Gmapping-SLAM mapping

This lesson uses a 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. Configuration and Operation Guide] -- [5. Entering the car Docker (Jetson-Nano and Raspberry Pi 5 users see here)]**.

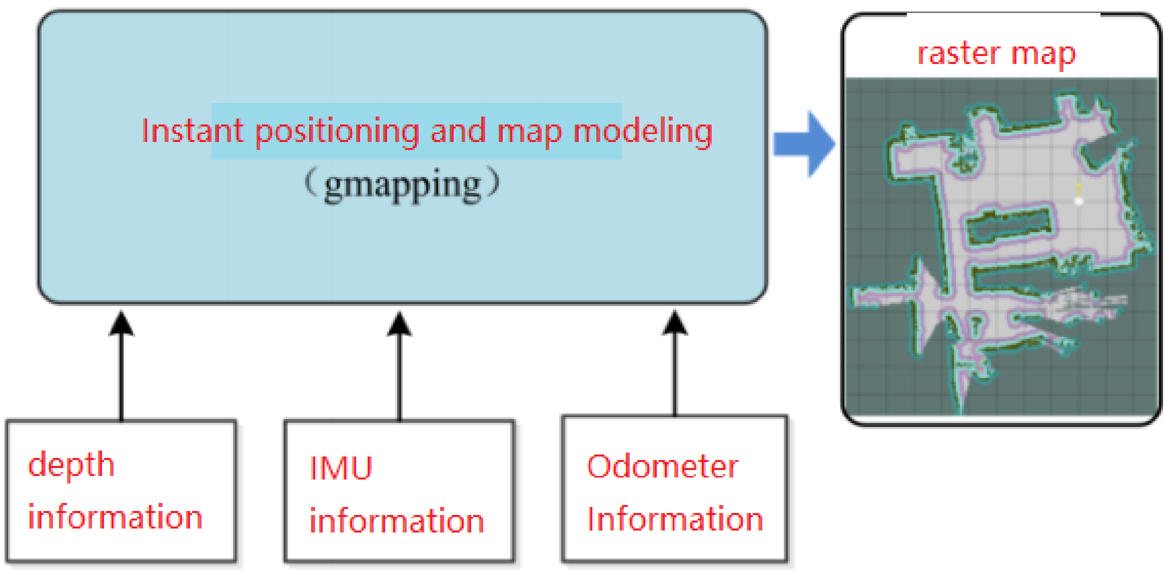
For Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

1. Algorithm Introduction

- gmapping only works when the number of 2D laser points in a single frame is less than 1440. If the number of laser points in a single frame is greater than 1440, the error [[mapping-4] process has died] will occur.
- gmapping is a commonly used open-source SLAM algorithm based on the filter SLAM framework.
- gmapping is based on the RBPF particle filter algorithm, separating the real-time localization and mapping processes, performing localization first and then mapping. Gmapping makes two major improvements to the RBPF algorithm: improved proposal distribution and selective resampling.

Advantages: Gmapping can build indoor maps in real time. It requires less computation and offers high accuracy for small scene maps.

Disadvantages: As the scene size increases, more particles are required. Because each particle carries a map, the memory and computational overhead required for building large maps increases. Therefore, it is not suitable for building large scene maps. Furthermore, it lacks loop detection, which can cause map misalignment when loops are closed. While increasing the number of particles can close the map, it comes at the expense of increased computation and memory.



2. Program Functionality

After running the program, the map building interface will appear in rviz. Use the keyboard or gamepad to control the robot until the map is built. Then, run the save map command to save the map. **(For the Ackermann model car, if you want better mapping, try adding some stopping motion when turning, and avoid making a sharp turn.)**

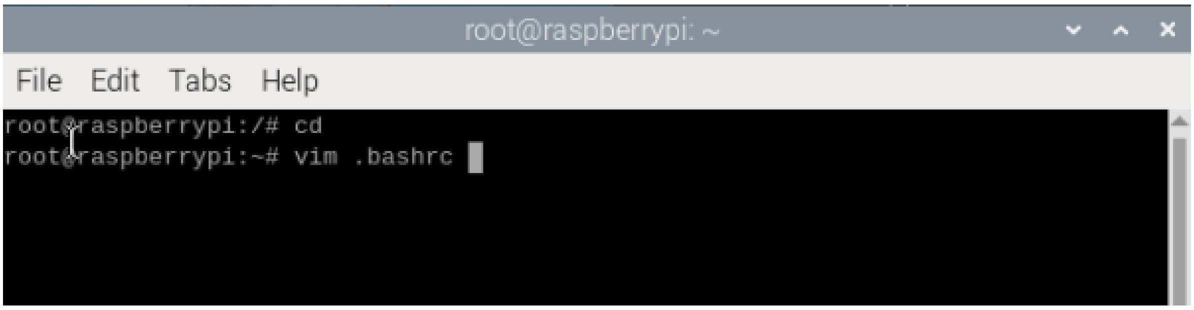
3. Pre-use Configuration

This car is equipped with a USB camera, a depth camera, and two different LiDAR models. However, since the product cannot be automatically identified, you need to manually set the machine type and LiDAR model.

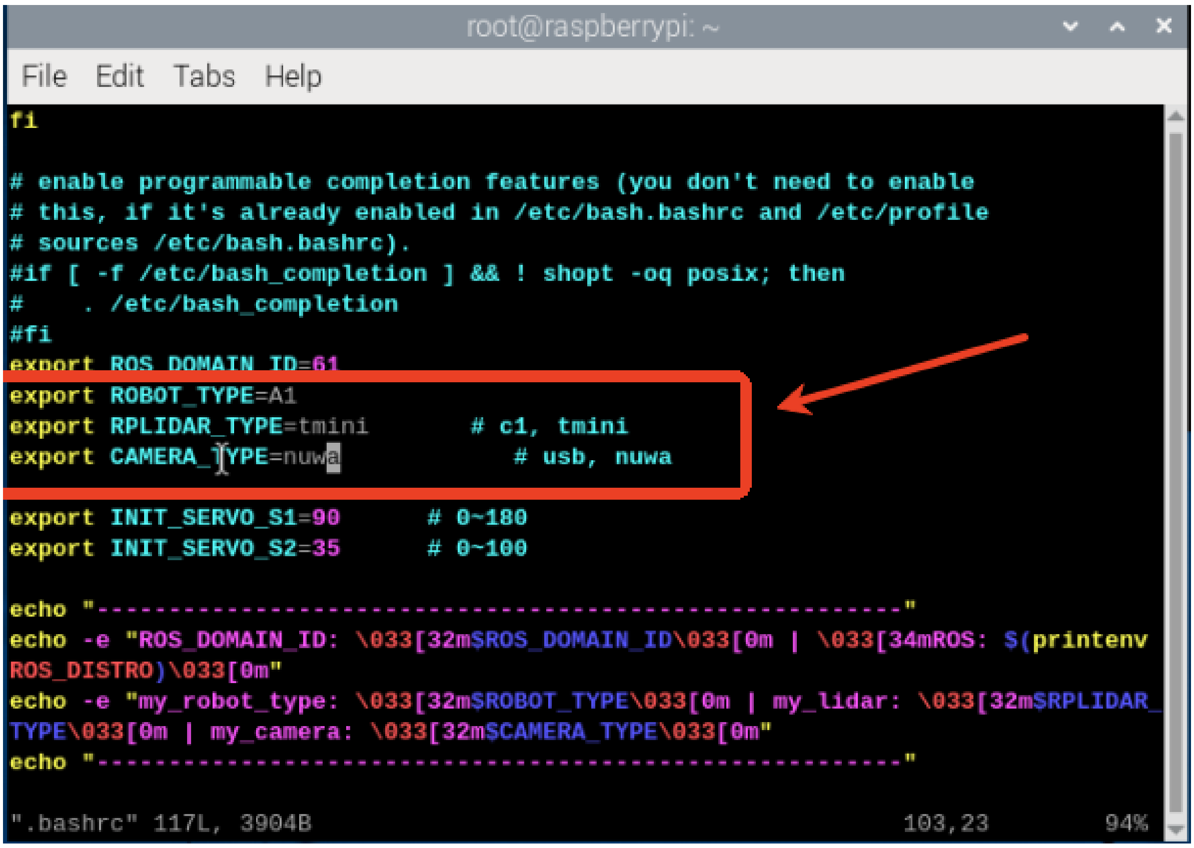
For the Raspberry Pi PI5 controller, you need to first enter the Docker container; the Orin motherboard does not require this.

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

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

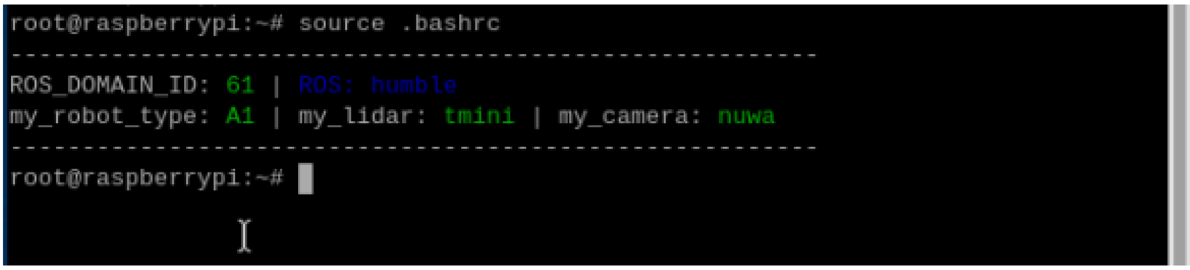


Find this location and press i on your keyboard to change the camera and LiDAR models to the corresponding ones. The default values are Tmini and Nuwa.



After making the changes, 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:~#
```



4. Program Startup

4.1. Startup Commands

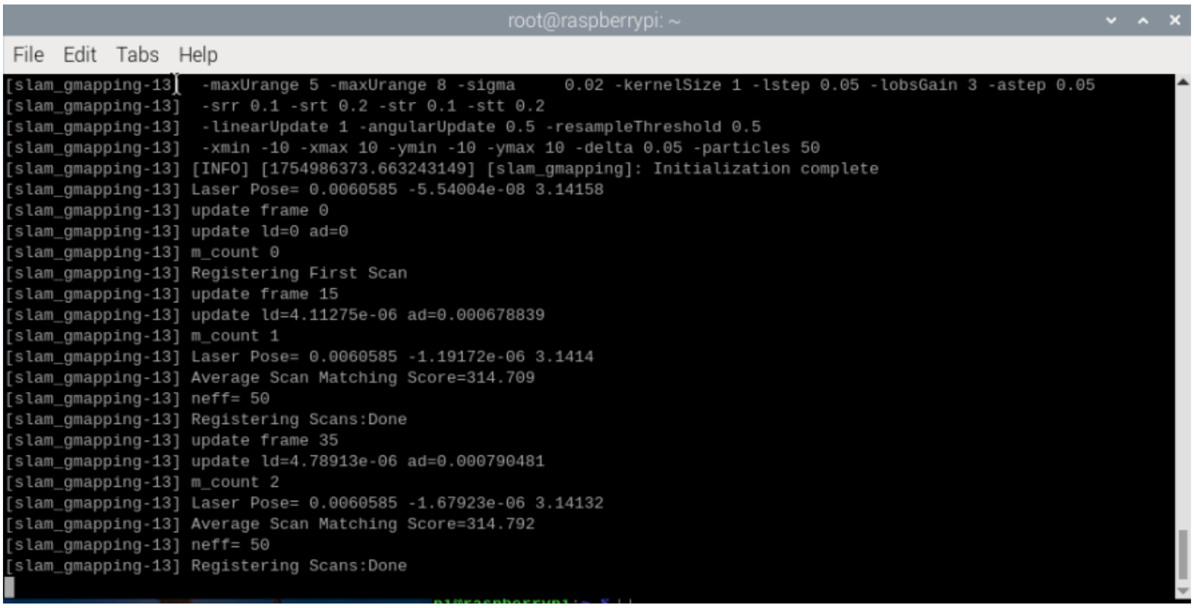
For the Raspberry Pi 5 controller, you must first enter the Docker container. For the Orin controller, this is not necessary.

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

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

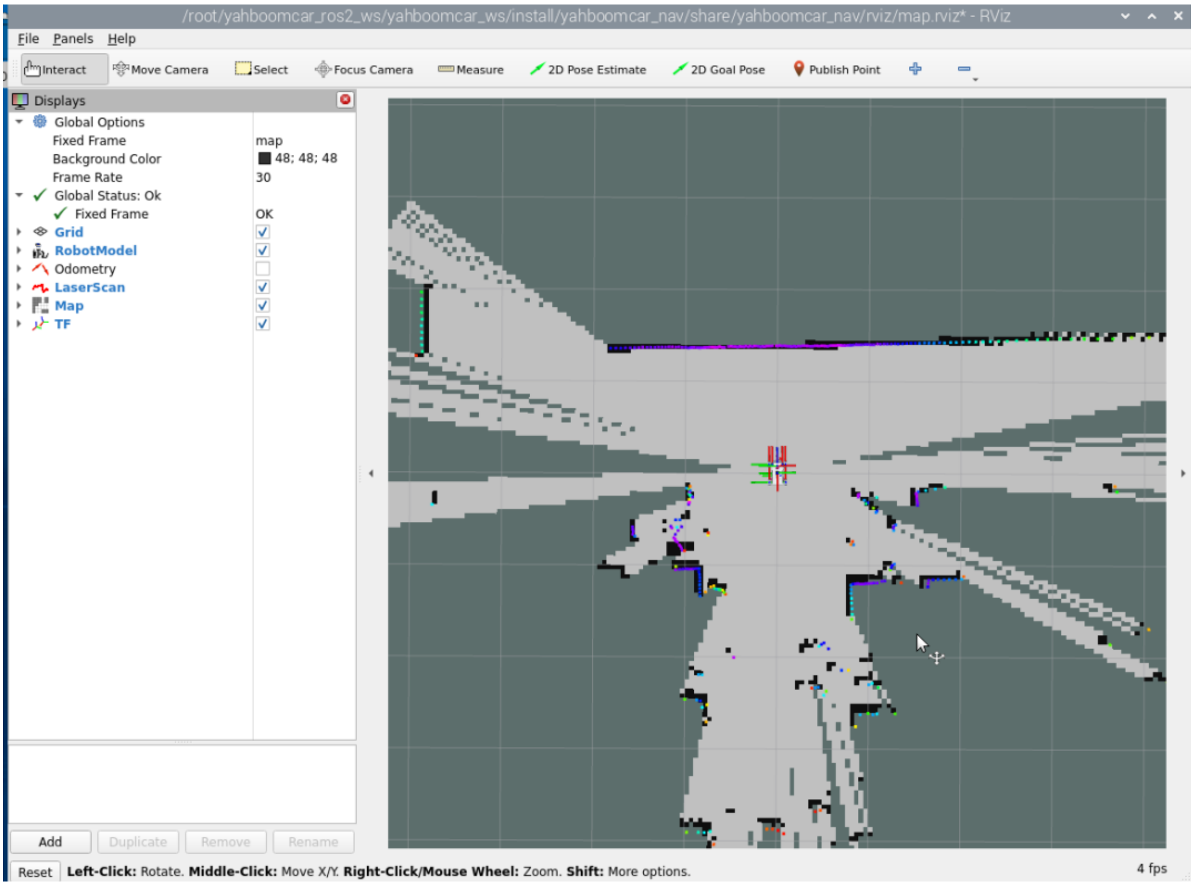
1. Start Mapping

```
ros2 launch yahboomcar_nav map_gmapping_launch.py
```



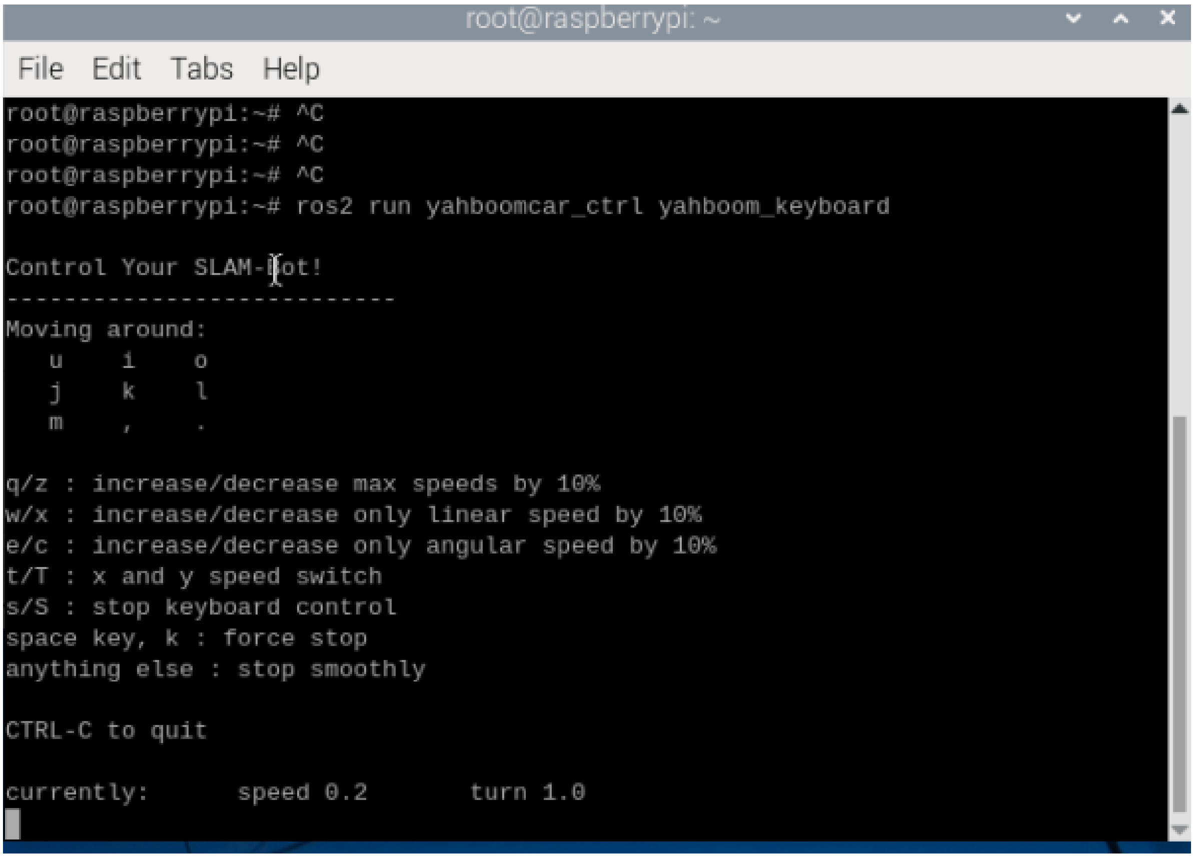
Enter the command to start rviz visualization of the map.

```
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:

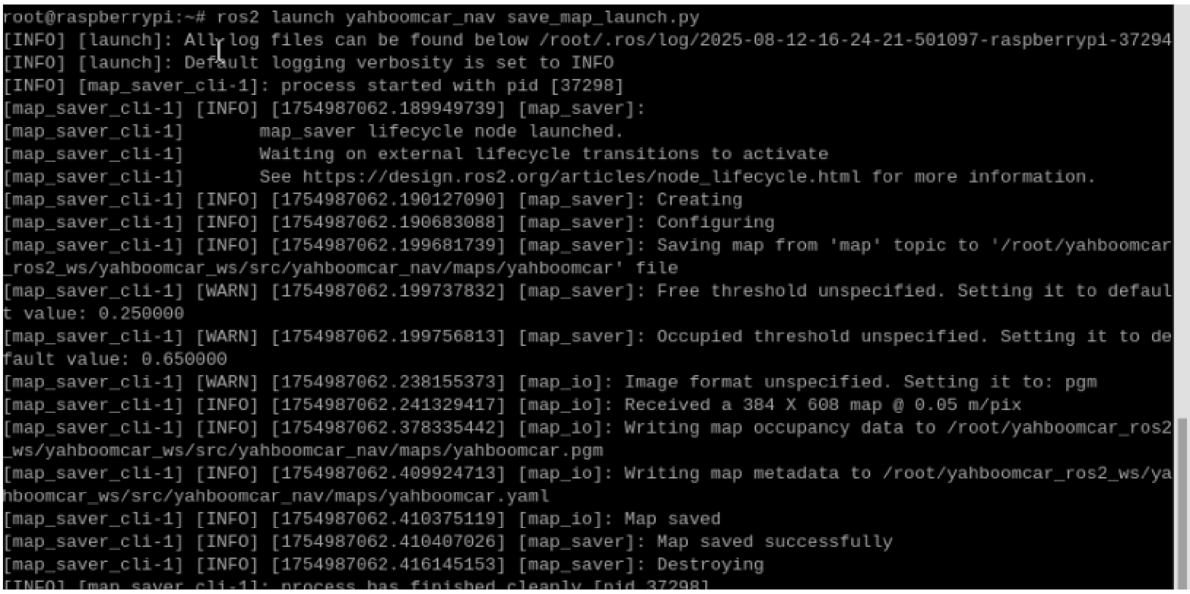
```
#keyboard
ros2 run yahboomcar_ctrl yahboom_keyboard
```



```
root@raspberrypi: ~
File Edit Tabs Help
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 you want to map. After mapping is complete, enter the following command to save the map. Enter:

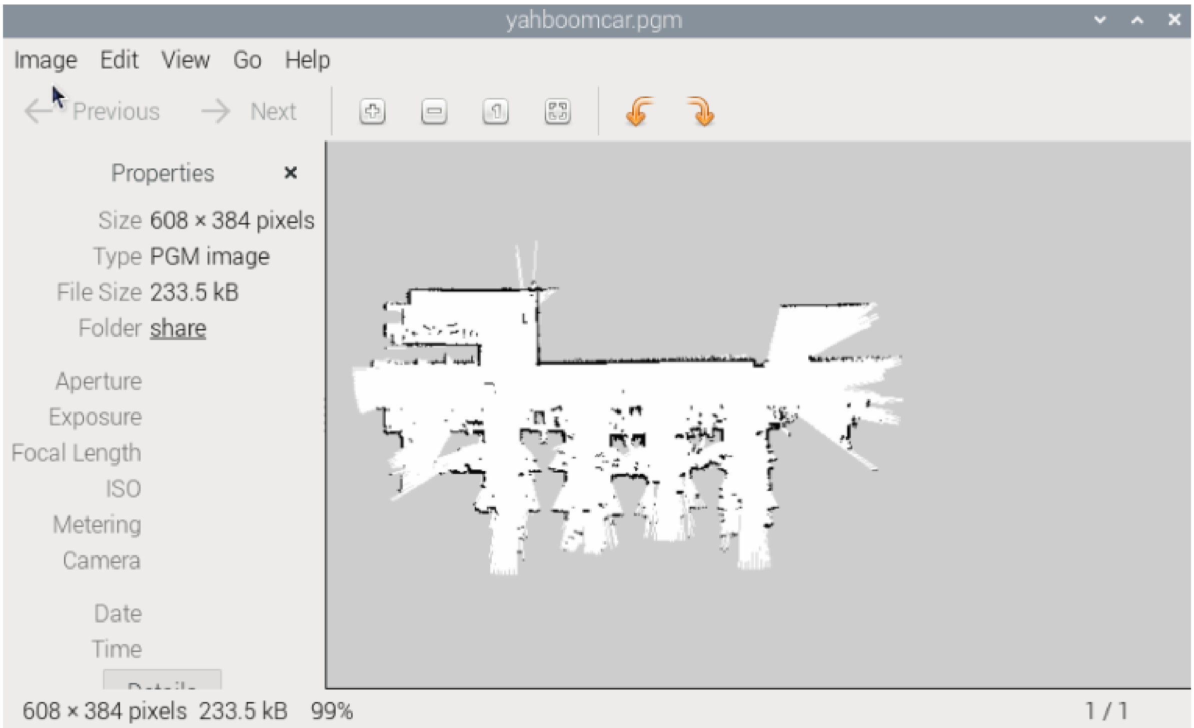
```
ros2 launch yahboomcar_nav save_map_launch.py
```



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 Orin boards, you can directly double-click the map. For Raspberry Pi boards, you need to copy the file to /root/share in Docker.) (This is the shared folder, then view it on the host machine.)



yahboomcar.yaml, take a look at the contents of the yaml.

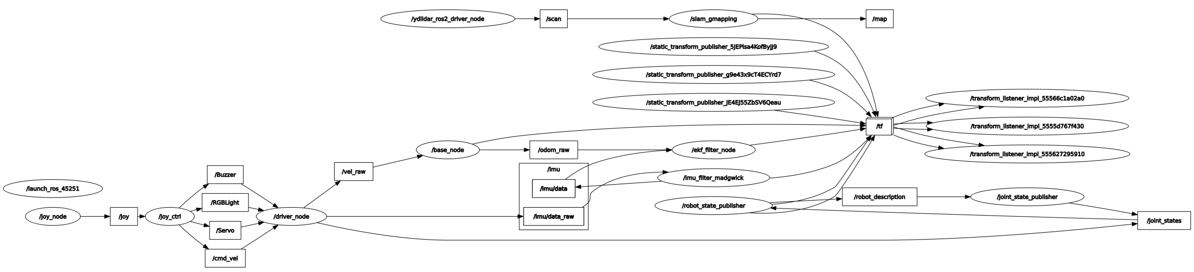
```yaml
image: yahboomcar.pgm
mode: trinary
resolution: 0.05
origin: [-10, -21.2, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

- 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 coordinate system at the bottom left corner of the map. Pose (x, y, yaw), where yaw is counterclockwise rotation (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.

5. View the Node Communication Graph

In the terminal, enter:

```
ros2 run rqt_graph rqt_graph
```



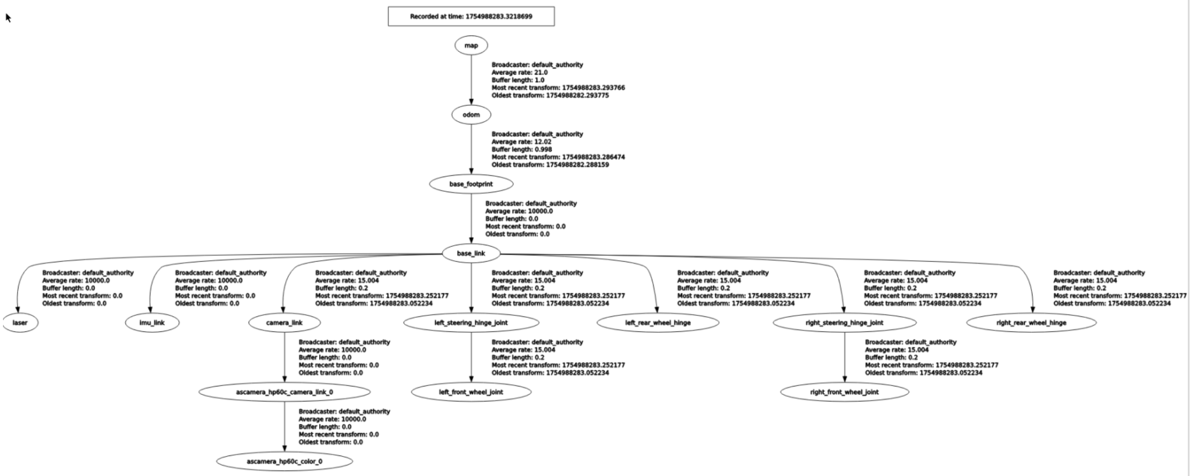
If the graph does not display initially, select [Nodes/Topics (all)] and click the refresh button in the upper left corner.

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_gmapping_launch.py file for map creation. The file path is:

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

map_gmapping_launch.py

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


def generate_launch_description():

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

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

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

Here a launch file - slam_gmapping_launch is started. See slam_gmapping_launch in detail. The file is located at

```
~/yahboomcar_ros2_ws/software/library_ws/src/ros2_gmapping/slam_gmapping/launch/
slam_gmapping.launch.py
```

slam_gmapping.launch.py,

```
from launch import LaunchDescription
from launch.substitutions import EnvironmentVariable
import launch.actions
import launch_ros.actions
import os
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():

    return LaunchDescription([
        launch_ros.actions.Node(
            package='slam_gmapping',
            executable='slam_gmapping',
            output='screen',
            parameters=
[os.path.join(get_package_share_directory("slam_gmapping"), "params",
"slam_gmapping.yaml")]),
    ])
```

Here the slam_gmapping node is started and the slam_gmapping.yaml parameter file is loaded. The file is located at

```
~/yahboomcar_ros2_ws/software/library_ws/src/ros2_gmapping/slam_gmapping/params/
slam_gmapping.yaml
```

slam_gmapping.yaml

```
/slam_gmapping:
  ros__parameters:
    angularUpdate: 0.5
    astep: 0.05
    base_frame: base_footprint
    map_frame: map
    odom_frame: odom
    delta: 0.05
    iterations: 4
    kernelSize: 1
    lasamplerange: 0.005
    lasamplestep: 0.005
    linearUpdate: 1.0
    llsamplerange: 0.01
    llsamplestep: 0.01
    lsigma: 0.075
    lskip: 0
    lstep: 0.05
    map_update_interval: 5.0
    maxRange: 8.0
    maxUrange: 5.0
    minimum_score: 0.1
    occ_thresh: 0.25
    ogain: 3.0
    particles: 50
    qos_overrides:
```

```yaml
    /parameter_events:
      publisher:
        depth: 1000
        durability: volatile
        history: keep_last
        reliability: reliable
    /tf:
      publisher:
        depth: 100
        durability: volatile
        history: keep_last
        reliability: reliable
  resampleThreshold: 0.5
  sigma: 0.02
  srr: 0.1
  srt: 0.2
  str: 0.1
  stt: 0.2
  temporalUpdate: 1.0
  transform_publish_period: 0.05
  use_sim_time: false
  xmax: 10.0
  xmin: -10.0
  ymax: 10.0
  ymin: -10.0
```