# 6. cartographer building algorithm

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

Cartographer ROS2： https://github.com/ros2/cartographer_ros

The operating environment and hardware and software reference configuration are as follows:

- Reference model: ROSMASTERR2
- Robot hardware configuration: Arm series main control, Silan A1 LiDAR, AstraPro Plus depth camera.
- Robot system: Ubuntu (version not required) + docker (version 20.10.21 and above)
- PC virtual machine: Ubuntu (20.04) + ROS2 (Foxy)
- Usage scenario: use on a relatively clean 2D plane

## 6.1. Introduction

- Cartographer

Cartographer is a Google open source ROS system support for 2D and 3D SLAM (simultaneous localization and mapping) library . Based on the graph optimization (multi-threaded back-end optimization , cere constructed problem optimization) approach to building mapping algorithms . Data from multiple sensors (e.g., LIDAR, IMU, and camera) can be combined to synchronously compute the sensor's location and map the sensor's surroundings.

The cartographer source code consists of three main parts: cartographer, cartographer_ros, and ceres-solver (back-end optimization).
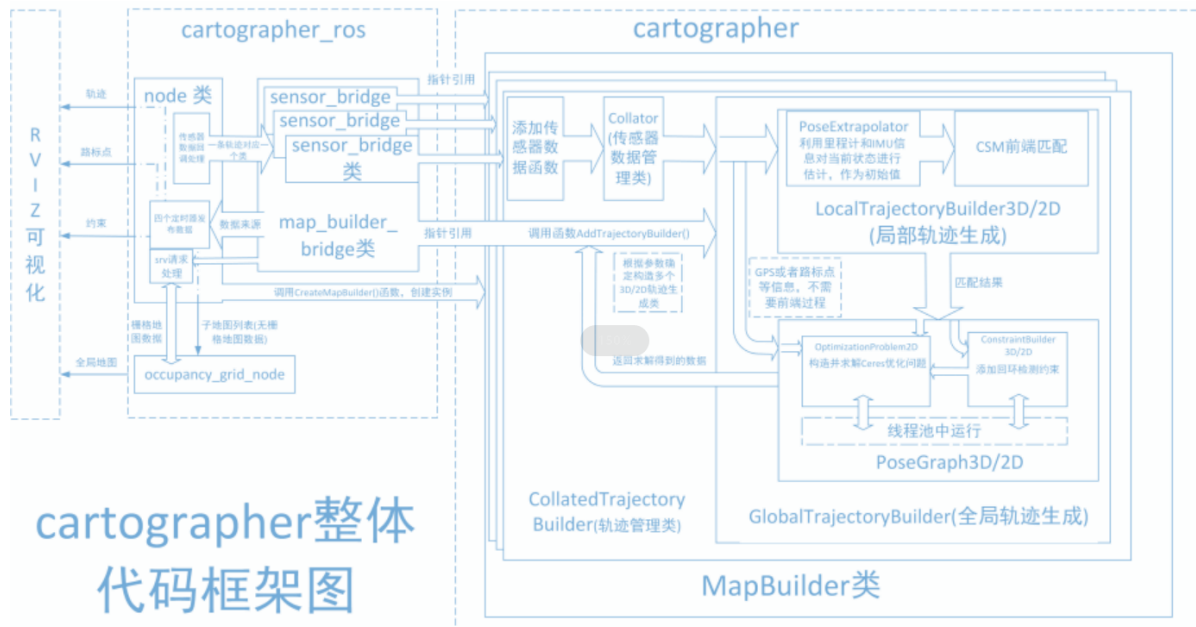
cartographer uses the mainstream SLAM framework, which is a three-stage approach of feature extraction, closed-loop detection, and back-end optimization. A submap subgraph is composed of a certain number of LaserScan, and a series of submap subgraphs constitute a global map. The short time process of constructing a submap with LaserScan has a small cumulative error, but the long time process of constructing a global map with a submap has a large cumulative error, so it is necessary to utilize closed-loop detection to correct the position of these submaps, and the basic unit of the closed-loop detection is the submap, and the closed-loop detection adopts the scan_match strategy. The focus of cartographer is the creation of submap subgraphs fusing multi-sensor data (odometry, IMU, LaserScan, etc.) and the implementation of the scan_match strategy for closed-loop detection.

- cartographer_ros

cartographer_ros runs under ROS and can accept various sensor data in the form of ROS messages, and after

processed and then published as messages for debugging and visualization.

# 6.2 Usage

## 6.2.1 Pre-use Configuration

Note: Since ROSMASTER series robots are divided into multiple robots as well as multiple devices, the factory system has been configured with routines for multiple devices, but since it is not possible to automatically recognize the products, you need to manually set the machine type and radar model.

After entering the container: according to the model of the trolley, the type of radar and the type of camera make the following changes:

```
root@ubuntu:/# cd
root@ubuntu:~# vim .bashrc
```

When the changes are complete, save and exit vim, then execute:

```
root@ubuntu:~# source .bashrc
--------------------------------------------------------
ROS_DOMAIN_ID: 12
my_robot_type: x3 | my_lidar: a1 | my_camera: astraplus
--------------------------------------------------------
root@ubuntu:~#
```

You can see the current model of the modified cart, the type of radar and the type of camera

## 6.2.2 Specific use

**Note: When building a map, the slower the better the result (note that if you rotate slower), too fast, the effect will be very poor.**

First of all, in the host [that is, the car on the jetson] in the need to do port binding operation [see the port binding tutorial chapter], where the main use of radar and serial port two devices;

Then check whether the radar and serial devices are in the port binding state: in the host [i.e., on the cart's jetson], refer to the following command to view the successful binding is the following state:



If it shows that the radar or serial device is not bound, you can plug and unplug the USB cable to check again.

Enter the docker container, see [docker course in ----- 5, enter the docker container of the robot], sub-terminal execution of the following LAUNCH file:

1. Launch build map

```
ros2 launch yahboomcar_nav map_cartographer_launch.py
```

2. Start rviz to display the map, this step is recommended to be executed in a virtual machine, virtual machines need to be configured to multi-machine communication

```
ros2 launch yahboomcar_nav display_map_launch.py
```
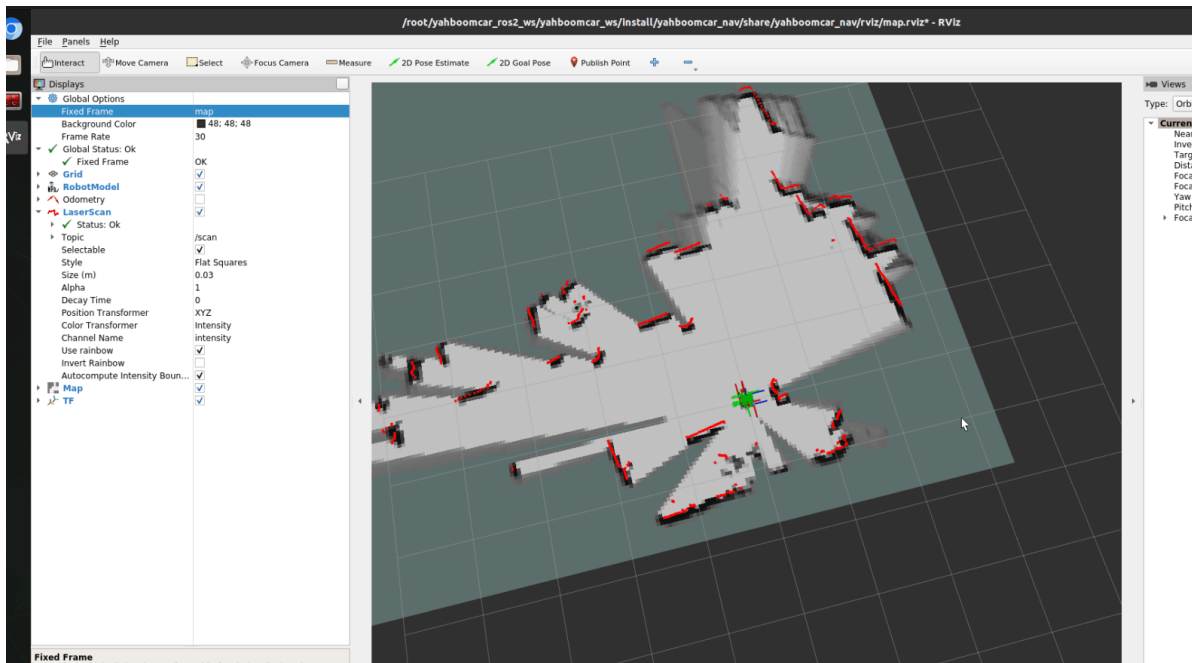
3. Start the keyboard control node, this step is recommended to perform in the virtual machine, virtual machine needs to be configured for multi-machine communication

Or use the remote control [slowly moving cart] to start building the map, until a complete map is built!

```
ros2 run yahboomcar_ctrl yahboom_keyboard
```
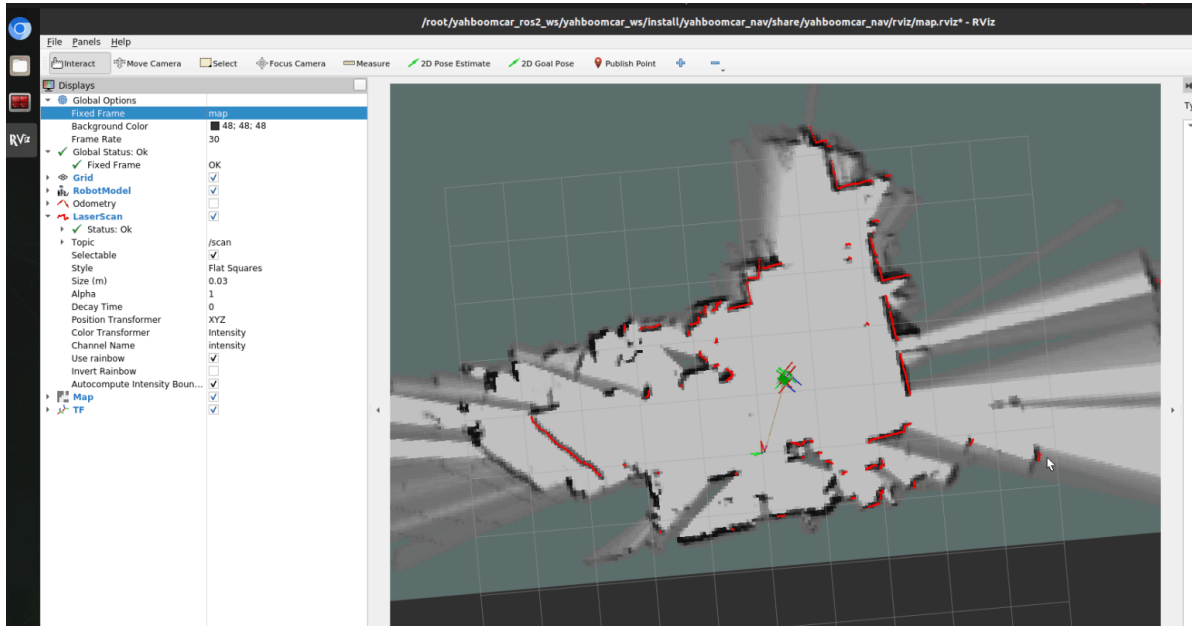


4. The map is saved with the following path:

```
ros2 launch yahboomcar_nav save_map_launch.py
```

The save path is as follows:

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/
```
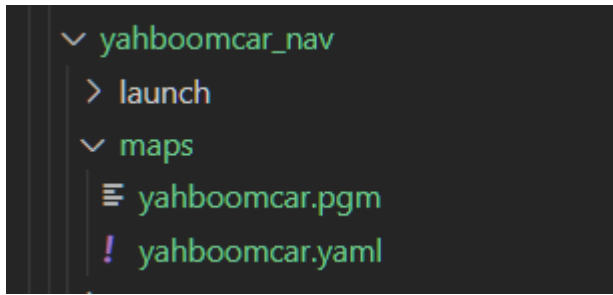


One pgm image, one yaml file yahboomcar.yaml

```
image:
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pgm
mode: trinary
resolution: 0.05
origin: [-10, -10, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

Parameter parsing:

- image: path of the map file, can be absolute or relative.
- mode: the attribute can be one of trinary, scale or raw, depends on the selected mode, trinary mode is the default.
- resolution: the resolution of the map, in meters/pixels
- origin: the 2D position (x,y,yaw) of the bottom left corner of the map, where yaw is rotated counterclockwise (yaw=0 means no rotation). Currently many parts of the system ignore the yaw value.
- negate: whether to reverse the meaning of white/black, free/occupied (the interpretation of the thresholds is not affected)
- occupied_thresh: pixels with occupancy probability greater than this threshold are considered fully occupied.
- free_thresh: pixels with occupancy probability less than this threshold are considered completely free.

# 6.3. Node resolution

## 6.3.1. Displaying a computational graph

```
rqt_graph
```

## 6.3.2. cartographer_node node details

```
rr@rr-pc:~/rviz$ ros2 node info /cartographer_node
/cartographer_node
  Subscribers:
    /odom: nav_msgs/msg/Odometry
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /scan: sensor_msgs/msg/LaserScan
  Publishers:
    /constraint_list: visualization_msgs/msg/MarkerArray
    /landmark_poses_list: visualization_msgs/msg/MarkerArray
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /scan_matched_points2: sensor_msgs/msg/PointCloud2
    /submap_list: cartographer_ros_msgs/msg/SubmapList
    /tf: tf2_msgs/msg/TFMessage
    /trajectory_node_list: visualization_msgs/msg/MarkerArray
  Service Servers:
    /cartographer_node/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /cartographer_node/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /cartographer_node/get_parameters: rcl_interfaces/srv/GetParameters
    /cartographer_node/list_parameters: rcl_interfaces/srv/ListParameters
    /cartographer_node/set_parameters: rcl_interfaces/srv/SetParameters
    /cartographer_node/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
    /finish_trajectory: cartographer_ros_msgs/srv/FinishTrajectory
    /start_trajectory: cartographer_ros_msgs/srv/StartTrajectory
    /submap_query: cartographer_ros_msgs/srv/SubmapQuery
    /write_state: cartographer_ros_msgs/srv/WriteState
  Service Clients:

  Action Servers:

  Action Clients:
```

```
rr@rr-pc:~/rviz$ ros2 node info /occupancy_grid_node
/occupancy_grid_node
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /submap_list: cartographer_ros_msgs/msg/SubmapList
  Publishers:
    /map: nav_msgs/msg/OccupancyGrid
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /occupancy_grid_node/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /occupancy_grid_node/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /occupancy_grid_node/get_parameters: rcl_interfaces/srv/GetParameters
    /occupancy_grid_node/list_parameters: rcl_interfaces/srv/ListParameters
    /occupancy_grid_node/set_parameters: rcl_interfaces/srv/SetParameters
    /occupancy_grid_node/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:
    /submap_query: cartographer_ros_msgs/srv/SubmapQuery
  Action Servers:

  Action Clients:
```

## 6.3.3 TF transformations

```
ros2 run tf2_tools view_frames.py
```