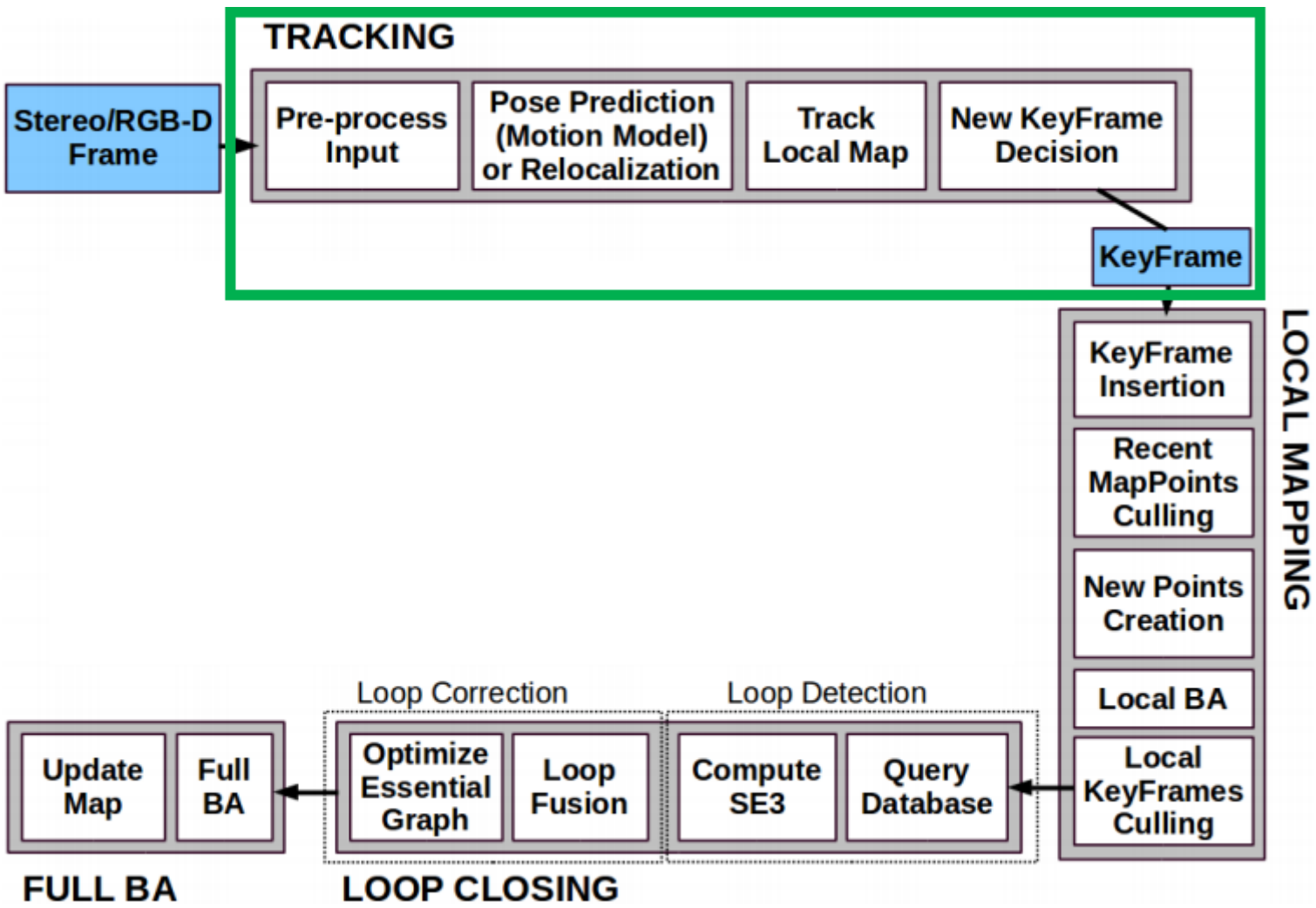
各成员函数/变量

- 跟踪状态
- 初始化
  - 单目相机初始化: `MonocularInitialization()`
  - 双目/RGBD相机初始化: `StereoInitialization()`
- 初始位姿估计
  - 根据恒速运动模型估计初始位姿: `TrackWithMotionModel()`
  - 根据参考帧估计位姿: `TrackReferenceKeyFrame()`
  - 通过重定位估计位姿: `Relocalization()`
- 跟踪局部地图: `TrackLocalMap()`
- 关键帧的创建
  - 判断是否需要创建新关键帧: `NeedNewKeyFrame()`
  - 创建新关键帧: `CreateNewKeyFrame()`
- 跟踪函数: `Track()`

`Tracking`流程中的关键问题(暗线)

- 地图点的创建与删除
- 关键帧与地图点间发生关系的时机
- 参考关键帧: `mpReferenceKF`



各成员函数/变量

跟踪状态

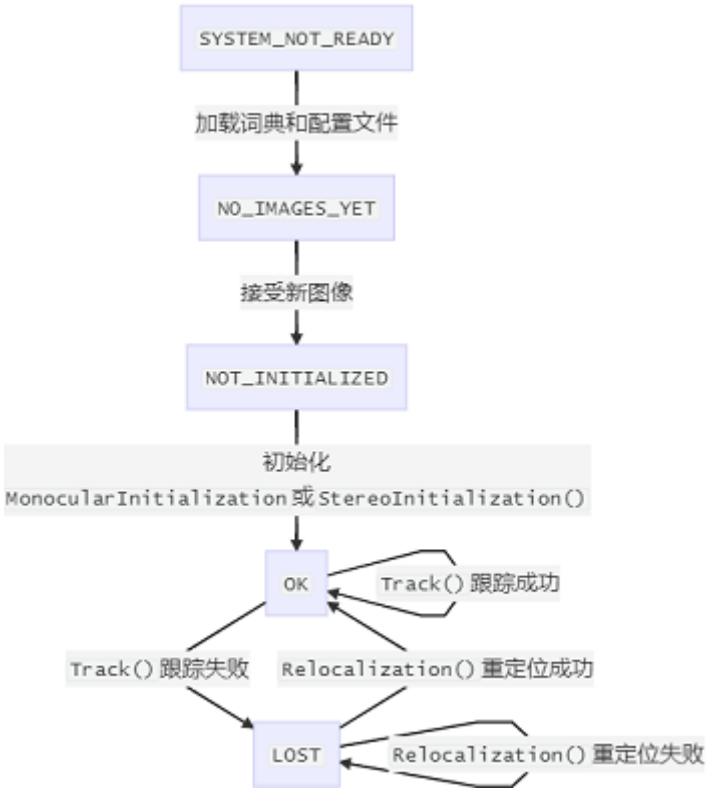
`Tracking`类中定义枚举类型`eTrackingState`,用于表示跟踪状态,其可能的取值如下

值	意义
`SYSTEM_NOT_READY`	系统没有准备好,一般就是在启动后加载配置文件和词典文件时候的状态
`NO_IMAGES_YET`	还没有接收到输入图像
`NOT_INITIALIZED`	接收到图像但未初始化成功
`OK`	跟踪成功
`LOST`	跟踪失败

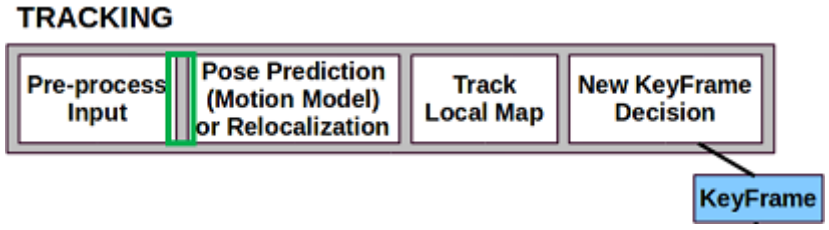
`Tracking`类的成员变量`mState`和`mLastProcessedState`分别表示**当前帧的跟踪状态**和**上一帧的跟踪状态**.

成员变量	访问控制	意义
`eTrackingState mState`	`public`	当前帧`mCurrentFrame`的跟踪状态
`eTrackingState mLastProcessedState`	`public`	前一帧`mLastFrame`的跟踪状态

跟踪状态转移图如下:



初始化

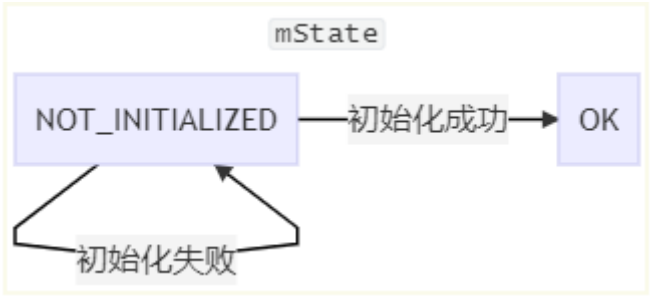


成员函数/变量	访问控制	意义
`Frame mCurrentFrame`	`public`	当前帧
`KeyFrame* mpReferenceKF`	`protected`	参考关键帧 初始化成功的帧会被设为参考关键帧
`std::vector<KeyFrame*> mvpLocalKeyFrames`	`protected`	局部关键帧列表,初始化成功后向其中添加局部关键帧
`std::vector<MapPoint*> mvpLocalMapPoints`	`protected`	局部地图点列表,初始化成功后向其中添加局部地图点

初始化用于SLAM系统刚开始接收到图像的几帧,初始化成功之后就进入正常的跟踪操作.

`Tracking`类主函数`Tracking::Track()`检查到当前系统的跟踪状态`mState`为`NOT_INITIALIZED`时,就会进行初始化.

```
1  void Tracking::Track() {
2
3      // ...
4
5      unique_lock<mutex> lock(mpMap->mMutexMapUpdate);
6
7      // step1. 若还没初始化,则尝试初始化
8      if (mState == NOT_INITIALIZED) {
9          if (mSensor == System::STEREO || mSensor == System::RGBD)
10             StereoInitialization();
11         else
12             MonocularInitialization();
13         if (mState != OK)
14             return;
15     }
16
17     // ...
18 }
```



单目相机初始化: `MonocularInitialization()`

成员函数/变量	访问控制	意义
`void MonocularInitialization()`	`protected`	单目相机初始化
`void CreateInitialMapMonocular()`	`protected`	单目初始化成功后建立初始局部地图
`Initializer* mpInitializer`	`protected`	单目初始化器
`Frame mInitialFrame`	`public`	单目初始化参考帧(实际上就是前一帧)
`std::vector<cv::Point3f> mvIniP3D`	`public`	单目初始化中三角化得到的地图点坐标
`std::vector<cv::Point2f> mvbPrevMatched`	`public`	单目初始化参考帧地图点
`std::vector<int> mvIniMatches`	`public`	单目初始化中参考帧与当前帧的匹配关系

单目相机初始化条件: 连续两帧间成功三角化超过`100`个点,则初始化成功.

```
void Tracking::MonocularInitialization() {
    // step1. 若单目初始化器还没创建,则创建初始化器
    if (!mpInitializer) {
        if (mCurrentFrame.mvKeys.size() > 100) {
            mInitialFrame = Frame(mCurrentFrame);
            mLastFrame = Frame(mCurrentFrame);
            mvbPrevMatched.resize(mCurrentFrame.mvKeysUn.size());
            for (size_t i = 0; i < mCurrentFrame.mvKeysUn.size(); i++)
                mvbPrevMatched[i] = mCurrentFrame.mvKeysUn[i].pt;
            mpInitializer = new Initializer(mCurrentFrame, 1.0, 200);
            fill(mvIniMatches.begin(), mvIniMatches.end(), -1);
            return;
        }
    } else  {
        // step2. 若上一帧特征点足够,但当前帧特征点太少,则匹配失败,删除初始化器
        if ((int) mCurrentFrame.mvKeys.size() <= 100) {
            delete mpInitializer;
            mpInitializer = static_cast<Initializer *>(NULL);
            fill(mvIniMatches.begin(), mvIniMatches.end(), -1);
            return;
        }

        // step3. 在mInitialFrame和mLastFrame间进行匹配搜索
        ORBmatcher matcher(0.9, true);
        int nmatches = matcher.SearchForInitialization(mInitialFrame, mCurrentFrame, mvbPrevMatched,
mvIniMatches, 100);

        // step4. 匹配的特征点数目太少,则匹配失败,删除初始化器
        if (nmatches < 100) {
            delete mpInitializer;
            mpInitializer = static_cast<Initializer *>(NULL);
            return;
        }

        // step5. 进行单目初始化
        cv::Mat Rcw;
        cv::Mat tcw;
        vector<bool> vbTriangulated;
        if (mpInitializer->Initialize(mCurrentFrame, mvIniMatches, Rcw, tcw, mvIniP3D, vbTriangulated)) {
            // step6. 单目初始化成功后,删除未成功三角化的匹配点对
            for (size_t i = 0, iend = mvIniMatches.size(); i < iend; i++) {
                if (mvIniMatches[i] >= 0 && !vbTriangulated[i]) {
                    mvIniMatches[i] = -1;
                    nmatches--;
                }
            }

            // step7. 创建初始化地图,以mInitialFrame为参考系
            cv::Mat Tcw = cv::Mat::eye(4, 4, CV_32F);
            Rcw.copyTo(Tcw.rowRange(0, 3).colRange(0, 3));
            tcw.copyTo(Tcw.rowRange(0, 3).col(3));
            mInitialFrame.SetPose(cv::Mat::eye(4, 4, CV_32F));
            mCurrentFrame.SetPose(Tcw);
            CreateInitialMapMonocular();
        }
    }
}
```

单目初始化成功后调用函数`CreateInitialMapMonocular()`创建初始化地图

```
void Tracking::CreateInitialMapMonocular() {
    // mInitialFrame 和 mCurrentFrame 是最早的两个关键帧
    KeyFrame *pKFini = new KeyFrame(mInitialFrame, mpMap, mpKeyFrameDB);
    KeyFrame *pKFcur = new KeyFrame(mCurrentFrame, mpMap, mpKeyFrameDB);

    // step1. 计算两个关键帧的词袋
```

```
    pKFini->ComputeBoW();
    pKFcur->ComputeBoW();

    // step2. 将两个关键帧插入地图
    mpMap->AddKeyFrame(pKFini);
    mpMap->AddKeyFrame(pKFcur);

    // step3. 处理所有地图点
    for (size_t i = 0; i < mvIniMatches.size(); i++) {
        // 创建并添加地图点
        MapPoint *pMP = new MapPoint(mvIniP3D[i], pKFcur, mpMap);
        mpMap->AddMapPoint(pMP);
        // 添加关键帧到地图点的观测
        pKFini->AddMapPoint(pMP, i);
        pKFcur->AddMapPoint(pMP, mvIniMatches[i]);
        // 添加地图点到关键帧的观测
        pMP->AddObservation(pKFini, i);
        pMP->AddObservation(pKFcur, mvIniMatches[i]);
        // 计算地图点描述子并更新平均观测数据
        pMP->ComputeDistinctiveDescriptors();
        pMP->UpdateNormalAndDepth();
    }

    // 基于观测到的地图点,更新关键帧共视图
    pKFini->UpdateConnections();
    pKFcur->UpdateConnections();

    // step4. 全局BA: 优化所有关键帧位姿和地图点
    Optimizer::GlobalBundleAdjustemnt(mpMap, 20);

    // step5. 将两帧间的平移尺度归一化(以场景的中值深度为参考)
    float medianDepth = pKFini->ComputeSceneMedianDepth(2);
    float invMedianDepth = 1.0f / medianDepth;
    cv::Mat Tc2w = pKFcur->GetPose();
    Tc2w.col(3).rowRange(0, 3) = Tc2w.col(3).rowRange(0, 3) * invMedianDepth;
    pKFcur->SetPose(Tc2w);

    // step6. 将坐标点尺度也归一化
    vector<MapPoint *> vpAllMapPoints = pKFini->GetMapPointMatches();
    for (size_t iMP = 0; iMP < vpAllMapPoints.size(); iMP++) {
        if (vpAllMapPoints[iMP]) {
            MapPoint *pMP = vpAllMapPoints[iMP];
            pMP->SetWorldPos(pMP->GetWorldPos() * invMedianDepth);
        }
    }

    // step7. 将关键帧插入局部地图,更新归一化后的位姿和地图点坐标
    mpLocalMapper->InsertKeyFrame(pKFini);
    mpLocalMapper->InsertKeyFrame(pKFcur);
    mCurrentFrame.SetPose(pKFcur->GetPose());
    mnLastKeyFrameId = mCurrentFrame.mnId;
    mpLastKeyFrame = pKFcur;
    mvpLocalKeyFrames.push_back(pKFcur);
    mvpLocalKeyFrames.push_back(pKFini);
    mvpLocalMapPoints = mpMap->GetAllMapPoints();
    mpReferenceKF = pKFcur;
    mCurrentFrame.mpReferenceKF = pKFcur;
    mLastFrame = Frame(mCurrentFrame);
    mpMap->SetReferenceMapPoints(mvpLocalMapPoints);
    mpMapDrawer->SetCurrentCameraPose(pKFcur->GetPose());
    mpMap->mvpKeyFrameOrigins.push_back(pKFini);

    // step8. 更新跟踪状态变量mState
    mState = OK;
}
```

双目/RGBD相机初始化: `StereoInitialization()`

成员函数/变量	访问控制	意义
`void StereoInitialization()`	`protected`	双目/RGBD相机初始化

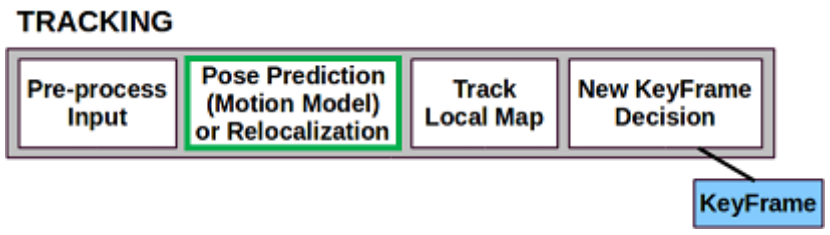
双目/RGBD相机的要求就宽松多了,只要左目图像能找到多于`500`个特征点,就算是初始化成功.

函数`StereoInitialization()`内部既完成了初始化,又构建了初始化局部地图.

```
void Tracking::StereoInitialization() {
    if (mCurrentFrame.N > 500) {

        // 基于当前帧构造初始关键帧
        mCurrentFrame.SetPose(cv::Mat::eye(4, 4, CV_32F));
        KeyFrame *pKFini = new KeyFrame(mCurrentFrame, mpMap, mpKeyFrameDB);
        mpMap->AddKeyFrame(pKFini);
```

```
        mpLocalMapper->InsertKeyFrame(pKFini);
        // 构造地图点
        for (int i = 0; i < mCurrentFrame.N; i++) {
            float z = mCurrentFrame.mvDepth[i];
            if (z > 0) {
                cv::Mat x3D = mCurrentFrame.UnprojectStereo(i);
                MapPoint *pNewMP = new MapPoint(x3D, pKFini, mpMap);
                pNewMP->AddObservation(pKFini, i);
                pNewMP->ComputeDistinctiveDescriptors();
                pNewMP->UpdateNormalAndDepth();
                mpMap->AddMapPoint(pNewMP);
                pKFini->AddMapPoint(pNewMP, i);
                mCurrentFrame.mvpMapPoints[i] = pNewMP;
            }
        }

        // 构造局部地图
        mvpLocalKeyFrames.push_back(pKFini);
        mvpLocalMapPoints = mpMap->GetAllMapPoints();
        mpReferenceKF = pKFini;
        mCurrentFrame.mpReferenceKF = pKFini;

        // 更新跟踪状态变量mState
        mState = OK;
    }
}
```

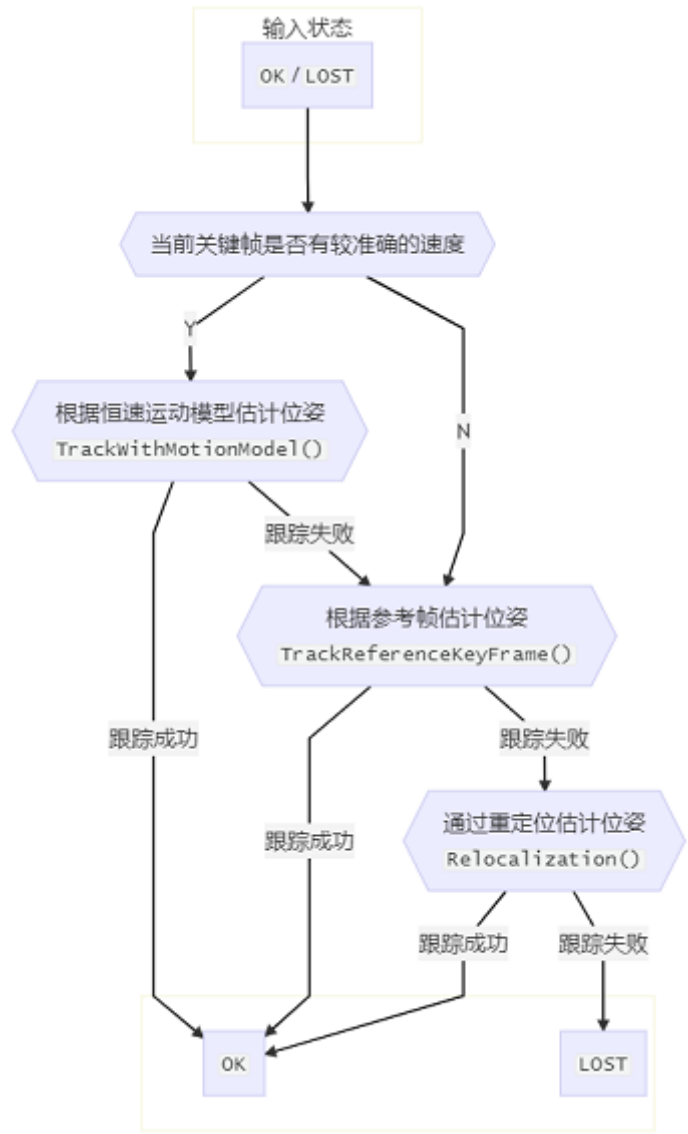
初始位姿估计



当`Tracking`线程接收到一帧图像后,会先估计其初始位姿,再根据估计出的初始位姿跟踪局部地图并进一步优化位姿.

初始位姿估计有三种手段:

- 根据恒速运动模型估计位姿`TrackWithMotionModel()`
- 根据参考帧估计位姿`TrackReferenceKeyFrame()`
- 通过重定位估计位姿`Relocalization()`




```
void Tracking::Track() {

    // ...

    unique_lock<mutex> lock(mpMap->mMutexMapUpdate);

    // step1. 若还没初始化,则尝试初始化
    if (mState == NOT_INITIALIZED) {
        // 初始化
```

```
    } else {
        // step2. 若系统已初始化,就进行跟踪(或重定位)
        bool bOK;

        // step2.1. 符合条件时,优先根据运动模型跟踪,如运动模型跟踪失败,就根据参考帧进行跟踪
        if (mState == OK) {
            if (mVelocity.empty() || mCurrentFrame.mnId < mnLastRelocFrameId + 2) {     // 判断当前关键帧是否具有较稳定的速度
                bOK = TrackReferenceKeyFrame();
            } else {
                bOK = TrackWithMotionModel();
                if (!bOK)
                    bOK = TrackReferenceKeyFrame();
            }
        } else {
            // step2.2. 若上一帧没跟踪丢失,则这一帧重定位
            bOK = Relocalization();
        }

        // ...
        if (bOK)
            mState = OK;
        else
            mState = LOST;
    }

    // ...
}
```

根据恒速运动模型估计初始位姿: `TrackWithMotionModel()`

恒速运动模型假定连续几帧间的运动速度是恒定的;基于此假设,根据运动速度 `mVelocity` 和上一帧的位姿 `mLastFrame.mTcw` 计算出本帧位姿的估计值,再进行位姿优化.

成员变量 `mVelocity` 保存前一帧的速度,主函数 `Tracking::Track()` 中调用完函数 `Tracking::TrackLocalMap()` 更新局部地图和当前帧位姿后,就计算速度并赋值给 `mVelocity`.

成员函数/变量	访问控制	意义
`TrackWithMotionModel()`	`protected`	根据恒速运动模型估计初始位姿
`Frame mLastFrame`	`protected`	前一帧, `TrackWithMotionModel()` 与该帧匹配搜索关键点
`cv::Mat mVelocity`	`protected`	相机前一帧运动速度,跟踪完局部地图后更新该成员变量
`list<MapPoint*> mlpTemporalPoints`	`protected`	双目/RGBD相机输入时,为前一帧生成的临时地图点 跟踪成功后该容器会被清空,其中的地图点会被删除

```
bool Tracking::TrackWithMotionModel() {
    ORBmatcher matcher(0.9, true);

    // step1. 更新上一帧的位姿,对于双目/RGBD相机,还会生成临时地图点
    UpdateLastFrame();
    // step2. 根据运动模型设置初始位姿估计值
    mCurrentFrame.SetPose(mVelocity * mLastFrame.mTcw);
    fill(mCurrentFrame.mvpMapPoints.begin(), mCurrentFrame.mvpMapPoints.end(), static_cast<MapPoint *>(NULL));

    // step3. 根据初始位姿估计值进行投影匹配
    int th;
    if (mSensor != System::STEREO)
        th = 15;//单目
    else
        th = 7;//双目
    // step3.1. 寻找匹配特征点
    int nmatches = matcher.SearchByProjection(mCurrentFrame, mLastFrame, th, mSensor == System::MONOCULAR);
    // step3.2. 匹配特征点数目太少就放宽条件重新搜索匹配
    if (nmatches < 20) {
        fill(mCurrentFrame.mvpMapPoints.begin(), mCurrentFrame.mvpMapPoints.end(), static_cast<MapPoint *>(NULL));
        nmatches = matcher.SearchByProjection(mCurrentFrame, mLastFrame, 2 * th, mSensor == System::MONOCULAR);
    }
    // step3.3. 实在找不到足够的匹配特征点,运动模型跟踪失败
    if (nmatches < 20)
        return false;

    // step4. 位姿BA: 只优化当前帧位姿
    Optimizer::PoseOptimization(&mCurrentFrame);

    // step5. 剔除外点
    int nmatchesMap = 0;
    for (int i = 0; i < mCurrentFrame.N; i++) {
```

```
        if (mCurrentFrame.mvpMapPoints[i]) {
            if (mCurrentFrame.mvbOutlier[i]) {
                MapPoint *pMP = mCurrentFrame.mvpMapPoints[i];
                mCurrentFrame.mvpMapPoints[i] = static_cast<MapPoint *>(NULL);
                mCurrentFrame.mvbOutlier[i] = false;
            } else if (mCurrentFrame.mvpMapPoints[i]->Observations() > 0)
                nmatchesMap++;
        }
    }

    // step6. 匹配的地图点数超过10个就认为是跟踪成功
    return nmatchesMap >= 10;
}
```

为保证位姿估计的准确性,对于双目/RGBD相机,为前一帧生成临时地图点.

```
void Tracking::UpdateLastFrame() {

    // step1. 根据参考关键帧确定当前帧的位姿,使用关键帧是因为参考关键帧位姿更准确
    KeyFrame *pRef = mLastFrame.mpReferenceKF;
    cv::Mat Tlr = mlRelativeFramePoses.back();
    mLastFrame.SetPose(Tlr * pRef->GetPose());


    // step2. 对于双目/RGBD相机,生成新的临时地图点
    if (mnLastKeyFrameId == mLastFrame.mnId || mSensor == System::MONOCULAR)
        return;
    // step2.1. 将上一帧种存在深度的特征点按深度从小到大排序
    vector<pair<float, int> > vDepthIdx;
    vDepthIdx.reserve(mLastFrame.N);
    for (int i = 0; i < mLastFrame.N; i++) {
        float z = mLastFrame.mvDepth[i];
        if (z > 0) {
            vDepthIdx.push_back(make_pair(z, i));
        }
    }
    sort(vDepthIdx.begin(), vDepthIdx.end());
    // step2.2. 将上一帧中没三角化的特征点三角化出来,作为临时地图点
    int nPoints = 0;        // 统计处理了多少特征点
    for (size_t j = 0; j < vDepthIdx.size(); j++) {
        int i = vDepthIdx[j].second;
        bool bCreateNew = false;
        MapPoint *pMP = mLastFrame.mvpMapPoints[i];
        if (!pMP || pMP->Observations() < 1) {
            bCreateNew = true;
        }
        // step2.3. 这些临时地图点在CreateNewKeyFrame之前会被删除,因此不用添加其他复杂属性
        if (bCreateNew) {
            cv::Mat x3D = mLastFrame.UnprojectStereo(i);
            MapPoint *pNewMP = new MapPoint(x3D, mpMap, &mLastFrame, i);
            mLastFrame.mvpMapPoints[i] = pNewMP;
            mlpTemporalPoints.push_back(pNewMP);
        }
        // step2.4. 临时地图点数目足够(多于100个)或者深度太大(深度越大位置越不准确),就停止生成临时地图点
        if (vDepthIdx[j].first > mThDepth && nPoints > 100)
            break;
        nPoints++;
    }
}
```

根据参考帧估计位姿: `TrackReferenceKeyFrame()`

成员变量`mpReferenceKF`保存`Tracking`线程当前的参考关键帧,参考关键帧有两个来源:

- 每当`Tracking`线程创建一个新的参考关键帧时,就将其设为参考关键帧.
- 跟踪局部地图的函数`Tracking::TrackLocalMap()`内部会将与当前帧共视点最多的局部关键帧设为参考关键帧.

成员函数/变量	访问控制	意义
`TrackReferenceKeyFrame()`	`protected`	根据参考帧估计位姿
`KeyFrame* mpReferenceKF`	`protected`	参考关键帧, `TrackReferenceKeyFrame()`与该关键帧匹配搜索关键点

```
bool Tracking::TrackReferenceKeyFrame() {

    // step1. 根据当前帧描述子计算词袋
    mCurrentFrame.ComputeBow();

    // step2. 根据词袋寻找匹配
    ORBmatcher matcher(0.7, true);
    vector<MapPoint *> vpMapPointMatches;
    int nmatches = matcher.SearchByBoW(mpReferenceKF, mCurrentFrame, vpMapPointMatches);
    if (nmatches < 15)  // 词袋匹配过少则跟踪失败
```

```
        return false;

    // step3. 将上一帧位姿设为初始位姿估计值
    mCurrentFrame.SetPose(mLastFrame.mTcw);
    mCurrentFrame.mvpMapPoints = vpMapPointMatches;

    // step4. 位姿BA: 只优化当前帧位姿
    Optimizer::PoseOptimization(&mCurrentFrame);

    // step5. 剔除外点
    int nmatchesMap = 0;
    for (int i = 0; i < mCurrentFrame.N; i++) {
        if (mCurrentFrame.mvpMapPoints[i]) {
            if (mCurrentFrame.mvbOutlier[i]) {
                MapPoint *pMP = mCurrentFrame.mvpMapPoints[i];
                mCurrentFrame.mvpMapPoints[i] = static_cast<MapPoint *>(NULL);
                mCurrentFrame.mvbOutlier[i] = false;
            } else if (mCurrentFrame.mvpMapPoints[i]->Observations() > 0)
                nmatchesMap++;
        }
    }

    // step6. 匹配的地图点数超过10个就认为是跟踪成功
    return nmatchesMap >= 10;
}
```

> 思考: 为什么函数`Tracking::TrackReferenceKeyFrame()`没有检查当前参考帧`mpReferenceKF`是否被`LocalMapping`线程删除了?
>
> 回答: 因为`LocalMapping`线程剔除冗余关键帧函数`LocalMapping::KeyFrameCulling()`不会删除最新的参考帧,有可能被删除的都是之前的参考帧.

通过重定位估计位姿: `Relocalization()`

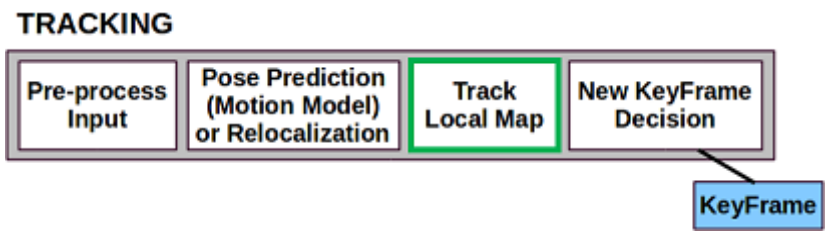
当`TrackWithMotionModel()`和`TrackReferenceKeyFrame()`都失败后,就会调用函数`Relocalization()`通过重定位估计位姿.

```
bool Tracking::Relocalization() {
    // step1. 根据当前帧描述子计算词袋
    mCurrentFrame.ComputeBoW();

    // step2. 根据词袋找到当前帧的候选参考关键帧
    vector<KeyFrame *> vpCandidateKFs = mpKeyFrameDB->DetectRelocalizationCandidates(&mCurrentFrame);
    const int nKFs = vpCandidateKFs.size();
    vector<PnPsolver *> vpPnPsolvers;
    vpPnPsolvers.resize(nKFs);

    // step3. 遍历所有的候选参考关键帧,通过BOW进行快速匹配,用匹配结果初始化PnPsolver
    vector<vector<MapPoint *> > vvpMapPointMatches;
    vector<bool> vbDiscarded;
    ORBmatcher matcher(0.75, true);
    int nCandidates = 0;
    for (int i = 0; i < nKFs; i++) {
        int nmatches = matcher.SearchByBoW(pKF, mCurrentFrame, vvpMapPointMatches[i]);
        PnPsolver *pSolver = new PnPsolver(mCurrentFrame, vvpMapPointMatches[i]);
        pSolver->SetRansacParameters(0.99, 10, 300, 4, 0.5, 5.991);
        vpPnPsolvers[i] = pSolver;
        nCandidates++;
    }

    // step4. 使用PnPsolver估计初始位姿,并根据初始位姿获取足够的匹配特征点
    bool bMatch = false;
    ORBmatcher matcher2(0.9, true);

    while (nCandidates > 0 && !bMatch) {
        for (int i = 0; i < nKFs; i++) {
            vector<bool> vbInliers;
            int nInliers;
            bool bNoMore;
            // step4.1. 通过PnPsolver估计初始位姿
            PnPsolver *pSolver = vpPnPsolvers[i];
            cv::Mat Tcw = pSolver->iterate(5, bNoMore, vbInliers, nInliers);
            if (bNoMore) {
                vbDiscarded[i] = true;
                nCandidates--;
            }
            // step4.2. 位姿BA: 只优化当前帧位姿
            Tcw.copyTo(mCurrentFrame.mTcw);
            set<MapPoint *> sFound;
            const int np = vbInliers.size();
            for (int j = 0; j < np; j++) {
                if (vbInliers[j]) {
                    mCurrentFrame.mvpMapPoints[j] = vvpMapPointMatches[i][j];
                    sFound.insert(vvpMapPointMatches[i][j]);
                } else
                    mCurrentFrame.mvpMapPoints[j] = NULL;
```

```
            }
            int nGood = Optimizer::PoseOptimization(&mCurrentFrame);

            // step4.3. 剔除外点
            for (int io = 0; io < mCurrentFrame.N; io++)
                if (mCurrentFrame.mvbOutlier[io])
                    mCurrentFrame.mvpMapPoints[io] = static_cast<MapPoint *>(NULL);

            // step4.4. 若匹配特征点数目太少,则尝试第2次进行特征匹配+位姿优化
            if (nGood < 50) {
                int nadditional = matcher2.SearchByProjection(mCurrentFrame, vpCandidateKFs[i], sFound, 10, 100);
                if (nadditional + nGood >= 50) {
                    nGood = Optimizer::PoseOptimization(&mCurrentFrame);
                    // step4.5. 若匹配特征点数目太少,则尝试第3次进行特征匹配+位姿优化
                    if (nGood > 30 && nGood < 50) {
                        sFound.clear();
                        for (int ip = 0; ip < mCurrentFrame.N; ip++)
                            if (mCurrentFrame.mvpMapPoints[ip])
                                sFound.insert(mCurrentFrame.mvpMapPoints[ip]);
                        nadditional = matcher2.SearchByProjection(mCurrentFrame, vpCandidateKFs[i], sFound, 3, 64);
                        if (nGood + nadditional >= 50) {
                            nGood = Optimizer::PoseOptimization(&mCurrentFrame);
                            for (int io = 0; io < mCurrentFrame.N; io++)
                                if (mCurrentFrame.mvbOutlier[io])
                                    mCurrentFrame.mvpMapPoints[io] = NULL;
                        }
                    }
                }
            }
            // step4.6. 若最后匹配数目终于足够了,则跟踪成功
            if (nGood >= 50) {
                bMatch = true;
                break;
            }
        }
    }

    // step5. 返回是否跟踪成功
    if (!bMatch) {
        return false;
    } else {
        mnLastRelocFrameId = mCurrentFrame.mnId;
        return true;
    }
}
```

跟踪局部地图: `TrackLocalMap()`

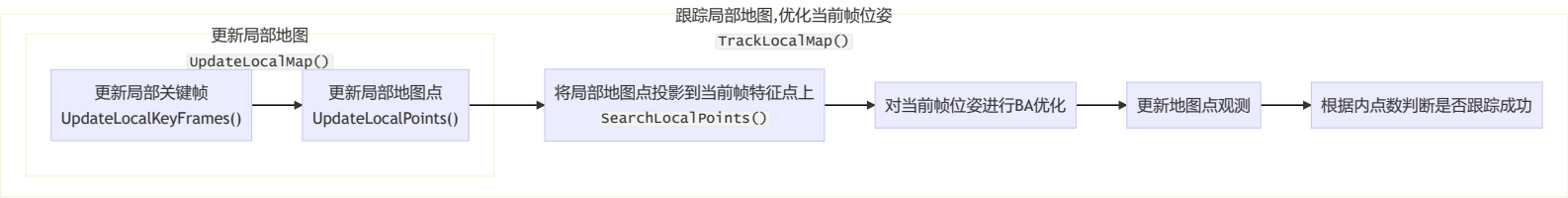
成员函数/变量	访问控制	意义
`bool TrackLocalMap()`	`protected`	更新局部地图并优化当前帧位姿
`void UpdateLocalMap()`	`protected`	更新局部地图
`std::vector<KeyFrame*> mvpLocalKeyFrames`	`protected`	局部关键帧列表
`std::vector<MapPoint*> mvpLocalMapPoints`	`protected`	局部地图点列表
`void SearchLocalPoints()`	`protected`	将局部地图点投影到当前帧特征点上



成功估计当前帧的初始位姿后,基于当前位姿更新局部地图并优化当前帧位姿,主要流程:

1. 更新局部地图,包括局部关键帧列表`mvpLocalKeyFrames`和局部地图点列表`mvpLocalMapPoints`.
2. 将局部地图点投影到当前帧特征点上.
3. 进行位姿BA,优化当前帧位姿.
4. 更新地图点观测数值,统计内点个数.

   这里的地图点观测数值会被用作`LocalMapping`线程中`LocalMapping::MapPointCulling()`函数剔除坏点的标准之一.
5. 根据内点数判断是否跟踪成功.




```cpp
bool Tracking::TrackLocalMap() {

    // step1. 更新局部地图,包括局部关键帧mvpLocalKeyFrames和局部地图点mvpLocalMapPoints
    UpdateLocalMap();

    // step2. 将局部地图点投影到当前帧特征点上
    SearchLocalPoints();

    // step3. 位姿BA: 只优化当前帧位姿
    Optimizer::PoseOptimization(&mCurrentFrame);

    // step4. 更新地图点观测,统计内点个数
    // 这里的地图点观测数值会被用作LocalMapping线程中MapPointCulling()函数剔除坏点的标准之一
    mnMatchesInliers = 0;
    for (int i = 0; i < mCurrentFrame.N; i++) {
        if (mCurrentFrame.mvpMapPoints[i]) {
            if (!mCurrentFrame.mvbOutlier[i]) {
                mCurrentFrame.mvpMapPoints[i]->IncreaseFound(); // 位姿估计用到该地图点
                if (mCurrentFrame.mvpMapPoints[i]->Observations() > 0)
                    mnMatchesInliers++;
            }
        }
    }

    // step5. 判断是否跟踪成功: 若刚发生过重定位,则标准严苛一点,否则标准宽松一点.(防止误闭环)
    if (mCurrentFrame.mnId < mnLastRelocFrameId + mMaxFrames && mnMatchesInliers < 50)
        return false;
    if (mnMatchesInliers < 30)
        return false;
    else
        return true;
}
```

函数`Tracking::UpdateLocalMap()`依次调用函数`Tracking::UpdateLocalKeyFrames()`更新局部关键帧列表`mvpLocalKeyFrames`和函数`Tracking::UpdateLocalPoints()`更新局部地图点列表`mvpLocalMapPoints`.

```cpp
void Tracking::UpdateLocalMap() {
    UpdateLocalKeyFrames();         // 更新局部关键帧列表mvpLocalKeyFrames
    UpdateLocalPoints();            // 更新局部地图点列表mvpLocalMapPoints
}
```

- 函数`Tracking::UpdateLocalKeyFrames()`内,局部关键帧列表`mvpLocalKeyFrames`会被清空并重新赋值,包括以下3部分:

  1. 当前地图点的所有共视关键帧.
  2. `1`中所有关键帧的父子关键帧.
  3. `1`中所有关键帧共视关系前`10`大的共视关键帧.

  更新完局部关键帧列表`mvpLocalKeyFrames`后,还将与当前帧共视关系最强的关键帧设为参考关键帧`mpReferenceKF`.

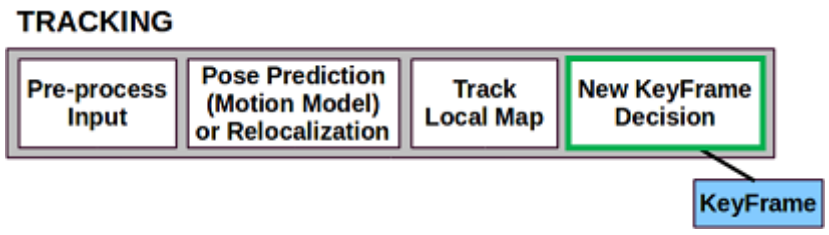
- 函数`Tracking::UpdateLocalPoints()`内,局部地图点列表`mvpLocalMapPoints`会被清空并赋值为局部关键帧列表`mvpLocalKeyFrames`的所有地图点.

函数`Tracking::SearchLocalPoints()`将局部地图点投影到当前帧特征点上

```cpp
void Tracking::SearchLocalPoints() {
    // step1. 当前帧地图点已经匹配了特征点
    for (MapPoint *pMP: mCurrentFrame.mvpMapPoints) {
        if (pMP) {
            if (pMP->isBad()) {
                *vit = static_cast<MapPoint *>(NULL);
            } else {
                pMP->IncreaseVisible();
                pMP->mnLastFrameSeen = mCurrentFrame.mnId;
                pMP->mbTrackInView = false;
            }
        }
    }
```

```
14
15      // step2. 统计视野内地图点数目
16      int nToMatch = 0;
17      for (MapPoint *pMP: mvpLocalMapPoints) {
18          if (pMP->mnLastFrameSeen == mCurrentFrame.mnId)
19              continue;
20          if (pMP->isBad())
21              continue;
22
23          if (mCurrentFrame.isInFrustum(pMP, 0.5)) {
24              pMP->IncreaseVisible();
25              nToMatch++;
26          }
27      }
28
29      // Step 3: 如果需要进行投影匹配的点的数目大于0，就进行投影匹配
30      if (nToMatch > 0) {
31          ORBmatcher matcher(0.8);
32          int th = 1;
33          if (mSensor == System::RGBD)
34              th = 3;
35          if (mCurrentFrame.mnId < mnLastRelocFrameId + 2)
36              th = 5;
37          matcher.SearchByProjection(mCurrentFrame, mvpLocalMapPoints, th);
38      }
39  }
```

关键帧的创建



判断是否需要创建新关键帧: `NeedNewKeyFrame()`

是否生成关键帧,需要考虑以下几个方面:

1. 最近是否进行过重定位,重定位后位姿不会太准,不适合做参考帧.
2. 当前系统的工作状态: 如果`LocalMapping`线程还有很多`KeyFrame`没处理的话,不适合再给它增加负担了.
3. 距离上次创建关键帧经过的时间: 如果很长时间没创建关键帧了的话,就要抓紧创建关键帧了.
4. 当前帧的质量: 当前帧观测到的地图点要足够多,同时与参考关键帧的重合程度不能太大.

具体的代码比较乱;不看了.

总体而言,ORB-SLAM2插入关键帧的策略还是比较宽松的,因为后面`LocalMapping`线程的函数`LocalMapping::KeyFrameCulling()`会剔除冗余关键帧,因此在系统处理得过来的情况下,要尽量多创建关键帧.

创建新关键帧: `CreateNewKeyFrame()`

创建新关键帧时,对于双目/RGBD相机输入情况下也创建新地图点.

```
1   void Tracking::CreateNewKeyFrame() {
2       // step1. 构造关键帧
3       KeyFrame *pKF = new KeyFrame(mCurrentFrame, mpMap, mpKeyFrameDB);
4
5       // step2. 将创建出的关键帧设为参考关键帧
6       mpReferenceKF = pKF;
7       mCurrentFrame.mpReferenceKF = pKF;
8
9       // step3. 对于双目/RGBD相机生成新的地图点
10      if (mSensor != System::MONOCULAR) {
11          mCurrentFrame.UpdatePoseMatrices();
12
13          // step3.1. 按深度从小到大排序关键点
14          vector<pair<float, int> > vDepthIdx;
15          vDepthIdx.reserve(mCurrentFrame.N);
16          for (int i = 0; i < mCurrentFrame.N; i++) {
17              float z = mCurrentFrame.mvDepth[i];
18              if (z > 0) {
19                  vDepthIdx.push_back(make_pair(z, i));
20              }
21          }
22
23          if (!vDepthIdx.empty()) {
24              sort(vDepthIdx.begin(), vDepthIdx.end());
25              // step3.2. 找出没对应地图点的特征点,并创建新地图点
26              int nPoints = 0;
27              for (size_t j = 0; j < vDepthIdx.size(); j++) {
28                  int i = vDepthIdx[j].second;
29                  bool bCreateNew = false;
30                  MapPoint *pMP = mCurrentFrame.mvpMapPoints[i];
```

```
                if (!pMP)
                    bCreateNew = true;
                else if (pMP->Observations() < 1) {
                    bCreateNew = true;
                    mCurrentFrame.mvpMapPoints[i] = static_cast<MapPoint *>(NULL);
                }
                if (bCreateNew) {
                    cv::Mat x3D = mCurrentFrame.UnprojectStereo(i);
                    MapPoint *pNewMP = new MapPoint(x3D, pKF, mpMap);
                    pNewMP->AddObservation(pKF, i);
                    pKF->AddMapPoint(pNewMP, i);
                    pNewMP->ComputeDistinctiveDescriptors();
                    pNewMP->UpdateNormalAndDepth();
                    mpMap->AddMapPoint(pNewMP);
                    mCurrentFrame.mvpMapPoints[i] = pNewMP;
                }
                nPoints++;
                // step3.3. 地图点过多(多于100个)或深度太深(误差太大),则停止生成地图点
                if (vDepthIdx[j].first > mThDepth && nPoints > 100)
                    break;
            }
        }
    }

    // step4. 插入关键帧
    mpLocalMapper->InsertKeyFrame(pKF);
    mpLocalMapper->SetNotStop(false);
    mnLastKeyFrameId = mCurrentFrame.mnId;
    mpLastKeyFrame = pKF;
}
```

跟踪函数: `Track()`

主要关注成员变量 `mState` 的变化:

值	意义
`SYSTEM_NOT_READY`	系统没有准备好,一般就是在启动后加载配置文件和词典文件时候的状态
`NO_IMAGES_YET`	还没有接收到输入图像
`NOT_INITIALIZED`	接收到图像但未初始化成功
`OK`	跟踪成功
`LOST`	跟踪失败

```
void Tracking::Track() {

    // 维护状态
    if (mState == NO_IMAGES_YET) {
        mState = NOT_INITIALIZED;
    }

    unique_lock<mutex> lock(mpMap->mMutexMapUpdate);

    // step1. 若还没初始化,则尝试初始化
    if (mState == NOT_INITIALIZED) {
        if (mSensor == System::STEREO || mSensor == System::RGBD)
            StereoInitialization();
        else
            MonocularInitialization();
        if (mState != OK)
            return;
    } else {
        // step2. 若系统已初始化,就进行跟踪(或重定位)
        bool bOK;

        // step2.1. 符合条件时,优先根据运动模型跟踪,如运动模型跟踪失败,就根据参考帧进行跟踪
        if (mState == OK) {
            CheckReplacedInLastFrame();
            if (mVelocity.empty() || mCurrentFrame.mnId < mnLastRelocFrameId + 2) {
                bOK = TrackReferenceKeyFrame();
            } else {
                bOK = TrackWithMotionModel();
                if (!bOK)
                    bOK = TrackReferenceKeyFrame();
            }
        } else {
            // step2.2. 若上一帧没跟踪丢失,则这一帧重定位
            bOK = Relocalization();
        }
        // step2.3. 设置当前帧的参考关键帧
        mCurrentFrame.mpReferenceKF = mpReferenceKF;
```

```
        // step3. 跟踪局部地图,进一步优化当前帧位姿
        //  之前的跟踪过程都是仅根据前面某一帧进行的位姿优化,TrackLocalMap()使用局部地图进行位姿优化
        if (bOK)
            bOK = TrackLocalMap();

        // step4. 根据跟踪结果判断跟踪状态
        if (bOK)
            mState = OK;
        else
            mState = LOST;

        // step5. 跟踪成功之后的后处理
        if (bOK) {
            // step5.1. 更新恒速运动模型
            if (!mLastFrame.mTcw.empty()) {
                cv::Mat LastTwc = cv::Mat::eye(4, 4, CV_32F);
                mLastFrame.GetRotationInverse().copyTo(LastTwc.rowRange(0, 3).colRange(0, 3));
                mLastFrame.GetCameraCenter().copyTo(LastTwc.rowRange(0, 3).col(3));
                mVelocity = mCurrentFrame.mTcw * LastTwc;   // mVelocity = Tcl = Tcw * Twl
            } else
                mVelocity = cv::Mat();

            // step5.2. 剔除失效地图点
            for (int i = 0; i < mCurrentFrame.N; i++) {
                MapPoint *pMP = mCurrentFrame.mvpMapPoints[i];
                if (pMP)
                    if (pMP->Observations() < 1) {
                        mCurrentFrame.mvbOutlier[i] = false;
                        mCurrentFrame.mvpMapPoints[i] = static_cast<MapPoint *>(NULL);
                    }
            }

            // step5.3. 清除恒速模型跟踪中UpdateLastFrame创建的当前帧临时地图点(跟踪失败的话就不清空临时地图点么?)
            for (list<MapPoint *>::iterator lit = mlpTemporalPoints.begin(), lend =
mlpTemporalPoints.end();
                 lit != lend; lit++) {
                MapPoint *pMP = *lit;
                delete pMP;
            }
            mlpTemporalPoints.clear();

            // step5.4. 检测并插入关键帧,双目/RGBD相机会创建新地图点
            if (NeedNewKeyFrame())
                CreateNewKeyFrame();

            // step5.5. 删除外点
            for (int i = 0; i < mCurrentFrame.N; i++) {
                if (mCurrentFrame.mvpMapPoints[i] && mCurrentFrame.mvbOutlier[i])
                    mCurrentFrame.mvpMapPoints[i] = static_cast<MapPoint *>(NULL);
            }
        }

        // step6. 若系统刚启动没多久就跟踪失败的话,就直接重启
        if (mState == LOST) {
            if (mpMap->KeyFramesInMap() <= 5) {
                cout << "Track lost soon after initialisation, reseting..." << endl;
                mpSystem->Reset();
                return;
            }
        }

        // step7. 更新上一帧数据
        mLastFrame = Frame(mCurrentFrame);
    }

    // step8. 记录位姿信息
    if (!mCurrentFrame.mTcw.empty()) {
        cv::Mat Tcr = mCurrentFrame.mTcw * mCurrentFrame.mpReferenceKF->GetPoseInverse();   // 相对于参考
帧Tcr = Tcw * Twr
        mlRelativeFramePoses.push_back(Tcr);
        mlpReferences.push_back(mpReferenceKF);
        mlFrameTimes.push_back(mCurrentFrame.mTimeStamp);
        mlbLost.push_back(mState == LOST);
    } else {
        // 跟踪失败就使用上一帧数据作为当前帧记录
        mlRelativeFramePoses.push_back(mlRelativeFramePoses.back());
        mlpReferences.push_back(mlpReferences.back());
        mlFrameTimes.push_back(mlFrameTimes.back());
        mlbLost.push_back(mState == LOST);
    }
}
```

Tracking流程中的关键问题(暗线)

地图点的创建与删除

1. `Tracking`线程中初始化过程(`Tracking::MonocularInitialization()`和`Tracking::StereoInitialization()`)会创建新的地图点.
2. `Tracking`线程中创建新的关键帧(`Tracking::CreateNewKeyFrame()`)会创建新的地图点.
3. `Tracking`线程中根据恒速运动模型估计初始位姿(`Tracking::TrackWithMotionModel()`)也会产生临时地图点,但这些临时地图点在跟踪成功后会被马上删除.

所有的非临时地图点都是由关键帧建立的,`Tracking::TrackWithMotionModel()`中由非关键帧建立的关键点被设为临时关键点,很快会被删掉,仅作增强帧间匹配用,不会对建图产生任何影响.这也不违反**只有关键帧才能参与`LocalMapping`和`LoppClosing`线程**的原则.

> 思考: 为什么跟踪失败的话不删除这些局部地图点
>
> 跟踪失败的话不会产生关键帧,这些地图点也不会被注册进地图,不会对之后的建图产生影响.
>
> 思考: 那会不会发生内存泄漏呢?
>
> 不会的,因为最后总会有一帧跟踪上,这些临时地图点都被保存在了成员变量`mlpTemporalPoints`中,跟踪成功后会删除所有之前的临时地图点.

关键帧与地图点间发生关系的时机

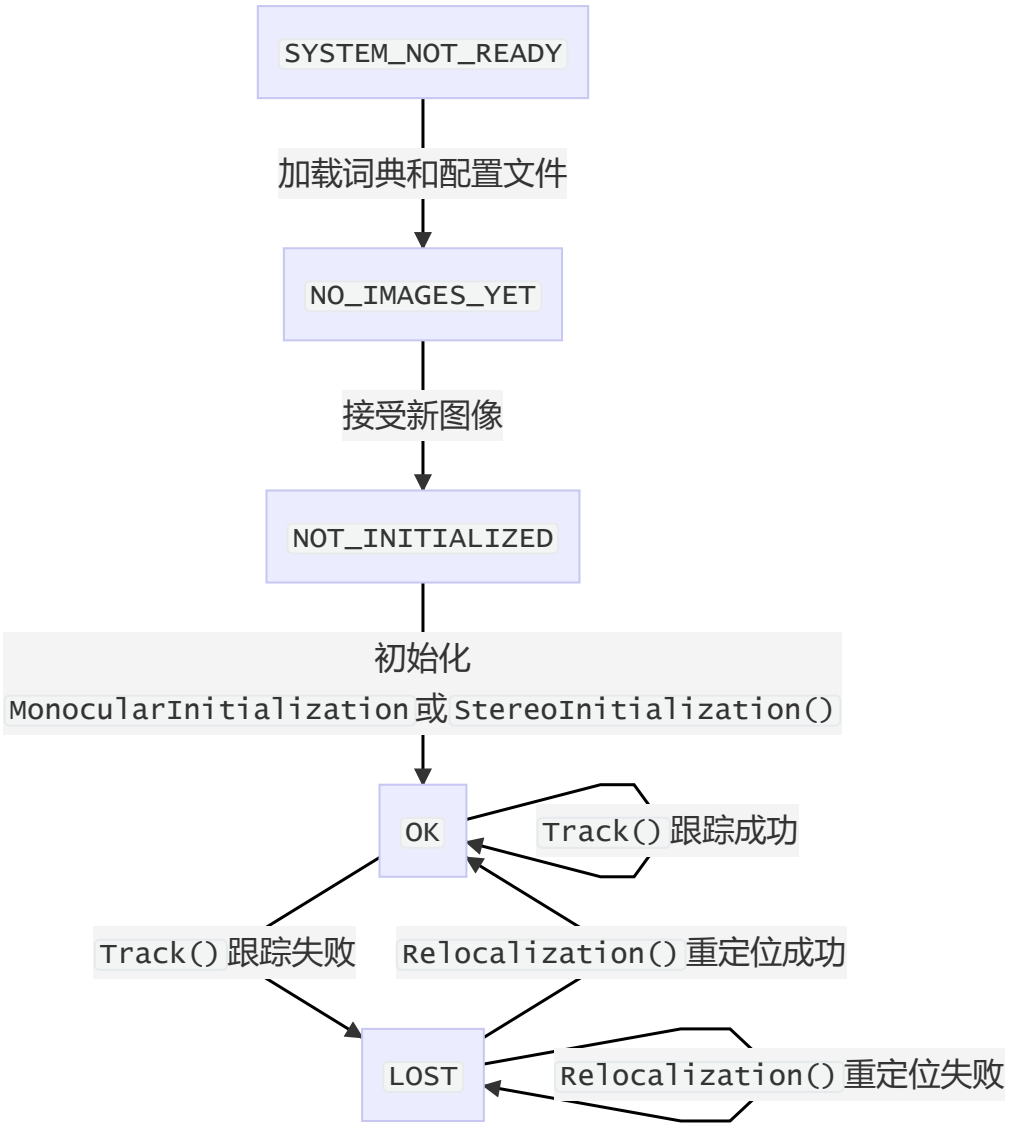
- 新创建出来的非临时地图点都会与创建它的关键帧建立双向连接.
- 通过`ORBmatcher::SearchByXXX()`函数匹配得到的帧点关系只建立单向连接:
    - 只在关键帧中添加了对地图点的观测(将地图点加入到关键帧对象的成员变量`mvpMapPoints`中了).
    - 没有在地图点中添加对关键帧的观测(地图点的成员变量`mObservations`中没有该关键帧).

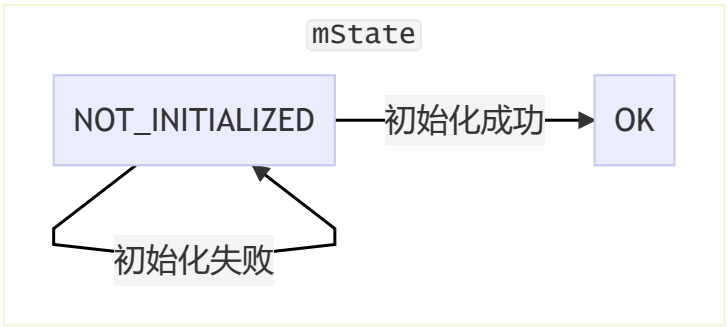
这为后文中`LocalMapping`线程中函数`LocalMapping::ProcessNewKeyFrame()`对关键帧中地图点的处理埋下了伏笔.该函数通过检查地图点中是否有对关键点的观测来判断该地图点是否是新生成的.

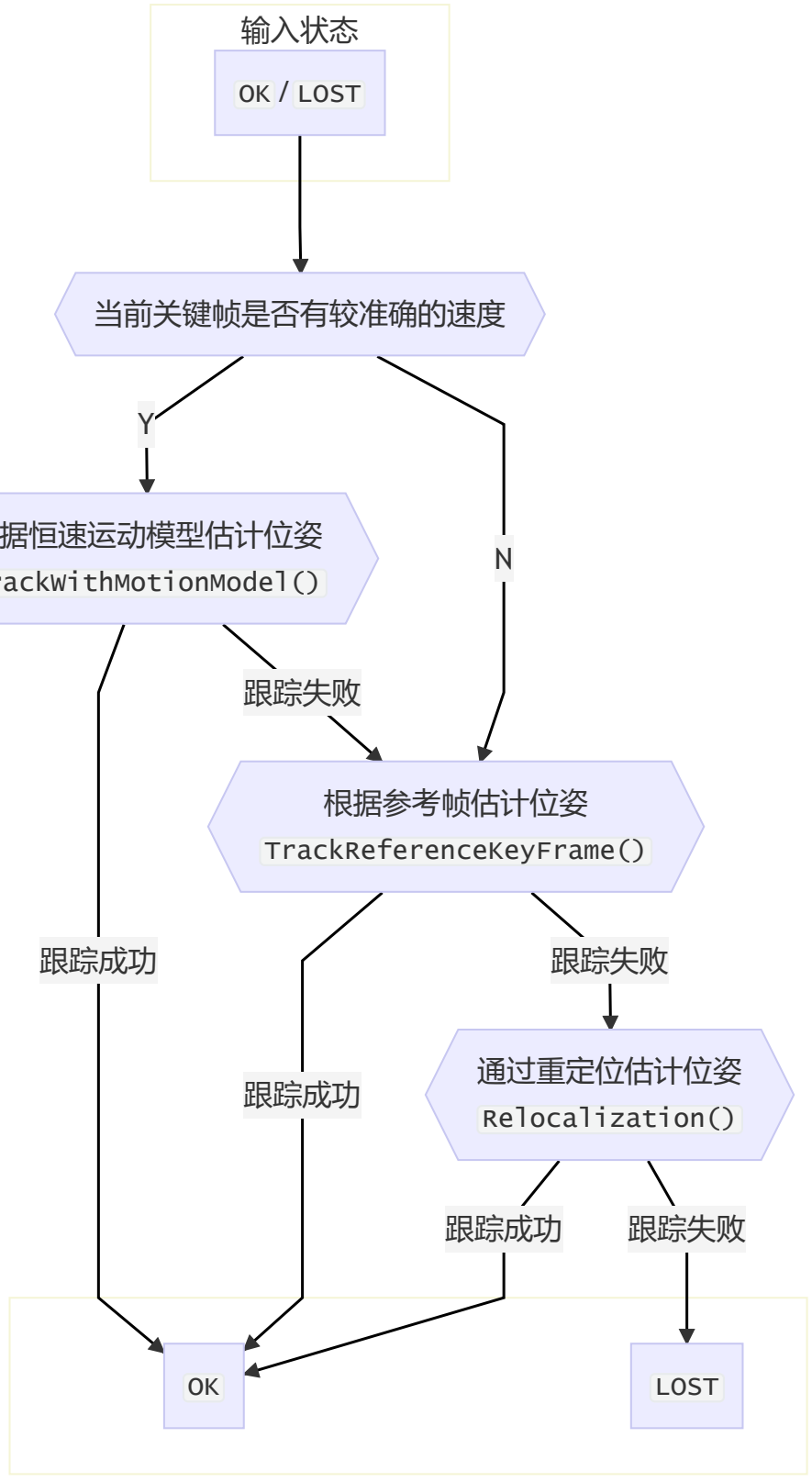
```
void LocalMapping::ProcessNewKeyFrame() {

    // 遍历关键帧中的地图点
    const vector<MapPoint *> vpMapPointMatches = mpCurrentKeyFrame->GetMapPointMatches();
    for (MapPoint *pMP : vpMapPointMatches) {
        if (!pMP->IsInKeyFrame(mpCurrentKeyFrame)) {
            // step3.1. 该地图点是跟踪本关键帧时匹配得到的,在地图点中加入对当前关键帧的观测
            pMP->AddObservation(mpCurrentKeyFrame, i);
            pMP->UpdateNormalAndDepth();
            pMP->ComputeDistinctiveDescriptors();
        } else
        {
            // step3.2. 该地图点是跟踪本关键帧时新生成的,将其加入容器mlpRecentAddedMapPoints待筛选
            mlpRecentAddedMapPoints.push_back(pMP);
        }
    }

    // ...
}
```

参考关键帧: `mpReferenceKF`

- 参考关键帧的用途:
    1. `Tracking`线程中函数`Tracking::TrackReferenceKeyFrame()`根据参考关键帧估计初始位姿.
    2. 用于初始化新创建的`MapPoint`的参考帧`mpRefKF`,函数`MapPoint::UpdateNormalAndDepth()`中根据参考关键帧`mpRefKF`更新地图点的平均观测距离.
- 参考关键帧的指定:
    1. `Traking`线程中函数`Tracking::CreateNewKeyFrame()`创建完新关键帧后,会将新创建的关键帧设为参考关键帧.
    2. `Tracking`线程中函数`Tracking::TrackLocalMap()`跟踪局部地图过程中调用函数`Tracking::UpdateLocalMap()`,其中调用函数`Tracking::UpdateLocalKeyFrames()`,将与当前帧共视程度最高的关键帧设为参考关键帧.


SYSTEM_NOT_READY
加载词典和配置文件
NO_IMAGES_YET
接受新图像
NOT_INITIALIZED
初始化
MonocularInitialization 或 StereoInitialization()
OK
Track() 跟踪成功
Track() 跟踪失败
Relocalization() 重定位成功
LOST
Relocalization() 重定位失败


mState
NOT_INITIALIZED
初始化成功
OK
初始化失败


输入状态
OK / LOST
当前关键帧是否有较准确的速度
Y
N
根据恒速运动模型估计位姿
TrackWithMotionModel()
跟踪失败
根据参考帧估计位姿
TrackReferenceKeyFrame()
跟踪成功
跟踪失败
跟踪成功
通过重定位估计位姿
Relocalization()
跟踪成功
跟踪失败
OK
LOST