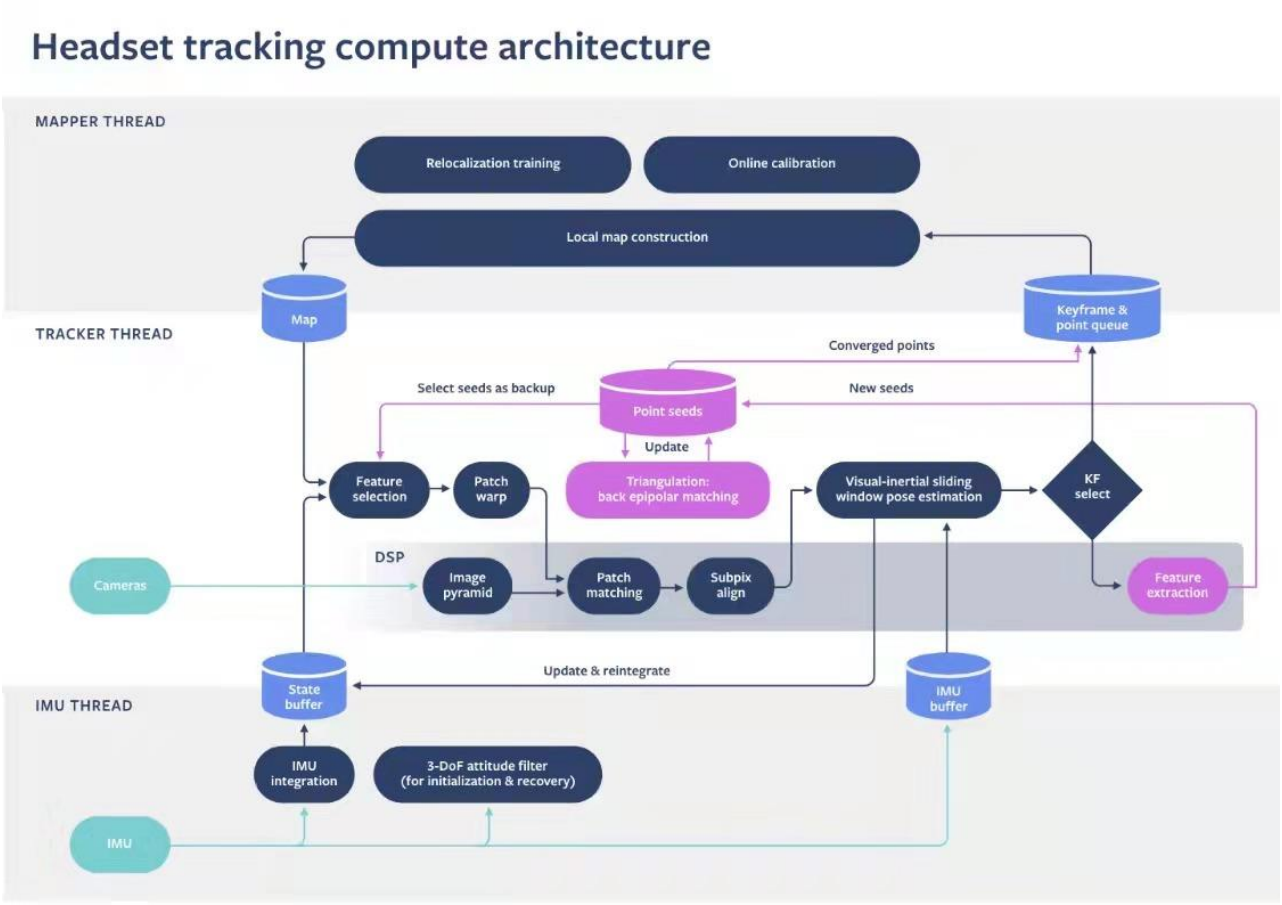
slam 工程说明文档

biotite 为 slam 项目，该项目主要参考下图设计框架。



1、 项目配置

本项目构建与手势项目类似，使用第三方库包含 eigen, g2o, opencv 和 pangolin，其中 pangolin 主要功能为 pc 端的可视化，编译到 jetson 时可剔除 pangolin 使用的相关功能。

因为与手势识别用到相同的第三方库，如果本机已经构建了 adularia 工程，可在 cmakelist 文件中设置 `set(USE_ADULARIA OFF)`，然后设定手势识别项目路径，例如，`set(ADULARIA_PATH "D:/project/adularia/download/")`

如果没有配置过 adularia 工程，需要下载 jom 编译工具并配置环境变量。

直接 cmake ..即可完成项目配置，详见 biotite 项目 readme。

项目测试：

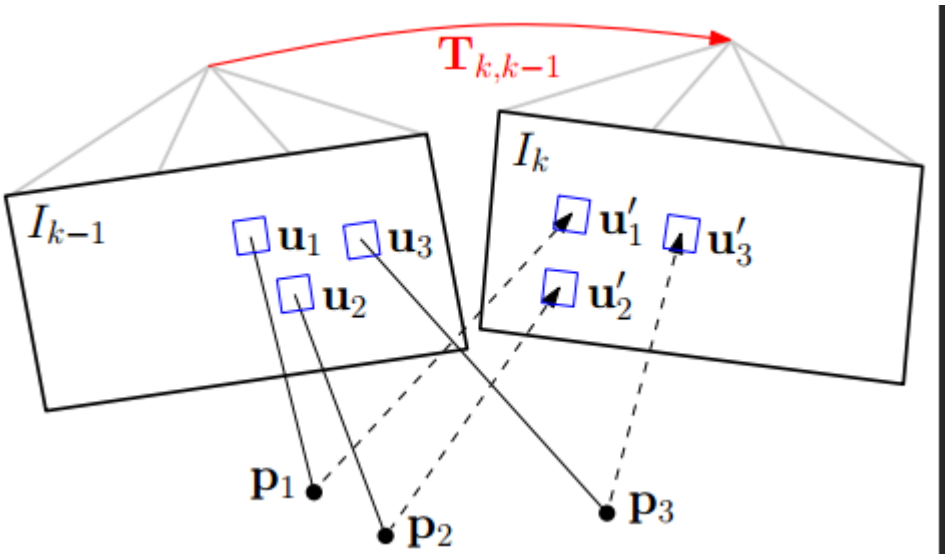
编译生成 testinit 工程，运行命令为-d 数据文件夹 -t 数据文件夹配置文件，例如“ -d D:\project\orbtmp\orb-slam3-win\MH01 -t D:\project\orbtmp\orb-slam3-win\Examples\Stereo\EuRoC_TimeStamps\MH01.txt”

2、 框架解析

数据输入包括图像数据和 IMU 数据，获取图像数据后会进行特征识别和匹配，IMU 数据会进行预计分，之后两种数据共同进行优化计算。

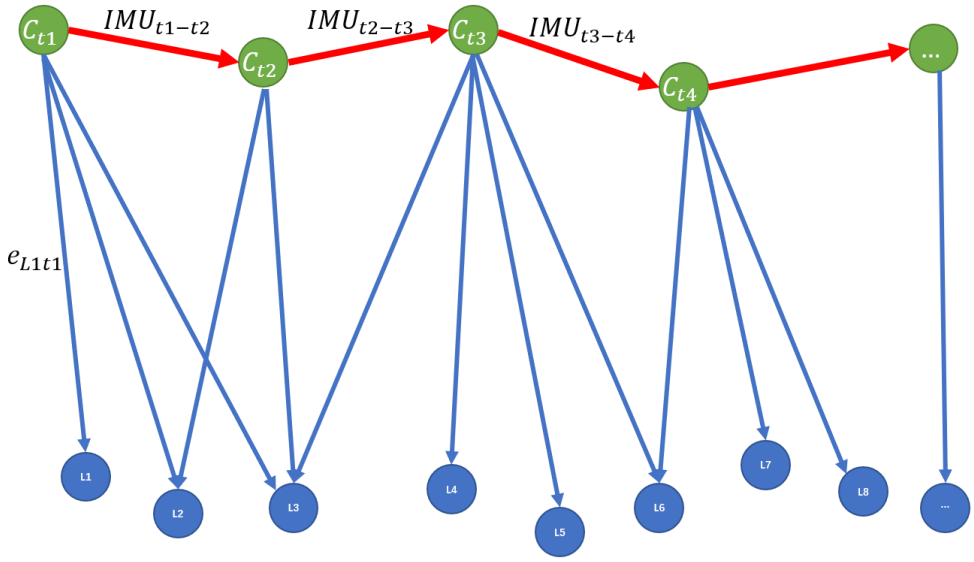
（1）IMU 数据处理位于 component 库中的 IMUtypes 文件中，对应原理可参考简明预积分推导 - 知乎 (zhihu.com)，即对 IMU 状态积分获得相邻两帧速度、位置、旋转三组状态之间的增量，然后通过与图像结果相融合进行相互验证。

（2）图像数据采用 fast 提取特征点，以特征点周围 8x8 方块的像素值代表特征点信息，采用直接法进行特征点匹配，如下图所示：



即在另一张图中与当前 8x8 方块像素值最相似的区域，该区域中心即为该特征点的匹配点，如果同时满足极线约束（左右两帧）或运动约束（前后两帧），则认定为匹配成功。

（3）非线性优化（BA）建模视觉观测和 imu 积分测量，优化结构如下图所示：



其中绿色结点表示待求解相机在时刻 ti 的状态，一般包含位置、姿态、速度，蓝色节点表示筛选出来的特征点（本方法中分两步计算，特征点位置的计算不在整体优化过程中），蓝色边为视觉观测，即利用特征点投影得到的像素点建立优化目标，红色边为 imu 测量观测约束，即通过 imu 预积分获得基于 imu 的两帧之间相机位姿变化，作为观测加入目标函数，即 min $error_{project} + error_{imu}$.

3、代码解析

（1）特征点识别及双目匹配

在第一帧或追踪的关键点数量不足时会触发特征点识别和双目匹配，主要通过 StereoTriangulation::compute 函数完成。

首先在左 cam 中检测 fast 特征点，并将图像网格化，每个网格保存一个特征点。该段实现在 feature/src/feature_detection.cpp 中。

```
void FastDetector::detect(const ImgPyr& img_pyr, const cv::Mat& mask, const size_t max_n
                          Keypoints& px_vec, Scores& score_vec, Levels& level_vec,
                          Gradients& grad_vec) {
  {
    // Detect fast corners.
    Corners corners(grid_.n_cols * grid_.n_rows, Corner(0, 0, options_.threshold_primary
    fastDetector(img_pyr, options_.threshold_primary, options_.border, options_.min_leve
                 options_.max_level, corners, grid_);
    fillFeatures(corners, mask, options_.threshold_primary, max_n_features, px_vec, scor
                 level_vec, grad_vec, grid_);
  }

  resetGrid();
}
```

其中 fastdetector 为检测 fast 特征点，fillfeatures 是通过网格过滤特征点。

然后通过 matcher 类完成双目匹配，入口函数为 matcher.findEpipolarMatchDirect，即对每一个左图中的特征点，假设其深度在 0.2 米到 20 米（通过 StereoTriangulationOptions 控制），将该范围点投影到右图上，获得极线，便利极线，寻找最匹配的 8x8 方格，如果匹配值小于阈值则认为匹配成功。遍历过程在下图函数，线性相机采用 unitplane，鱼眼相机采用 unitsphere，本质思想都是对极线均匀采样。

```
void Matcher::scanEpipolarLine(const component::Frame& frame, const Eigen::Vector3d& A,
                               const Eigen::Vector3d& B, const Eigen::Vector3d& C,
                               const PatchScore& patch_score, const int patch_level,
                               Keypoint* image_best, int* zmssd_best) {
  if (options_.scan_on_unit_sphere)
    scanEpipolarUnitSphere(frame, A, B, C, patch_score, patch_level, image_best, zmssd_best);
  else
    scanEpipolarUnitPlane(frame, A, B, C, patch_score, patch_level, image_best, zmssd_best);
}
```

匹配成功的点会分别存在左右两图的 frame 中，如果匹配数量大于，BaseOptions.kfselect_numkfs_lower_thresh 则认为初始化成功。

(2)帧间追踪

帧间追踪分为直接法追踪，精确匹配，非线性优化三部分，在 Tracking::processFrame()中完成。

直接法追踪通过比较像素值差异构建目标函数，求解相机位姿变换，在

SparseAlign 中完成。固定 3d 点位置，通过梯度调整相机位姿，获得最佳的像素值匹配。像素值梯度由离散点模拟生成，在 SparseAlign::precomputeReferencePatches 中计算。

精确匹配是在当前像素周围两个像素内，通过像素值的梯度，寻找最佳匹配点，理论上达到亚像素级的精度，在 FeatureAlign::Align 中完成。

非线性优化在 FrameOptimizer 中求解，模型为 2.（3）结果中去掉 imu 观测边的部分。只用到一种误差边 EdgeProject，即标准的通过 3d 点求解相机位姿问题。

（3）IMU 相机联合初始化

该部分的主要任务是将世界坐标系旋转到重力坐标系下，即修正 pitch 和 roll 两个角度，附带功能可以修正 imu 噪声，主要通过 imu 预计分获得相机运动观测，与图像信息相融合，在 InertialOptimization 中完成。运行为双目匹配初始化之后积累 10 帧（该过程可优化），触发该初始化操作。全部操作为求解完成的 2.（3）中的优化问题。其中 IMU 预计分在 component::IMUtypes 中完成。