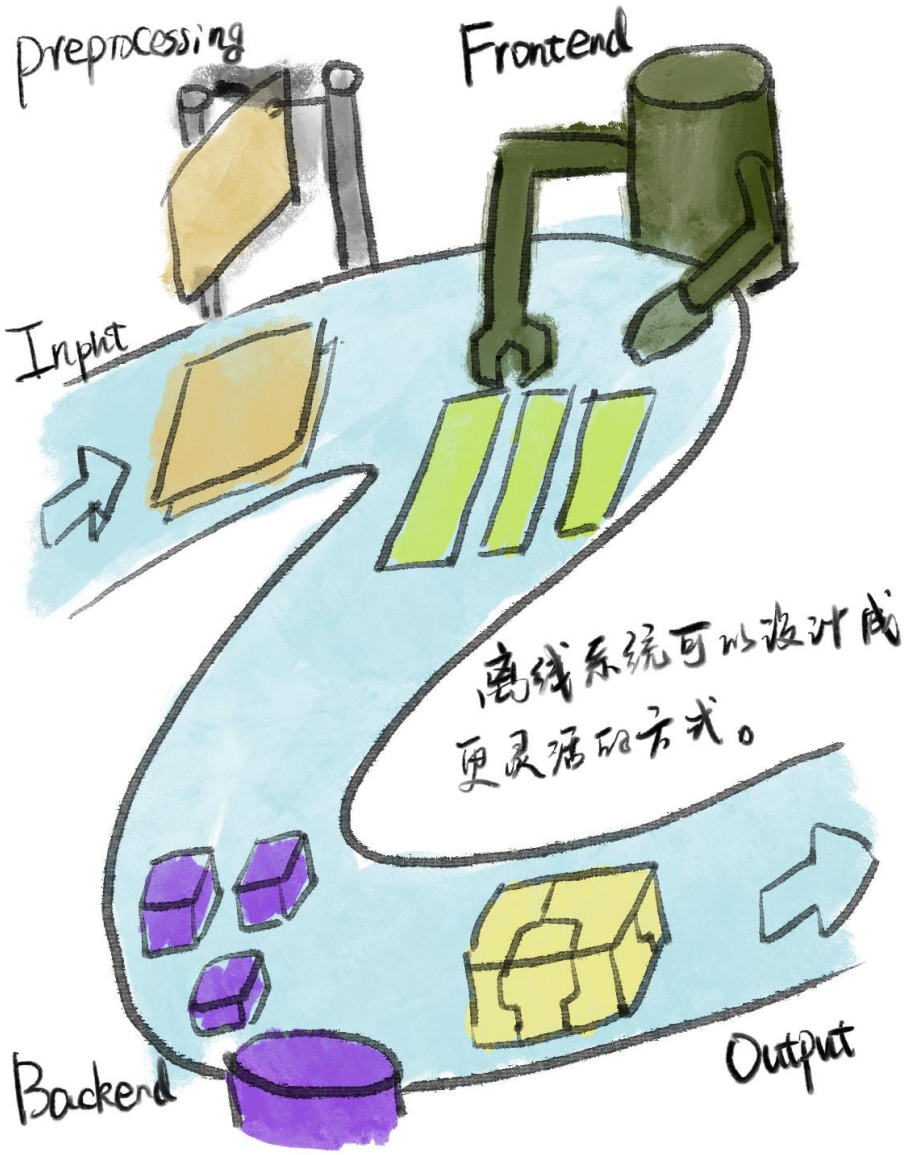
深蓝学院
shenlanxueyuan.com

自动驾驶与机器人中的SLAM技术

离线建图与在线定位系统

Contents

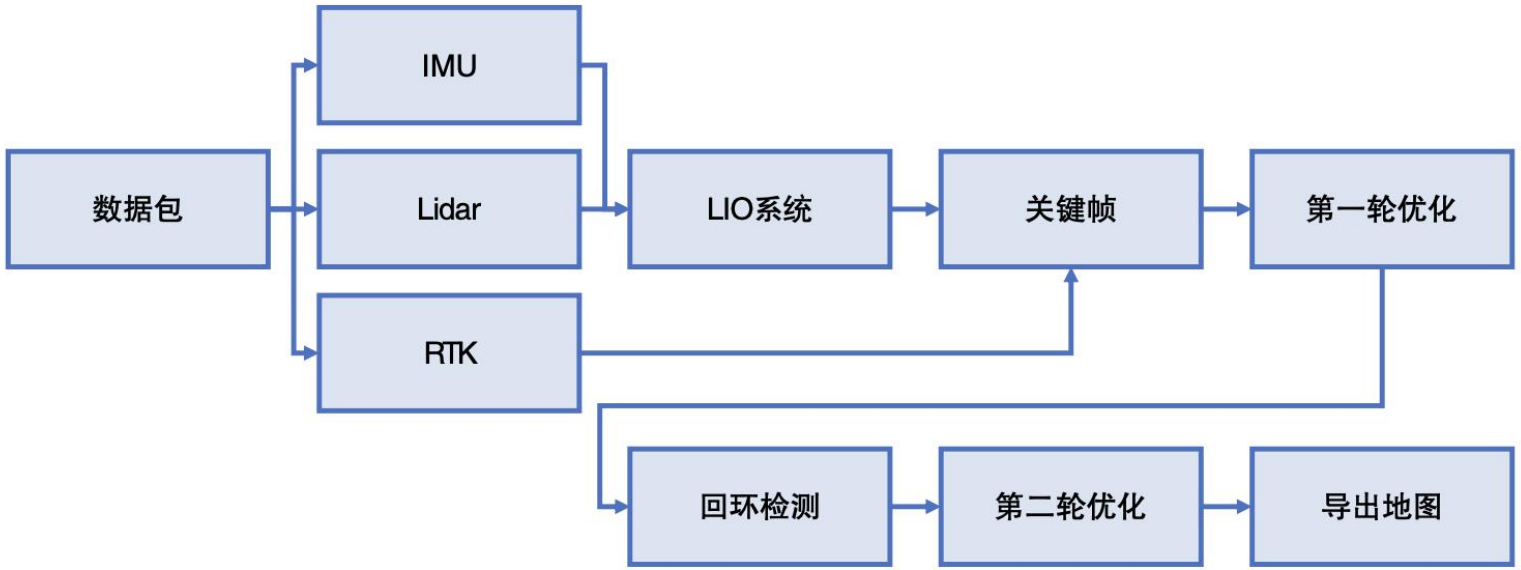
- 离线建图后端优化机制
- 地图切分、导出、在线定位



离线建图后端优化机制

离线建图后端优化机制

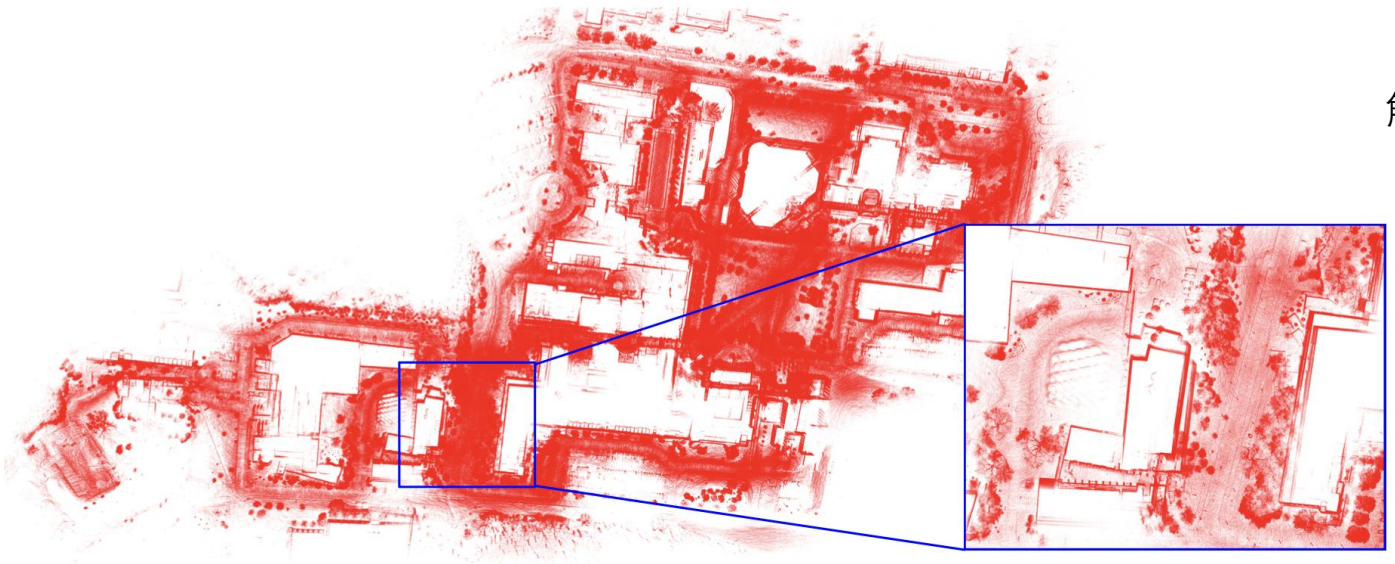
- **离线建图系统可以由分步骤的流水线模型组织而成：**
  - ✓ LIO系统可以作为系统的前端，估计轨迹的相对形状（初始估计）；
  - ✓ 但前端不可避免的有重影存在（上节课实践步骤已验证）。



离线建图后端优化机制

重影存在的原因

- LO的EKF并没有全局位姿的输入；
- 多次经过重复区域，导致估计的位姿存在误差；
- LO轨迹没有地理约束（geo referencing）。



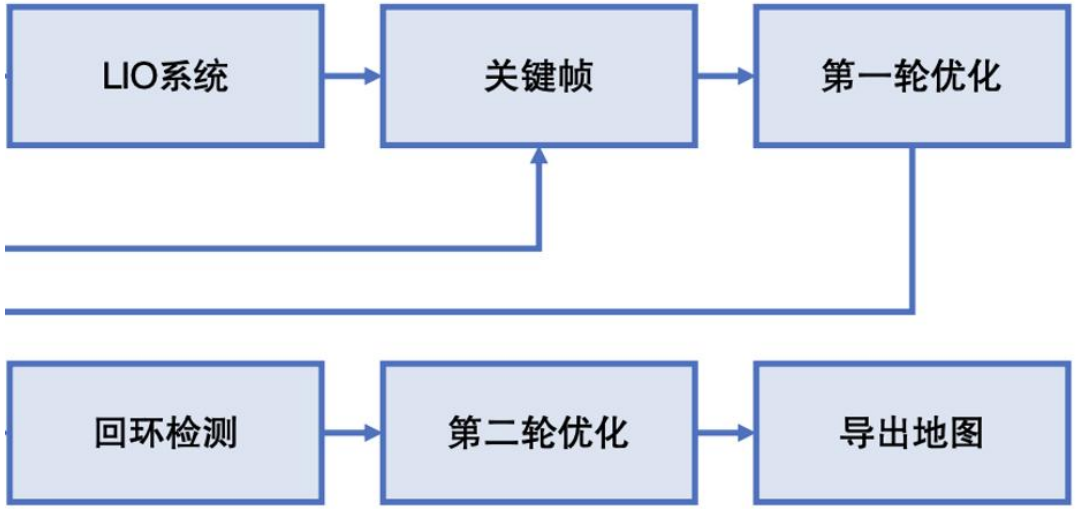
解决的方法：

1. 通过RTK引入地理约束
2. 通过充分的回环检测来保证地图没有重影
3. 两轮优化机制

离线建图后端优化机制

后端两轮优化机制

1. 第一轮优化：RTK+LIO，RTK无效点判定
2. 回环检测：基于第一轮优化位姿进行回环检测
3. 第二轮优化：RTK+LIO+回环结果



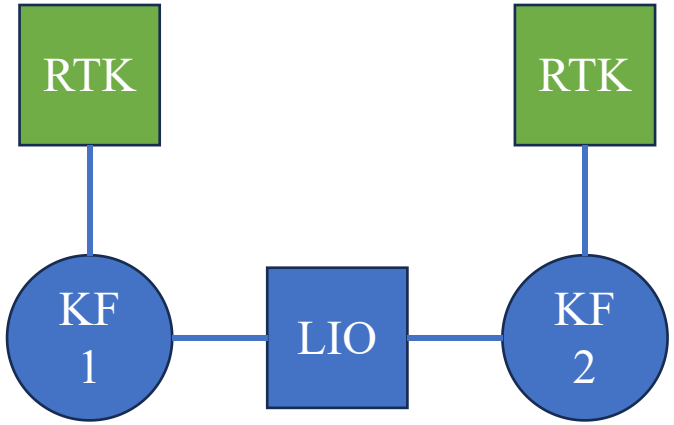
理由：

- 由于第1轮带有了RTK观测，整条轨迹的全局形状就可以确定（不会有太大的累计误差）
- 这为回环检测提供了依据，回环的检测部分可以根据位姿来确定
- 第2轮生成的点云应该不再带有重影

离线建图后端优化机制

带有RTK的图优化

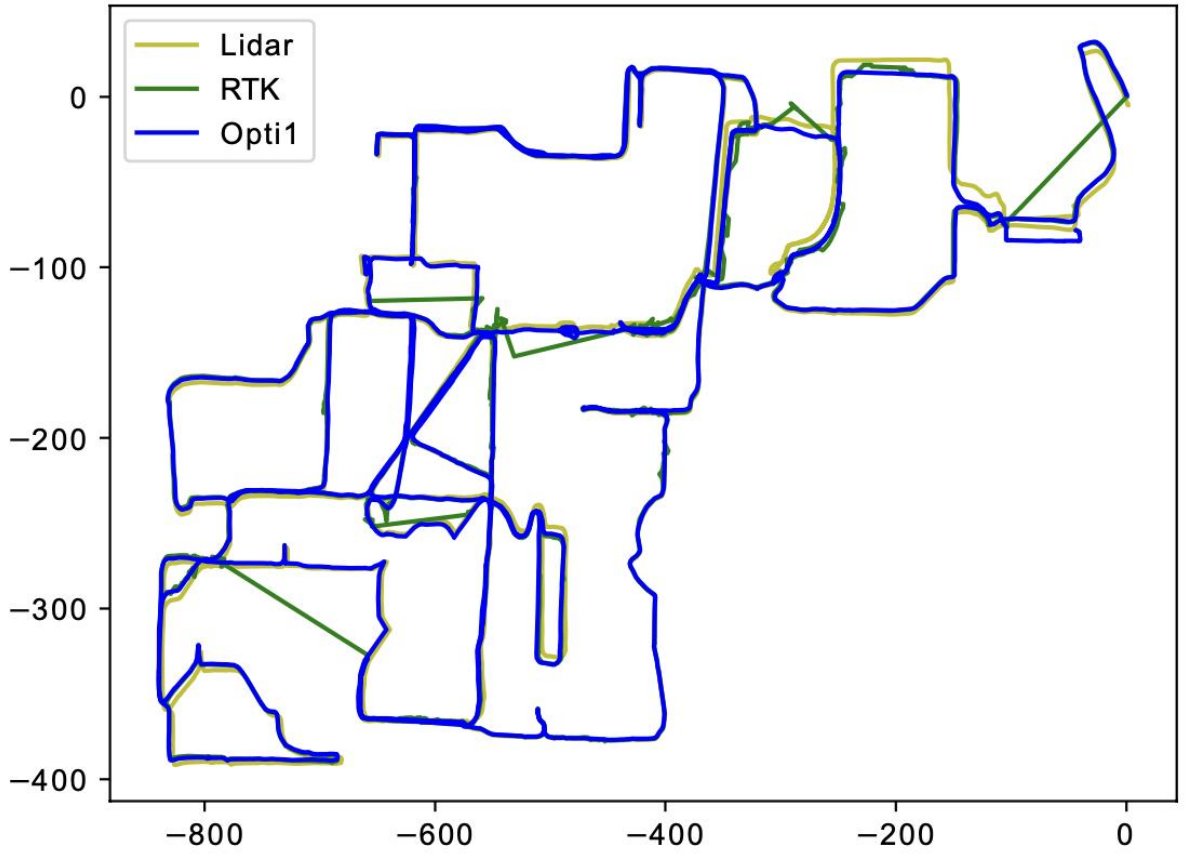
- RTK观测为一元边，约束绝对位姿
- LIO为二元边，约束相对位姿



要点：

1. 纯LIO轨迹没有绝对位姿和朝向，在优化之前，用ICP来估计RTK轨迹与LIO轨迹之间的相对变换
2. 利用robust kernel来排除RTK异常值的影响
3. 每轮优化中，使用两次优化来确认每一个RTK是否为异常值；若是异常值，则排除该RTK观测

离线建图后端优化机制



- 第一轮优化后，优化位姿应大体和RTK符合
- 可以看到RTK仍然有局部的跳动或不符的情形
- 但是该轨迹还没有进行点云匹配，所以导出的地图仍然可能会有错误

离线建图后端优化机制

回环检测

- 在离线建图程序中，我们可以使用更充分的回环检测过程（应检尽检）
- 利用多分辨率来处理较差的初始估计
- 利用并发来加速计算的过程

```
struct LoopCandidate {
  LoopCandidate() {}
  LoopCandidate(IdType id1, IdType id2, SE3 Tij) : idx1_(id1), idx2_(id2), Tij_(Tij) {}

  IdType idx1_ = 0;
  IdType idx2_ = 0;
  SE3 Tij_;
  double ndt_score_ = 0.0;
```

回环采样点：由两个关键帧组成

如果轨迹中两个关键帧距离在一定范围内，就记一个采样点

同时，通过关键帧ID来保持采样点的疏密程度

离线建图后端优化机制

- 对采样到的回环点进行并发的计算

```
void LoopClosure::ComputeLoopCandidates() {
  // 执行计算
  std::for_each(std::execution::par_unseq, loop_candiates_.begin(), loop_candiates_.end(),
  [this](LoopCandidate& c) { ComputeForCandidate(c); });
  // 保存成功的候选
  std::vector<LoopCandidate> succ_candidates;
  for (const auto& lc : loop_candiates_) {
    if (lc.ndt_score_ > ndt_score_th_) {
      succ_candidates.emplace_back(lc);
    }
  }
  LOG(INFO) << "success: " << succ_candidates.size() << "/" << loop_candiates_.size();

  loop_candiates_.swap(succ_candidates);
}
```

离线建图后端优化机制

回环点计算过程：

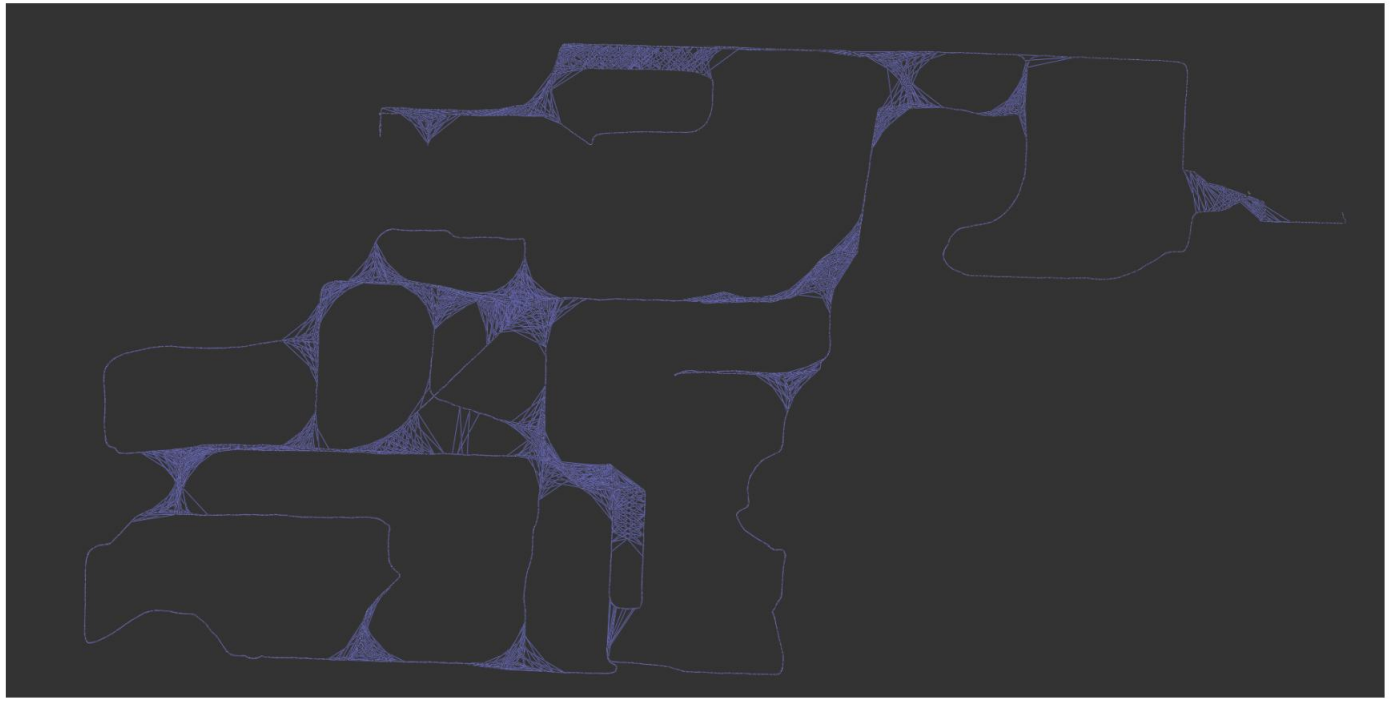
- KF1附近的关键帧构成一个局部地图
- 将KF2配准到这个地图中
- 利用多分辨率NDT计算位姿分值
- 利用分值来确认回环是否成立

这里使用10m, 5m, 4m, 3m的阶梯分辨率NDT

```
/// 不同分辨率下的匹配
CloudPtr output(new PointCloudType);
std::vector<double> res{10.0, 5.0, 4.0, 3.0};
for (auto& r : res) {
  ndt.setResolution(r);
  auto rough_map1 = VoxelCloud(submap_kf1, r * 0.1);
  auto rough_map2 = VoxelCloud(submap_kf2, r * 0.1);
  ndt.setInputTarget(rough_map1);
  ndt.setInputSource(rough_map2);

  ndt.align(*output, Tw2);
  Tw2 = ndt.getFinalTransformation();
}
```

离线建图后端优化机制

- **最后将回环约束引入Pose graph，注意添加robust kernel以避免错误回环的影响**

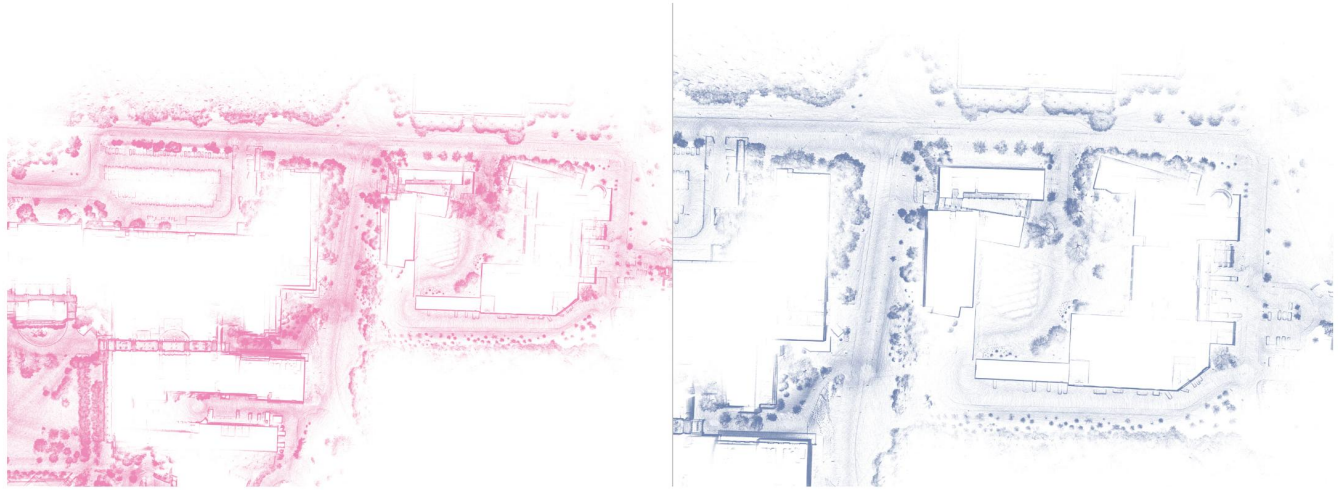


- 带回环检测的位姿图
- 蓝色边表示两个关键帧之间存在回环约束
- 我们在重叠区域进行了充分的回环检测

离线建图后端优化机制

现在可以使用第2轮优化的位姿导出地图

- 此时地图重影已经消除，而且位姿和RTK有对应关系



前后端的局部地图对比

全局地图

离线建图后端优化机制

- 最后我们将地图切分导出
- 通常按一定边长进行切片，例如100x100m/片，
  使用的时候按片进行动态加载
- 每块点云生成一个区块ID，然后按照ID进行存取

离线建图后端优化机制

把各个步骤组合在一起，就构成了整个建图软件

我们可以自由地封装该软件，现在它并不依赖ros core来通信。

只要把地图文件夹分开，一次可以启动任意多个建图实例，例如：

将建图封装成Python，通过脚本来调用；

也可以封装成服务器形式，由网页来调用。

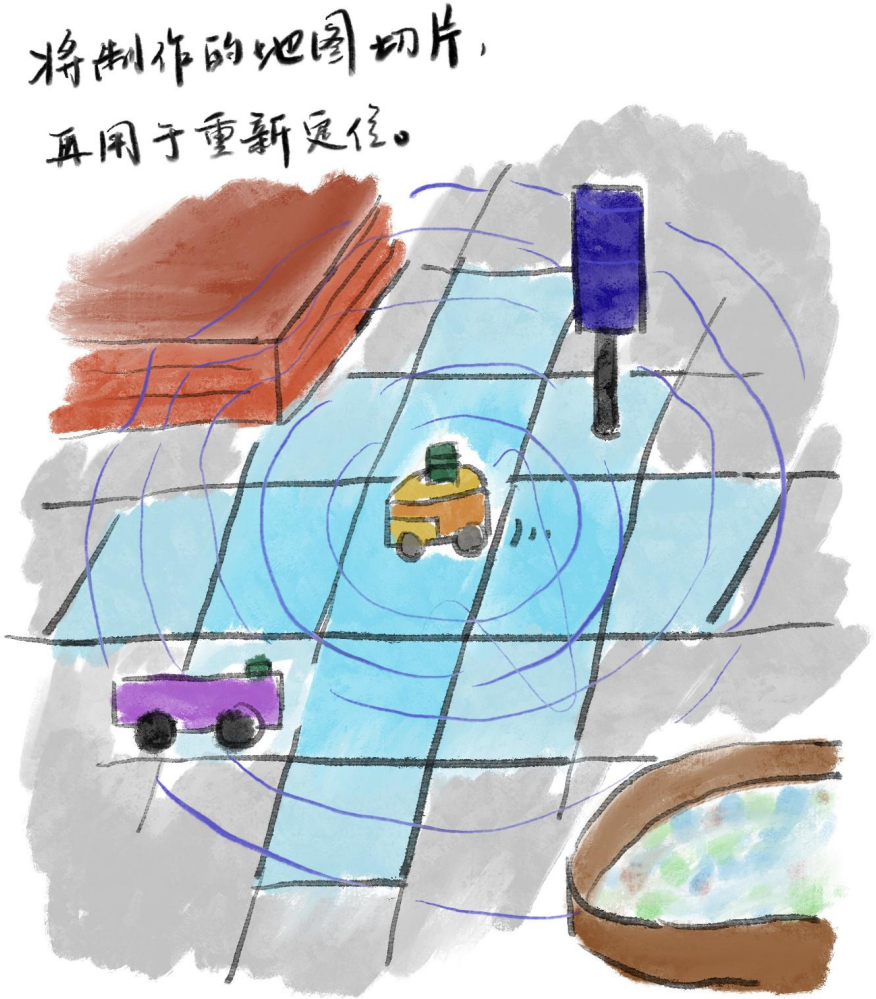
这部分属于infrastructure内容，不属于算法，因此不展开。

```
sad::Frontend frontend(FLAGS_config_yaml);
if (!frontend.Init()) {
    LOG(ERROR) << "failed to init frontend.";
    return -1;
}

frontend.Run();

sad::Optimization opti(FLAGS_config_yaml);
if (!opti.Init(1)) {
    LOG(ERROR) << "failed to init opti1.";
    return -1;
}
opti.Run();

sad::LoopClosure lc(FLAGS_config_yaml);
if (!lc.Init()) {
    LOG(ERROR) << "failed to init loop closure.";
    return -1;
}
lc.Run();

sad::Optimization opti2(FLAGS_config_yaml);
if (!opti2.Init(2)) {
    LOG(ERROR) << "failed to init opti2.";
    return -1;
}
opti2.Run();
```

地图切分、导出，在线定位

地图切分、导出，在线定位

- **建完的地图在切分导出后，可以用于在线定位**
- **需要解决的几个问题：**
  - 地图分片加载与卸载
  - 地图发生改变之后，匹配算法的目标点云需要更新
  - RTK不带有航向时，初始航向的搜索问题
  - 实时运行时，激光定位的初值问题



地图切分、导出，在线定位

地图分片加载逻辑

- 首先确认车辆在哪一个区块的地图中
- 加载周围2格地图
- 同时，遍历已经加载的部分，如果超过车辆3格，则卸载
- 这可以防止车辆处于网格边界时的反复加载卸载的情况（缓冲）
- 当地图发生改变时，重新设置NDT的target（PCL版本NDT需要重建所有体素）

```
// 一个区域的周边地图，我们认为9个就够了
std::set<Vec2i, less_vec<2>> surrounding_index{
    key + Vec2i(0, 0), key + Vec2i(-1, 0), key + Vec2i(-1, -1), key + Vec2i(-1, 1), key + Vec2i(0, -1),
    key + Vec2i(0, 1), key + Vec2i(1, 0),  key + Vec2i(1, -1),  key + Vec2i(1, 1),
};

// 加载必要区域
bool map_data_changed = false;
int cnt_new_loaded = 0, cnt_unload = 0;
for (auto& k : surrounding_index) {
    if (map_data_index_.find(k) == map_data_index_.end()) {
        // 该地图数据不存在
        continue;
    }

    if (map_data_.find(k) == map_data_.end()) {
        // 加载这个区块
        CloudPtr cloud(new PointCloudType);
        pcl::io::loadPCDFile(data_path_ + std::to_string(k[0]) + "_" + std::to_string(k[1]) + ".pcd", *cloud);
        map_data_.emplace(k, cloud);
        map_data_changed = true;
        cnt_new_loaded++;
    }
}

// 卸载不需要的区域，这个稍微加大一点，不需要频繁卸载
for (auto iter = map_data_.begin(); iter != map_data_.end();) {
    if ((iter->first - key).cast<float>().norm() > 3.0) {
        // 卸载本区块
        iter = map_data_.erase(iter);
        cnt_unload++;
        map_data_changed = true;
    } else {
        iter++;
    }
}
```

地图切分、导出，在线定位

RTK初始化

- NCLT数据集中的RTK没有航向，只有位置
- 在刚开始阶段，利用RTK位置来定位，而角度使用遍历搜索
- 同样利用多分辨率处理初始值精度不够的问题

```
/// 由于RTK不带角度，这里按固定步长扫描RTK角度
double grid_ang_range = 360.0, grid_ang_step = 10;  // 角度搜索范围与步长
for (double ang = 0; ang < grid_ang_range; ang += grid_ang_step) {
    SE3 pose(SO3::rotZ(ang * math::kDEG2RAD), Vec3d(0, 0, 0) + last_gnss_->utm_pose_.translation());
    GridSearchResult gr;
    gr.pose_ = pose;
    search_poses.emplace_back(gr);
}

LOG(INFO) << "grid search poses: " << search_poses.size();
std::for_each(std::execution::par_unseq, search_poses.begin(), search_poses.end(),
              [this](GridSearchResult& gr) { AlignForGrid(gr); });

// 选择最优的匹配结果
auto max_ele = std::max_element(search_poses.begin(), search_poses.end(),
                                [](const auto& g1, const auto& g2) { return g1.score_ < g2.score_; });
```

搜索角度部分

多分辨率部分

```
/// 多分辨率
pcl::NormalDistributionsTransform<PointType, PointType> ndt;
ndt.setTransformationEpsilon(0.05);
ndt.setStepSize(0.7);
ndt.setMaximumIterations(40);

ndt.setInputSource(current_scan_);
auto map = ref_cloud_;

CloudPtr output(new PointCloudType);
std::vector<double> res{10.0, 5.0, 4.0, 3.0};
Mat4f T = gr.pose_.matrix().cast<float>();
for (auto& r : res) {
    auto rough_map = VoxelCloud(map, r * 0.1);
    ndt.setInputTarget(rough_map);
    ndt.setResolution(r);
    ndt.align(*output, T);
    T = ndt.getFinalTransformation();
}

gr.score_ = ndt.getTransformationProbability();
gr.result_pose_ = Mat4ToSE3(ndt.getFinalTransformation());
```

地图切分、导出，在线定位

融合定位框架

- 融合定位沿用松耦合LIO章节的框架代码
- 利用ESKF进行融合，ESKF的预测作为NDT配准的初值
- 同时，将NDT的观测位姿作为ESKF的观测源进行融合

融合逻辑

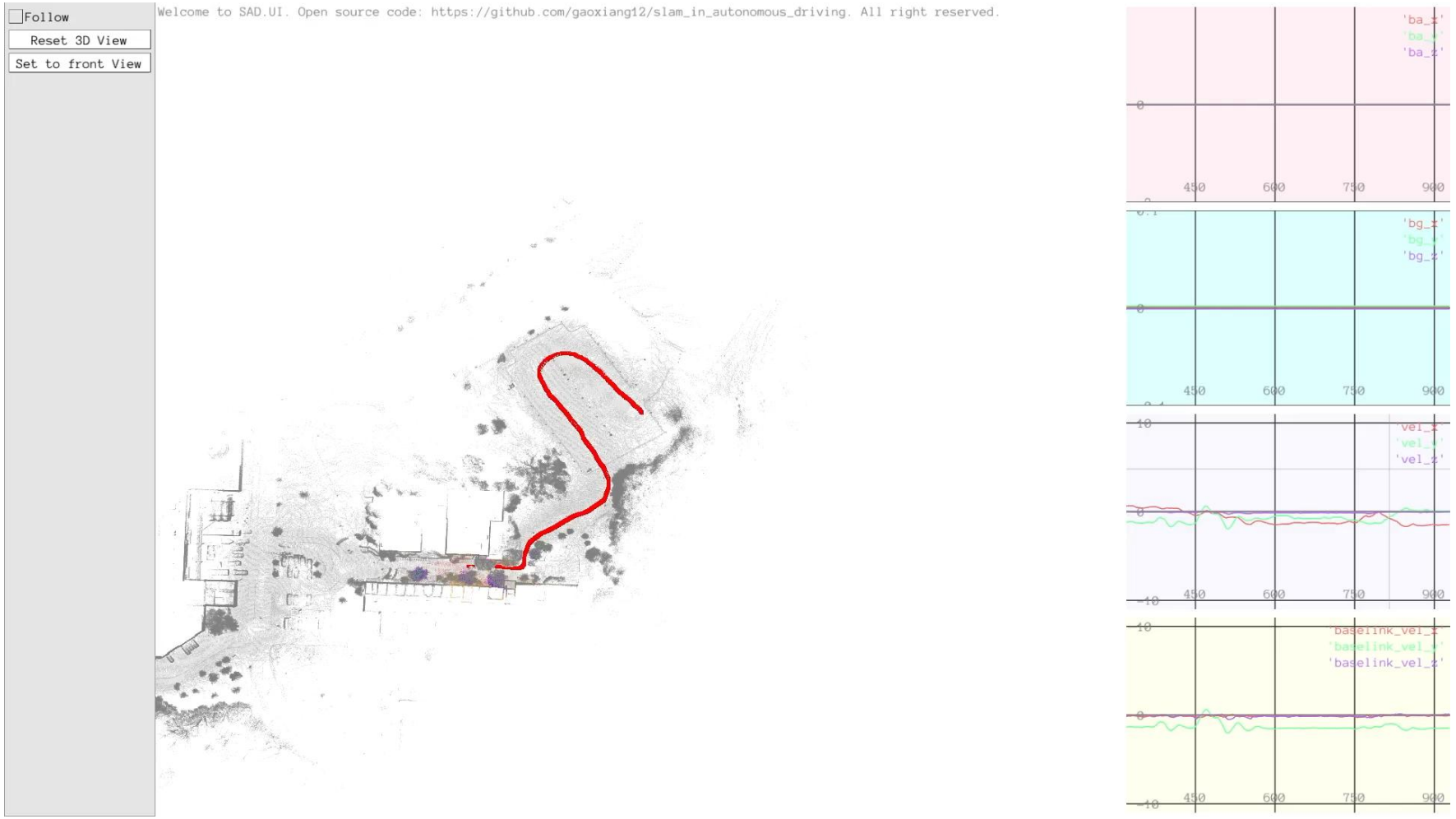
```
ndt_.setInputCloud(current_scan_);
CloudPtr output(new PointCloudType);
ndt_.align(*output, pred.matrix().cast<float>());

SE3 pose = Mat4ToSE3(ndt_.getFinalTransformation());
eskf_.ObserveSE3(pose, 1e-1, 1e-2);
```

注：

- RTK也可以用类似的方法来融合
- 这种方法容易受IMU异常值影响（从ESKF拿预测），如果希望更稳定，也可以用NDT的位姿进行递推
- 有轮速也可以融轮速
- 激光定位并不像RTK那样能够直接给出定位结果，而是需要有初始估计和收敛范围。通常认为NDT在栅格大小的2倍以上无法正确收敛

地图切分、导出，在线定位

- 实现效果



地图切分、导出，在线定位

- 代码实现

大作业

本书大部分Demo都基于NDT实现，与PCL版本的NDT有明显不同。请根据本书提供的内容，实现以下功能：

1. 为本书的NDT设计一个匹配度评估指标，利用该指标可以判断NDT匹配的好坏。可以参考PCL NDT的transprobability值，也可以自己来设计。
2. 利用第1题的指标，修改程序，实现mapping部分的回环检测。注意要利用分值好坏来判断回环检测是否有效。
3. 将建图结果导出为NDT map，即将NDT体素内的均值和协方差都存储成文件。同样建议分块存储。
4. 实现基于NDT map的激光定位。根据车辆实时位姿，加载所需的NDT体素并完成定位。同样，要根据初始的RTK位置进行角度搜索。为了方便可视化，也可以保留点云地图以作对比。
5. 给出上述结果相比于PCL NDT的性能、存储空间等关键指标。

深蓝学院
shenlanxueyuan.com

感谢聆听！

Thanks for Listening