

第5周：ORB_SLAM2 课程课件

本周课程重点：

1 恒速模型跟踪TrackWithMotionModel

应用场景

思想

具体流程

2 跟踪丢失后的重定位Relocalization

应用场景

思想

具体流程

检测重定位候选关键帧

PnP简介

坐标系变换

地图点

倒排索引

3.局部地图跟踪TrackLocalMap

简介

mvpLocalMapPoints 示意图

mvpLocalKeyFrames示意图

换底公式

局部地图的构成以及工作流程

跟踪过程整体流程图

4. 关键帧

什么是关键帧

为什么需要关键帧

如何选择关键帧

第5周：ORB_SLAM2 课程课件

本课件是公众号 计算机视觉life 旗下课程 [《全网最详细的ORB-SLAM2精讲：原理推导+逐行代码分析》](#)（点击可跳转课程详情）的课程课件。谢谢各位学员的支持！

本课程对应的注释代码：https://github.com/electech6/ORBSLAM2_detailed_comments

由于源码注释和课件在持续更新，所以：

如视频课程中注释与上述GitHub中有不同，**以GitHub上最新源码为准。**

如视频课程中课件与本课件不同，**以本课件为准。**

本周课程重点：

1. 掌握恒速模型跟踪、重定位、局部建图跟踪原理及代码（非常重要）。
2. 理解关键帧的概念和应用。

1 恒速模型跟踪TrackWithMotionModel

对应函数Tracking::TrackWithMotionModel()，示意图如下

GN, LM 方法

待优化函数 $\|f(x_k + \alpha_k x)\|^2$

$$\text{泰勒展开: } f(x) = \frac{f(x_0)}{0!} + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_n(x)$$

牛顿法: 将目标函数二阶泰勒展开

$$\|f(x + \alpha x)\|^2 \approx \|f(x)\|^2 + J(x) \Delta x + \frac{1}{2} \Delta x^T H \Delta x$$

Jacobian: 设 \mathbf{x} 由 m 维空间, 对应 n 个自变量

$$J = \left[\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_m} \right] = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \dots & \frac{\partial f}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_1} & \dots & \frac{\partial f}{\partial x_m} \end{bmatrix}_{m \times n}$$

Hessian

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_m \partial x_1} & \frac{\partial^2 f}{\partial x_m \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_m^2} \end{bmatrix}_{n \times n}$$

问题转化为: $\underset{\Delta x}{\operatorname{argmin}} \|f(x) + J(x)\Delta x + \frac{1}{2} \Delta x^T H \Delta x\|^2$ (求极值, 对 Δx)

$$\Rightarrow \Delta x = H^{-1} J(x)$$

Pros: 梯度变化越快, 二阶导数越大, H^{-1} 越小, 步长越小

Cons: Hessian 计算量大, 甚至无法计算

GN:

对 $f(x + \alpha x)$ 一阶泰勒展开, 而不是牛顿法中的 $\|f(x + \alpha x)\|^2$

问题转化为: $\underset{\Delta x}{\operatorname{argmin}} \|f(x) + J(x)\Delta x\|^2$ 表示 $f(x)$, 即可能是

$$\|f(x) + J(x)\Delta x\|^2 = (f(x) + J(x)\Delta x)^T (f(x) + J(x)\Delta x) \quad \uparrow \text{极小值解}$$

$$= (f(x))^T + 2(f(x)^T J(x))\Delta x + \Delta x^T J(x)^T J(x)\Delta x$$

$$\text{求解} = 0: J(x)^T J(x) \Delta x = -J(x)^T f(x)$$

$$\Delta x = (J(x)^T J(x))^{-1} J(x)^T f(x)$$

$H \rightarrow$ 可能是病态的

$f(x): 2 \times 1$ 维 $\|f(x)\|^2 = f(x)^T f(x) = 1 \times 1$ 维

针对 2-6 情况

$f(x)^T: 1 \times 2$ 维 $J(x): 2 \times 6$ 维

↓

$\Delta x: 6 \times 1$ 维 (6 个自变量 update)

↓

result $\rightarrow 1 \times 1$ 维

↓

重投影维 3 维修改

x, y 3 维

(C_{low} 在点形, 最小维度)

↓

其中 Δx 是更新量

可以理解为 $f(x)$ 还不

是极小值解, 通过一阶泰勒展开

的 $f(x)$ 是趋近趋近极小值的,

Δx 是更新量

LM: 信赖域 (Trust Region) 方法

$$\min_{\Delta x_k} \frac{1}{2} \|f(x_k) + J(x_k)^T \Delta x_k\|^2, \text{ s.t. } \|\Delta x_k\| \leq \mu$$

信赖区域半径

$$L(\Delta x, \lambda) = \frac{1}{2} \|f(x) + J(x)^T \Delta x\|^2 + \frac{\lambda}{2} (\|\Delta x\|^2 - \mu^2)$$

拉格朗日函数

↓ 对 Δx 求导为 0

$$\text{跟约束} \Delta x^T J(x)^T J(x) \Delta x + 2f(x)^T J(x) \Delta x + \frac{\lambda}{2} \Delta x^T D^T D \Delta x$$

$$\Rightarrow 2J(x)^T J(x) \Delta x + 2J(x)^T f(x) + \lambda D^T D \Delta x$$

$$\Rightarrow (2J(x)^T J(x) + \lambda D^T D) \Delta x = -2J(x)^T f(x)$$

$$\Rightarrow \text{简化} \quad (H + \lambda I) \Delta x = -g$$

$$\Rightarrow \Delta x = -(J^T J + \lambda I)^{-1} J^T f(x)$$

$\mu > 0$ 能保证系数矩阵正定, 从而保证迭代的下降方向

μ 很大时, 退化为梯度下降法 $\Delta x = -\frac{1}{\mu} J^T f(x)$

μ 很小时, 退化为 G-N, 从而使得接近解时快速收敛

$$A_0 = J(x_0)^T J(x_0)$$

$\mu_0 = T \cdot \max_i \{A_{0,ii}\}$ 对角线上最大元素

用 $P = \frac{F(x_{old}) - F(x)}{L(x) - L(x)}$ 判断近似的好坏

分子是实际函数迭代下降的值, 分母是近似模型下降的

$$L(\Delta x) - L(0) = F(x_0) + 2f(x)^T J(x) \Delta x + \Delta x^T J(x)^T J(x) \Delta x - F(x_0)$$

$$\downarrow \frac{\|f(x)\|^2}{\|f(x)\|^2}$$

$$= 2f(x)^T J(x) \Delta x + \Delta x^T J(x)^T J(x) \Delta x$$

$\uparrow P > 3/4 \rightarrow \mu = 1/3 \mu \rightarrow G-N$

$\downarrow P < 1/4 \rightarrow \mu = 2\mu \rightarrow$ 并减小步长.

$$\mu = \mu * \max\left\{\frac{1}{3}, 1 - (2P - 1)^3\right\} (1.999 \rightarrow L - \mu)$$

$$\text{else } \mu = \mu * v; v = 2 * v \quad (v \text{ origin} = 2)$$

Pros

Newton:

二阶导数决定了根据梯度变化, 改变更新大小

Hessian 不好算

GN:

$J(x)^T J(x)$ 省略了 H 矩阵. 但 $J^T J$ 可能是病态的, 需矩阵

LM:

避免系数矩阵病态问题 失了一些收敛速度

Kalman Filter

应用场景:
线性高斯系统

卡尔曼公式理解:

- ① 使用上一次的最优结果预测当前的值。
- ② 使用观测值修正当前值，得到最优结果。

线性系统:

$$\begin{aligned} \dot{x}_t &= A_t x_{t-1} + u_t + w_t \\ z_t &= C_t x_t + v_t \\ w_t &\sim N(0, Q) \\ v_t &\sim N(0, R) \end{aligned}$$

五步公式:

预测

$$\begin{aligned} \hat{x}_t &= F \hat{x}_{t-1} + B u_{t-1} \\ \hat{P}_t &= F \hat{P}_{t-1} F^T + Q \end{aligned}$$

注: \hat{x}_t 先验。

① 先验估计

$$\hat{x}_t = \hat{F} \hat{x}_{t-1} + B u_t + \text{过程噪声}$$

② 先验估计协方差。
补充: $Cov(Ax+b) = \sum_i [(Ax_i + b) - (Ax + b)] (Ax_i + b - Ax)^T$

$$\begin{aligned} &= \sum_i [A(x_i - \mu) + b - A\mu - b] (A(x_i - \mu) + b - A\mu - b)^T \\ &= A \sum_i (x_i - \mu)(x_i - \mu)^T A^T \\ &= A \text{Cov}(x) A^T \end{aligned}$$

$$\text{故 } \Rightarrow \hat{P}_t = F \hat{P}_{t-1} F^T + Q$$

③ 测量方程。

$$z_t = H x_t + v_t \quad (z_t \text{ 与 } x_t \text{ 维度不一定相同})$$

④ 修正估计
观测值与估计算法的误差。

$$\begin{aligned} \hat{x}_t &= \hat{x}_t + K_t (z_t - H \hat{x}_t) \\ &= K_t z_t + (I - K_t H) \hat{x}_t \end{aligned}$$

⑤ 更新卡尔曼增益。

$$K_t = \frac{\hat{P}_t H^T}{H \hat{P}_t H^T + R} \quad \text{与 } Q, R \text{ 相关}, \hat{P}_t \text{ 与 } R \text{ 相关}$$

⑥ 更新后验估计协方差。

$$\hat{P}_t = (I - K_t H) \hat{P}_t$$

【公式推导】 (根据“十四讲”推导, 参数有所变化。
 $\rightarrow k, H_t \rightarrow C_k$)

$$P(x_k | x_0, u_{1:k}, z_{1:k}) \propto P(z_k | x_0, u_{1:k}, z_{1:k})$$

后验。

$$N(\hat{x}_k, \hat{P}_k) = \eta N(C_k x_k, R) \cdot N(\hat{x}_k, \hat{P}_k)$$

$$\text{高斯正态分布: } \frac{1}{\sqrt{(2\pi)^n \det(R)}} \exp\left(-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$$N(\hat{x}_k, \hat{P}_k) = \eta N(C_k x_k, R) \cdot N(\hat{x}_k, \hat{P}_k)$$

指数部分展开:

$$(x_k - \hat{x}_k)^T \hat{P}_k^{-1} (x_k - \hat{x}_k) = (x_k - C_k x_k)^T R^{-1} (x_k - C_k x_k) + (\hat{x}_k - C_k x_k)^T R^{-1} (x_k - \hat{x}_k)$$

取二次项:

$$\hat{P}_k^{-1} = C_k^T R^{-1} C_k + \hat{P}_k^V$$

左右左乘 \hat{P}_k^V 得:

$$\begin{aligned} I &= \hat{P}_k^V C_k^T R^{-1} C_k + \hat{P}_k^V \hat{P}_k^V \\ &\text{令此为 } K \\ &= K C_k + \hat{P}_k^V \hat{P}_k^V \end{aligned}$$

$$\Rightarrow \hat{P}_k^V = (I - K C_k) \hat{P}_k^V$$

$$\Rightarrow \hat{P}_k^V \hat{P}_k^V = I - K C_k$$

取一灰项:

$$\cancel{2 \hat{P}_k^V \hat{P}_k^V x_k} = \cancel{2 z_k^T R^{-1} C_k x_k} - \cancel{2 \hat{x}_k^T \hat{P}_k^V x_k}$$

$$\hat{x}_k^T \hat{P}_k^V = 2 z_k^T R^{-1} C_k + \hat{x}_k^T \hat{P}_k^V$$

取转换:

$$\hat{P}_k^V \hat{x}_k = C_k^T R^{-1} Z_k + \hat{P}_k^V \hat{x}_k$$

同乘 \hat{P}_k^V 得

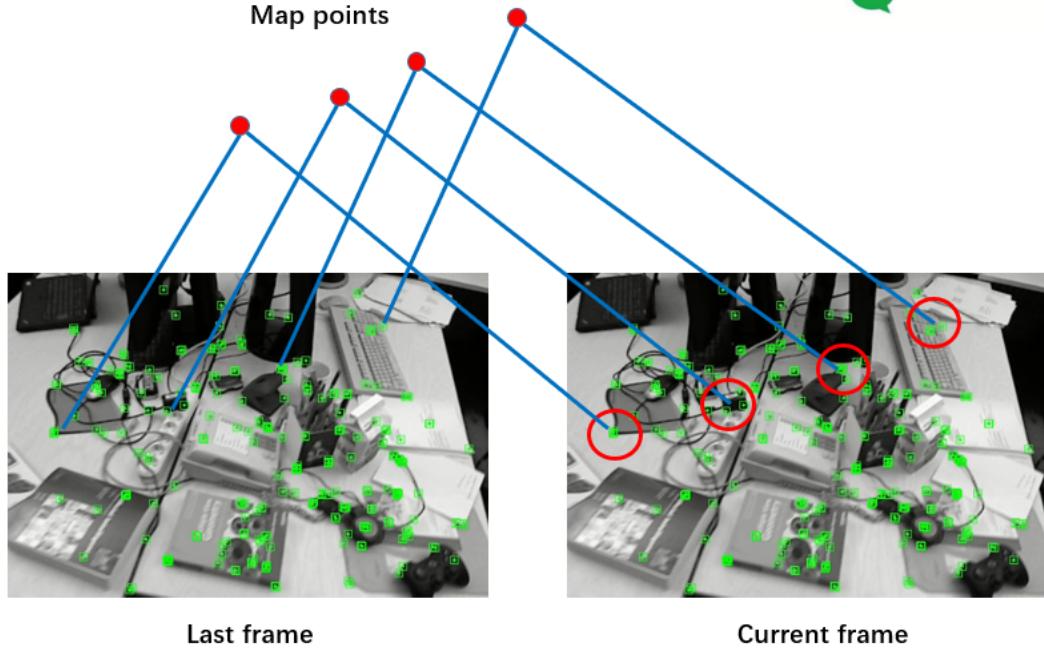
$$\begin{aligned} \hat{x}_k &= \hat{P}_k^V C_k^T R^{-1} Z_k + \hat{P}_k^V \hat{P}_k^V \hat{x}_k \\ &= K z_k + \hat{P}_k^V \hat{x}_k \\ &= K z_k + (I - K C_k) \hat{x}_k \end{aligned}$$

$$K = \hat{P}_k^V C_k^T R^{-1} = (C_k^T R^{-1} C_k + \hat{P}_k^V)^{-1} C_k^T R^{-1}$$

用SMW恒等式中 $(D + CAB)^{-1} CA \equiv D^{-1} C (A^{-1} + BD^{-1} C)^{-1}$

推得:

$$K = \hat{P}_k^V C_k^T (R + C_k \hat{P}_k^V C_k^T)^{-1}$$



应用场景

大部分时间都用这个跟踪，只利用到了上一帧的信息。

1. 用恒速模型先估计一个初始位姿
2. 用该位姿进行投影匹配 SearchByProjection，候选点来自GetFeaturesInArea，未使用BoW
3. BA优化（仅优化位姿），提供比较粗糙的位姿

思想

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。

移动模式跟踪 跟踪前后两帧 得到 变换矩阵。

上一帧的地图3d点反投影到当前帧图像像素坐标上，在不同尺度下不同的搜索半径内，做描述子匹配 搜索 可以加快匹配。

在投影点附近根据描述子距离进行匹配（需要>20对匹配，否则匀速模型跟踪失败，运动变化太大时会出现这种情况），然后以运动模型预测的位姿为初值，优化当前位姿，

优化完成后再剔除外点，若剩余的匹配依然>=10对，则跟踪成功，否则跟踪失败，需要Relocalization

$$\begin{aligned}
 &\text{当前帧 } R_{cw}, t_{cw} \text{ 取出后, } R_{cw}^{-1} = R_{cw}^T \\
 &t_{cw}^{-1} \leftarrow \begin{bmatrix} R_{cw} & t_{cw} \end{bmatrix} \begin{bmatrix} R_{cw}^T & -R_{cw}^T t_{cw} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ 0 & 1 \end{bmatrix} \\
 &\text{上一帧 } t_{lw}, R_{lw} \text{ 取出, 在 } t_{lc} \\
 &T_{lc} = T_{lw} + T_{wc} = \begin{bmatrix} R_{lw} & t_{lw} \end{bmatrix} \begin{bmatrix} R_{wc}^T & -R_{wc}^T t_{wc} \\ 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} R_{lw} R_{wc}^T & -R_{lw} R_{wc}^T t_{wc} + t_{lw} \\ 0 & 1 \end{bmatrix} \rightarrow t_{lc} = R_{lw} t_{wc} + t_{lw}
 \end{aligned}$$

具体流程

1：创建 ORB特征点匹配器 最小距离 $< 0.9 * \text{次小距离}$ 匹配成功

2：更新上一帧的位姿和地图点

单目：只计算了上一帧世界坐标系位姿就退出了。 $T_{lr} * T_{rw} = T_{lw}$

双目或rgbd相机：根据上一帧有效的深度值产生为上一帧生成新的临时地图点，之所以说是“临时”因为这些新加的地图点不加入到Map中，只是为了当前帧间跟踪更稳定，用完会删除，过河拆桥啊！

3：使用当前的运动速度(之前前后两帧位姿变换)和上一帧的位姿来初始化 当前帧的位姿 R, t

t_{LC} 表示从 camera 生标系到 last frame
生标系的变换，相当于把本在 cam 生标系上的生标变换到 last frame 生标系下。

4: 在当前帧和上一帧之间搜索匹配点 (matcher.SearchByProjection)

通过投影(使用当前帧的位姿 R, t)，对上一帧的特征点(地图点)进行跟踪。

上一帧3d点投影到当前坐标系下，在该2d点半径 th 范围内搜索可以匹配的匹配点

遍历可以匹配的点，计算描述子距离，记录最小的匹配距离，小于阈值的，再记录匹配点特征方向差值

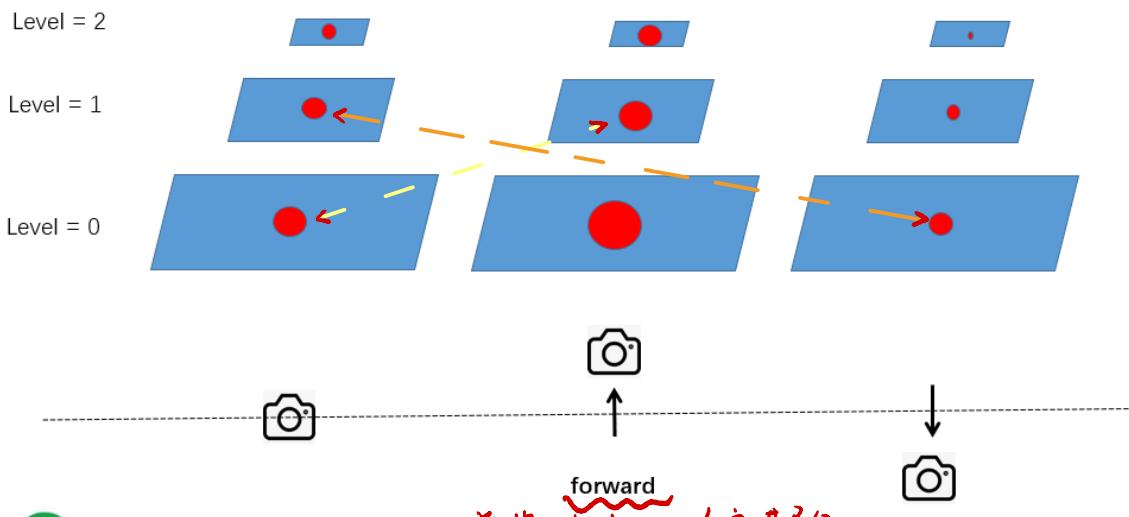
进行方向验证，剔除方向差直方图统计中，方向差值数量少的点对，保留前三个数量多的点对。

5: 如果找到的匹配点对如果少于20，则扩大搜索半径 $th=2*th$, 使用SearchByProjection()再次进行搜索。

6: 使用匹配点对对当前帧的位姿进行优化 G2O图优化

7: 如果2d-3d匹配效果差，被标记为外点，则当前帧2d点对于的3d点设置为空，留着以后再优化

8: 根据内点的匹配数量，判断 跟踪上一帧是否成功。



计算机视觉life

2 跟踪丢失后的重定位Relocalization

对应函数Tracking::Relocalization()

应用场景

跟踪丢失的时候使用，很少使用。利用到了相似候选帧的信息。

1. 用BoW先找到与该帧相似的候选关键帧 (函数DetectRelocalizationCandidates)
2. 遍历候选关键帧，用SearchByBoW快速匹配，
3. 匹配点足够的情况下用EPnP 计算位姿并取出其中内点做BA优化 (仅优化位姿) ，
4. 如果优化完内点较少，通过关键帧投影生成新的匹配 (函数SearchByProjection) ，
5. 对匹配结果再做BA优化 (仅优化位姿) 。

思想

当TrackWithMotionModel 和 TrackReferenceKeyFrame 都没有跟踪成功，位置丢失后，需要在之前的关键帧中匹配最相近的关键帧，进而求出位姿信息。

使用当前帧的BoW特征映射，在关键帧数据库中寻找相似的候选关键帧，因为这里没有好的初始位姿信息，需要使用传统的3D-2D匹配点的EPnP算法来求解一个初始位姿，之后再使用最小化重投影误差来优化更新位姿。

具体流程

1: 计算当前帧的BoW向量和Feature向量

2: 在关键帧数据库中找到与当前帧相似的候选关键帧组

3: 创建 ORB特征点匹配器 最小距离 $< 0.75 * \text{次小距离}$ 匹配成功。

ORBmatcher matcher(0.75,true);

4: 遍历每一个候选关键帧使用BOW特征向量加速匹配，匹配太少的去掉，选择符合要求的候选关键帧用其地图点为其创建pnp优化器

5: 使用PnPsolver 位姿变换求解器,增加3d-2d匹配点

6点直接线性变换DLT,后使用QR分解得到 R,t, 或者使用(P3P), 3点平面匹配算法求解。

这里会结合 Ransac 随采样序列一致性算法, 来提高求解的鲁棒性。

6: EPnP算法迭代估计姿态作为当前帧的初始位姿, 使用最小化重投影误差BA算法来优化位姿

7: 如果优化时记录的匹配点对内点数量少于50, 想办法再增加匹配点数量: 通过投影的方式对之前未匹配的点进行3D-2D匹配, 又给了一次重新做人机会

8: 如果新增的数量加上之前的匹配点数量 大于50, 再次使用位姿优化算法进行优化

9: 如果上面优化后的内点数量还比较少, 还想挽留一下, 就缩小搜索窗口重新投影匹配 (比之前使用更多的地图点了), 如果最后匹配次数大于50, 就认为是可以勉强扶起来的阿斗, 再给BA位优化一次。否则, 放弃了 (真的已经仁至义尽了!)

10: 如果经过上面一系列的挽救操作, 内点数量 大于等于50, 则重定位成功。

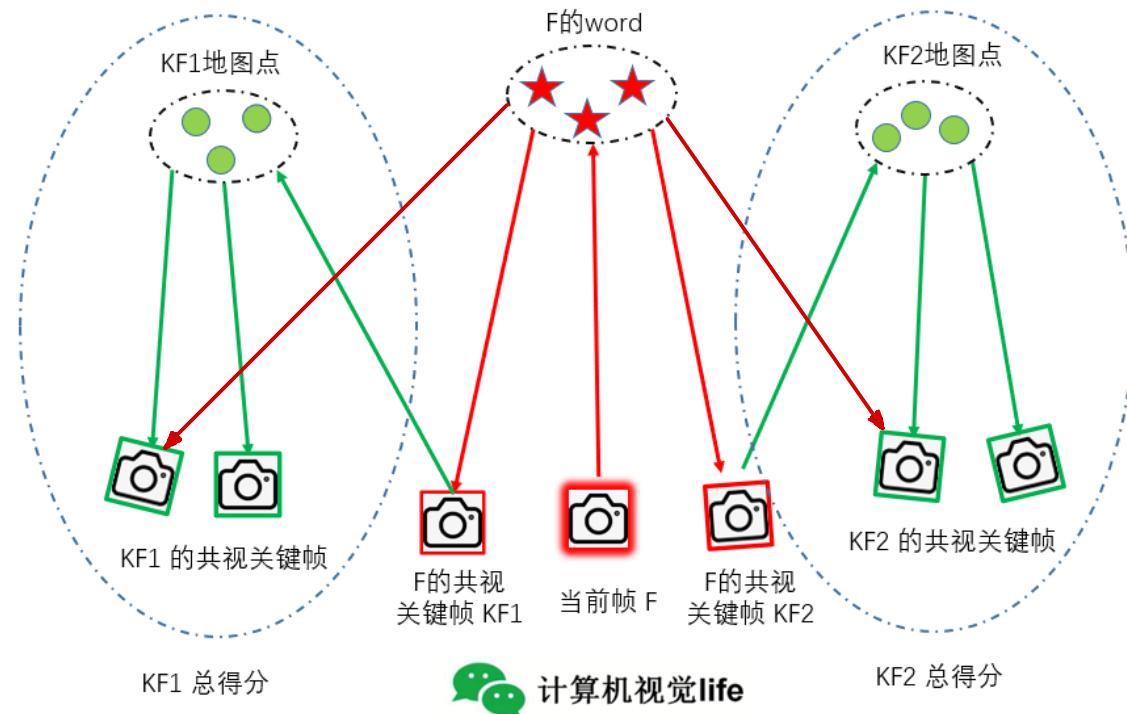
选用了视的且在候选关键帧的几帧
因为一组, 算 AccScore, 取总得分
大的那组。

理解: reloc 对, 要找到相似的 (即临
近的, 场景变化不大的) 帧。
这些帧会有多个, 且应该是相邻的,
存在关联关系。

② 分组另一个原因: 以组为单位
来判断筛选去 keyframe, 加速
筛选

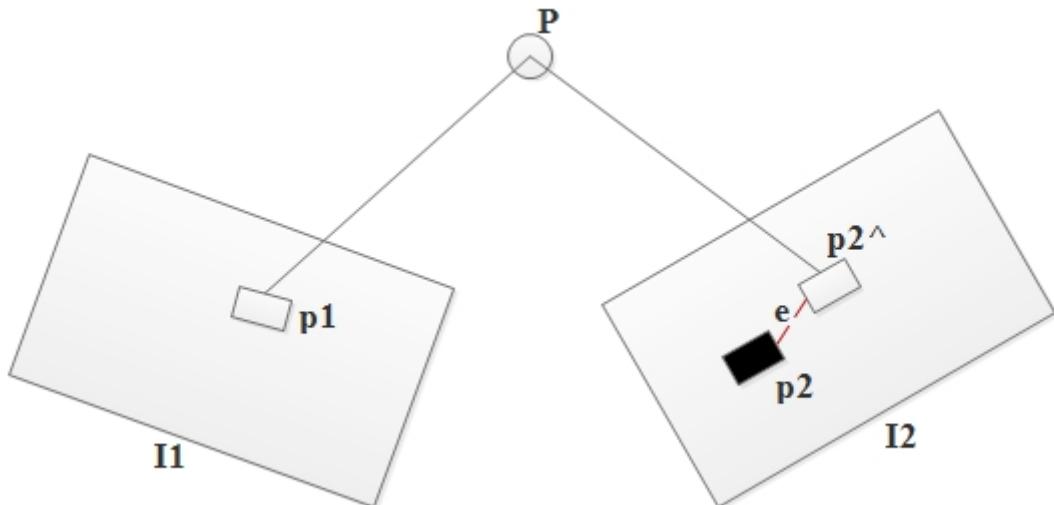
检测重定位候选关键帧

对应函数 DetectRelocalizationCandidates



PnP简介

PnP投影示意图



Tracking 流程：

初始化 Monocular Initialization

判断是否又跟踪 mbOnly Tracking

相比起右侧 Only tracking
多了一步数 mbVO

在恒速模型和参考关键
帧前判断，若为 False

则认为此帧匹配了足够
的地图点

用的是重投影后 GetFeatureInArea 得到的
匹配关系 恒速模型可用 mVelocity ✓
TrackWithMotionModel

用的BoW + featureVec 找的匹配关系
or mVelocity × 用参考关键帧
TrackReferenceKeyFrame

若都不行，重定位 Relocalization
用 word 应推，构成组
求 BoW 评分

局部地图跟踪
Track Local Map
也用的重投影
SearchByProjection.

对于 Only Tracking，只有
mbVO = false 相当于有足够的
MapPoints，才可以
执行 TLM

参考

ORBmatcher::SearchByProjection

为什么要用EPnP?

论文

EPnP: Accurate Non-Iterative O(n) Solution to the PnP Problem

- 计算复杂度低，是 O(n) 的
- 只需要4个及以上的控制点，适用于平面、非平面
- 精度比较高

坐标系变换

疑问： $t_{lc} = R_{lw} * t_{wc} + t_{lw}$; 原因?

$$\begin{aligned}
T_{cw} &= \begin{bmatrix} R_{cw} & t_{cw} \\ 0^T & 1 \end{bmatrix} \\
T_{cw}^{-1} &= \begin{bmatrix} R_{cw}^T & -R_{cw}^T t_{cw} \\ 0^T & 1 \end{bmatrix} \\
T_{lw} &= \begin{bmatrix} R_{lw} & t_{lw} \\ 0^T & 1 \end{bmatrix} \\
T_{lc} &= T_{lw} * T_{cw}^{-1} \\
&= \begin{bmatrix} R_{lw} & t_{lw} \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} R_{cw}^T & -R_{cw}^T t_{cw} \\ 0^T & 1 \end{bmatrix} \\
&= \begin{bmatrix} R_{lw} R_{cw}^T & -R_{lw} R_{cw}^T t_{cw} + t_{lw} \\ 0^T & 1 \end{bmatrix} \\
t_{lc} &= -R_{lw} R_{cw}^T t_{cw} + t_{lw}
\end{aligned}$$

地图点

地图点是三维点，来自真实世界的三维物体，有唯一的id。不同帧里的特征点可能对应三维空间中同一个三维点。

特征点是二维点，是特征提取的点，大部分二维点在三维空间中没有对应地图点

关于生成地图点

主要有2个地方：

- 1、初始化时 前两帧匹配生成地图点
- 2、local mapping里共视关键帧之间用 LocalMapping::CreateNewMapPoints() 生成地图点
- 3、Tracking::UpdateLastFrame() 和 Tracking::CreateNewKeyFrame() 中为双目和RGB-D生成了新的临时地图点，单目不生成

倒排索引

// mvInvertedFile[i]表示包含了第i个word id的所有关键帧

```
std::vector<list<KeyFrame*>> mvInvertedFile;
```

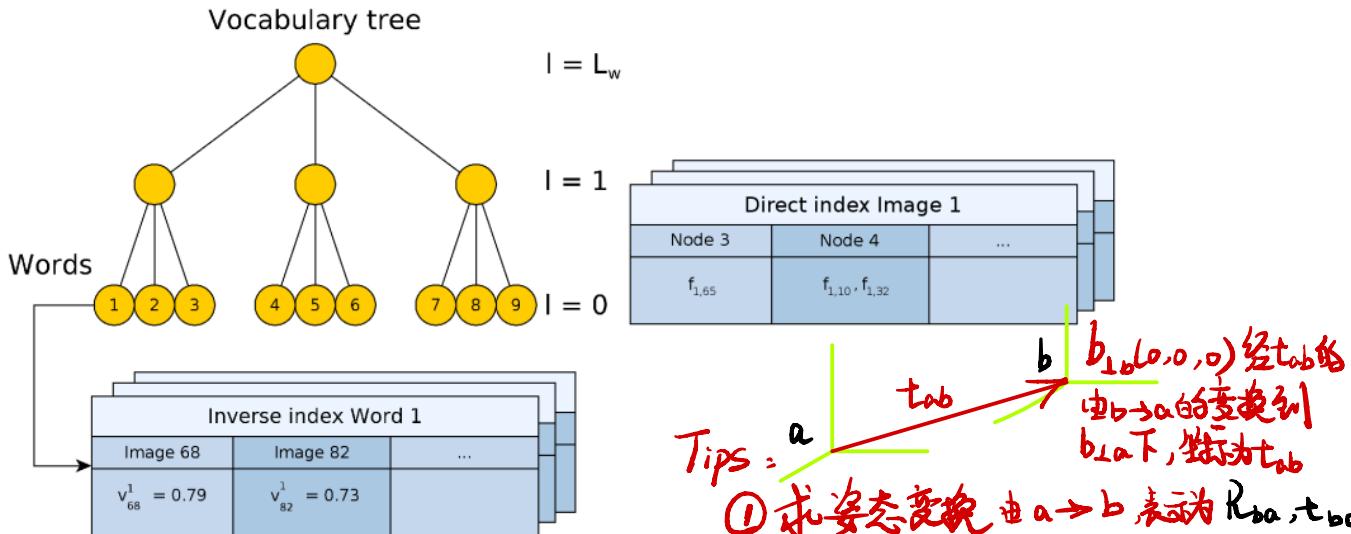
直接索引表direct index 和 逆向（倒排）索引表inverse index。

直接索引表 是以 图像为基础的，存储图像中的特征以及和该图像相关联的节点 node (vocabulary tree 的某一层级) 。

直接索引的优势：

通过同一个节点下特征点描述子对应关系 加速几何关系验证，比如用BOW快速匹配

逆向（倒排）索引表 是以 Word 为基础，存储Word的权重以及该Word出现在哪个图像里。这对于查询数据库非常方便，因为它可以很方便对比哪些图像有共同的Word



3. 局部地图跟踪 TrackLocalMap

对应函数 tracking::TrackLocalMap

简介 重定位后点的 GetFeatureInArea

应用场景: $\text{radius}_{\text{增大}}$

前面3种跟踪方式得到当前帧地图点后的后处理，每次跟踪都使用。前提是必须知道当前帧的位姿和地图点（尽管不准确），利用到了当前帧的两级共视关键帧的信息，使得位姿更加准确。

具体步骤：

- 首先根据前面得到的当前帧的地图点来寻找能观测到当前帧的一级共视关键帧，将这些一级共视关键帧的二级关键共视帧、子关键帧、父关键帧一起作为局部关键帧；
- 取出上述局部关键帧中所有的地图点作为局部地图点；
- 将局部地图点投影到当前帧，去掉不在视野内的无效的地图点，剩下的局部地图点投影到当前帧进行匹配（函数SearchByProjection）
此步骤中，会剔除掉在可靠距离范围之外的地图点（计算地图点与光心的距离）
- 对匹配结果再做BA优化（仅优化位姿）

当前帧: mCurrentFrame (当前帧是普通帧)

Max Distance 设计为 $1.2 * dist * \text{图层深度} / \text{Level Scale Factor}$

由 mvMapPoint

计算 dist 得到的，即本是 CurrentFrame 里的图点。

这点非常巧妙！

参考关键帧：与当前帧共视程度最高的关键帧作为参考关键帧，mCurrentFrame.mpReferenceKF

在 KeyFrame::UpdateConnections() 里确定关键帧的父子关系（当前帧必须是关键帧） 我们拿到图层间确定的尺度信息，但有相对信息。

父关键帧：和当前关键帧共视程度最高的关键帧

子关键帧：是上述父关键帧的子关键帧

对于高图层（即指尖），dist 自然会小；
加上 scale，即能恢复深度。

dist \propto scale

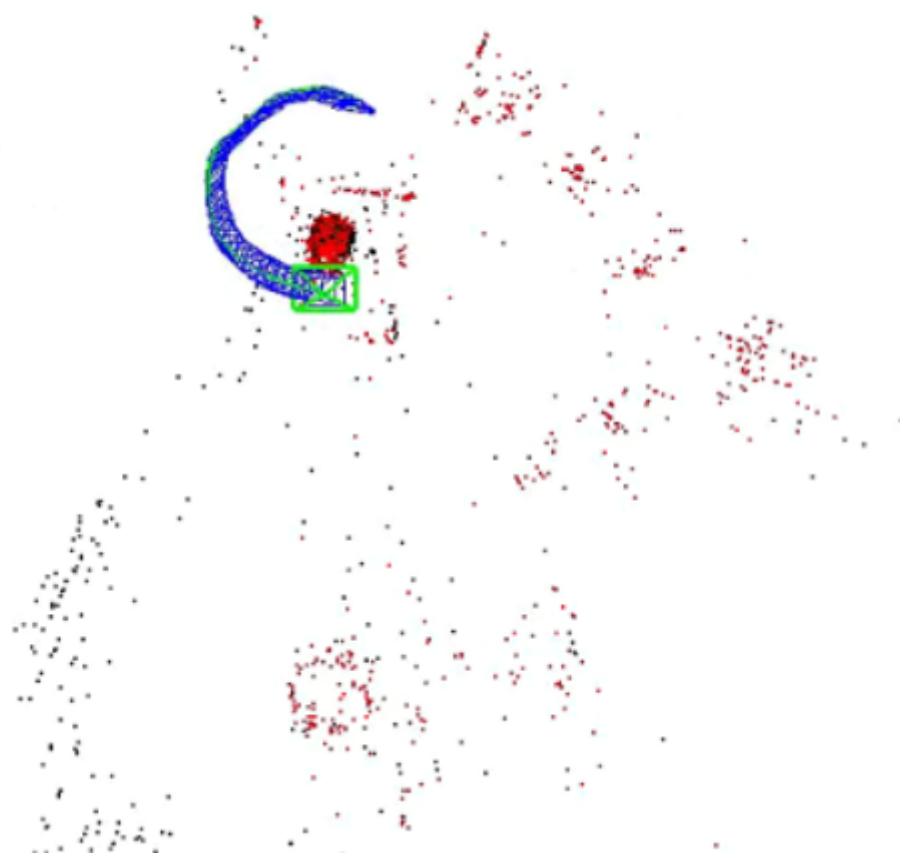
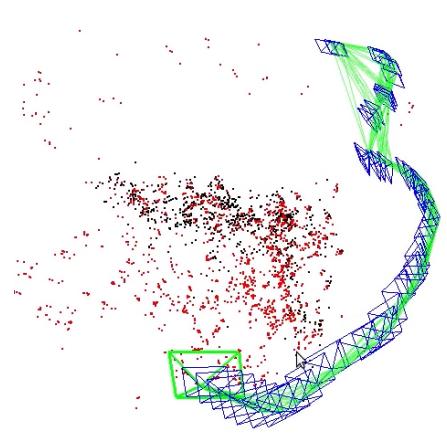
mvpLocalMapPoints 示意图

就是下图中红色地图点，用来在 TrackLocalMap 里跟踪当前帧

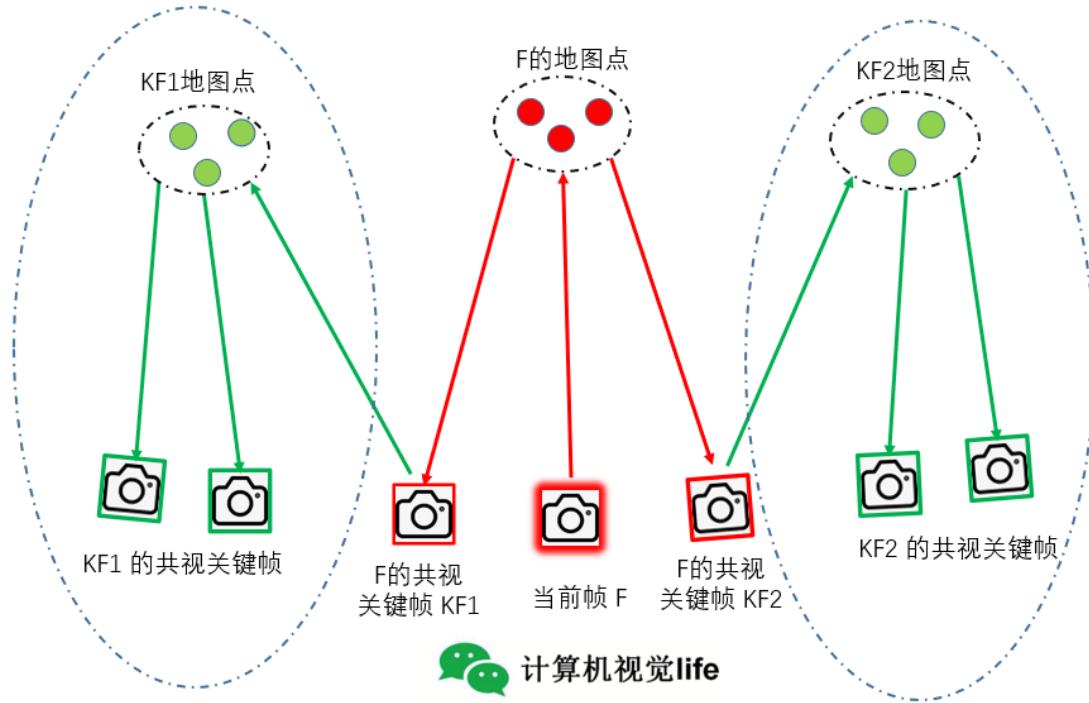
```

Tracking: BEGIN! ID = 188
Tracking: Begin TrackWithMotionModel! frame-id = 188
Tracking: Finsh TrackWithMotionModel! frame-id = 188 matches = 73
Tracking: Begin TrackLocalMap! frame-id = 188
Tracking: Finsh TrackLocalMap! frame-id = 188, matches = 105
Tracking: Need create keyframe ID = 188

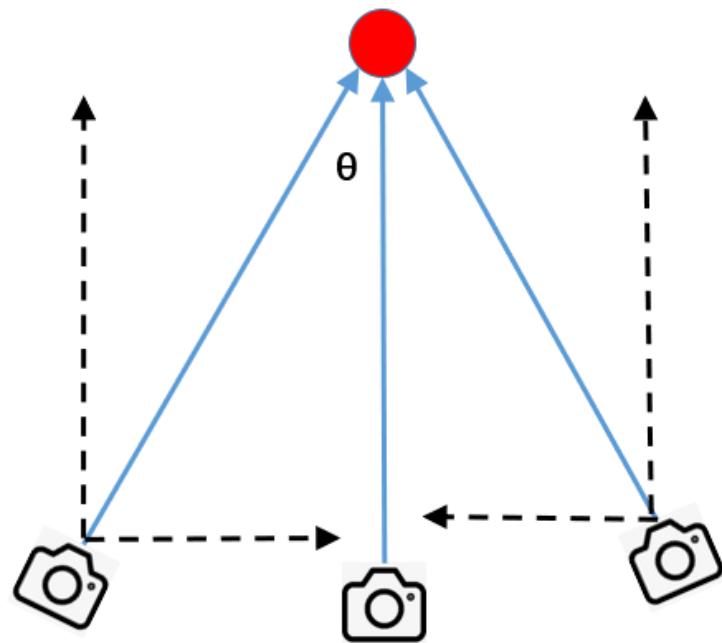
```



mvpLocalKeyFrames示意图



地图点的平均观测方向



换底公式

$$\log_a b = \frac{\log_c b}{\log_c a}.$$

$$1.2^m = \frac{d_{max}}{d}$$

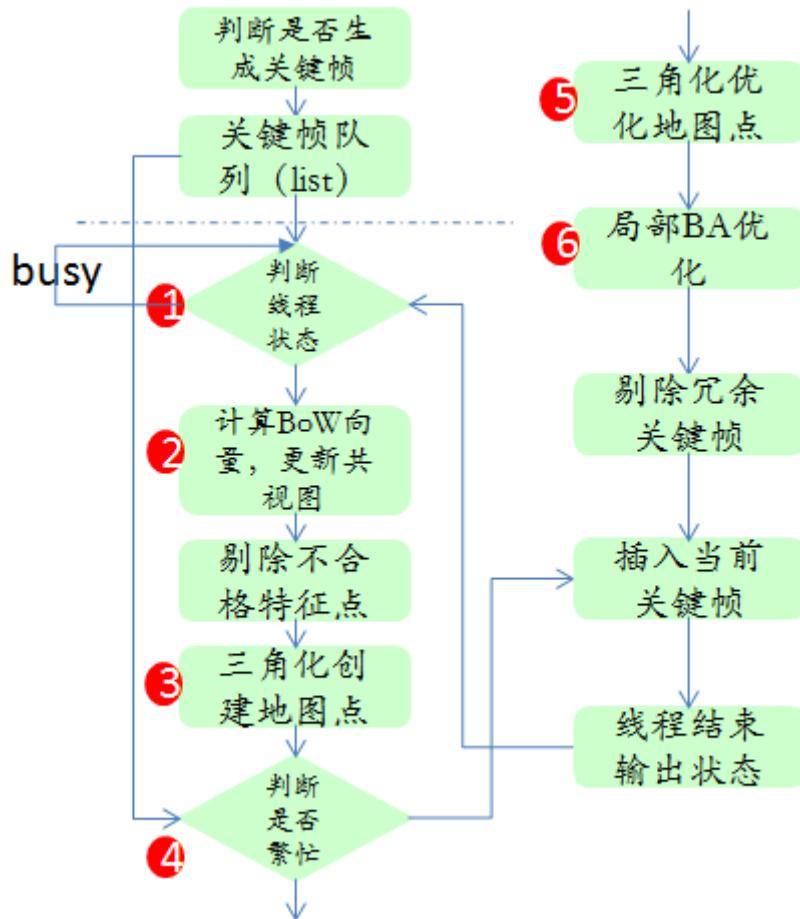
$$m = \log_{1.2} \frac{d_{max}}{d}$$

$$= \frac{\log(d_{max}/d)}{\log 1.2}$$

局部地图的构成以及工作流程

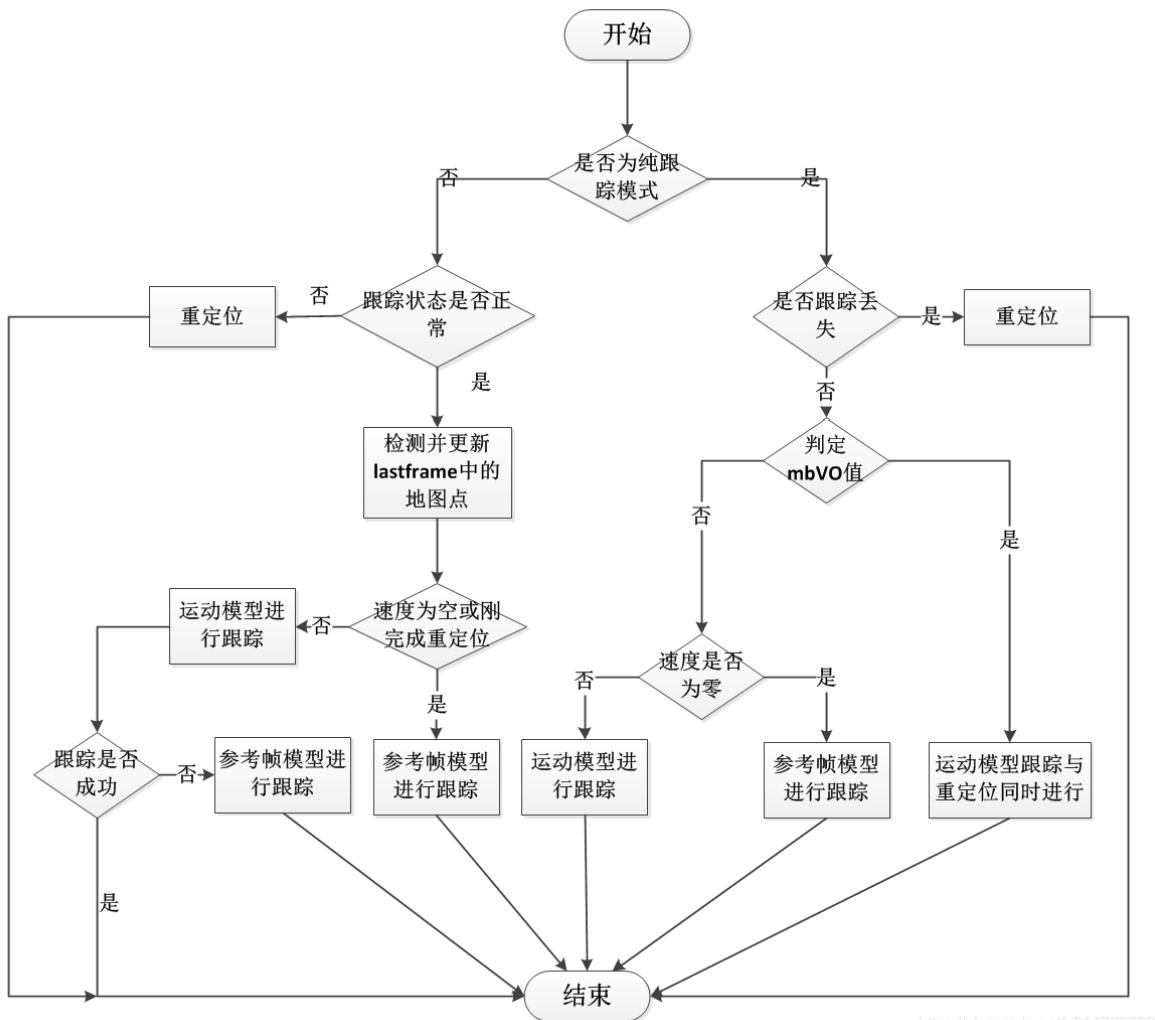


流程：



[参考](#)

跟踪过程整体流程图



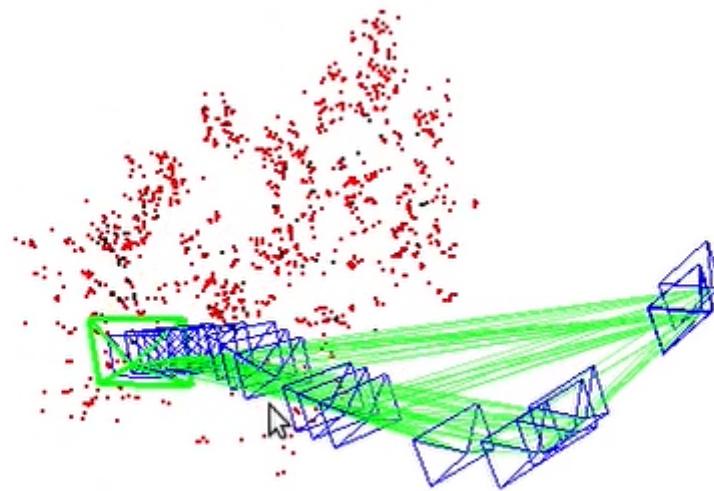
https://blog.csdn.net/u014709760

参考：https://blog.csdn.net/u014709760/article/details/89465842#21_Tracking_6

4. 关键帧

什么是关键帧

通俗来说，关键帧就是几帧普通帧里面具有代表性的一帧。



为什么需要关键帧

- 相近帧之间信息冗余度很高，关键帧是取局部相近帧中最有代表性的一帧，可以降低信息冗余度。举例来说，摄像头放在原处不动，普通帧还是要记录的，但关键帧不会增加。
- 关键帧选择时还会对图片质量、特征点质量等进行考察，在Bundle Fusion、RKD SLAM等RGB-D SLAM相关方案中常常用普通帧的深度投影到关键帧上进行深度图优化，一定程度上关键帧是普通帧滤波和优化的结果，防止无用的或错误的信息进入优化过程而破坏定位建图的准确性。
- 如果所有帧全部参与计算，不仅浪费了算力，对内存也是极大的考验，这一点在前端vo中表现不明显，但在后端优化里是一个大问题，所以关键帧主要作用是面向后端优化的算力与精度的折中，使得有限的计算资源能够用在刀刃上，保证系统的平稳运行。假如你放松ORB_SLAM2 关键帧选择条件，大量产生的关键帧不仅耗计算资源，还会导致local mapping 计算不过来，出现误差累积

如何选择关键帧

选择关键帧主要从关键帧自身和关键帧与其他关键帧的关系2方面来考虑。

- 关键帧自身质量要好**，例如不能是非常模糊的图像、特征点数量要充足、特征点分布要尽量均匀等等；
- 关键帧与其他关键帧之间的关系，需要和局部地图中的其他关键帧有一定的共视关系但又不能重复度太高，以达到既存在约束，又尽量少的信息冗余的效果。

选取的指标主要有：

(1) 距离上一关键帧的帧数是否足够多 **(时间)**。比如我每隔固定帧数选择一个关键帧，这样编程简单但效果不好。比如运动很慢的时候，就会选择大量相似的关键帧，冗余，运动快的时候又丢失了很多重要的帧。

(2) 距离最近关键帧的距离是否足够远 **(空间)** /运动

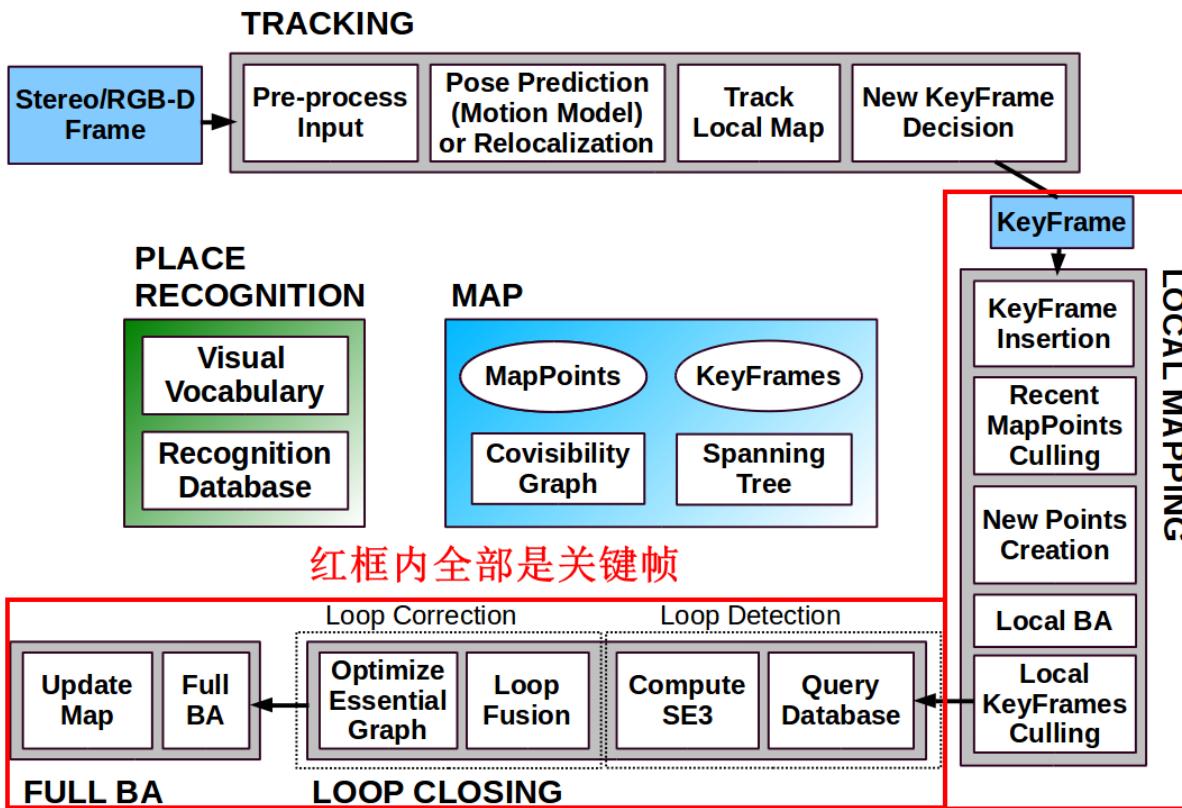
比如相邻帧根据pose计算运动的相对大小，可以是位移也可以是旋转或者两个都考虑，运动足够大（超过一定阈值）就新建一个关键帧，这种方法比第一种好。但问题是如果对着同一个物体来回扫就会出现大量相似关键帧。

✓ 跟踪局部地图质量 **(共视特征点数目)** **用共视地图点选择关键帧 = 无视即新信息**

记录当前视角下跟踪的特征点数或者比例，当相机离开当前场景时（双目或比例明显降低）才会新建关键帧，避免了第2种方法的问题。缺点是数据结构和逻辑比较复杂。

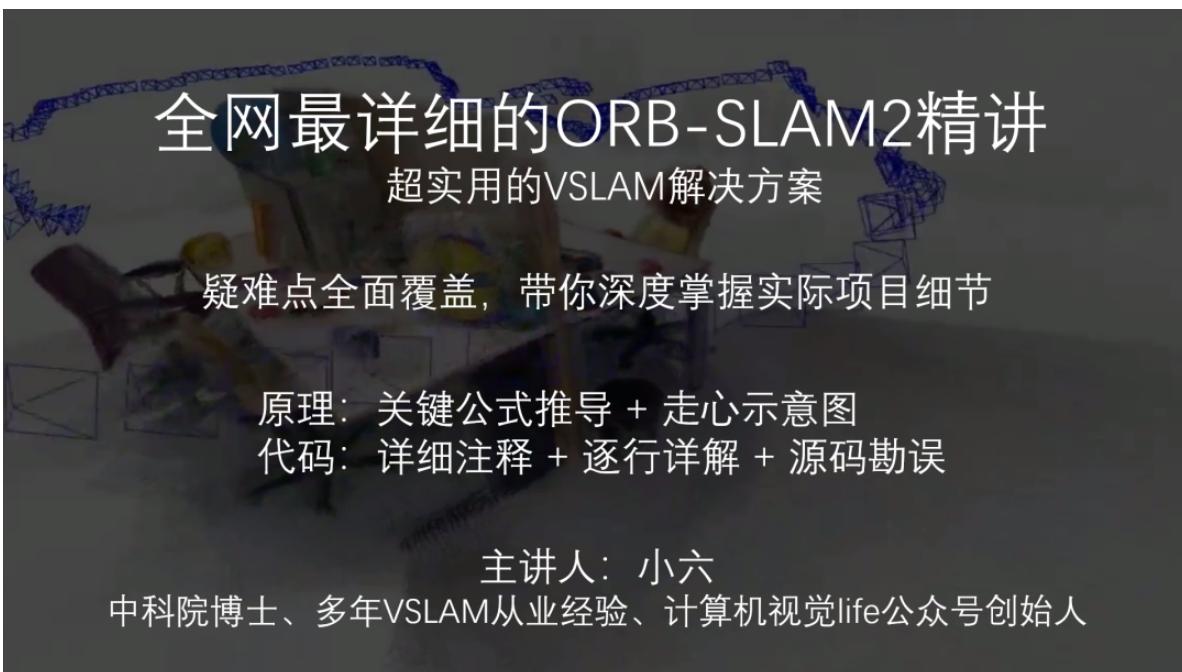
用mnMatchesInliers 和 nRefMatches * thRatio 比较

参考帧地图点被观测次数超过3次的点 → 越大说明越宽松



在关键帧的运用上，我认为orbslam2做的非常好，跟踪线程选择关键帧标准较宽松，局部建图线程再根据共视冗余度进行剔除，尤其是在回环检测中使用了以关键帧为代表的帧“簇”的概念，回环筛选中有一步将关键帧前后10帧为一组，计算组内总分，以最高分的组的0.75为阈值，滤除一些组，再在剩下的组内各自找最高分的一帧作为备选帧，这个方法非常好地诠释了“**局部共视关键帧**”的作用。

[《全网最详细的ORB-SLAM2精讲：原理推导+逐行代码分析》](#) (点击可跳转课程详情)



长按或扫描二维码查看课程介绍和购买方式:

