cartographer mapping

This chapter needs to be used with a car chassis and other sensors to operate. This is just an explanation of the implementation method. In fact, it cannot be run. It needs to be used with the company's rosmaster-X3 car to achieve this function.

If you need to transplant it to your own motherboard, you need to install other dependency files.

1. Function Description

After the program is started, control the car through the handle or keyboard.

The car will use radar scanning data while moving, use the cartographer algorithm to build a map, and save the map after the construction is completed.

2. Install cartographer

Taking ros2-foxy as an example, input following command:

```
sudo apt install ros-foxy-cartographer* -y
```

3. Code path

The source code of this function is located in the supporting virtual machine system.

```
~/ydlidar_ws/src/yahboomcar_nav/launch
```

4. Start up

4.1 Mapping

Input following command:

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

4.2 Save map

Input following command:

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

The map saving path is as follows:

```
~/ydlidar_ws/src/yahboomcar_nav/maps
```

There is a .pgm image and a .yaml file.

The contents of the yaml file are as follows:

```
image: /home/yahboom/rplidar_ws/src/yahboomcar_nav/maps/yahboom_map.pgm
mode: trinary
resolution: 0.05
origin: [-5.95, -3.26, 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: resolution of the map, meters/pixel
- Origin: 2D pose (x, y, yaw) in the lower left corner of the map. The yaw here 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 occupation probability greater than this threshold will be considered fully occupied.
- free_thresh: Pixels with occupancy probability less than this threshold will be considered completely free.

5. Code analysis

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


def generate_launch_description():

    laser_bringup_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([os.path.join(
        get_package_share_directory('yahboomcar_nav'), 'launch'),
         '/laser_bringup_launch.py'
        ])
    )

    cartographer_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([os.path.join(
        get_package_share_directory('yahboomcar_nav'), 'launch'),
         '/cartographer_launch.py'
        ])
    )

    return LaunchDescription([laser_bringup_launch, cartographer_launch])
```

laser_bringup_launch.py starts the radar chassis,

cartographer_launch.py starts gmapping mapping related nodes.

cartographer_launch

```
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='cartographer_config.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]),

        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(),
        ),
    ])
```