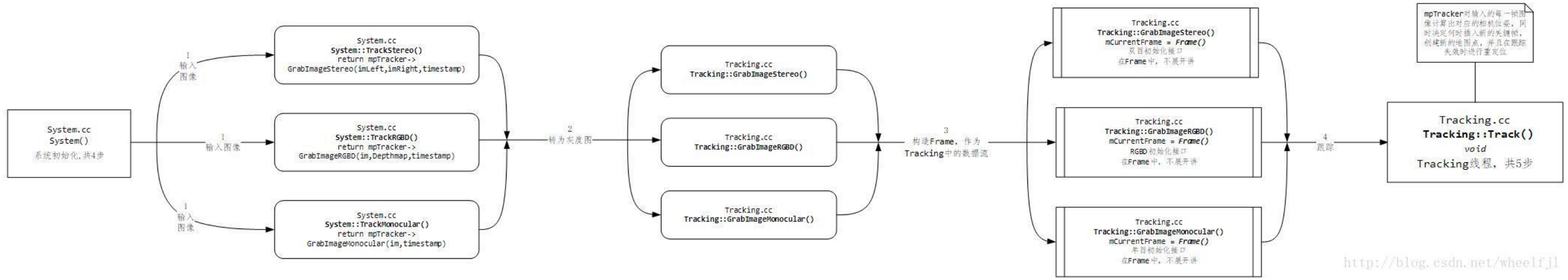
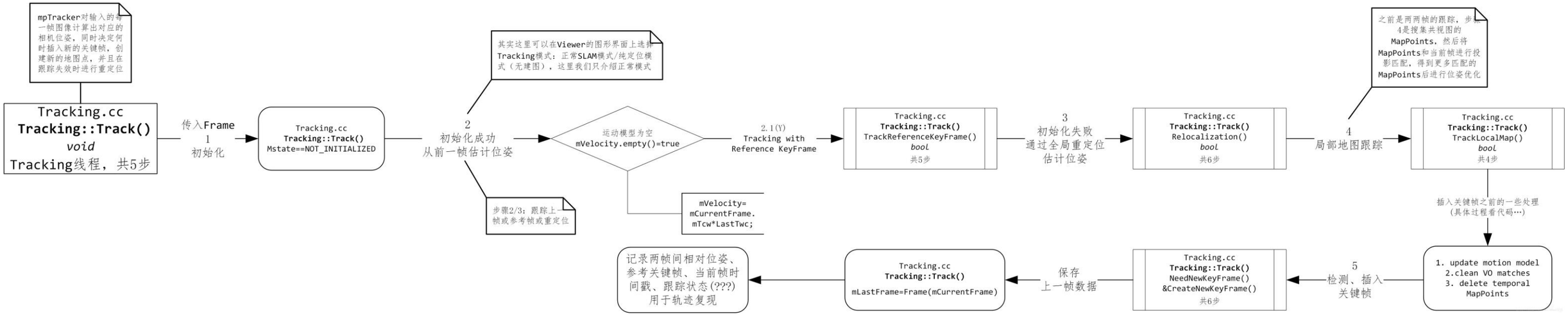
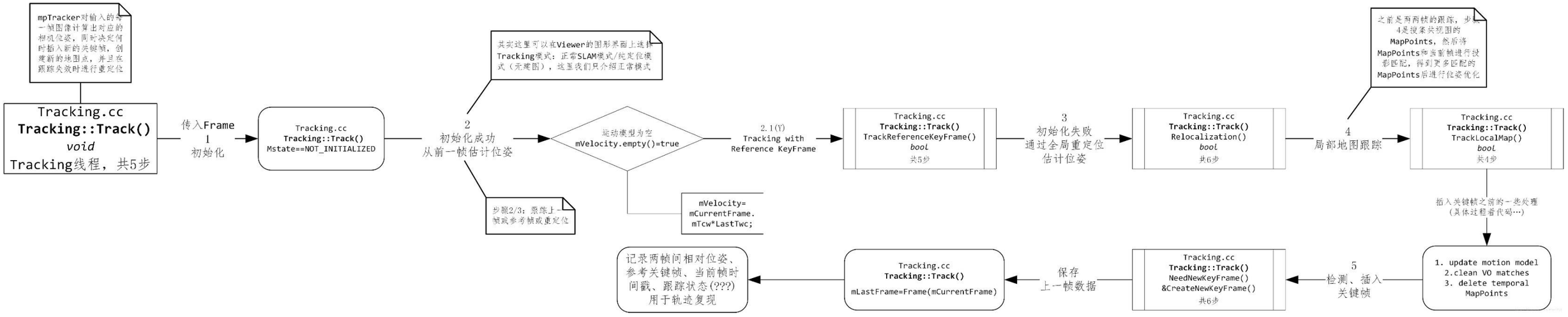
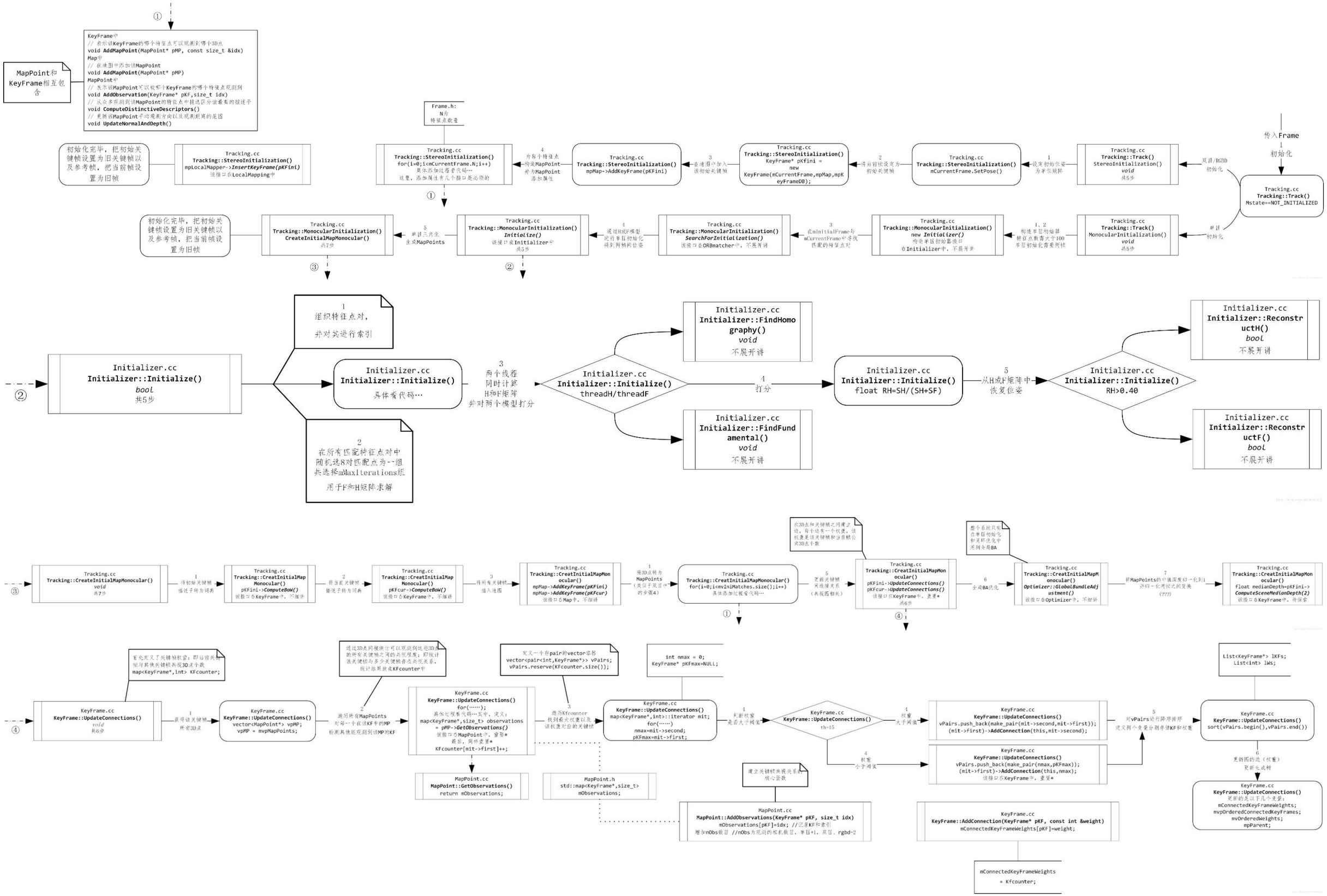


http://blog.csdn.net/wheelfjl






①
KeyFrame中
// 表示该KeyFrame的哪个特征点可以观测到哪个3D点
void AddMapPoint(MapPoint* pMP, const size_t &idx)
Map中
// 在地图中添加该MapPoint
void AddMapPoint(MapPoint* pMP)
MapPoint中
// 表示该MapPoint可以被哪个KeyFrame的哪个特征点观测到
void AddObservation(KeyFrame* pKF,size_t idx)
// 从众多观测到该MapPoint的特征点中挑选区分该最佳的描述子
void ComputeDistinctiveDescriptors()
// 更新该MapPoint平均观测方向以及观测距离的范围
void UpdateNormalAndDepth()
MapPoint和KeyFrame相互包含
Frame.h:
N为
特征点数量
初始化完毕，把初始关键帧设置为旧关键帧以及参考帧，把当前帧设置为旧帧
Tracking.cc
Tracking::StereoInitialization()
mpLocalMapper->InsertKeyFrame(pKFini)
该接口在LocalMapping中
Tracking.cc
Tracking::StereoInitialization()
for(i=0;i<mCurrentFrame.N;i++)
具体添加过程看代码…
这里，添加属性有几个接口是必须的
4
为每个特征点
均设MapPoint
并为MapPoint
添加属性
Tracking.cc
Tracking::StereoInitialization()
mpMap->AddKeyFrame(pKFini)
3
在地图中加入
该初始关键帧
Tracking.cc
Tracking::StereoInitialization()
KeyFrame* pKFini =
new
KeyFrame(mCurrentFrame,mpMap,mpKeyFrameDB);
2
将当前帧设定为
初始关键帧
Tracking.cc
Tracking::StereoInitialization()
mCurrentFrame.SetPose();
1
设定初始位姿
为单位矩阵
Tracking.cc
Tracking::Track()
StereoInitialization()
void
共5步
双目/RGBD
初始化
传入Frame
1
初始化
Tracking.cc
Tracking::Track()
Mstate==NOT_INITIALIZED
初始化完毕，把初始关键帧设置为旧关键帧以及参考帧，把当前帧设置为旧帧
Tracking.cc
Tracking::MonocularInitialization()
CreateInitialMapMonocular()
共7步
5
单目三角化
生成MapPoints
Tracking.cc
Tracking::MonocularInitialization()
Initialize()
该接口在Initializer中
共5步
4
通过H或F模型
进行单目初始化
得到两帧间位姿
Tracking.cc
Tracking::MonocularInitialization()
SearchForInitialization()
该接口在ORBmatcher中，不展开讲
3
在mInitialFrame与
mCurrentFrame中寻找
匹配的特征点对
Tracking.cc
Tracking::MonocularInitialization()
new Initializer()
构造单目初始器接口
在Initializer中，不展开讲
1，2
构造单目初始器
特征点数需大于100
单目初始化需要两帧
Tracking.cc
Tracking::Track()
MonocularInitialization()
void
共5步
单目
初始化
②
Initializer.cc
Initializer::Initialize()
bool
共5步
1
组织特征点对，
并对其进行索引
Initializer.cc
Initializer::Initialize()
具体看代码…
2
在所有匹配特征点对中
随机选8对匹配点为一组
共选择mMaxIterations组
用于F和H矩阵求解
3
两个线程
同时计算
H和F矩阵
并对两个模型打分
Initializer.cc
Initializer::Initialize()
threadH/threadF
Initializer.cc
Initializer::FindHomography()
void
不展开讲
Initializer.cc
Initializer::FindFundamental()
void
不展开讲
4
打分
Initializer.cc
Initializer::Initialize()
float RH=SH/(SH+SF)
5
从H或F矩阵中
恢复位姿
Initializer.cc
Initializer::Initialize()
RH>0.40
Initializer.cc
Initializer::ReconstructH()
bool
不展开讲
Initializer.cc
Initializer::ReconstructF()
bool
不展开讲
③
Tracking.cc
Tracking::CreateInitialMapMonocular()
void
共7步
1
将初始关键帧
描述子转为词袋
Tracking.cc
Tracking::CreatInitialMapMonocular()
pKFini->ComputeBoW()
该接口在KeyFrame中，不细讲
2
将当前关键帧
描述子转为词袋
Tracking.cc
Tracking::CreatInitialMapMonocular()
pKFcur->ComputeBoW()
该接口在KeyFrame中，不细讲
3
将所有关键帧
插入地图
Tracking.cc
Tracking::CreatInitialMapMonocular()
mpMap->AddKeyFrame(pKFini)
mpMap->AddKeyFrame(pKFcur)
该接口在Map中，不细讲
4
将3D点转为
MapPoints
（类似于双目中
的步骤4）
Tracking.cc
Tracking::CreatInitialMapMonocular()
for(i=0;i<nvIniMatches.size();i++)
具体添加过程看代码…
在3D点和关键帧之间建立
边，每个边有一个权重，该
权重是该关键帧和当前帧公
共3D点个数
5
更新关键帧
间连接关系
（共视图相关）
Tracking.cc
Tracking::CreatInitialMapMonocular()
pKFini->UpdateConnections()
pKFcur->UpdateConnections()
该接口在KeyFrame中，重要*
共6步
整个系统只有
在单目初始化
和闭环优化中
用到全局BA
6
全局BA优化
Tracking.cc
Tracking::CreatInitialMapMonocular()
Optimizer::GlobalBundleAdjustment()
该接口在Optimizer中，不细讲
7
将MapPoints的平均深度归一化到1
进行一定程度之间变换
(???)
Tracking.cc
Tracking::CreatInitialMapMonocular()
float medianDepth=pKFini->ComputeSceneMedianDepth(2)
该接口在KeyFrame中，待探索
④
KeyFrame.cc
KeyFrame::UpdateConnections()
void
共6步
首先定义了关键帧权重：即当前关键帧与其他关键帧共视3D点个数
map<KeyFrame*,int> KFcounter;
1
获得该关键帧
所有3D点
KeyFrame.cc
KeyFrame::UpdateConnections()
vector<MapPoint*> vpMP;
vpMP = mvpMapPoints;
2
遍历所有MapPoints
对每一个在该KF中的MP
检测其能观测到该MP的KF
通过3D点间接性可以观测到这些3D点
的所有关键帧之间的共视观测；即统计
该关键帧与多少关键帧存在共视关系，
统计结果放在KFcounter中
KeyFrame.cc
KeyFrame::UpdateConnections()
for(……);
具体过程看代码…其中，定义：
map<KeyFrame*,size_t> observations
= pMP->GetObservations()
该接口在MapPoint中，重要*
最后，同样重要*
KFcounter[mit->first]++;
MapPoint.cc
MapPoint::GetObservations()
return mObservations;
定义一个存pair的vector容器
vector<pair<int,KeyFrame*>> vPairs;
vPairs.reserve(KFcounter.size());
3
遍历Kfcounter
找到最大权重以及
该权重对应的关键帧
MapPoint.h
std::map<KeyFrame*,size_t>
mObservations;
int nmax = 0;
KeyFrame* pKFmax=NULL;
KeyFrame.cc
KeyFrame::UpdateConnections()
map<KeyFrame*,int>::iterator mit;
for(……)
nmax=mit->second;
pKFmax=mit->first;
4
判断权重
是否大于阈值
KeyFrame.cc
KeyFrame::UpdateConnections()
th=15
建立关键帧共视关系的
核心函数
MapPoint.cc
MapPoint::AddObservations(KeyFrame* pKF, size_t idx)
mObservations[pKF]=idx; //记录KF和索引
增加nObs数目 //nObs为观测的相机数目，单目+1，双目、rgbd-2
4
权重
大于阈值
KeyFrame.cc
KeyFrame::UpdateConnections()
vPairs.push_back(make_pair(mit->second,mit->first));
(mit->first)->AddConnection(this,mit->second);
4
权重
小于阈值
KeyFrame.cc
KeyFrame::UpdateConnections()
vPairs.push_back(make_pair(nmax,pKFmax));
(mit->first)->AddConnection(this,nmax);
该接口在KeyFrame中，重要*
KeyFrame.cc
KeyFrame::AddConnection(KeyFrame* pKF, const int &weight)
mConnectedKeyFrameWeights[pKF]=weight;
mConnectedKeyFrameWeights
= Kfcounter;
5
对vPairs进行降序排序
定义两个变量分别存储KF和权重
List<KeyFrame*> lKFs;
List<int> lWs;
KeyFrame.cc
KeyFrame::UpdateConnections()
sort(vPairs.begin(),vPairs.end())
6
更新图的边（权重）
更新生成树
KeyFrame.cc
KeyFrame::UpdateConnections()
更新的是以下几个变量：
mConnectedKeyFrameWeights;
mvpOrderedConnectedKeyFrames;
mvOrderedWeights;
mpParent;

2
初始化成功，从前一帧估计位姿

运动模型为空
mVelocity.empty()=true

mVelocity=
mCurrentFrame.
mTcw*LastTwc;

2.1(N)
Tracking
with
Motion
Model

Tracking.cc
Tracking::Track()
TrackWithMotionModel()
bool
共4步

1
外部目前没需要
对上一帧产生一些
新的、临时MapPoints

Tracking.cc
Tracking::TrackWithMotionModel()
UpdateLastFrame()
这考虑1:这里只生成临时地图点
不会为MapPoints添加属性
注意2:只会将距离小于mTHDepth阈值的深度
点包装成MapPoints，并且只新添加100个
MapPoints

过程约等于CreateNewKeyFrame()

2.1(Y)
Tracking with
Reference KeyFrame

Tracking.cc
Tracking::Track()
TrackReferenceKeyFrame()
bool
共5步

⑥

匹配数
小于10

Tracking.cc
Tracking::TrackReferenceKeyFrame()
bool
共5步

1
将当前描述子
转成BoW向量

Tracking.cc
Tracking::TrackReferenceKeyFrame()
mCurrentFrame.*ComputeBoW()*
该接口在Frame中，待探索

2
通过BoW对KF和F中
特征点进行匹配
返回匹配成数量

Tracking.cc
Tracking::TrackReferenceKeyFrame()
SearchByBoW()
该接口在ORBmathcer中，待探索

重定位中也用到该成员
函数

3
用上一帧位姿
作为初始值

Tracking.cc
Tracking::TrackReferenceKeyFrame()
mCurrentFrame.*SetPose(mLastFrame.mTcw)*
该接口在Frame中

3
通过优化
求得位姿

Tracking.cc
Tracking::TrackWithMotionModel()
Optimizer::PoseOptimization(&mCurrentFrame)
该接口在Optimizer中，不展开讲

4
剔除外点
匹配到的地图点
数量大于10
返回ture

Tracking.cc
Tracking::TrackWithMotionModel()
for(……);具体过程看代码
return nmatchesMap>=10;

3D-2D 最小化重投影误差
e = (u,v) - project(Tcw*Pw)
只优化Frame的Tcw，不优化MapPoints的坐标

5
剔除外点
匹配到的地图点
数量大于10
返回ture

Tracking.cc
Tracking::TrackReferenceKeyFrame()
for(……);具体过程看代码
return nmatchesMap>=10;

Tracking.cc
Tracking::TrackReferenceKeyFrame()
Optimizer::PoseOptimization(&mCurrentFrame)
该接口在Optimizer中，不展开讲

SearchByProjection成员函数一共有4种重载，
当前用在：
Tracking线程中的TrackWithMotionModel
剩下三种分别用在：
Tracking线程中的Relocalisation
Tracking线程中的Local Map
LoopClosing线程中的Loop Detect

⑤

ORBmatcher.h
int SearchByProjection(Frame
&CurrentFrame, const Frame
&LastFrame, const float th,
const bool bMono)

⑤

Tracking.cc
Tracking::TrackWithMotionModel()
SearchByProjection()
该接口在ORBmatcher中，待探索

若跟踪到的特征点太少，则扩大搜索半径再来一次

跟踪过程中需要将当前帧与上帧进行特征点匹配，将上一帧的MapPoints投影到当前帧可以缩小匹配范围

3
初始化失败，通过全局重定位估计位姿

Tracking.cc
Tracking::Track()
Relocalization()
bool
共6步

1
将当前描述子
转成BoW向量

Tracking.cc
Tracking::Relocalization()
mCurrentFrame.*ComputeBoW()*
该接口在Frame中，待探索

2
找到与当前帧相似的
候选关键帧

Tracking.cc
Tracking::TrackReferenceKeyFrame()
vector<KeyFrame*> vpCandidateKFs
= mpKeyFrame->
DetectRelocalizationCandidates(&mCurrentFrame)
该接口在KeyFrameDatabase中
共5步

⑦
除了个别变量不同，过程同闭环候选帧检测

3
通过BoW对KF和F中
特征点进行匹配
返回匹配成数量

Tracking.cc
Tracking::Relocalization()
SearchByBoW()
该接口在ORBmathcer中

⑱

SearchByBow成员函数一共有2种重载，当前用在：
Tracking线程中的Relocalisation
剩下1种用在：
LoopClosing线程中的ComputeSim3

4
通过EPnP算法
估计位姿

Tracking.cc
Tracking::Relocalization()
cv::Mat Tcw = pSolver->
iterate(5,bNoMore,vbInliers,nInliers);
该接口在PnPsolver中，不展讲

5
通过优化
求得位姿
返回内点个数

Tracking.cc
Tracking::Relocalization()
Optimizer::PoseOptimization(&mCurrentFrame)
该接口在Optimizer中，不展开讲

6
若内点少于50
通过投影方式
对之前未匹配点
进行匹配
并再次进行优化

Tracking.cc
Tracking::Relocalization()
SearchByProjection()
该接口在Optimizer中，参考⑰

⑰

Tracking.cc
Tracking::Relocalization()
Optimizer::PoseOptimization(&mCurrentFrame)
该接口在Optimizer中，不展开讲

Tracking.cc
Tracking::Relocalization()
第一次得到足够的内点来得到粗糙
姿态就代表重定位成功

⑦

KeyFrameDatabase.cc
KeyFrameDatabase::DetectLoopCandidates(KeyFrame* pKF,
float minScore)
vector<KeyFrame>*
共5步

1
遍历本一步得关
KF中的每一个word
找出相当相邻
有公有单词的
所有关键帧

KeyFrameDatabase.cc
KeyFrameDatabase::DetectLoopCandidates()
for(……)
具体过程看代码……

2
在所有关联候选帧中
找出与当前关键帧中
公有单词最多的单词数
并权重最小单词阈值

KeyFrameDatabase.cc
KeyFrameDatabase::DetectLoopCandidates()
int maxCommonWords;
int minCommonWords =
maxCommonWords*0.8f;

3
再次遍历关联候选帧
都出公有单词数大于minCommonWords
且单词匹配度大于minScore
存入lScoreAndMatch

list<pair<float,KefFrame*>> lScoreAndMatch;

KeyFrameDatabase.cc
KeyFrameDatabase::DetectLoopCandidates()
lScoreAndMatch.push_back(make_pair(s1,pKF1));

4
遍历lScoreAndMatch
将关联帧分组
计算累计得分
4.1
对lScoreAndMatch中
每一个KF获得一个
关联程度较高的组

KeyFrameDatabase.cc
KeyFrameDatabase::DetectLoopCandidates()
for(…遍历lScoreAndMatch…)
vector<KeyFrame*> vpNeighs =
pKF1->*GetBestCovisibilityKeyFrames(10);*
该接口在KeyFrame中，返回连接的前N个关键帧

4.2
依次计算组内总分accScore
组内最高分bestScore
最高分对应关键帧pBestKF
高分组阈值minScoreToRetain

list<pair<float,KefFrame*>> lAccScoreAndMatch;
float bestAccScore = minScore; // 所有组中得分最高组的得分

KeyFrameDatabase.cc
KeyFrameDatabase::DetectLoopCandidates()
lAccScoreAndMatch.push_back(make_pair(accScore,pBestKF));
float minScoreToRetain = 0.75f*bestAccScore;

5
遍历lAccScoreAndMatch
得到组得分大于
minScoreToRetain的组

KeyFrameDatabase.cc
KeyFrameDatabase::DetectLoopCandidates()
lAccScoreAndMatch.push_back(make_pair(accScore,pBestKF));
float minScoreToRetain = 0.75f*bestAccScore;

返回候选闭环帧

ORBmatcher.cc
ORBmatcher::SearchByProjection(Frame
&CurrentFrame, KeyFrame *pKF, const
set<MapPoint*> &sAlreadyFound, const
float th , const int ORBdist)
int
共4步

⑰

1
遍历该
关键帧
地图点

const vector<MapPoint*> vpMPs =
pKF->*GetMapPointMatches();*

ORBmatcher.cc
ORBmatcher::SearchByProjection()
for(…vpMPs…)
具体过程看代码…

2
将关键帧
的地图点
投影到当前帧
前后进行一些判断
并确定搜索范围

ORBmatcher.cc
ORBmatcher::SearchByProjection()
具体步骤基本同⑯
const vector<size_t> vIndices2 =
CurrentFrame.GetFeaturesInArea(u, v,
radius, nPredictedLevel-1,
nPredictedLevel+1);

通过地图点的距离dist3D，预测特征对应金字塔层nPredictedLevel，并获取搜索window大小（th*scale），在以上约束的范围内，搜索得到候选匹配点集合向量vIndices2

3
计算地图点的
描述子和候选匹配点描述子距离
获得最近距离的最佳匹配
注意要满足距离小于
ORBdist

ORBmatcher.cc
ORBmatcher::SearchByProjection()
const int dist =
DescriptorDistance(dMP,dKF);

4
通过直方图
验证描述子
的方向是否匹配

ORBmatcher.cc
ORBmatcher::SearchByProjection()
具体过程同SearchByBoW第3步

角度直方图是用来剔除不满足两帧之间角度旋转的外点的，也就是所谓的旋转一致性检测
1. 将关键帧与当前帧匹配点的angle相减，得到rot（0<=rot<360），放入一个直方图中，对于每一对匹配点的角度差，均可以放入一个bin的范围内（360/HISTO_LENGTH）。
2. 统计直方图最高的三个bin保留，其他范围内的匹配点剔除。另外，若最高的比第二高的高10倍以上，则只保留最高的bin中的匹配点。

最后该函数会
1. 为当前帧生成和关键帧匹配上的地图点
2. 统计通过投影匹配上的点

ORBmatcher.cc
ORBmatcher::SearchByBoW(KeyFrame
*pKF, Frame &F,
std::vector<MapPoint*>
&vpMapPointMatches);
int
共5步

⑱

1
分别取出属于同一
node的ORB特征点
（只有属于同一Node
才有可能是匹配点）

ORBmatcher.cc
ORBmatcher::SearchByProjection()
for(…vpMPs…)
具体过程看代码…

2
分别遍历
关键帧中该node特征点
当前帧中该node特征点
找到了最佳匹配点
计算描述子距离

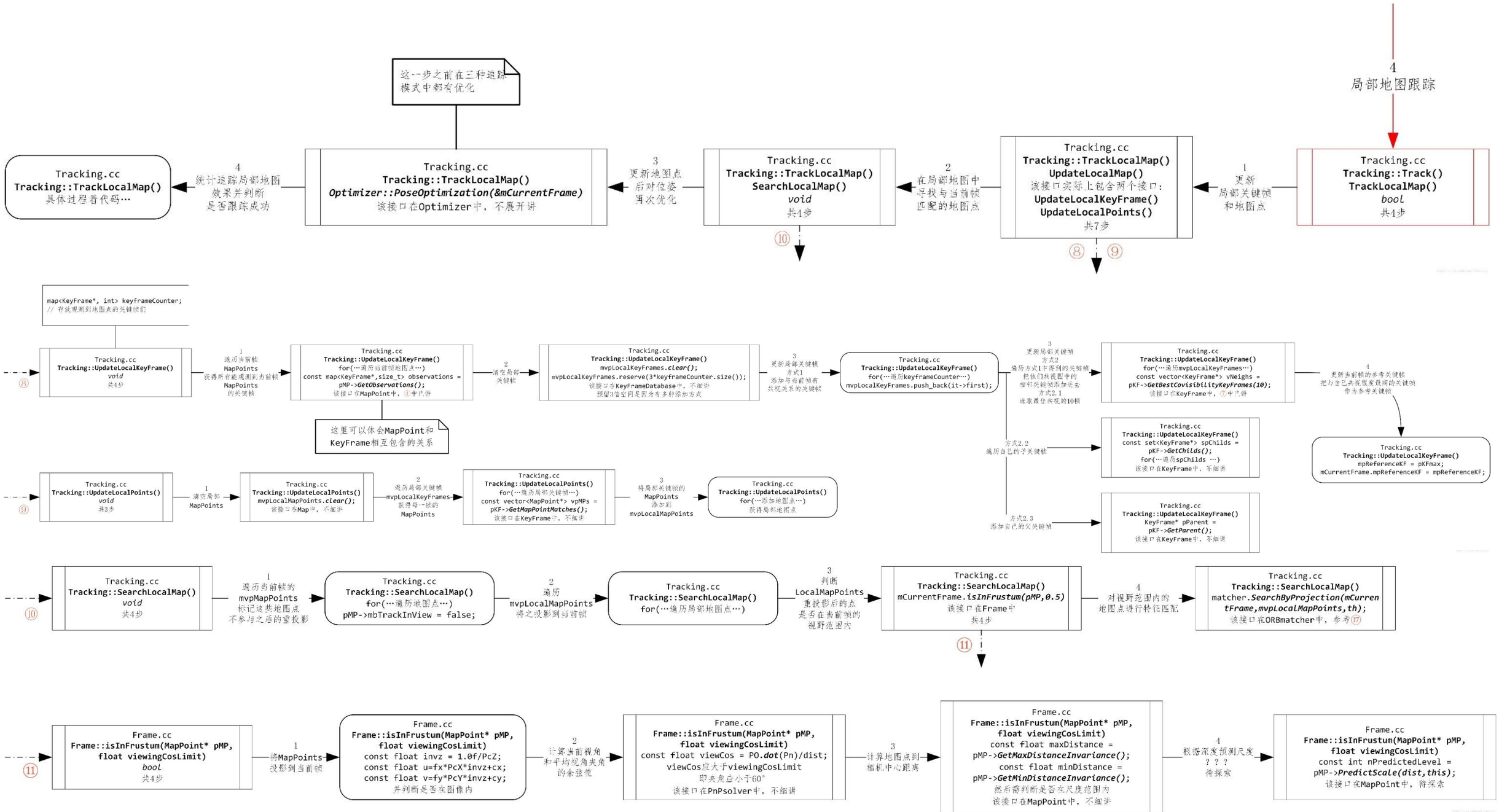
ORBmatcher.cc
ORBmatcher::SearchByProjection()
for(……)
具体过程看代码…

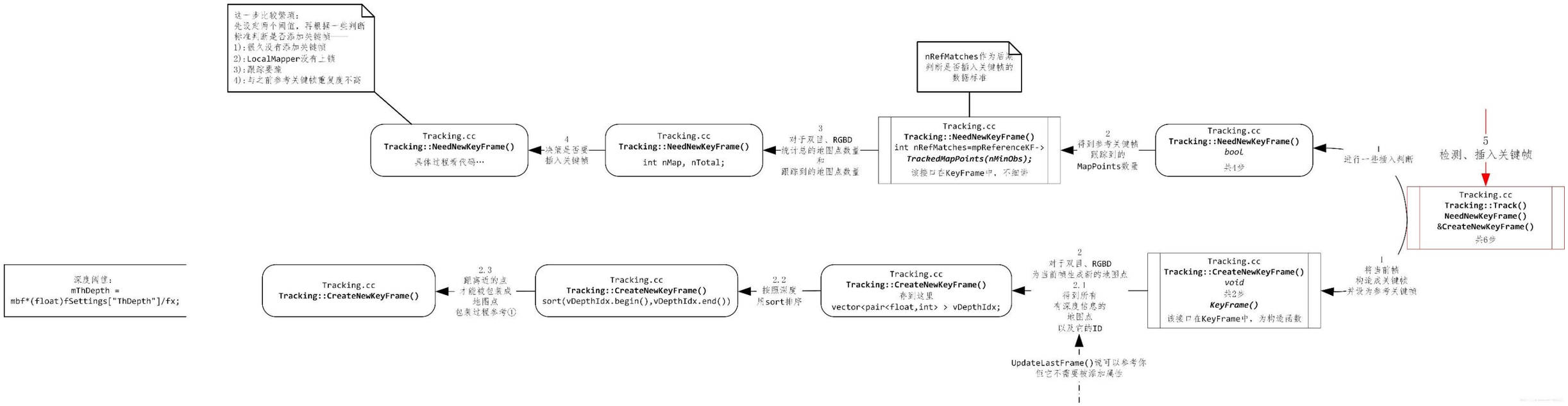
3
根据阈值
为
角度投票剔除误匹配

ORBmatcher.cc
ORBmatcher::SearchByProjection()
具体过程同SearchByProjection第4步

4
根据方向
剔除误匹配的点

ORBmatcher.cc
ORBmatcher::SearchByProjection()
ComputeThreeMaxima(rotHist,HISTO_L
ENGTH,ind1,ind2,ind3);
计算rotHist中最大的三个的Index
如果特征点的旋转角度变化量属于这三个
组，则保留
将除了ind1 ind2 ind3以外的匹配点去掉


4
局部地图跟踪
Tracking.cc
Tracking::Track()
TrackLocalMap()
bool
共1步
1
更新
局部关键帧
和地图点
Tracking.cc
Tracking::TrackLocalMap()
UpdateLocalMap()
该接口实际上包含两个接口：
UpdateLocalKeyFrame()
UpdateLocalPoints()
共7步
2
在局部地图中
寻找与当前帧
匹配的地图点
Tracking.cc
Tracking::TrackLocalMap()
SearchLocalMap()
void
共1步
3
更新地图点
后对位姿
再次优化
这一步之前在三种追踪
模式中都有优化
Tracking.cc
Tracking::TrackLocalMap()
Optimizer::PoseOptimization(&mCurrentFrame)
该接口在Optimizer中，不展开讲
4
统计追踪局部地图
效果并判断
是否跟踪成功
Tracking.cc
Tracking::TrackLocalMap()
具体过程看代码…
⑧
⑨
⑩
map<KeyFrame*, int> keyframeCounter;
// 存放观测到地图点的关键帧们
Tracking.cc
Tracking::UpdateLocalKeyFrame()
void
共4步
1
遍历当前帧
MapPoints
获得所有能观测到当前帧
MapPoints
的关键帧
Tracking.cc
Tracking::UpdateLocalKeyFrame()
for(…遍历当前帧地图点…)
const map<KeyFrame*,size_t> observations =
pMP->GetObservations();
该接口在MapPoint中，④中已讲
这里可以体会MapPoint和
KeyFrame相互包含的关系
2
清空局部
关键帧
Tracking.cc
Tracking::UpdateLocalKeyFrame()
mvpLocalKeyFrames.clear();
mvpLocalKeyFrames.reserve(3*keyframeCounter.size());
该接口在KeyFrameDatabase中，不细讲
预留3倍空间是因为考多种添加方式
3
更新局部关键帧
方式1
添加与当前帧有
共视关系的关键帧
Tracking.cc
Tracking::UpdateLocalKeyFrame()
for(…遍历keyframeCounter…)
mvpLocalKeyFrames.push_back(it->first);
3
更新局部关键帧
方式2
遍历方式1中得到的关键帧
把他们共视图中的
相邻关键帧添加进去
方式2.1
选取最佳共视的10帧
Tracking.cc
Tracking::UpdateLocalKeyFrame()
for(…遍历mvpLocalKeyFrames…)
const vector<KeyFrame*> vNeighs =
pKF->GetBestCovisibilityKeyFrames(10);
该接口在KeyFrame中，⑦中已讲
方式2.2
遍历自己的子关键帧
Tracking.cc
Tracking::UpdateLocalKeyFrame()
const set<KeyFrame*> spChilds =
pKF->GetChilds();
for(…遍历spChilds …)
该接口在KeyFrame中，不细讲
方式2.3
添加自己的父关键帧
Tracking.cc
Tracking::UpdateLocalKeyFrame()
KeyFrame* pParent =
pKF->GetParent();
该接口在KeyFrame中，不细讲
4
更新当前帧的参考关键帧
把与自己共视程度最高的关键帧
作为参考关键帧
Tracking.cc
Tracking::UpdateLocalKeyFrame()
mpReferenceKF = pKFmax;
mCurrentFrame.mpReferenceKF = mpReferenceKF;
Tracking.cc
Tracking::UpdateLocalPoints()
void
共3步
1
清空局部
MapPoints
Tracking.cc
Tracking::UpdateLocalPoints()
mvpLocalMapPoints.clear();
该接口在Map中，不细讲
2
遍历局部关键帧
mvpLocalKeyFrames
获得每一帧的
MapPoints
Tracking.cc
Tracking::UpdateLocalPoints()
for(…遍历局部关键帧…)
const vector<MapPoint*> vpMPs =
pKF->GetMapPointMatches();
该接口在KeyFrame中，不细讲
3
将局部关键帧的
MapPoints
添加到
mvpLocalMapPoints
Tracking.cc
Tracking::UpdateLocalPoints()
for(…添加地图点…)
获得局部地图点
Tracking.cc
Tracking::SearchLocalMap()
void
共4步
1
遍历当前帧的
mvpMapPoints
标记这些地图点
不参与之后的重投影
Tracking.cc
Tracking::SearchLocalMap()
for(…遍历地图点…)
pMP->mbTrackInView = false;
2
遍历
mvpLocalMapPoints
将之投影到当前帧
Tracking.cc
Tracking::SearchLocalMap()
for(…遍历局部地图点…)
3
判断
LocalMapPoints
重投影后的点
是否在当前帧的
视野范围内
Tracking.cc
Tracking::SearchLocalMap()
mCurrentFrame.isInFrustum(pMP,0.5)
该接口在Frame中
共4步
4
对视野范围内的
地图点进行特征匹配
Tracking.cc
Tracking::SearchLocalMap()
matcher.SearchByProjection(mCurrentFrame,mvpLocalMapPoints,th);
该接口在ORBmatcher中，参考⑰
Frame.cc
Frame::isInFrustum(MapPoint* pMP,
float viewingCosLimit)
bool
共4步
1
将MapPoints
投影到当前帧
Frame.cc
Frame::isInFrustum(MapPoint* pMP,
float viewingCosLimit)
const float invz = 1.0f/PcZ;
const float u=fx*PcX*invz+cx;
const float v=fy*PcY*invz+cy;
并判断是否在图像内
2
计算当前视角
和平均视角夹角
的余弦值
Frame.cc
Frame::isInFrustum(MapPoint* pMP,
float viewingCosLimit)
const float viewCos = PO.dot(Pn)/dist;
viewCos应大于viewingCosLimit
即夹角应小于60°
该接口在PnPsolver中，不细讲
3
计算地图点到
相机中心距离
Frame.cc
Frame::isInFrustum(MapPoint* pMP,
float viewingCosLimit)
const float maxDistance =
pMP->GetMaxDistanceInvariance();
const float minDistance =
pMP->GetMinDistanceInvariance();
然后需判断是否在尺度范围内
该接口在MapPoint中，不细讲
4
根据深度预测尺度
？？？
待探索
Frame.cc
Frame::isInFrustum(MapPoint* pMP,
float viewingCosLimit)
const int nPredictedLevel =
pMP->PredictScale(dist,this);
该接口在MapPoint中，待探索


这一步比较繁琐：
先设定两个阈值，再根据一些判断
标准判断是否添加关键帧——
1):很久没有添加关键帧
2):LocalMapper没有上锁
3):跟踪要强
4):与之前参考关键帧重复度不高
Tracking.cc
Tracking::NeedNewKeyFrame()
具体过程看代码…
4
决策是否要
插入关键帧
Tracking.cc
Tracking::NeedNewKeyFrame()
int nMap, nTotal;
3
对于双目、RGBD
统计总的地图点数量
和
跟踪到的地图点数量
nRefMatches作为后期
判断是否插入关键帧的
数据标准
Tracking.cc
Tracking::NeedNewKeyFrame()
int nRefMatches=mpReferenceKF->
TrackedMapPoints(nMinObs);
该接口在KeyFrame中，不细讲
2
得到参考关键帧
跟踪到的
MapPoints数量
Tracking.cc
Tracking::NeedNewKeyFrame()
bool
共4步
1
进行一些插入判断
5
检测、插入关键帧
Tracking.cc
Tracking::Track()
NeedNewKeyFrame()
&CreateNewKeyFrame()
共6步
1
将当前帧
构造成关键帧
并设为参考关键帧
Tracking.cc
Tracking::CreateNewKeyFrame()
void
共2步
KeyFrame()
该接口在KeyFrame中，为构造函数
2
对于双目、RGBD
为当前帧生成新的地图点
2.1
得到所有
有深度信息的
地图点
以及它的ID
UpdateLastFrame()说可以参考你
但它不需要被添加属性
Tracking.cc
Tracking::CreateNewKeyFrame()
在到这里
vector<pair<float,int> > vDepthIdx;
2.2
按照深度
后sort排序
Tracking.cc
Tracking::CreateNewKeyFrame()
sort(vDepthIdx.begin(),vDepthIdx.end())
2.3
距离近的点
才能被包装成
地图点
包装过程参考①
Tracking.cc
Tracking::CreateNewKeyFrame()
深度阈值：
mThDepth =
mbf*(float)fSettings["ThDepth"]/fx;