**IDEA**

算法对比：WLS，EKF，FGO

赫尔默特方差+ AMHE本质上就是滑动窗口

目前的FGO问题：

计算成本显著增加；

定位精度仍有上限，残余误差依然较大；

无法完全解决异常值的“长尾”问题，算法虽然能抑制大部分显著的异常值，但无法完全将误差模型修正为标准的高斯分布，仍有一部分非高斯噪声残留；

Geman McClure 函数的核宽 (kernel width) 是“根据数据实验性选择的”。 在GNC的迭代过程中，控制参数 θ 的缩放因子被设为 1.4，但作者承认 **“如何明智地选择缩放因子仍然是一个悬而未决的问题；**

**依赖于批处理，影响实时性（滑动窗口解决即可，赫尔默特方差和AMHE）；**

**算法退化问题。**

MyThinking：

问题分解，将一个大的、非凸的复杂问题，分解为一系列在凸空间内的、更小、更易于处理的子问题。一个长时间的 FGO 问题可以被分解为多个时间段上的子图。每个子图可以被独立或并行优化，从而大幅降低单个优化问题的维度。可以先用一个计算代价小但精度较低的全局求解器（类似 Liang 论文中的 MAPF 引导）来确定每个子图之间的边界约束（即上一段的终点状态是下一段的起点状态），然后再调用 FGO-GNC 对每个子图进行精细优化。

Zhang et al. (2025) 在路径规划中，利用失败的搜索尝试来构建一个“估计信息集”（Estimated Informed Set），从而为后续的搜索提供一个更优的初始引导。FGO-GNC 是一个迭代优化过程，一个好的初始值至关重要。可以借鉴这个思路，在运行昂贵的 FGO-GNC 之前，先用一个快速但粗糙的定位算法（如 WLS）得到一个初始轨迹，或者直接利用上一时刻 FGO 的优化结果作为当前时刻的初始值。这种“智能初始化”能让优化器从一个更好的起点开始，从而减少迭代次数，并降低陷入局部最优的风险。

障碍函数的融入，将一个纯粹的**状态估计算法**，转变为一个能够进行**安全验证的系统。**

多普勒几何强度”因子 (Doppler Geometry Factor)：LEO多普勒测量对用户速度的解算能力极度依赖于其几何构型。一颗从头顶飞过的LEO卫星和一颗从地平线升起的卫星，它们提供的速度信息维度是完全不同的。我们可以设计一种新的因子，它不直接约束位置或速度，而是约束解的几何强度。在每个时刻，我们计算当前所有LEO卫星构成的“多普勒几何精度因子”（D-DOP）。然后，在FGO中加入一个 “几何约束因子”，该因子会惩罚那些导致D-DOP恶化的速度解。换句话说，优化器会被引导去寻找一个既满足测量残差，又在几何上最“可信”的速度状态。这为“松散”的速度状态节点之间提供了强大的隐式约束。（有点儿像我之前看的那个DGDOP）

信号机会的拓扑推断 (Topology Inference from Signal Opportunity)

核心思想: 在恶劣环境中，信号的出现和消失本身就蕴含了丰富的几何信息。

具体做法:

“首次捕获”因子 (First-Acquisition Factor): 当一颗刚刚从地平线升起的LEO或GNSS卫星首次被捕获时，我们可以推断出用户与这颗卫星之间大概率没有遮挡。这可以在FGO中形成一个\*\*“视线通畅”的几何约束\*\*，帮助限定用户的位置。

“信号中断”因子 (Signal-Loss Factor): 当一颗高仰角卫星的信号突然中断，很可能是进入了遮挡区（如隧道、高楼背后）。如果有粗略的地图模型，这个“信号消失”事件可以作为一个非常强的\*\*“位置约束因子”，将用户的位置限定在地图上的遮挡区域内。这是一种利用负信息（Negative Information）\*\*来增强FGO约束的全新思路。

**图的稀疏化 (Graph Sparsification)**: 当因子图变得过于庞大和稠密时，可以根据信息论的准则，**剪掉**那些信息量较低的边（因子）。例如，可以计算每个因子对整体信息矩阵的贡献度，周期性地移除贡献度最低的5%的因子，从而在不显著牺牲精度的前提下，有效控制计算量。**剪枝？**确实有说法，和滑动窗口优化可能还是要做区分一些？

噪声可以建设为随机游走模型，用多个因子来约束其随时间的变化，而不是简单地认为每一时刻的IMU测量都只与当前时刻的Bias有关。

**NOSTRA**论文虽然领域是贝叶斯优化（Bayesian Optimization），但其核心思想对于优化因子图（FGO）具有非常重要的启发意义。

FGO 本质上也是一个优化问题：寻找一组状态（轨迹）来最好地解释所有的测量数据。NOSTRA 论文要解决的核心问题与 FGO 遇到的困境惊人地相似：**如何在数据稀疏、稀少且充满噪声的情况下，高效地找到最优解？**

这篇论文为 FGO 的优化提供了两个极具价值的、可借鉴的核心思想：

**启发 1：信任域采样 (Trust Region Sampling) —— 从“全局蛮力搜索”到“重点区域精细挖掘”**

**NOSTRA 的做法：** 传统优化方法会在整个设计空间内盲目搜索。而 NOSTRA 则更加“聪明”，它不会浪费计算资源在那些“不太可能有好结果”的区域。它通过一个巧妙的机制，动态地在整个解空间中划定出一个\*\*“信任域” (Trust Region)**。这个信任域被认为是**最有可能包含帕累托最优解（Pareto frontier）的区域\*\*。然后，算法会集中火力在这个信任域内进行更密集的采样和优化，从而大大加快收敛速度。（其实就类似之前说的剪枝）

**对 FGO 的启发：** 当前的 FGO 求解器（如 Levenberg-Marquardt）在每次迭代时，本质上是在当前解附近的一个局部区域内寻找下一个更优的解。但是，这个“局部区域”的选择通常是基于简单的数学梯度（LAMBDA算法好像也是这玩意儿），缺乏更高层面的引导。

我们可以借鉴 NOSTRA 的思想，为 FGO 引入一个\*\*“状态空间信任域”\*\*：

**识别关键区域**: 在 FGO 优化前，可以先用一个计算成本极低的粗略算法（如 WLS）跑一遍，或者分析卫星几何构型（DOP值），识别出那些**不确定性可能非常大**的区域（例如，卫星信号严重遮挡的时段）。

**构建信任域**: 我们可以定义一个信任域，即**优先优化那些不确定性较小的、解比较可靠的区域**。

**分阶段优化策略**:

**阶段一（稳定阶段）**: 在信任域内（即信号良好、不确定性低的时段），FGO 可以采用更激进的优化策略，快速求解，并将这部分已经收敛的解作为整个图的“锚点”。

**阶段二（挑战阶段）**: 对于信任域之外的“坏数据”区域，FGO 可以采取更保守、更鲁棒的优化策略（例如，启用更强的鲁棒核函数，增加更多迭代次数），并利用第一阶段已经固定的“锚点”作为强大的先验信息来约束这部分的求解。

**解决了什么核心问题？** 这种方法将一个庞大而复杂的 FGO 问题，分解为了“先易后难”的两步走策略。它避免了让求解器从一开始就在最困难的数据上挣扎，而是先稳住基本盘，再集中优势兵力去攻克难点，从而**提高了整体求解的收敛性和鲁棒性**。

*FGO中的信任域：优先优化高质量数据区域，将其作为锚点来约束低质量区域的解*

**启发 2：先验知识注入 (Prior Knowledge Integration) —— 从“无知”的拟合到“有经验”的推理**

**NOSTRA 的做法：** 在数据稀少且有噪声时，单纯依赖数据进行模型拟合（例如，高斯过程的超参数估计）很容易出错。NOSTRA 强调，工程师通常对自己实验设备或仿真模型的噪声水平有一个**先验的认知**（Prior Knowledge）。NOSTRA 将这个先验知识（例如，噪声方差大概在什么范围）作为额外信息**注入**到高斯过程的训练中，从而在数据不足时，也能得到一个更合理、更稳定的代理模型。

**对 FGO 的启发：** FGO 的权重（信息矩阵）通常是基于一些固定的模型（如卫星仰角模型）来计算的，这本质上是一种“硬编码”的先验。我们可以让这个过程变得更灵活、更智能。

**可配置的先验因子**: 我们可以为 FGO 增加一系列\*\*“软性先验因子” (Soft Prior Factors)**。这些因子不是来自直接的测量，而是来自**领域知识\*\*：

**运动学先验**: “车辆的加速度通常不会超过5 m/s²”，“车辆的转弯半径通常大于5米”。这些都可以作为软约束因子加入图中。

**场景先验**: “在这条高速公路上，车辆大概率会以80-120 km/h的速度行驶”。这个速度范围可以作为一个先验因子，在 GNSS 信号质量差时提供强大的约束。

**LEO 卫星特性先验**: “Orbcomm 卫星的轨道误差通常在 XX 米量级，并且主要体现在 cross-track 方向上”。这个先验可以直接用来初始化 LEO 卫星轨道误差变量的协方差。

**贝叶斯化的参数估计**: FGO 中的很多参数，比如 IMU 的偏差、接收机钟差的漂移率，通常是被当作待优化的确定性变量。我们可以将它们**贝叶斯化**，即为它们设定一个先验分布。例如，我们可以根据传感器的手册，为 IMU 零偏的初始值设定一个高斯先验。在优化的过程中，算法将结合这个先验和数据，计算出一个后验分布，而不仅仅是一个点估计。

**解决了什么核心问题？** 通过注入先验知识，FGO 不再是一个单纯的“数学优化器”，而变成了一个\*\*“贝叶斯推理引擎”\*\*。它能够在测量数据本身不足以提供确定解时，利用先验知识做出最合理的推断，这对于处理 GNSS/LEO 信号频繁中断的挑战性场景至关重要，**极大地增强了算法在信息稀疏环境下的鲁棒性**。

**总结来说，NOSTRA 论文给 FGO 带来的最大启发是两种思维模式的转变：**

**从全局优化到重点突破**: 不要试图一次性解决所有问题，而是识别出问题的“薄弱环节”和“关键支撑点”，采用不同的策略来处理。

**从数据驱动到知识+数据双轮驱动**: 不要完全依赖有噪声的测量数据，而要将人类已知的、关于世界如何运作的先验知识，以数学化的方式融入到优化框架中。

**GraphGNSSLib-LEO**

fgo\_leo

-> src -> GraphGNSSLib\_LEO\_V1.2/ -> docker(存镜像的地方)、img(一些基础图片)、support\_files(理论文档)、nlosexclution(存放.msg文件，在使用catkin\_make时根据这些文件自动生成C++头文件和Python脚本，并放到devel目录中，二次使用catkin时就是将这些文件打包成可执行程序；.msg文件定义了项目中所有节点之间通信时使用的自定义消息类型，主要还是用来规范消息标准，保证消息能正常传递；其余的python函数主要是坐标转换，以及北斗/GNSS组合定位、ROS监听/订阅节点、经纬度转KML文件等等)、rviz\_satellite(在轨迹下面叠上可视化的谷歌地图，包括对GPS节点的订阅处理等)、global\_fusion(核心代码所在)

-> result

-> devel -> 可执行文件

-> build

整体运行流程就是，调用对应的launch文件，载入文件中提前设定好的数据，启动各个节点去运行指定的代码，并将运行结果储存在文件提前设定好的保存目录中，同时启动RVIZ可视化。（注：有些launch文件里面是把RVIZ那部分给注释掉了，还有fake gps位置的发布，导致RVIZ不启动/地图资源无法加载，意思是先调试，正式运行的时候再把注释去掉）

接下来主要讲解global\_fusion下的核心代码。

# 0. ROS的一些概念

1. 主题（Topic）

主题是命名的通信通道，用于在节点之间传递消息。可以把它想象成一个广播电台的频道：每个主题都有一个唯一的名称；主题具有特定的数据类型，定义了可以传输的消息格式；多个节点可以同时向同一个主题发布消息；多个节点也可以同时订阅同一个主题来接收消息。

2. 发布（Publish）

发布是指节点向特定主题发送消息的行为：例如，节点创建了一个发布者，它会向名为"WLS\_spp"的主题发布nav\_msgs::Odometry类型的消息，当调用publish()方法时，消息被发送到该主题，任何订阅该主题的节点都会收到这个消息。

3. 订阅（Subscribe）

订阅是指节点接收特定主题消息的行为：例如，节点订阅了名为"/gnss\_preprocessor\_node/GNSSPsrCarRov1"的主题，队列大小为500，意味着系统会缓存最多500条未处理的消息，当有新消息到达时，会自动调用gnss\_raw\_msg\_callback回调函数。

// 创建订阅者，指定主题名称、队列大小和回调函数

gnss\_raw\_sub = nh.subscribe("/gnss\_preprocessor\_node/GNSSPsrCarRov1", 500,&gnssSinglePointPositioning::gnss\_raw\_msg\_callback, this);

再举个例子：

文本

AI 生成的内容可能不正确。

# 1.launch

ROS启动文件。对应src中的代码，主要包括：

1.evo：用于启动评估工具EVO(Evaluationof Odometry and SLAM)。它包括2个节点，首先是启动evo\_traj脚本。它接收一个ROSbag文件作为输入，并将其中多个话题(/FGO,/WLS)的轨迹数据导出为TUM格式的.txt文件，保存在/tmp/目录下。然后启动evo\_ape脚本，用于计算绝对位姿误差(AbsolutePoseError)。它会比较导出的FGO轨迹(/tmp/FGO.txt)和WLS轨迹(/tmp/WLS.txt)，并生成一个误差图(--plot)。

2.gnss\_estimator：psr\_spp.launch启动global\_fusion包中的psr\_spp\_node可执行文件，启动一个基于**伪距的单点定位**(SPP,SinglePointPositioning)算法节点，并自启完成RVIZ可视化；psr\_doppler\_fusion.launch启动global\_fusion包中的psr\_doppler\_fusion\_node可执行文件，实现基础的**伪距+多普勒的FGO算法**；psr\_doppler\_car\_rtk.launch启动global\_fusion包中的psr\_doppler\_car\_rtk\_node可执行文件，启动**FGO-RTK**；psr\_doppler\_car\_rtk\_dyna.launch则是启动global\_fusion包中的psr\_doppler\_car\_rtk\_dyna\_node可执行文件，调用ceresSolvers的DynamicAutoDiffCostFunction，允许因子图中的状态变量（特别是整周模糊度）数量在每个时刻动态变化，这使得算法能更灵活地适应不同时刻卫星数量的变化，实现**更鲁棒的RTK定位**；psr\_spp\_dyna\_auto.launch是一个自动化测试或演示用的启动文件，它组合了数据回放和伪距单点定位算法。

3.gnss\_preprocessor：主要由gnss\_preprocessor\_node节点（这个节点由/src/ gnss\_preprocessor.cpp初始化创建）控制，gnss\_preprocessor.launch主要是启动nlosexclusion包中的gnss\_preprocessor的节点来启动GNSS预处理器，加载默认配置参数；dataublox\_TST20190428.launch和dataublox\_TST20200603.launch，主要用于启动数据回放和预处理测试数据，相关部件包括rviz，tf2等都配置完毕，前者输入和输出对应dataset的gps\_solution\_TST2文件夹，后者则对应gps\_solution\_TST文件夹；rosbag\_generator.launch是启动nlosexclusion包中的rosbag\_generator节点，将特定格式（如NMEA、RTCM或自定义二进制格式）的GNSS日志文件转换成可以被ROS系统其他部分使用的.bag文件。

4.whampoa：这部分应该是作者新加的模块，数据是新的。data\_Whampoa\_20210521\_GNSS\_only.launch是单GNSS定位，调用gnss\_preprocessor\_node节点，指明数据文件路径（流动站，基准站的观测值，GPS/北斗的星历文件），并读取上述文件，进行数据预处理，设置定位模式为kinematic模式，并使用双频数据，将WLS和RTK导出GNSS\_only\_WLS\_result.csv和GNSS\_only\_RTK\_result.csv文件中；data\_Whampoa\_20210521\_GNSSLEO.launch就比较强，它同时运行和对比**三种不同的定位算法（**单GNSS的伪距定位psr\_spp\_gnssleo\_node，单GNSS的FGO优化psr\_doppler\_fusion\_node和GNSS+LEO融合的FGO优化psr\_doppler\_fusion\_gnssleo\_node**），**输出文件存储在GNSSLEO\_\_SPP\_trajectoryllh\_psr\_dop\_fusion.csv，GNSSonly\_FGO\_trajectoryllh\_psr\_dopp\_fusion.csv和GNSSLEO\_FGO\_trajectoryllh\_psr\_dopp\_fusion.cs里面。同时启用rviz可视化，加载gnss\_positioning.rviz配置文件，用于实时显示和对比上述三种算法的定位结果。除此之外还有rostopicpub发布假GPS位置（用于加载卫星地图）和static\_transform\_publisher来发布一个静态的map到base\_link的TF变换，确保RViz中的坐标系正确建立。最后是leo\_raw\_publisher.launch文件，这个文件用于**读取和发布模拟的LEO（低轨卫星）原始数据**，并将其与真实的GNSS数据结合。通过启动leo\_rover\_raw\_publisher\_node节点，读取对应的CSV文件，并将其中的模拟LEO观测数据以ROS消息的形式发布出去。同时，启动gnss\_preprocessor\_node来处理真实的GNSS观测数据。启动gnss\_leo\_msg\_combination\_node节点。该节点的功能是订阅gnss\_preprocessor\_node发布的GNSS数据和leo\_rover\_raw\_publisher\_node发布的LEO数据，并将两者合并成一个统一的数据流（可能是nlosexclusion::GNSS\_Raw\_Array消息），供后续的融合定位算法使用。

注意！必须是先在一个终端启用gnss\_preprocessor或者whampoa文件夹中有关GNSS原始观测值数据的**预处理**launch文件，它首先会启动ROS Master（如果尚未运行），并启动 gnss\_preprocessor\_node 节点，该节点开始读取硬盘上的数据文件，并将处理后的GNSS测量值发布到特定的ROS话题上。**只有这个节点成功运行后，数据才会在ROS网络中开始流通。**

然后再打开gnss\_estimator中对应各个算法融合的launch节点。这个新启动的节点会立即在ROS网络中查找并订阅它所需要的数据话题。由于第一个节点已经在发布数据，因此这个文件夹中的相关算法节点就能成功接收到数据并开始执行其融合定位算法。

# 2.ThirdParty

第三方库。主要包括LLH->ECEF，笛卡尔坐标系转换等文件。

# 3.Include

大多数文件头文件都在这儿写，建议对照src里面的业务代码的调用情况来理解（其实感觉把src放在include之前比较合理）。

## 3.1 gnss\_tools.h

基础函数，提供坐标系转换，数据提取，数据预处理，误差建模，RTK/DD辅助，加权最小二乘法函数等。

## 3.2 pseudorange\_factor.h

基于 Google 的 Ceres Solver 库实现。定义了与伪距测量相关的因子，主要是标准伪距因子和双差伪距因子。

## 3.3 carrier\_phase\_factor.h

基于 Google 的 Ceres Solver 库实现。定义了与载波相位测量相关的因子，主要是标准载波相位因子和双差载波相位因子。

## 3.4 doppler\_factor.hpp

基于 Google 的 Ceres Solver 库实现。定义了与**多普勒频移 (Doppler)** 测量相关的因子。将不同时刻的状态节点连接起来形成时序约束。

## 3.5 psr\_doppler\_fusion.h

上层应用，完成伪距和多普勒因子的组合和优化的求解，调用 Ceres Solver 求解，并保存优化结果。

## 3.6 psr\_doppler\_car\_rtk.h

进阶应用，完成伪距、载波相位(均为双差)和多普勒因子的组合优化求解(RTK)。调用 Ceres Solver 求解得到浮点解后，调用LAMBDA函数搜索固定解。

## 3.7 psr\_doppler\_car\_rtk\_dyna

终极版本，额外使用了 Ceres Solver 的 **DynamicAutoDiffCostFunction** (动态自动微分代价函数)。

# 4.rviz

**ROS 可视化工具 RViz 的配置文件**。它们的功能是**保存一个预先设定好的可视化界面布局**，让开发者可以一键加载，方便地查看和调试 ROS 节点发布的数据。用户无需每次打开 RViz 都手动添加和配置各种显示项。

**rtk.rviz**

这个配置文件主要用于可视化和对比RTK(Real-Time Kinematic)定位算法的结果，完成开源卫星地图的显示，并订阅/gps/fix话题来确定地图的中心位置。

(1)PNT(WLS):显示一个Odometry消息，表示通过加权最小二乘法(WLS)计算出的定位结果。它订阅/rtk\_estimator/pntpos\_odometry话题，并用红色箭头表示。

(2)RTKLIB(integer):显示来自RTKLIB库的整周模糊度固定解(integer)。它订阅/rtk\_estimator/rtk\_integer\_odometry话题，并用绿色箭头表示。

(3)FGO\_RTK:显示本项目因子图优化(FGO)的RTK结果。它订阅/FGO话题，并用蓝色箭头表示。

(4)Path:显示一条蓝色的路径，用于可视化/rtk\_estimator/global\_path话题发布的全局轨迹。

其他显示项如GroundTruth(地面真值)和RTKLIB(float)(RTKLIB浮点解)被配置但默认未启用(Enabled:false)。

**gnss\_positioning.rviz**

这个配置文件用于更广泛地对比多种不同的GNSS定位算法，包括WLS、RTK和因子图优化(FGO)的结果。

核心功能:

多算法对比:设计用于同时可视化多达五种不同的定位轨迹和状态，并用不同颜色加以区分，是进行算法评估和性能比较的核心工具。

在线地图:同样加载了在线地图(https://cartodb-basemaps-a.global.ssl.fastly.net/light\_all/{z}/{x}/{y}.png)作为背景。

统一坐标系:所有显示项都在map这个固定坐标系下进行显示。

主要显示项(Displays):

GNSS-LEO-WLSENURTKLIB(Green):显示融合了LEO(低轨卫星)数据的WLS定位结果，用绿色箭头表示。订阅/psr\_spp\_gnssleo\_node/WLS\_spp\_GNSS\_LEO话题。

GNSS-only-WLSENUGoGPS(Yellow):显示仅使用传统GNSS的WLS定位结果，用黄色箭头表示。订阅/gnss\_preprocessor\_node/WLSENUGoGPS话题。

GNSS-only-RTKENURTKLIB(Red):显示仅使用传统GNSS的RTK结果，用红色箭头表示。订阅/gnss\_preprocessor\_node/ENUSolutionRTK话题。

GNSS-only-WLSENURTKLIB(Blue):另一个仅使用GNSS的WLS结果，用蓝色箭头表示。订阅/gnss\_preprocessor\_node/WLSENURTKLIB话题。

GNSS-only-FGOPath:显示仅使用GNSS的因子图优化输出的轨迹，为紫色路径。订阅/FGOGlobalPath话题。

GNSS-LEO-FGOPath(Cyan):显示融合了LEO数据的因子图优化输出的轨迹，为青色路径。订阅/GNSS\_LEO\_FGOGlobalPath话题。

**psr\_spp\_LEO.rviz**

这个配置文件专注于可视化融合了LEO(低轨卫星)的伪距单点定位(SPP)结果，并与因子图优化的路径进行对比。

核心功能:

LEO融合可视化:主要用于展示和调试融合LEO卫星数据的WLS(加权最小二乘法)定位算法的效果。

谷歌卫星地图:加载了谷歌卫星地图(http://mt3.google.com/vt/lyrs=s&x={x}&y={y}&z={z})作为高清背景。

主要显示项(Displays):

WLS\_spp\_LEO(Purple):显示融合LEO数据的WLS单点定位结果，用绿色箭头表示(配置文件中名称为紫色，但颜色设置为绿色)。它订阅/psr\_spp\_gnssleo\_node/WLS\_spp\_GNSS\_LEO话题。

FGO\_path:显示因子图优化的结果路径，为红色路径。它订阅/FGOGlobalPath话题。

GroundTruth:配置了地面真值显示项，用于对比，但默认未启用。

综上所述，这三个 .rviz 文件是该项目进行算法开发、调试和成果展示的重要工具。它们通过将不同来源和算法的定位结果在同一个 3D 空间中（通常带有卫星地图背景）进行可视化，使得开发者能够非常直观地：

**1.对比精度**: 将不同算法的轨迹与地面真值或其他高精度结果进行比较。

**2.调试问题**: 观察轨迹的漂移、跳变等异常现象，从而定位算法问题。

**3.展示效果**: 清晰地展示不同算法（如 WLS vs RTK vs FGO，或 GNSS-only vs GNSS+LEO）之间的性能差异。

# 5.dataset

主要存着需要用到的数据集。

1.2021\_5021\_0607：主要包括hkqt测站的相关官方GNSS测站观测值文件，以及UranBav\_Whampoa…香港黄埔的城市峡谷的观测值数据集，运行程序时输出的GNSS\_only\_WLS\_result，rtklibResult.pos也在文件夹中。

2.gps\_solution\_TST：同上，hksc测站的数据集。

3.gps\_solution\_TST2：同上，hkcs测站的数据集，只不过时间是2020年6月3日了。

4.odometry\_data.txt：好像是里程计的数据？用来确定多普勒速度的。

# 6.RTKLIB

主要是从观测值中读取伪距，载波相位观测值等数据，以及调用LAMBDA等函数操作。建议直接看src，观察有哪些地方用到了这部分代码。

# 7.src

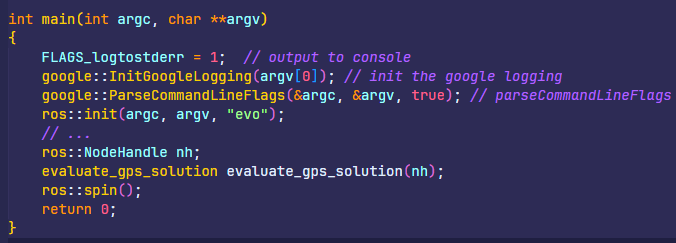
核心代码库。链接include文件夹中的各个头文件和函数完成核心任务。

## 7.0 tic\_toc.h

计时工具，测量代码段的执行时间。

## 7.1 evo

该程序的核心目标是**评估一个GNSS（全球导航卫星系统，如GPS）定位解的精度**。具体来说，它会读取一个由某种算法计算出的定位轨迹文件，然后将其与一个高精度的参考轨迹（地面真值或Ground Truth）进行比较。通过计算两者在每个时间点的空间位置误差，来量化待评估算法的性能。最终，程序会输出两个文件：一个是逐点的误差记录文件，另一个是包含两条轨迹坐标的对比文件，方便进行可视化分析。

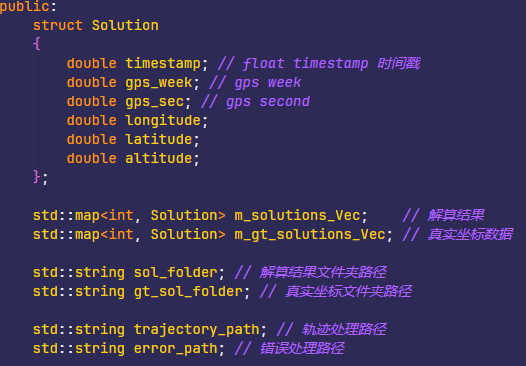


主函数代码主要是设置Google日志到控制台，初始化evo这个ROS节点，并创建节点句柄和GPS评估对象，最后则是启动ROS事件循环，等待回调函数执行。

文本

AI 生成的内容可能不正确。

主函数调用的evaluate\_gps\_solution函数的初始化变量部分。接下来是它的类内函数部分。



定义结构体和定位结果及其存储路径。

文本

AI 生成的内容可能不正确。

evaluate\_gps\_solution函数的构造函数和析构函数。图中获取文件的两个函数就不看了，直接看evaluatesol函数：

文本

AI 生成的内容可能不正确。

其中getMSE函数就是单纯的xyz平方开根号计算。

## 7.2 gnss\_preprocessor文件夹

该文件夹下主要包括两个文件。

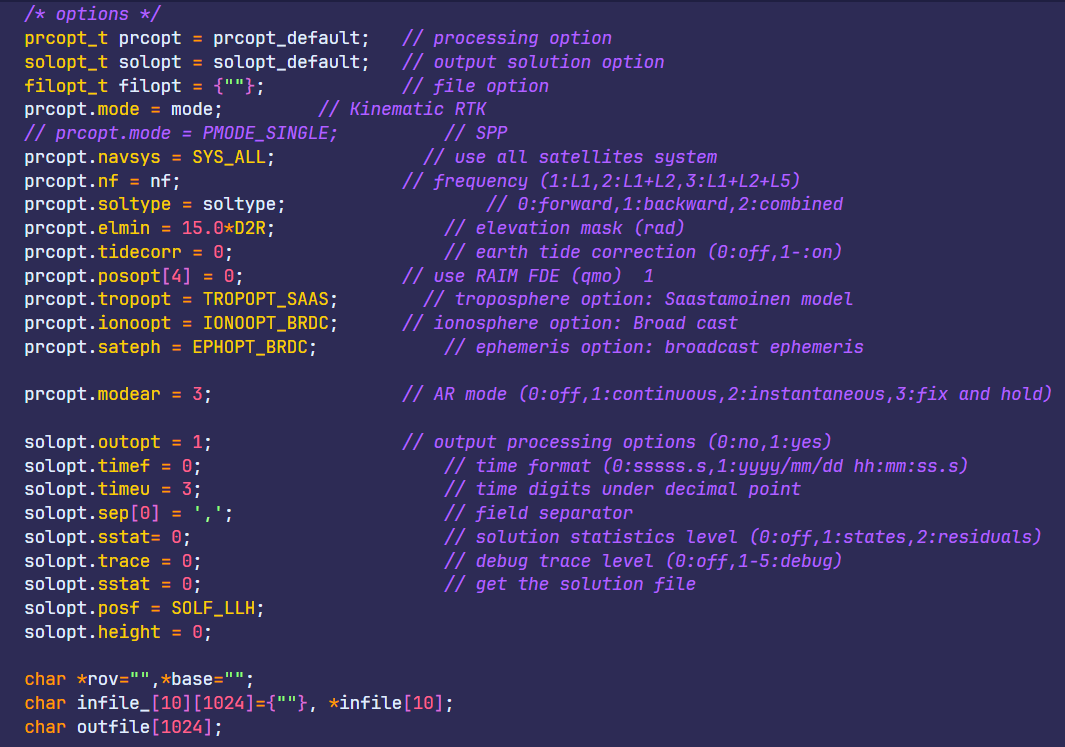
### 7.2.1 gnss\_preprocessor.cpp

GNSS数据预处理器的主要实现，其主要作用是解析RINEX格式的GNSS观测数据文件，并通过ROS话题发布GNSS原始测量数据。

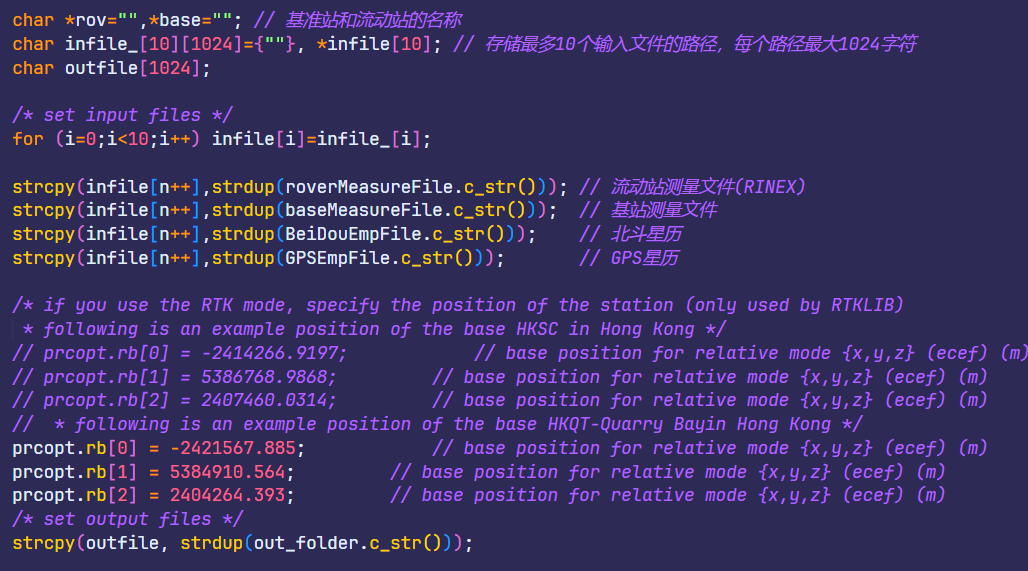
文本

AI 生成的内容可能不正确。

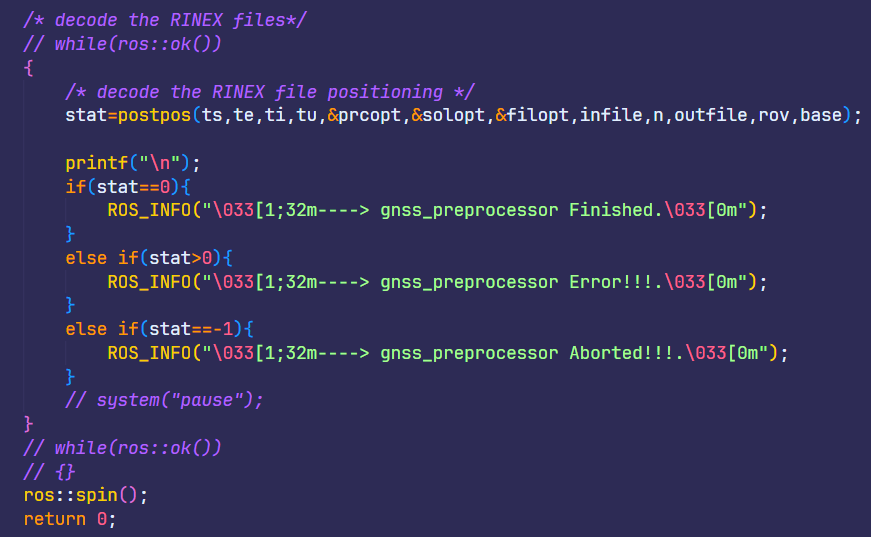
初始化ROS节点gnss\_preprocessor\_node，后面的launch文件大都是启动这个节点进行定位。



定义处理选项、定位解算和文件输出三个结构体，并分别赋初值。对理解结构体中的变量来说注释已经足够了，只有这个sstat不知道为啥连续初始化了两次。



这里是通过使用strdup创建字符串副本并用strcpy复制到infile数组中，确保每个输入文件都有独立的存储空间。然后定义接收机的真实坐标（这里用的是HKQT的数据，原本的是HKSC的数据）。这里的rb数组疑似是来自RTKLib的。



最后这部分是RINEX的解码函数。主要是导入处理开始、结束时间，时间间隔等参数，调用postpos函数（RTKLib的核心代码）执行GNSS后处理定位，并检查postpos函数的返回状态stat，等于0就说明成功了，否则报警。ros::spin();是ROS函数，使程序进入循环等待状态，处理ROS消息回调。

最后还有个显示消息的showmsg函数，以及settspan设置时间跨度函数，接收起始和结束时间参数。settime设置时间函数，接收时间参数：



### 7.2.2 rosbag\_generator.cpp

主要功能是**将原始GNSS数据添加到rosbag文件中，并与来自SPAN-CPT的ROS话题进行时间对齐**。（头文件调用了rtklib.h）

主要功能如下：

1.数据同步与整合：从三个不同的数据源接收数据

移动站GNSS测量数据 (/gnss\_preprocessor\_node/GNSSPsrCarRov1)

基准站GNSS测量数据 (/gnss\_preprocessor\_node/GNSSPsrCarStation1)

多普勒速度数据 (/gnss\_preprocessor\_node/GNSSDopVelRov1)

并使用时间同步器将这三个数据流进行同步处理

2.数据验证和过滤：

检查数据有效性（checkValidEpoch）

验证GNSS数据可用性（通过GNSS\_Tools.checkAvailability）

检查重复卫星数据（checkRepeating）

3.加权最小二乘定位计算：使用WeightedLeastSquare算法计算WLS解；将结果从ECEF坐标转换为ENU坐标。

4.数据重发布：

监听SPAN-CPT的INS数据作为触发信号；根据GPS时间戳查找并发布对应的GNSS原始数据；将处理后的数据发布到新的ROS话题中，供后续使用。

文本

AI 生成的内容可能不正确。

主函数部分和evo基本一致，只是这里初始化的ROS节点是rosbag\_generator\_node。然后进入rosbag\_generator类的分析：

文本

AI 生成的内容可能不正确。

这部分定义类内成员变量，接下来则是类内函数。

文本

AI 生成的内容可能不正确。

上述构造函数的主要作用包括：

1.初始化了多个订阅者，用于接收GNSS原始数据、基站数据和多普勒速度数据；

2.设置了时间同步器，确保三种数据在时间上同步；

3.注册了相应的回调函数来处理同步后的数据；

4.创建了多个发布者，用于发布处理后的数据；

5.设置了ENU坐标系的参考点；

6.从参数服务器获取处理时间范围参数。

文本

AI 生成的内容可能不正确。

这个回调函数的主要作用是当GNSS数据读取完成后（[finishGNSSReader](javascript:void(0))为true），根据SPAN-CPT设备提供的时间戳，查找并发布对应的移动站GNSS数据、基站GNSS数据和多普勒速度数据。这是一种同步机制，确保发布的数据与SPAN-CPT设备的时间戳对齐。

文本

AI 生成的内容可能不正确。

总的来说，这个函数负责接收GNSS和多普勒数据，进行时间同步检查，计算位置解算，并将数据存储以供后续处理使用。它在GNSS数据处理流程中起到了数据收集和初步处理的关键作用。

## 7.3 gnss\_estimator

重头戏，该文件夹下存放的就是各种定位的实现代码（说是实现代码，实际上属于业务代码？因为核心函数都在include下面，这部分主要是编排调用）。

### 7.3.1 psr\_spp.cpp（SPP定位）

这个程序是一个GNSS单点定位节点，实现了基于伪距测量的GNSS定位功能。它使用Ceres Solver优化库来解决非线性最小二乘问题，以获得接收机的最佳位置估计。

首先是调用RTKLIB头文件：

屏幕上有字

AI 生成的内容可能不正确。

然后是主函数：

文本

AI 生成的内容可能不正确。

形式和evo, rosbag\_generator.cpp大差不差，初始化的ros节点是psr\_spp\_node，主要还是核心函数gnssSinglePointPositioning的实现：

文本

AI 生成的内容可能不正确。

函数的成员变量部分。

文本

AI 生成的内容可能不正确。

函数的成员结构体，主要根据接收机状态计算预测伪距，并与实际观测值进行比较，生成用于优化的残差。它是GNSS定位算法中连接观测值和理论模型的关键桥梁，被Ceres优化库用于迭代求解接收机的最佳位置和钟差参数。

文本

AI 生成的内容可能不正确。

这个结构体和上个差不多，不过引入了ceres::DynamicAutoDiffCostFunction，允许动态调整参数块的数量和大小，且状态向量访问方式不同，使用双重索引state[keyIndex][i]而非单一索引state[i]，增加了keyIndex参数用于指定在参数数组中的位置，这种设计使得优化过程可以处理更复杂的情况，例如在FGO中包含多个时间步的状态变量时，可以根据需要动态选择特定时间步的状态进行计算。

文本

AI 生成的内容可能不正确。

这个工厂方法Create的主要功能是简化pseudorangeConstraint代价函数的创建过程；自动处理参数块的添加，确保优化器知道该代价函数依赖哪些状态变量；正确设置残差数量；返回配置好的代价函数，可以直接添加到Ceres优化问题中。

这种方法的优势在于它封装了创建复杂代价函数的所有步骤，使得在优化问题中添加新的伪距约束变得更加简单和不容易出错。通过使用动态自动微分代价函数，可以灵活地处理不同数量的状态变量，这对于处理因子图中多个时间步的优化问题特别有用。

文本

AI 生成的内容可能不正确。

构造函数和析构函数。这个构造函数主要完成以下初始化工作：

1.建立数据订阅：订阅GNSS预处理节点发布的伪距和载波相位数据。

2.启动处理线程：创建独立线程执行定位优化算法。

3.配置结果发布：设置WLS和FGO两种定位结果的发布通道。

4.初始化坐标参考：设置ENU坐标系的参考点，用于后续坐标转换。

文本

AI 生成的内容可能不正确。

这个回调函数主要完成以下功能：

1.数据接收与缓存：接收GNSS原始数据并存储到缓冲区和映射表中

2.WLS定位解算：使用加权最小二乘法计算接收机位置

3.坐标转换：将ECEF坐标转换为ENU坐标以便于理解和可视化

4.结果发布：将定位结果以Odometry消息的形式发布出去

5.日志记录：记录定位结果用于调试和分析

6.线程安全：使用互斥锁确保多线程环境下的数据安全

这是GNSS单点定位系统中的核心处理流程，实现了从原始观测数据到定位结果的完整处理链路。

最后则是基于ceres的定位优化算法的成员函数逻辑：

文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

这个函数实现了基于伪距测量的GNSS定位优化。它使用Ceres Solver对所有GNSS测量数据进行批量优化，通过最小化观测伪距与预测伪距之间的差异来估计接收机位置和时钟偏差。整个过程包括初始化参数、构建优化问题、求解优化问题以及结果输出等步骤。

至此，该函数逻辑结束。

### 7.3.2 psr\_spp\_gnssleo.cpp（低轨+GNSS组合SPP定位）

这个函数要对应psr\_spp去看，逻辑大差不差，但是增添了很多新函数，所以要展开说一下。

### 7.3.3 psr\_spp\_dyna\_auto.cpp（SPP+ceres）

psr\_spp.cpp 和 psr\_spp\_dyna\_auto.cpp的核心目标一致：都是利用ROS接收GNSS原始观测数据，并使用Ceres-Solver进行因子图优化来进行单点定位（Single Point Positioning, SPP）。最核心的区别在于它们构建Ceres优化问题时使用了不同类型的代价函数（Cost Function），这导致了状态变量管理和参数块添加方式的差异。psr\_spp.cpp实现基本的GNSS单点定位，使用标准Ceres自动微分代价函数，而psr\_spp\_dyna\_auto.cpp 专门演示DynamicAutoDiffCostFunction的使用，可以处理可变参数大小。其实psr\_spp.cpp里面注释掉的一部分代码替换回来就是psr\_spp\_dyna\_auto.cpp。暂时不知道最后系统调用的哪个，一会找找。

### 7.3.4 psr\_spp\_doppler\_fusion.cpp（FGO优化）

多普勒和伪距的FGO优化，经典的模型代码！

文本

AI 生成的内容可能不正确。

相较于普通spp代码，它引入了include中的psr\_doppler\_fusion.h头文件和ceres的ceres/dynamic\_autodiff\_cost\_function.h（自动微分代价函数）来完成解算。

屏幕上有字

AI 生成的内容可能不正确。

主函数代码依旧大差不差，初始化了ROS节点psr\_doppler\_fusion\_node。接下来从它定义的psr\_doppler\_fusion类开始看起。

文本

AI 生成的内容可能不正确。

首先是类的成员变量部分，和之前的函数差不多，只不过这个滑动窗口的值有点夸张了。

文本

AI 生成的内容可能不正确。

随后定义了一个类私有函数，主要是创建另外一个FactorGraph类的实例，并将ceres的相关配置参数传入进去。

文本

AI 生成的内容可能不正确。

接下来是构造函数。比较奇怪的是他因为换了数据集，所以话题名字也变了？可能要去看v1.0版本的gnss\_fgo\_path存在哪了。

文本

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

优化函数基本上都是调用FactorGraph类中规定的相关函数，而这个函数存储在Include中，具体实现逻辑可以跳过去看。这个函数是整个因子图优化系统的核心，它负责持续监控新数据、构建优化问题、执行优化计算、发布结果、记录日志、管理滑动窗口（这个没怎么看懂，是怎么管理的？）。

电脑萤幕的截图

AI 生成的内容可能不正确。

构造函数那部分有个时间同步器，这个就是里面说的回调函数的逻辑。当同时收到GNSS和多普勒数据时会调用此函数。逻辑和其他代码大差不差。这个回调函数是数据处理流水线的入口点，主要功能包括接收时间同步的GNSS和多普勒数据、存储数据到内存映射表中、输入数据到因子图优化器、实时计算并发布WLS解算结果、线程安全的数据访问控制。

至此，函数结束。其实调用的函数大部分都在include下面的头文件部分，要改的话应该也是在那边改。

### 7.3.5 psr\_doppler\_car\_rtk.cpp（FGO优化RTK）

利用双差载波相位观测值，伪距观测值和多普勒因子组合做FGO。头文件部分和FGO差不多，只不过换成了psr\_doppler\_car\_rtk.h。主函数也是对应初始化了ROS节点psr\_doppler\_car\_rtk\_node。主要区别是多定义了一个基准站的主题station\_gnss\_raw\_map；用FactorGraph添加因子的时候是用addDDPsrCarFactors添加双差伪距/载波相位观测值的因子；在回调函数gnssraw\_doppler\_msg\_callback中加了基准站数据的处理，别的都差不多。

原理上，状态向量多了一个载波整周模糊度，所以它是先求出一个包含小数模糊度的初始解，然后用LAMBDA算法固定为整数得到高精度解。

### 7.3.6 psr\_dopple\_car\_rtk\_dyna.cpp（FGO优化RTK+Ceres）

这个版本就是在上个代码的基础上改了头文件，外加使用了 Ceres 的 DynamicAutoDiffCostFunction。这种代价函数可以处理动态变化数量的参数，非常适合卫星数量不断变化的场景。不同的是，上个代码是使用预先定义大小的代价函数先求解浮点解然后直接求固定解，而这个是先求解浮点解，然后显式地获取浮点解的协方差矩阵，最后用上一步得到的协方差进行固定解的求解，不过这个代码的计算成本会更高。

### 7.3.7 psr\_doppler\_fusion\_gnssleo（FGO优化低轨+GNSS）

## 7.4 leo\_preprocessor

### 7.4.1 gnss\_leo\_msg\_combination.cpp

### 7.4.2 leo\_rover\_raw\_publisher.cpp

### 7.4.3 leo\_station\_raw\_publisher.cpp

### 7.4.4 rtkpos\_leo.cpp

### 7.4.5 rosbag\_align.py