Cartographer Mapping Algorithm

7.1, Program Function Description

After the programs on the virtual machine and the car are started, the car is controlled by the handle or keyboard. The car will use the radar scanning data during the movement, use the cartographer algorithm to build a map, and save the map after the construction is completed. This process is visualized in rviz.

7.2, Cartographer Introduction

> Cartographer: https://google-cartographer.readthedocs.io/en/latest/
>
> Cartographer ROS2: https://github.com/ros2/cartographer_ros

- **Cartographer**

Cartographer is a 2D and 3D SLAM (simultaneous localization and mapping) library supported by Google's open source ROS system. The mapping algorithm is based on the method of graph optimization (multi-threaded backend optimization, problem optimization built by cere). It can combine data from multiple sensors (such as LIDAR, IMU and camera) to synchronously calculate the position of the sensor and draw the environment around the sensor.

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

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

- **cartographer_ros**

cartographer_ros runs under ROS and can receive various sensor data in the form of ROS messages. After processing, it is published in the form of messages for easy debugging and visualization.

7.3, Program code reference path

After SSH connects to the car, the location of the function source code is,

```
/home/sunrise/yahboomcar_ws/src/yahboomcar_nav/launch/map_gmapping_launch.py
```

The source code of the virtual machine is located at,

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_rviz/launch/yahboomcar_mapping_launch.py
```

7.4, Program startup

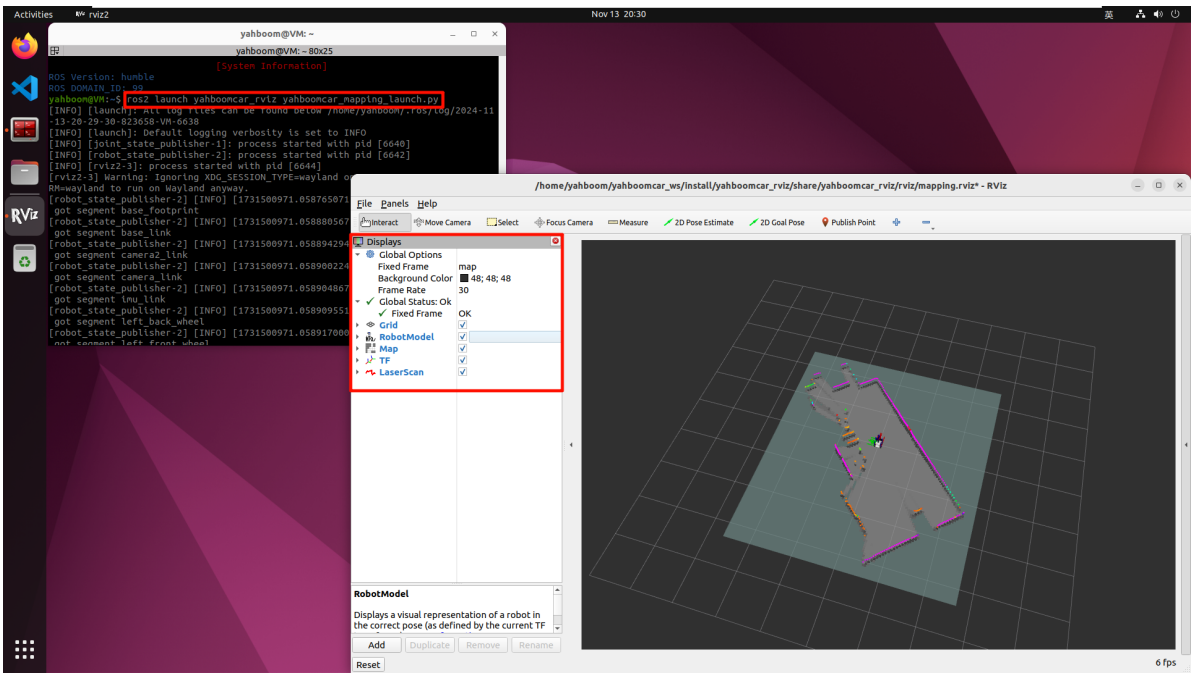
7.4.1, Map building

Open the virtual machine terminal and enter,

```
ros2 launch yahboomcar_rviz yahboomcar_mapping_launch.py
```

After SSH connects to the car, enter in the terminal,

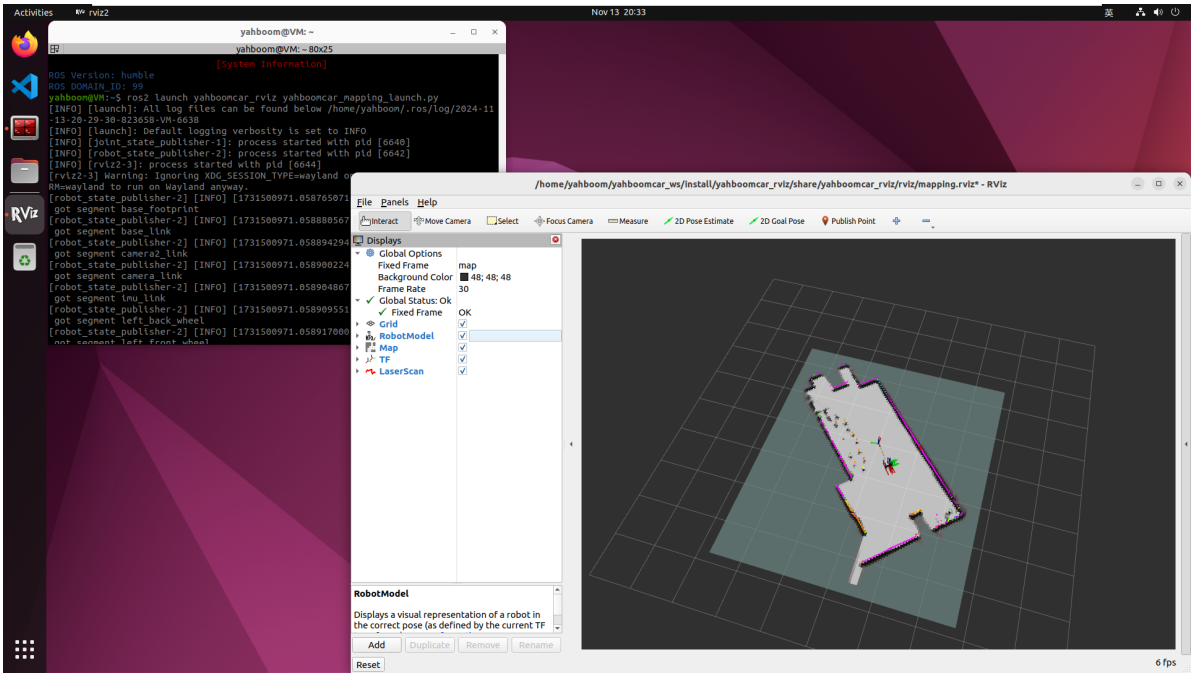
```
ros2 launch yahboomcar_nav map_cartographer_launch.py
```



Unlock the [L1] button on the handle to control the car's movement. If you don't use the handle, you can use the keyboard to control the car's terminal input,

```
ros2 run yahboomcar_ctrl yahboom_keyboard
```

[Slowly move the car] to start building the map until the complete map is built.



Note: When building the map, the slower the speed, the better the effect (especially the slower the rotation speed). If the speed is too fast, the effect will be very poor.

In addition, the path of cartographer related configuration parameters is,

```
/home/sunrise/yahboomcar_ws/src/yahboomcar_nav/params/cartographer_config.lua
```

You can modify it as needed, compile and run.

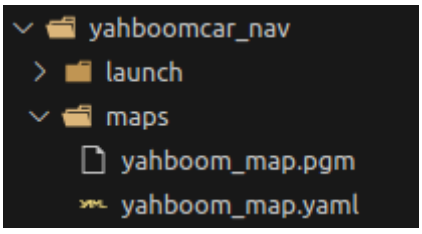
7.4.2, map saving

Input in the car terminal:

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

The map saving path is as follows:

```
/home/sunrise/yahboomcar_ws/src/yahboomcar_nav/maps/yahboom_map
```



Includes a .pgm picture and a .yaml file. Among them, the .yaml file content is as follows:

```
image: yahboom_map.pgm
mode: trinary
resolution: 0.05
origin: [-3.8, -2.79, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

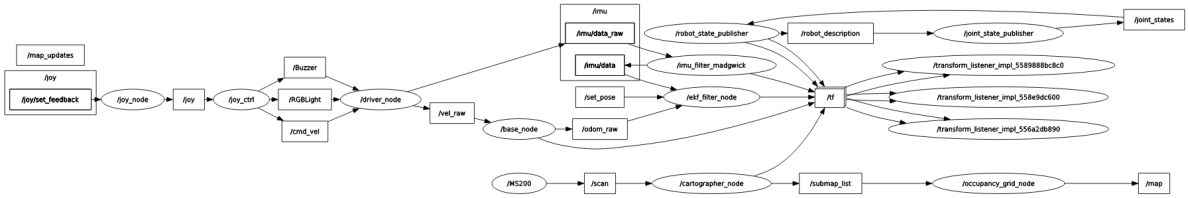
Parameter analysis:

- image: the path of the map file, which can be an absolute path or a relative path
- mode: this attribute can be one of trinary, scale or raw, depending on the selected mode, trinary mode is the default mode
- resolution: the resolution of the map, meters/pixels
- 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). Currently, many parts of the system will ignore the yaw value.
- negate: whether to invert 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 are considered fully occupied.
- free_thresh: pixels with an occupation probability less than this threshold are considered fully free.

7.4.3, view the topic communication node graph

Virtual machine terminal input,

```
ros2 run rqt_graph rqt_graph
```



7.4.4, view TF tree

Open the virtual machine terminal, input,

```
#Save tf tree
ros2 run tf2_tools view_frames
#View tf tree
evince frames.pdf
```