9. Cartographer mapping algorithm

Cartographer: https://google-cartographer.readthedocs.io/en/latest/

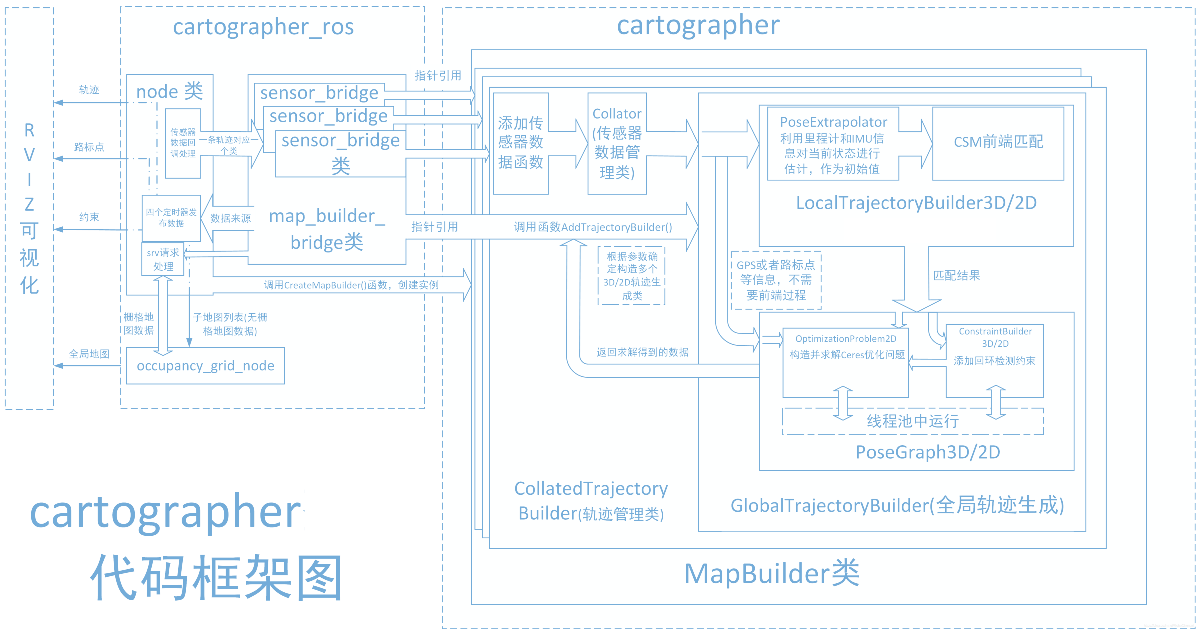
Cartographer ROS: https://google-cartographer-ros.readthedocs.io/en/latest/

map_server: https://wiki.ros.org/map_server

9.1. Introduction

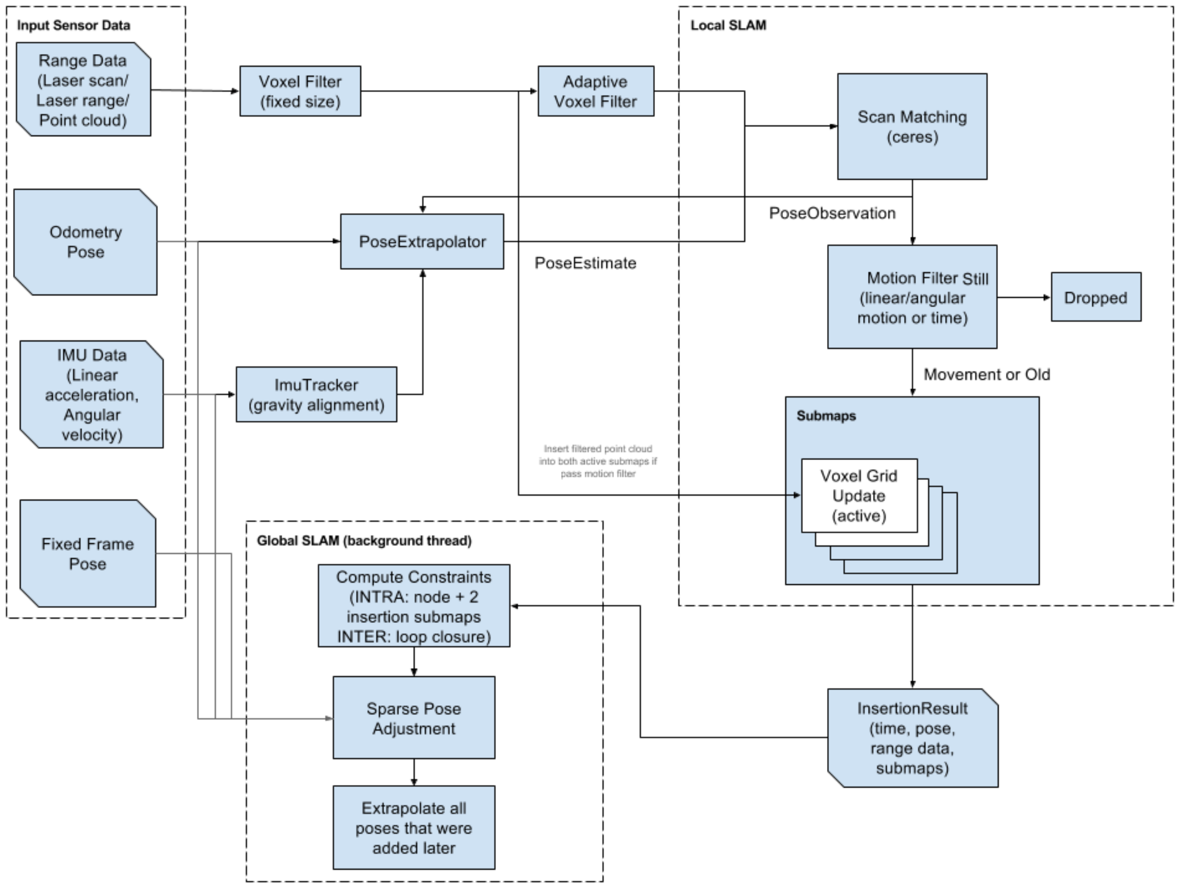
Cartographer is a 2D and 3D SLAM (simultaneous localization and mapping) library supported by Google's open source ROS system. A graph-building algorithm based on graph optimization (multi-threaded backend optimization, problem optimization of ceiling construction). Data from multiple sensors (such as LIDAR, IMU, and cameras) can be combined to simultaneously calculate the sensor's position and map the environment around the sensor.

The source code of cartographer mainly includes three parts: cartographer, cartographer_ros and ceres-solver (back-end optimization).



RVIZ可视化
cartographer_ros
轨迹
路标点
约束
全局地图
node 类
传感器数据回调处理
一条轨迹对应一个类
四个定时器发布数据
数据来源
srv请求处理
sensor_bridge
sensor_bridge
sensor_bridge类
map_builder_bridge类
指针引用
指针引用
调用CreateMapBuilder()函数，创建实例
栅格地图数据
子地图列表(无栅格地图数据)
occupancy_grid_node
cartographer
添加传感器数据函数
Collator(传感器数据管理类)
调用函数AddTrajectoryBuilder()
根据参数确定构造多个3D/2D轨迹生成类
返回求解得到的数据
PoseExtrapolator 利用里程计和IMU信息对当前状态进行估计，作为初始值
CSM前端匹配
LocalTrajectoryBuilder3D/2D
GPS或者路标点等信息，不需要前端过程
匹配结果
OptimizationProblem2D 构造并求解Ceres优化问题
ConstraintBuilder 3D/2D 添加回环检测约束
线程池中运行
PoseGraph3D/2D
CollatedTrajectory Builder(轨迹管理类)
GlobalTrajectoryBuilder(全局轨迹生成)
MapBuilder类
cartographer 代码框架图

- cartographer


Input Sensor Data
Range Data (Laser scan/ Laser range/ Point cloud)
Odometry Pose
IMU Data (Linear acceleration, Angular velocity)
Fixed Frame Pose
Voxel Filter (fixed size)
Adaptive Voxel Filter
PoseExtrapolator
ImuTracker (gravity alignment)
PoseEstimate
Local SLAM
Scan Matching (ceres)
PoseObservation
Motion Filter Still (linear/angular motion or time)
Dropped
Movement or Old
Submaps
Voxel Grid Update (active)
Insert filtered point cloud into both active submaps if pass motion filter
Global SLAM (background thread)
Compute Constraints (INTRA: node + 2 insertion submaps INTER: loop closure)
Sparse Pose Adjustment
Extrapolate all poses that were added later
InsertionResult (time, pose, range data, submaps)

Cartographer uses the mainstream SLAM framework, which is a three-stage structure of feature extraction, closed-loop detection, and back-end optimization. A certain number of LaserScans form a submap, and a series of submaps constitute the global map. The cumulative error in the short-term process of using LaserScan to construct a submap is not large, but the long-term process of using a submap to construct a global map will have a large cumulative error. Therefore, closed-loop detection is needed to correct the positions of these submaps. The basic unit of closed-loop detection is submap, closed-loop detection uses the scan_match strategy. The focus of cartographer is the creation of submap subgraphs that integrate multi-sensor data (odometry, IMU, LaserScan, etc.) and the implementation of the scan_match strategy for closed-loop detection.

- cartographer_ros

The cartographer_ros package runs under ROS and can accept various sensor data in the form of ROS messages. After processing, it can be published in the form of messages to facilitate debugging and visualization.

9.2. Use

Note: When building a map, the slower the speed, the better the effect (note that the rotation speed should be slower). If the speed is too fast, the effect will be poor.

According to different models, you only need to set the purchased model in [.bashrc], X1 (normal four-wheel drive) X3 (Mailun) Take X3 as an example

```
#Raspberry Pi 5 master needs to enter docker first, please perform this step
#If running the script into docker fails, please refer to ROS/07, Docker tutorial
~/run_docker.sh
```

Open the [.bashrc] file

```
sudo vim .bashrc
```

Find the [ROBOT_TYPE] parameters and modify the corresponding car model

```
export ROBOT_TYPE=X3 # ROBOT_TYPE: X1 X3 X3plus R2 X7
```

To replace the configuration file, copy the launch file and lua file required by cartographer to the specified directory. It is configured before leaving the factory.

```
sudo bash ~/yahboomcar_ws/src/yahboomcar_nav/scripts/copy_carto.sh
```

9.2.1. Start

Start the command (robot side). For the convenience of operation, this section takes [mono + laser + yahboomcar] as an example.

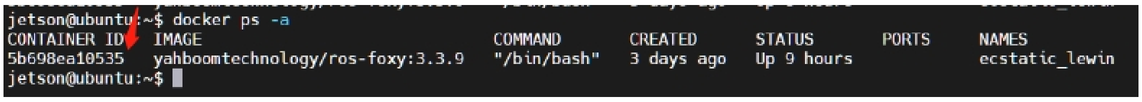
```
roslaunch yahboomcar_nav laser_bringup.launch # laser + yahboomcar
roslaunch yahboomcar_nav laser_usb_bringup.launch # mono + laser + yahboomcar
roslaunch yahboomcar_nav laser_astrapro_bringup.launch # Astra + laser + yahboomcar
```

Mapping command (robot side)

<PI5 needs to open another terminal to enter the same docker container

1. In the above steps, a docker container has been opened. You can open another terminal on the host (car) to view:

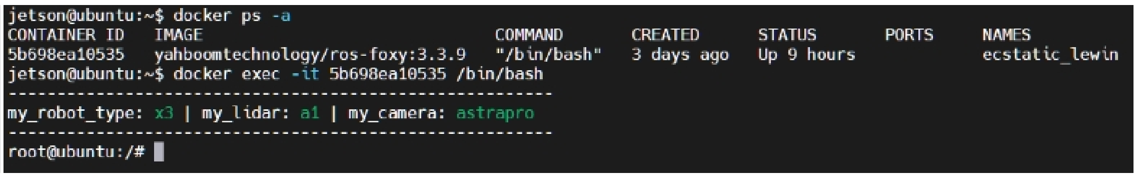
```
docker ps -a
```



```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                                  COMMAND       CREATED      STATUS       PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9       "/bin/bash"   3 days ago   Up 9 hours             ecstatic_lewin
jetson@ubuntu:~$
```

2. Now enter the docker container in the newly opened terminal:

```
docker exec -it 5b698ea10535 /bin/bash
```



```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                                  COMMAND       CREATED      STATUS       PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9       "/bin/bash"   3 days ago   Up 9 hours             ecstatic_lewin
jetson@ubuntu:~$ docker exec -it 5b698ea10535 /bin/bash
---------------------------------------------------------
my_robot_type: x3 | my_lidar: a1 | my_camera: astrapro
---------------------------------------------------------
root@ubuntu:/#
```

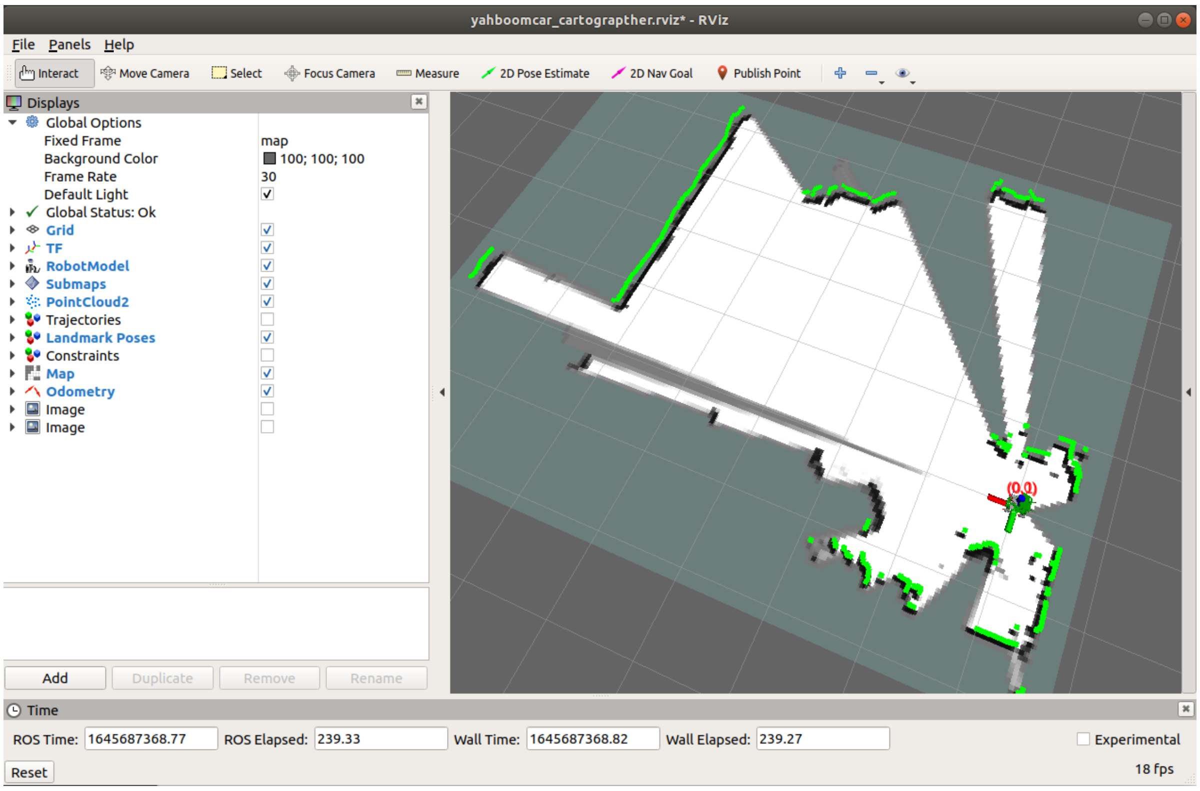
After successfully entering the container, you can open countless terminals to enter the container.

```
roslaunch yahboomcar_nav yahboomcar_map.launch use_rviz:=false
map_type:=cartographer
```

- [use_rviz] parameter: whether to enable rviz visualization.
- [map_type] parameter: Set the mapping algorithm [cartographer].

Turn on the visual interface (virtual machine side)

```
roslaunch yahboomcar_nav view_cartographer.launch
```



Global Options
Fixed Frame map
Background Color 100; 100; 100
Frame Rate 30
Default Light ✓
Global Status: Ok

ROS Time: 1645687368.77 ROS Elapsed: 239.33 Wall Time: 1645687368.82 Wall Elapsed: 239.27 Experimental

18 fps

The gap at the back of the robot is due to the obstruction caused by the installation position of the display screen, so a certain range of radar data is blocked. The shielding range can be adjusted, or it can not be blocked according to the actual situation. For specific operations, see [01. Radar Basic Course].

9.2.2. Controlling the robot

- Keyboard controls robot movement

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py # System integration
roslaunch yahboomcar_ctrl yahboom_keyboard.launch # Custom
```

- Control the robot movement with the handle

Make the robot cover the area to be mapped and the map should be as closed as possible.

There may be some scattered points during the mapping process. If the mapping environment is well closed, relatively regular, and the movement is slow, the scattering phenomenon will be much smaller.

9.2.3. Map saving

```
bash ~/yahboomcar_ws/src/yahboomcar_nav/maps/carto_map.sh
```

The map will be saved to the ~/yahboomcar_ws/src/yahboomcar_nav/maps/ folder, a pgm image and a yaml file.

map.yaml

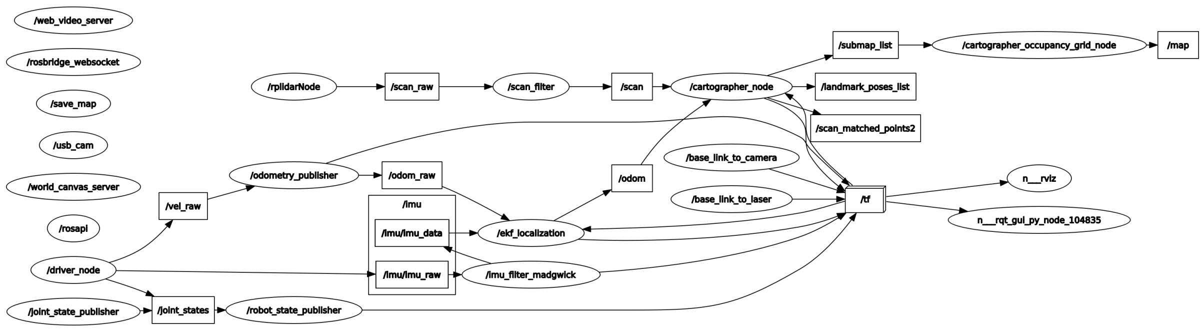
```
image: map.pgm
resolution: 0.05
origin: [-15.4,-12.2,0.0]
Negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Parameter analysis:

- image: The path of the map file, which can be an absolute path or a relative path.
- resolution: resolution of the map, meters/pixel
- Origin: 2D pose (x, y, yaw) in the lower left corner of the map. The yaw here is rotated counterclockwise (yaw=0 means no rotation). Many parts of the current system ignore the yaw value.
- negate: whether to reverse the meaning of white/black and free/occupied (the interpretation of the threshold is not affected)
- occupied_thresh: Pixels with an occupation probability greater than this threshold will be considered fully occupied.
- free_thresh: Pixels with occupancy probability less than this threshold will be considered completely free.

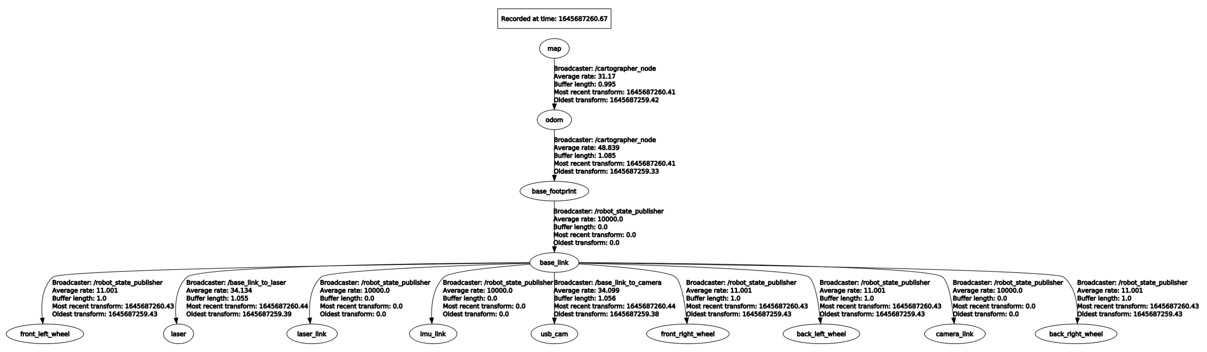
9.2.4. Node View

```
rqt_graph
```



9.2.5. View tf tree

```
rosrun rqt_tf_tree rqt_tf_tree
```


Recorded at time: 1645687260.67
map
Broadcaster: /cartographer_node
Average rate: 31.17
Buffer length: 0.995
Most recent transform: 1645687260.41
Oldest transform: 1645687259.42
odom
Broadcaster: /cartographer_node
Average rate: 48.839
Buffer length: 1.085
Most recent transform: 1645687260.41
Oldest transform: 1645687259.33
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
base_link
Broadcaster: /robot_state_publisher
Average rate: 11.001
Buffer length: 1.0
Most recent transform: 1645687260.43
Oldest transform: 1645687259.43
Broadcaster: /base_link_to_laser
Average rate: 34.134
Buffer length: 1.055
Most recent transform: 1645687260.44
Oldest transform: 1645687259.39
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /base_link_to_camera
Average rate: 34.099
Buffer length: 1.056
Most recent transform: 1645687260.44
Oldest transform: 1645687259.38
Broadcaster: /robot_state_publisher
Average rate: 11.001
Buffer length: 1.0
Most recent transform: 1645687260.43
Oldest transform: 1645687259.43
Broadcaster: /robot_state_publisher
Average rate: 11.001
Buffer length: 1.0
Most recent transform: 1645687260.43
Oldest transform: 1645687259.43
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 11.001
Buffer length: 1.0
Most recent transform: 1645687260.43
Oldest transform: 1645687259.43
front_left_wheel
laser
laser_link
imu_link
usb_cam
front_right_wheel
back_left_wheel
camera_link
back_right_wheel

9.3. Configuration parameters

lua file

Parameters	Description
map_frame	map coordinate system
tracking_frame	Convert all sensor data to this coordinate system
published_frame	The coordinate system pointed to by map
odom_frame	Do you provide odom's tf? If it is true, the tf tree is map->odom->footprint; if it is false, the tf tree is map->footprint
provide_odom_frame	If true, the local, non-loop-closed, continuous pose will be published as odom_frame in map_frame?
publish_frame_projected_to_2d	If enabled, published poses will be restricted to pure 2D poses (no roll, pitch or z-offset)
use_odometry	Whether to use odometer, if required, you must have odom tf
use_nav_sat	Whether to use gps
use_landmarks	Whether to use landmarks
num_laser_scans	Whether to use single-line laser data
num_multi_echo_laser_scans	Whether to use multi_echo_laser_scans data
num_subdivisions_per_laser_scan	1 frame of data is divided into several times for processing, usually 1
num_point_clouds	Whether to use point cloud data
lookup_transform_timeout_sec	Timeout when looking up tf
submap_publish_period_sec	Time interval for publishing submap (seconds)
pose_publish_period_sec	The time interval for publishing pose, when the value is 5e-3, it is 200HZ
trajectory_publish_period_sec	The time interval for publishing trajectory markers (trajectory nodes), the value is 30e-3 is 30ms
rangefinder_sampling_ratio	Fixed sampling frequency for lidar messages
odometry_sampling_ratio	Fixed sampling frequency of odometry messages
fixed_frame_pose_sampling_ratio	Fixed sampling frequency of fixed coordinate system messages
imu_sampling_ratio	Fixed sampling frequency of IMU messages
landmarks_sampling_ratio	Fixed sampling frequency of landmark messages