Cartographer Repositioning navigation

1. Course Content

1. Learn the robot's rapid relocalization and navigation capabilities
2. When you run the program, a real-time map and robot model will load into RVIZ. The robot can quickly locate its position on the map and perform real-time SLAM mapping and navigation.

2. Principle Overview

2.1 Introduction

- Navigation 2 (Nav 2) is the navigation framework included with ROS 2. Its purpose is to safely move mobile robots from point A to point B.
- The default localization algorithm for Navigation 2 is AMCL (Adaptive Monte Carlo Localization). This will replace the default AMCL localization method with Cartographer localization, while other navigation functions remain unchanged.

2.2 Differences Between the Two Localization Methods

- **AMCL**: Suitable for scenarios with a known map. For example, in a mapped indoor environment, a robot can use AMCL to determine its position when performing repetitive tasks or navigating.
- **Cartographer**: More suitable for scenarios in unknown environments where a robot needs to simultaneously build a map and localize itself. For example, when a robot enters a new area for the first time and needs to quickly build a map and determine its position, Cartographer is more effective.

3. Preparation

3.1 Content Description

This lesson uses the Raspberry Pi as an example.

For Raspberry Pi and Jetson-Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to **[01. Robot Configuration and Operation Guide] -- [4.Enter Docker (For JETSON Nano and RPi 5)]**.

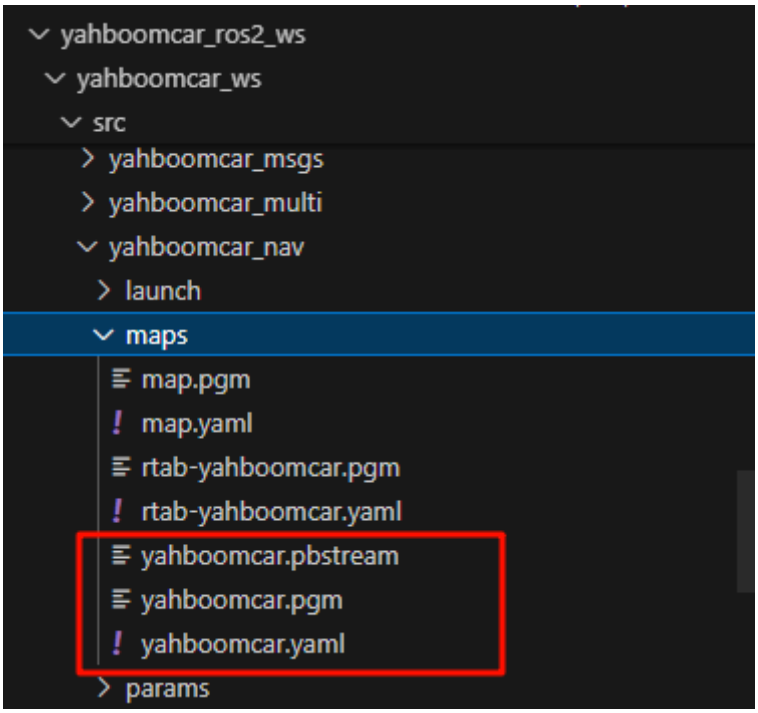
For RDKX5 and Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

3.2 Map Preparation

- The fast relocation navigation in this course requires you to first follow the tutorial [11.LiDAR Course] - [6. Cartographer Mapping] to save a pbstream format map and a yaml file. The pbstream and yaml files will be saved in

  ```
  ~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pbstream
  ~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.yaml
  ```

Directory (Raspberry Pi and Jetson Nano are saved in the same directory in Docker).



3.3 Navigation Function

Note:

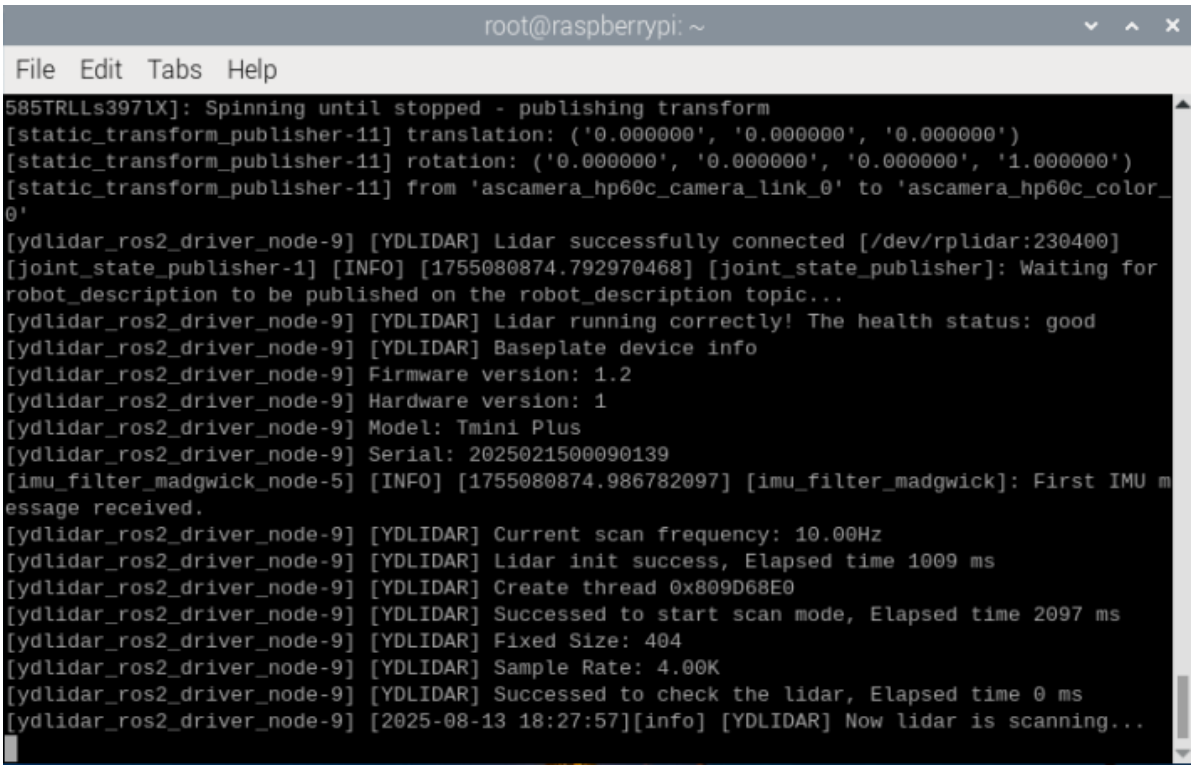
For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

Enter the Docker container (see [20. Docker course] --- [4. Docker Startup Script] for steps).

All the following commands must be executed within the same Docker container **(see [20. Docker course] --- [3. Docker Submission and Multi-Terminal Access] for steps).**

The robot vehicle terminal activates the underlying sensor command:

```
ros2 launch yahboomcar_nav laser_bringup_launch.py
```

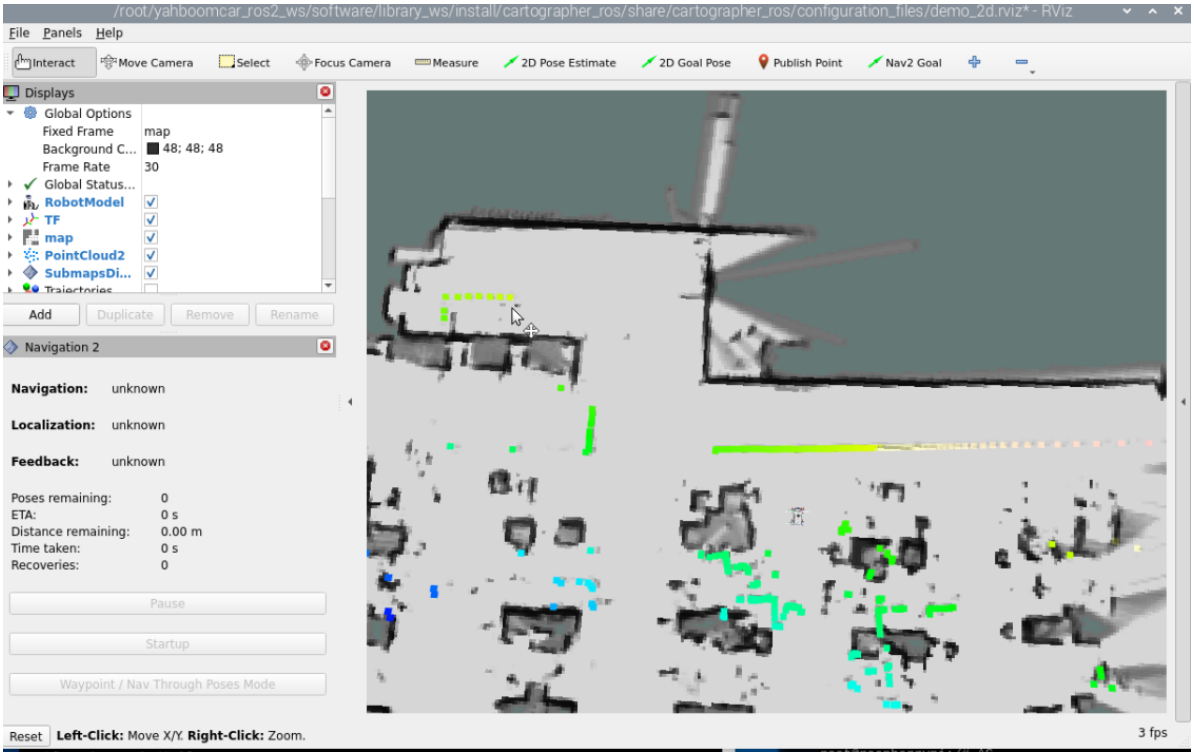


Enter the command to start RViz visualization for mapping. Then start the cartographer node for localization; change `use_rviz:=true` to `true` to enable RViz visualization.

```
#Raspberry Pi, Jetson Nano
ros2 launch yahboomcar_nav localization_imu_odom.launch.py use_rviz:=true
load_state_filename:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/m
aps/yahboomcar.pbstream
#ORIN
ros2 launch yahboomcar_nav localization_imu_odom.launch.py use_rviz:=true
load_state_filename:=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomca
r_nav/maps/yahboomcar.pbstream
#RDK X5
ros2 launch yahboomcar_nav localization_imu_odom.launch.py use_rviz:=true
load_state_filename:=/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomc
ar_nav/maps/yahboomcar.pbstream
```

> Parameter Description

```
# Select whether to enable rviz. True to enable, false to disable.
use_rviz:=true
# Replace the target map file.
# .pbstream map files refer to the cartograph mapping algorithm to save the map.
load_state_filename:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/m
aps/yahboomcar.pbstream
```



You'll see the map has loaded, but the car's positioning isn't accurate. Click [2D Pose Estimate] to set the car's initial pose. Based on the car's approximate position in the real environment, click and drag the mouse in rviz to quickly locate it on the map. As shown below, the radar scan area roughly overlaps with the actual obstacle.

Note: If positioning is still not possible, the car may be in an obstruction area. Place it in a clear area with no obstacles nearby and restart the car. Alternatively, leave the terminal open, restart the car, and wait for the data to automatically recover before relocating. Alternatively, the map may have too few feature points during construction, requiring more detailed mapping.

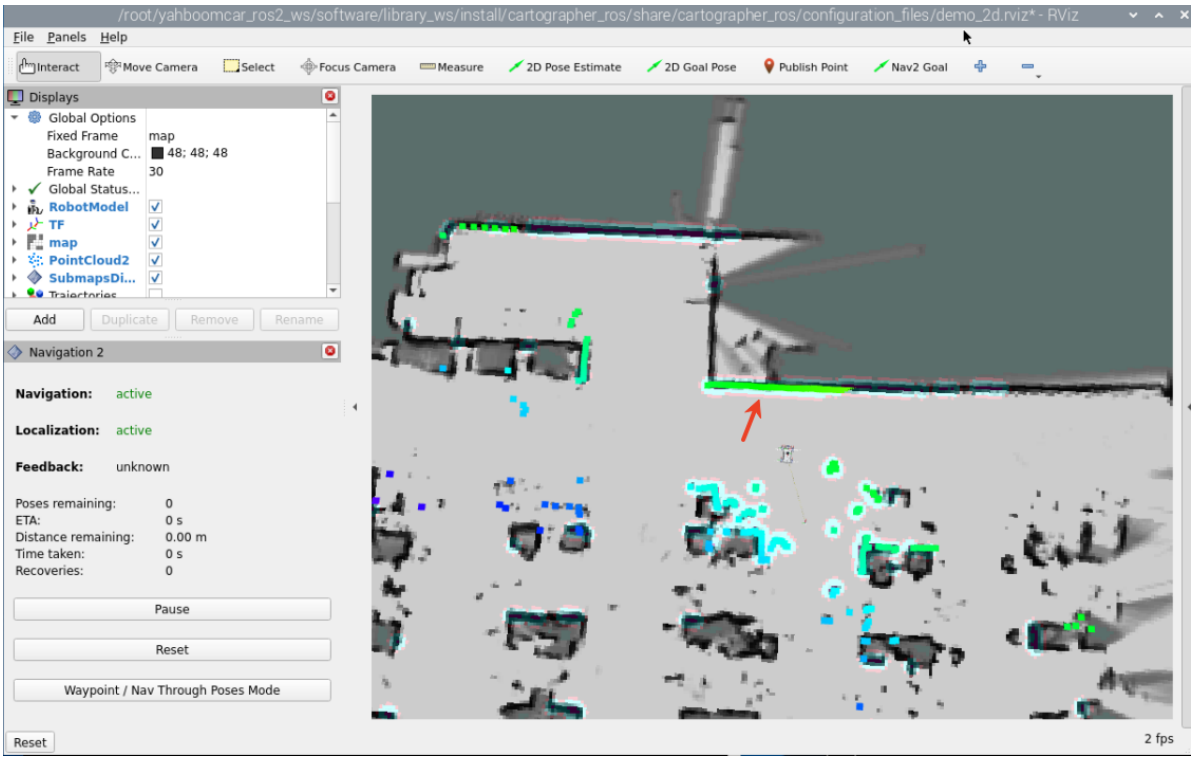
Finally, launch Navigation 2.

```
#Raspberry Pi, Jetson Nano
ros2 launch yahboomcar_nav navigation_cartodwb_launch.py
maps:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.
yaml
#ORIN
ros2 launch yahboomcar_nav navigation_cartodwb_launch.py
maps:=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahb
oomcar.yaml
#RDK X
ros2 launch yahboomcar_nav navigation_cartodwb_launch.py
maps:=/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yah
boomcar.yaml
```

> Parameter Description

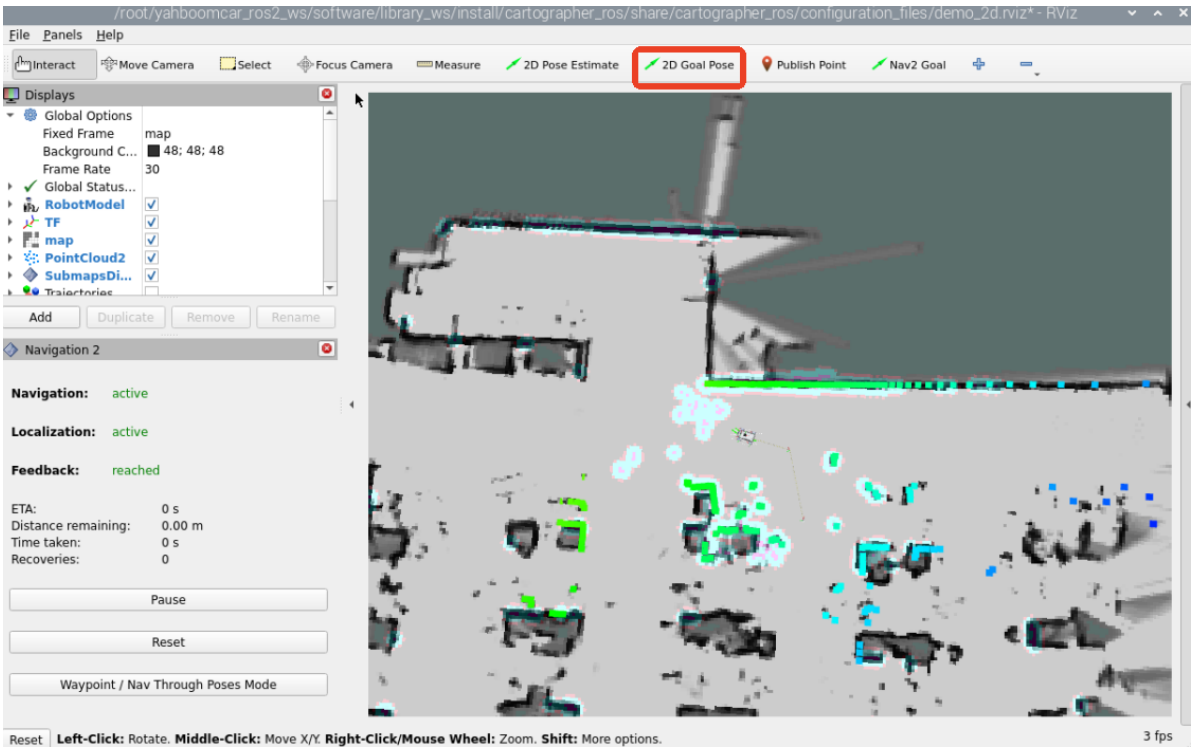
```
# Load map parameters: (Can replace the target map)
maps:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.
yaml
# The .pgm file must be in the same directory as the .yaml file.
```

Note: The yahboomcar.yaml file and the yahboomcar.pbstream file must be created at the same time, meaning they are the same map. Refer to the cartograph mapping algorithm for saving maps.

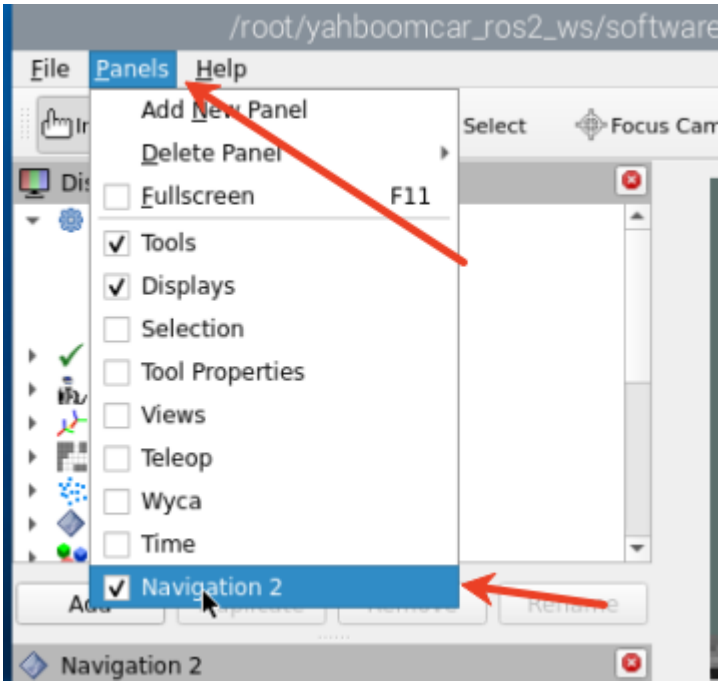


Navigation can begin when an expansion area appears.

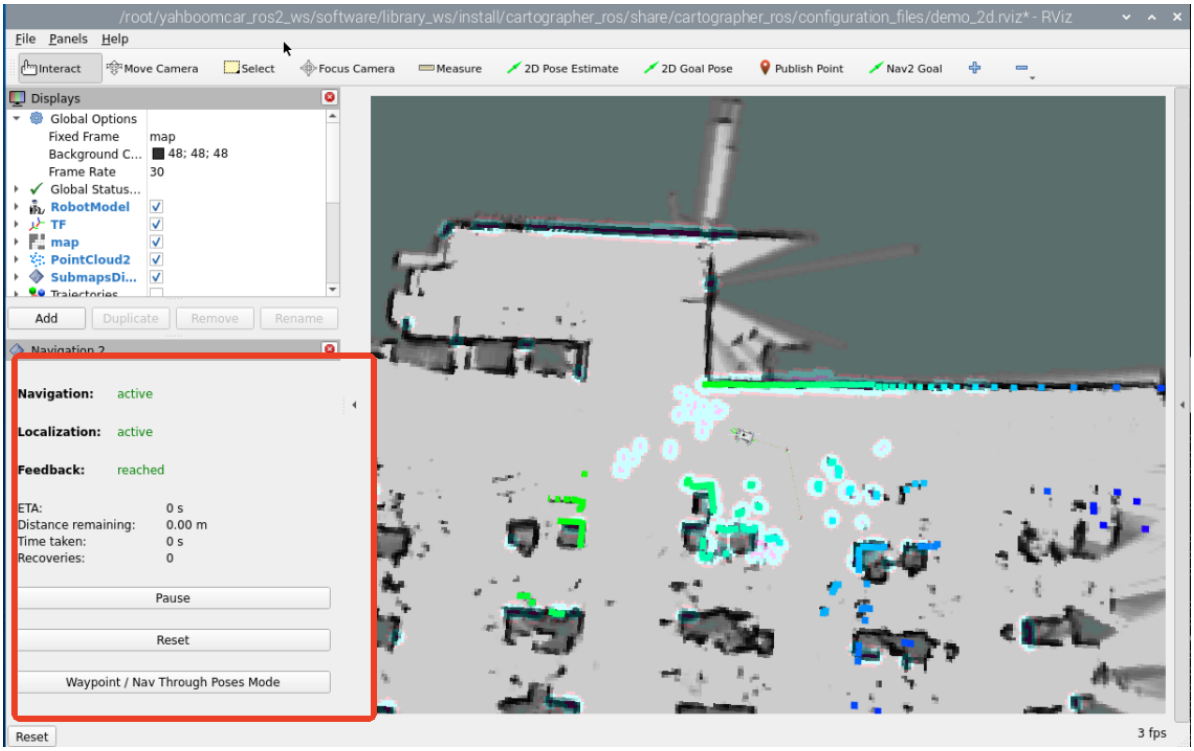
For single-point navigation, click the 【2D Goal Pose】 tool and select a target point in rviz. The robot will calculate a path based on the surroundings and move along it to the target point.



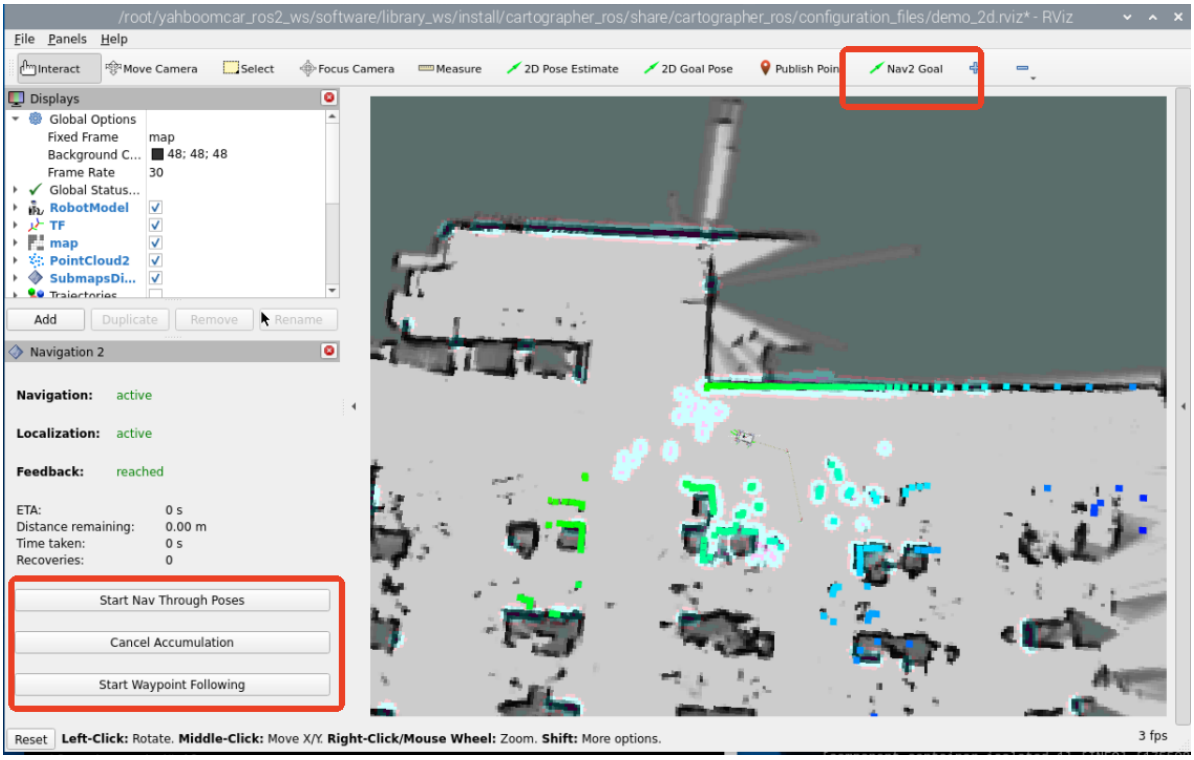
To enable multi-point navigation, you need to add the nav2 plugin.



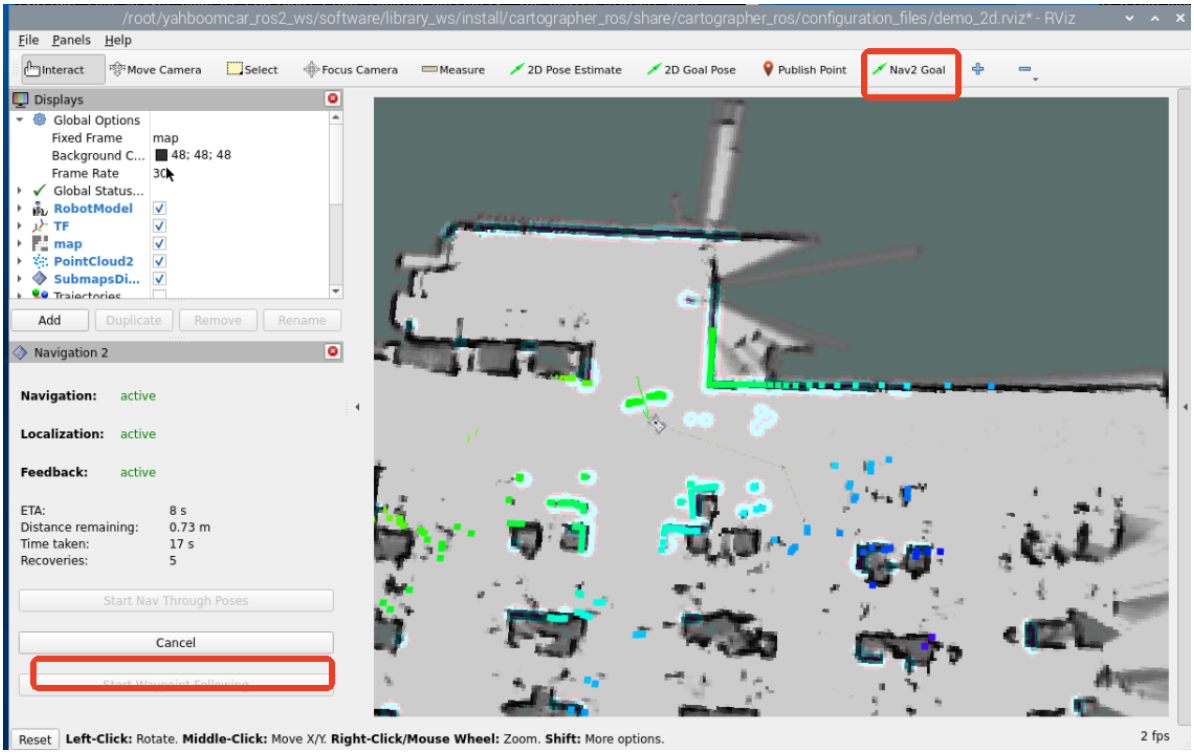
After adding this, rviz displays the following:



Then click 【Waypoint/Nav Through Poses Mode】 and use【Nav2 Goal】 in the rivz toolbar to set any target point.



Then click 【Start Waypoint Following】 to begin route navigation. The car will automatically proceed to the next point after reaching the target point, according to the order of the selected points. No operation is required. After reaching the last point, the car stops and waits for the next instruction.

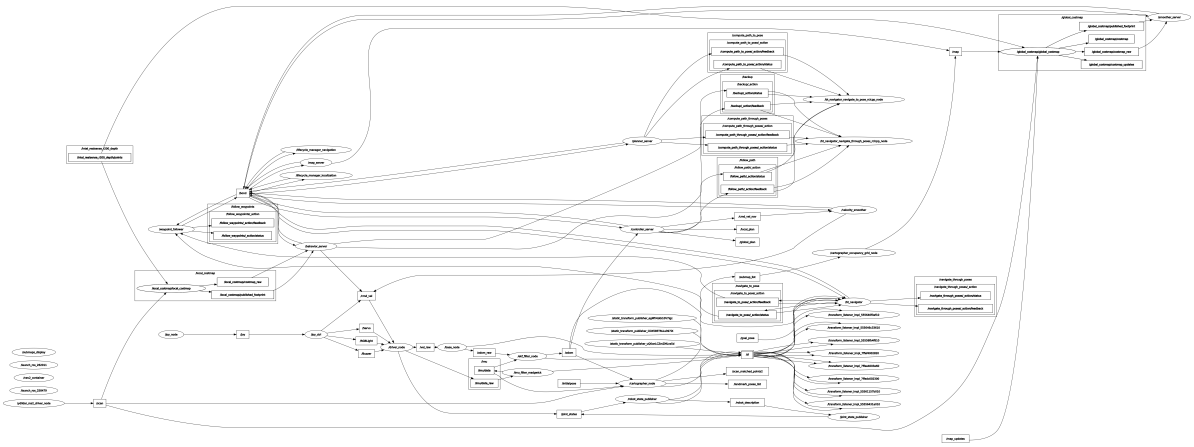


4. View the Node Communication Graph

Enter in the terminal:

```
ros2 run rqt_graph rqt_graph
```

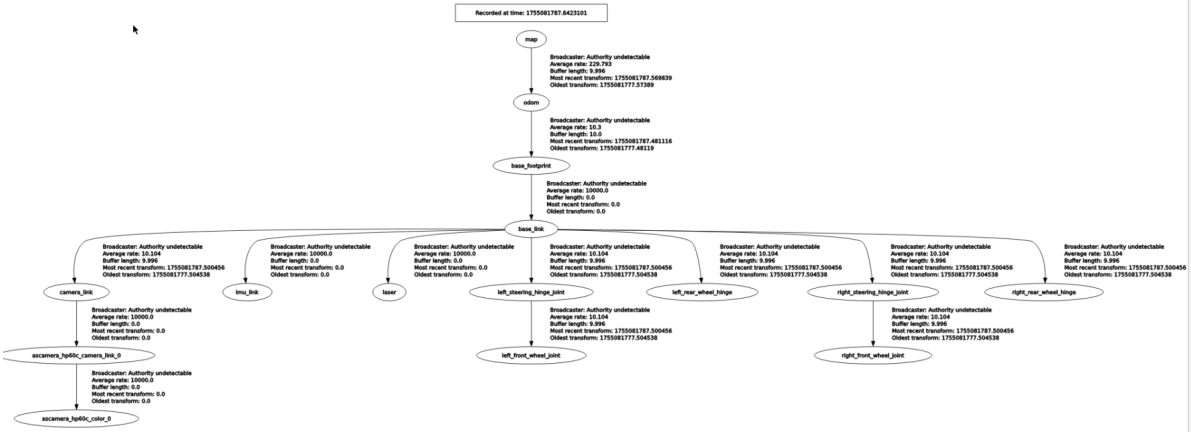
If the graph doesn't appear initially, select [Nodes/Topics (all)] and click the refresh button in the upper left corner.



5. View the TF tree

Enter the terminal:

```
ros2 run rqt_tf_tree rqt_tf_tree
```



6. Principle Explanation

The key to fast re-localization navigation is replacing the default amcl positioning method in navigation2 with Cartographer's positioning method. All other details remain unchanged. The following explains the replacement method.

Source code location:

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/launch/
```

Find the `cartographer_localization.launch.py` file in the launch directory, and its contents are as follows:

- This file starts the cartographer node, replacing the default positioning method in navigation2. The default navigation2 startup file has been modified to remove the node that originally started the amcl positioning method, while the rest of the navigation stack remains unchanged.

```
# Copyright (c) 2018 Intel Corporation
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import os

from ament_index_python.packages import get_package_share_directory

from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, GroupAction,
SetEnvironmentVariable
from launch.conditions import IfCondition
```

```
from launch.substitutions import LaunchConfiguration, PythonExpression
from launch_ros.actions import LoadComposableNodes
from launch_ros.actions import Node
from launch_ros.descriptions import ComposableNode, ParameterFile
from nav2_common.launch import RewrittenYaml


def generate_launch_description():
    # Get the launch directory
    bringup_dir = get_package_share_directory('nav2_bringup')

    namespace = LaunchConfiguration('namespace')
    map_yaml_file = LaunchConfiguration('map')
    use_sim_time = LaunchConfiguration('use_sim_time')
    autostart = LaunchConfiguration('autostart')
    params_file = LaunchConfiguration('params_file')
    use_composition = LaunchConfiguration('use_composition')
    container_name = LaunchConfiguration('container_name')
    container_name_full = (namespace, '/', container_name)
    use_respawn = LaunchConfiguration('use_respawn')
    log_level = LaunchConfiguration('log_level')

    # Remove 'amcl' from lifecycle_nodes
    lifecycle_nodes = ['map_server']

    # Map fully qualified names to relative ones so the node's namespace can be
prepended.
    # In case of the transforms (tf), currently, there doesn't seem to be a
better alternative
    # https://github.com/ros/geometry2/issues/32
    # https://github.com/ros/robot_state_publisher/pull/30
    # TODO(orduno) Substitute with `PushNodeRemapping`
    #              https://github.com/ros2/launch_ros/issues/56
    remappings = [('/tf', 'tf'),
                  ('/tf_static', 'tf_static')]

    # Create our own temporary YAML files that include substitutions
    param_substitutions = {
        'use_sim_time': use_sim_time,
        'yaml_filename': map_yaml_file}

    configured_params = ParameterFile(
        RewrittenYaml(
            source_file=params_file,
            root_key=namespace,
            param_rewrites=param_substitutions,
            convert_types=True),
        allow_substs=True)

    stdout_linebuf_envvar = SetEnvironmentVariable(
        'RCUTILS_LOGGING_BUFFERED_STREAM', '1')

    declare_namespace_cmd = DeclareLaunchArgument(
        'namespace',
        default_value='',
        description='Top-level namespace')

    declare_map_yaml_cmd = DeclareLaunchArgument(
```

```
        'map',
        description='Full path to map yaml file to load')

    declare_use_sim_time_cmd = DeclareLaunchArgument(
        'use_sim_time',
        default_value='false',
        description='Use simulation (Gazebo) clock if true')

    declare_params_file_cmd = DeclareLaunchArgument(
        'params_file',
        default_value=os.path.join(bringup_dir, 'params', 'nav2_params.yaml'),
        description='Full path to the ROS2 parameters file to use for all
launched nodes')

    declare_autostart_cmd = DeclareLaunchArgument(
        'autostart', default_value='true',
        description='Automatically startup the nav2 stack')

    declare_use_composition_cmd = DeclareLaunchArgument(
        'use_composition', default_value='False',
        description='Use composed bringup if True')

    declare_container_name_cmd = DeclareLaunchArgument(
        'container_name', default_value='nav2_container',
        description='the name of conatiner that nodes will load in if use
composition')

    declare_use_respawn_cmd = DeclareLaunchArgument(
        'use_respawn', default_value='False',
        description='Whether to respawn if a node crashes. Applied when
composition is disabled.')

    declare_log_level_cmd = DeclareLaunchArgument(
        'log_level', default_value='info',
        description='log level')

    load_nodes = GroupAction(
        condition=IfCondition(PythonExpression(['not ', use_composition])),
        actions=[
            Node(
                package='nav2_map_server',
                executable='map_server',
                name='map_server',
                output='screen',
                respawn=use_respawn,
                respawn_delay=2.0,
                parameters=[configured_params],
                arguments=['--ros-args', '--log-level', log_level],
                remappings=remappings),
            Node(
                package='nav2_lifecycle_manager',
                executable='lifecycle_manager',
                name='lifecycle_manager_localization',
                output='screen',
                arguments=['--ros-args', '--log-level', log_level],
                parameters=[{'use_sim_time': use_sim_time},
                            {'autostart': autostart},
                            {'node_names': lifecycle_nodes}])
```

```
        ]
    )

    load_composable_nodes = LoadComposableNodes(
        condition=IfCondition(use_composition),
        target_container=container_name_full,
        composable_node_descriptions=[
            ComposableNode(
                package='nav2_map_server',
                plugin='nav2_map_server::MapServer',
                name='map_server',
                parameters=[configured_params],
                remappings=remappings),
            ComposableNode(
                package='nav2_lifecycle_manager',
                plugin='nav2_lifecycle_manager::LifecycleManager',
                name='lifecycle_manager_localization',
                parameters=[{'use_sim_time': use_sim_time,
                             'autostart': autostart,
                             'node_names': lifecycle_nodes}]),
        ],
    )

    # Create the launch description and populate
    ld = LaunchDescription()

    # Set environment variables
    ld.add_action(stdout_linebuf_envvar)

    # Declare the launch options
    ld.add_action(declare_namespace_cmd)
    ld.add_action(declare_map_yaml_cmd)
    ld.add_action(declare_use_sim_time_cmd)
    ld.add_action(declare_params_file_cmd)
    ld.add_action(declare_autostart_cmd)
    ld.add_action(declare_use_composition_cmd)
    ld.add_action(declare_container_name_cmd)
    ld.add_action(declare_use_respawn_cmd)
    ld.add_action(declare_log_level_cmd)

    # Add the actions to launch all of the localiztion nodes
    ld.add_action(load_nodes)
    ld.add_action(load_composable_nodes)

    return ld
```

Find the `navigation_cartodwb_launch.py` file in the launch directory. The contents are as follows:

The following nodes are started here:

- base_link_to_laser_tf: Publishes static TF transforms;
- map_yaml_path: Loads the map file. The default value is `/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_navmaps/yahboom_map.yaml`, which specifies the path to the map file to load;
- cartographer_bringup_launch.py: Launches the Cartographer positioning-only launch file. The file is located at

`/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/launch/cartographer_bringup_launch.py`

- ORIN , located in /home/jetson/
- RDK X5 , located in /home/sunrise/

A navigation parameter configuration file, cartoteb_nav_params.yaml, is also loaded.

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

def generate_launch_description():
    package_path = get_package_share_directory('yahboomcar_nav')

    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    namespece = LaunchConfiguration('namespece', default='')
    map_yaml_path = LaunchConfiguration(
        'maps',
default=os.path.join('/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav'
, 'maps', 'yahboomcar.yaml'))
    nav2_param_path = LaunchConfiguration('params_file', default=os.path.join(
        package_path, 'params', 'cartoteb_nav_params.yaml'))

    return LaunchDescription([
        DeclareLaunchArgument('use_sim_time', default_value=use_sim_time,
                              description='Use simulation (Gazebo) clock if
true'),
        DeclareLaunchArgument('namespece', default_value=namespece,
                              description='Use simulation (Gazebo) clock if
true'),
        DeclareLaunchArgument('maps', default_value=map_yaml_path,
                              description='Full path to map file to load'),
        DeclareLaunchArgument('params_file', default_value=nav2_param_path,
                              description='Full path to param file to load'),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                [package_path, '/launch', '/cartographer_bringup_launch.py']),
            launch_arguments={
                'map': map_yaml_path,
                'use_sim_time': use_sim_time,
                'namespece': namespece,
                'params_file': nav2_param_path}.items(),
        ),
    ])
```