Cartographer mapping

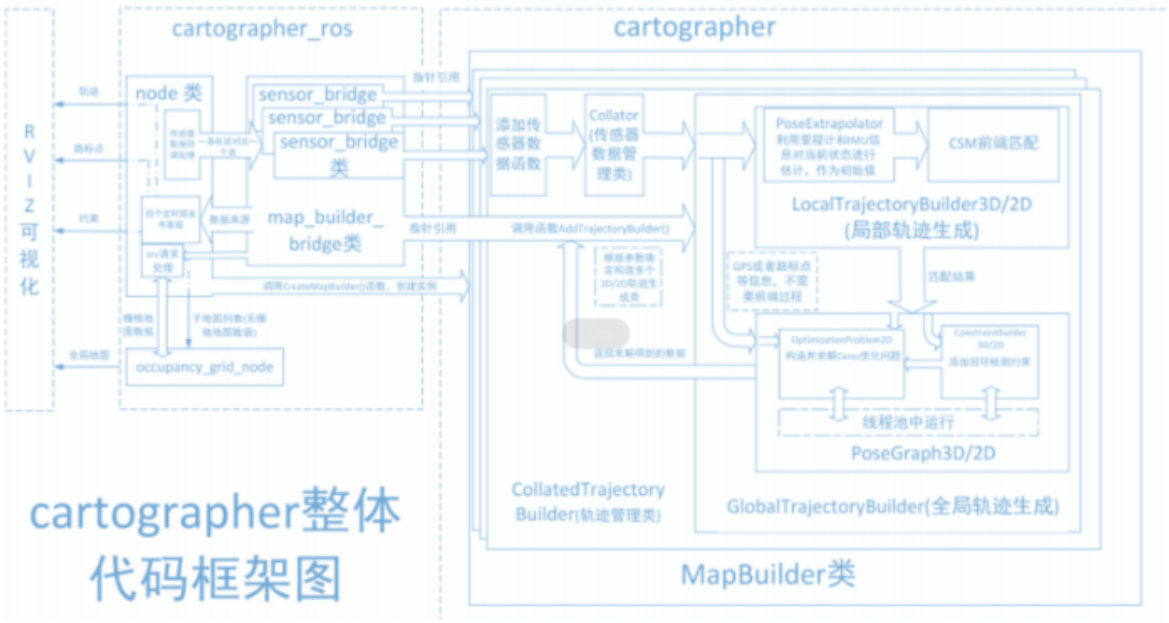
Note: The ROS_DOMAIN_ID of the Raspberry Pi and the microROS control board need to be consistent. You can check [MicroROS Control Board Parameter Configuration] to set the microROS control board ROS_DOMAIN_ID. Check the tutorial [Connect MicroROS Agent] to determine whether the IDs are consistent.

1、Introduction

Cartographer is a 2D and 3D SLAM (simultaneous localization and mapping) library supported by Google's open source ROS system. Method graph construction algorithm based on graph optimization (multi-threaded backend optimization, problem optimization built by CERE). Data from multiple sensors, such as LIDAR, IMUs, 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 Cers-Solver (back-end optimization).

cartographer的源码主要包括三个部分：cartographer、cartographer_ros和ceres-solver（后端优化）。



Cartographer adopts the mainstream SLAM framework, that is, the three-stage of feature extraction, closed-loop detection, and back-end optimization. A certain number of LaserScan forms a submap submap, and a series of submap submaps make up the global map. The cumulative error of the short-term process of building a submap with LaserScan is not large, but the long-term process of building a global map with a submap will have a large cumulative error, so it is necessary to use closed-loop detection to correct the position of these submaps, the basic unit of closed-loop detection is submap, and closed-loop detection adopts a 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 scan_match strategies for closed-loop inspection.

- cartographer_ros

cartographer_ros is running under ROS and can accept various sensor data in the form of ROS messages

After processing, it is published in the form of a message for easy debugging and visualization.

2、Program function description

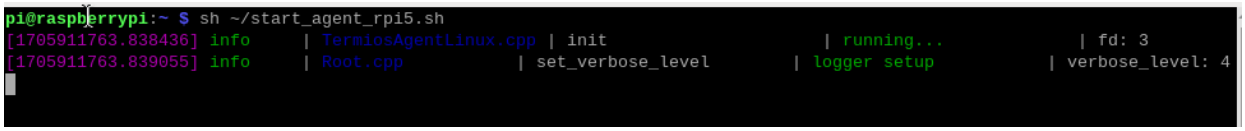
Connect the car to the agent and run the program. The mapping interface will be displayed in rviz. Use the keyboard or handle to control the movement of the car until the map is completed. Then run the save map command to save the map.

3、Query car information

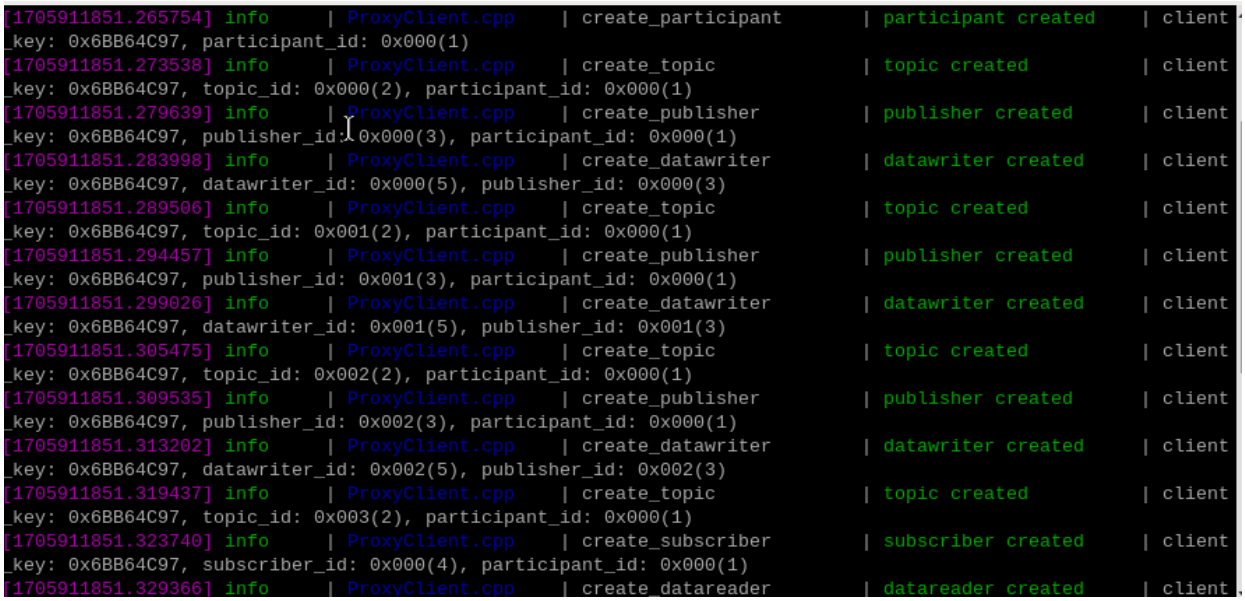
3.1、Start and connect to the agent

After the Raspberry Pi is successfully powered on, open the terminal and enter the following command to open the agent.

```
sh ~/start_agent_rpi5.sh
```



Press the reset button on the microROS control board and wait for the car to connect to the agent. The connection is successful as shown in the figure below.



3.2、Enter the car docker

Open another terminal and enter the following command to enter docker.

```
sh ros2_humble.sh
```

When the following interface appears, you have successfully entered docker. Now you can control the car through commands.

```
pi@raspberrypi:~ $ ./ros2_humble.sh
access control disabled, clients can connect from any host
Successful
MY_DOMAIN_ID: 20
```

4、starting program

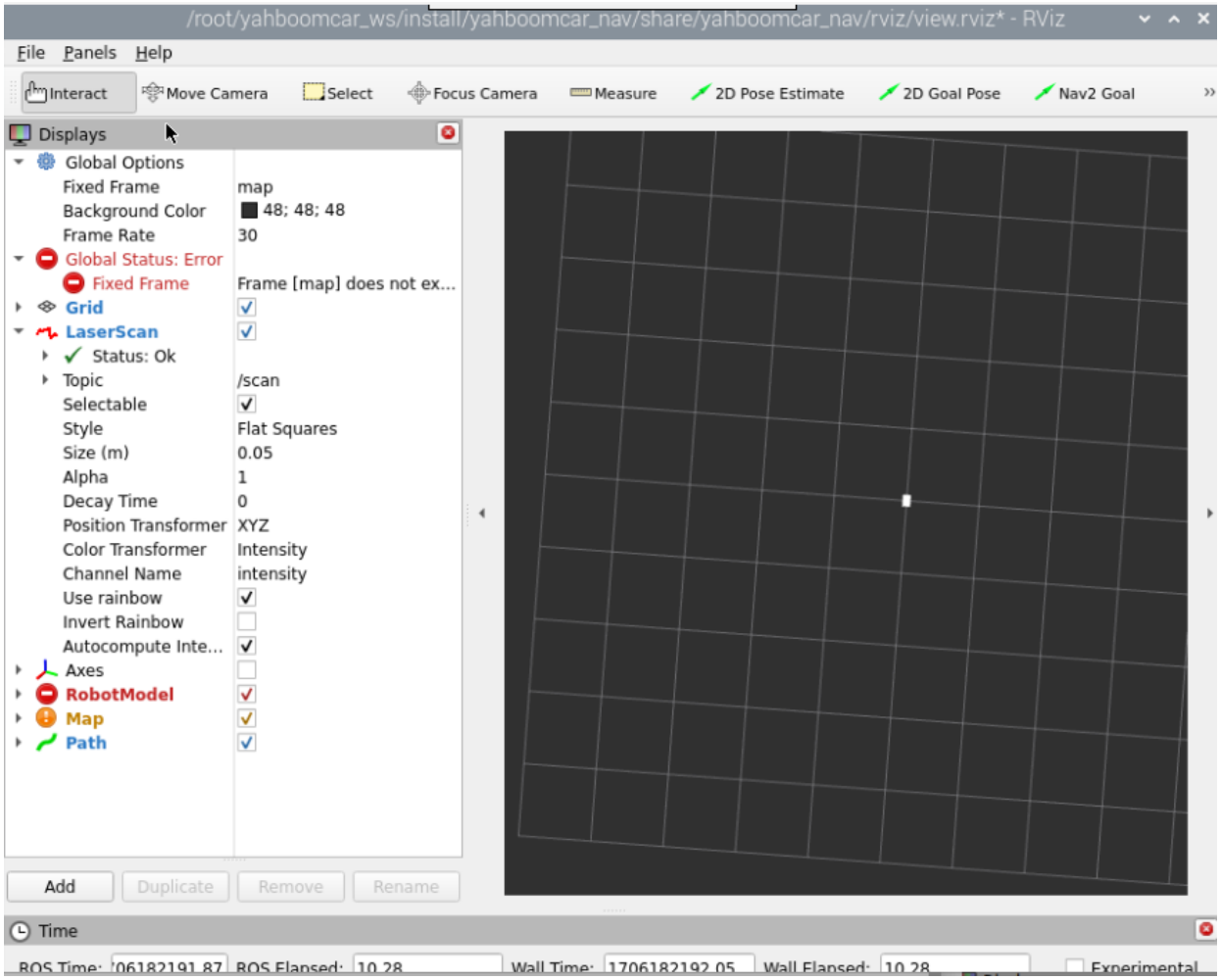
First, start the car to process the underlying data program, and enter the following command in the terminal.

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

```
[INFO] [imu_filter_madgwick_node-1]: process started with pid [6263]
[INFO] [ekf_node-2]: process started with pid [6265]
[INFO] [static_transform_publisher-3]: process started with pid [6267]
[INFO] [joint_state_publisher-4]: process started with pid [6269]
[INFO] [robot_state_publisher-5]: process started with pid [6271]
[INFO] [static_transform_publisher-6]: process started with pid [6286]
[static_transform_publisher-3] [WARN] [1706181650.342105372] []: Old-style arguments are deprecated; see --help for new-style arguments
[static_transform_publisher-3] [INFO] [1706181650.459314055] [base_link_to_base_imu]: Spinning until stopped - publishing transform
[static_transform_publisher-3] translation: ('-0.002999', '-0.003000', '0.031701')
[static_transform_publisher-3] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-3] from 'base_link' to 'imu_frame'
[imu_filter_madgwick_node-1] [INFO] [1706181650.478942143] [imu_filter]: Starting ImuFilter
[imu_filter_madgwick_node-1] [INFO] [1706181650.480114862] [imu_filter]: Using dt computed from message headers
[imu_filter_madgwick_node-1] [INFO] [1706181650.480197121] [imu_filter]: The gravity vector is kept in the IMU message.
[imu_filter_madgwick_node-1] [INFO] [1706181650.480631749] [imu_filter]: Imu filter gain set to 0.100000
[imu_filter_madgwick_node-1] [INFO] [1706181650.480707508] [imu_filter]: Gyro drift bias set to 0.000000
[imu_filter_madgwick_node-1] [INFO] [1706181650.480720249] [imu_filter]: Magnetometer bias values: 0.000000 0.000000 0.000000
[static_transform_publisher-6] [WARN] [1706181650.493533858] []: Old-style arguments are deprecated; see --help for new-style arguments
[robot_state_publisher-5] [WARN] [1706181650.639387954] [kdl_parser]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia.  As a workaround, you can add an extra dummy link to your URDF.
[robot_state_publisher-5] [INFO] [1706181650.640061915] [robot_state_publisher]: got segment base_link
[robot_state_publisher-5] [INFO] [1706181650.640174267] [robot_state_publisher]: got segment imu_Link
[robot_state_publisher-5] [INFO] [1706181650.640191229] [robot_state_publisher]: got segment jq1_Link
[robot_state_publisher-5] [INFO] [1706181650.640201822] [robot_state_publisher]: got segment jq2_Link
[robot_state_publisher-5] [INFO] [1706181650.640211952] [robot_state_publisher]: got segment radar_Link
[robot_state_publisher-5] [INFO] [1706181650.640221211] [robot_state_publisher]: got segment yh_Link
[robot_state_publisher-5] [INFO] [1706181650.640229452] [robot_state_publisher]: got segment yq_Link
[robot_state_publisher-5] [INFO] [1706181650.640238470] [robot_state_publisher]: got segment zh_Link
[robot_state_publisher-5] [INFO] [1706181650.640246655] [robot_state_publisher]: got segment zq_Link
[imu_filter_madgwick_node-1] [INFO] [1706181650.655098332] [imu_filter]: First IMU message received.
[static_transform_publisher-6] [INFO] [1706181650.681177050] [static_transform_publisher_JarNTEa1OrW2k0Zb]: Spinning until stopped - publishing transform
[static_transform_publisher-6] translation: ('0.000000', '0.000000', '0.050000')
[static_transform_publisher-6] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-6] from 'base_footprint' to 'base_link'
[joint_state_publisher-4] [INFO] [1706181650.989117137] [joint_state_publisher]: Waiting for robot_description to be published on the robot_description topic...
```

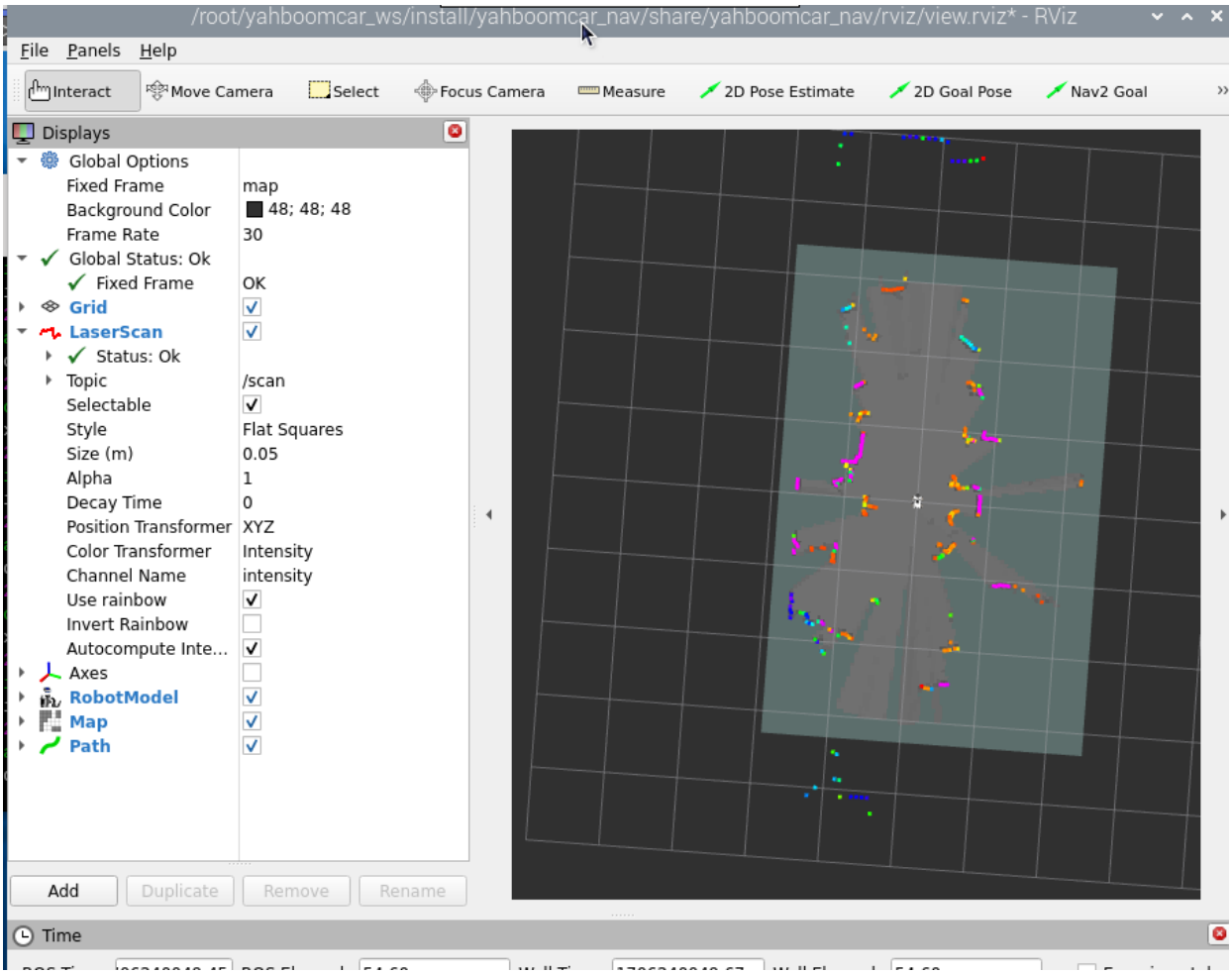
Then enter the following command in the terminal to start rviz for visual mapping.

```
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 the following command in the terminal.

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



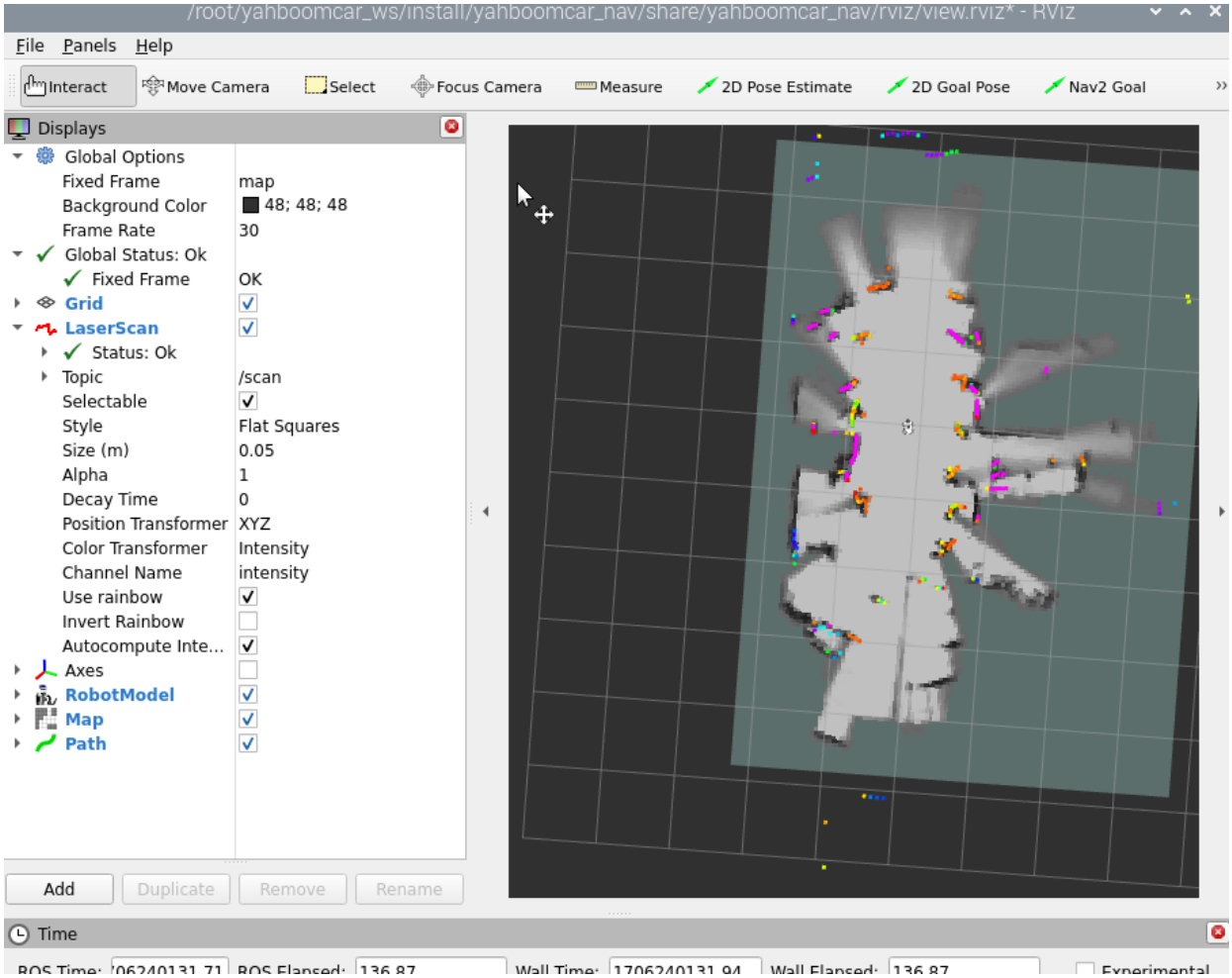
Then run handle control or keyboard control, choose one of the two, and enter the following command on the terminal.

```
#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 the following command on the terminal.

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

```
root@raspberrypi:~# ros2 launch yahboomcar_nav save_map_launch.py
[INFO] [launch]: All log files can be found below /root/.ros/log/2024-01-26-03-36-34-416746-raspberrypi-634
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [map_saver_cli-1]: process started with pid [635]
[map_saver_cli-1] [INFO] [1706240194.634520023] [map_saver]:
[map_saver_cli-1]       map_saver lifecycle node launched.
[map_saver_cli-1]       Waiting on external lifecycle transitions to activate
[map_saver_cli-1]       See https://design.ros2.org/articles/node_lifecycle.html for more information.
[map_saver_cli-1] [INFO] [1706240194.634715356] [map_saver]: Creating
[map_saver_cli-1] [INFO] [1706240194.634827355] [map_saver]: Configuring
[map_saver_cli-1] [INFO] [1706240194.637876840] [map_saver]: Saving map from 'map' topic to '/root/yahboomcar_ws/src/yahboomcar_nav/maps/yahboom_map' file
[map_saver_cli-1] [WARN] [1706240194.637935284] [map_saver]: Free threshold unspecified. Setting it to default value: 0.250000
[map_saver_cli-1] [WARN] [1706240194.637956024] [map_saver]: Occupied threshold unspecified. Setting it to default value: 0.650000
[map_saver_cli-1] [INFO] [1706240195.279553187] [map_saver]: Map saved successfully
[map_saver_cli-1] [INFO] [1706240195.280750533] [map_saver]: Destroying
[INFO] [map_saver_cli-1]: process has finished cleanly [pid 635]
root@raspberrypi:~#
```



A map named yahboom_map will be saved. This map is saved in the path below.

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

Two files will be generated, one is yahboom_map.pgm and the other is yahboom_map.yaml. Below are some contents in the yaml file.

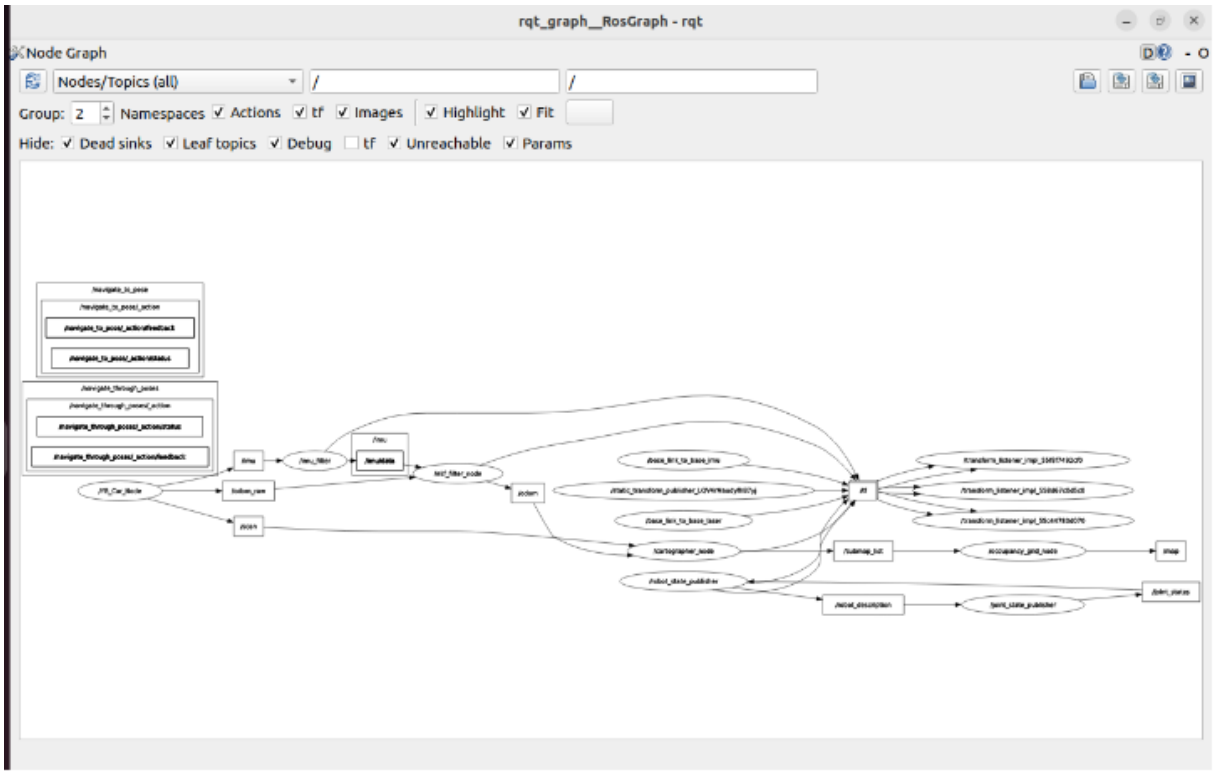
```
image: yahboom_map.pgm
mode: trinary
resolution: 0.05
origin: [-10, -10, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

- 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 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 node communication diagram

Enter the following command in the terminal

```
ros2 run rqt_graph rqt_graph
```



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

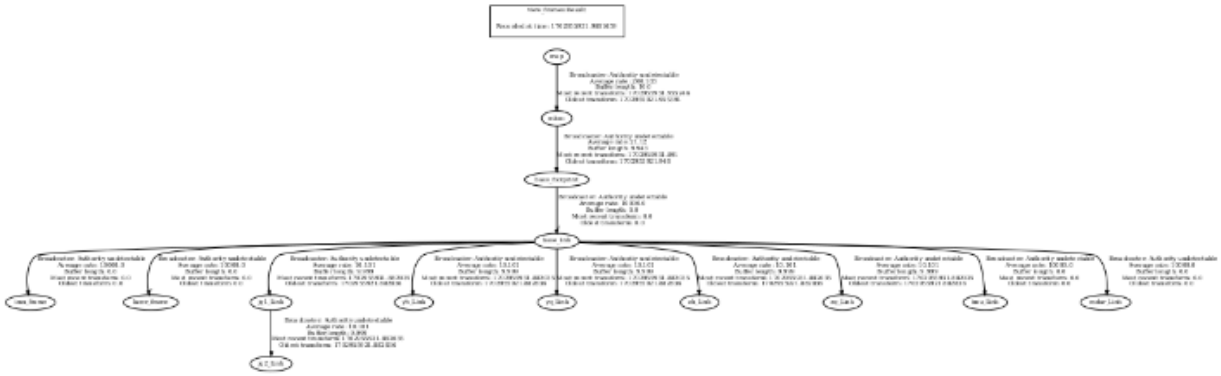
6、View TF tree

Enter the following command in the terminal.

```
ros2 run tf2_tools view_frames
```

```
root@raspberrypi:~# ros2 run tf2_tools view_frames
[INFO] [1706240274.802480879] [view_frames]: Listening to tf data for 5.0 seconds...
[INFO] [1706240279.805539455] [view_frames]: Generating graph in frames.pdf file...
[INFO] [1706240279.811022965] [view_frames]: Result:tf2_msgs.srv.FrameGraph_Response(frame_yaml=
"odom: \n  parent: 'map'\n  broadcaster: 'default_authority'\n  rate: 193.930\n  most_recent_tra
nsform: 1706240279.804413\n  oldest_transform: 1706240274.771664\n  buffer_length: 5.033\nimu_fr
ame: \n  parent: 'base_link'\n  broadcaster: 'default_authority'\n  rate: 10000.000\n  most_rece
nt_transform: 0.000000\n  oldest_transform: 0.000000\n  buffer_length: 0.000\nbase_link: \n  par
ent: 'base_footprint'\n  broadcaster: 'default_authority'\n  rate: 10000.000\n  most_recent_tran
sform: 0.000000\n  oldest_transform: 0.000000\n  buffer_length: 0.000\nbase_footprint: \n  paren
t: 'odom'\n  broadcaster: 'default_authority'\n  rate: 19.545\n  most_recent_transform: 17062402
79.773000\n  oldest_transform: 1706240274.810000\n  buffer_length: 4.963\nlaser_frame: \n  paren
t: 'base_link'\n  broadcaster: 'default_authority'\n  rate: 10000.000\n  most_recent_transform:
0.000000\n  oldest_transform: 0.000000\n  buffer_length: 0.000\njq1_Link: \n  parent: 'base_link
'\n  broadcaster: 'default_authority'\n  rate: 10.204\n  most_recent_transform: 1706240279.71725
1\n  oldest_transform: 1706240274.817256\n  buffer_length: 4.900\njq2_Link: \n  parent: 'jq1_Lin
k'\n  broadcaster: 'default_authority'\n  rate: 10.204\n  most_recent_transform: 1706240279.7172
51\n  oldest_transform: 1706240274.817256\n  buffer_length: 4.900\nyh_Link: \n  parent: 'base_li
nk'\n  broadcaster: 'default_authority'\n  rate: 10.204\n  most_recent_transform: 1706240279.717
251\n  oldest_transform: 1706240274.817256\n  buffer_length: 4.900\nyq_Link: \n  parent: 'base_l
ink'\n  broadcaster: 'default_authority'\n  rate: 10.204\n  most_recent_transform: 1706240279.71
7251\n  oldest_transform: 1706240274.817256\n  buffer_length: 4.900\nzh_Link: \n  parent: 'base_
link'\n  broadcaster: 'default_authority'\n  rate: 10.204\n  most_recent_transform: 1706240279.7
17251\n  oldest_transform: 1706240274.817256\n  buffer_length: 4.900\nzq_Link: \n  parent: 'base
_link'\n  broadcaster: 'default_authority'\n  rate: 10.204\n  most_recent_transform: 1706240279.
717251\n  oldest_transform: 1706240274.817256\n  buffer_length: 4.900\nimu_Link: \n  parent: 'ba
se_link'\n  broadcaster: 'default_authority'\n  rate: 10000.000\n  most_recent_transform: 0.0000
00\n  oldest_transform: 0.000000\n  buffer_length: 0.000\nradar_Link: \n  parent: 'base_link'\n
 broadcaster: 'default_authority'\n  rate: 10000.000\n  most_recent_transform: 0.000000\n  oldes
t_transform: 0.000000\n  buffer_length: 0.000\n")
root@raspberrypi:~#
```

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.

```
/root/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 check cartographer_launch, the file path is as follows.

```
/root/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 the cartographer_node mapping node and occupation_grid_launch.py, and also load the parameter configuration file. The path to the parameter file is as follows.

```
/root/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
```