Cartographer-SLAM mapping

This lesson uses the Raspberry Pi as an example.

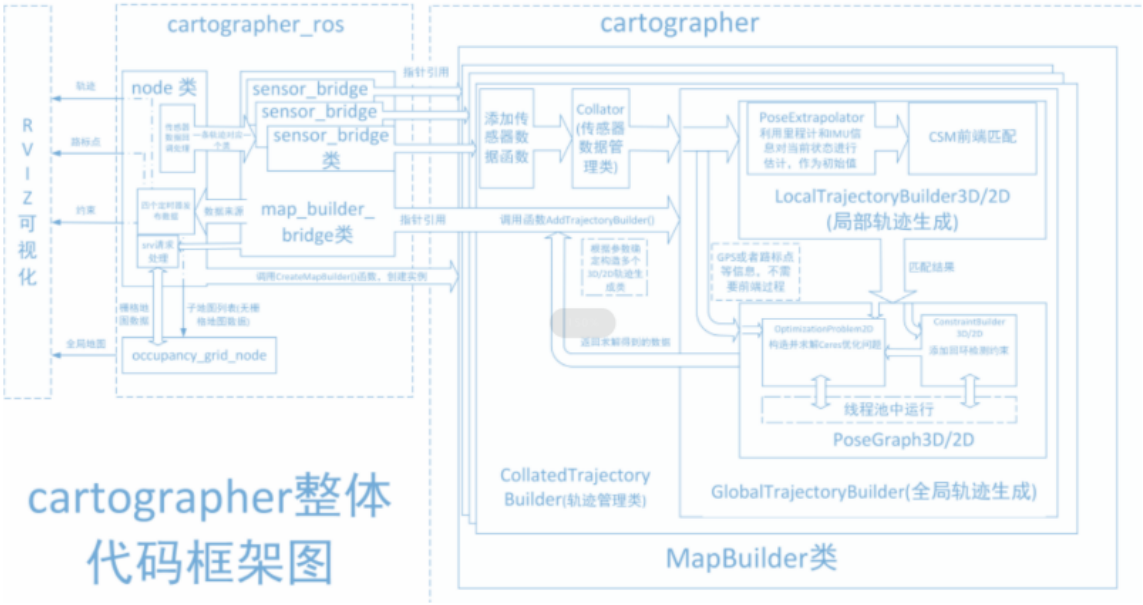
For Raspberry Pi and Jetson-Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host 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.

1. Algorithm Introduction

Cartographer is an open-source 2D and 3D SLAM (simultaneous localization and mapping) library from Google, supported by the ROS system. This mapping algorithm uses graph optimization (multi-threaded backend optimization and problem optimization built using Cere). It combines data from multiple sensors (such as LIDAR, IMU, and cameras) to simultaneously calculate the sensor positions and map the surrounding environment.

The Cartographer source code consists of three main parts: cartographer, cartographer_ros, and ceres-solver (backend optimization).



Cartographer uses a mainstream SLAM framework, specifically a three-step process consisting of feature extraction, loop closure detection, and backend optimization. A certain number of laser scans form a submap, and a series of submaps form the global map. While the short-term cumulative error of constructing submaps using laser scans is small, the long-term cumulative error of constructing the global map using submaps is significant. Therefore, loop closure detection is required to correct the positions of these submaps. The basic unit of loop closure detection is the submap, which uses the scan_match strategy. Cartographer focuses on creating submaps that fuse multi-sensor data (odometry, IMU, laser scans, etc.) and implementing the scan_match strategy for loop closure detection.

cartographer_ros runs under ROS and receives various sensor data as ROS messages. After processing, it publishes the data as messages, facilitating debugging and visualization.

2. Program Functionality

After running the program, the map creation interface will appear in rviz. Use the keyboard or gamepad to control the car's movements until the map is complete. Then, run the save map command to save the map.

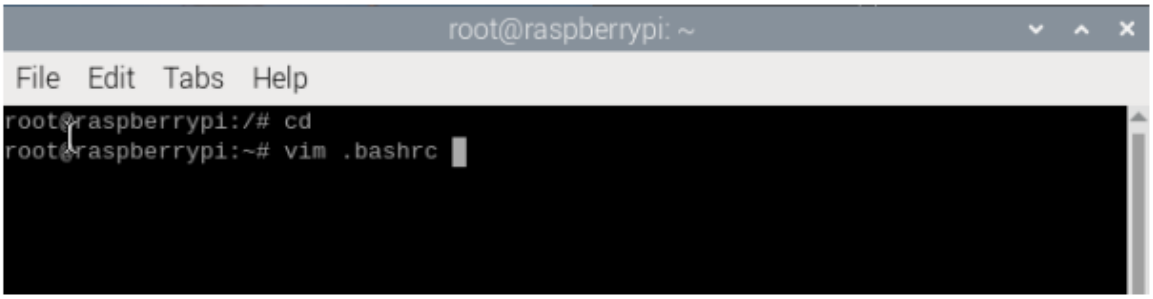
3. Pre-Use Configuration

This car is equipped with a USB camera, a depth camera, and two different types of lidar. However, since it cannot automatically identify the devices, you need to manually set the machine type and lidar model.

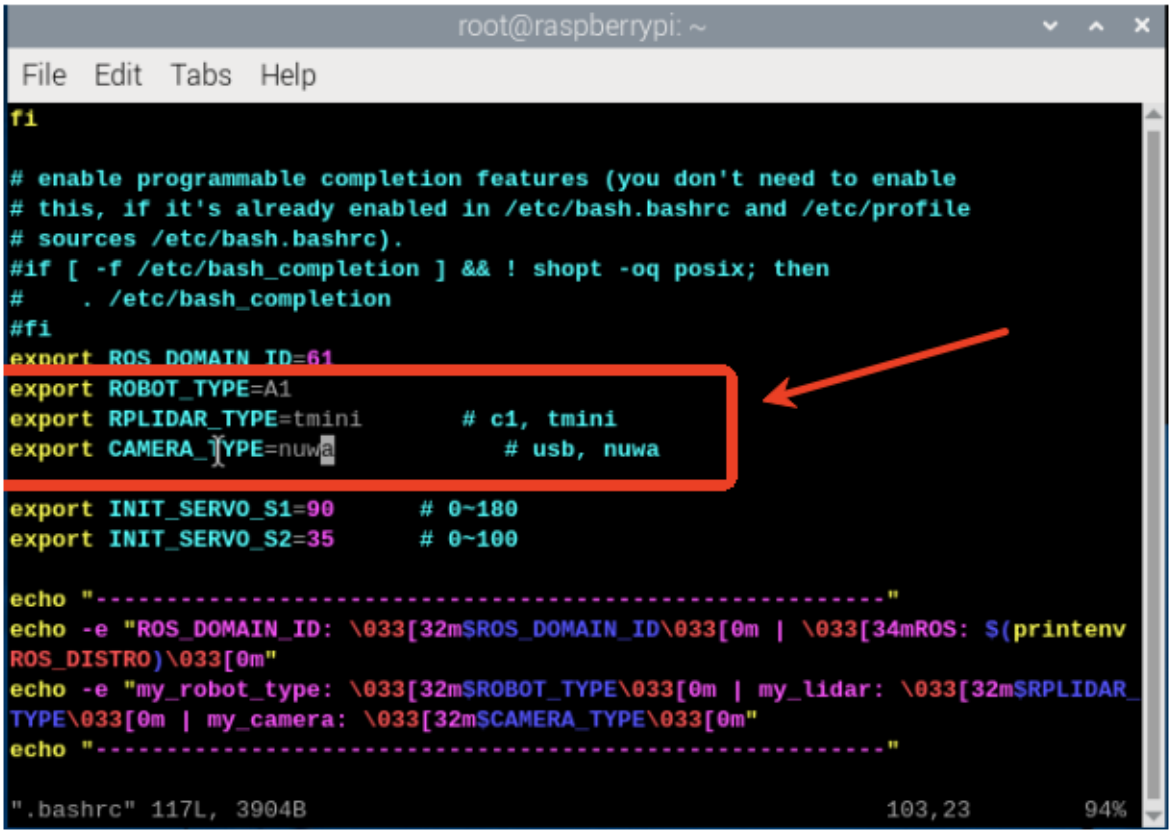
For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

Change the following settings based on the car model, radar type, and camera type.

```
root@ubuntu:/# cd
root@ubuntu:~# vim .bashrc
```

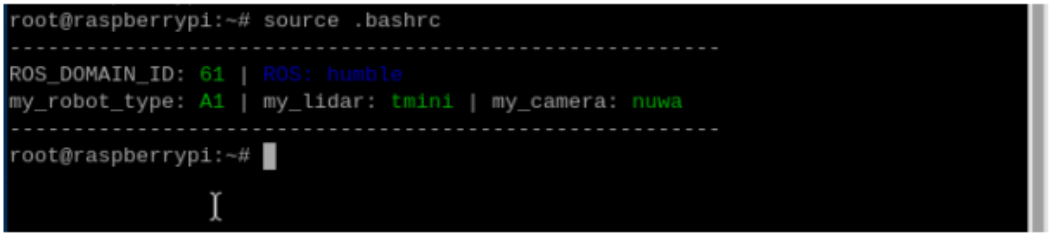


Find this location and press i on your keyboard to change the settings to the corresponding camera and radar models. The default settings are tmini and nuwa.



After completing the modification, save and exit vim, then execute:

```
root@raspberrypi:~# source .bashrc
------------------------------------------------------
ROS_DOMAIN_ID: 61 | ROS: humble
my_robot_type: M1 | my_lidar: tmini | my_camera: nuwa
------------------------------------------------------
root@raspberrypi:~#
```



4. Program Startup

4.1. Startup Command

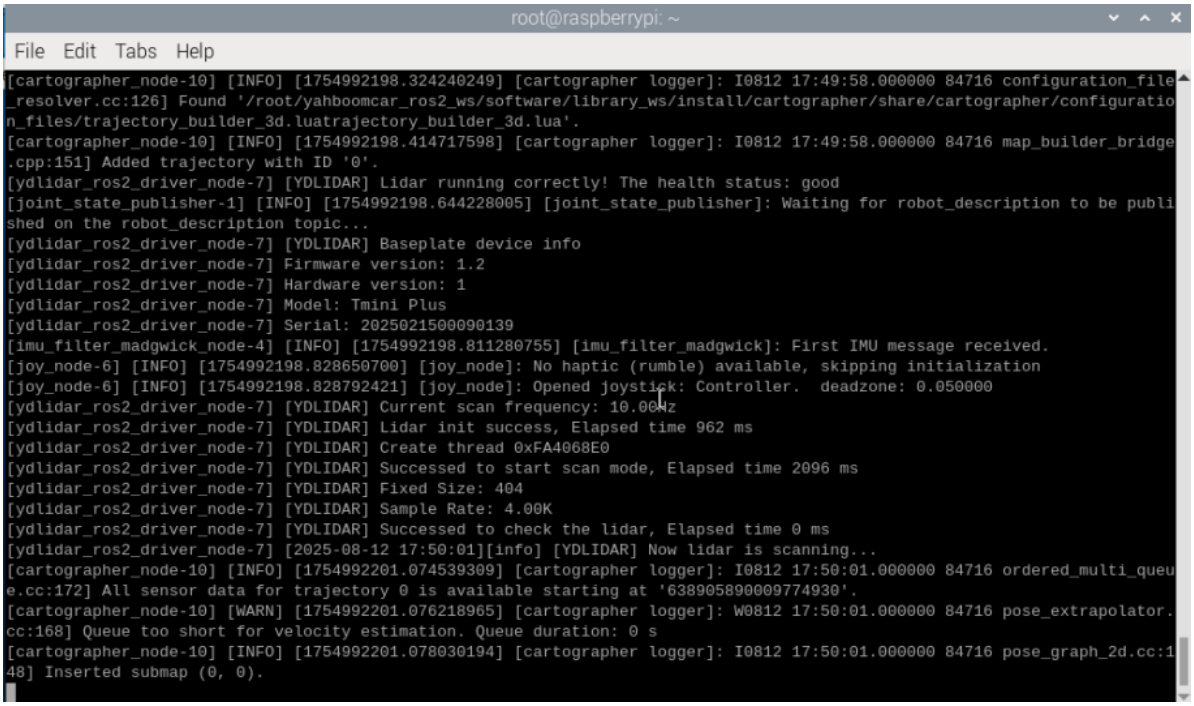
For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

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

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

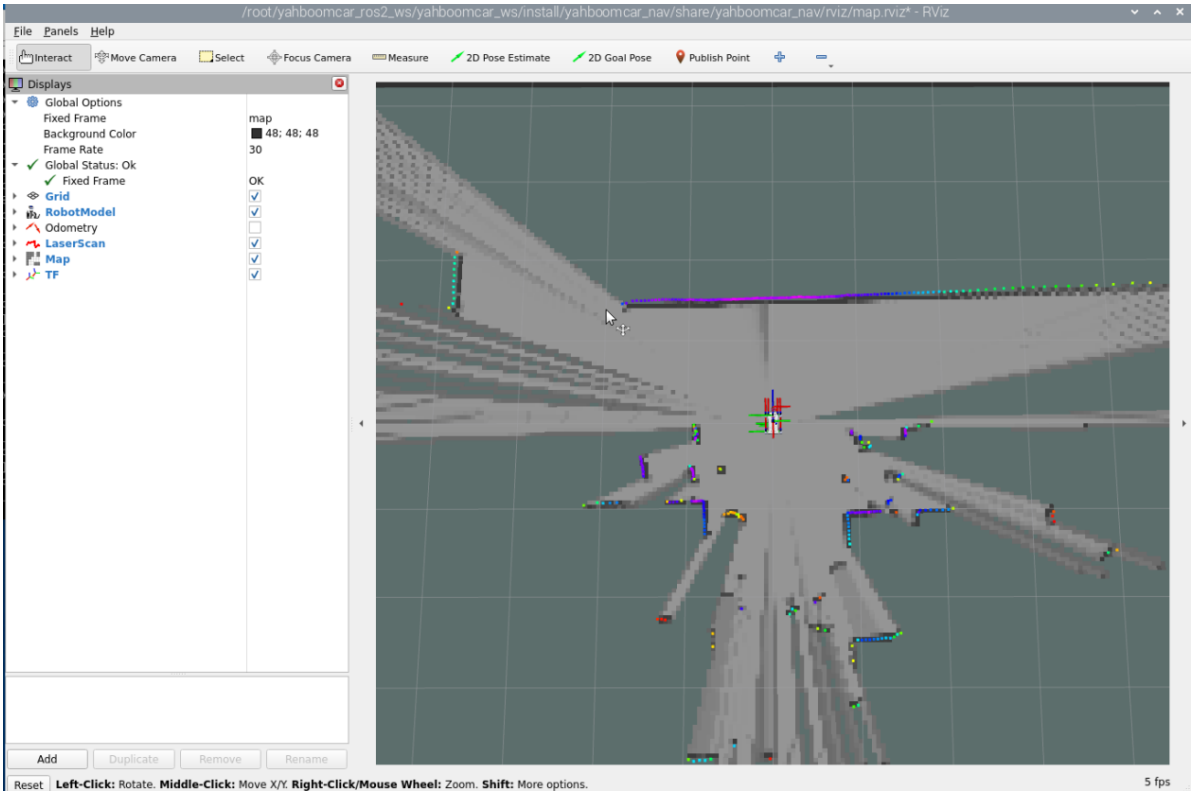
Start Mapping

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



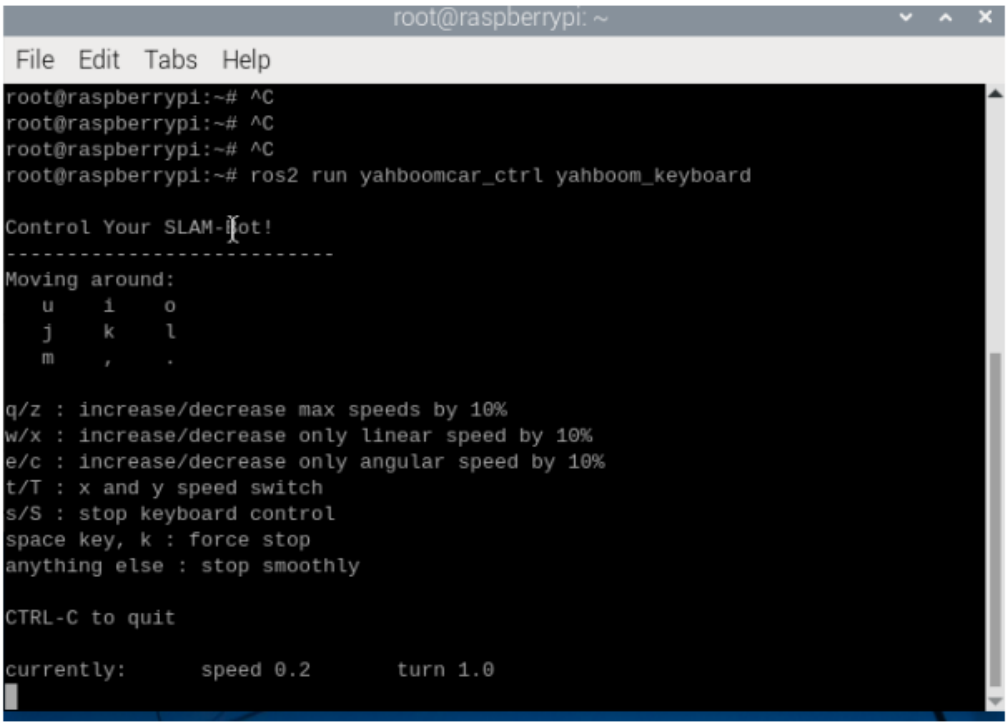
Enter the command to start rviz visualization mapping.

```
ros2 launch yahboomcar_nav display_map_launch.py
```

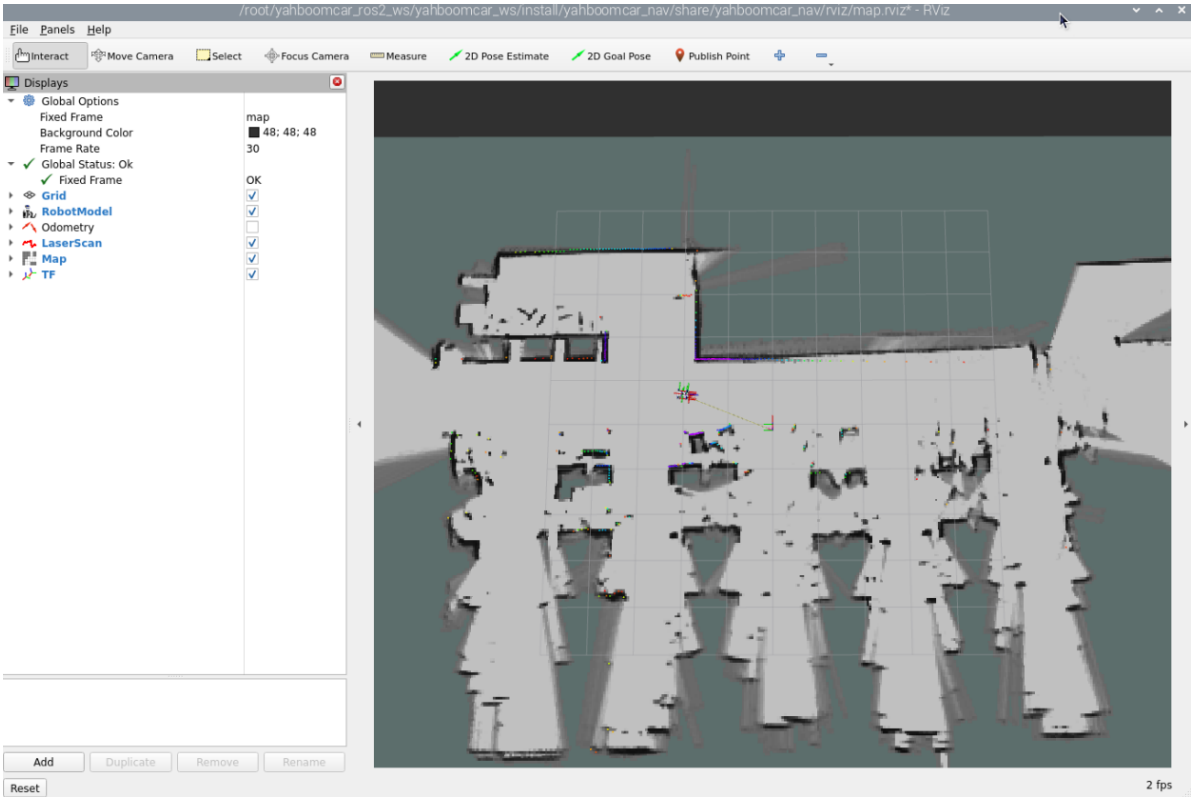


The program has controller control enabled by default. If you're using a controller, you can now connect it directly to the receiver for control. If you prefer keyboard control, enter this in the terminal:

```
#keyboard
ros2 run yahboomcar_ctrl yahboom_keyboard
```

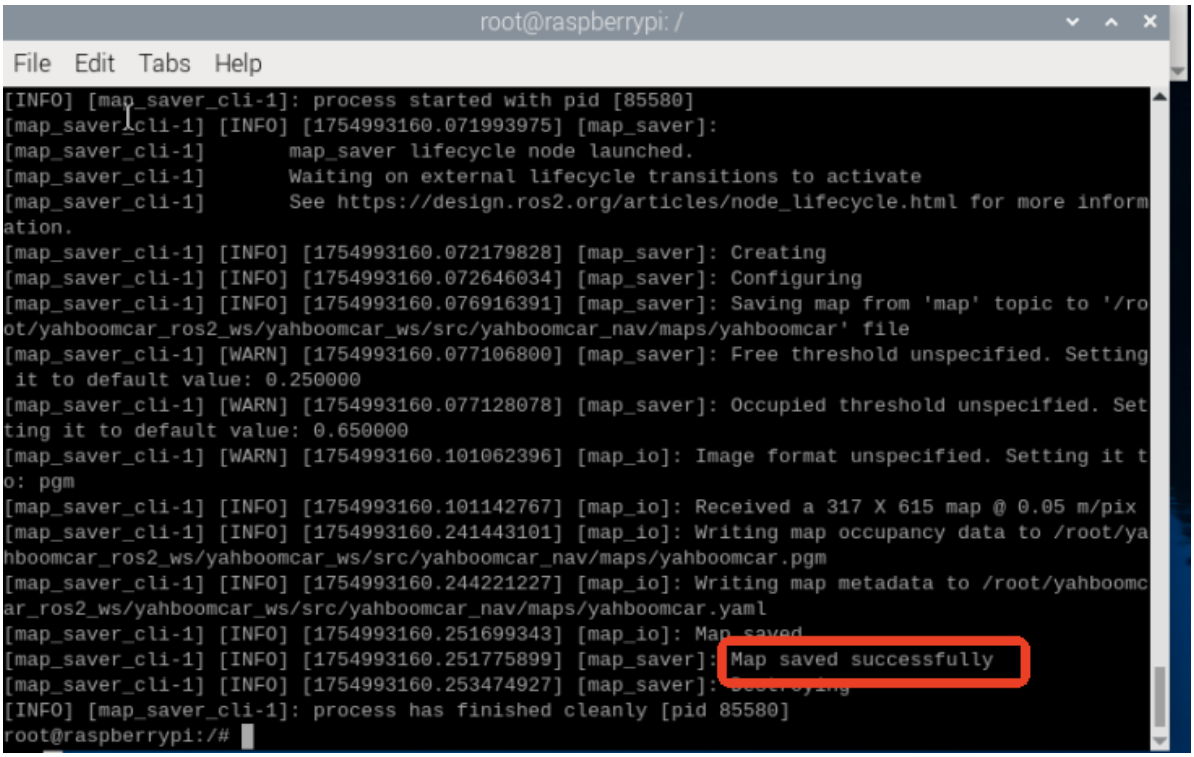


Then control the car to slowly navigate the area to be mapped.



After mapping, enter the following command to save the map. In the terminal, enter:

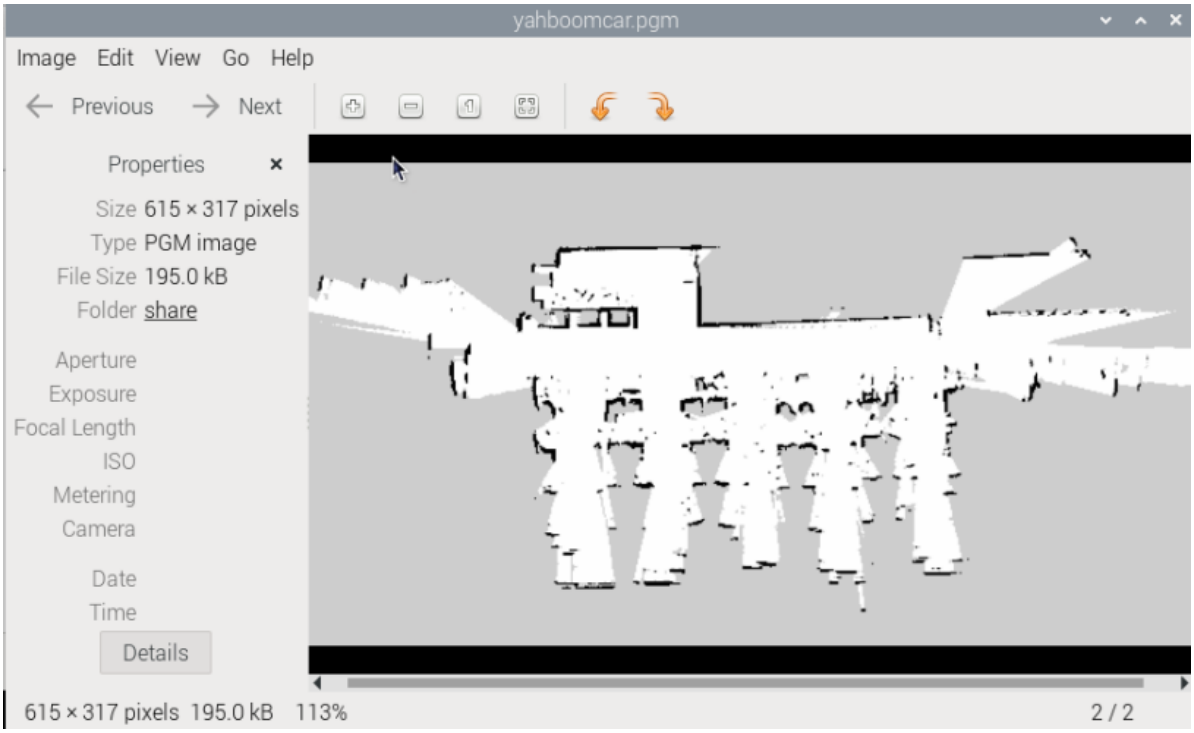
```
ros2 launch yahboomcar_nav save_map_launch.py
```



A map named yahboomcar will be saved. This map is saved in:

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

Two files will be generated, one named yahboomcar.pgm. The map is as follows. (For RDK X5 and Orin , you can directly double-click to view the map. For Raspberry Pi and Jetson nano, you need to copy this file to the /root/share shared folder on Docker and then view it on the host machine.)



yahboom_map.yaml, take a look at the yaml contents.

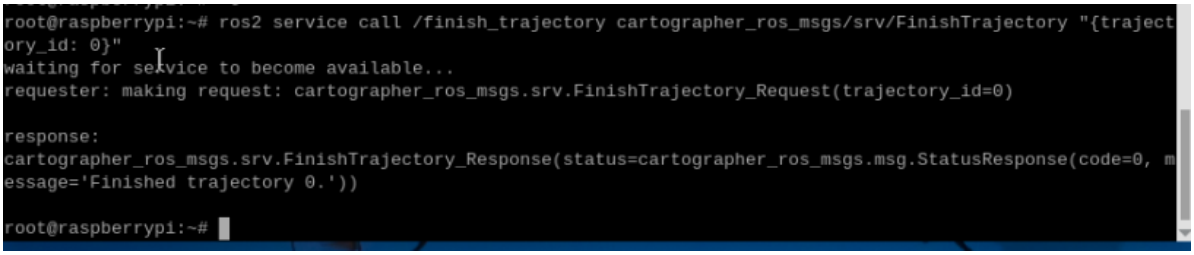
```
image: yahboomcar.pgm
mode: trinary
resolution: 0.05
origin: [-9.02, -15.5, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

- image: The image representing the map, i.e., yahboomcar.pgm
- mode: This property can be one of trinary, scale, or raw, depending on the selected mode. Trinary is the default mode.
- resolution: The resolution of the map, in meters/pixels.
- origin: The 2D pose (x, y, yaw) of the lower left corner of the map. The yaw value here is counterclockwise (yaw=0 means no rotation). Currently, many parts of the system ignore the yaw value.
- negate: Whether to invert the meaning of white/black and free/occupied (this does not affect the interpretation of thresholds).
- occupied_thresh: Pixels with an occupied probability greater than this threshold are considered fully occupied.
- free_thresh: Pixels with an occupied probability less than this threshold are considered completely free.

4.2. Saving the Rapid Relocation Map

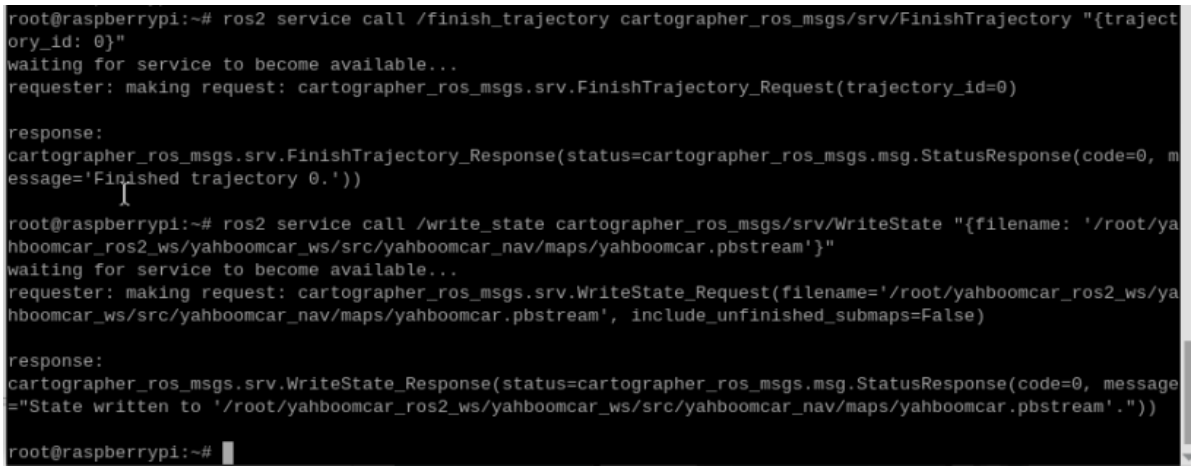
Then enter the following command to stop building the map.

```
ros2 service call /finish_trajectory cartographer_ros_msgs/srv/FinishTrajectory
"{trajectory_id: 0}"
```



Then enter the following command to save the pbstream file.

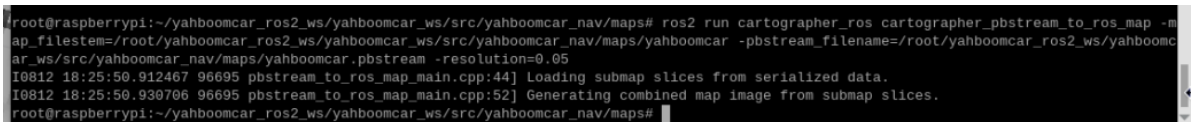
```
#Raspberry Pi, Jetson Nano
ros2 service call /write_state cartographer_ros_msgs/srv/WriteState "{filename:
'/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pbstr
eam'}"
#ORIN
ros2 service call /write_state cartographer_ros_msgs/srv/WriteState "{filename:
'/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomca
r.pbstream'}"
#RDK X5
ros2 service call /write_state cartographer_ros_msgs/srv/WriteState "{filename:
'/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomc
ar.pbstream'}"
```



The path of the filename parameter is the path where the map's pbstream file is saved.

Finally, enter the following command to convert the pbstream file to a pgm file.

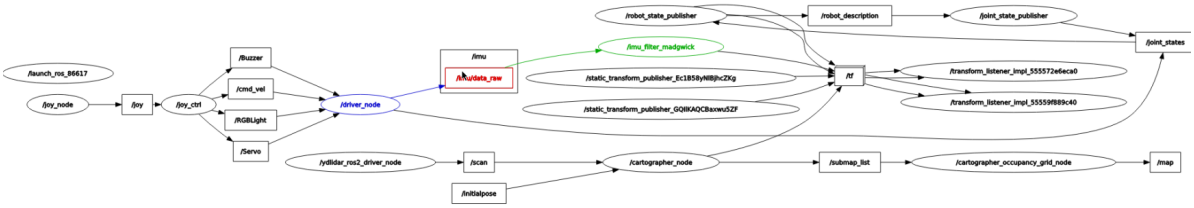
```
#Raspberry Pi, jetson nano
ros2 run cartographer_ros cartographer_pbstream_to_ros_map -
map_filestem=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahb
oomcar -
pbstream_filename=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps
/yahboomcar.pbstream -resolution=0.05
#ORIN
ros2 run cartographer_ros cartographer_pbstream_to_ros_map -
map_filestem=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/ma
ps/yahboomcar -
pbstream_filename=/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_n
av/maps/yahboomcar.pbstream -resolution=0.05
#RDK X5
ros2 run cartographer_ros cartographer_pbstream_to_ros_map -
map_filestem=/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/m
aps/yahboomcar -
pbstream_filename=/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_
nav/maps/yahboomcar.pbstream -resolution=0.05
```



5. View the Node Communication Graph

In the terminal, enter:

```
ros2 run rqt_graph rqt_graph
```



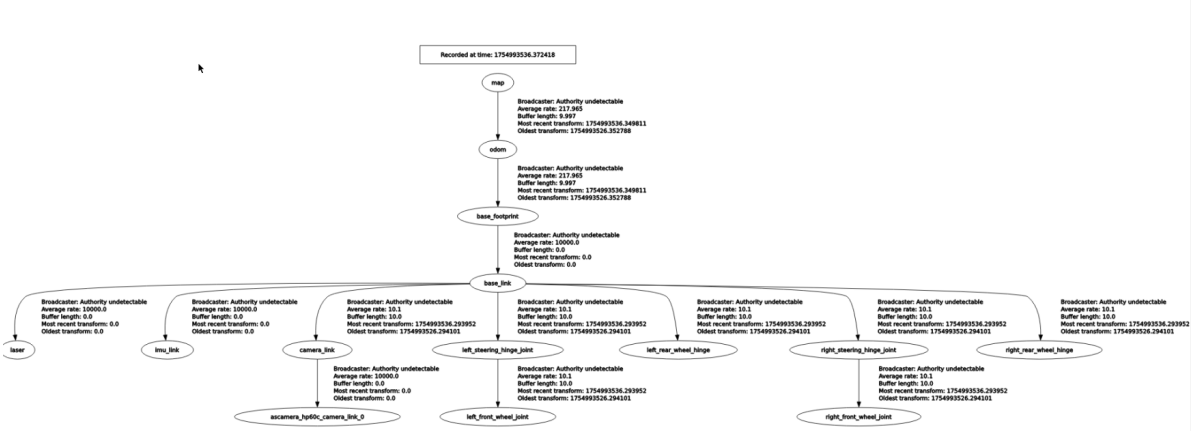
If the graph does not display initially, select [Nodes/Topics (all)] and click the refresh button in the upper left corner.

6. View the TF tree

Enter in the terminal:

```
ros2 run rqt_tf_tree rqt_tf_tree
```

After the program finishes running, a TF conversion interface will appear.



7. Code Analysis

This section only explains the map_cartographer_launch.py file for map creation. The file path is:

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/launch/map_cartographer_launch.py
```

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():
```

```
    package_launch_path
=os.path.join(get_package_share_directory('yahboomcar_nav'), 'launch')

    laser_bringup_launch =
IncludeLaunchDescription(PythonLaunchDescriptionSource(
        [package_launch_path,'/laser_bringup_no_odom_launch.py'])
    )

    cartographer_launch =
IncludeLaunchDescription(PythonLaunchDescriptionSource(
        [package_launch_path, '/cartographer_launch.py'])
    )

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

This runs the launch file - cartographer_launch, which is located at

```
~/yahboomcar_ros2_ws/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]),

        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(),
        ),
    ])
```

This mainly involves running the cartographer_node mapping node and occupancy_grid_launch.py, and also loading the parameter configuration file, which is located at

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/params/lds_2d.lua
```

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 = "base_footprint",
  odom_frame = "odom",
  provide_odom_frame = true,
  publish_frame_projected_to_2d = false,
  use_odometry = false,
  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.min_range = 0.1
TRAJECTORY_BUILDER_2D.max_range = 10.0
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 0.5
TRAJECTORY_BUILDER_2D.use_imu_data = false
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 = 0.7
POSE_GRAPH.constraint_builder.global_localization_min_score = 0.7

--POSE_GRAPH.optimize_every_n_nodes = 30

return options
```