9.cartographer mapping algorithm

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

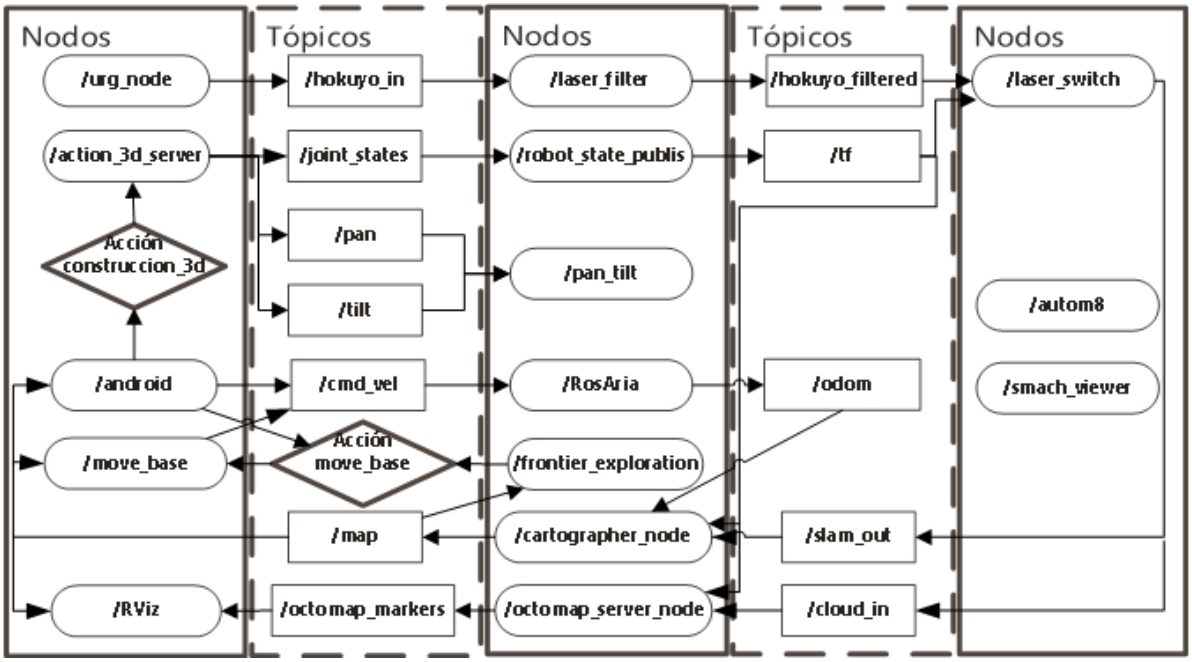
Cartographer ROS： https://google-cartographer-ros.readthedocs.io/en/latest/

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

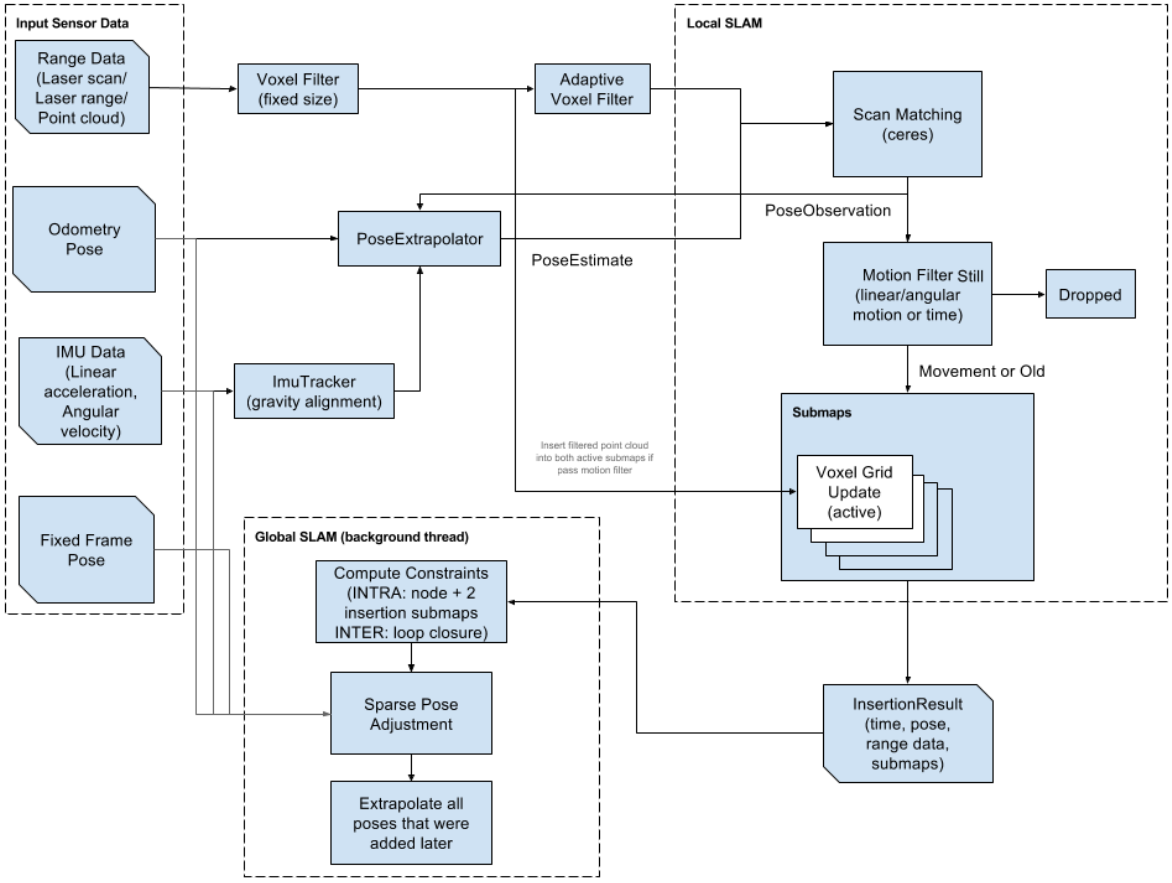
9.1 Introduction

Cartographer is a 2D and 3D SLAM(simultaneous localization and mapping) library supported by Google's open source ROS system. Graphing algorithm based on graph optimization(multi-threaded backend optimization, problem optimization built by cere). 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).



cartographer



Cartographer uses the mainstream SLAM framework, which is a three-stage method of feature extraction, closed-loop detection, and back-end optimization. A submap submap is composed of a certain number of LaserScans, and a series of submap submaps constitute the global map. The short-term process of building submaps with LaserScan has little cumulative error, but the long-term process of building global maps with submaps will have large cumulative errors, so it is necessary to use closed-loop detection to correct the positions of these submaps. The basic unit of closed-loop detection is Submap, closed loop detection adopts scan_match strategy. The focus of cartographer is the creation of submap submaps that fuse 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. It can receive various sensor data in the form of ROS messages, and publish it in the form of messages after processing, which is convenient for debugging and visualization.

9.2 Use

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

According to different models, you only need to set the purchased model in [.bashrc], X1(ordinary four-wheel drive) X3(Mike wheel) X3plus(Mike wheel mechanical arm) R2(Ackerman differential) and so on. Section takes X3 as an example

Open the [.bashrc] file

```
sudo vim .bashrc
```

Find the [ROBOT_TYPE] parameter and modify the corresponding 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 has been configured before leaving the factory.

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

9.2.1 Start

Start 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
```

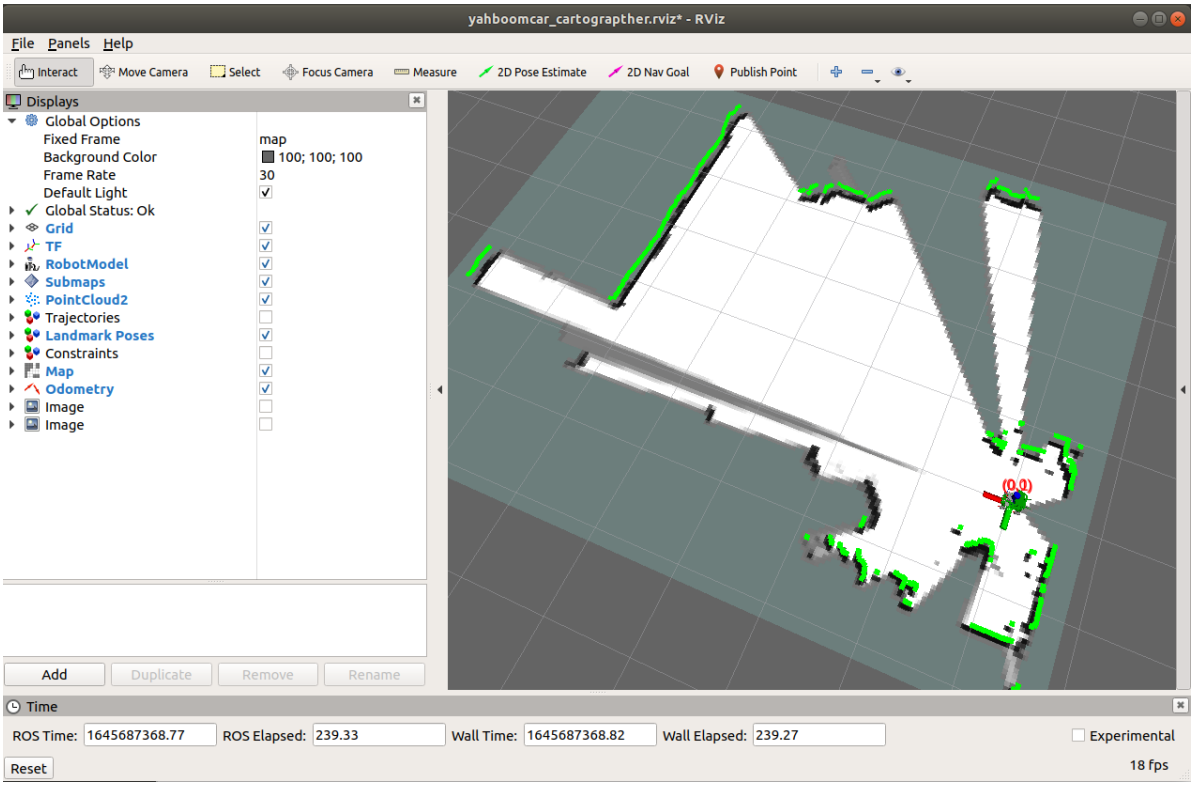
Mapping command(robot side)

```
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].

Open the visual interface(virtual machine side)

```
roslaunch yahboomcar_nav view_cartographer.launch
```



The gap at the back of the robot is due to the occlusion of the installation position of the display screen, so a certain range of Lidar data is shielded. The shielding range can be adjusted or not shielded according to the actual situation. For details, see [01. Lidar Basic Course].

9.2.2 Control the robot

- The keyboard controls the movement of the robot

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py   # system integration
roslaunch yahboomcar_ctrl yahboom_keyboard.launch       # custom
```

- The handle controls the movement of the robot

Make the robot walk all over the area to be built, and the map is as closed as possible.

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

9.2.3 Map save

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

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

map.yaml

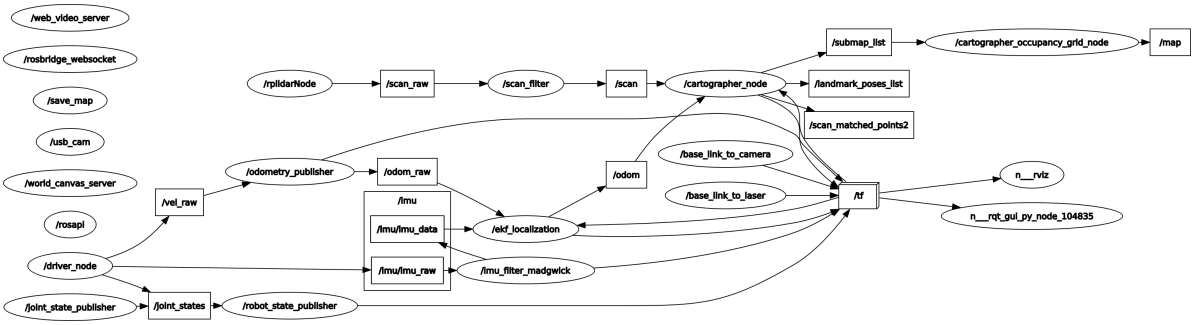
```
image :  map.pgm
resolution :  0.05
origin :  [-15.4,-12.2,0.0]
negate :  0
occupied_thresh :  0.65
free_thresh:  0.196
```

Parameter parsing:

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

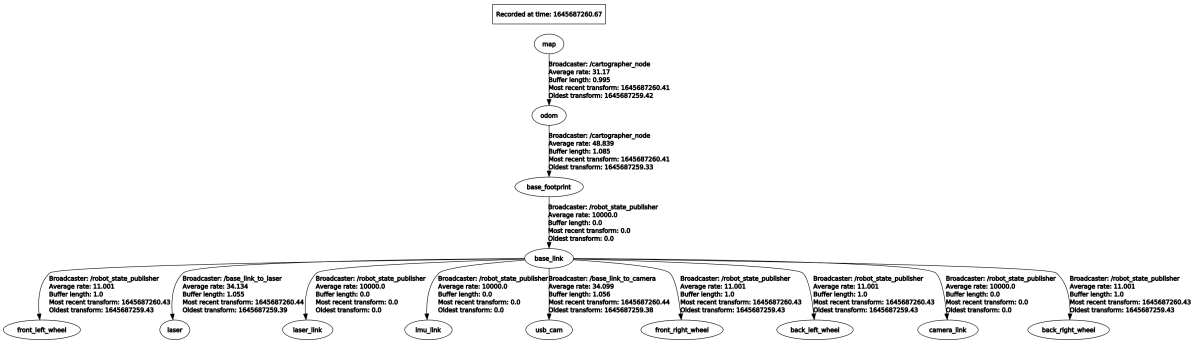
9.2.4 Node View

```
rqt_graph
```



9.2.5 View tf tree

```
rosrun rqt_tf_tree rqt_tf_tree
```



9.3 configuration parameters

lua file

parameter	illustrate
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 true, the tf tree is map->odom->footprint; if 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 the odometer, if it is required, there must be odom's tf
use_nav_sat	whether to use gps
use_landmarks	whether to use landmark
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 processing, usually 1
num_point_clouds	Whether to use point cloud data
lookup_transform_timeout_sec	Timeout when looking for tf
submap_publish_period_sec	Time interval(seconds) for publishing submaps
pose_publish_period_sec	The time interval for releasing the pose, when the value is 5e-3, it is 200HZ
trajectory_publish_period_sec	The time interval for publishing the 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 odometer messages
fixed_frame_pose_sampling_ratio	Fixed sampling frequency for fixed coordinate system messages
imu_sampling_ratio	Fixed sampling frequency for IMU messages
landmarks_sampling_ratio	Fixed sampling frequency for signpost messages