ROS Robot App Map Building

Note: The virtual machine needs to be in the same LAN as the car, and the ROS_DOMAIN_ID needs to be consistent. You can view the [Expansion Board Parameter Configuration] to set the IP and ROS_DOMAIN_ID on the board.

1. Program Function Description

The car connects to the proxy, runs the program, opens the [ROS Robot] app downloaded on the mobile phone, enters the IP address of the car, selects ROS2, clicks Connect, and the car can be connected. Place the handheld radar in parallel and slowly control the walking area to complete the map building. Finally, click Save Map, and the car will save the current built map.

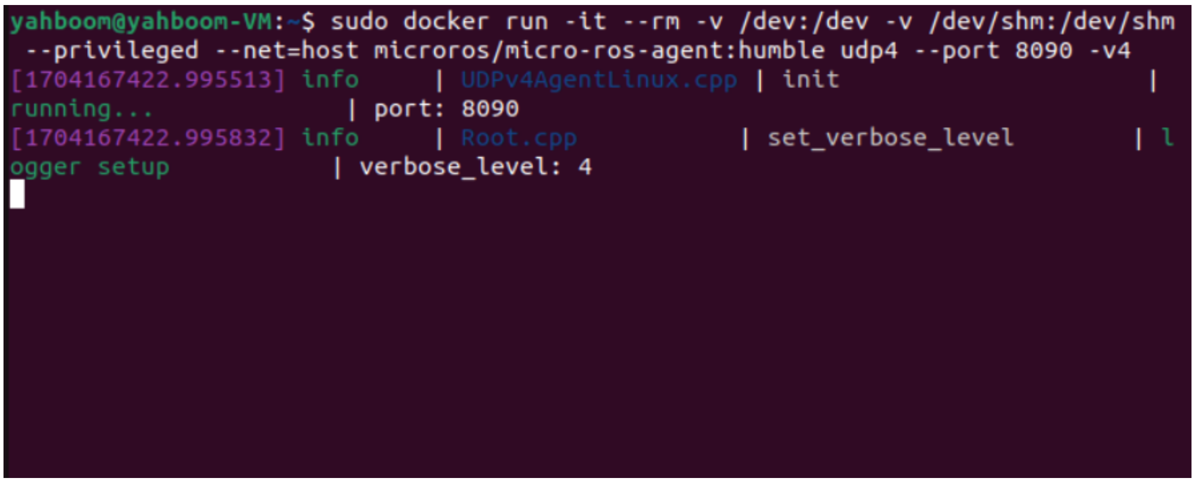
2. Start and connect the agent

If you are using the factory virtual machine system, you can enter in the terminal:

```
sh ~/start_agent_computer.sh
```

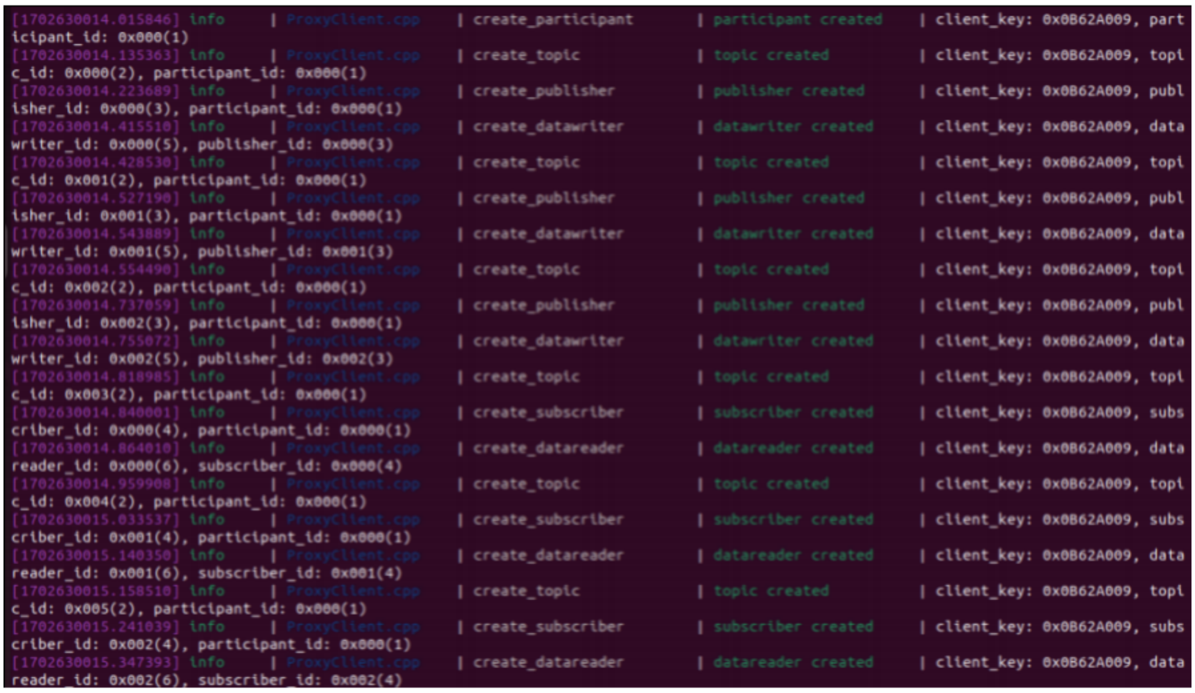
If you are using a third-party virtual machine system, you need to install the docker development environment first, and open the terminal and enter:

```
#CarAgency
sudo docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --
net=host microros/micro-ros-agent:humble udp4 --port 8090 -v4
```



```
yahboom@yahboom-VM:~$ sudo docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm
 --privileged --net=host microros/micro-ros-agent:humble udp4 --port 8090 -v4
[1704167422.995513] info     | UDPv4AgentLinux.cpp | init                     |
running...             | port: 8090
[1704167422.995832] info     | Root.cpp            | set_verbose_level        | l
ogger setup            | verbose_level: 4
```

Then, turn on the car, select the warning level and radar model, and wait for the car to connect to the proxy. The connection is successful as shown in the figure below.



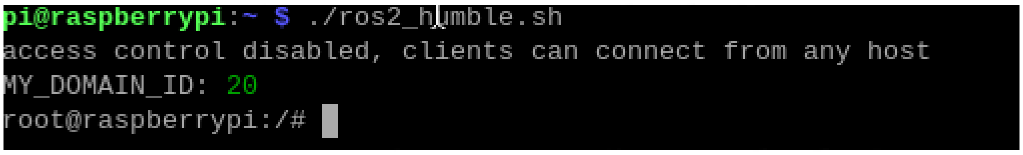
```
[1702630014.015846] info     | ProxyClient.cpp    | create_participant       | participant created    | client_key: 0x0B62A009, part
icipant_id: 0x000(1)
[1702630014.135363] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0B62A009, topi
c_id: 0x000(2), participant_id: 0x000(1)
[1702630014.223689] info     | ProxyClient.cpp    | create_publisher         | publisher created      | client_key: 0x0B62A009, publ
isher_id: 0x000(3), participant_id: 0x000(1)
[1702630014.415510] info     | ProxyClient.cpp    | create_datawriter        | datawriter created     | client_key: 0x0B62A009, data
writer_id: 0x000(5), publisher_id: 0x000(3)
[1702630014.428530] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0B62A009, topi
c_id: 0x001(2), participant_id: 0x000(1)
[1702630014.527190] info     | ProxyClient.cpp    | create_publisher         | publisher created      | client_key: 0x0B62A009, publ
isher_id: 0x001(3), participant_id: 0x000(1)
[1702630014.543889] info     | ProxyClient.cpp    | create_datawriter        | datawriter created     | client_key: 0x0B62A009, data
writer_id: 0x001(5), publisher_id: 0x001(3)
[1702630014.554490] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0B62A009, topi
c_id: 0x002(2), participant_id: 0x000(1)
[1702630014.737059] info     | ProxyClient.cpp    | create_publisher         | publisher created      | client_key: 0x0B62A009, publ
isher_id: 0x002(3), participant_id: 0x000(1)
[1702630014.755072] info     | ProxyClient.cpp    | create_datawriter        | datawriter created     | client_key: 0x0B62A009, data
writer_id: 0x002(5), publisher_id: 0x002(3)
[1702630014.818985] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0B62A009, topi
c_id: 0x003(2), participant_id: 0x000(1)
[1702630014.840001] info     | ProxyClient.cpp    | create_subscriber        | subscriber created     | client_key: 0x0B62A009, subs
criber_id: 0x000(4), participant_id: 0x000(1)
[1702630014.864010] info     | ProxyClient.cpp    | create_datareader        | datareader created     | client_key: 0x0B62A009, data
reader_id: 0x000(6), subscriber_id: 0x000(4)
[1702630014.959908] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0B62A009, topi
c_id: 0x004(2), participant_id: 0x000(1)
[1702630015.033537] info     | ProxyClient.cpp    | create_subscriber        | subscriber created     | client_key: 0x0B62A009, subs
criber_id: 0x001(4), participant_id: 0x000(1)
[1702630015.140350] info     | ProxyClient.cpp    | create_datareader        | datareader created     | client_key: 0x0B62A009, data
reader_id: 0x001(6), subscriber_id: 0x001(4)
[1702630015.158510] info     | ProxyClient.cpp    | create_topic             | topic created          | client_key: 0x0B62A009, topi
c_id: 0x005(2), participant_id: 0x000(1)
[1702630015.241039] info     | ProxyClient.cpp    | create_subscriber        | subscriber created     | client_key: 0x0B62A009, subs
criber_id: 0x002(4), participant_id: 0x000(1)
[1702630015.347393] info     | ProxyClient.cpp    | create_datareader        | datareader created     | client_key: 0x0B62A009, data
reader_id: 0x002(6), subscriber_id: 0x002(4)
```

3. Start the program

If it is the Raspberry Pi desktop version and the Jetson Nano desktop version, you need to enter docker in advance and enter the terminal.

```
sh ~/ros2_humble.sh
```

When the following interface appears, it means that you have successfully entered docker.



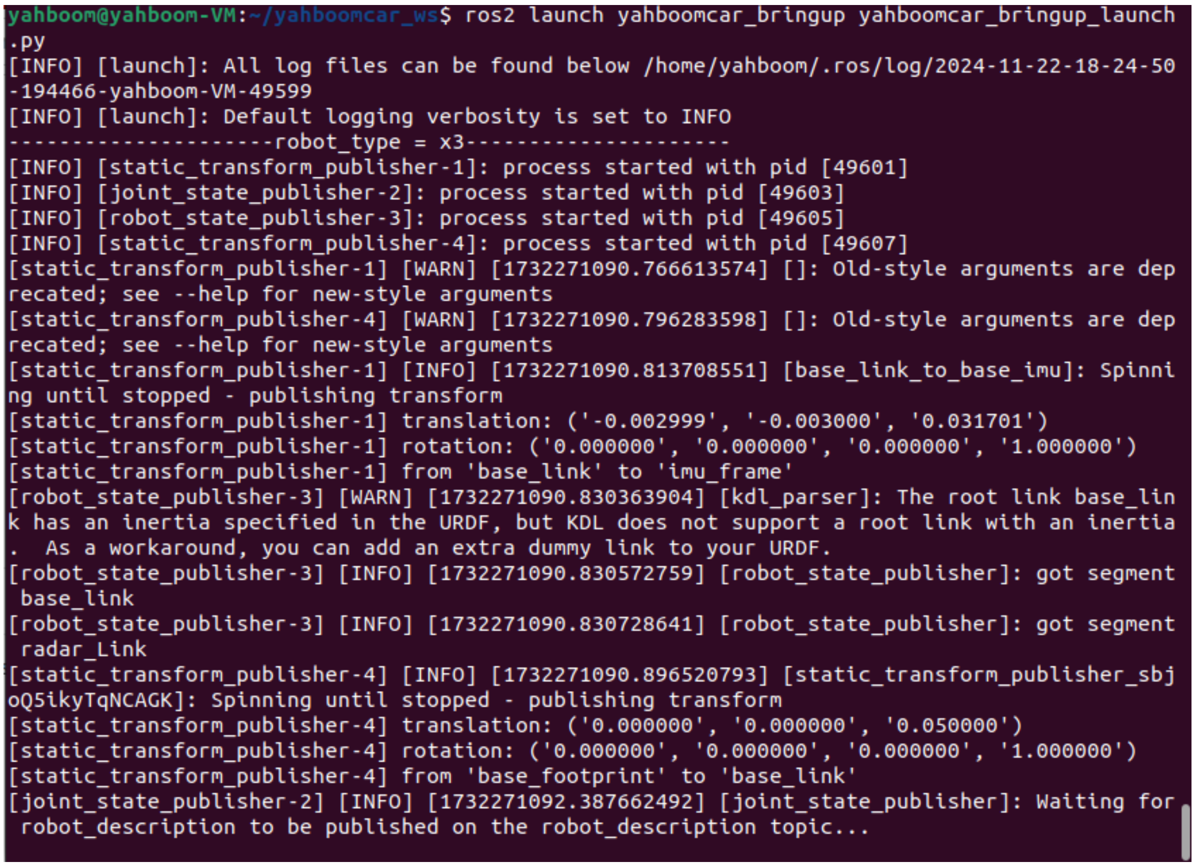
```
pi@raspberrypi:~ $ ./ros2_humble.sh
access control disabled, clients can connect from any host
MY_DOMAIN_ID: 20
root@raspberrypi:/#
```

Then enter in docker separately, **(see [Enter the same docker terminal] section)**

```
ros2 launch yahboomcar_bringup yahboomcar_bringup_launch.py    #underlying data
program
ros2 launch rf2o_laser_odometry rf2o_laser_odometry.launch.py
ros2 launch yahboomcar_nav map_gmapping_app_launch.xml
```

Take the matching virtual machine as an example, first start the car to process the underlying data program, terminal input,

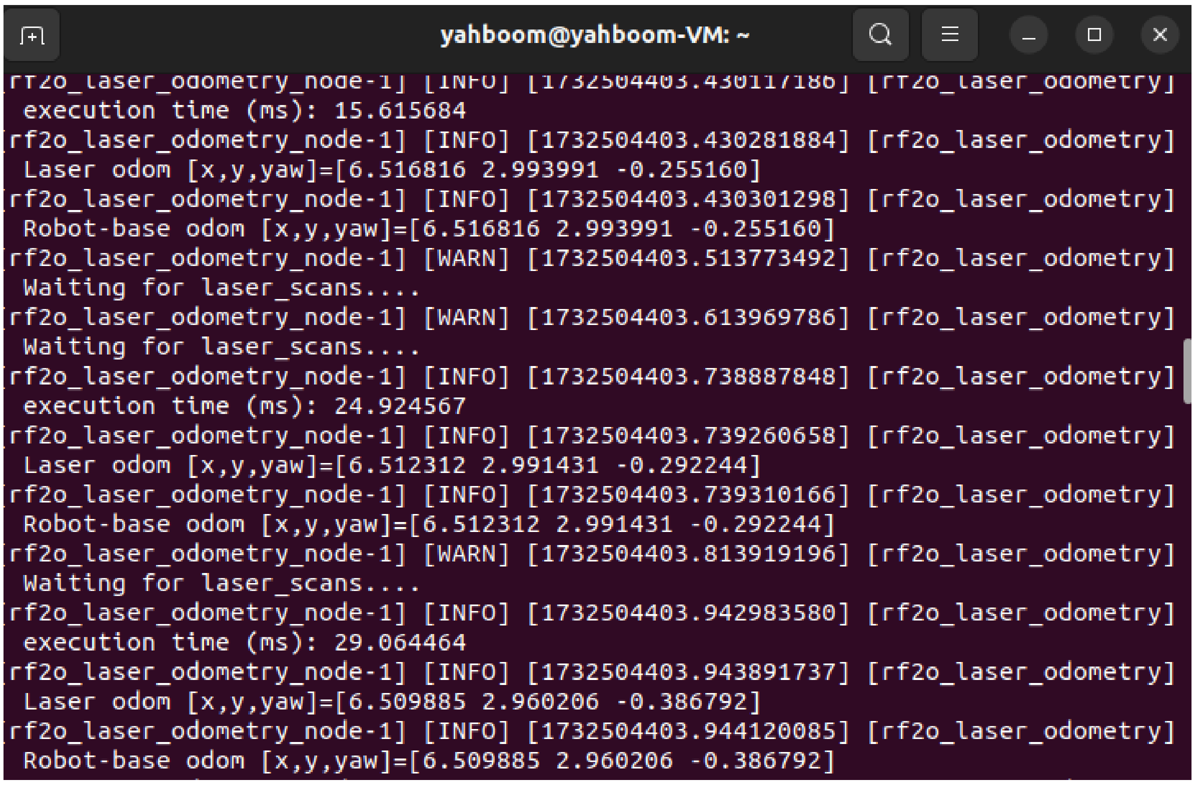
```
ros2 launch yahboomcar_bringup yahboomcar_bringup_launch.py
```



```
yahboom@yahboom-VM:~/yahboomcar_ws$ ros2 launch yahboomcar_bringup yahboomcar_bringup_launch
.py
[INFO] [launch]: All log files can be found below /home/yahboom/.ros/log/2024-11-22-18-24-50
-194466-yahboom-VM-49599
[INFO] [launch]: Default logging verbosity is set to INFO
---------------------robot_type = x3--------------------
[INFO] [static_transform_publisher-1]: process started with pid [49601]
[INFO] [joint_state_publisher-2]: process started with pid [49603]
[INFO] [robot_state_publisher-3]: process started with pid [49605]
[INFO] [static_transform_publisher-4]: process started with pid [49607]
[static_transform_publisher-1] [WARN] [1732271090.766613574] []: Old-style arguments are dep
recated; see --help for new-style arguments
[static_transform_publisher-4] [WARN] [1732271090.796283598] []: Old-style arguments are dep
recated; see --help for new-style arguments
[static_transform_publisher-1] [INFO] [1732271090.813708551] [base_link_to_base_imu]: Spinni
ng until stopped - publishing transform
[static_transform_publisher-1] translation: ('-0.002999', '-0.003000', '0.031701')
[static_transform_publisher-1] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-1] from 'base_link' to 'imu_frame'
[robot_state_publisher-3] [WARN] [1732271090.830363904] [kdl_parser]: The root link base_lin
k has an inertia specified in the URDF, but KDL does not support a root link with an inertia
.  As a workaround, you can add an extra dummy link to your URDF.
[robot_state_publisher-3] [INFO] [1732271090.830572759] [robot_state_publisher]: got segment
 base_link
[robot_state_publisher-3] [INFO] [1732271090.830728641] [robot_state_publisher]: got segment
 radar_Link
[static_transform_publisher-4] [INFO] [1732271090.896520793] [static_transform_publisher_sbj
oQ5ikyTqNCAGK]: Spinning until stopped - publishing transform
[static_transform_publisher-4] translation: ('0.000000', '0.000000', '0.050000')
[static_transform_publisher-4] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-4] from 'base_footprint' to 'base_link'
[joint_state_publisher-2] [INFO] [1732271092.387662492] [joint_state_publisher]: Waiting for
 robot_description to be published on the robot_description topic...
```

Start the laser odometer, input in the terminal,

```
ros2 launch rf2o_laser_odometry rf2o_laser_odometry.launch.py
```

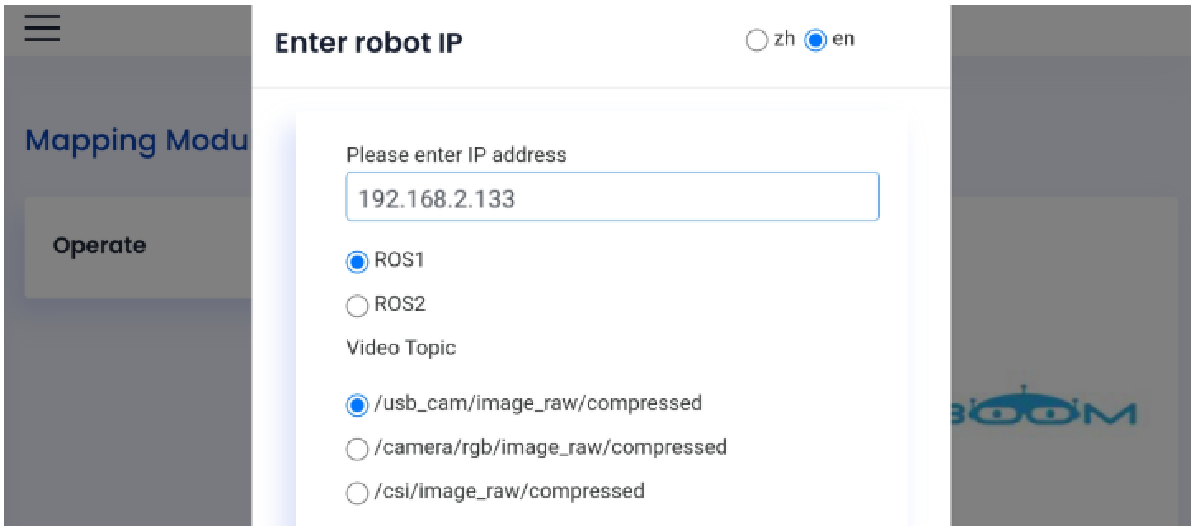


```
rf2o_laser_odometry_node-1] [INFO] [1732504403.430117186] [rf2o_laser_odometry]
 execution time (ms): 15.615684
rf2o_laser_odometry_node-1] [INFO] [1732504403.430281884] [rf2o_laser_odometry]
 Laser odom [x,y,yaw]=[6.516816 2.993991 -0.255160]
rf2o_laser_odometry_node-1] [INFO] [1732504403.430301298] [rf2o_laser_odometry]
 Robot-base odom [x,y,yaw]=[6.516816 2.993991 -0.255160]
rf2o_laser_odometry_node-1] [WARN] [1732504403.513773492] [rf2o_laser_odometry]
 Waiting for laser_scans....
rf2o_laser_odometry_node-1] [WARN] [1732504403.613969786] [rf2o_laser_odometry]
 Waiting for laser_scans....
rf2o_laser_odometry_node-1] [INFO] [1732504403.738887848] [rf2o_laser_odometry]
 execution time (ms): 24.924567
rf2o_laser_odometry_node-1] [INFO] [1732504403.739260658] [rf2o_laser_odometry]
 Laser odom [x,y,yaw]=[6.512312 2.991431 -0.292244]
rf2o_laser_odometry_node-1] [INFO] [1732504403.739310166] [rf2o_laser_odometry]
 Robot-base odom [x,y,yaw]=[6.512312 2.991431 -0.292244]
rf2o_laser_odometry_node-1] [WARN] [1732504403.813919196] [rf2o_laser_odometry]
 Waiting for laser_scans....
rf2o_laser_odometry_node-1] [INFO] [1732504403.942983580] [rf2o_laser_odometry]
 execution time (ms): 29.064464
rf2o_laser_odometry_node-1] [INFO] [1732504403.943891737] [rf2o_laser_odometry]
 Laser odom [x,y,yaw]=[6.509885 2.960206 -0.386792]
rf2o_laser_odometry_node-1] [INFO] [1732504403.944120085] [rf2o_laser_odometry]
 Robot-base odom [x,y,yaw]=[6.509885 2.960206 -0.386792]
```

Start the APP map building command, input in the terminal,

```
ros2 launch yahboomcar_nav map_gmapping_app_launch.xml
```

The mobile APP is displayed as shown below. Enter the IP address of the car. [zh] means Chinese and [en] means English. Select ROS2. Select /usb_cam/image_raw/compressed in the Video Tpoic below. Finally, click [Connect].


Enter robot IP ○ zh [✓ en]

Please enter IP address

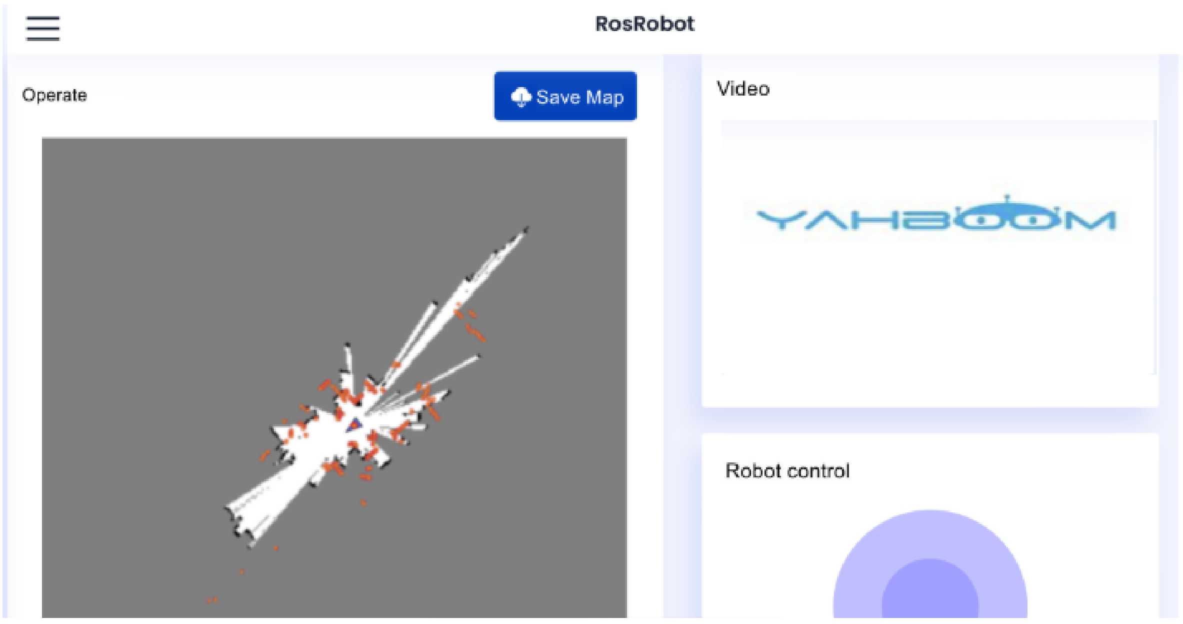
192.168.2.133

[✓ ROS1]
○ ROS2

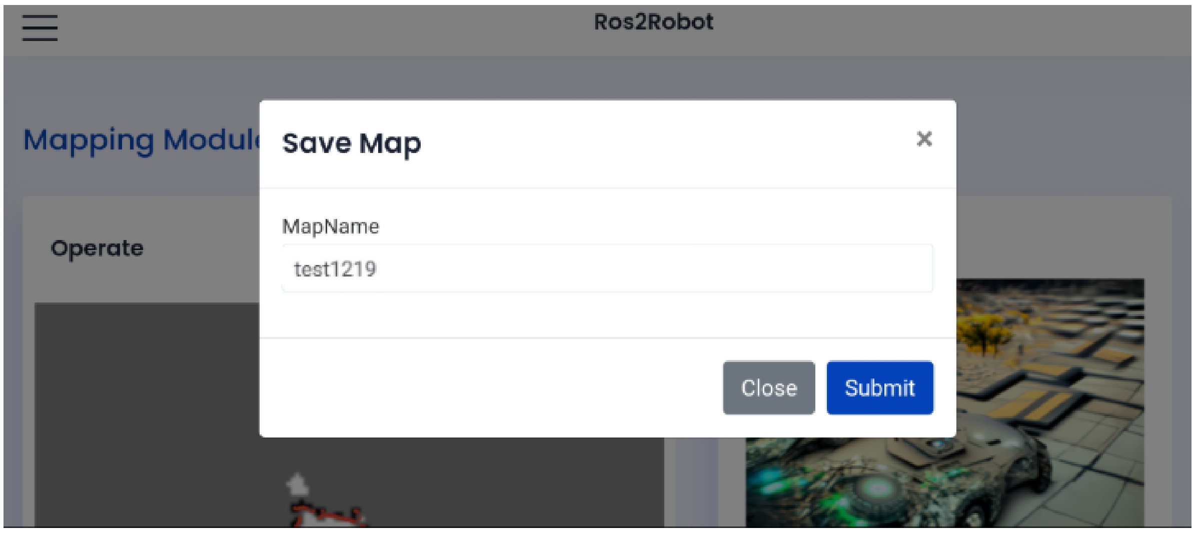
Video Topic

[✓ /usb_cam/image_raw/compressed]
○ /camera/rgb/image_raw/compressed
○ /csi/image_raw/compressed

After successful connection, the display is as follows.



Control the handheld radar to slowly move through the area where the map needs to be built, and then click Save Map. Enter the map name and click Submit to save the map

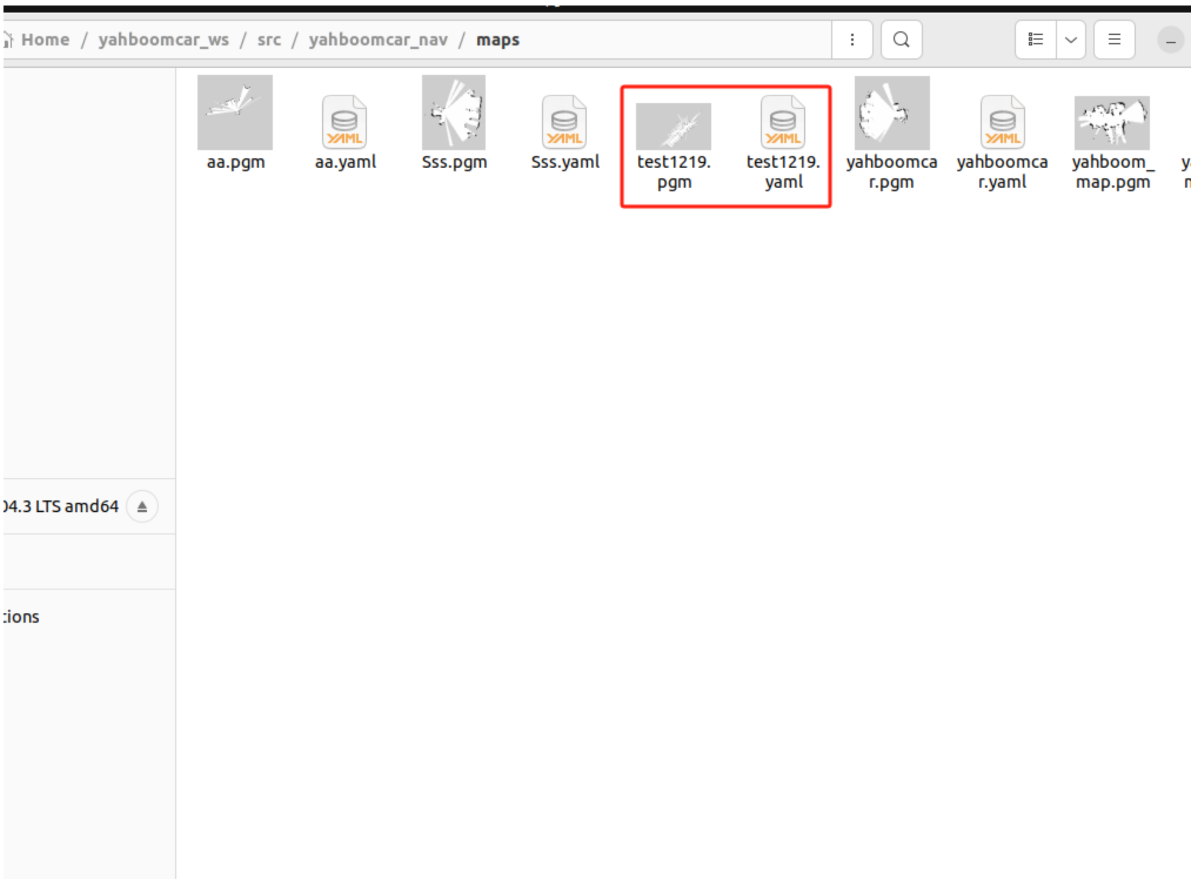

Save Map

MapName

test1219

The map is saved in,

```
/home/yahboom/yahboomcar_ws/src/yahboomcar_nav/maps
```



4. Code analysis

Here is the launch file for opening the APP map, taking gmapping map as an example,

map_gmapping_app_launch.xml

```
<launch>
    <include file="$(find-pkg-share
rosbridge_server)/launch/rosbridge_websocket_launch.xml"/>
    <node name="laserscan_to_point_publisher" pkg="laserscan_to_point_publisher"
exec="laserscan_to_point_publisher"/>
    <include file="$(find-pkg-share
yahboomcar_nav)/launch/map_gmapping_launch.py"/>
    <include file="$(find-pkg-share
robot_pose_publisher_ros2)/launch/robot_pose_publisher_launch.py"/>
    <include file="$(find-pkg-share
yahboom_app_save_map)/yahboom_app_save_map.launch.py"/>
</launch>
```

Here are several launch files and nodes:

- rosbridge_websocket_launch.xml: Open rosbridge service related nodes. After startup, you can connect to ROS through the network
- laserscan_to_point_publisher: Publish the point cloud conversion of the radar to the APP for visualization
- map_gmapping_launch.py: gmapping map building program
- robot_pose_publisher_launch.py: Car pose publishing program, car pose is visualized in the APP
- yahboom_app_save_map.launch.py: program for saving maps