ORB extractor

some background:

这里用到的feature detection方法（ORB）的一些资料：

FAST特征点：

http://blog.csdn.net/hujingshuang/article/details/46898007

BRIEF特征描述子：

http://blog.csdn.net/hujingshuang/article/details/46910259

ORB：

ORB是一种结合FAST和BRIEF，并引入旋转不变性的一种特征点和描述子。

http://blog.csdn.net/hujingshuang/article/details/46984411

http://doc.okbase.net/ronny/archive/111309.html

原始paper：
http://www.cs.zju.edu.cn/~qpan/course/materials/ORB.pdf

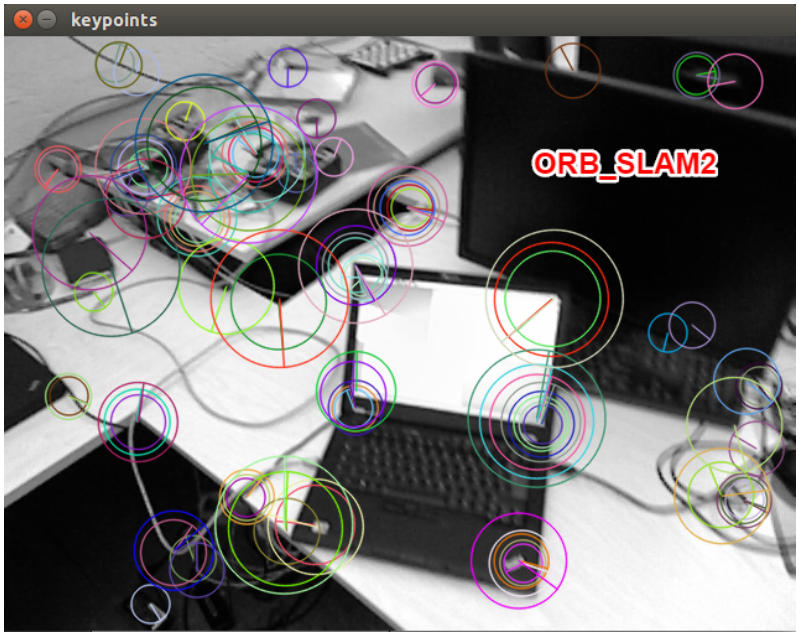
找到的一些资料：
http://blog.csdn.net/awww797877/article/details/52140995
http://www.fengbing.net/2016/04/03/%E4%B8%80%E6%AD%A5%E6%AD%A5%E5%AE%9E%E7%8E%B0slam2-orb%E7%89%B9%E5%BE%81%E6%A3%80%E6%B5%8B/
http://www.cnblogs.com/ronny/p/4083537.html

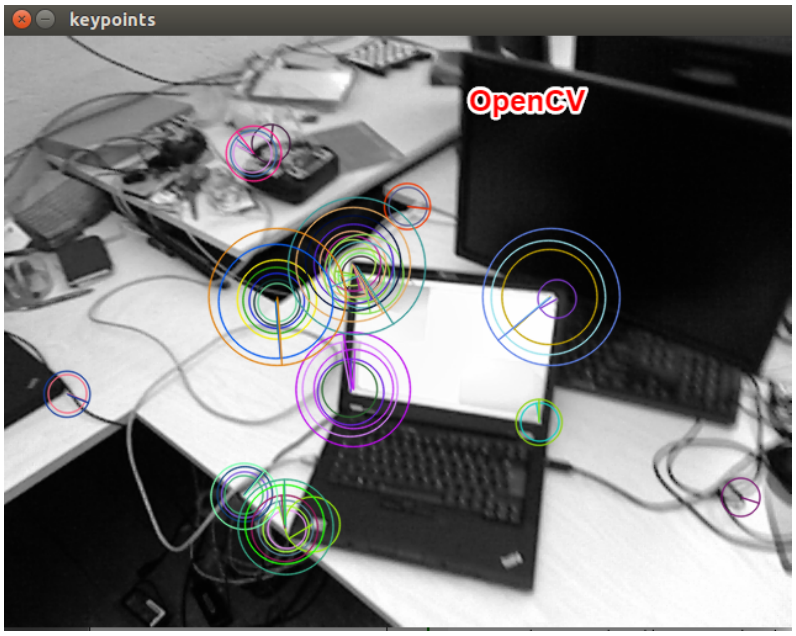
void **FAST**(InputArray **image**, vector<KeyPoint>& **keypoints**, int **threshold**,
bool **nonmaxSuppression**=true)

思路很清晰：
//
1.ComputePyramid() 建立图像金字塔，解决scale问题要用的
2.ComputeKeyPointsOctTree() 计算每一层的特征点，then按照四 叉树的方式进行空间划分，保证图像中的特征均匀，放入vectors中（虽然写的是OctTree但是貌似实际上是四叉树，难道是还没来得及实现ヽ(ゝ∇ゝ)ノ）
3.计算特征的方向，oFAST嘛，保证特征有旋转不变的特性。
4..根据论文中计算rBRIEF的方法计算描述子，实现还比较简单，就是按照paper的步骤，就不具体描述了

总之，我们从一副图片中得到了各个层次的下的 特征点（**KeyPoints**）和 描述子（**descriptors**） 这是ORB_Extrator所有能做的事情.

看看效果：





相较于opencv自带ORB，ORB_SLAM这样特征更加均匀。
由于成像存在透视畸变，我们调用UndistortKeyPoints()来得到畸变前的特征点mvKeysUn
然后在frame上画格子，把mvKeysUn按照空间特征放到格子里面去

OK，到现在为止我们可以得到每个Frame的KeyPts,Descriptor,然后把纠正后的坐标放进Frame的格子里面

数据结构：

MapPoint:

3D **position**
mean **viewing direction**

代表性的一个ORB描述子：跟其他能看到这破点的关键帧hamming距离最小的描述子 (有啥用？Update:在TrackLocalMap的时候寻找Local MP与当前帧的匹配的时候用的)
根据ORB特征尺度不变性约束，可观测的云点的最大距离dmax和最小距离dmin(有啥用？)

KeyFrame

Camera **pose** T, from world to camera.
相机内参
ORB feature->对应或者不对应着一个一个MapPoint(undistorted)

多种方法产生的keyframe和mappoint的选取和cull都很严格（why？点太少了能干啥？）

Co-visibility Graph

Node: KeyFrame
Edge: share the same MapPoint of at least 15
weight of edge: NUM of common MPs

pose optimazation 优化方法将闭环回路的误差
Covisibility太dense了，所以搞了个Essential Graph

Essential Graph

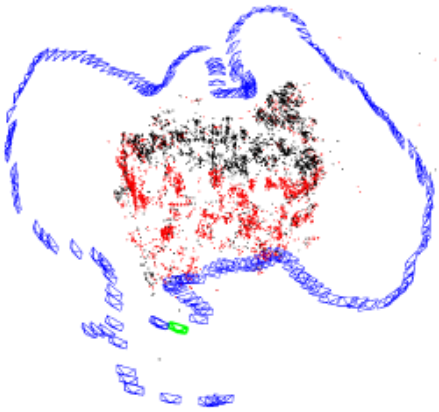
The Essential Graph contains the spanning tree, the subset of edges from the covisibility graph with high covisibility (weight $_{min}$ = 100), and the loop closure edges, resulting in a strong

Spanning Tree + 强关联（**>100**）的**covisibility**的边 + **Loop Closing**
network of cameras.

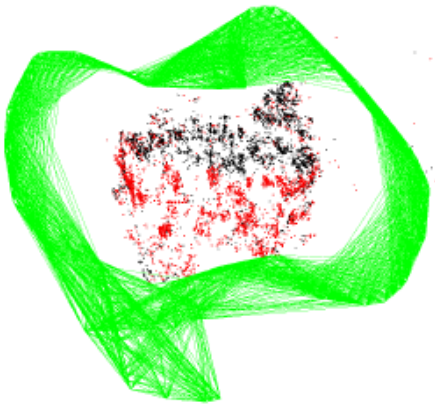
Spanning Tree

When a new keyframe is inserted, it is included in the tree linked to the keyframe which shares most point observations

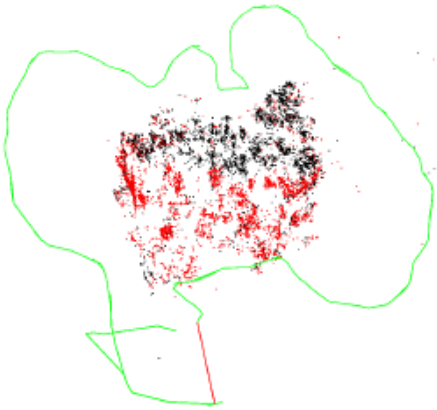
只有共视度最高的边



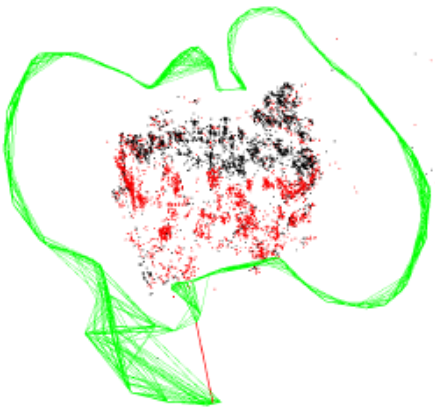
(a) KeyFrames (blue), Current Camera (green), MapPoints (black, red), Current Local MapPoints (red)



(b) Covisibility Graph



(c) Spanning Tree (green) and Loop Closure (red)



(d) Essential Graph

BOW(词袋) ->位置识别
TBD

1.初始化：

数学上研究半天了，囧，挺久的吧。有了基本认识，但是对尺度不确定性的不确定性没有数学上的理性认识->从原理上了解了这个不确定性，但是怎么消除这个不确定性，没人给出简单的解决方案，如果仅仅是用在AR这样的场景是不是会简单写呢？考虑单目所得到的深度都是不确定的，相对的。

具体有两篇文章：

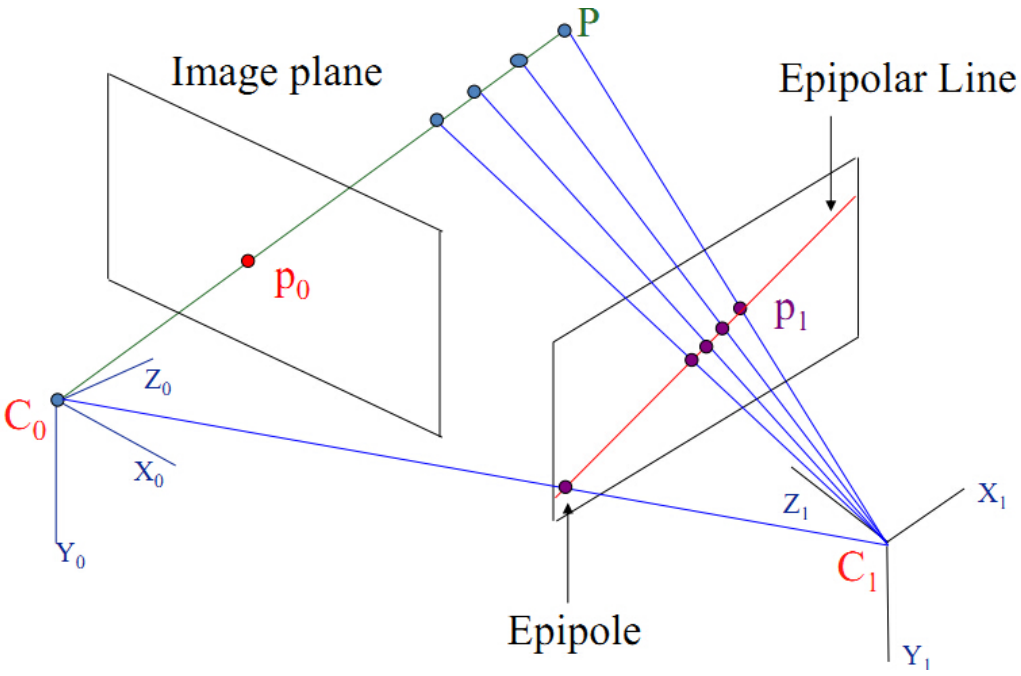
http://blog.csdn.net/heyijia0327/article/details/50758944
http://blog.csdn.net/heyijia0327/article/details/50774104

基本就是把MVG（多视图几何）里面的东西拿过来了，但是去掉了很多冗余的推导，定理

计算过程基本上比较熟悉了，以后写

对归一化坐标的微小的理解

归一化坐标 考虑分解为 $P=K[R|t]$ 的摄像机矩阵，并令 $x=PX$ 为图像上的一点．如果知道标定矩阵 K，用它的逆矩阵作用于点 x 得到点 $\hat{x}=K^{-1}x$．则 $\hat{x}=[R|t]X$，其中 $\hat{x}$ 是图像上的点在**归一化坐标**下的表示．它可以被视为空间点 X 在摄像机 $[R|t]$ 的标定矩阵等于单位矩阵 I 情形下的像．摄像机矩阵 $K^{-1}P=[R|t]$ 称为**归一化摄像机矩阵**，对于它，标定矩阵的影响已经去掉了．



其实就是从二维图像坐标 （u，v），在假定 f = 1，相机中心在(0,0)的情况下，p0在摄像机C0坐标系下的三维坐标

通过H模型或F模型进行单目初始化，得到相对运动，初始的MapPoints(某个scale下)，匹配关系
也就是：

假定第一帧为原点，相机位姿为 （I / 0），当前帧为位姿为（R，T），当前的匹配关系，从帧1到帧2 mvIniMatches -1代表姜了，和初始的Points: mvInitP3D 再次强调，这是在某个**scale**下！不准确，相对，怎么得到准确的是关键问题！

Tracking::CreateInitialMapMonocular()

//生成初始化的MP和KF

1. 用初始化->生成两KeyFrame，把keyFrame描述子计算成为BOW
2. 将关键帧加入Map **PS:分清楚 Map LocalMap Frame KeyFrame啥的**
3. 把初始化得到的3D点 mvInitP3D 包装成 MapPoints 回忆我们上面介绍数据结构MP和KF都有啥
    1. 把KeyFrame的特征点和MapPoints连起来
    2. 把MapPoints和 KeyFrame的特征点连起来
    3. 计算 代表性的一个ORB描述子
    4. 更新该MP的观测方向和距离的范围，上面说过了
4. 把MP加入当前Frame(干嘛？)
5. 在Map中加入MP

也就是 **Map**：多了初始两个关键帧，一些MPs
KF：里面的描述子对应这MP
MP：着KFs里面的相应的描述子
CurrentFrame：里面的描述子对应MP

//更新KFs之间的共视关系 Covisibility Graph Spanning Tree
然后
pKFini->UpdateConnections();

1. 获得KF的所有MP
2. 对于某个MP来说，得到该MP所有的共视的其他帧，一个循环过后，得到当前所有其他帧与当前帧的共视权重，存在map<KeyFrame*,int> 类似这样：灵魂画手

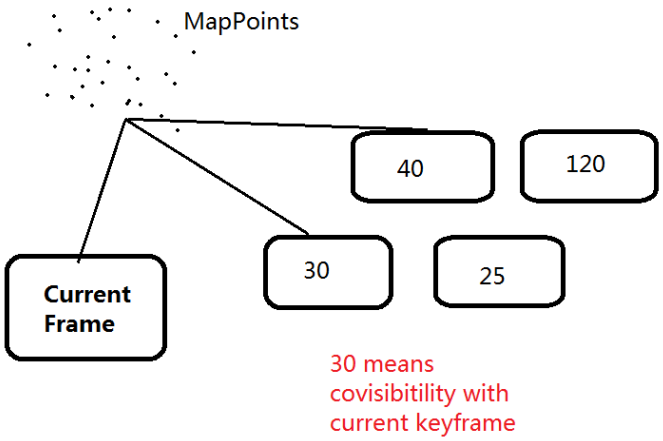


figure 1

3.遍历与当前KF有共视关系的帧，干掉小于15的，对于满足条件的添加到Covisibility Graph中去
如果没有超过阈值的，那么只加入最大权重的帧的边
4.对当前满足阈值要求的关键帧按照权重排序，方便后续处理
5.如果没有初始化过，初始化链接权重最大的边

// pMap中所有的MapPoints和关键帧做bundle adjustment优化(全局BA)

*3D-2D 最小化重投影误差 e = (u,v) - project(Tcw*Pw) \n*

Optimizer::GlobalBundleAdjustemnt(mpMap,20);
算出来的 位姿，3D点都有误差，用BA来优化

// !!!将MapPoints的中值深度归一化到1，并归一化两帧之间变换

1.计算当前帧所有点的中值深度
2.将深度归一化到1，可以这么理解，位移T =(X,Y,Z),现在 T = (X/Z , Y/Z, 1);
3.3D点的尺度也归一

//局部地图 添加KFs 局部KFs添加 KFs

1.设置 上一个KF，KFid,更新RefKF
2.Map ->添加全局的Ref Points
3.LocalMapPoints = 当前所有的MPs

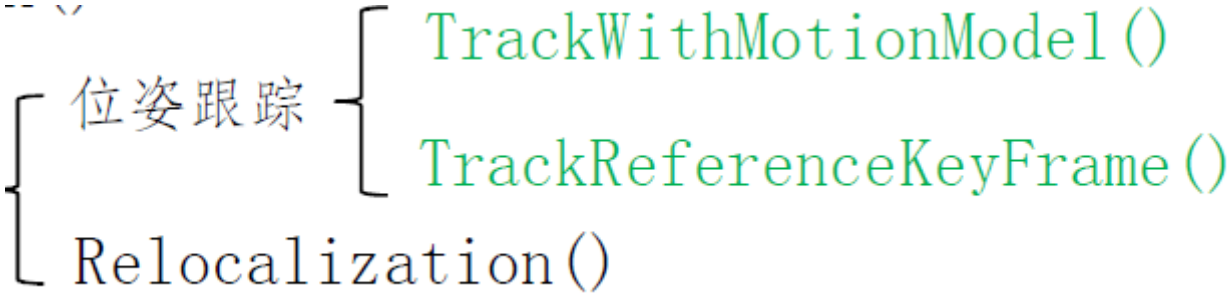
//初始化成功，好特么累

2.追踪：

1.检查更新上一帧被替换的MPs

2.追踪上一帧或者参考帧进行重定位（两两跟踪）

3种追踪方式：



TrackWithMotionModel() 如果上一帧的追踪成功，我们就用同样的速率模型计算相机位置，搜索上一帧观测到的MPs。

TrackReferenceKeyFrame() 跟踪上一帧/参考帧，通过bow加速匹配，然后通过优化得到优化后位姿

1.mCurrFrame->ComputeBoW 将当前帧的描述子转化为Bow

2.通过BoW加快当前帧与参考帧的特征点匹配 int nmatches = matcher.SearchByBoW(mpReferenceKF,mCurrentFrame,vpMapPointMatches);

SearchByBoW（KeyFrame* pKF,Frame &F, vector<MapPoint*> &vpMapPointMatches）

```
/**
 * @brief 通过词包，对关键帧的特征点进行跟踪
 * 通过bow对pKF和F中的特征点进行快速匹配（keyidea:不属于同一node的特征点直接跳过匹配）
 * 对属于同一node的特征点通过描述子距离进行匹配
 */
```

返回的是 vpMapPointMatches 是存着 Frame是否存在对于的MP的，没有就是NULL.（浪费）

3.match足够？ -> 通过优化来计算位姿

当前位姿设为初始值，通过3D-2D重投影来计算位姿 只优化Frame的Tcw，不优化MapPoints的坐标

4.剔除优化后的Outliers

5.最终结果 > 10？匹配成功

Relocalization()

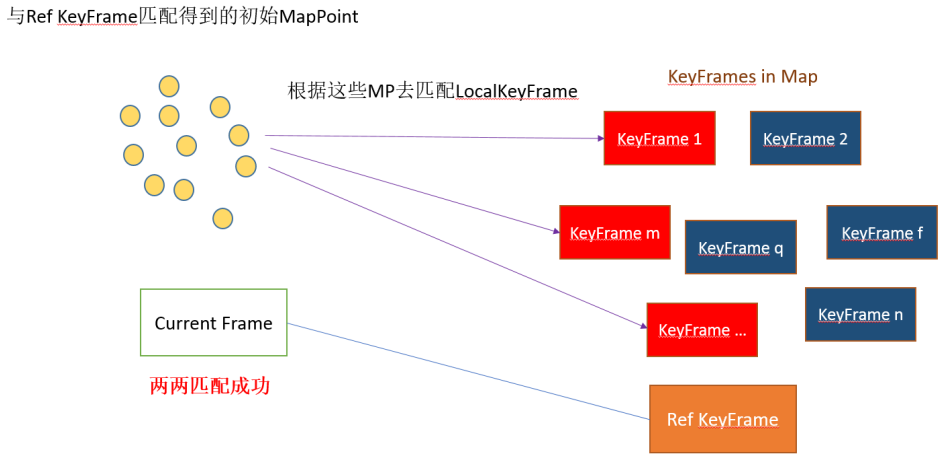
3.通过TrackLocalMap() 寻找更多的匹配，继续优化位姿 （搜索局部关键帧后收集所有的**MapPoints**，再次优化位姿）

1. 更新局部地图，包括局部关键帧和关键点

UpdateLocalMap()

UpdateLocalKeyFrames();

1.遍历当前帧的MPs,记录所有能观测到当前帧MP的关键帧 效果类似figure 1



2.添加 LocalKeyFrames! 3个策略：

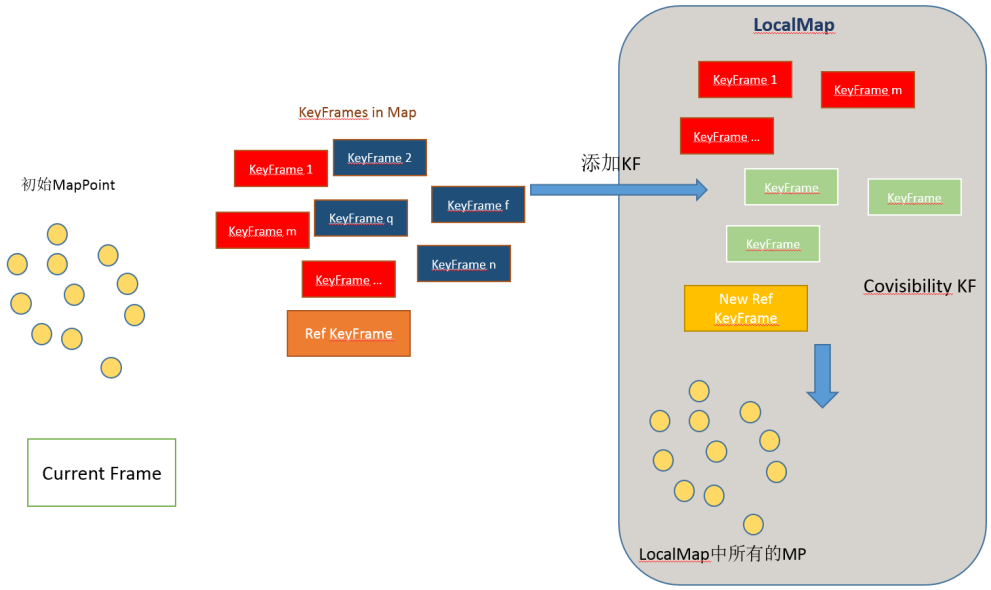
2.1.上面能看到当前MP的关键帧作为局部关键帧（显然）

2.2.与2.1得到的局部关键帧共视程度高的KF作为局部关键帧

2.3.更新参考帧，与自己共视程度高的关键帧作为参考帧

UpdateLocalPoints();

1.遍历上面拿到的LocalKeyFrames,然后把MapPoint全部加进去（防止重复添加）得到 LocalMapPoints



初始的局部地图建造好了，也就是把有可能相关的KF加进局部地图，然后把相关的KF的MP也加进去

2. 对局部MapPoints进行投影匹配

SearchLocalPoints();

1.标记当前帧的MP，不参与搜索（肯定在，不用投影匹配）

2.将所有局部MP投影到当前帧，判断是否在视野内，（在？投影匹配：再见）

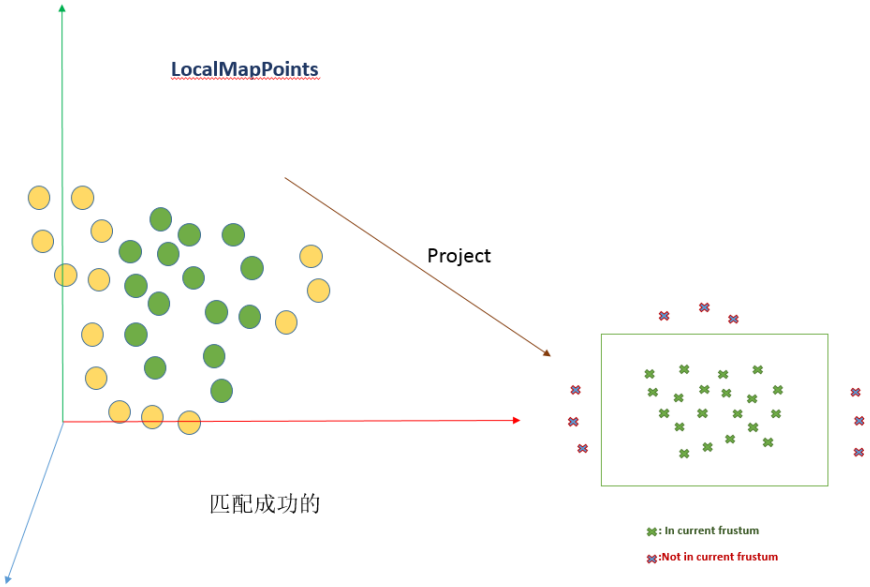
2.1 **IsInFrustum()**

把当前MP反投影回到图像内

计算MP到相机中心的距离，看在不在距离限制之内（哪来的？）

计算MP的平均视角与当前视角cos（？？？）

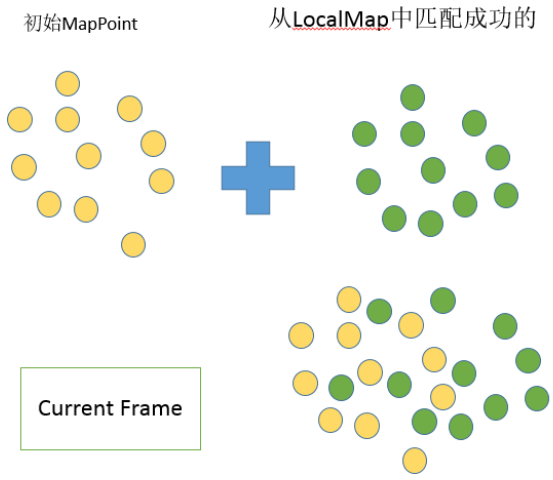
根据深度预测尺度（？？？）



2.2 **SearchByProjection(mCurrentFrame,mvpLocalMapPoints,th);** 投影匹配

遍历MP找到在 以x,y为中心,边长为2r的方形内且在[minLevel, maxLevel]的特征点

然后计算匹配，匹配上了，就建立当前帧特征与LocalMapPoint的匹配 （添加匹配成功的LocalMapPoint回到CurrentFrame中）



返回匹配数量

3. 根据匹配对估计当前帧的位姿

多了一些新的点，根据3D-2D匹配重投影误差进一步优化位姿 **PoseOptimization(Frame)**（**PS**: 两两跟踪的时候已经优化过一次位姿了 ）

4. 根据姿态剔除误匹配

更新当前帧的MapPoints被观测程度，并统计跟踪局部地图的效果

如果匹配数目大于某个值->匹配成功

1) Compute the map point projection $\mathbf{x}$ in the current frame. Discard if it lays out of the image bounds.
2) Compute the angle between the current viewing ray $\mathbf{v}$ and the map point mean viewing direction $\mathbf{n}$. Discard if $\mathbf{v} \cdot \mathbf{n} < \cos(60°)$.
3) Compute the distance d from map point to camera center. Discard if it is out of the scale invariance region of the map point $d \notin [d_{\min}, d_{\max}]$.
4) Compute the scale in the frame by the ratio $d/d_{\min}$.
5) Compare the representative descriptor $\mathbf{D}$ of the map point with the still unmatched ORB features in the frame, at the predicted scale, and near $\mathbf{x}$, and associate the map point with the best match.

The camera pose is finally optimized with all the map points found in the frame.

4.确定是否加新的KeyFrame

NeedNewKeyFrame()

1.很长时间没有插入关键帧
2.LocalMapper处于空闲状态
3.与邻近参考帧重复率低（0.9）

CreateNewKeyFrame()

1.将当前帧构造为关键帧
2.将当前关键帧设置为当前帧的参考关键帧

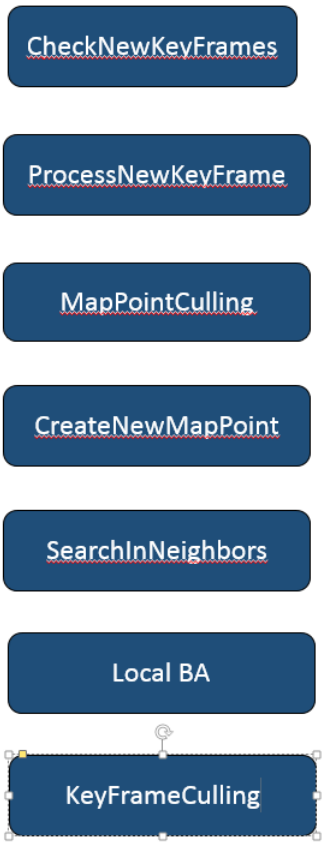
InsertKeyFrame(KF)

Tracking完毕，需要一个总结

3.localMapping：

1.从Tracking线程会插入新的KeyFrame到LocalMapping线程
放入mlNewKeyFrames中，等待处理

2.LocalMapping线程



处理KFs
ProcessNewKeyFrame()
对于当前KF

1. **CurrentKeyFrame->ComputeBow()** 计算当前KF的Bow
2. 跟踪局部地图过程中新匹配上的MapPoints和当前帧绑定
   //在TrackLocalMap中MP与当前KF进行了匹配，但是没有关联（帧中有点，点中无帧）
   2.1添加MP对当前帧的观测 pMP->AddObservation(mpCurrentKeyFrame, i);
   2.2更新该点的平均观测方向和Depth pMP->UpdateNormalAndDepth();

3.3更新该点的平均观测方向 pMP->ComputeDistinctiveDescriptors();

3.更新关键帧间的连接关系，Covisibility图和Essential图(tree)

mpCurrentKeyFrame->UpdateConnections();

4.将该关键帧插入到地图中

mpMap->AddKeyFrame(mpCurrentKeyFrame);