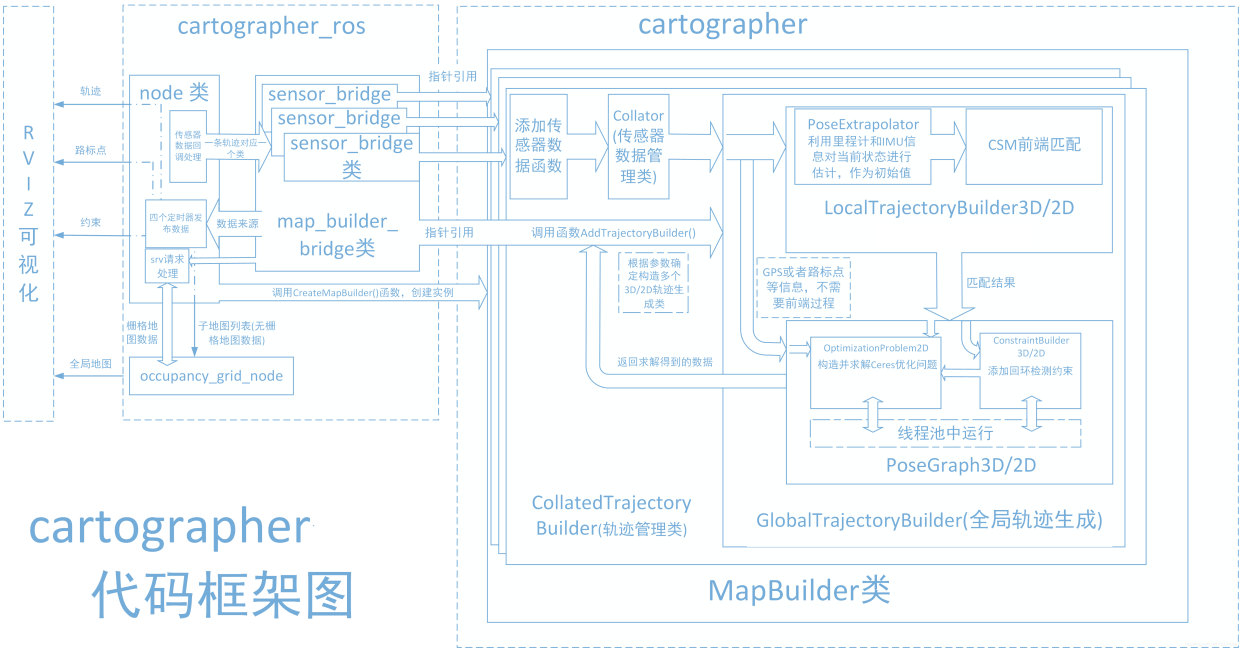
Cartographer mapping

Note: The virtual machine needs to be in the same LAN as the car, and the ROS_DOMAIN_ID needs to be consistent. You can check [Must read before use] to set the IP and ROS_DOMAIN_ID on the board.

1、Introduction to Cartographer

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 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.

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.

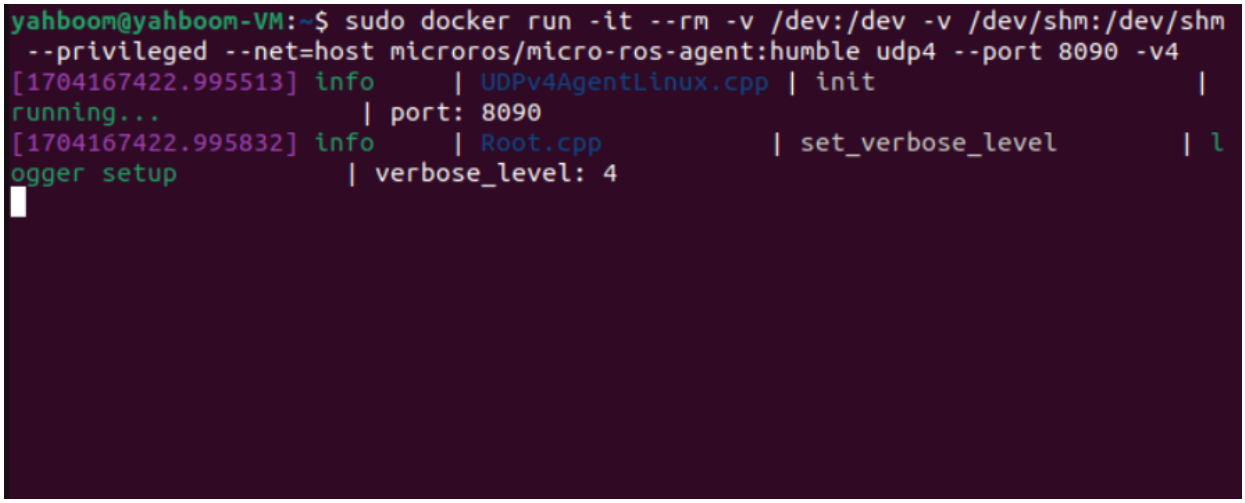
2. Program function description

Connect the car to the agent and run the program. The mapping interface will be displayed in rviz. Use the keyboard or handle to control the movement of the car until the map is completed. Then run the save map command to save the map.

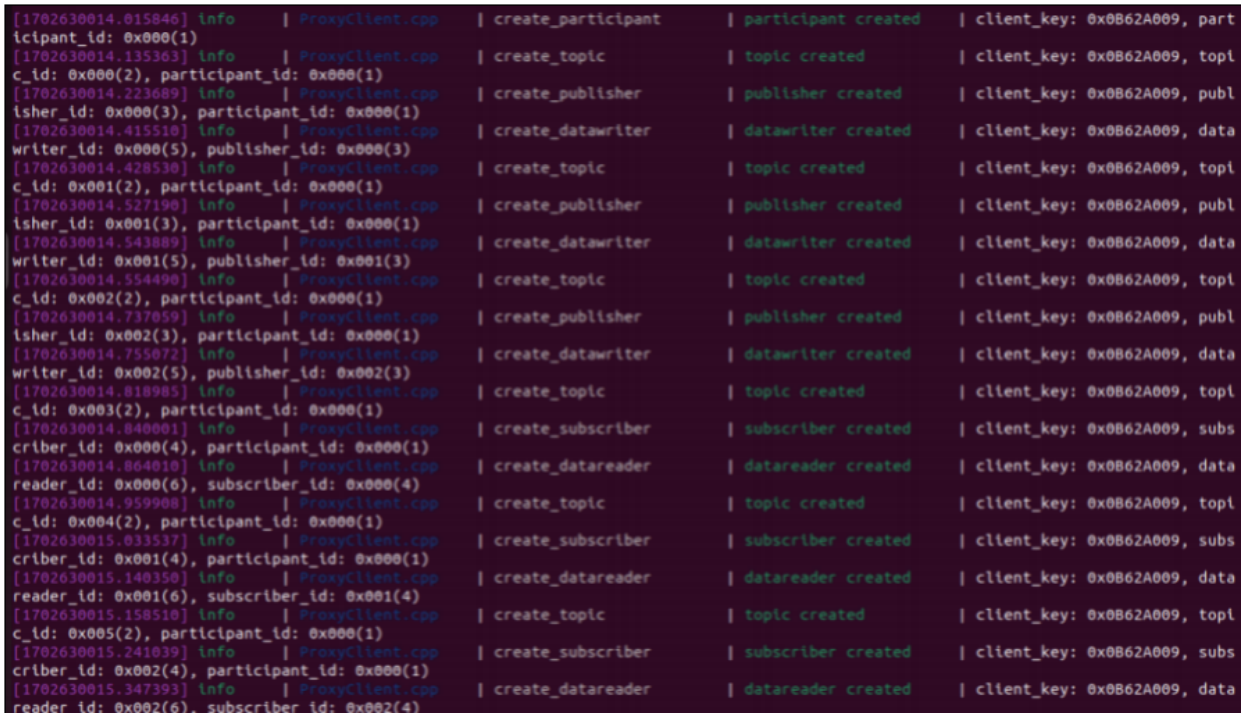
3. Start and connect to the agent

Taking the supporting virtual machine as an example, enter the following command to start the agent:

```
sudo docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --net=host
microros/micro-ros-agent:humble udp4 --port 8090 -v4
```



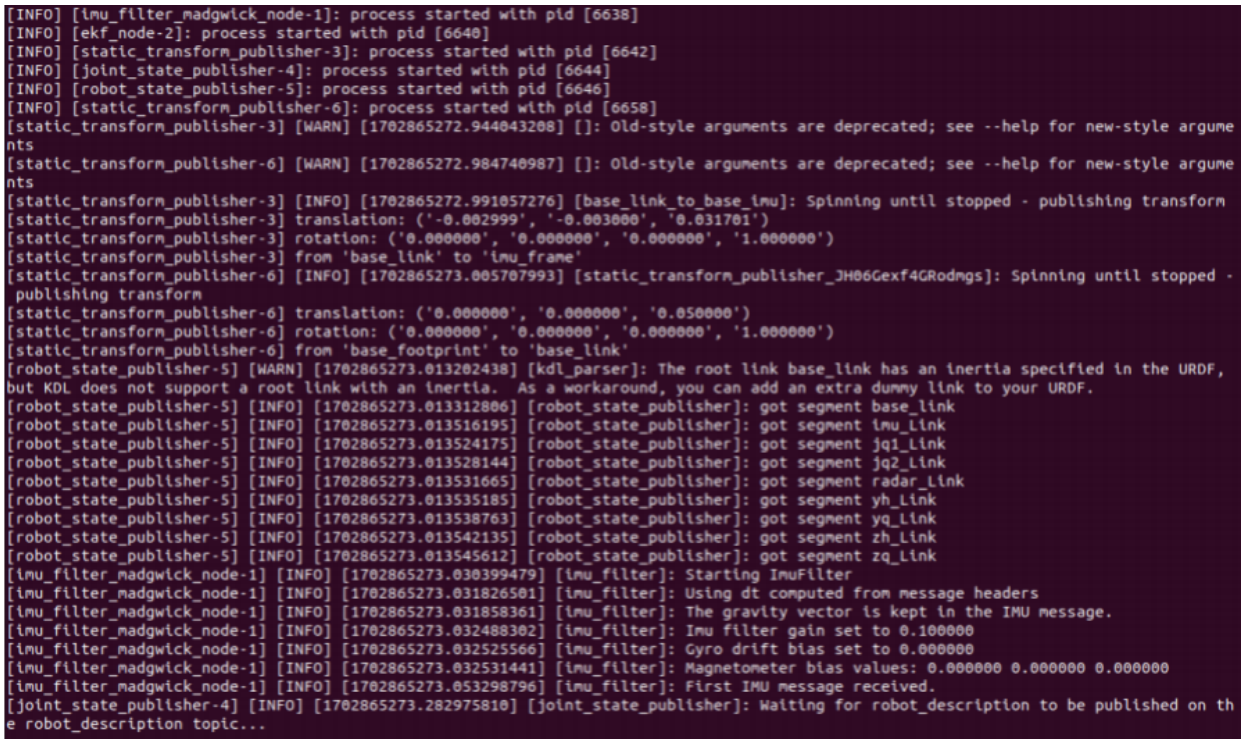
Then, turn on the car switch and wait for the car to connect to the agent. The connection is successful, as shown in the figure below.



4. Start the program

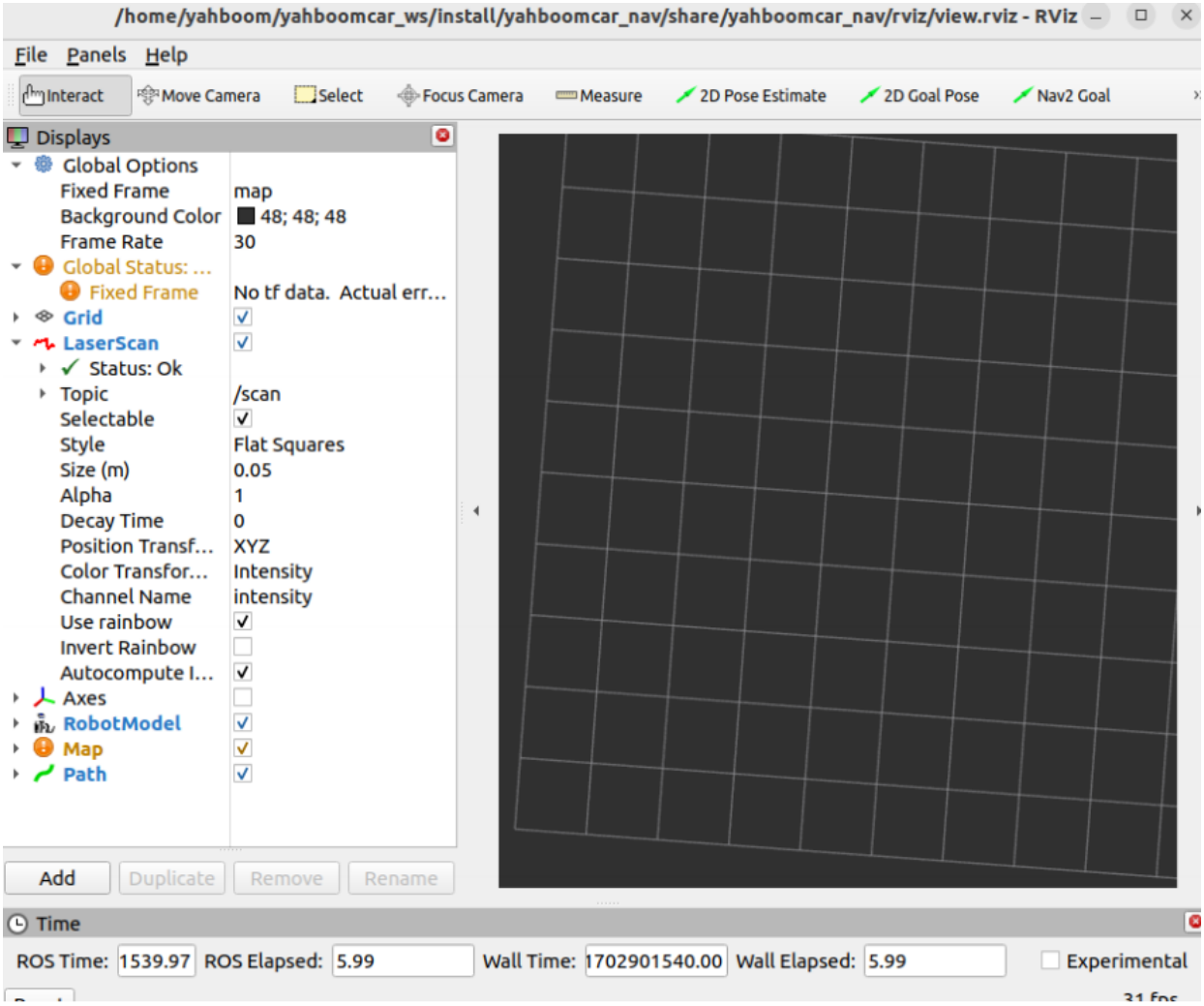
First, start the car to process the underlying data program and enter the terminal.

```
ros2 launch yahboomcar_bringup yahboomcar_bringup_launch.py
```



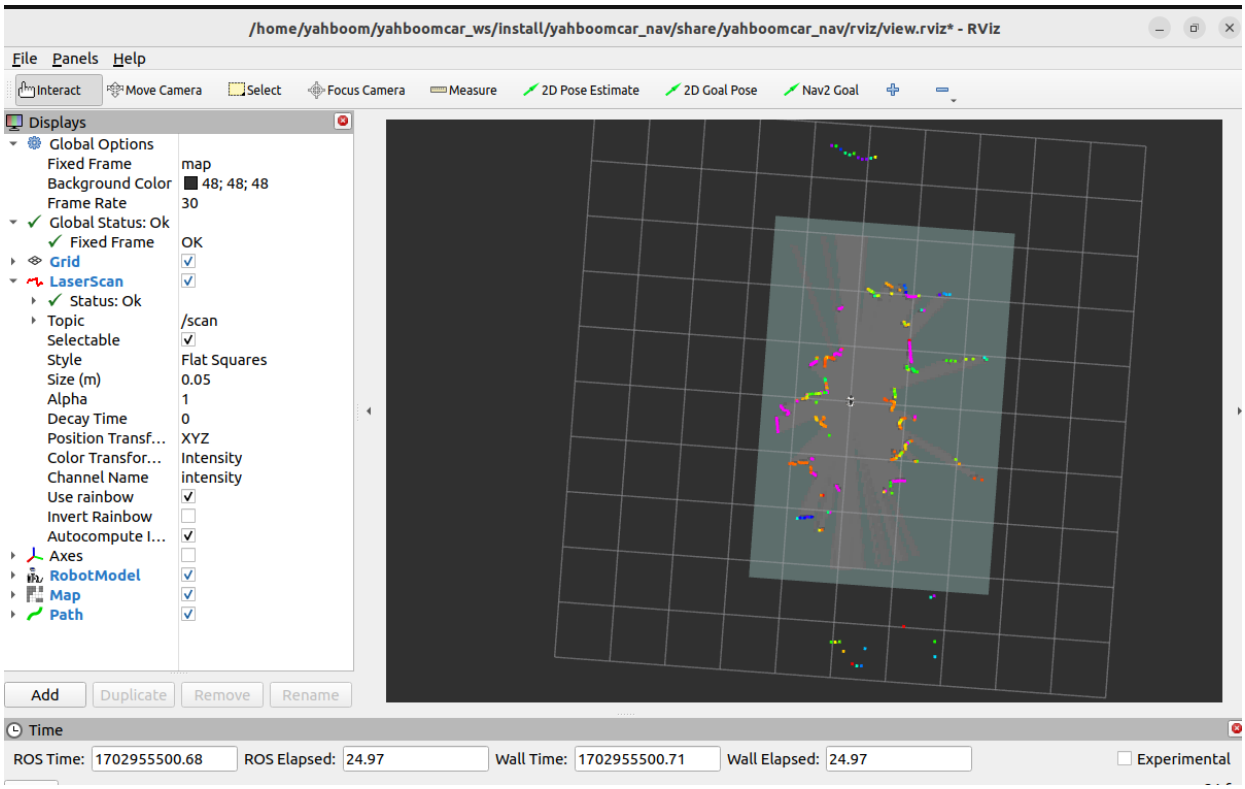
Then, start rviz, visualize the mapping, and enter in the terminal.

```
ros2 launch yahboomcar_nav display_launch.py
```



The mapping node has not been run yet, so there is no data. Next, run the mapping node and enter in the terminal,

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

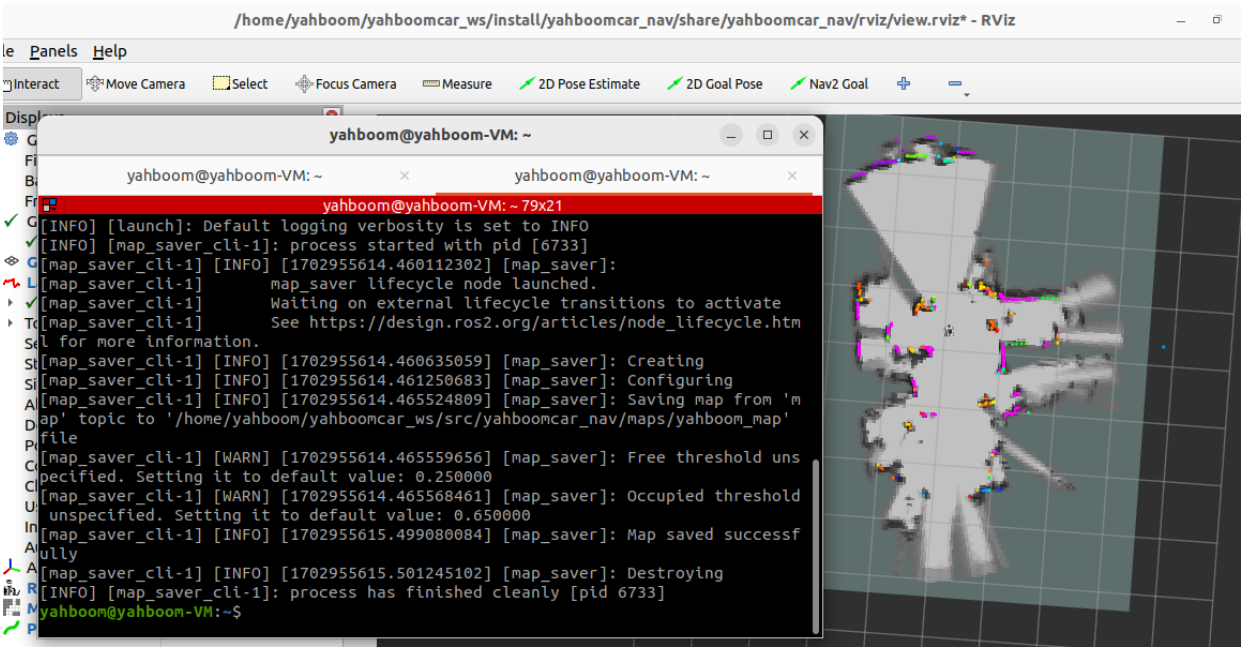


Then run handle control or keyboard control, choose one of the two, terminal input,

```
#keyboard
ros2 run yahboomcar_ctrl yahboom_keyboard
#handle
ros2 run yahboomcar_ctrl yahboom_joy
ros2 run joy joy_node
```

Then control the car and slowly walk through the area that needs to be mapped. After the map is completed, enter the following command to save the map and enter it in the terminal.

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



A map named yahboom_map will be saved. This map is saved in,

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/maps
```

Two files will be generated, one is yahboom_map.pgm and the other is yahboom_map.yaml. Take a look at the content of yaml.

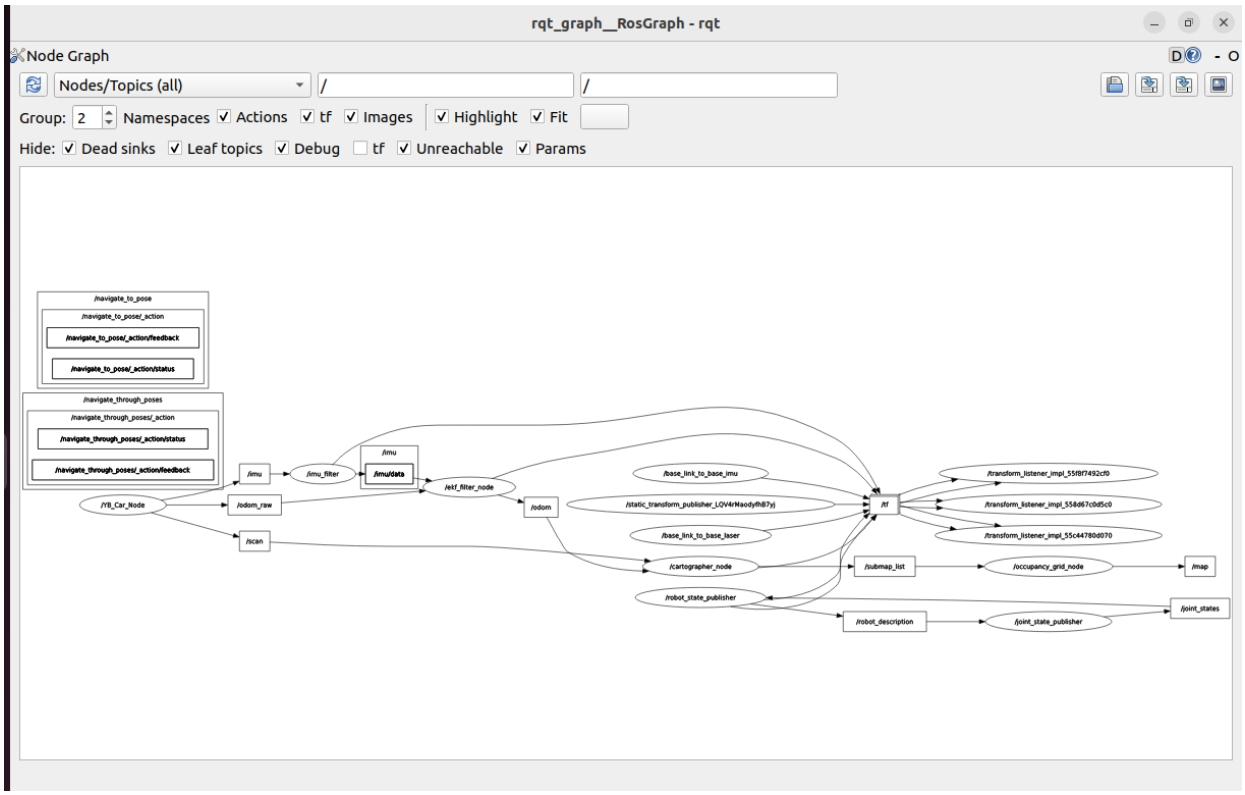
```yaml
image: yahboom_map.pgm
mode: trinary
resolution: 0.05
origin: [-10, -10, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

- image：The picture representing the map, that is, yahboom_map.pgm
- mode：This attribute can be one of trinary, scale or raw, depending on the selected mode. trinary mode is the default mode.
- resolution：Map resolution, meters/pixel
- The 2D pose (x, y, yaw) in the lower left corner of the map, where yaw 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 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.

5. View the node communication diagram

Terminal input,

```
ros2 run rqt_graph rqt_graph
```



If it is not displayed at first, select [Nodes/Topics(all)], and then click the refresh button in the upper left corner.

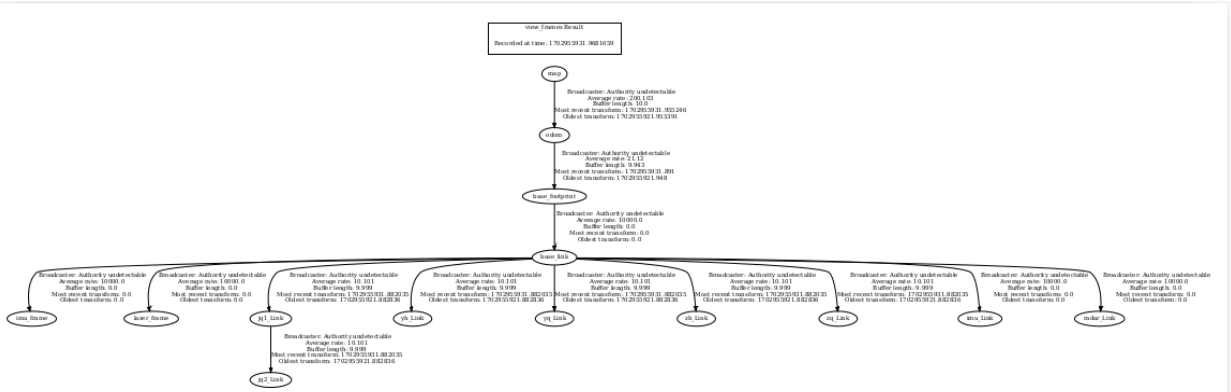
6. View TF tree

Terminal input,

```
ros2 run tf2_tools view_frames
```

```
yahboom@yahboom-VM:~$ ros2 run tf2_tools view_frames
[INFO] [1702955926.954207298] [view_frames]: Listening to tf data for 5.0 seconds...
[INFO] [1702955931.956461115] [view_frames]: Generating graph in frames.pdf file...
[INFO] [1702955931.959678222] [view_frames]: Result:tf2_msgs.srv.FrameGraph_Response(frame_yaml="imu_frame: \n  parent: 'base_link'\
n  broadcaster: 'Authority undetectable'\n  rate: 10000.000\n  most_recent_transform: 0.000000\n  oldest_transform: 0.000000\n  buff
er_length: 0.000\nbase_link: \n  parent: 'base_footprint'\n  broadcaster: 'Authority undetectable'\n  rate: 10000.000\n  most_recent
_transform: 0.000000\n  oldest_transform: 0.000000\n  buffer_length: 0.000\nlaser_frame: \n  parent: 'base_link'\n  broadcaster: 'Au
thority undetectable'\n  rate: 10000.000\n  most_recent_transform: 0.000000\n  oldest_transform: 0.000000\n  buffer_length: 0.000\nb
ase_footprint: \n  parent: 'odom'\n  broadcaster: 'Authority undetectable'\n  rate: 21.120\n  most_recent_transform: 1702955931.8910
00\n  oldest_transform: 1702955921.948000\n  buffer_length: 9.943\nodom: \n  parent: 'map'\n  broadcaster: 'Authority undetectable'\
n  rate: 200.103\n  most_recent_transform: 1702955931.955246\n  oldest_transform: 1702955921.955391\n  buffer_length: 10.000\njq1_Li
nk: \n  parent: 'base_link'\n  broadcaster: 'Authority undetectable'\n  rate: 10.101\n  most_recent_transform: 1702955931.882035\n
oldest_transform: 1702955921.882836\n  buffer_length: 9.999\njq2_Link: \n  parent: 'jq1_Link'\n  broadcaster: 'Authority undetectabl
e'\n  rate: 10.101\n  most_recent_transform: 1702955931.882035\n  oldest_transform: 1702955921.882836\n  buffer_length: 9.999\nyh_Li
nk: \n  parent: 'base_link'\n  broadcaster: 'Authority undetectable'\n  rate: 10.101\n  most_recent_transform: 1702955931.882035\n
oldest_transform: 1702955921.882836\n  buffer_length: 9.999\nyq_Link: \n  parent: 'base_link'\n  broadcaster: 'Authority undetectabl
e'\n  rate: 10.101\n  most_recent_transform: 1702955931.882035\n  oldest_transform: 1702955921.882836\n  buffer_length: 9.999\nzh_Li
nk: \n  parent: 'base_link'\n  broadcaster: 'Authority undetectable'\n  rate: 10.101\n  most_recent_transform: 1702955931.882035\n
oldest_transform: 1702955921.882836\n  buffer_length: 9.999\nzq_Link: \n  parent: 'base_link'\n  broadcaster: 'Authority undetectabl
e'\n  rate: 10.101\n  most_recent_transform: 1702955931.882035\n  oldest_transform: 1702955921.882836\n  buffer_length: 9.999\nimu_L
ink: \n  parent: 'base_link'\n  broadcaster: 'Authority undetectable'\n  rate: 10000.000\n  most_recent_transform: 0.000000\n  oldes
t_transform: 0.000000\n  buffer_length: 0.000\nradar_Link: \n  parent: 'base_link'\n  broadcaster: 'Authority undetectable'\n  rate:
 10000.000\n  most_recent_transform: 0.000000\n  oldest_transform: 0.000000\n  buffer_length: 0.000\n")
```

After the operation is completed, two files will be generated in the terminal directory, namely .gv and .pdf files. The pdf file is the TF tree.



7. Code analysis

Here we only describe map_cartographer_launch.py for mapping. The file path is as follows

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/launch
```

map_cartographer_launch.py

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node


def generate_launch_description():
    package_launch_path =os.path.join(get_package_share_directory('yahboomcar_nav'),
'launch')

    cartographer_launch = IncludeLaunchDescription(PythonLaunchDescriptionSource(
        [package_launch_path, '/cartographer_launch.py'])
    )
    base_link_to_laser_tf_node = Node(
     package='tf2_ros',
     executable='static_transform_publisher',
     name='base_link_to_base_laser',
     arguments=['-0.0046412', '0' ,
'0.094079','0','0','0','base_link','laser_frame']
    )
    return LaunchDescription([cartographer_launch,base_link_to_laser_tf_node])
```

A launch file-cartographer_launch and a node that publishes static transformation-base_link_to_laser_tf_node are run here. Mainly look at cartographer_launch, which is located at,

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/launch
```

cartographer_launch.py,

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch_ros.actions import Node
from launch.substitutions import LaunchConfiguration
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir


def generate_launch_description():
    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    package_path = get_package_share_directory('yahboomcar_nav')
    configuration_directory = LaunchConfiguration('configuration_directory',
default=os.path.join(
                                                  package_path, 'params'))
    configuration_basename = LaunchConfiguration('configuration_basename',
default='lds_2d.lua')

    resolution = LaunchConfiguration('resolution', default='0.05')
    publish_period_sec = LaunchConfiguration(
        'publish_period_sec', default='1.0')

    return LaunchDescription([
        DeclareLaunchArgument(
            'configuration_directory',
            default_value=configuration_directory,
            description='Full path to config file to load'),
        DeclareLaunchArgument(
            'configuration_basename',
            default_value=configuration_basename,
            description='Name of lua file for cartographer'),
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='false',
            description='Use simulation (Gazebo) clock if true'),

        Node(
            package='cartographer_ros',
            executable='cartographer_node',
            name='cartographer_node',
            output='screen',
            parameters=[{'use_sim_time': use_sim_time}],
            arguments=['-configuration_directory', configuration_directory,
                       '-configuration_basename', configuration_basename],
            remappings=[('/odom','/odom')]
                       ),

        DeclareLaunchArgument(
            'resolution',
            default_value=resolution,
```

```
            description='Resolution of a grid cell in the published occupancy
grid'),

        DeclareLaunchArgument(
            'publish_period_sec',
            default_value=publish_period_sec,
            description='OccupancyGrid publishing period'),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                [ThisLaunchFileDir(), '/occupancy_grid_launch.py']),
            launch_arguments={'use_sim_time': use_sim_time, 'resolution':
resolution,
                              'publish_period_sec': publish_period_sec}.items(),
        ),
    ])
```

Here we mainly run cartographer_node mapping node and occupation_grid_launch.py, and also load the parameter configuration file, which is located at,

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/params
```

lds_2d.lua,

```
include "map_builder.lua"
include "trajectory_builder.lua"

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_footprint",
  published_frame = "odom",
  odom_frame = "odom",
  provide_odom_frame = false,
  publish_frame_projected_to_2d = false,
  use_odometry = true,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
```

```
  landmarks_sampling_ratio = 1.,
}


MAP_BUILDER.use_trajectory_builder_2d = true

TRAJECTORY_BUILDER_2D.use_imu_data = false
TRAJECTORY_BUILDER_2D.min_range = 0.10
TRAJECTORY_BUILDER_2D.max_range = 3.5
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 3.
TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = true
TRAJECTORY_BUILDER_2D.motion_filter.max_angle_radians = math.rad(0.1)

POSE_GRAPH.constraint_builder.min_score = 0.65
POSE_GRAPH.constraint_builder.global_localization_min_score = 0.7

return options
```