Cartographer-SLAM mapping

This lesson uses the Raspberry Pi 5 as an example.

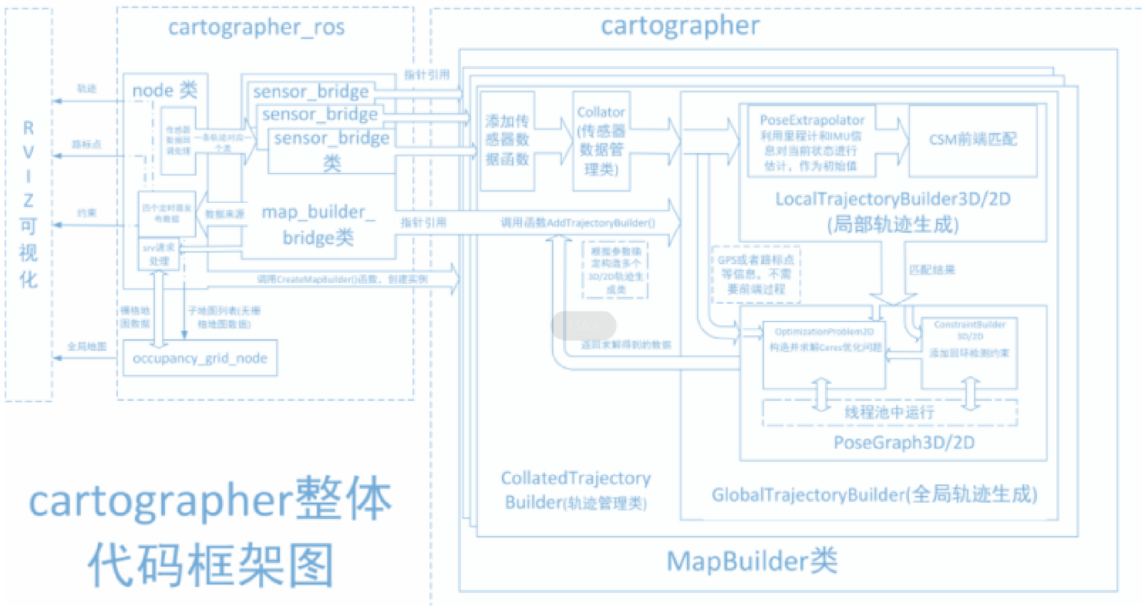
For Raspberry Pi and Jetson Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host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

For 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_ros
cartographer
RVIZ可视化
node 类
sensor_bridge
sensor_bridge
sensor_bridge类
map_builder_bridge类
指针引用
轨迹
路标点
约束
全局地图
数据来源
srv请求处理
栅格地图数据
子地图列表(无栅格地图数据)
occupancy_grid_node
调用CreateMapBuilder()函数，创建实例
调用函数AddTrajectoryBuilder()
添加传感器数据函数
Collator(传感器数据管理类)
PoseExtrapolator 利用里程计和IMU信息对当前状态进行估计，作为初始值
CSM前端匹配
LocalTrajectoryBuilder3D/2D (局部轨迹生成)
根据参数确定构造多个3D/2D轨迹生成类
GPS或者路标点等信息，不需要前端过程
匹配结果
返回求解得到的数据
OptimizationProblem2D 构造并求解Ceres优化问题
ConstraintBuilder 3D/2D 添加回环检测约束
线程池中运行
PoseGraph3D/2D
CollatedTrajectory Builder(轨迹管理类)
GlobalTrajectoryBuilder(全局轨迹生成)
MapBuilder类
cartographer整体代码框架图

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. **(For Ackermann-type cars, for optimal map creation, try adding some stops when turning, rather than making a sharp turn.)**

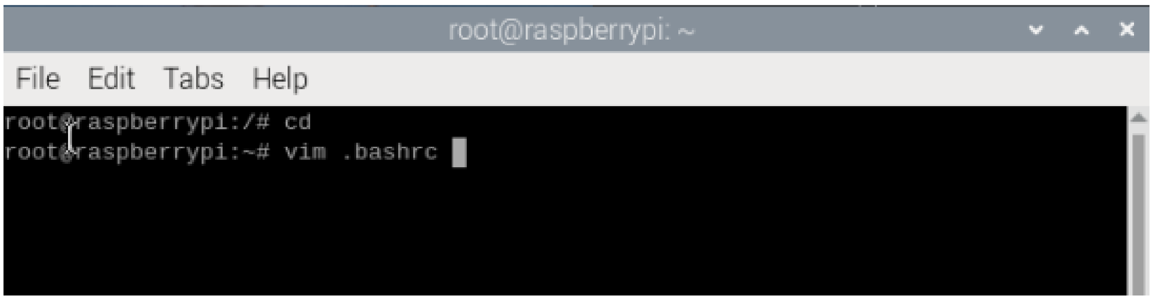
3. Pre-Use Configuration

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 the Raspberry Pi 5 controller, you need to first enter the Docker container. The Orin board does not require this.

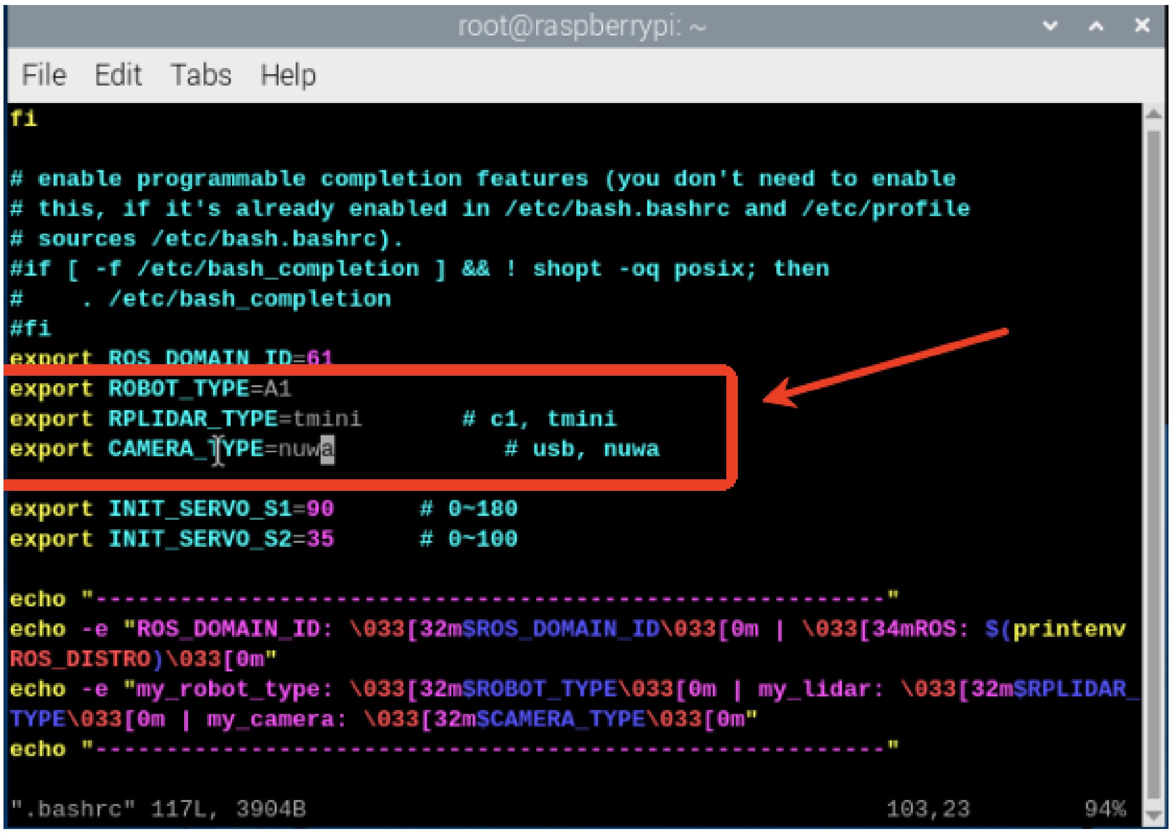
Change the following settings based on the car model, radar type, and camera type.

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



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

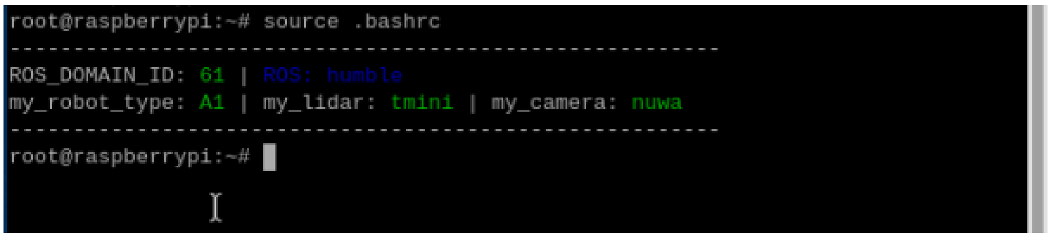
Find this location and press i on your keyboard to change the settings to the corresponding camera and radar models. The default settings are tmini and nuwa.



```
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#    . /etc/bash_completion
#fi
export ROS_DOMAIN_ID=61
export ROBOT_TYPE=A1
export RPLIDAR_TYPE=tmini        # c1, tmini
export CAMERA_TYPE=nuwa             # usb, nuwa

export INIT_SERVO_S1=90      # 0~180
export INIT_SERVO_S2=35      # 0~100

echo "--------------------------------------------------------------"
echo -e "ROS_DOMAIN_ID: \033[32m$ROS_DOMAIN_ID\033[0m | \033[34mROS: $(printenv ROS_DISTRO)\033[0m"
echo -e "my_robot_type: \033[32m$ROBOT_TYPE\033[0m | my_lidar: \033[32m$RPLIDAR_TYPE\033[0m | my_camera: \033[32m$CAMERA_TYPE\033[0m"
echo "--------------------------------------------------------------"

".bashrc" 117L, 3904B                                          103,23        94%
```

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

```
root@raspberrypi:~# source .bashrc
-------------------------------------------------------
ROS_DOMAIN_ID: 61 | ROS: humble
my_robot_type: A1 | my_lidar: tmini | my_camera: nuwa
-------------------------------------------------------
root@raspberrypi:~#
```



```
root@raspberrypi:~# source .bashrc
-------------------------------------------------------
ROS_DOMAIN_ID: 61 | ROS: humble
my_robot_type: A1 | my_lidar: tmini | my_camera: nuwa
-------------------------------------------------------
root@raspberrypi:~#
```

4. Program Startup

4.1. Startup Command

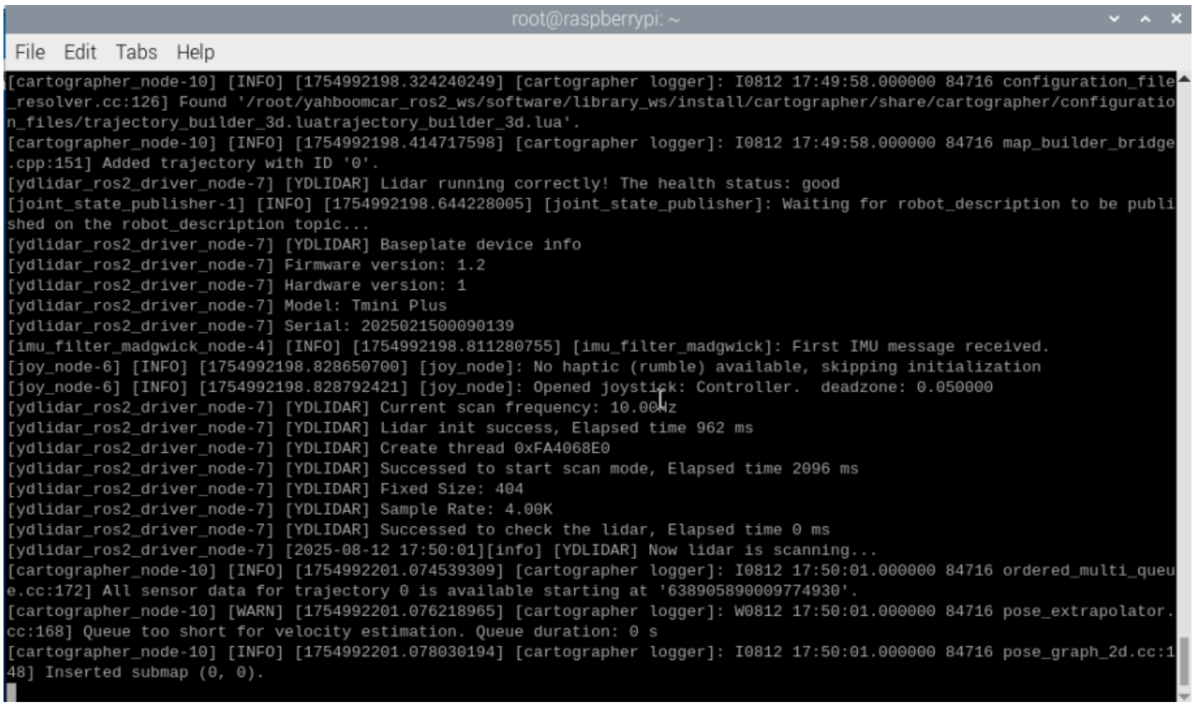
For the Raspberry Pi 5 controller, you must first enter the Docker container. For the Orin controller, this is not necessary.

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

All the following commands must be executed from the Docker terminal within the same Docker container.**(For steps, see [Docker Course] --- [3. Docker Submission and Multi-Terminal Access]).**

Start Mapping

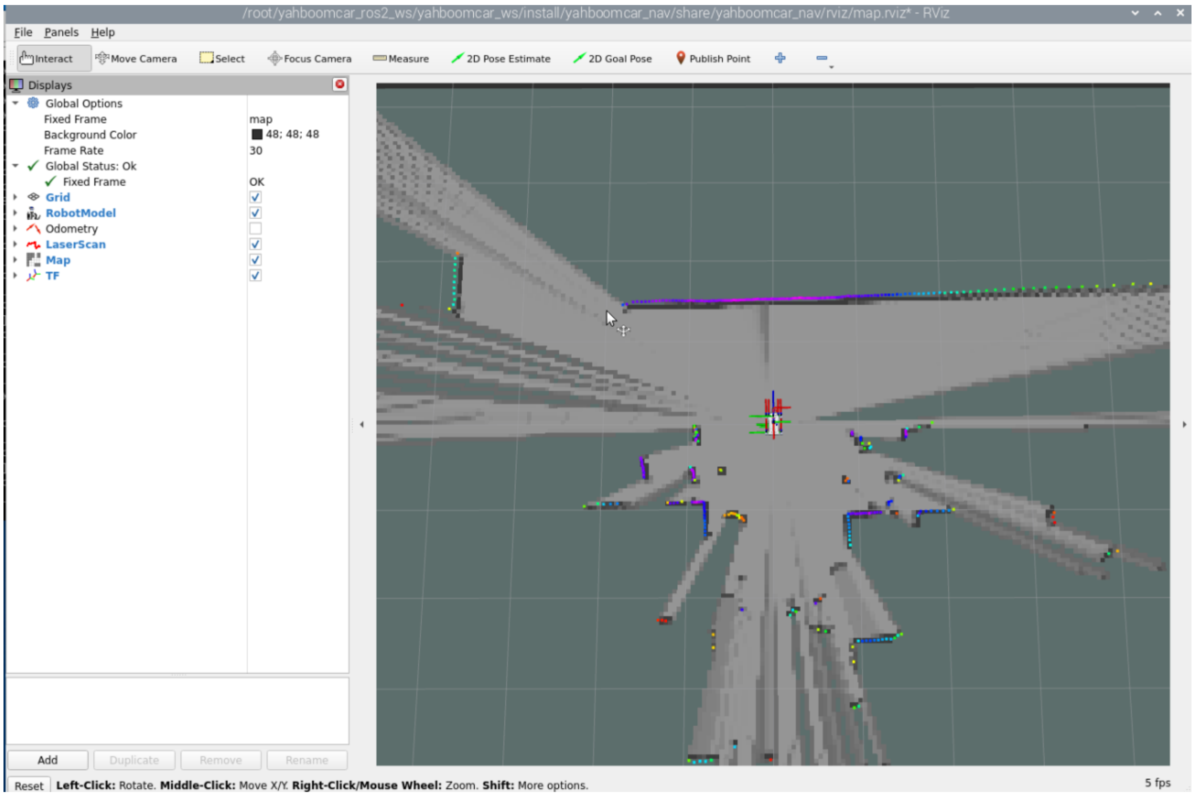
```
ros2 launch yahboomcar_nav map_cartographer_launch.py
```



```
[cartographer_node-10] [INFO] [1754992198.324240249] [cartographer logger]: I0812 17:49:58.000000 84716 configuration_file
_resolver.cc:126] Found '/root/yahboomcar_ros2_ws/software/library_ws/install/cartographer/share/cartographer/configuratio
n_files/trajectory_builder_3d.luatrajectory_builder_3d.lua'.
[cartographer_node-10] [INFO] [1754992198.414717598] [cartographer logger]: I0812 17:49:58.000000 84716 map_builder_bridge
.cpp:151] Added trajectory with ID '0'.
[ydlidar_ros2_driver_node-7] [YDLIDAR] Lidar running correctly! The health status: good
[joint_state_publisher-1] [INFO] [1754992198.644228005] [joint_state_publisher]: Waiting for robot_description to be publi
shed on the robot_description topic...
[ydlidar_ros2_driver_node-7] [YDLIDAR] Baseplate device info
[ydlidar_ros2_driver_node-7] [YDLIDAR] Firmware version: 1.2
[ydlidar_ros2_driver_node-7] [YDLIDAR] Hardware version: 1
[ydlidar_ros2_driver_node-7] [YDLIDAR] Model: Tmini Plus
[ydlidar_ros2_driver_node-7] [YDLIDAR] Serial: 2025021500090139
[imu_filter_madgwick_node-4] [INFO] [1754992198.811280755] [imu_filter_madgwick]: First IMU message received.
[joy_node-6] [INFO] [1754992198.828650700] [joy_node]: No haptic (rumble) available, skipping initialization
[joy_node-6] [INFO] [1754992198.828792421] [joy_node]: Opened joystick: Controller.  deadzone: 0.050000
[ydlidar_ros2_driver_node-7] [YDLIDAR] Current scan frequency: 10.00Hz
[ydlidar_ros2_driver_node-7] [YDLIDAR] Lidar init success, Elapsed time 962 ms
[ydlidar_ros2_driver_node-7] [YDLIDAR] Create thread 0xFA4068E0
[ydlidar_ros2_driver_node-7] [YDLIDAR] Successed to start scan mode, Elapsed time 2096 ms
[ydlidar_ros2_driver_node-7] [YDLIDAR] Fixed Size: 404
[ydlidar_ros2_driver_node-7] [YDLIDAR] Sample Rate: 4.00K
[ydlidar_ros2_driver_node-7] [YDLIDAR] Successed to check the lidar, Elapsed time 0 ms
[ydlidar_ros2_driver_node-7] [2025-08-12 17:50:01][info] [YDLIDAR] Now lidar is scanning...
[cartographer_node-10] [INFO] [1754992201.074539309] [cartographer logger]: I0812 17:50:01.000000 84716 ordered_multi_queu
e.cc:172] All sensor data for trajectory 0 is available starting at '638905890009774930'.
[cartographer_node-10] [WARN] [1754992201.076218965] [cartographer logger]: W0812 17:50:01.000000 84716 pose_extrapolator.
cc:168] Queue too short for velocity estimation. Queue duration: 0 s
[cartographer_node-10] [INFO] [1754992201.078030194] [cartographer logger]: I0812 17:50:01.000000 84716 pose_graph_2d.cc:1
48] Inserted submap (0, 0).
```

Enter the command to start rviz visualization mapping.

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

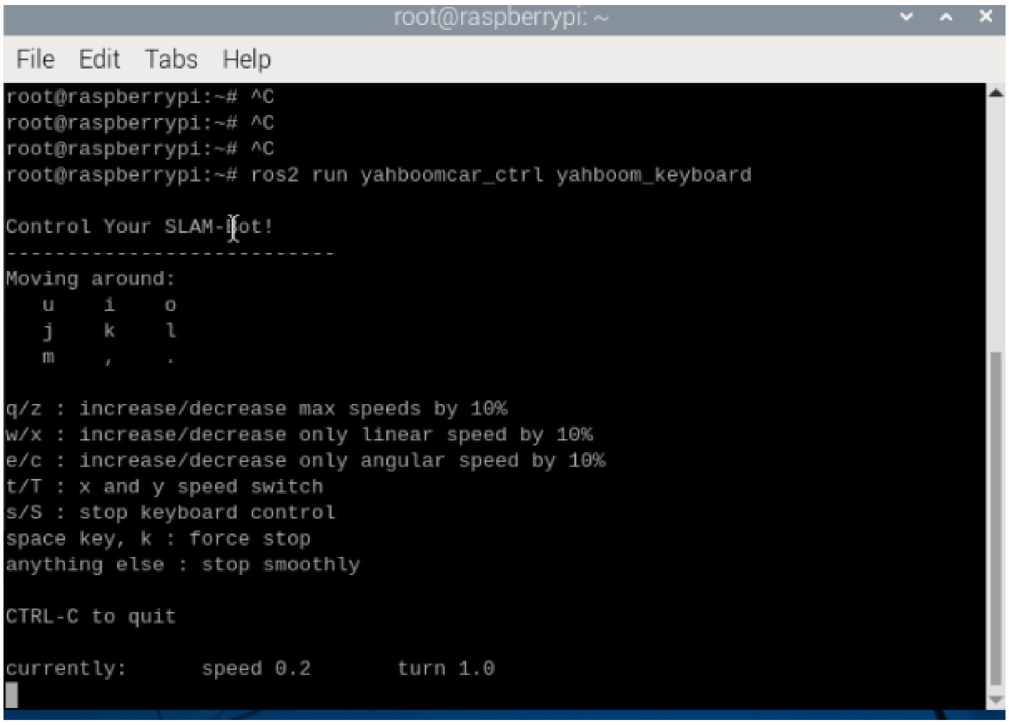


```
Global Options
  Fixed Frame          map
  Background Color     48; 48; 48
  Frame Rate           30
Global Status: Ok
  Fixed Frame          OK
Grid                   ✓
RobotModel             ✓
Odometry               
LaserScan              ✓
Map                    ✓
TF                     ✓
```

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options.

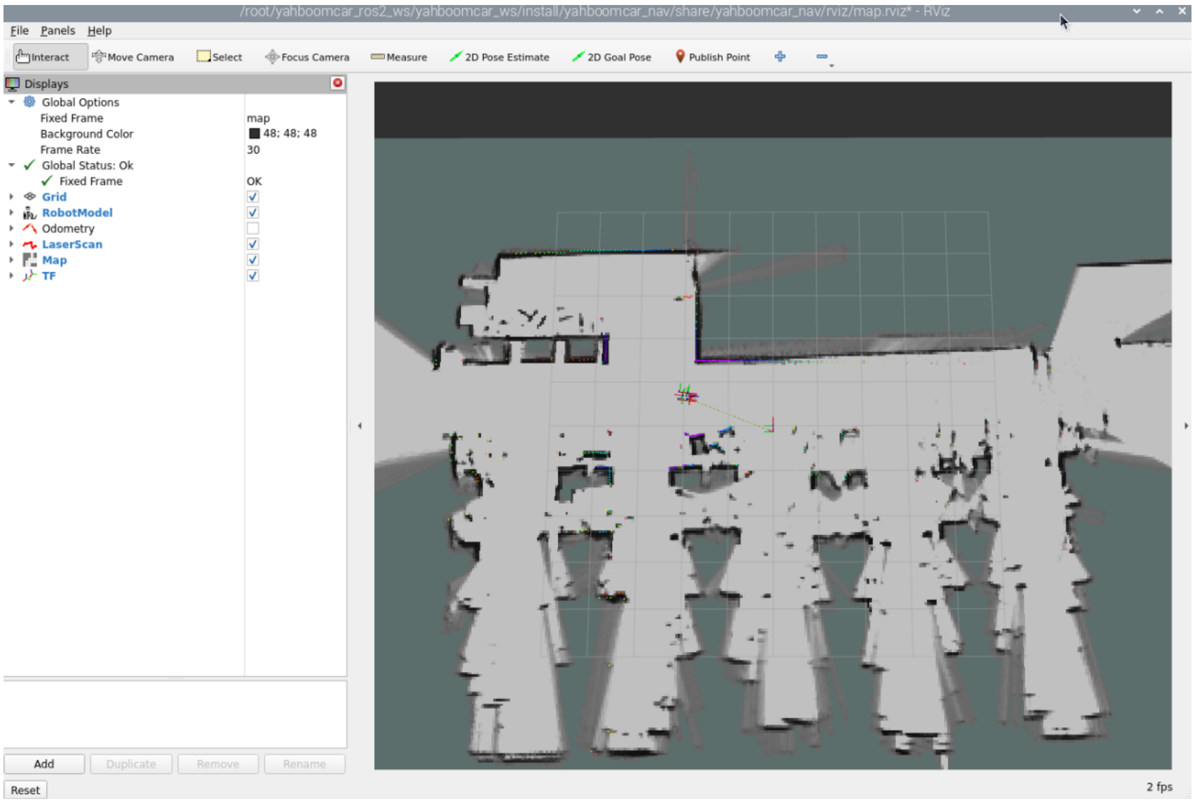
The program has controller control enabled by default. If you're using a controller, you can now connect it directly to the receiver for control. If you prefer keyboard control, enter this in the terminal:

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



```
root@raspberrypi:~# ^C
root@raspberrypi:~# ^C
root@raspberrypi:~# ^C
root@raspberrypi:~# ros2 run yahboomcar_ctrl yahboom_keyboard

Control Your SLAM-Bot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
t/T : x and y speed switch
s/S : stop keyboard control
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2       turn 1.0
```

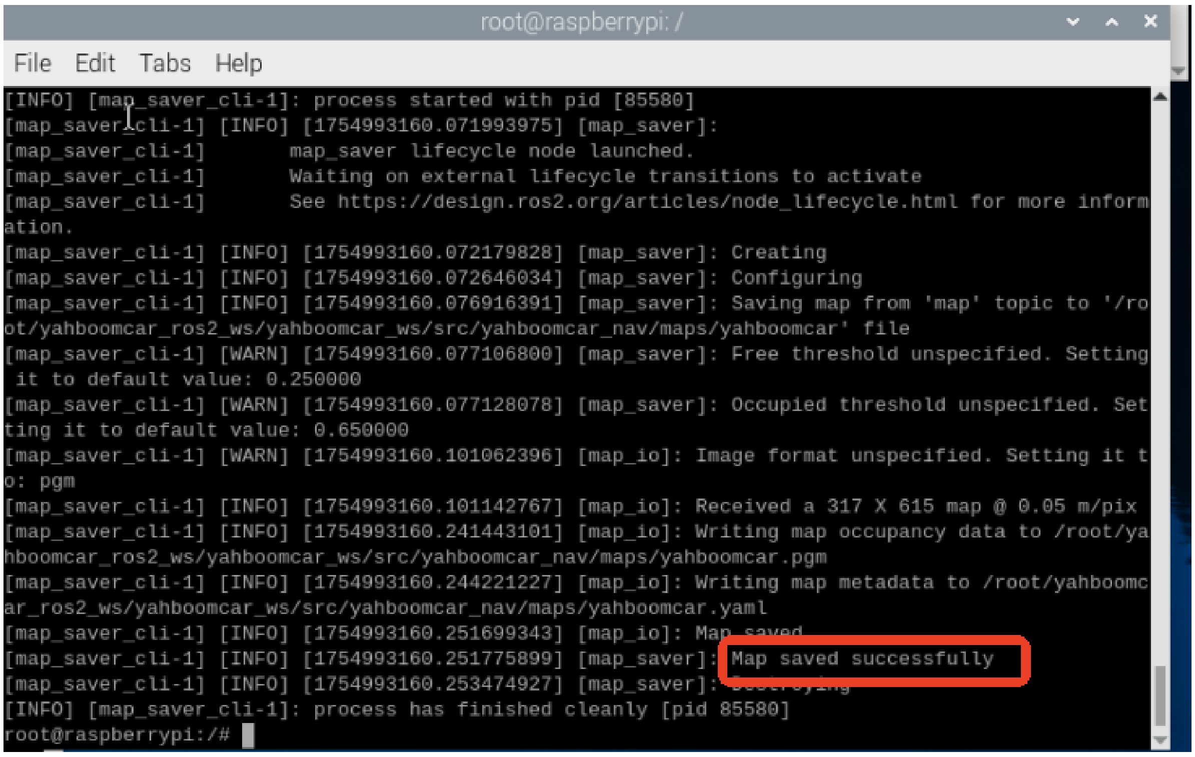
Then control the car to slowly navigate the area to be mapped.



```
Displays
  Global Options
    Fixed Frame          map
    Background Color     48; 48; 48
    Frame Rate           30
  ✓ Global Status: Ok
    ✓ Fixed Frame        OK
  Grid                   ✓
  RobotModel             ✓
  Odometry
  LaserScan              ✓
  Map                    ✓
  TF                     ✓
```

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

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

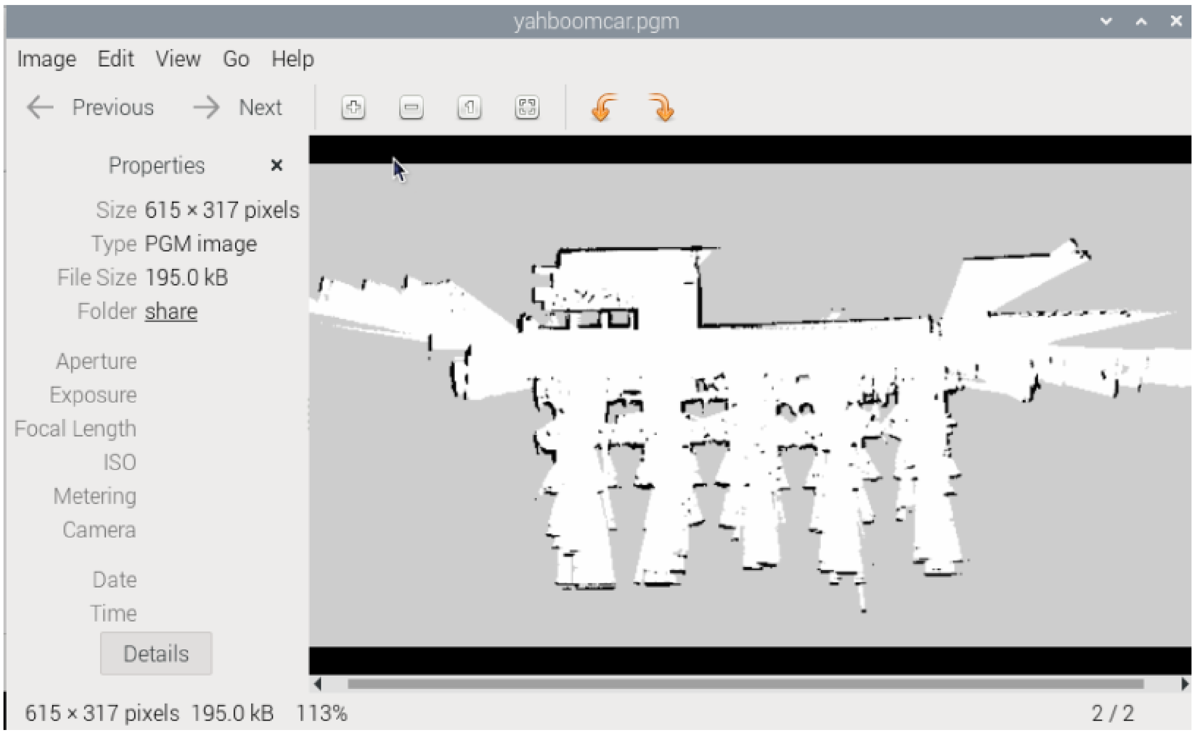


```
[INFO] [map_saver_cli-1]: process started with pid [85580]
[map_saver_cli-1] [INFO] [1754993160.071993975] [map_saver]:
[map_saver_cli-1]       map_saver lifecycle node launched.
[map_saver_cli-1]       Waiting on external lifecycle transitions to activate
[map_saver_cli-1]       See https://design.ros2.org/articles/node_lifecycle.html for more inform
ation.
[map_saver_cli-1] [INFO] [1754993160.072179828] [map_saver]: Creating
[map_saver_cli-1] [INFO] [1754993160.072646034] [map_saver]: Configuring
[map_saver_cli-1] [INFO] [1754993160.076916391] [map_saver]: Saving map from 'map' topic to '/ro
ot/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar' file
[map_saver_cli-1] [WARN] [1754993160.077106800] [map_saver]: Free threshold unspecified. Setting
 it to default value: 0.250000
[map_saver_cli-1] [WARN] [1754993160.077128078] [map_saver]: Occupied threshold unspecified. Set
ting it to default value: 0.650000
[map_saver_cli-1] [WARN] [1754993160.101062396] [map_io]: Image format unspecified. Setting it t
o: pgm
[map_saver_cli-1] [INFO] [1754993160.101142767] [map_io]: Received a 317 X 615 map @ 0.05 m/pix
[map_saver_cli-1] [INFO] [1754993160.241443101] [map_io]: Writing map occupancy data to /root/ya
hboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pgm
[map_saver_cli-1] [INFO] [1754993160.244221227] [map_io]: Writing map metadata to /root/yahboomc
ar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.yaml
[map_saver_cli-1] [INFO] [1754993160.251699343] [map_io]: Map saved
[map_saver_cli-1] [INFO] [1754993160.251775899] [map_saver]: Map saved successfully
[map_saver_cli-1] [INFO] [1754993160.253474927] [map_saver]: Destroying
[INFO] [map_saver_cli-1]: process has finished cleanly [pid 85580]
root@raspberrypi:/#
```

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. (On an Orin board, you can directly double-click the map. On a Raspberry Pi board, copy the file to the shared folder /root/share in Docker, then view it on the host machine.)



Size 615 × 317 pixels
Type PGM image
File Size 195.0 kB
Folder share

615 × 317 pixels 195.0 kB 113% 2 / 2

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

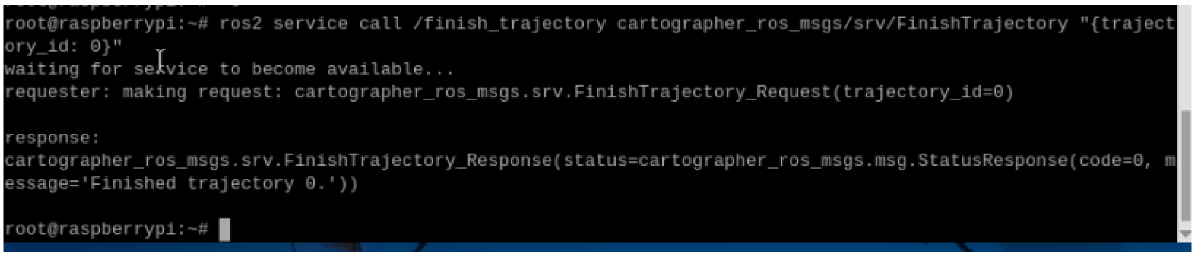
```
image: yahboomcar.pgm
mode: trinary
resolution: 0.05
origin: [-9.02, -15.5, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

- 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.

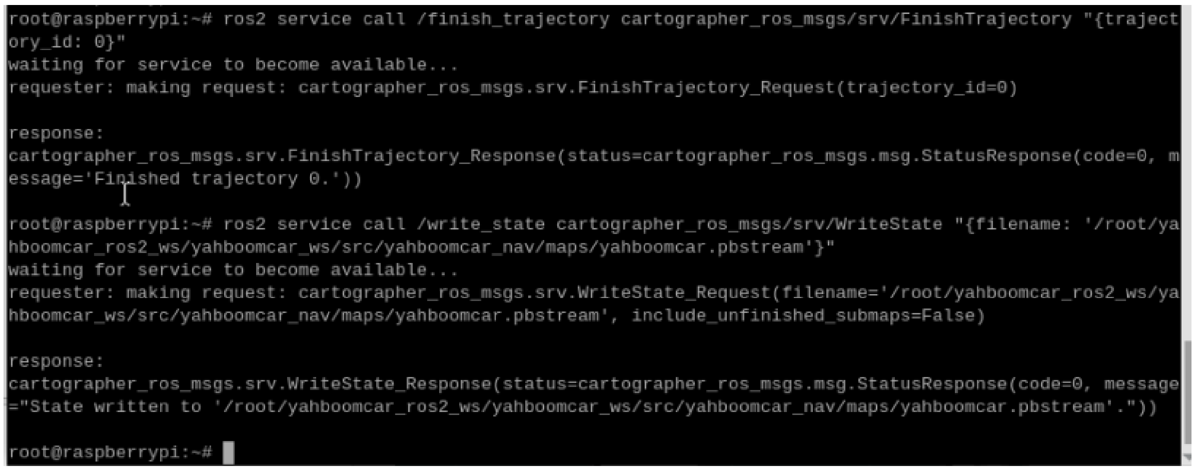
```
ros2 service call /finish_trajectory cartographer_ros_msgs/srv/FinishTrajectory
"{trajectory_id: 0}"
```



```
root@raspberrypi:~# ros2 service call /finish_trajectory cartographer_ros_msgs/srv/FinishTrajectory "{traject
ory_id: 0}"
waiting for service to become available...
requester: making request: cartographer_ros_msgs.srv.FinishTrajectory_Request(trajectory_id=0)

response:
cartographer_ros_msgs.srv.FinishTrajectory_Response(status=cartographer_ros_msgs.msg.StatusResponse(code=0, m
essage='Finished trajectory 0.'))

root@raspberrypi:~#
```

Then enter the following command to save the pbstream file.

```
#Raspberry Pi, Jetson Nano controller
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 master control
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'}"
```

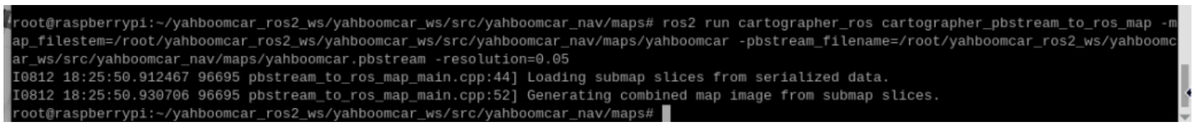


```
root@raspberrypi:~# ros2 service call /finish_trajectory cartographer_ros_msgs/srv/FinishTrajectory "{traject
ory_id: 0}"
waiting for service to become available...
requester: making request: cartographer_ros_msgs.srv.FinishTrajectory_Request(trajectory_id=0)

response:
cartographer_ros_msgs.srv.FinishTrajectory_Response(status=cartographer_ros_msgs.msg.StatusResponse(code=0, m
essage='Finished trajectory 0.'))

root@raspberrypi:~# ros2 service call /write_state cartographer_ros_msgs/srv/WriteState "{filename: '/root/ya
hboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pbstream'}"
waiting for service to become available...
requester: making request: cartographer_ros_msgs.srv.WriteState_Request(filename='/root/yahboomcar_ros2_ws/ya
hboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pbstream', include_unfinished_submaps=False)

response:
cartographer_ros_msgs.srv.WriteState_Response(status=cartographer_ros_msgs.msg.StatusResponse(code=0, message
="State written to '/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.pbstream'."))

root@raspberrypi:~# 
```

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 master control
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
#ORINmaster
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
```

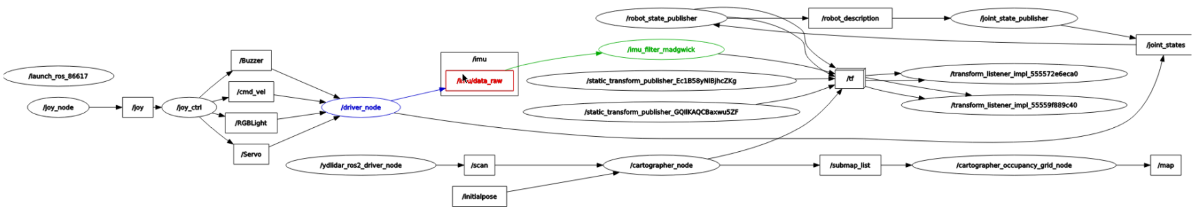


```
root@raspberrypi:~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps# ros2 run cartographer_ros cartographer_pbstream_to_ros_map -m
ap_filestem=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar -pbstream_filename=/root/yahboomcar_ros2_ws/yahboomc
ar_ws/src/yahboomcar_nav/maps/yahboomcar.pbstream -resolution=0.05
I0812 18:25:50.912467 96695 pbstream_to_ros_map_main.cpp:44] Loading submap slices from serialized data.
I0812 18:25:50.930706 96695 pbstream_to_ros_map_main.cpp:52] Generating combined map image from submap slices.
root@raspberrypi:~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps# 
```

5. View the Node Communication Graph

In the terminal, enter:

```
ros2 run rqt_graph rqt_graph
```



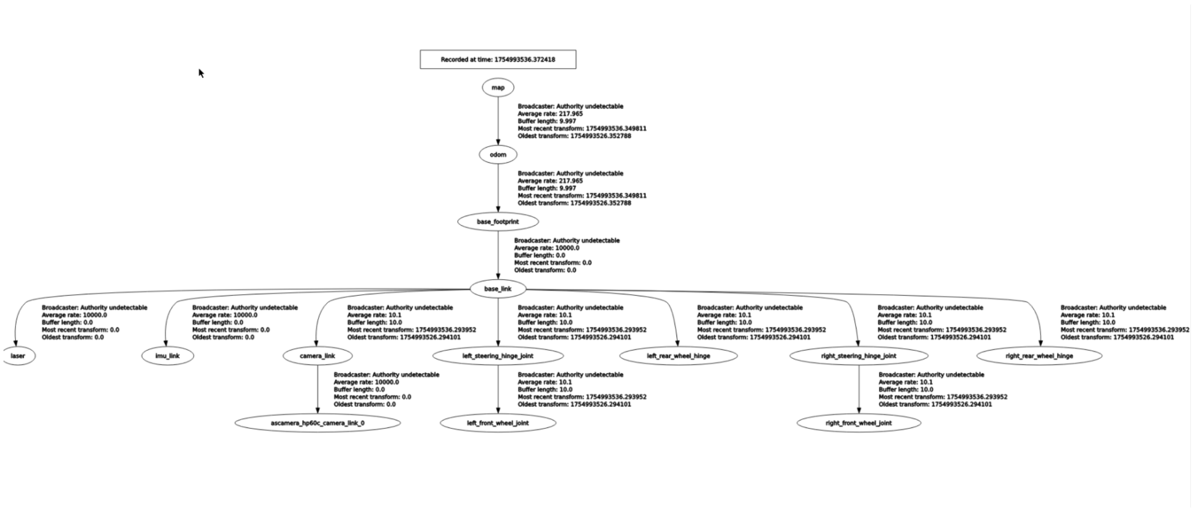
If the graph does not display initially, select [Nodes/Topics (all)] and click the refresh button in the upper left corner.

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.


Recorded at time: 1754993536.372418
map
Broadcaster: Authority undetectable Average rate: 217.965 Buffer length: 9.997 Most recent transform: 1754993536.349811 Oldest transform: 1754993526.352788
odom
Broadcaster: Authority undetectable Average rate: 217.965 Buffer length: 9.997 Most recent transform: 1754993536.349811 Oldest transform: 1754993526.352788
base_footprint
Broadcaster: Authority undetectable Average rate: 10000.0 Buffer length: 0.0 Most recent transform: 0.0 Oldest transform: 0.0
base_link
Broadcaster: Authority undetectable Average rate: 10000.0 Buffer length: 0.0 Most recent transform: 0.0 Oldest transform: 0.0
Broadcaster: Authority undetectable Average rate: 10000.0 Buffer length: 0.0 Most recent transform: 0.0 Oldest transform: 0.0
Broadcaster: Authority undetectable Average rate: 10.1 Buffer length: 10.0 Most recent transform: 1754993536.293952 Oldest transform: 1754993526.294101
Broadcaster: Authority undetectable Average rate: 10.1 Buffer length: 10.0 Most recent transform: 1754993536.293952 Oldest transform: 1754993526.294101
Broadcaster: Authority undetectable Average rate: 10.1 Buffer length: 10.0 Most recent transform: 1754993536.293952 Oldest transform: 1754993526.294101
Broadcaster: Authority undetectable Average rate: 10.1 Buffer length: 10.0 Most recent transform: 1754993536.293952 Oldest transform: 1754993526.294101
Broadcaster: Authority undetectable Average rate: 10.1 Buffer length: 10.0 Most recent transform: 1754993536.293952 Oldest transform: 1754993526.294101
laser
imu_link
camera_link
left_steering_hinge_joint
left_rear_wheel_hinge
right_steering_hinge_joint
right_rear_wheel_hinge
Broadcaster: Authority undetectable Average rate: 10000.0 Buffer length: 0.0 Most recent transform: 0.0 Oldest transform: 0.0
Broadcaster: Authority undetectable Average rate: 10.1 Buffer length: 10.0 Most recent transform: 1754993536.293952 Oldest transform: 1754993526.294101
Broadcaster: Authority undetectable Average rate: 10.1 Buffer length: 10.0 Most recent transform: 1754993536.293952 Oldest transform: 1754993526.294101
ascamera_hp60c_camera_link_0
left_front_wheel_joint
right_front_wheel_joint

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_la
unch.py
```

map_cartographer_launch.py

```python
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])
```

这里运行了一个launch文件-cartographer_launch，该文件位于，

```
~/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(),
        ),
    ])
```

这里主要是运行了cartographer_node建图节点以及occupancy_grid_launch.py，另外加载了参数配置文件，该参数文件位于，

```
~/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
```

```lua
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
```