

第1季

第8章：激光SLAM系统



主讲人：张虎

(小虎哥哥爱学习)

- 先导课
- 第1季：快速梳理知识要点与学习方法 ✓
- 第2季：详细推导数学公式与代码解析
- 第3季：代码实操以及真实机器人调试
- 答疑课

----- (永久免费 • 系列课程 • 长期更新) -----

本书内容安排

一、编程基础篇

第1章：ROS入门必备知识

第2章：C++编程范式

第3章：OpenCV图像处理

二、硬件基础篇

第4章：机器人传感器

第5章：机器人主机

第6章：机器人底盘

三、SLAM篇

第7章：SLAM中的数学基础

第8章：激光SLAM系统

第9章：视觉SLAM系统

第10章：其他SLAM系统

四、自主导航篇

第11章：自主导航中的数学基础

第12章：典型自主导航系统

第13章：机器人SLAM导航综合实战

回顾：

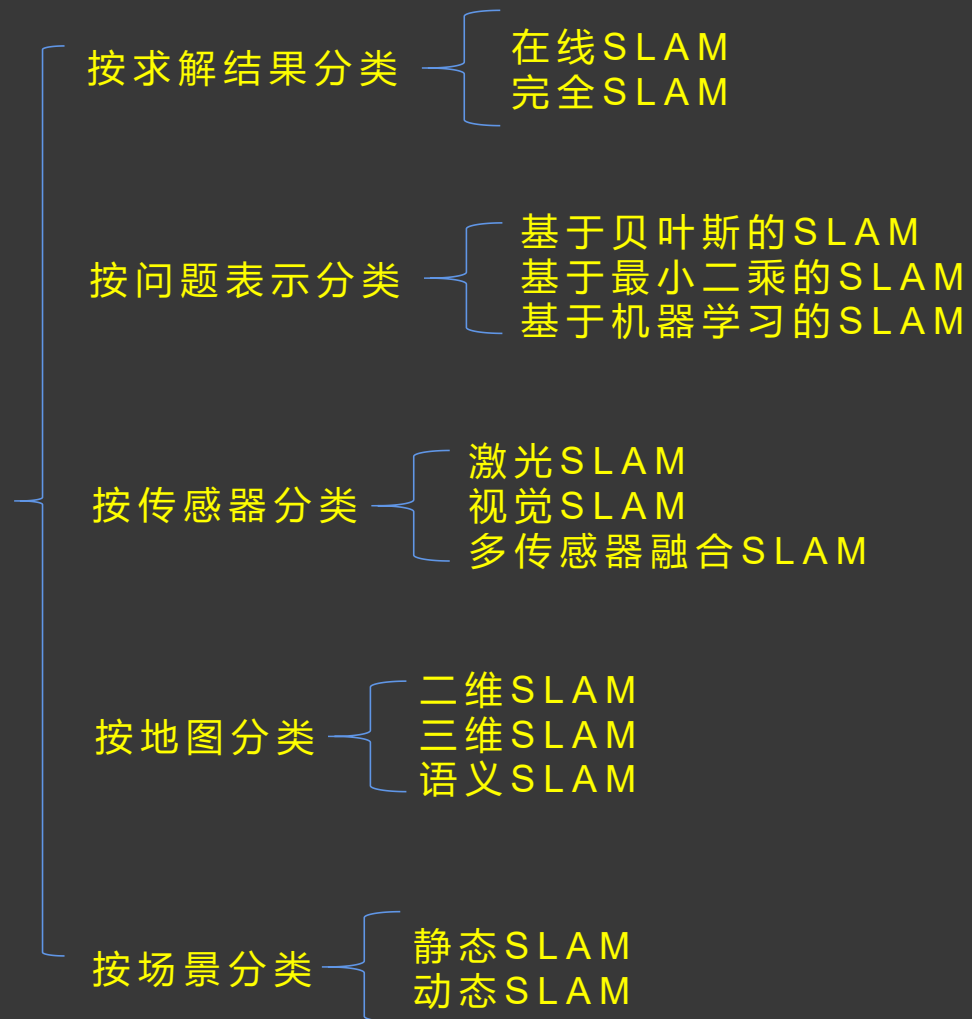
■ SLAM学习路线

■ SLAM学习方法

先学激光SLAM，还是先学视觉SLAM？

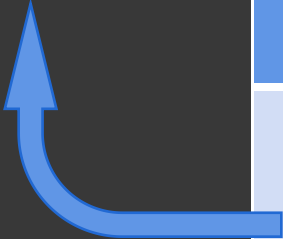
？

？



先学激光SLAM，还是先学视觉SLAM？

第4章 机器人传感器



	激光SLAM	视觉SLAM
传感器数据	激光雷达 离线数据集 虚拟仿真	相机 离线数据集 虚拟仿真
成本	高	低
实现难度	简单	困难

小虎哥的观点：

- ① 建议新手先学激光SLAM，入门简单一点。
- ② 其实激光SLAM和视觉SLAM都要会，并且实际工作还远远不够。

激光SLAM算法有哪些？



比较老的激光SLAM算法：

- karto SLAM
- Hector SLAM
- ...

比较新的激光SLAM算法：

- LIO-SAM
- SUMA++
- ...

阅读代码的正确方式

① 开源代码并不完美，也非唯一答案

使用成熟的开源项目，追求开发效率，重点是适配自身需求

使用早期的开源项目，参考设计思路，逐句阅读照搬代码没有必要

② 计算机开发经验靠日积月累，绝非一朝一夕

遇事不慌，放平心态

内容概要

8.1 Gmapping算法

8.2 Cartographer算法

8.3 LOAM算法

8.1 Gmapping算法

Gmapping算法是不是过时了，还有必要学习吗？

- ① 越简单（纯粹）的算法，越容易理解原理
- ② 学习早期成果，便于理解一项技术的发展脉络
- ③ 没有真正过时的技术，技术发展都是有周期的

8.1 Gmapping算法

- Gmapping原理分析

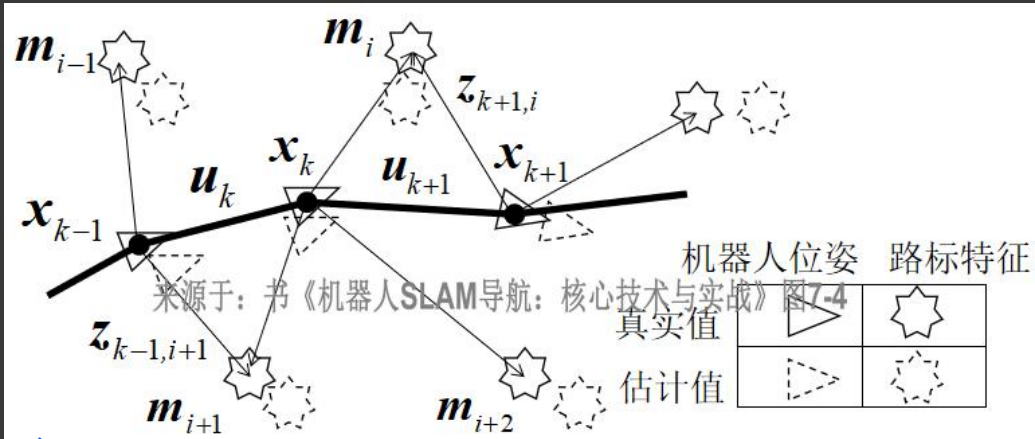
■ Gmapping源码解读

■ Gmapping安装与运行
- ① RBPF的滤波过程

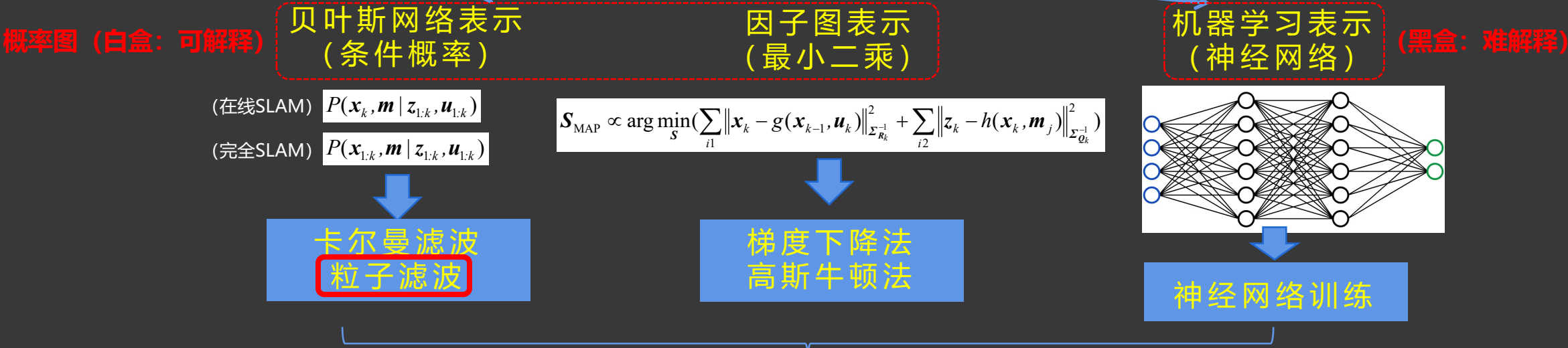
② RBPF的建议分布改进

③ RBPF的重采样改进

④ 改进RBPF的滤波过程



回顾：SLAM问题的表示



SLAM问题的求解

8.1 Gmapping算法

- Gmapping原理分析

■ Gmapping源码解读

■ Gmapping安装与运行
- ① RBPF的滤波过程

② RBPF的建议分布改进

③ RBPF的重采样改进

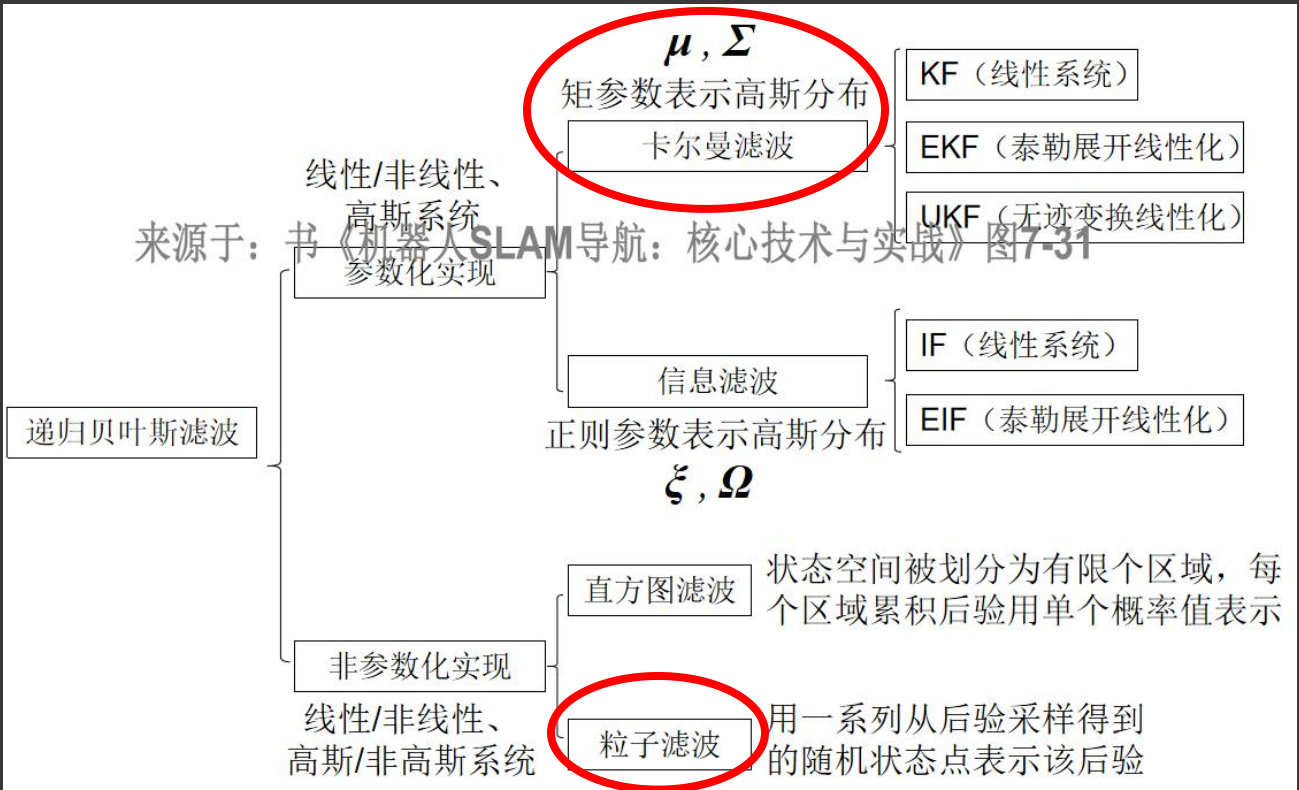
④ 改进RBPF的滤波过程

SLAM问题
贝叶斯网络表示
(条件概率)

(在线SLAM) $P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{z}_{1:k}, \mathbf{u}_{1:k})$

(完全SLAM) $P(\mathbf{x}_{1:k}, \mathbf{m} \mid \mathbf{z}_{1:k}, \mathbf{u}_{1:k})$

卡尔曼滤波
粒子滤波



8.1 Gmapping算法

■ Gmapping原理分析

■ Gmapping源码解读

■ Gmapping安装与运行

- ① RBPF的滤波过程
- ② RBPF的建议分布改进
- ③ RBPF的重采样改进
- ④ 改进RBPF的滤波过程

RBPF: Rao-Blackwellization Particle Filter

Rao-Blackwell 定理 以一个任意的原始估计为起点，寻找最小方差无偏估计量 (MVUE)
(两个作者的名字)

SLAM问题
贝叶斯网络表示
(条件概率)

一般形式

利用条件独立性
将条件概率化简展开

RBPF形式
(先求定位问题，再求建图问题)

$P(x_{1:k}, m | z_{1:k}, u_{1:k})$

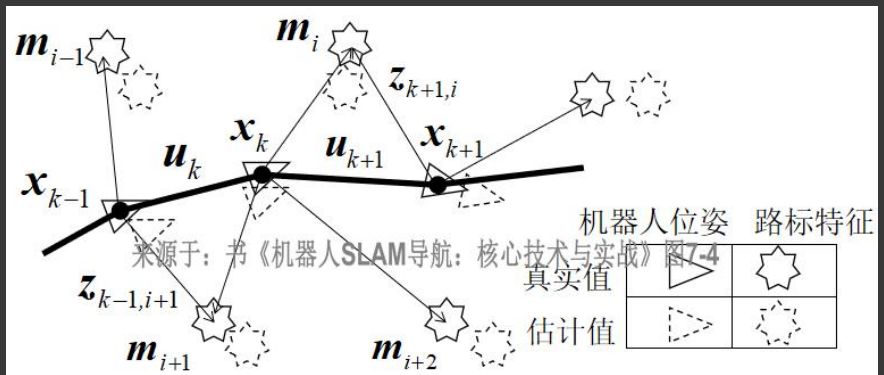
$P(x_{1:t}, m | z_{1:t}, u_{1:t-1}) = P(m | x_{1:t}, z_{1:t}) \cdot P(x_{1:t} | z_{1:t}, u_{1:t-1})$

建图问题：点云拼接
(简单)

定位问题：比如SIR (sampling importance resampling)
(核心)

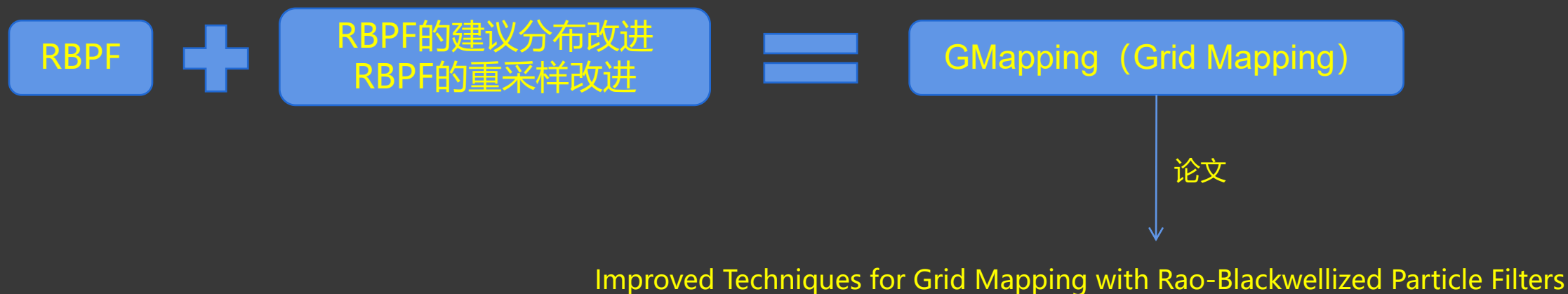
- ① 采样
- ② 重要性权重
- ③ 重采样
- ④ 地图估计

粒子滤波器



8.1 Gmapping算法

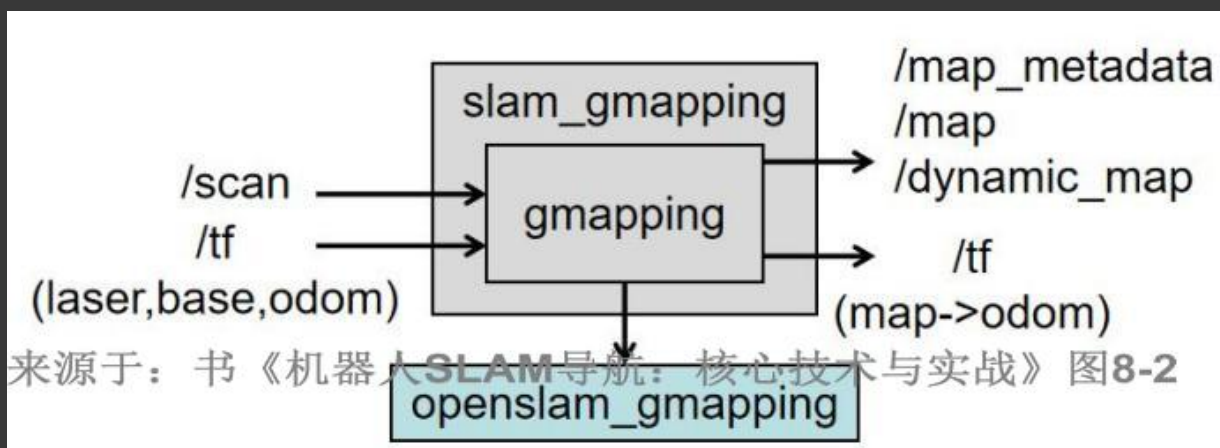
- Gmapping原理分析 →
 - ① RBPF的滤波过程
 - ② RBPF的建议分布改进
 - ③ RBPF的重采样改进
 - ④ 改进RBPF的滤波过程
- Gmapping源码解读
- Gmapping安装与运行



8.1 Gmapping算法

代码阅读技巧： 程序调用流程 + 数据调用流程

- Gmapping原理分析
- Gmapping源码解读
- Gmapping安装与运行



ROS元功能包：slam_gmapping
ROS功能包：gmapping
ROS功能包：openslam_gmapping

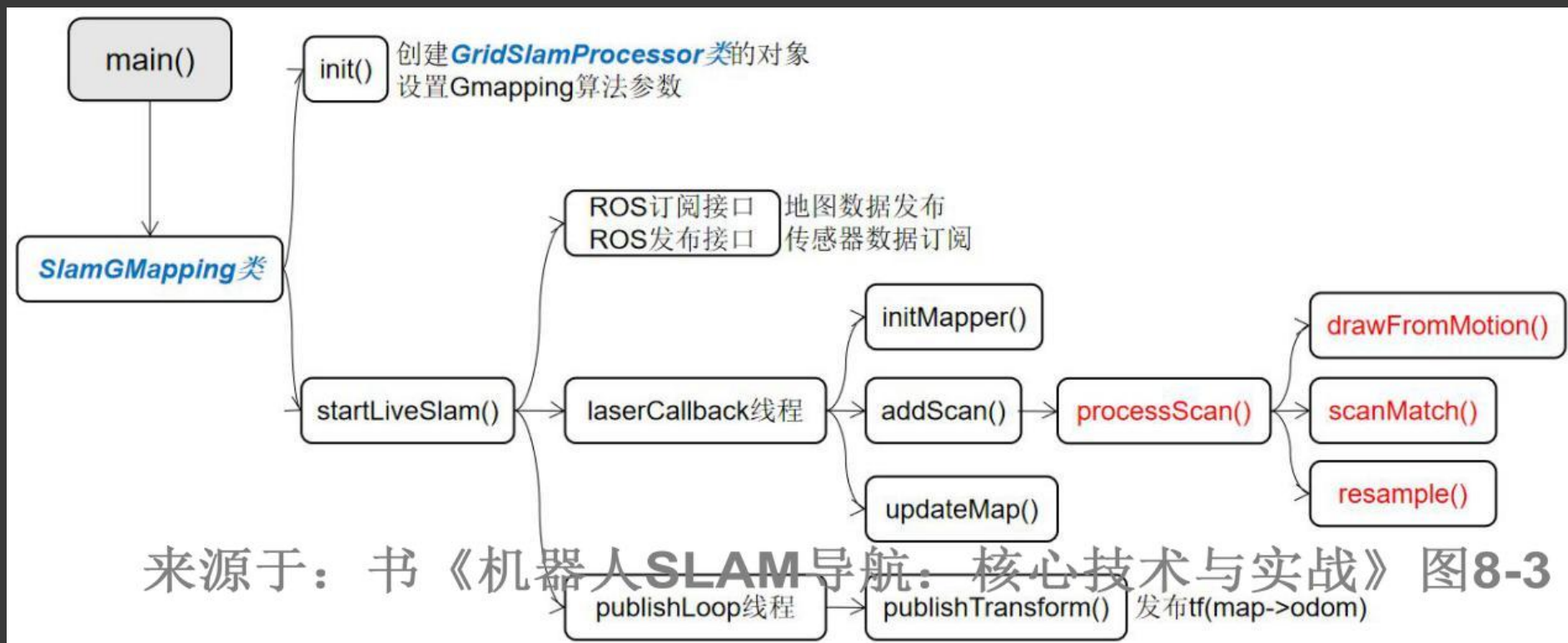
https://github.com/ros-perception/slam_gmapping

https://github.com/ros-perception/openslam_gmapping

8.1 Gmapping算法

代码阅读技巧：程序调用流程 + 数据调用流程

- Gmapping原理分析
- Gmapping源码解读
- Gmapping安装与运行



来源于：书《机器人SLAM导航：核心技术与实战》图8-3

8.1 Gmapping算法

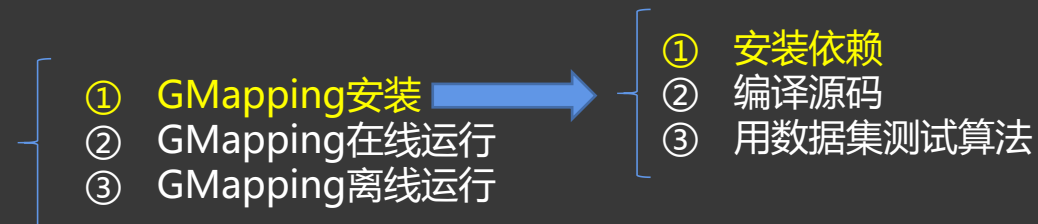
代码实操建议： 计算机**开发经验**靠日积月累，绝非一朝一夕，遇事不慌，放平心态

■ Gmapping原理分析

■ Gmapping源码解读

■ **Gmapping安装与运行**

环境：Ubuntu18.04+ROS melodic



```
#安装openslam_gmapping和slam_gmapping功能包及其依赖
sudo apt install ros-melodic-openslam-gmapping ros-melodic-gmapping
#卸载openslam_gmapping和slam_gmapping功能包但保留其依赖
sudo apt remove ros-melodic-openslam-gmapping ros-melodic-gmapping
```

8.1 Gmapping算法

代码实操建议： 计算机**开发经验**靠日积月累，绝非一朝一夕，遇事不慌，放平心态

■ Gmapping原理分析

■ Gmapping源码解读

■ **Gmapping安装与运行**

环境：Ubuntu18.04+ROS melodic

- ① GMapping安装
- ② GMapping在线运行
- ③ GMapping离线运行

- ① 安装依赖
- ② 编译源码
- ③ 用数据集测试算法

```
#切换到工作空间目录
cd ~/catkin_ws/src/
#下载slam_gmapping功能包源码
git clone https://github.com/ros-perception/slam_gmapping.git
cd slam_gmapping
#查看代码版本是否为molodic，如果不是请使用git checkout命令切换到对应版本
git branch
#下载openslam_gmapping功能包源码
git clone https://github.com/ros-perception/openslam_gmapping.git
cd openslam_gmapping
#同样查看代码版本是否为molodic，如果不是请使用git checkout命令切换到对应版本
git branch
#编译
cd ~/catkin_ws/
catkin_make -DCATKIN_WHITELIST_PACKAGES="openslam_gmapping"
catkin_make -DCATKIN_WHITELIST_PACKAGES="gmapping"
```


8.1 Gmapping算法

代码实操建议：计算机开发经验靠日积月累，绝非一朝一夕，遇事不慌，放平心态

■ Gmapping原理分析

■ Gmapping源码解读

■ Gmapping安装与运行

环境：Ubuntu18.04+ROS melodic

- ① GMapping安装
- ② GMapping在线运行
- ③ GMapping离线运行

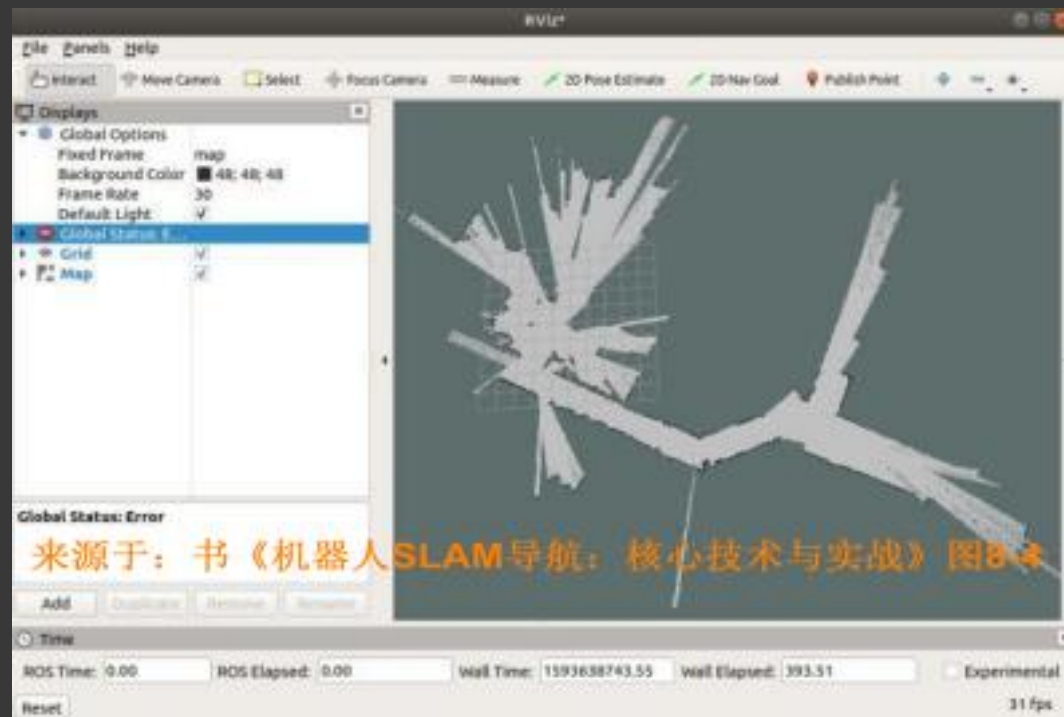
- ① 安装依赖
- ② 编译源码
- ③ 用数据集测试算法

```
#用默认launch文件启动gmapping
roslaunch gmapping slam_gmapping_pr2.launch

#切换到数据集存放目录
cd ~/Downloads/
#播放数据集
rosbag play basic_localization_stage_indexed.bag

#启动rviz
rviz
```

Gmapping数据集测试效果



来源于：书《机器人SLAM导航：核心技术与实战》图8-4

8.1 Gmapping算法

代码实操建议： 计算机开发经验靠日积月累，绝非一朝一夕，遇事不慌，放平心态

■ Gmapping原理分析

■ Gmapping源码解读

■ Gmapping安装与运行

环境：Ubuntu18.04+ROS melodic

- ① GMapping安装
- ② GMapping在线运行
- ③ GMapping离线运行

```
#启动激光雷达
roslaunch ydlidar my_x4.launch

#启动底盘
roslaunch xiihoo_bringup minimal.launch

#启动底盘urdf描述
roslaunch xiihoo_description xiihoo_description.launch

#启动建图
roslaunch gmapping slam_gmapping_xiihoo.launch

#首次使用键盘遥控，需要先安装对应功能包
sudo apt install ros-melodic-teleop-twist-keyboard

#启动键盘遥控
roslaunch teleop_twist_keyboard teleop_twist_keyboard.py

#启动rviz
rviz
```

slam_gmapping_xiihoo.launch启动文件：

```
1 <launch>
2 <!--param name="use_sim_time" value="true"/-->
3
4 <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
5 <remap from="scan" to="/scan"/>
6 <param name="base_frame" value="base footprint"/>
7 <param name="map_frame" value="map"/>
8 <param name="odom_frame" value="odom"/>
9
10 <param name="map_update_interval" value="5.0"/>
11 <param name="maxOrange" value="16.0"/>
12 <param name="sigma" value="0.05"/>
13 <param name="kernelSize" value="1"/>
14 <param name="lstep" value="0.05"/>
15 <param name="astep" value="0.05"/>
16 <param name="iterations" value="5"/>
17 <param name="lsigma" value="0.075"/>
18 <param name="ogain" value="3.0"/>
19 <param name="lskip" value="0"/>
20 <param name="srr" value="0.1"/>
21 <param name="srt" value="0.2"/>
22 <param name="str" value="0.1"/>
23 <param name="stt" value="0.2"/>
24 <param name="linearUpdate" value="1.0"/>
25 <param name="angularUpdate" value="0.5"/>
26 <param name="temporalUpdate" value="3.0"/>
27 <param name="resampleThreshold" value="0.5"/>
28 <param name="particles" value="30"/>
29 <param name="xmin" value="-1.0"/>
30 <param name="ymin" value="-1.0"/>
31 <param name="xmax" value="1.0"/>
32 <param name="ymax" value="1.0"/>
33 <param name="delta" value="0.05"/>
34 <param name="llsamplerange" value="0.01"/>
35 <param name="llsamplestep" value="0.01"/>
36 <param name="lasamplerange" value="0.005"/>
37 <param name="lasamplestep" value="0.005"/>
38 </node>
39 </launch>
```

关于Gmapping算法中众多可配置参数的详细用法
请参考官方文档：<http://wiki.ros.org/gmapping>

8.1 Gmapping算法

代码实操建议： 计算机开发经验靠日积月累，绝非一朝一夕，遇事不慌，放平心态

■ Gmapping原理分析

■ Gmapping源码解读

■ Gmapping安装与运行

环境：Ubuntu18.04+ROS melodic

- ① GMapping安装
- ② GMapping在线运行
- ③ GMapping离线运行

#录制数据集，并保存文件为gmapping_xiihoo.bag (需要先启动真实机器人上的传感器)
roslaunch gmapping slam_gmapping_xiihoo.launch

#启动建图 (需要打开use_sim_time参数)
roslaunch gmapping slam_gmapping_xiihoo.launch

#播放数据集
roslaunch gmapping_xiihoo.bag

#启动rviz
rviz

slam_gmapping_xiihoo.launch启动文件：

```
1 <launch>
2   <param name="use_sim_time" value="true"/>
3
4   <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
5     <remap from="scan" to="/scan"/>
6     <param name="base_frame" value="base footprint"/>
7     <param name="map_frame" value="map"/>
8     <param name="odom_frame" value="odom"/>
9
10    <param name="map_update_interval" value="5.0"/>
11    <param name="maxOrange" value="16.0"/>
12    <param name="sigma" value="0.05"/>
13    <param name="kernelSize" value="1"/>
14    <param name="lstep" value="0.05"/>
15    <param name="astep" value="0.05"/>
16    <param name="iterations" value="5"/>
17    <param name="lsigma" value="0.075"/>
18    <param name="ogain" value="3.0"/>
19    <param name="lskip" value="0"/>
20    <param name="srr" value="0.1"/>
21    <param name="srt" value="0.2"/>
22    <param name="str" value="0.1"/>
23    <param name="stt" value="0.2"/>
24    <param name="linearUpdate" value="1.0"/>
25    <param name="angularUpdate" value="0.5"/>
26    <param name="temporalUpdate" value="3.0"/>
27    <param name="resampleThreshold" value="0.5"/>
28    <param name="particles" value="30"/>
29    <param name="xmin" value="-1.0"/>
30    <param name="ymin" value="-1.0"/>
31    <param name="xmax" value="1.0"/>
32    <param name="ymax" value="1.0"/>
33    <param name="delta" value="0.05"/>
34    <param name="llsamplerange" value="0.01"/>
35    <param name="llsamplestep" value="0.01"/>
36    <param name="lasamplerange" value="0.005"/>
37    <param name="lasamplestep" value="0.005"/>
38  </node>
39 </launch>
```

内容概要

8.1 Gmapping算法

8.2 Cartographer算法

8.3 LOAM算法

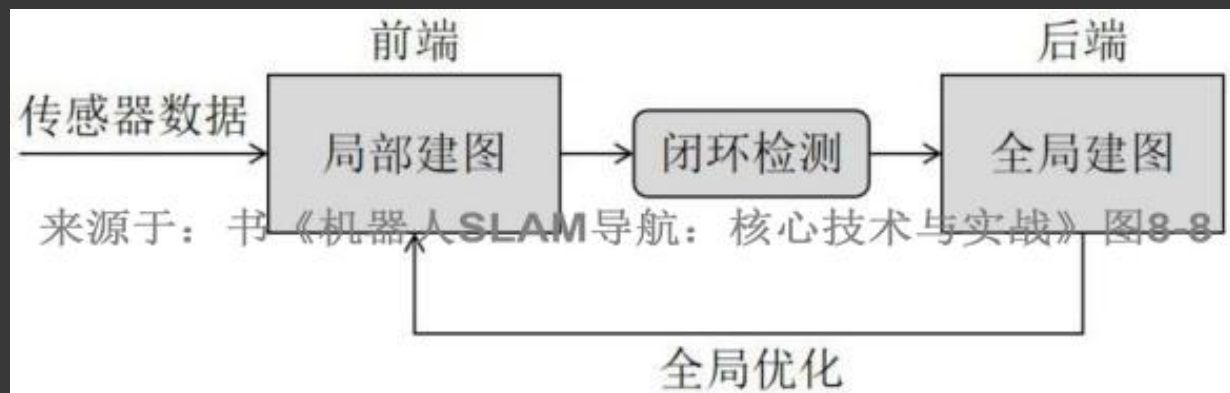
8.2 Cartographer算法

为什么基于滤波方法的Gmapping不能构建大规模地图？

地图很大时，粒子点更新的计算量和实时性有问题

基于优化方法的SLAM为何能构建大规模地图呢？

优化方法，分为局部建图过程、闭环检测、非实时要求的全局优化，计算量和实时性进行了有效的平衡



8.2 Cartographer算法

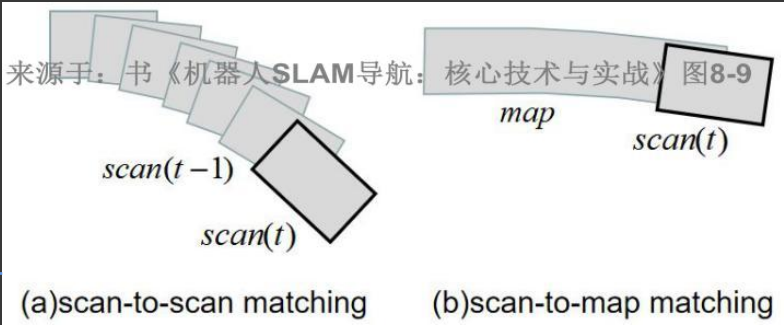
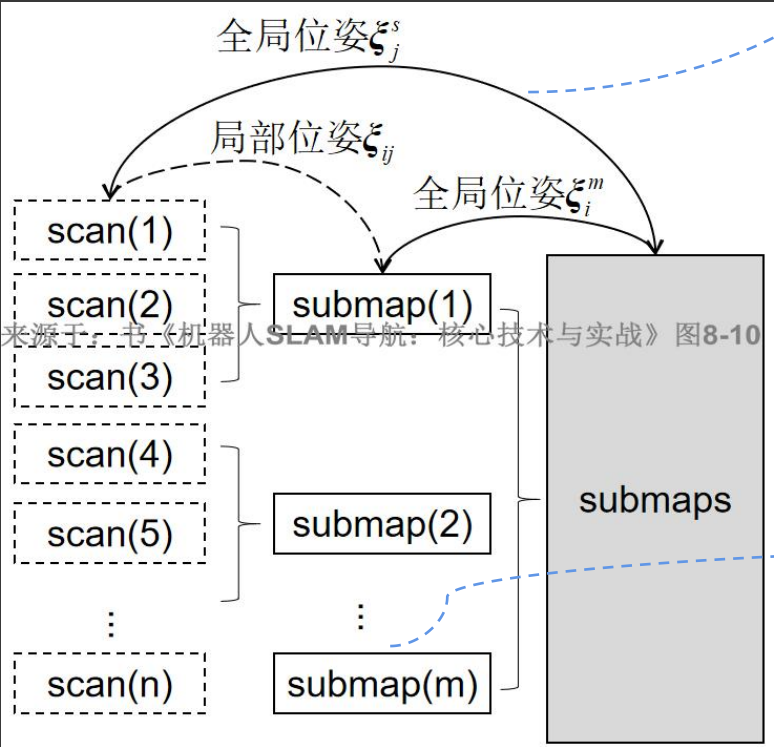
- Cartographer原理分析

■ Cartographer源码解读

■ Cartographer安装与运行
- ① 局部建图

② 闭环检测

③ 全局建图

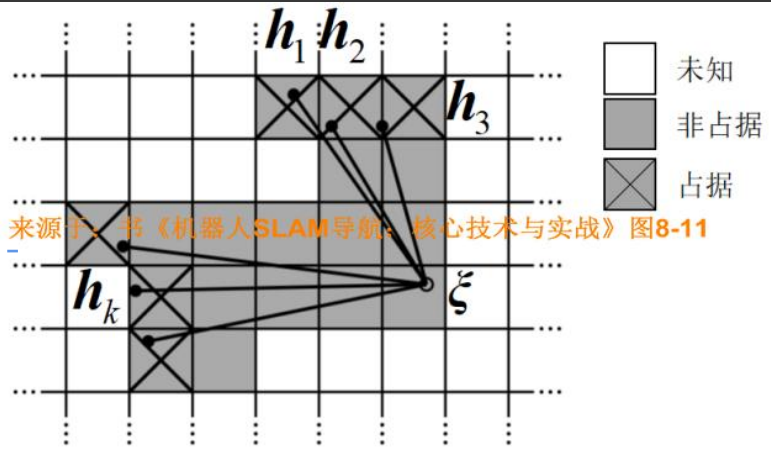


- 扫描匹配（数据关联）：
- Scan-to-scan matching
 - Scan-to-map matching
 - Pixel-accurate scan matching

$$M_{new}(x) = \begin{cases} P_{hit}, & \text{当 } state(x) = \text{hit} \text{ 时} \\ P_{miss}, & \text{当 } state(x) = \text{miss} \text{ 时} \end{cases}$$

$$M_{new}(x) = \begin{cases} clamp(odds^{-1}(odds(M_{old}(x)) \cdot odds(P_{hit}))), & \text{当 } state(x) = \text{hit} \text{ 时} \\ clamp(odds^{-1}(odds(M_{old}(x)) \cdot odds(P_{miss}))), & \text{当 } state(x) = \text{miss} \text{ 时} \end{cases}$$

其中, $odds(prob) = \frac{prob}{1 - prob}$



思考：
从栅格地图更新机制，分析其如何降低动态障碍物的干扰问题？

8.2 Cartographer算法

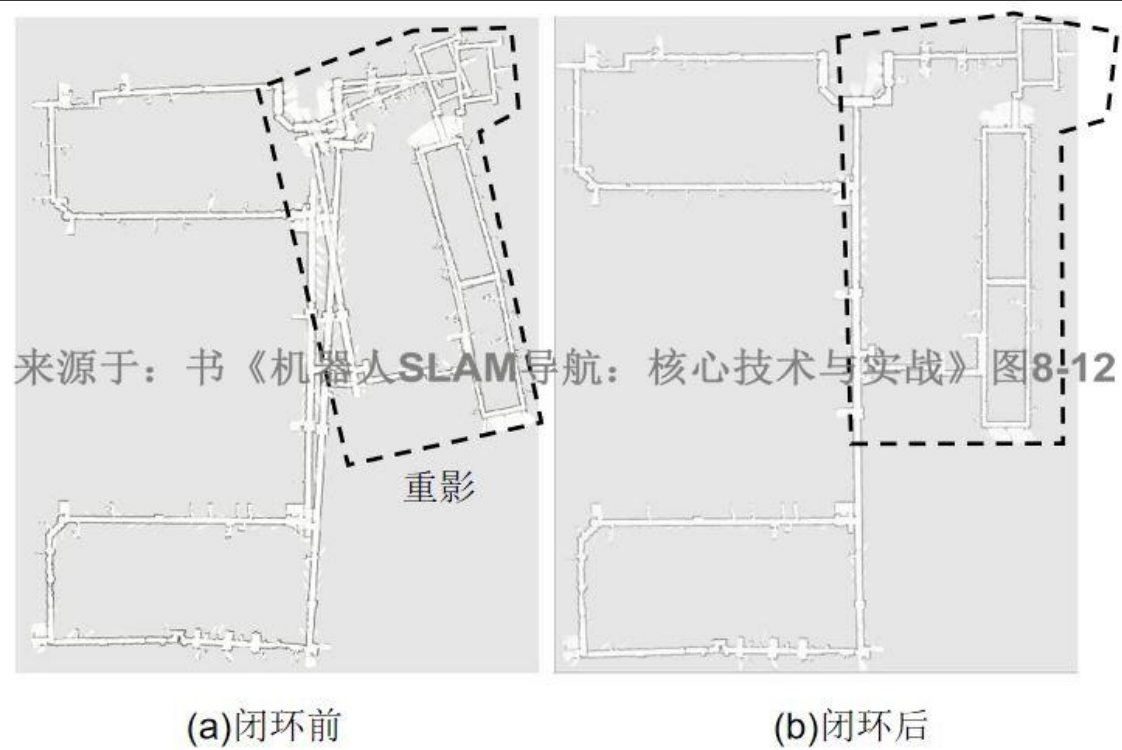
- Cartographer原理分析

■ Cartographer源码解读

■ Cartographer安装与运行
- ① 局部建图

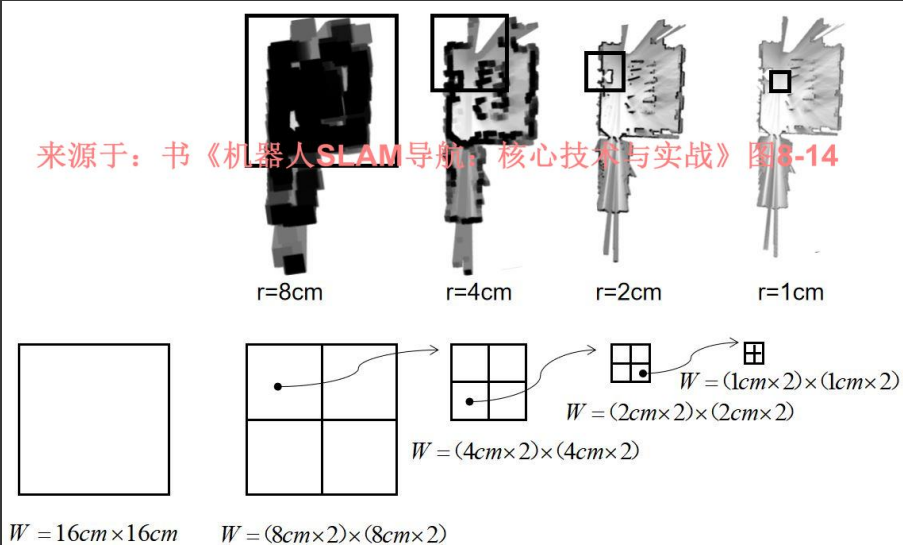
② 闭环检测

③ 全局建图



- 闭环检测：
- 暴力搜索匹配 Pixel-accurate scan matching
 - 分支界定匹配 Auto Pixel-accurate scan matching

分支界定匹配



- * **暴力搜索匹配**可认为是**分支界定匹配**处于**最高分辨率**情况下的特例
- * 分支界定属于**广度优先搜索**

8.2 Cartographer算法

- Cartographer原理分析
 - Cartographer源码解读
 - Cartographer安装与运行
- ① 局部建图

② 闭环检测

③ 全局建图

闭环检测成功后，会触发后端全局优化，并更新全局地图

所有雷达扫描帧对应的机器人全局位姿： $\mathbf{E}^s = \{\xi_j^s\}, j = 1, 2, \dots, n$

所有局部子图对应的全局位姿： $\mathbf{E}^m = \{\xi_i^m\}, i = 1, 2, \dots, m$

雷达扫描数据在局部子图中的局部位姿： ξ_{ij}

位姿约束构建

$$\operatorname{argmin}_{\mathbf{E}^m, \mathbf{E}^s} \frac{1}{2} \sum_{ij} \rho(E^2(\xi_i^m, \xi_j^s; \Sigma_{ij}, \xi_{ij}))$$

其中，

$$E^2(\xi_i^m, \xi_j^s; \Sigma_{ij}, \xi_{ij}) = e(\xi_i^m, \xi_j^s; \xi_{ij})^T \Sigma_{ij}^{-1} e(\xi_i^m, \xi_j^s; \xi_{ij})$$
$$e(\xi_i^m, \xi_j^s; \xi_{ij}) = \xi_{ij} - \begin{bmatrix} R_{\xi_i^m}^{-1} \bullet [t_{\xi_i^m} - t_{\xi_j^s}] \\ \xi_{i;\theta}^m - \xi_{j;\theta}^s \end{bmatrix}$$

非线性优化

Ceres-Solver

8.2 Cartographer算法

代码阅读技巧：程序调用流程 + 数据调用流程

- Cartographer原理分析

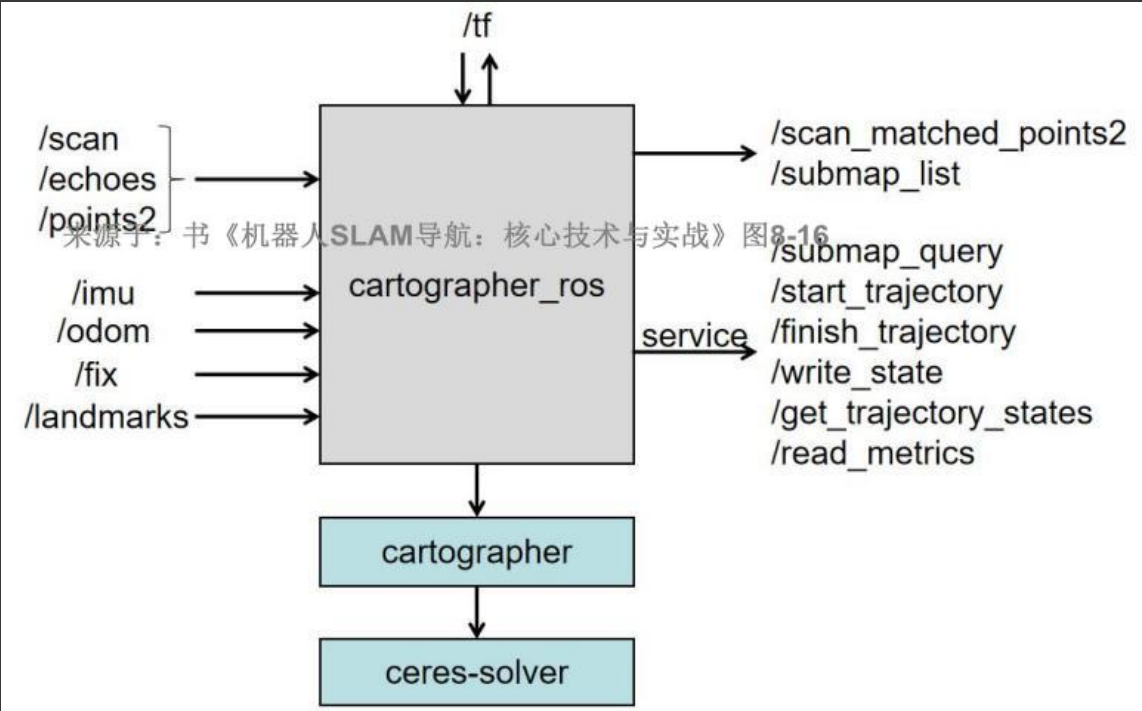
■ Cartographer源码解读

■ Cartographer安装与运行
- ① Cartographer代码框架

② cartographer_ros功能包

③ cartographer核心库

④ Ceres-Solver非线性优化库



https://github.com/cartographer-project/cartographer_ros
<https://github.com/cartographer-project/cartographer>
<https://github.com/ceres-solver/ceres-solver>

8.2 Cartographer算法

代码阅读技巧：程序调用流程 + 数据调用流程

- Cartographer原理分析

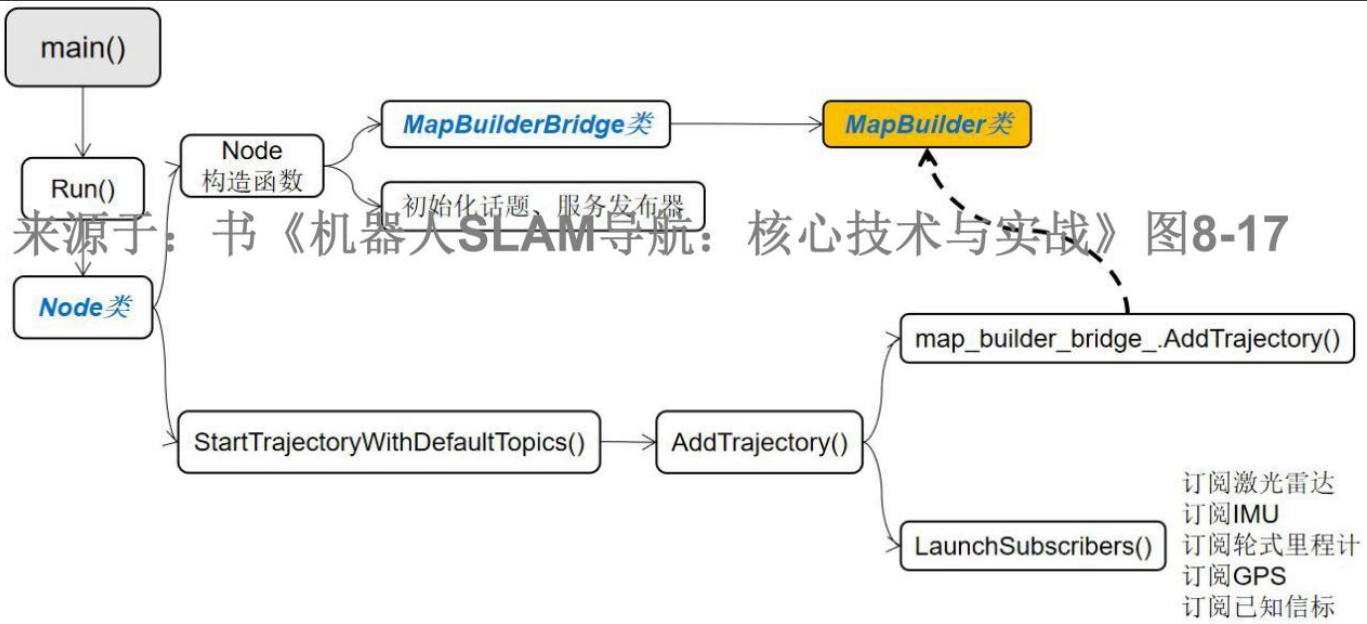
■ Cartographer源码解读

■ Cartographer安装与运行
- ① Cartographer代码框架

② cartographer_ros功能包

③ cartographer核心库

④ Ceres-Solver非线性优化库



cartographer_ros程序调用流程

cartographer_ros功能包中包含的节点

节点	源码	功能
cartographer_assets_writer	assets_writer_main.cc ros_map_writing_points_processor.h ros_map_writing_points_processor.cc	利用已完成轨迹精细化处理地图
cartographer_node	node_main.cc	建图主节点
cartographer_offline_node	offline_node_main.cc	利用数据包离线建图
cartographer_occupancy_grid_node	occupancy_grid_node_main.cc	将 cartographer 中的 submaps 地图转换成 ROS 中的 grid 地图并发布。
cartographer_pbstream_to_ros_map	pbstream_to_ros_map_main.cc	*.pbstream 文件到 *.pgm 文件地图格式转换
cartographer_pbstream_map_publisher	pbstream_map_publisher_main.cc	读取 *.pbstream 文件并转换成 ROS 中的 grid 地图并发布
...

8.2 Cartographer算法

代码阅读技巧：程序调用流程 + 数据调用流程

- Cartographer原理分析

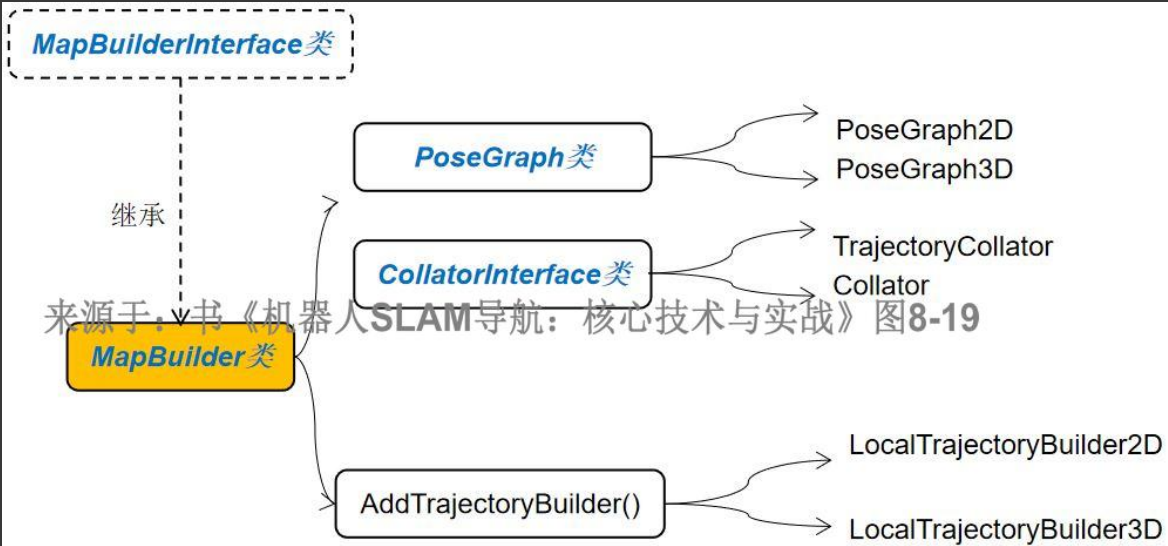
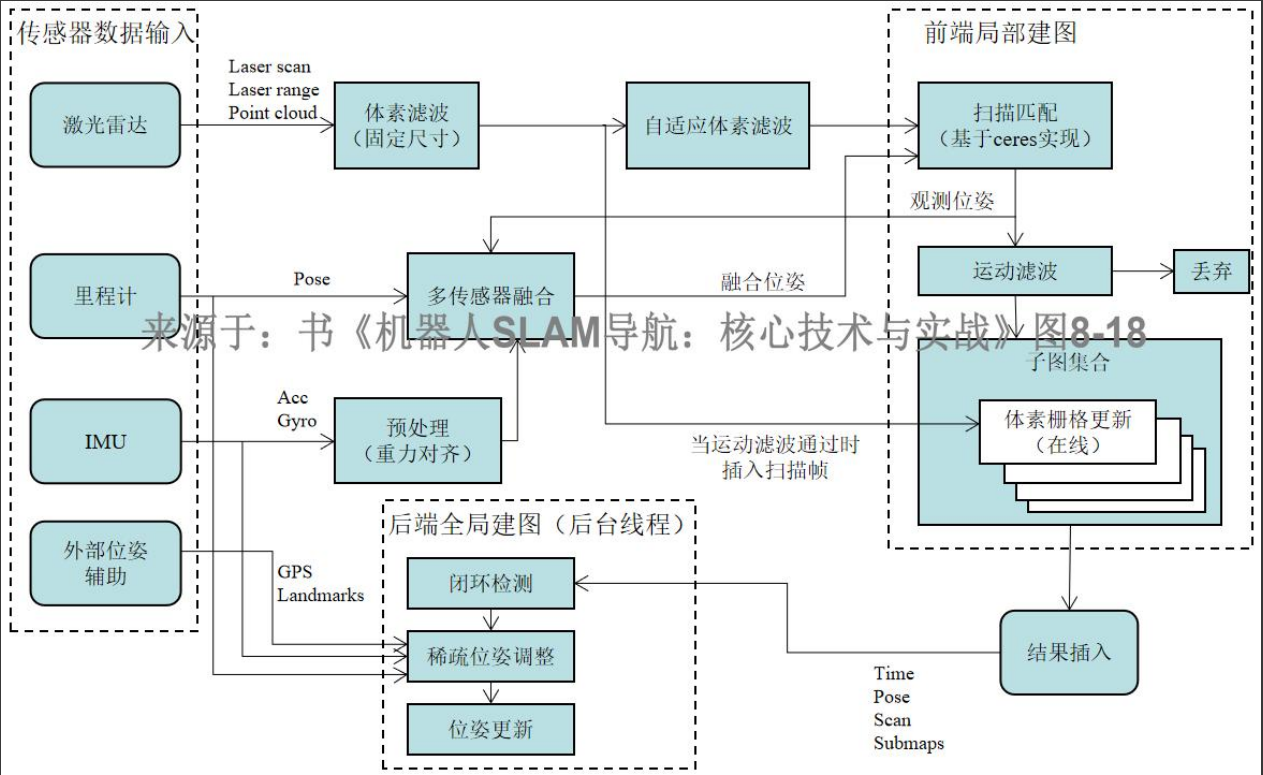
■ Cartographer源码解读

■ Cartographer安装与运行
- ① Cartographer代码框架

② cartographer_ros功能包

③ cartographer核心库

④ Ceres-Solver非线性优化库



8.2 Cartographer算法

代码阅读技巧：程序调用流程 + 数据调用流程

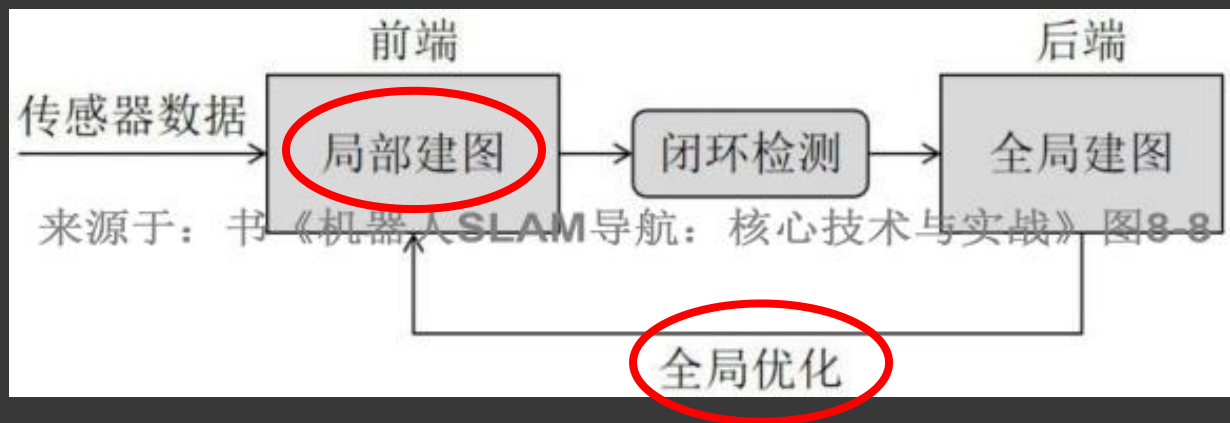
■ Cartographer原理分析

■ Cartographer源码解读

■ Cartographer安装与运行

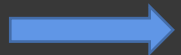
- ① Cartographer代码框架
- ② cartographer_ros功能包
- ③ cartographer核心库
- ④ Ceres-Solver非线性优化库

- ① 执行局部优化
- ② 执行全局优化



8.2 Cartographer算法

代码阅读技巧： 程序调用流程 + 数据调用流程

- Cartographer原理分析
 - Cartographer源码解读 
 - Cartographer安装与运行
- ① Cartographer代码框架
 - ② cartographer_ros功能包
 - ③ cartographer核心库
 - ④ Ceres-Solver非线性优化库

```
int main(int argc, char** argv)
{
    google::InitGoogleLogging(argv[0]);

    // 指定求解的未知数的初值, 这里设置为5.0
    double initial_x = 5.0;
    double x = initial_x;

    // 建立Problem
    Problem problem;

    // 建立CostFunction (残差方程)
    CostFunction* cost_function = new AutoDiffCostFunction<CostFunctor, 1, 1>(new CostFunctor);
    problem.AddResidualBlock(cost_function, NULL, &x);

    // 求解方程
    Solver::Options options;
    options.linear_solver_type = ceres::DENSE_QR;
    options.minimizer_progress_to_stdout = true;
    Solver::Summary summary;
    Solve(options, &problem, &summary);

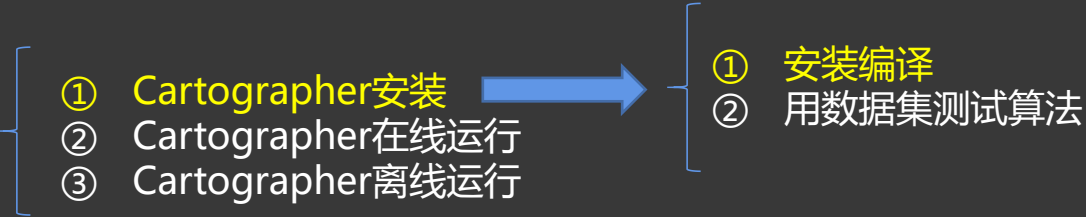
    std::cout << summary.BriefReport() << "\n";
    std::cout << "x : " << initial_x << " -> " << x << "\n";
    return 0;
}
```

8.2 Cartographer算法

代码实操建议： 计算机开发经验靠日积月累，绝非一朝一夕，遇事不慌，放平心态

- Cartographer原理分析
- Cartographer源码解读
- Cartographer安装与运行

环境：Ubuntu18.04+ROS melodic



```
#安装编译工具
sudo apt-get update
sudo apt-get install -y python-wstool python-rosdep ninja-build

#创建存放cartographer_ros的专门工作空间
mkdir catkin_ws_carto
cd catkin_ws_carto
wstool init src
#下载自动安装脚本
wstool merge -t src
https://raw.githubusercontent.com/googlecartographer/cartographer_ros/master/cartographer_ros.rosinstall
#执行下载
wstool update -t src

#安装依赖项
src/cartographer/scripts/install_proto3.sh
#如果报错，可以先将已有sources.list删除
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO} -y

#编译和安装
catkin_make_isolated --install --use-ninja
```

修改后的src/.rosinstall文件：

```
- git:
  local-name: cartographer
  uri: https://github.com/cartographer-project/cartographer.git
  version: 1.0.0
- git:
  local-name: cartographer_ros
  uri: https://github.com/cartographer-project/cartographer_ros.git
  version: 1.0.0
- git:
  local-name: ceres-solver
  uri: https://github.com/ceres-solver/ceres-solver.git
  version: 1.13.0
```


8.2 Cartographer算法

代码实操建议： 计算机**开发经验**靠日积月累，绝非一朝一夕，遇事不慌，放平心态

■ Cartographer原理分析

■ Cartographer源码解读

■ Cartographer安装与运行

环境：Ubuntu18.04+ROS melodic

① Cartographer安装

② Cartographer在线运行

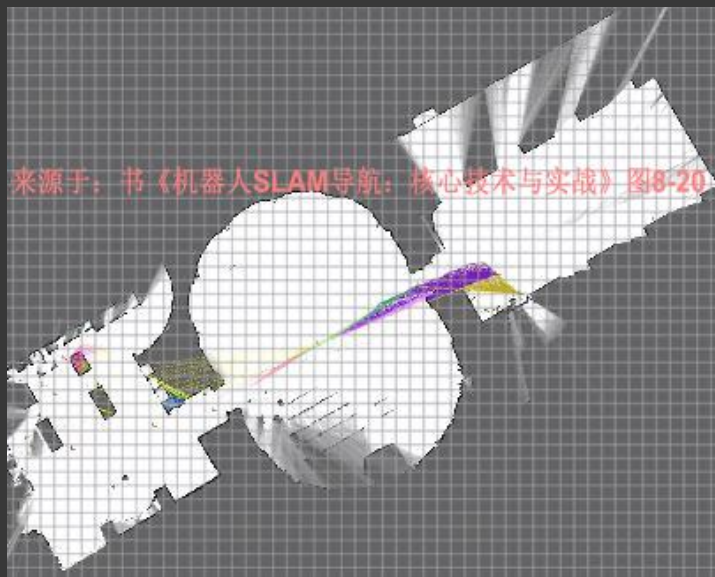
③ Cartographer离线运行

① 安装编译

② 用数据集测试算法

```
source ~/catkin_ws_carto/install_isolated/setup.bash
#下载2D建图测试数据集
wget -P ~/Downloads https://storage.googleapis.com/cartographer-public-data/bags/backpack_2d/cartographer_paper_deutsches_museum.bag
#启动2D建图
roslaunch cartographer_ros demo_backpack_2d.launch bag_filename:=${HOME}/Downloads/cartographer_paper_deutsches_museum.bag
```

Cartographer数据集测试效果



8.2 Cartographer算法

代码实操建议： 计算机**开发经验**靠日积月累，绝非一朝一夕，遇事不慌，放平心态

■ Cartographer原理分析

■ Cartographer源码解读

■ Cartographer安装与运行

环境：Ubuntu18.04+ROS melodic

- ① Cartographer安装
- ② Cartographer在线运行
- ③ Cartographer离线运行

```
#启动激光雷达
roslaunch ydlidar my_x4.launch
#启动底盘，并发布轮式里程计
roslaunch xiihoo_bringup minimal.launch
#启动IMU
roslaunch xiihoo_imu imu.launch

#重新编译整个工作空间，使配置文件修正生效
cd ~/catkin_ws_carto
catkin_make_isolated --install --use-ninja
#启动建图
source ~/catkin_ws_carto/install_isolated/setup.bash
roslaunch cartographer_ros xiihoo_mapbuild.launch

#首次使用键盘遥控，需要先安装对应功能包
sudo apt install ros-melodic-teleop-twist-keyboard
#启动键盘遥控
roslaunch teleop_twist_keyboard teleop_twist_keyboard.py
#启动rviz
rviz

#保存地图
rosservice call /write_state /home/ubuntu/map/carto_map.pbstream
```

xiihoo_mapbuild.launch启动文件：

```
1 <launch>
2   <node name="cartographer_node" pkg="cartographer_ros"
3     type="cartographer_node" args="
4       -configuration_directory $(find cartographer_ros)/configuration_files
5       -configuration_basename xiihoo_mapbuild.lua"
6     output="screen">
7   <remap from="scan" to="/scan" />
8   <remap from="imu" to="/imu" />
9   <remap from="odom" to="/odom" />
10  </node>
11
12  <node name="cartographer_occupancy_grid_node" pkg="cartographer_ros"
13    type="cartographer_occupancy_grid_node" args="
14      -resolution 0.05
15      -publish_period_sec 1.0" />
16 </launch>
```


8.2 Cartographer算法

代码实操建议： 计算机**开发经验**靠日积月累，绝非一朝一夕，遇事不慌，放平心态

■ Cartographer原理分析

■ Cartographer源码解读

■ Cartographer安装与运行

环境：Ubuntu18.04+ROS melodic

- ① Cartographer安装
- ② Cartographer在线运行
- ③ Cartographer离线运行

关于Cartographer算法中众多可配置参数的详细用法请参考官方文档：
<https://google-cartographer-ros.readthedocs.io>
<https://google-cartographer.readthedocs.io>

xiihoo_mapbuild.lua配置文件：

```
1 include "map_builder.lua"
2 include "trajectory_builder.lua"
3
4 options = {
5   map_builder = MAP_BUILDER,
6   trajectory_builder = TRAJECTORY_BUILDER,
7   map_frame = "map",
8   tracking_frame = "imu_link",
9   published_frame = "odom",
10  odom_frame = "odom",
11  provide_odom_frame = false,
12  publish_frame_projected_to_2d = false,
13  use_odometry = true,
14  use_nav_sat = false,
15  use_landmarks = false,
16  num_laser_scans = 1,
17  num_multi_echo_laser_scans = 0,
18  num_subdivisions_per_laser_scan = 10,
19  num_point_clouds = 0,
20  lookup_transform_timeout_sec = 0.2,
21  submap_publish_period_sec = 0.3,
22  pose_publish_period_sec = 5e-3,
23  trajectory_publish_period_sec = 30e-3,
24  rangefinder_sampling_ratio = 1.,
25  odometry_sampling_ratio = 1.,
26  fixed_frame_pose_sampling_ratio = 1.,
27  imu_sampling_ratio = 1.,
28  landmarks_sampling_ratio = 1.,
29 }
30
31 MAP_BUILDER.use_trajectory_builder_2d = true
32 TRAJECTORY_BUILDER_2D.num_accumulated_range_data = 10
33
34 TRAJECTORY_BUILDER_2D.min_range = 0.20
35 TRAJECTORY_BUILDER_2D.max_range = 16.0
36 TRAJECTORY_BUILDER_2D.submaps.num_range_data = 50
37 TRAJECTORY_BUILDER_2D.use_imu_data = true
38 TRAJECTORY_BUILDER_2D.imu_gravity_time_constant = 10.0
39 TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = false
40 TRAJECTORY_BUILDER_2D.ceres_scan_matcher.translation_weight = 10
41 TRAJECTORY_BUILDER_2D.ceres_scan_matcher.rotation_weight = 40
42 POSE_GRAPH.constraint_builder.max_constraint_distance = 4.
43
44 return options
```

8.2 Cartographer算法

代码实操建议： 计算机**开发经验**靠日积月累，绝非一朝一夕，遇事不慌，放平心态

■ Cartographer原理分析

■ Cartographer源码解读

■ Cartographer安装与运行

环境：Ubuntu18.04+ROS melodic

- ① Cartographer安装
- ② Cartographer在线运行
- ③ Cartographer离线运行

由于用cartographer_ros提供的/write_state方法保存的地图是*.pbstream的格式，而要在后续的自主导航中使用这个地图，我们需要将其转换为ROS中通用的GridMap格式。

地图格式转换：

```
#重新编译整个工作空间，使配置文件修正生效
cd ~/catkin_ws_carto
catkin_make_isolated --install --use-ninja
#启动地图格式转换
source ~/catkin_ws_carto/install_isolated/setup.bash
roslaunch cartographer_ros xiihoo_pbstream2rosmap.launch pbstream_filename:=/home/ubuntu/map/carto_map.pbstream map_filestem:=/home/ubuntu/map/carto_map
```

xiihoo_pbstream2rosmap.launch启动文件：

```
1 <launch>
2 <node name="cartographer_pbstream_to_ros_map_node" pkg="cartographer_ros"
3   type="cartographer_pbstream_to_ros_map" args="
4     -pbstream_filename $(arg pbstream_filename)
5     -map_filestem $(arg map_filestem)"
6   output="screen">
7 </node>
8 </launch>
```

思考：

请结合代码谈谈Cartographer地图格式与ROS默认地图格式的区别是什么？


8.2 Cartographer算法

代码实操建议： 计算机开发经验靠日积月累，绝非一朝一夕，遇事不慌，放平心态

- Cartographer原理分析
- Cartographer源码解读

■ Cartographer安装与运行

环境：Ubuntu18.04+ROS melodic

- 
- ① Cartographer安装

② Cartographer在线运行

③ Cartographer离线运行

cartographer_ros功能包中包含的节点

节点	源码	功能
cartographer_assets_writer	assets_writer_main.cc ros_map_writing_points_processor.h ros_map_writing_points_processor.cc	利用已完成轨迹精细化处理地图
cartographer_node	node_main.cc	建图主节点
cartographer_offline_node	offline_node_main.cc	利用数据包离线建图
cartographer_occupancy_grid_node	occupancy_grid_node_main.cc	将 cartographer 中的 submaps 地图转换成 ROS 中的 grid 地图并发布。
cartographer_pbstream_to_ros_map	pbstream_to_ros_map_main.cc	*.pbstream 文件到 *.pgm 文件地图格式转换
cartographer_pbstream_map_publisher	pbstream_map_publisher_main.cc	读取 *.pbstream 文件并转换成 ROS 中的 grid 地图并发布
...

来源于：书《机器人SLAM导航：核心技术与实战》表8-1

内容概要

8.1 Gmapping算法

8.2 Cartographer算法

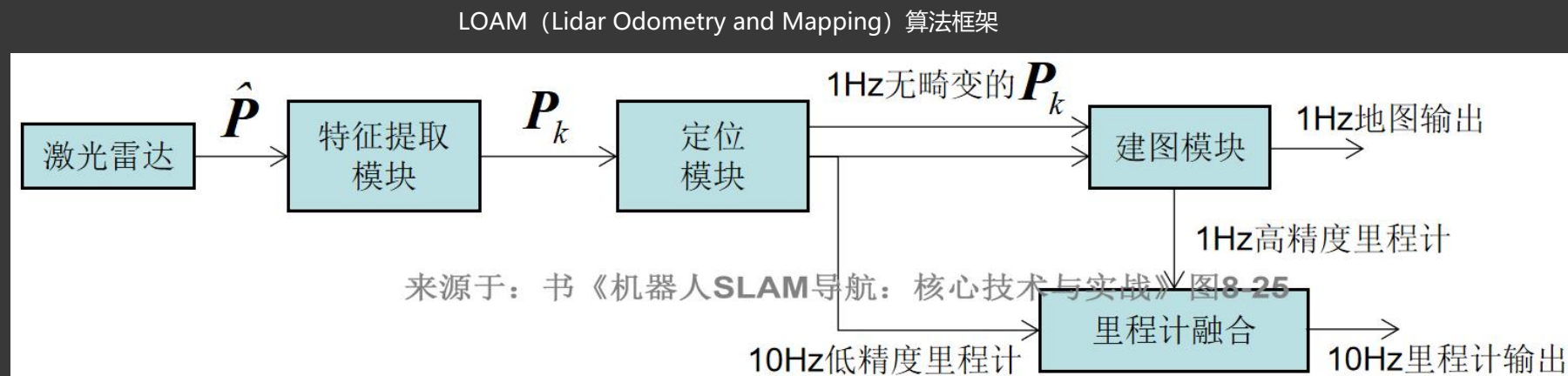
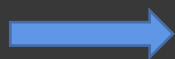
8.3 LOAM算法

8.3 LOAM算法

Gmapping	单线激光雷达	2D地图
Cartographer	单线/多线激光雷达	2D地图
LOAM	多线激光雷达	3D地图

8.3 LOAM算法

- LOAM原理分析
- LOAM源码解读
- LOAM安装与运行



■ 特征提取模块

LOAM从雷达点云中提取特征点corner和surface

■ 定位模块

利用scan-to-scan方法对相邻两帧雷达点云中的特征点进行匹配

■ 建图模块

利用scan-to-map方法进行高精度定位，该方法以前面低精度的里程计作为位姿初始值，将校正后的雷达特征点云与地图进行匹配

■ 里程计融合

以1Hz的高精度里程计为基准，利用10Hz的低精度里程计对其进行插值

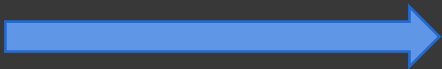
8.3 LOAM算法

代码阅读技巧：程序调用流程 + 数据调用流程

- LOAM原理分析
- LOAM源码解读
- LOAM安装与运行

- 特征提取模块
 - 定位模块
 - 建图模块
 - 里程计融合

LOAM的4个ROS节点及其接口



节点	订阅话题	发布话题
scanRegistration	/imu/data /multi_scan_points	/velodyne_cloud_2 /laser_cloud_sharp /laser_cloud_less_sharp /laser_cloud_flat /laser_cloud_less_flat /imu_trans
laserOdometry	/velodyne_cloud_2 /laser_cloud_sharp /laser_cloud_less_sharp /laser_cloud_flat /laser_cloud_less_flat /imu_trans	/laser_cloud_corner_last /laser_cloud_surf_last /velodyne_cloud_3 /laser_odom_to_init
laserMapping	/laser_cloud_corner_last /laser_cloud_surf_last /velodyne_cloud_3 /laser_odom_to_init /imu/data	/laser_cloud_surround /velodyne_cloud_registered /aft_mapped_to_init
transformMaintenance	/laser_odom_to_init /aft_mapped_to_init	/integrated_to_init

8.3 LOAM算法

代码实操建议： 计算机**开发经验**靠日积月累，绝非一朝一夕，**遇事不慌，放平心态**

- LOAM原理分析
- LOAM源码解读
- **LOAM安装与运行**

由于LOAM原作者（Ji Zhang）已经将其开源代码关闭了，现在已经无法找到原版源码了。另外就是原版LOAM代码还存在诸多问题，后来的开发者基于原版LOAM推出来多种改进版本，比较流行的有以下几种：

- https://github.com/laboshinl/loam_velodyne

(loam_velodyne版本代码简洁层次清晰，并且最接近原版LOAM)

- <https://github.com/HKUST-Aerial-Robotics/A-LOAM>

(A-LOAM版本用Eigen和Ceres-Solver对原版LOAM代码结构进行了简化，使得代码非常适合初学者入门学习)

- <https://github.com/RobustFieldAutonomyLab/LeGO-LOAM>

(LeGO-LOAM版本为原版LOAM添加了闭环检测，并利用GTSAM进行后端全局优化，大大提高了建图稳定性)

- 例程源码下载: https://github.com/xiihoo/Books_Robot_SLAM_Navigation
- 课件PPT下载: www.xiihoo.com

敬请关注,长期更新...

下集预告