YUHESEN

# OS-mate2

# Open Source ROS Robot Development Platform

Version Ubuntu 22.04 Humble

**User manual V1.0.1**

Shenzhen yuhesen Technology Co., Ltd.

# CONTENTS

# Foreword

ROS is a robot development system built on Ubuntu, consisting of a series of software packages and communication architecture. For users who may be more familiar with Windows, Ubuntu might be less familiar, which can present additional challenges when learning ROS. Fortunately, the rapid development of the internet industry in China has led to numerous books and online resources about Ubuntu and Linux, making self-learning much more accessible. When it comes to learning ROS, developers can choose suitable methods based on their own circumstances. Here are two suggestions that may be helpful:

For developers who have sufficient time and patience for learning, it is recommended to start by studying the usage of Ubuntu through books and online tutorials. Then, explore the ROS official wiki to gain a detailed understanding of ROS and its various concepts. Implement the tutorials and examples provided in the ROS official wiki to gain hands-on experience. Finally, return to this tutorial (note that some programming experiments in this tutorial may need modifications due to different navigation components, but they can provide programming insights).

For developers with tight project deadlines or less patience for theoretical learning, it is possible to start practicing with this tutorial directly. Quickly get the robot up and running to avoid extinguishing the enthusiasm for learning with tedious theory. After gaining a general experience with ROS through the exercises in this tutorial, go back and fill any gaps in theoretical knowledge.

# Safety Information

- Regular Inspection and Maintenance:
  Regularly inspect and maintain all components of the robot, including sensors, batteries, motors, and structural parts, to ensure they are functioning properly and in good condition. Replace worn-out parts in a timely manner to prevent accidents caused by robot failures.

- Safe Operation:
  Before operating the ROS2 mobile robot, make sure you understand and are familiar with the relevant operation manuals and safety guidelines. Follow the correct operating procedures to avoid unnecessary risks. Avoid using the robot for tasks beyond its design and capabilities.

- Keep the Surrounding Environment Clean:
  Ensure that the work area around the robot is clean and tidy, avoiding the presence of obstacles and debris. Remove any wires, ropes, or other objects on the floor that may cause the robot to trip.

- Safety Isolation:
  When operating the robot, ensure that no personnel enter the robot's workspace to prevent accidental injuries. Use appropriate safety isolation devices, such as fences or signage, as needed to alert others not to approach the robot.

- Emergency Stop Device:
  At all times, ensure that the robot is equipped with an emergency stop device, and that the device is easily accessible. In case of an emergency, being able to quickly cut off the power to the robot can prevent injuries.

- Data Backup and Network Security:
  For critical data and configuration files on the ROS2 mobile robot, regularly perform backups and take appropriate measures to protect the robot and network security. Ensure that the robot's software and firmware are always up to date to ensure system reliability and security.

## Navigation Operating Attentions

- Regarding Software Reboot:
  After rebooting the industrial control computer using the "reboot" command or through the interface, the CAN card on the car cannot be used normally, so you can only perform a power cycle reboot.

- Ubuntu System Version:
  The Yuhesen ROS2 robot embedded computer is installed with the version of Ubuntu 22.04 Humble. (**Note: If the Ubuntu interface on the robot prompts to upgrade the system, close it and do not update the system version.**)

- Software Installation and Uninstallation:
  When installing or uninstalling software, carefully review the information printed in the terminal, as it may involve other software. If necessary, you can back up the system first (refer to Appendix 2).

- Software Packages:
  The Yuhesen robot's computer comes pre-installed with ROS2, IDE, and the Yuhesen robot's source code package, which can be used directly.

- File Modifications:
  It is recommended to make backups of any files provided in the Yuhesen source code package before making any modifications.

- Regarding Internet Connectivity:
  All operations described in this document (except for software package installation) do not require internet connectivity. It is recommended to use USB tethering with a mobile phone for internet access.

- Connecting a Screen:
  For all terminal operations mentioned in this instruction document, the industrial control computer on the car needs to be connected to a screen.

- Operating Methods without Connecting a Screen:
  Please refer to Appendix 1.

# Navigation Function

- Obstacle Detecting

| Sensors | Function |
|---|---|
| 3D LiDAR | Detecting dynamic/static obstacles |
| Depth Camera | Detecting dynamic/static obstacles |
| Ultrasonic Radar | Detecting dynamic/static obstacles |

- 2D Mapping

| Scene | Function |
|---|---|
| Indoor | Scan and create 2D grid map. |

- 3D Mapping

| Scene | Function |
|---|---|
| Outdoor | Scan and create 3D point cloud map. |

- 2D Positioning

| Scene | Function |
|---|---|
| Indoor | Using AMCL positioning |

- 3D Positioning

| Scene | Function |
|---|---|
| Indoor | Using NDT positioning |
| Outdoor | Using NDT positioning |

- 3D Navigation

| Scene | Function |
|---|---|
| Indoor | Single point navigation |
| | Multiple points navigation |
| | Path recording navigation |
| Outdoor | Single point navigation |
| | Multiple points navigation |
| | Path recording navigation |

- Obstacle Avoidance

| Scene | Function |
|---|---|
| Indoor | obstacle avoiding during navigation |
| Outdoor | obstacle avoiding during navigation |

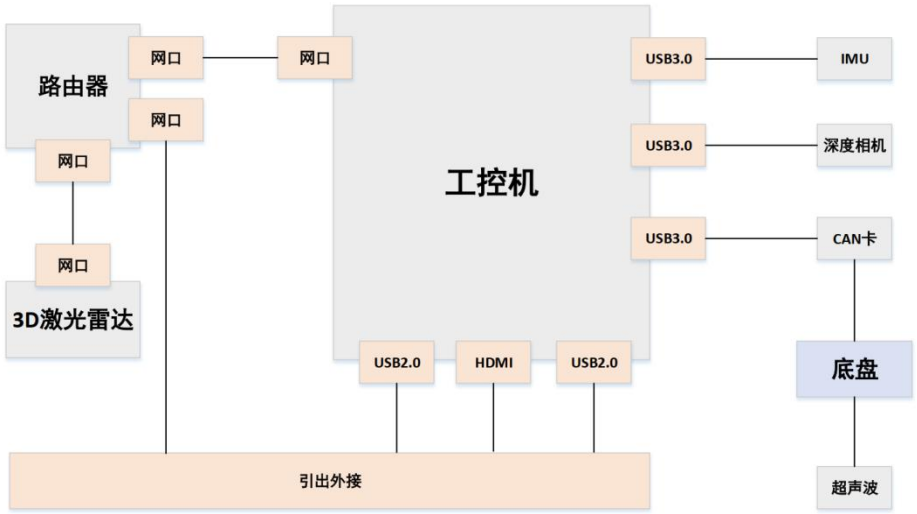
# Navigation Hardwares

Components Parameter

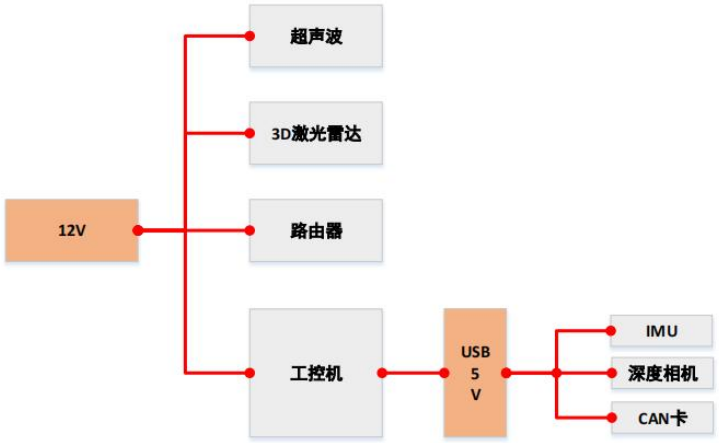
| | |
|---|---|
| Embedded Computer | CPU: i5-8265U/1.6GHz |
| | Memory: 8G DDR4 |
| | Storage: 256G |
| | USB：USB3.0*4　USB2.0*2 |
| | Network interface: Gigabit network interface*2 |
| | Display interface: HDMI |
| | Serial port: DB9 RS232*4 |
| 3D LiDAR | Measurement Range: 0.2m~150m |
| | FOV – Horizontal: 360° |
| | FOV – Vertical: -15°~ +15° |
| | Ranging Accuracy: 2cm |
| | Angle resolution: 0.18°（10HZ） |
| | Scanning frequency: 10HZ |
| | Communication: Network interface |
| IMU | Gyro Measurement Range: ±2000°/s<br>Accelerometer Measurement Range: ±8G(1G = 1x gravitational acceleration) |
| | Gyro Resolution ratio: 0.01°/s<br>Accelerometer Resolution ratio: 1uG |
| | Gyro Zero-bias stability: 8°/h<br>Accelerometer Zero-bias stability: 60uG |
| | Communication: USB to serial port |
| Depth Camera | Range: 0.2~4m |
| | Working Temperature: -10~50℃ |
| | FOV Depth: H73.8°× V58.8°<br>RGB: H80.9°× V51.7° |
| | Resolution ratio@fps: 640×480@30fps |
| | Communication: USB2.0 |
| CAN Adapter | Communication: CAN to USB |
| Router | Network standards: 802.11ac |
| | Wireless network standard: 2.4G/5G<br>Network interface: 1900M Gigabit Port*3 |

# Navigation Topology

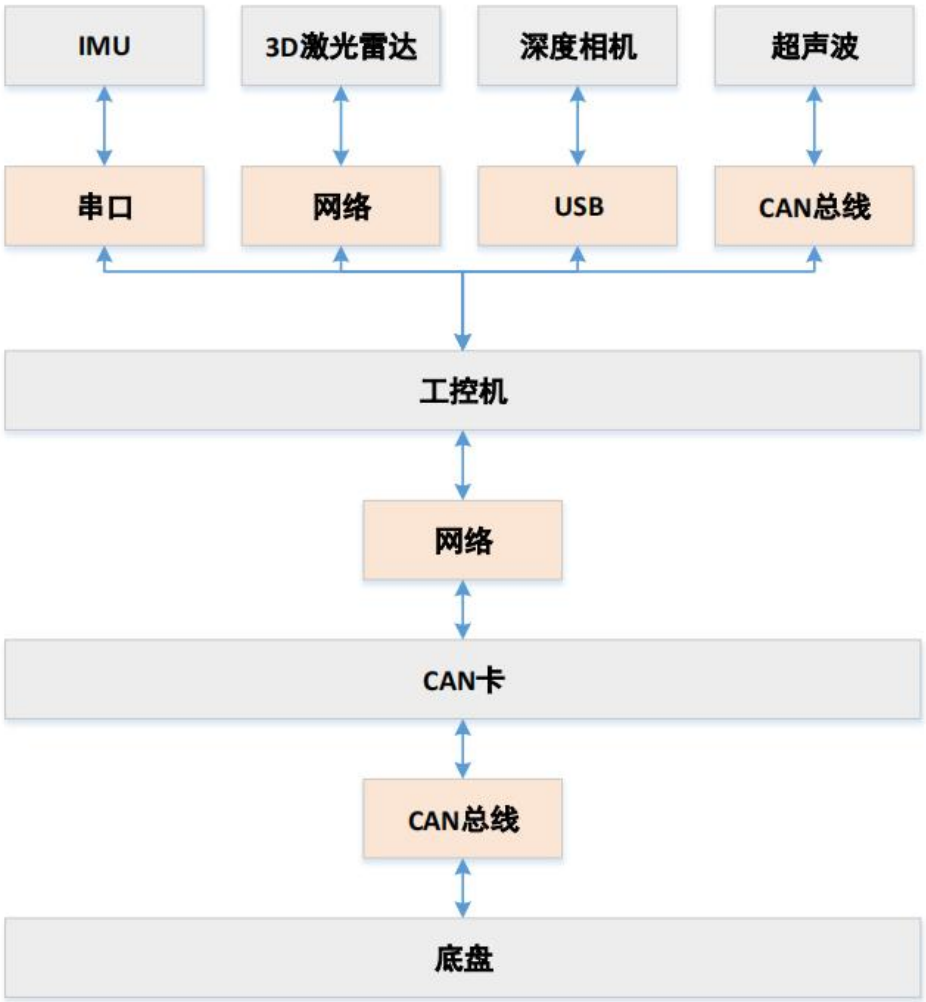
- Components Connecting Topology




- Power Supply Connecting Topology



- Communication Connecting Topology



# Quick Start Guide

- Router WiFi Name and Password

| | |
|---|---|
| Wifi Name | YHS-ROS2-XX（XX indicate the model no. of chassis） |
| Wifi password, router login password | 12345678 |

- Username and Password

| | |
|---|---|
| Ubuntu Username | yhs |
| Root password | 123456 |
| Source code content | /home/yhs/ros2_ws/src |

- Before powering on the robot, please carefully read the user manual provided with the robot and familiarize yourself with the operation of the remote controller.

- After powering on, drive the robot remotely to a relatively open area to avoid collisions, especially if you are not yet proficient. It is recommended to set the control speed to the lowest level.

- If you are not familiar with the Ubuntu system, you can read the Terminal Operations Introduction in the next chapter. Follow the steps to become more familiar before attempting to enable the car's autonomous driving function.

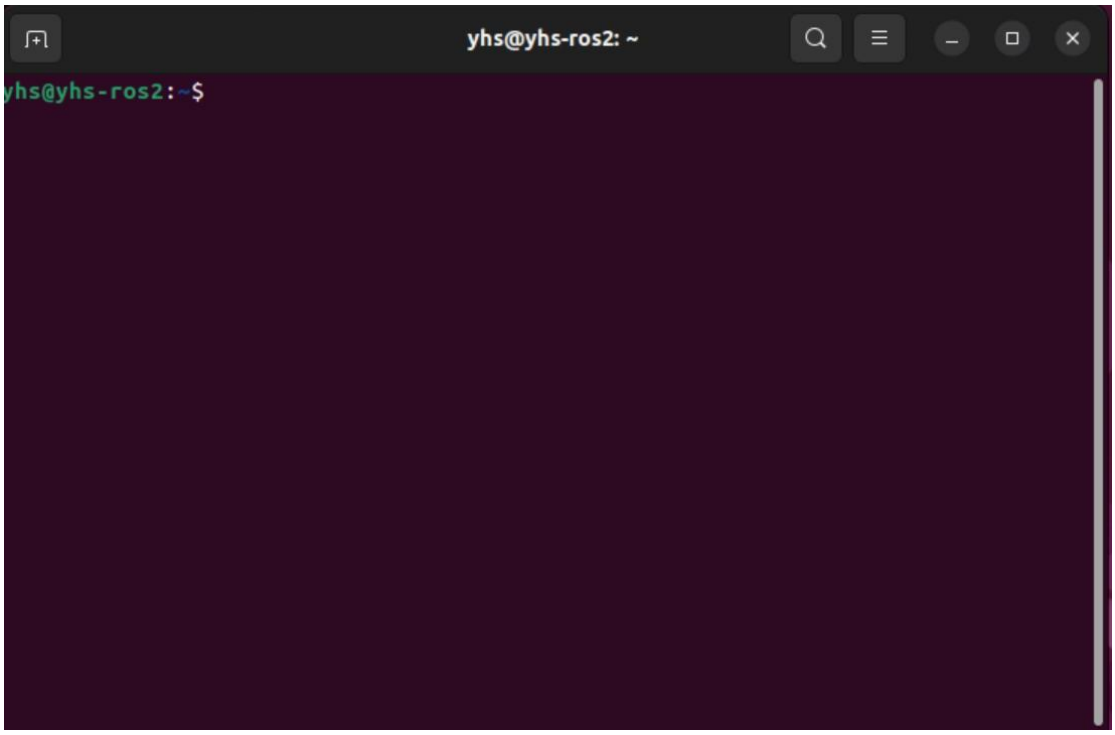
- To enable the car's autonomous driving function without a connected screen, you will need a computer with ROS2 installed and connected to the car's Wi-Fi. Follow the instructions in Appendix One to remotely control the car. Then, you can skip the previous experimental sections and proceed with the instructions in Experiment 6, Experiment 7, and Experiment 9.

# Command Operation at Terminal

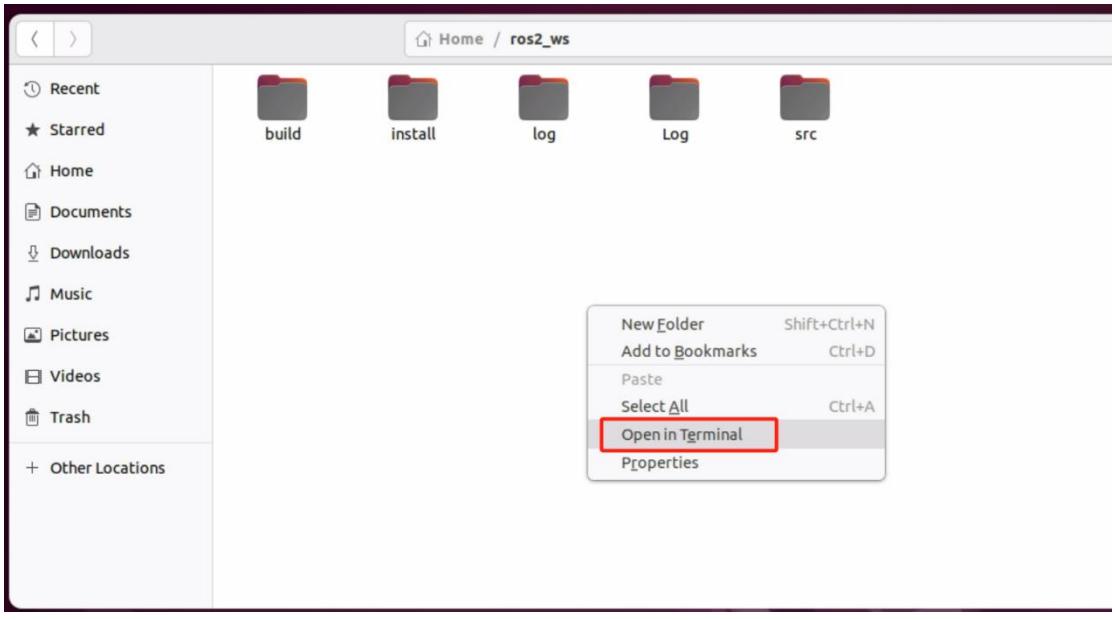
This document may involve the following terminal command operations in the upcoming sections.

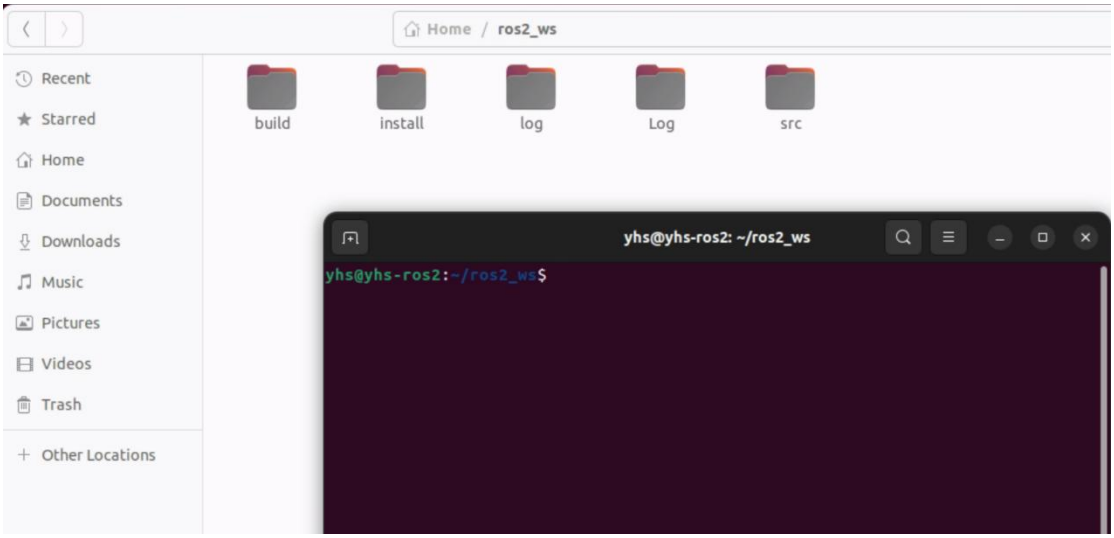
1. Opening the Terminal

（1）Press the shortcut keys "Ctrl + Alt + T". The default directory is the "home" directory.




（2）Open in the file, the path is the location of the file.



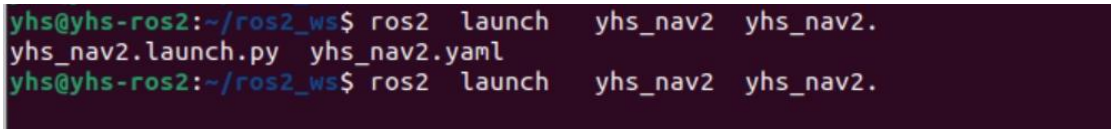


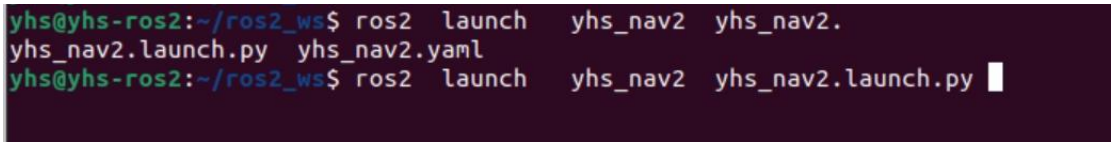

2. Tab" Key Command Completion

（1）When running a program in the terminal, if a command is long or complex, you can use the "Tab" key for completion. The complete command is as follows:

```
ros2 launch yhs_nav2 yhs_nav2.launch.py
```

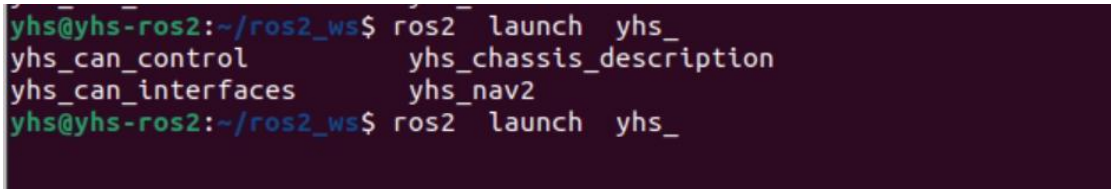
（2）After entering "ros2 launch yhs_nav2 yhs_nav2." and pressing the "Tab" key, you may need to press it several times. Two options will appear because there are two files starting with "yhs_nav2.". If there are more files, they will all be displayed. Therefore, we should try to input more information so that the options displayed after pressing the "Tab" key will be fewer. Based on the listed options, continue entering and pressing the "Tab" key for completion. If none of the listed options include the target file or target command, check if the entered command is correct, if the target file or command exists, and if the environment variables are set correctly.




（3）Please execute the command and press the "Tab" key to see the available completion options.

```
ros2 launch yhs_
```



（4）Please execute the command and press the "Tab" key to see the available completion options.(The completion options can vary depending on the specific files present in the directory. )

```
ros2  launch  yhs_nav2
```

```
yhs@yhs-ros2:~/ros2_ws$ ros2  launch  yhs_nav2
-a                              g6.yaml
c10.yaml                        gmapping.launch.py
c11.yaml                        --launch-prefix
c12.yaml                        --launch-prefix-filter
c13.yaml                        localization_launch.py
c1.yaml                         -n
c2.yaml                         navigation_launch.py
c3.yaml                         --noninteractive
c5.yaml                         -p
c6.yaml                         package.xml
c7.yaml                         --print
c8.yaml                         --print-description
cartographer.launch.py          -s
-d                              --show-all-subprocesses-output
--debug                         --show-args
g1.yaml                         --show-arguments
g3.yaml                         yhs_nav2.launch.py
g4.yaml                         yhs_nav2.yaml
g5.yaml
yhs@yhs-ros2:~/ros2_ws$ ros2  launch  yhs_nav2
```

3. Terminal Termination Command

（1）Assuming a terminal is running a LiDAR driver program, and you want to execute a mapping program command which will start the LiDAR. This will result in an error, so you need to terminate the LiDAR driver program before executing the mapping command. In the terminal where the LiDAR driver program is running, press "Ctrl + C" and wait for the program to terminate. Before moving on to the next experiment, make sure to terminate all the programs started during this experiment.

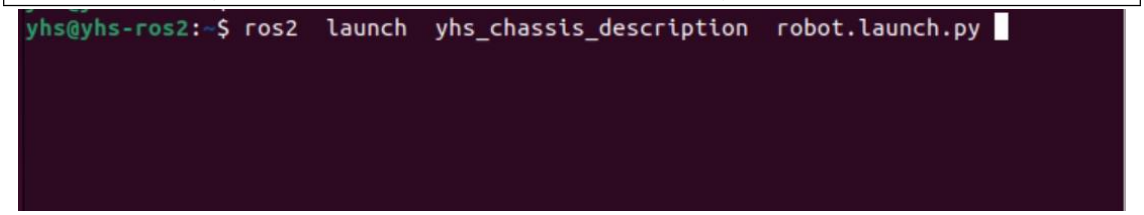
```
^C[WARNING] [launch]: user interrupted with ctrl-c (SIGINT)
[nvilidar_ros2_node-1] [INFO] [1701658937.472366697] [rclcpp]: signal_handler(signum=2)
[static_transform_publisher-2] [INFO] [1701658937.472635905] [rclcpp]: signal_handler(signum=2)
[nvilidar_ros2_node-1] [INFO] [1701658937.491504888] [nvilidar_ros2_node]: [NVILIDAR INFO] Now NVILIDAR is stopping ......
.
[INFO] [static_transform_publisher-2]: process has finished cleanly [pid 6801]
[INFO] [nvilidar_ros2_node-1]: process has finished cleanly [pid 6799]
[nvilidar_ros2_node-1]
yhs@yhs-ros2:~/ros2_ws$
```

（2）For the "rviz2" software interface, there is no need to repeatedly open and close it. You can adjust the relevant settings as needed without reopening it.

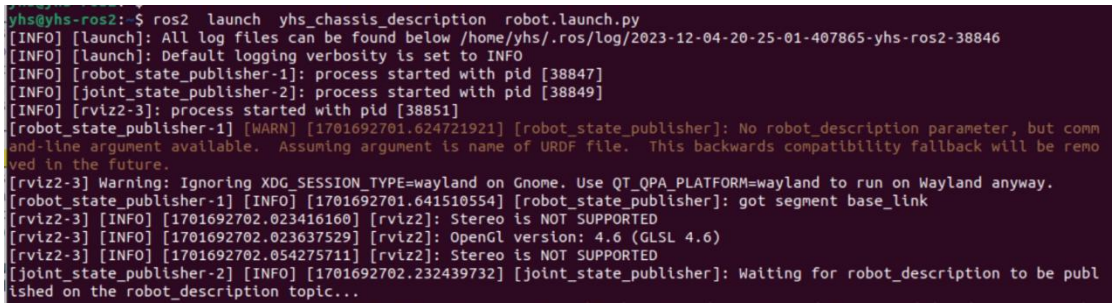
# Experiment 1: ROS2 Instruction

1. Open a terminal and enter the following command:

```
ros2 launch yhs_chassis_description robot.launch.py
```



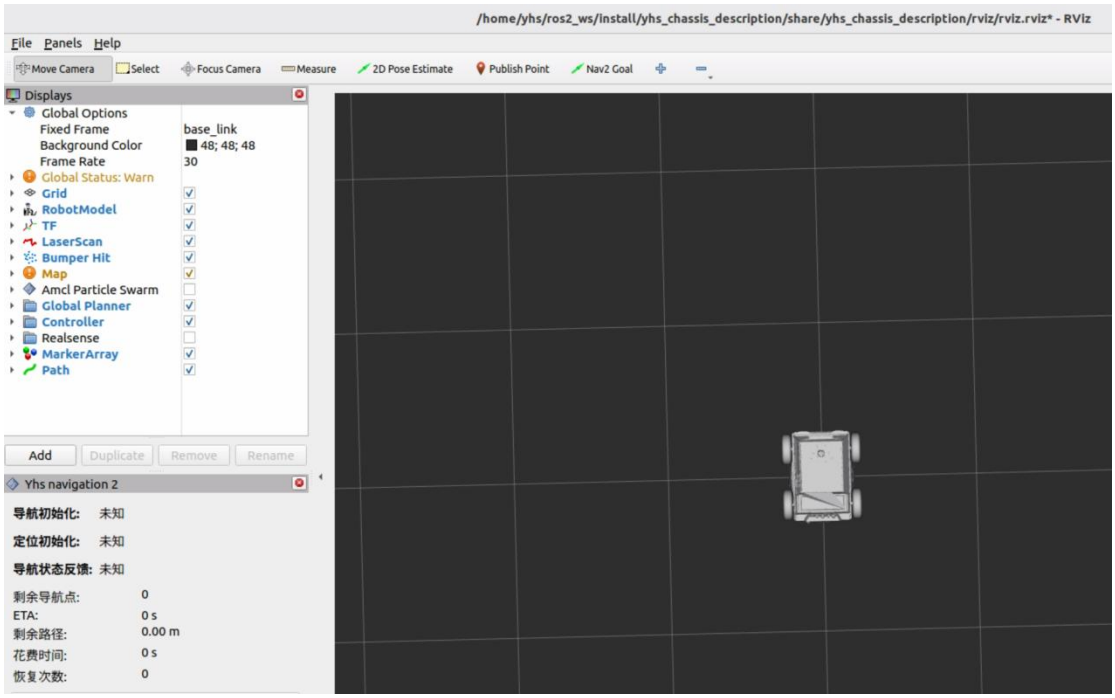

After entering the command, press Enter. At the same time, you will see an "Rviz" startup logo on the desktop. After a moment of initialization, a graphical window will appear on the desktop. This is Rviz2, the most commonly used graphical display interface in ROS2. It provides a convenient way to visualize the results generated by sensors and algorithms used in robot programming and debugging.




Usually, the main interface of Rviz2 is divided into two columns: the left column is "Displays," and the right column is the main display area called "View."

The right column, "View," represents a three-dimensional world display. By default, you can see a grid-like ground plane, and you can adjust the three-dimensional perspective by dragging with the mouse.

The left column, "Displays," is used to configure the types and quantities of information displayed in the "View."




2. Going back to the command we entered earlier (ros2 launch yhs_chassis_description robot.launch.py), it launched the "robot.launch.py" launch file from the package named "yhs_chassis_description."

A launch file is a text description file in ROS2 used to start multiple program nodes (Nodes) in batch. Now, let's take a look at the contents of the "robot.launch.py" file.

```
import os
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.conditions import IfCondition, UnlessCondition
from launch.substitutions import Command, LaunchConfiguration
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():

    pkg_dir = get_package_share_directory('yhs_chassis_description')

    urdf_model = LaunchConfiguration('urdf_model')
    rviz_config_file = LaunchConfiguration('rviz_config_file')
    use_robot_state_pub = LaunchConfiguration('use_robot_state_pub')
    use_rviz = LaunchConfiguration('use_rviz')
    use_sim_time = LaunchConfiguration('use_sim_time')

    declare_urdf_model_path_cmd = DeclareLaunchArgument(
        name='urdf_model',
        default_value=os.path.join(pkg_dir, 'urdf', 'robot.urdf'),
        description='Absolute path to robot urdf file')
```

```
declare_rviz_config_file_cmd = DeclareLaunchArgument(
    name='rviz_config_file',
    default_value=os.path.join(pkg_dir, 'rviz', 'rviz.rviz'),
    description='Full path to the RVIZ config file to use')

declare_use_robot_state_pub_cmd = DeclareLaunchArgument(
    name='use_robot_state_pub',
    default_value='True',
    description='Whether to start the robot state publisher')

declare_use_rviz_cmd = DeclareLaunchArgument(
    name='use_rviz',
    default_value='True',
    description='Whether to start RVIZ')

declare_use_sim_time_cmd = DeclareLaunchArgument(
    name='use_sim_time',
    default_value='False',
    description='Use simulation (Gazebo) clock if true')

start_robot_state_publisher_cmd = Node(
    package='robot_state_publisher',
    executable='robot_state_publisher',
    name='robot_state_publisher',
    output='screen',
    parameters=[{'use_sim_time': use_sim_time}],
    arguments=[urdf_model])
start_joint_state_publisher_cmd = Node(
    package='joint_state_publisher',
    executable='joint_state_publisher',
    name='joint_state_publisher',
    output='screen')

start_rviz_cmd = Node(
    condition=IfCondition(use_rviz),
    package='rviz2',
    executable='rviz2',
    name='rviz2',
    output='screen',
    arguments=['-d', rviz_config_file])

ld = LaunchDescription()
```

```
    ld.add_action(declare_urdf_model_path_cmd)
    ld.add_action(declare_rviz_config_file_cmd)
    ld.add_action(declare_use_robot_state_pub_cmd)
    ld.add_action(declare_use_rviz_cmd)
    ld.add_action(declare_use_sim_time_cmd)

    ld.add_action(start_robot_state_publisher_cmd)
    ld.add_action(start_joint_state_publisher_cmd)
    ld.add_action(start_rviz_cmd)

    return ld
```

This is a Python file, and let's briefly understand its contents:

(1) It imports some necessary modules and packages for starting ROS2 nodes and managing launch configurations.

```
import os
from launch import LaunchDescription
…
```

(2) It defines the "generate_launch_description" function.

```
def generate_launch_description():
```

(3) It defines launch configuration variables.

```
urdf_model = LaunchConfiguration('urdf_model')
…
```

(4) It declares launch parameters.

```
declare_urdf_model_path_cmd = DeclareLaunchArgument(
    name='urdf_model',
    default_value=os.path.join(pkg_dir, 'urdf', 'robot.urdf'),
    description='Absolute path to robot urdf file')
…
```

(5) It defines nodes. The "package" specifies the package name to locate the node program. The "executable" specifies the executable program name as declared in the CMakeLists file. The "name" specifies the name displayed in ROS2 after the node is launched (can be chosen freely). The "output" sets the output mode of the node to the screen, displaying the node's output in the terminal. The "parameters" passes parameters to the node, where the node program has a declaration for "use_sim_time." The "arguments" is similar to specifying parameters directly in the terminal using "-xxx" format, which is used internally within the node.

```
start_robot_state_publisher_cmd = Node(
    package='robot_state_publisher',
    executable='robot_state_publisher',
```

```
    name='robot_state_publisher',
    output='screen',
    parameters=[{'use_sim_time': use_sim_time}],
    arguments=[urdf_model])
…
```

(6) It creates a LaunchDescription object used to collect, organize, and describe all the configurations and behaviors for launching ROS2 nodes.

```
ld = LaunchDescription()
```

(7) It adds declaration variables, node launching actions, and other actions to the LaunchDescription object.

```
ld.add_action(declare_urdf_model_path_cmd)
…
```

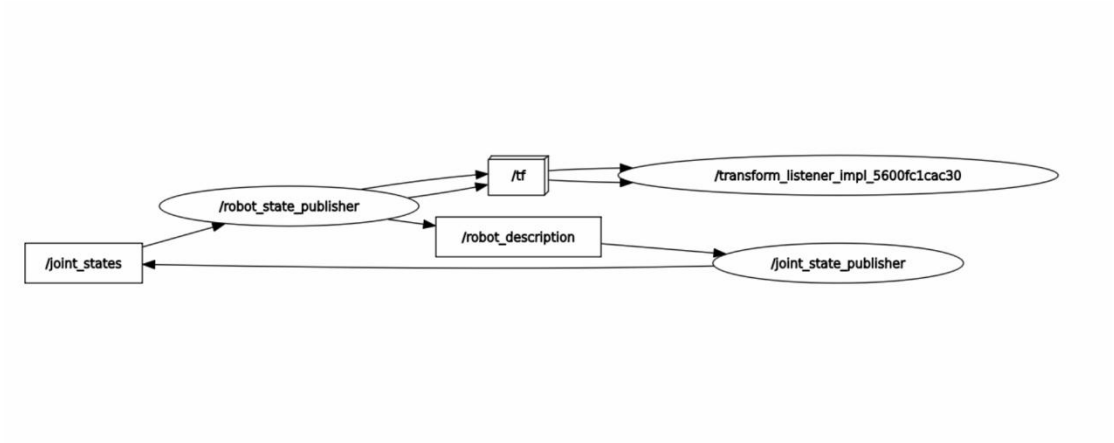
(8) It returns the LaunchDescription object, which is used to launch ROS2 nodes.

```
return ld
```

3. After reviewing the contents of the launch file, let's take a look at what happens behind the scenes. Please keep the Rviz2 interface launched by "robot.launch.py" at the beginning of this section open and do not close it. Open a new terminal and enter a new command.

```
rqt_graph
```


When you press Enter, a dialog box will pop up.



This diagram represents the currently running nodes and their data relationships. The ellipses represent nodes, the curved arrows represent data flow, and the rectangular boxes represent topics. Nodes communicate with each other through the "topics" to exchange data. In the diagram, you can see three nodes that correspond to the <node> entries in the launch file.

(1) The joint_state_publisher node reads the joint values of the robot from the robot_description parameter specified in the URDF file. It passes these values to the robot_state_publisher node through the /joint_states topic.

(2) The robot_state_publisher node converts the received joint values into TF format data and publishes it on the /tf topic.

(3) The rviz2 node (the graphical interface we see) reads data from the /tf topic and updates its displayed 3D model in real-time.

The launch file serves as the entry point for most ROS2 packages. In the early stages of learning ROS2, it can be confusing to navigate through a package with numerous directories. In such cases, you can start by exploring the launch folder and read through the launch files one by one. This will give you a general idea of which package and which node is being launched. Then, you can navigate to the corresponding package's src directory to find the CPP or PY files for that node and understand their relationships. This way, you can gradually understand the structure and functionality of the software package.

## Experiment 2: Chassis Drive and Control

Yuhesen Robotics chassis adopts the CAN communication method, and direct communication between the computer and the chassis is not possible. A USB-to-CAN tool, also known as a CAN card, is required. Based on the chassis communication protocol and the usage of the CAN card, a dedicated driver package is needed.

1. CAN card configuration: The CAN card provided with the robot should be connected to an industrial computer or a laptop running Ubuntu system. In order to use it properly, two commands need to be executed in the terminal (the industrial computer has already been set up during the boot process).

| |
|---|
| sudo ip link set can0 type can bitrate 500000 |
| sudo ifconfig can0 up |

```
yhs@yhs-ros2:~$ sudo ip link set can0 type can bitrate 500000
yhs@yhs-ros2:~$ sudo ifconfig can0 up
```

2. Reading and printing data from the CAN card: After installing the "can-utils" software package, you can use commands in the terminal to read, view, or send data.

| |
|---|
| sudo apt install can-utils |
| candump can0 |

```
yhs@yhs-ros2:~$ candump  can0
  can0  18C4D1EF   [8]  00 00 00 00 00 00 00 00
  can0  18C4D7EF   [8]  00 00 F3 FF FF FF 00 0C
  can0  18C4D8EF   [8]  00 00 01 00 00 00 00 01
  can0  18C4EAEF   [8]  E3 B0 00 31 01 00 50 33
  can0  18C4E2EF   [8]  64 01 00 40 10 FA 80 4F
  can0  18C4D1EF   [8]  00 00 00 00 00 00 10 10
  can0  18C4D7EF   [8]  00 00 F3 FF FF FF 10 1C
  can0  18C4D8EF   [8]  00 00 01 00 00 00 10 11
  can0  18C4EAEF   [8]  E3 B0 00 31 01 00 60 03
  can0  18C4DAEF   [8]  00 10 00 00 00 00 00 10
  can0  18C4E1EF   [8]  90 0A 00 00 AC 03 80 B5
  can0  18C4D1EF   [8]  00 00 00 00 00 00 20 20
  can0  18C4D7EF   [8]  00 00 F3 FF FF FF 20 2C
  can0  18C4D8EF   [8]  00 00 01 00 00 00 20 21
  can0  18C4D1EF   [8]  00 00 00 00 00 00 30 30
  can0  18C4D7EF   [8]  00 00 F3 FF FF FF 30 3C
  can0  18C4D8EF   [8]  00 00 01 00 00 00 30 31
  can0  18C4EAEF   [8]  E3 B0 00 31 01 00 70 13
  can0  18C4D1EF   [8]  00 00 00 00 00 00 40 40
  can0  18C4D7EF   [8]  00 00 F3 FF FF FF 40 4C
  can0  18C4D8EF   [8]  00 00 01 00 00 00 40 41
```

3. To read or send extended frame data, the most significant bit of the CAN address must be set to 1. Otherwise, data transmission and reception will not work properly. Therefore, in the driver program, the address prefix will change from "0x18" to "0x98". The following is an explanation about addresses from the header file "/usr/include/linux/can.h":

```
/*
 * Controller Area Network Identifier structure
 *
 * bit 0-28      : CAN identifier (11/29 bit)
 * bit 29        : error message frame flag (0 = data frame, 1 = error message)
 * bit 30        : remote transmission request flag (1 = rtr frame)
 * bit 31        : frame format flag (0 = standard 11 bit, 1 = extended 29 bit)
 */
```

4. After understanding the CAN card setup and usage, the next step is to run the chassis driver program. The driver program utilizes the CAN card to read chassis information and send commands to control the chassis. Upon entering the command and pressing Enter, you will see that the node is successfully initialized.

```
ros2 launch yhs_can_control yhs_can_control.launch.py
```

```
yhs@yhs-ros2:~$ ros2 launch yhs_can_control yhs_can_control.launch.py
[INFO] [launch]: All log files can be found below /home/yhs/.ros/log/2023-12-05-10-14-19-043983-yhs-ros2-6888
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [yhs_can_control_node-1]: process started with pid [6889]
[yhs_can_control_node-1] [INFO] [1701742459.241878420] [yhs_can_control_node]: yhs_can_control_node initialized successfully
```

If the CAN setup is not successful or if there is an abnormality in the CAN line connection, a warning will be displayed.

```
yhs@yhs-ros2:~$ ros2 launch yhs_can_control yhs_can_control.launch.py
[INFO] [launch]: All log files can be found below /home/yhs/.ros/log/2023-12-05-10-18-00-629672-yhs-ros2-7020
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [yhs_can_control_node-1]: process started with pid [7021]
[yhs_can_control_node-1] [INFO] [1701742680.789093916] [yhs_can_control_node]: yhs_can_control_node initialized successfully
[yhs_can_control_node-1] [WARN] [1701742680.799279064] [yhs_can_control_node]: Timeout waiting for CAN frame! Please check whether the can0 setting is correct,whether the can line is connected correctly, and whether the chassis is powered on.
[yhs_can_control_node-1] [WARN] [1701742680.809463221] [yhs_can_control_node]: Timeout waiting for CAN frame! Please check whether the can0 setting is correct,whether the can line is connected correctly, and whether the chassis is powered on.
[yhs_can_control_node-1] [WARN] [1701742680.819667055] [yhs_can_control_node]: Timeout waiting for CAN frame! Please check whether the can0 setting is correct,whether the can line is connected correctly, and whether the chassis is powered on.
```

5. Once the driver program is running, to view the chassis data or send commands to control the chassis, communication with the driver program is typically done through subscribing and publishing to "topics". To view all the available topics, enter the following command in another terminal:

```
ros2 topic list
```

```
yhs@yhs-ros2:~$ ros2 topic list
/chassis_info_fb
/cmd_vel
/ctrl_cmd
/free_ctrl_cmd
/imu_data
/io_cmd
/odom
/parameter_events
/rosout
/tf
yhs@yhs-ros2:~$
```

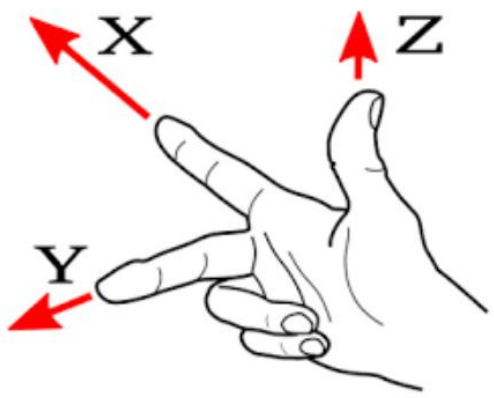
To view all the information about the chassis, you can enter the following command. Similarly, to view the data of other topics, use "ros2 topic echo /topic_name".

```
ros2 topic echo /chassis_info_fb
```

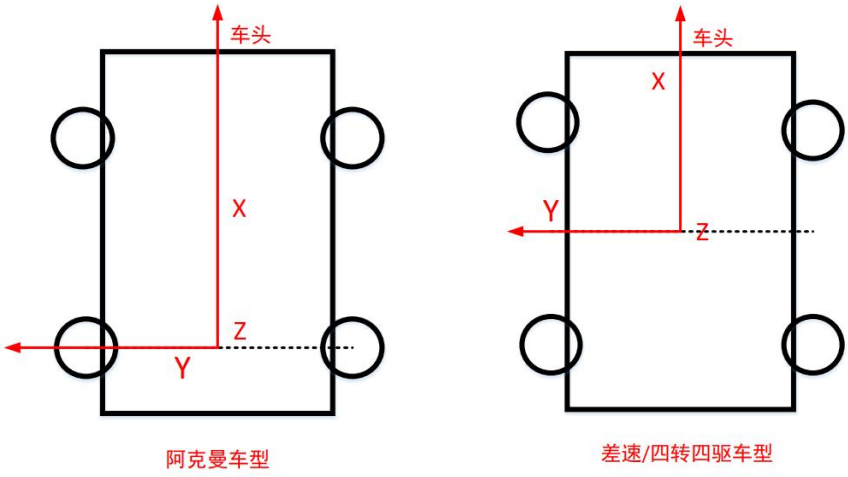
```
yhs@yhs-ros2:~$ ros2  topic  echo  /chassis_info_fb
header:
  stamp:
    sec: 1701744437
    nanosec: 891422657
  frame_id: ''
ctrl_fb:
  ctrl_fb_target_gear: 0
  ctrl_fb_linear: 0.0
  ctrl_fb_angular: 0.0
io_fb:
  io_fb_lamp_ctrl: false
  io_fb_unlock: false
  io_fb_lower_beam_headlamp: false
  io_fb_upper_beam_headlamp: false
  io_fb_turn_lamp: 0
  io_fb_braking_lamp: true
  io_fb_clearance_lamp: false
  io_fb_fog_lamp: false
  io_fb_speaker: false
  io_fb_fl_impact_sensor: false
  io_fb_fm_impact_sensor: false
  io_fb_fr_impact_sensor: false
  io_fb_rl_impact_sensor: false
  io_fb_rm_impact_sensor: false
```

6. Before controlling the movement of the chassis, let's first understand the coordinate system used in the ROS2 system:

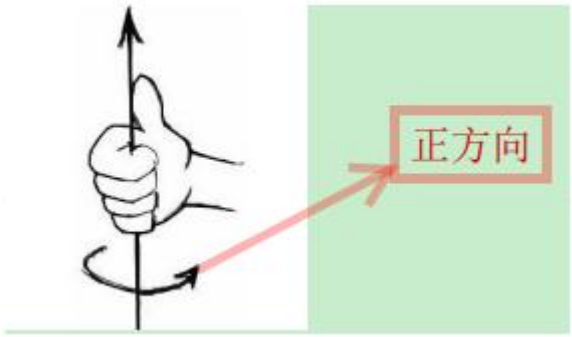
（1）The ROS2 coordinate system is defined using the right-hand rule.




On the chassis of the robot, this corresponds to the following orientation: the front of the vehicle is the positive direction of the X-axis, the left side is the positive direction of the Y-axis, and the vertical upward direction is the positive direction of the Z-axis.




（2）The rotation axis direction also follows the right-hand rule. A leftward rotation is considered the positive direction, while a rightward rotation is considered the negative direction.




7. After understanding the coordinate system of the chassis, let's proceed to send commands to control the chassis. As seen in step 5, there is a "/cmd_vel" topic through which control data for chassis movement can be sent to the driver program. Let's first examine the message type and content of the "/cmd_vel" topic.

(1) Open a terminal and enter the following command, then press Enter. This will provide detailed information about the topic, including the data type, number of publishers and subscribers, etc. Similarly, to view information about other topics, use the command "ros2 topic info /topic_name -v".

```
ros2 topic info /cmd_vel -v
```

```
yhs@yhs-ros2:~$ ros2 topic info /cmd_vel -v
Type: geometry_msgs/msg/Twist

Publisher count: 0

Subscription count: 1

Node name: yhs_can_control_node
Node namespace: /
Topic type: geometry_msgs/msg/Twist
Endpoint type: SUBSCRIPTION
GID: 01.0f.2a.fc.c6.1d.72.42.01.00.00.00.00.00.16.04.00.00.00.00.00.00.00.00
QoS profile:
  Reliability: RELIABLE
  History (Depth): UNKNOWN
  Durability: VOLATILE
  Lifespan: Infinite
  Deadline: Infinite
  Liveliness: AUTOMATIC
  Liveliness lease duration: Infinite
```

(2) To view the content of the "geometry_msgs/msg/Twist" message, enter the following command and press Enter. This will display all the contents of the message, including comments. Similarly, to view the content of other messages, use the command "ros2 interface show message_type".

| ros2 interface show geometry_msgs/msg/Twist |
|---|

```
yhs@yhs-ros2:~$ ros2 interface show geometry_msgs/msg/Twist
# This expresses velocity in free space broken into its linear and angular parts.

Vector3  linear
        float64 x
        float64 y
        float64 z
Vector3  angular
        float64 x
        float64 y
        float64 z
```

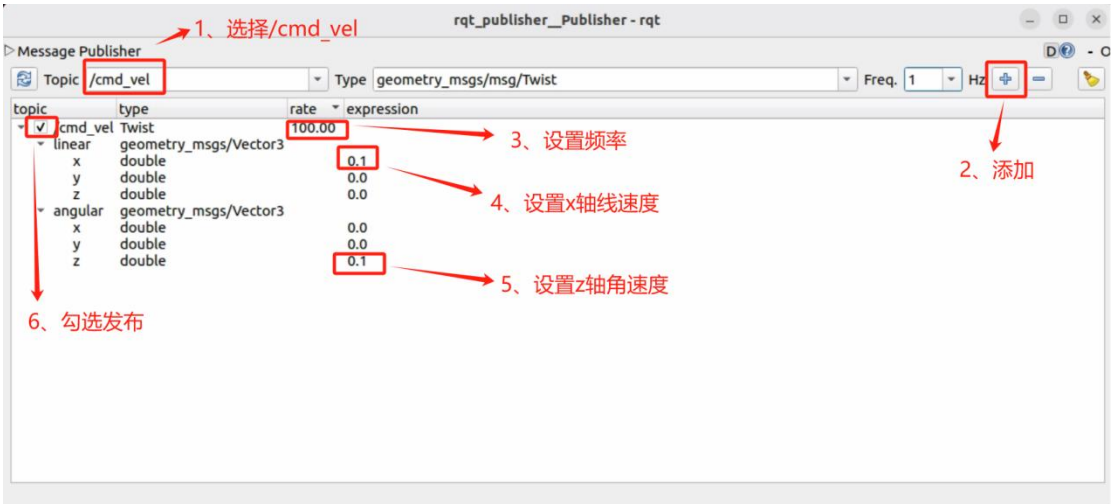
(3) The meanings of some of the message content are as follows:

| | | |
|---|---|---|
| Vector3 | linear | linear speed |
| | float64 x | The translation speed along the X-axis (m/s) |
| | float64 y | |
| | float64 z | |
| Vector3 | angular | angular speed |
| | float64 x | |
| | float64 y | |
| | float64 z | The rotation speed along the Z-axis (rad/s) (**It's only as steering angle of front wheel when the chassis is Ackermann steering type)** |

(4) To send velocity commands using the "rqt_publisher" software tool, follow these steps:

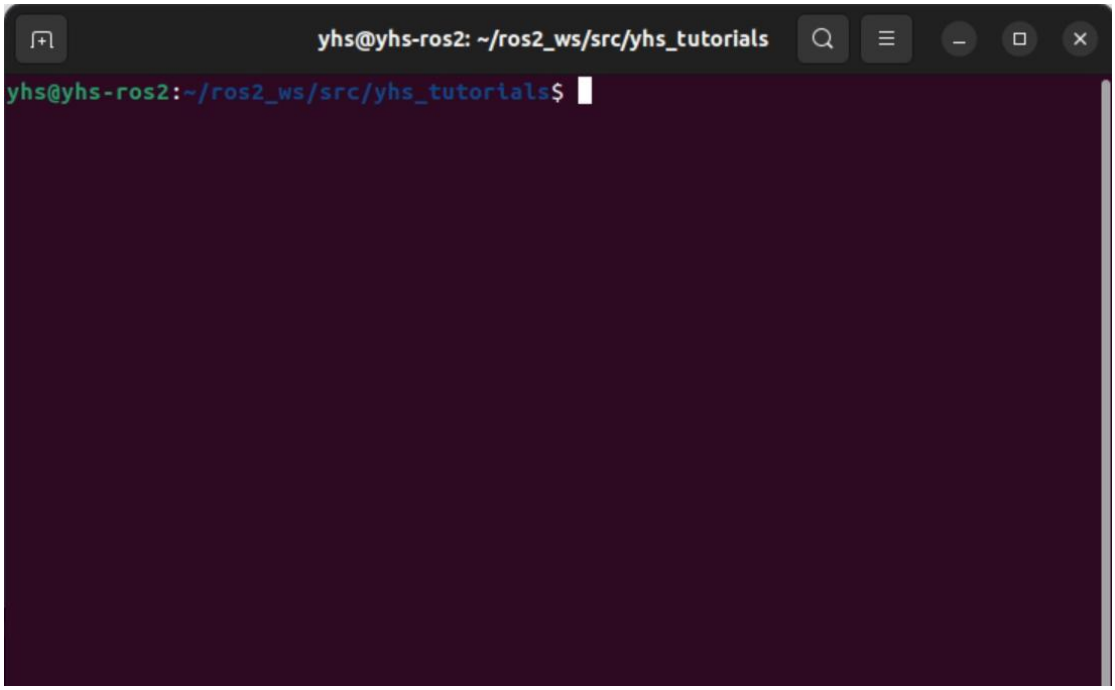
• Open a terminal and enter the "rqt_publisher" command, then press Enter.

• A software interface will open. Follow the steps shown in the image to configure the settings.



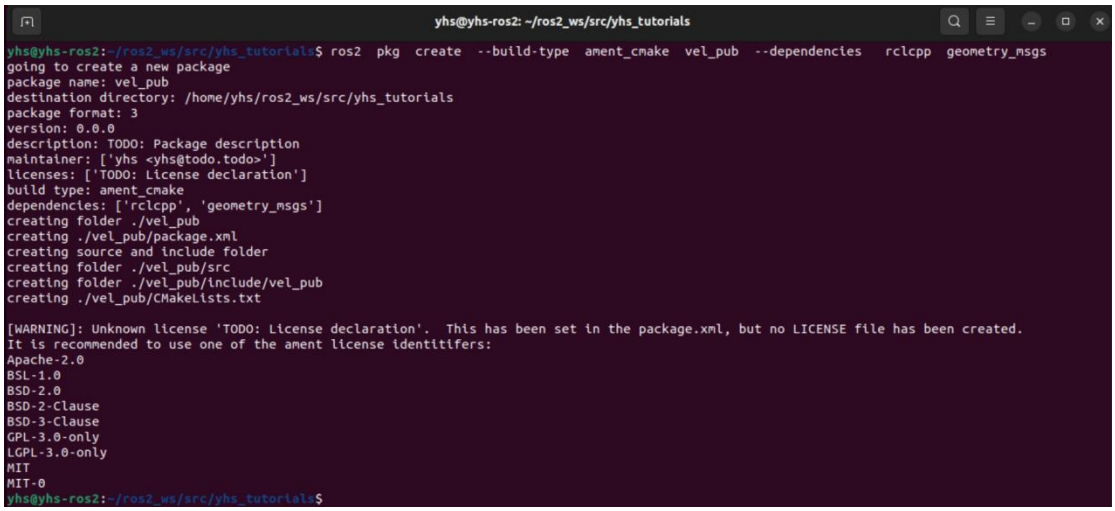

8. Sending velocity commands through a program node.

(1) Firstly, you need to create a new ROS2 package in the source code directory. Open a terminal and navigate to the directory "/home/yhs/ros2_ws/src/yhs_tutorials".



(2) Then, enter the following command to create a new ROS2 package:

```
ros2 pkg create --build-type ament_cmake vel_pub --dependencies rclcpp geometry_msgs
```

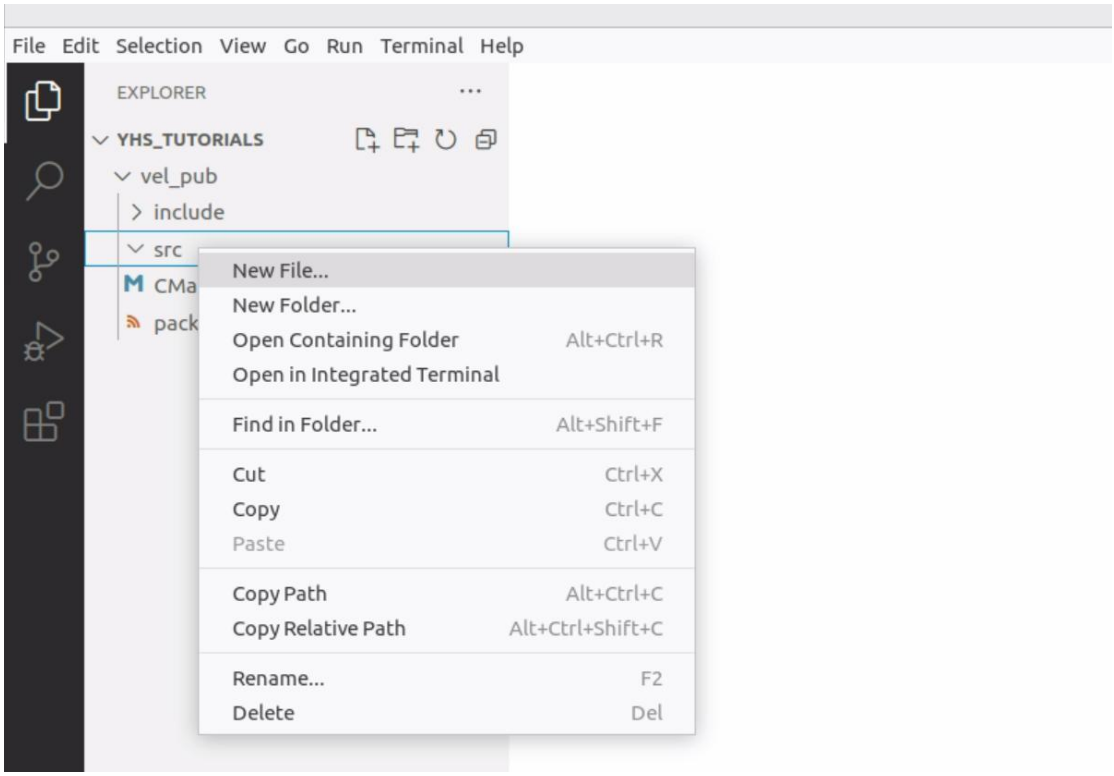



The specific meaning of this command is:

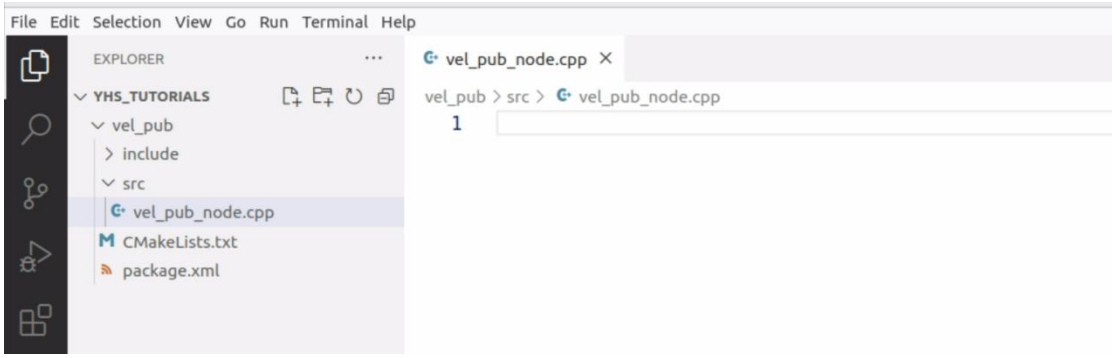
| Command | Meaning |
|---|---|
| ros2 pkg create | creating source code package of ROS2 |
| --build-type ament_cmake | creating and managing package by using ament_cmake |
| vel_pub | the name of new creating source code package of ROS2 |
| --dependencies | designated dependence |
| roscpp | C++ dependencies: This example is written in C++, so it requires C++ dependencies. |
| geometry_msgs | The package name that contains the message package format file for robot movement speed is needed. |

(3) After creating the "vel_pub" package, open the "yhs_tutorials" directory using an IDE. You will see that a new folder named "vel_pub" has been added under the "yhs_tutorials" directory. Right-click on the "src" subfolder and select "New File" to create a new code file.

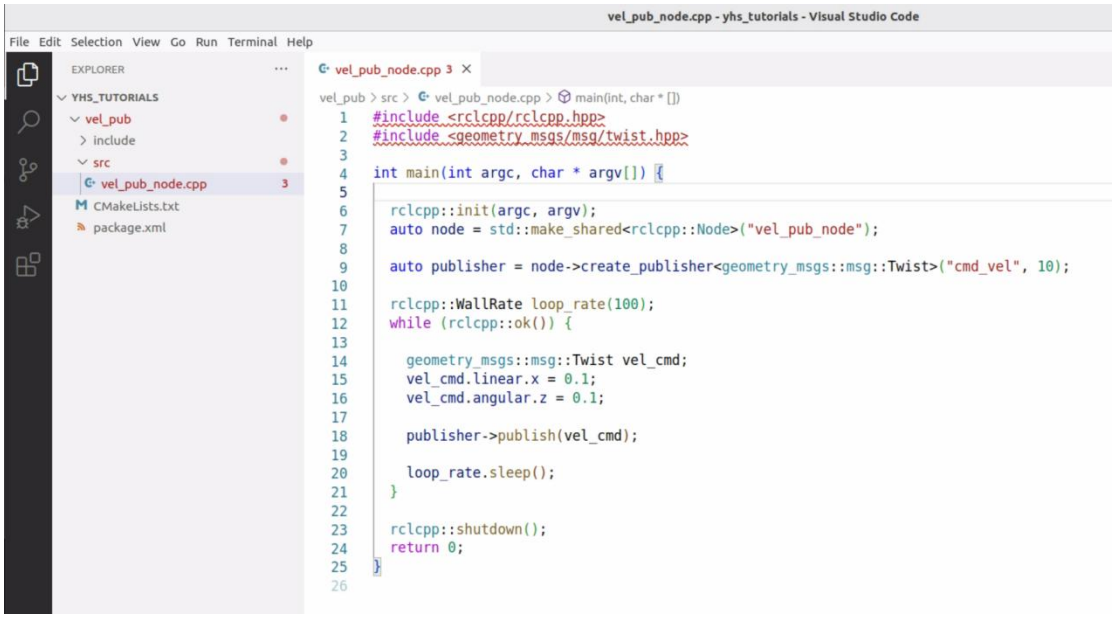
Here, we are using Visual Studio Code as an example, but you can also use other text editing tools such as Vim, Gedit, or any other of your preference to open or create the file.




(4) The newly created code file should be named "vel_pub_node.cpp".



(5) Once named, you can start writing the code for "vel_pub_node.cpp" on the right side of the IDE. Here is the
code:




```
#include <rclcpp/rclcpp.hpp>
#include "geometry_msgs/msg/twist.hpp"

int main(int argc, char * argv[]) {

   rclcpp::init(argc, argv);
   auto node = std::make_shared<rclcpp::Node>("vel_pub_node");

   auto publisher = node->create_publisher<geometry_msgs::msg::Twist>("cmd_vel", 10);

   rclcpp::WallRate loop_rate(100);
   while (rclcpp::ok()) {

      geometry_msgs::msg::Twist vel_cmd;
      vel_cmd.linear.x = 0.1;
      vel_cmd.angular.z = 0.1;

      publisher->publish(vel_cmd);

      loop_rate.sleep();
   }

   rclcpp::shutdown();
   return 0;
}
```

1) The code starts by including two header files: one for "rclcpp" and another for the definition of the motion speed structure type "geometry_msgs::msg::Twist".

2) The main function of the ROS2 node is defined as "int main(int argc, char** argv)", which follows the standard format for C++ programs.

3) Inside the main function, the code first calls "rclcpp::init(argc, argv)" to initialize the node.

4) Then, a node object named "vel_pub_node" is created and assigned to the variable "node".

5) Next, a publisher is created using the "node" object and assigned to the variable "publisher". The parameters passed to the create_publisher function specify that the publisher will broadcast data of type "geometry_msgs::msg::Twist" on the topic "/cmd_vel". This is how we control the robot's movement by broadcasting velocity commands. Here, a question arises: Why do we broadcast data on the "/cmd_vel" topic specifically? How does the robot know which topic to listen to for executing the velocity commands? The answer lies in the conventions established in ROS2. For example, the topic for publishing laser scan data is commonly named "/scan", and the topic for publishing coordinate transform relationships is usually named "/tf". Similarly, the choice of "/cmd_vel" as the topic for robot velocity control follows this convention.

6) To continuously send velocity commands, a while loop with the condition "while (rclcpp::ok())" is used. The loop will continue as long as the "rclcpp::ok()" function returns true, allowing the loop to exit gracefully when the program is shut down.

7) To send the velocity values, a variable of type "geometry_msgs::msg::Twist" named "vel_cmd" is declared, and the desired velocity values are assigned to its members.

8) "vel_cmd.linear.x" represents the linear velocity for the robot's forward/backward translation. Positive values indicate forward motion, while negative values indicate backward motion. The unit is "m/s".

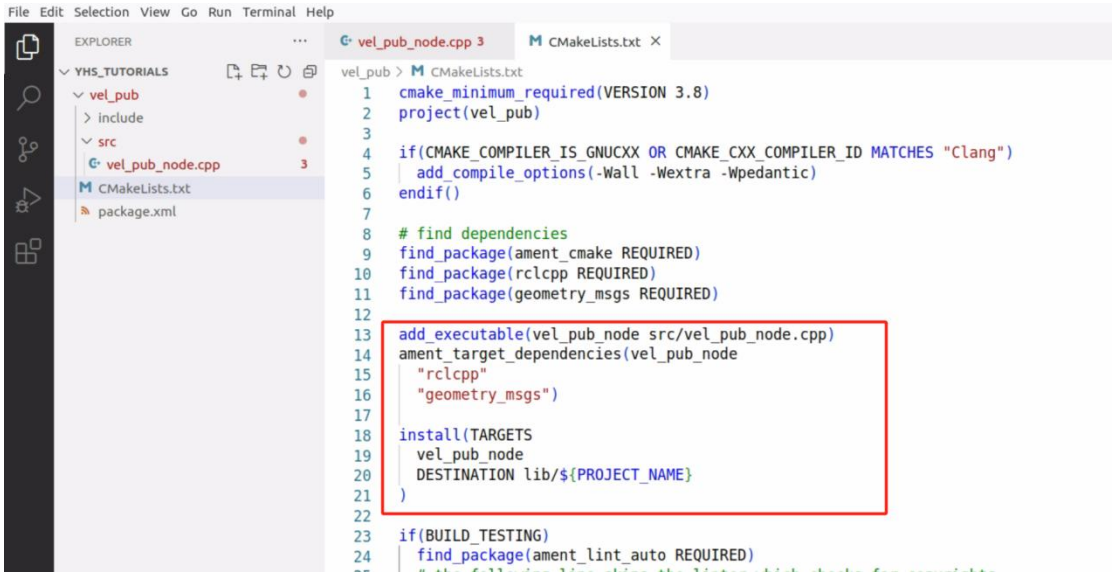
9) "vel_cmd.angular.z" (note the "angular" part) represents the angular velocity for the robot's rotational motion. Positive values indicate left turns, while negative values indicate right turns. The unit is "rad/s".

10) Once "vel_cmd" is assigned with the desired values, it is published to the "/cmd_vel" topic using the "publisher" object. The robot's chassis control node will receive the velocity values from this topic and forward them to the chassis for execution, typically via a CAN bus.

(6) Once the code is written, you need to add the file name to the compilation file for the compilation process.

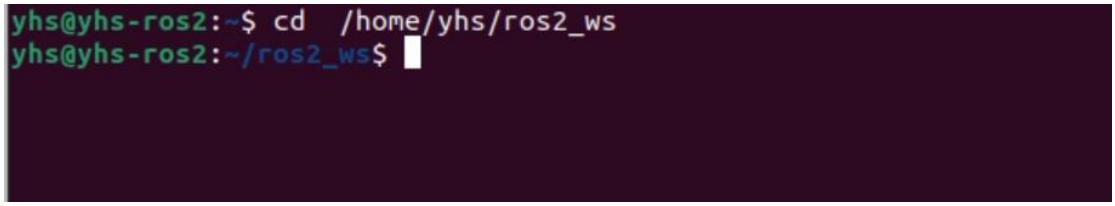
The compilation file is located in the "vel_pub" directory and named "CMakeLists.txt". On the IDE interface, click on that file on the left side, and the file content will be displayed on the right side. In the middle of the "CMakeLists.txt" file, add a new compilation rule for "vel_pub_node.cpp". The content should be as follows:

```
add_executable(vel_pub_node src/vel_pub_node.cpp)
ament_target_dependencies(vel_pub_node
  "rclcpp"
  "geometry_msgs")

install(TARGETS
  vel_pub_node
  DESTINATION lib/${PROJECT_NAME}
)
```




(7) Similarly, after making the modifications, press the keyboard shortcut Ctrl+S to save the changes. The small black dot on the right side of the file name at the top of the code will turn into an "X", indicating that the file has been successfully saved. Now, let's proceed with the compilation of the code file. Start a terminal program and enter the following command to navigate to the ROS2 workspace:

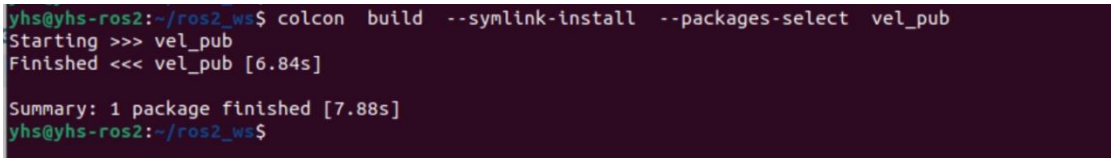
```
cd  /home/yhs/ros2_ws
```



(8) Then, execute the following command to start the compilation:

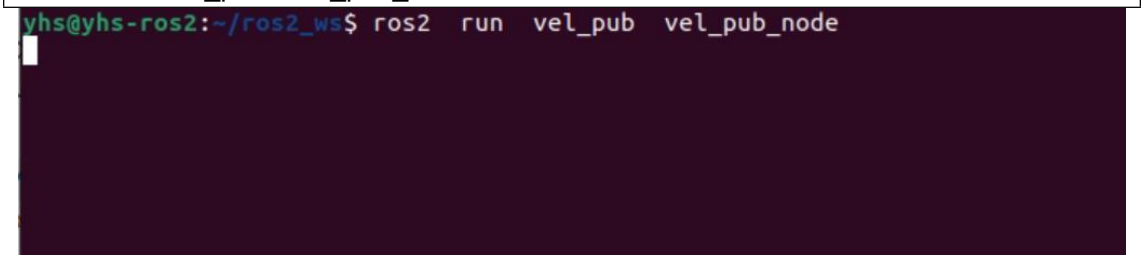
```
colcon build --symlink-install --packages-select vel_pub
```


After executing this command, you will see the scrolling compilation information. It will continue until you see the message "Summary: 1 package finished," indicating that the new vel_pub_node node has been successfully compiled.




(9) Since we have already started the chassis CAN control node earlier, we can now directly launch the vel_pub_node node. Open a terminal and enter the following command:

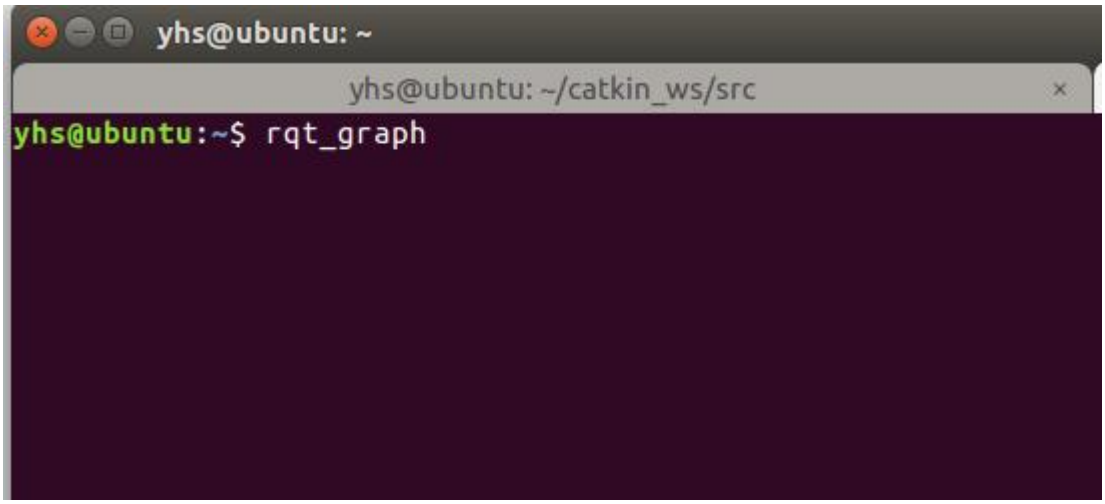
```
ros2 run vel_pub vel_pub_node
```



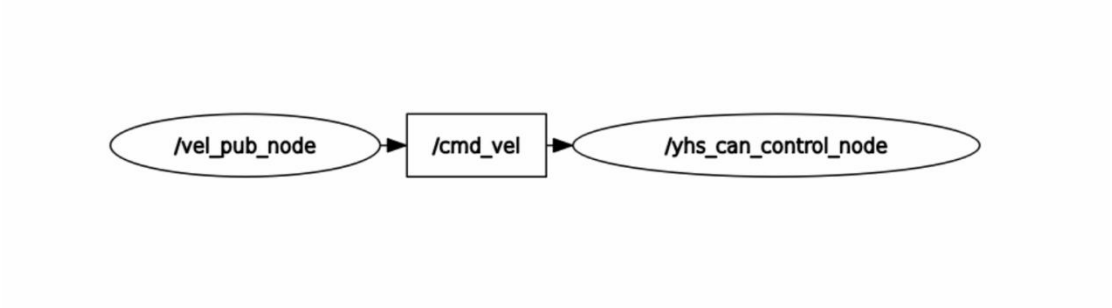

Note that here we are using the "ros2 run" command instead of the previous "ros2 launch". "ros2 run" is used to start an individual ROS2 node. After pressing Enter, you will see the robot slowly performing circular motion with a linear velocity of 0.1 m/s and an angular velocity of 0.1 rad/s.

(10) We can use a command to view the ROS2 node network status. Open a terminal and enter the following command:

```
rqt_graph
```




Pressing Enter will open a window displaying the current node network status in the ROS2 system.



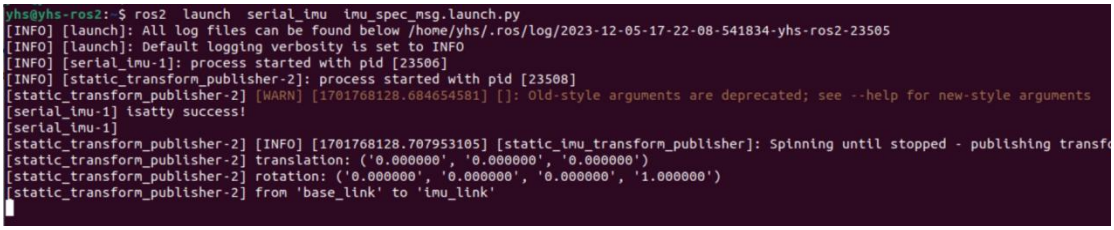

As observed, the vel_pub_node node we wrote sends velocity messages to the chassis control node via the topic "/cmd_vel". Once the yhs_can_control_node receives the velocity messages, it communicates with the chassis VCU using the CAN protocol to control the chassis movement.

## Experiment 3: IMU Data Processing

During the motion of the robot, relying solely on the data from the chassis wheel encoders to calculate the rotation angle will result in accumulating errors over time. Therefore, a high-precision IMU (Inertial Measurement Unit) is needed to provide information such as angles and accelerations.

1. To run the IMU driver package, enter the following command and press Enter:

```
ros2 launch serial_imu imu_spec_msg.launch.py
```



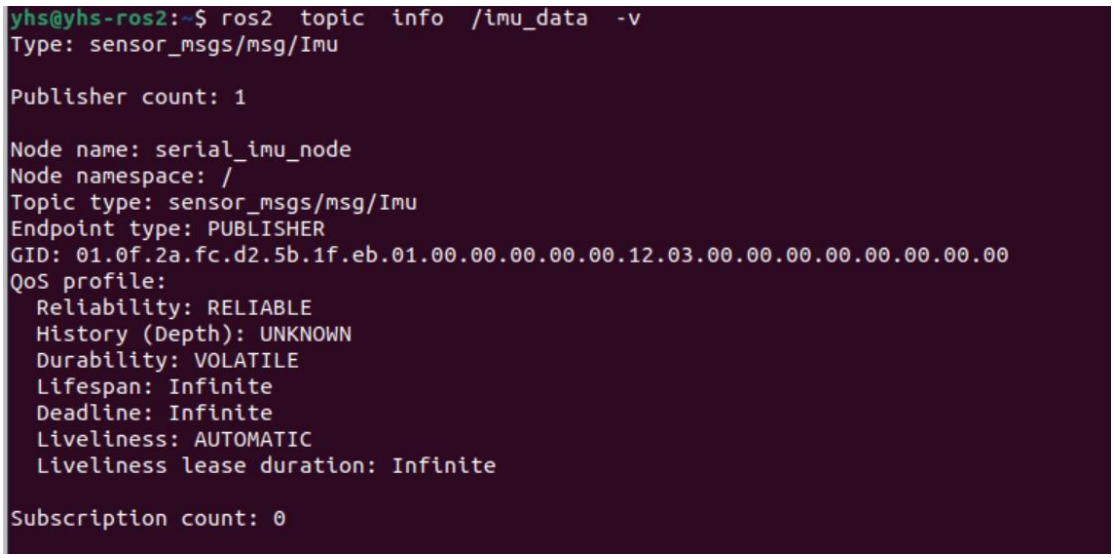

2. To check the topics published by the IMU node, enter the following command and press Enter:

```
ros2 topic list
```


3. To view information about the "/imu_data" topic:

(1) Upon executing the command, you will see that the message type for the "/imu_data" topic is "sensor_msgs/msg/Imu".

```
ros2 topic info /imu_data -v
```



(2) View information about the "sensor_msgs/msg/Imu" and entering below command

```
ros2 interface show sensor_msgs/msg/Imu
```

```
yhs@yhs-ros2:~$ ros2 interface show sensor_msgs/msg/Imu
# This is a message to hold data from an IMU (Inertial Measurement Unit)
#
# Accelerations should be in m/s^2 (not in g's), and rotational velocity should be in rad/sec
#
# If the covariance of the measurement is known, it should be filled in (if all you know is the
# variance of each measurement, e.g. from the datasheet, just put those along the diagonal)
# A covariance matrix of all zeros will be interpreted as "covariance unknown", and to use the
# data a covariance will have to be assumed or gotten from some other source
#
# If you have no estimate for one of the data elements (e.g. your IMU doesn't produce an
# orientation estimate), please set element 0 of the associated covariance matrix to -1
# If you are interpreting this message, please check for a value of -1 in the first element of each
# covariance matrix, and disregard the associated estimate.

std_msgs/Header header
        builtin_interfaces/Time stamp
                int32 sec
                uint32 nanosec
        string frame_id

geometry_msgs/Quaternion orientation
        float64 x 0
        float64 y 0
        float64 z 0
        float64 w 1
float64[9] orientation_covariance # Row major about x, y, z axes

geometry_msgs/Vector3 angular_velocity
        float64 x
        float64 y
        float64 z
float64[9] angular_velocity_covariance # Row major about x, y, z axes

geometry_msgs/Vector3 linear_acceleration
        float64 x
        float64 y
        float64 z
float64[9] linear_acceleration_covariance # Row major x, y z
```

The meanings of some of the contents are as follows:

```
std_msgs/Header header
    builtin_interfaces/Time stamp
        int32 sec
        uint32 nanosec
    string frame_id

geometry_msgs/Quaternion orientation      #Quaternion
    float64 x 0
    float64 y 0
    float64 z 0
    float64 w 1
float64[9] orientation_covariance          # Quaternion Covariance

geometry_msgs/Vector3 angular_velocity     # Angular Velocity
    float64 x
    float64 y
    float64 z
float64[9] angular_velocity_covariance     # Angular Velocity Covariance
```

```
geometry_msgs/Vector3 linear_acceleration       # Linear Velocity
      float64 x
      float64 y
      float64 z
float64[9] linear_acceleration_covariance       # Linear Velocity Covariance
```

(3) To view the data of the "/imu_data" topic when the robot is placed horizontally and stationary, enter the following command:

```
ros2 topic echo /imu_data --flow-style
```

```
yhs@yhs-ros2:~$ ros2 topic echo /imu_data --flow-style
header:
  stamp:
    sec: 1701825961
    nanosec: 160387857
  frame_id: imu_link
orientation:
  x: 0.003388338489457965
  y: 0.005457390565425157 5
  z: -4.3451738747535273e-05
  w: 0.9999794363975525
orientation_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
angular_velocity:
  x: 0.0
  y: 0.0
  z: 0.0
angular_velocity_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
linear_acceleration:
  x: -0.06860000211745501
  y: 0.05880000051110983
  z: 9.751000046730042
linear_acceleration_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
```

Executing this command will display the data being published on the "/imu_data" topic in real-time. In the case of the robot being placed horizontally and stationary, you will observe that the angular velocities on all three axes of the IMU are zero. The angular velocities on the X and Y axes may be close to zero, while the acceleration on the Z axis should be approximately 9.751 m/s².

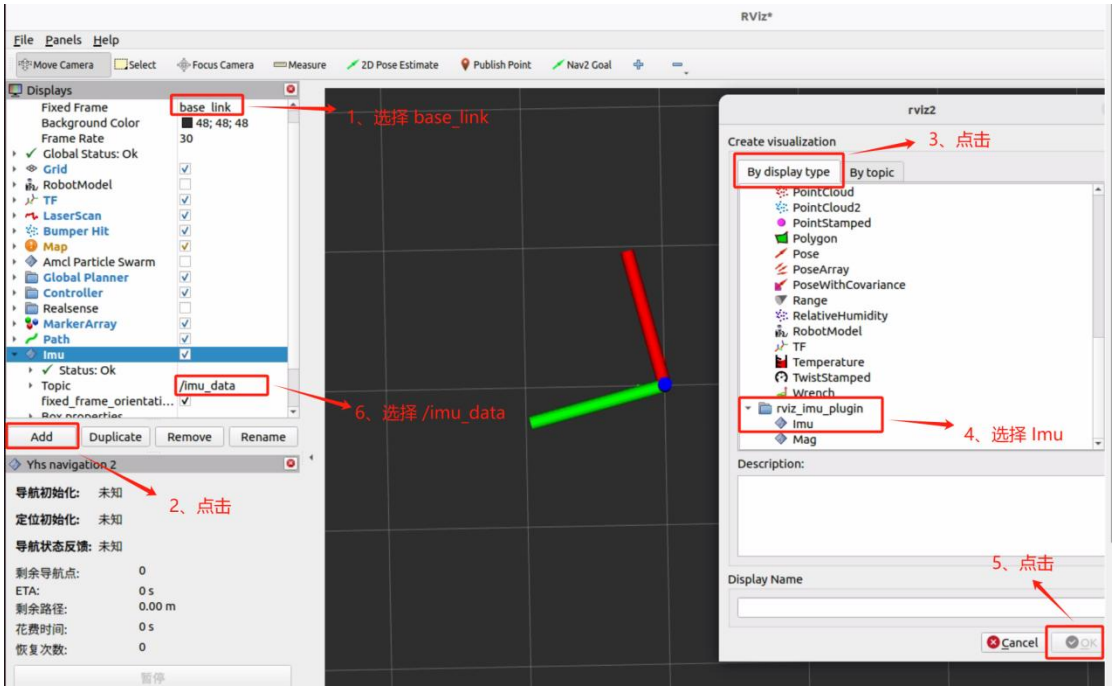
(4) To visualize the IMU data, open another terminal and enter the following command:

```
rviz2
```

```
yhs@yhs-ros2:~$ rviz2
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
[INFO] [1701826589.850440926] [rviz2]: Stereo is NOT SUPPORTED
[INFO] [1701826589.850557430] [rviz2]: OpenGl version: 4.6 (GLSL 4.6)
[INFO] [1701826589.884555243] [rviz2]: Stereo is NOT SUPPORTED
```

Press Enter to execute the command. This will launch RViz2, a visualization tool for ROS2.

In RViz2, follow the steps outlined in the image below:




1) Click on the "Add" button at the bottom-left corner of the RViz2 interface.
2) From the list of available plugins, select the "IMU" plugin and click on it.
3) Once the IMU plugin is added, you will see a coordinate frame displayed in the black square on the right side of the interface.
4) Use the scroll wheel of your mouse to zoom in or out to adjust the view.

The three axes on the black grid are represented as follows: the red axis represents the X-axis, the green axis represents the Y-axis, and the blue axis represents the Z-axis.

To assess the rotational accuracy of the IMU, you can perform the following steps:

1) Use the remote control to command the robot to rotate in place to the left by 90 degrees.
2) Observe the orientation of the three axes in RViz2. Check if the directions of the axes remain consistent with the initial setup.
3) Next, command the robot to rotate in place to the right by 90 degrees.
4) Once again, observe the orientation of the three axes in RViz2 and compare them to the initial setup.
5) By comparing the axis directions before and after the rotations, you can make a simple judgment about the rotational accuracy of the IMU.

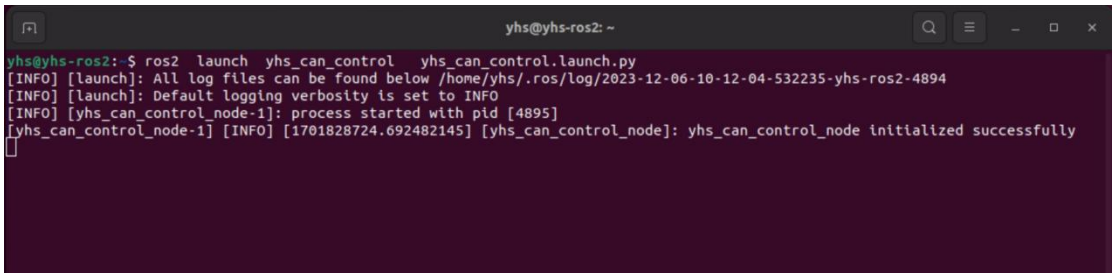
## Experiment 4: Odometry Fusion

Gmapping and Cartographer rely heavily on odometry for mapping, as the accuracy of the generated maps largely depends on the quality of odometry information. Subscribing to high-precision odometry data is crucial for building large-scale and loop-closure maps.

1. First, let's understand the message type of odometry. In the previous chassis control experiment, we observed the "/odom" topic, which publishes odometry data calculated based on the linear velocity from the chassis and the angles from the IMU.

(1) Open a terminal and run the chassis driver program.

```
ros2 launch yhs_can_control yhs_can_control.launch.py
```

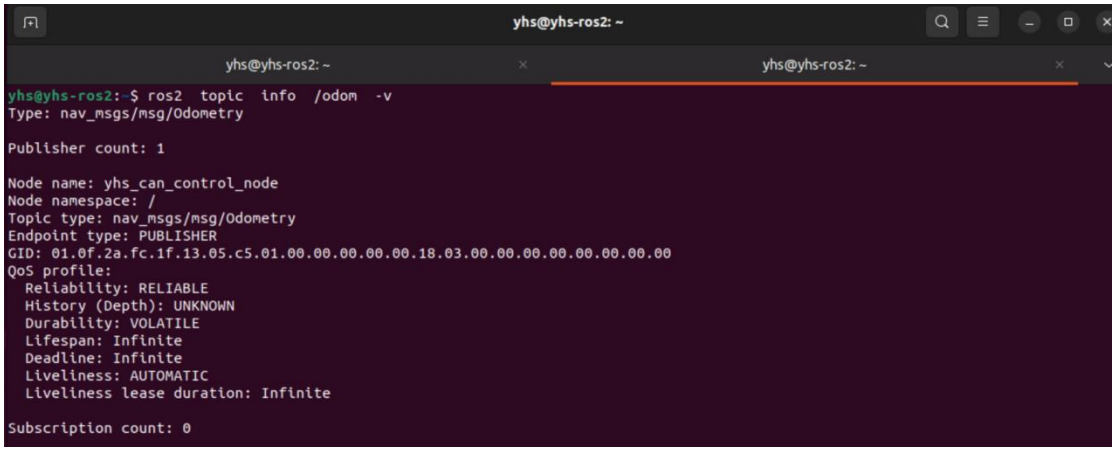



(2) Open another terminal and list all topics, view “/odom”.

```
ros2 topic list
```


(3) To check the message type of the topic, you can see that the message type is "nav_msgs/msg/Odometry".

```
ros2 topic info /odom -v
```



(3)Viewing contents of messages

```
ros2 interface show nav_msgs/msg/Odometry
```

```
yhs@yhs-ros2:~$ ros2 interface show nav_msgs/msg/Odometry
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id
# The twist in this message should be specified in the coordinate frame given by the child_frame_id

# Includes the frame id of the pose parent.
std_msgs/Header header
        builtin_interfaces/Time stamp
                int32 sec
                uint32 nanosec
        string frame_id

# Frame id the pose points to. The twist is in this coordinate frame.
string child_frame_id

# Estimated pose that is typically relative to a fixed world frame.
geometry_msgs/PoseWithCovariance pose
        Pose pose
                Point position
                        float64 x
                        float64 y
                        float64 z
                Quaternion orientation
                        float64 x 0
                        float64 y 0
                        float64 z 0
                        float64 w 1
        float64[36] covariance

# Estimated linear and angular velocity relative to child_frame_id.
geometry_msgs/TwistWithCovariance twist
        Twist twist
                Vector3  linear
                        float64 x
                        float64 y
                        float64 z
                Vector3  angular
                        float64 x
                        float64 y
                        float64 z
        float64[36] covariance
```

Partial meaning of message “nav_msgs/msg/Odometry” as below:

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by
header.frame_id
# The twist in this message should be specified in the coordinate frame given by the
child_frame_id

# Includes the frame id of the pose parent.
std_msgs/Header header
    builtin_interfaces/Time stamp
        int32 sec
        uint32 nanosec
    string frame_id

# Frame id the pose points to. The twist is in this coordinate frame.
string child_frame_id

# Estimated pose that is typically relative to a fixed world frame.
geometry_msgs/PoseWithCovariance pose
    Pose pose # The real-time coordinates of the robot only utilize the x and y axes.
```

```
        Point position
            float64 x
            float64 y
            float64 z
        Quaternion orientation              # The real-time orientation of robot
            float64 x 0
            float64 y 0
            float64 z 0
            float64 w 1
    float64[36] covariance                  # Covariance

# Estimated linear and angular velocity relative to child_frame_id.
geometry_msgs/TwistWithCovariance twist
    Twist twist
        Vector3  linear
            float64 x                       # The real-time linear velocity of robot
            float64 y
            float64 z
        Vector3  angular                    # The real-time angular velocity of robot
            float64 x
            float64 y
            float64 z
    float64[36] covariance                  # Covariance
```

(5) When debugging mapping or navigation, you may need to view the "odom" data. Enter the following command and press Enter, making sure to add "--flow-style" to prevent the covariance data from being displayed on multiple lines, thus allowing you to see the complete data.

```
ros2  topic  echo  /odom  --flow-style
```

```
yhs@yhs-ros2:~$ ros2  topic  echo  /odom  --flow-style
header:
  stamp:
    sec: 1701834241
    nanosec: 938273605
  frame_id: odom
child_frame_id: base_link
pose:
  pose:
    position:
      x: 0.0
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

2. Now that we know the message type of the odometry topic "odom," let's publish data to this topic using a program. Below is the function code for publishing odometry in the "/home/yhs/ros2_ws/src/yhs_can_control/src/yhs_can_control_node.cpp" file. In this code, we directly use the heading angle from the IMU as the robot's heading angle. Therefore, it is crucial to have high accuracy in the IMU's heading angle.

```
//transsmission real-time linear and angular velocities of the robot
void CanControl::publish_odom(const double linear_vel, const double angular_vel)
{
    static double x_ = 0.0;
    static double y_ = 0.0;
    static double theta_ = 0.0;
    static rclcpp::Time last_time_ = node_->now();

    rclcpp::Time current_time = node_->now();

    double dt = (current_time - last_time_).seconds();

    //calculating the real-time coordinate of X and Y axes
    x_ += linear_vel * cos(theta_) * dt;
    y_ += linear_vel * sin(theta_) * dt;

    auto publisher_count = node_->count_publishers("/imu_data");

    //If you cannot subscribe to IMU data, use the robot's angular velocity to calculate the robot's heading angle. Otherwise, treat the IMU's heading angle directly as the robot's
```

```
heading angle
      if(publisher_count == 0) {
        theta_ += angular_vel * dt;
      }
      else {
        std::lock_guard<std::mutex> lock(mutex_);
        theta_ = imu_yaw_;
        RCLCPP_INFO_ONCE(rclcpp::get_logger("yhs_can_control_node"), "Get
imu_data");
      }

      nav_msgs::msg::Odometry odom_msg;
      odom_msg.header.stamp = current_time;
      odom_msg.header.frame_id = "odom";
      odom_msg.child_frame_id = "base_link";

      geometry_msgs::msg::PoseWithCovariance pose_cov;
      pose_cov.pose.position.x = x_;
      pose_cov.pose.position.y = y_;
      pose_cov.pose.position.z = 0.0;
      tf2::Quaternion quat;
      quat.setRPY(0.0, 0.0, theta_);
      pose_cov.pose.orientation.x = quat.x();
      pose_cov.pose.orientation.y = quat.y();
      pose_cov.pose.orientation.z = quat.z();
      pose_cov.pose.orientation.w = quat.w();
      odom_msg.pose = pose_cov;

      geometry_msgs::msg::TwistWithCovariance twist_cov;
      twist_cov.twist.linear.x = linear_vel;
      twist_cov.twist.linear.y = 0.0;
      twist_cov.twist.linear.z = 0.0;
      twist_cov.twist.angular.x = 0.0;
      twist_cov.twist.angular.y = 0.0;
      twist_cov.twist.angular.z = angular_vel;
      odom_msg.twist = twist_cov;

      //publish “odom” to tf of “base_link”
      geometry_msgs::msg::TransformStamped odom_tf;
      odom_tf.header.stamp = node_->now();
      odom_tf.header.frame_id = "odom";
      odom_tf.child_frame_id = "base_link";
      odom_tf.transform.translation.x = odom_msg.pose.pose.position.x;
      odom_tf.transform.translation.y = odom_msg.pose.pose.position.y;
```

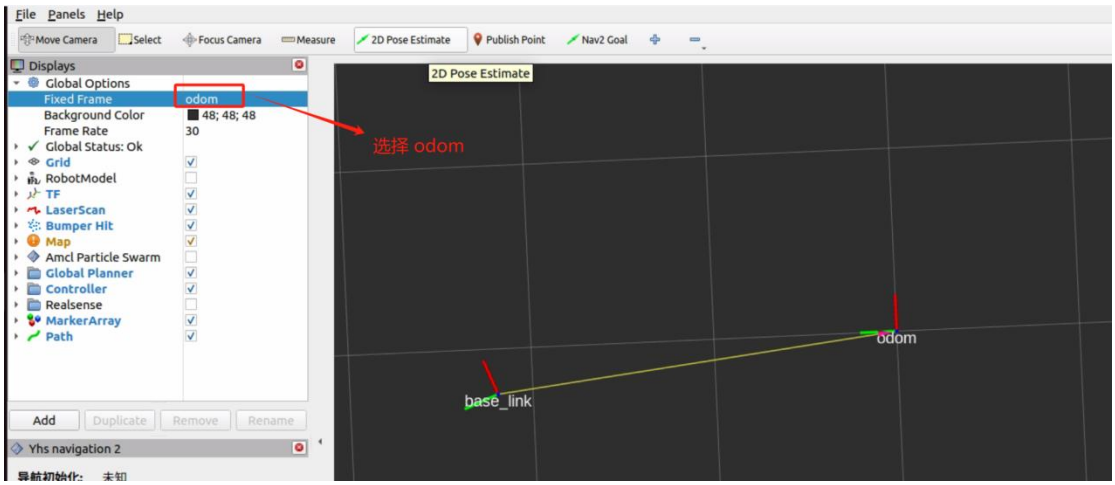
```
    odom_tf.transform.translation.z = odom_msg.pose.pose.position.z;
    odom_tf.transform.rotation = odom_msg.pose.pose.orientation;

    if(use_tf_)
      tf_broadcaster_->sendTransform(odom_tf);

    odom_pub_->publish(odom_msg);

    last_time_ = current_time;
}
```

3. Comparative testing: Comparing the accuracy of odometry data in the "odom" topic using the IMU heading angle and without using the heading angle.

(1) Without using the IMU heading angle, calculate the robot's heading angle using the robot's angular velocity. Do not run the IMU node and control the base's movement using a remote controller. During the process, try to rotate the robot as much as possible. After a certain period, return to the starting point. By observing the position and orientation of the robot (base_link) relative to the starting point (odom), it can be noticed that there is a significant difference in distance and direction between "base_link" and "odom" in rviz2 after returning to the starting point.

rviz2

```
yhs@yhs-ros2:~$ rviz2
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
[INFO] [1701843113.312435630] [rviz2]: Stereo is NOT SUPPORTED
[INFO] [1701843113.312523682] [rviz2]: OpenGl version: 4.6 (GLSL 4.6)
[INFO] [1701843113.348250527] [rviz2]: Stereo is NOT SUPPORTED
```

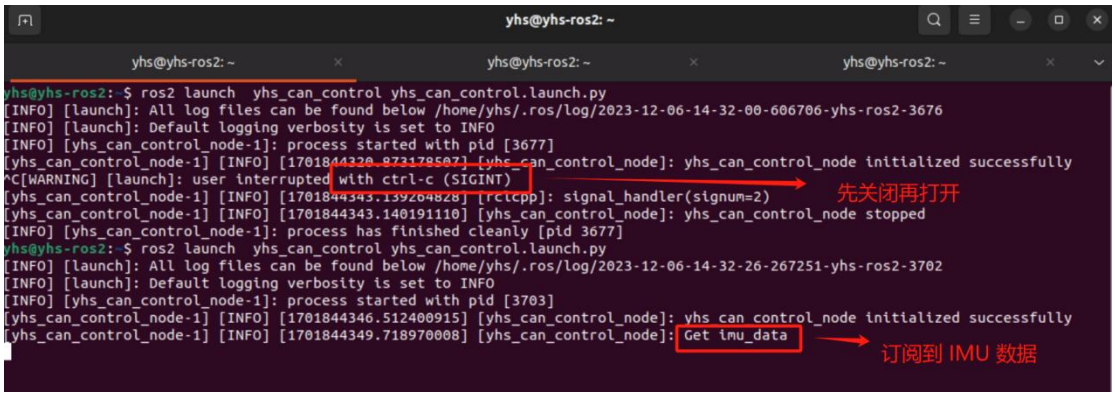


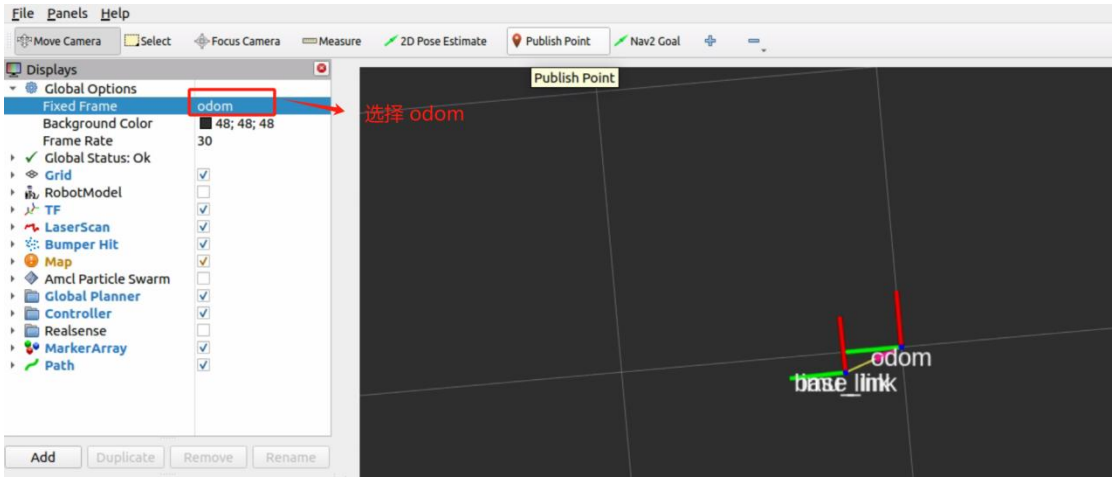

(2) Considering the IMU heading angle as the robot's heading angle, first, shut down the base control node and then run the IMU node. Control the base's movement using a remote controller. During the process, try to rotate the robot as much as possible. After a certain period, return to the starting point. By observing the position and orientation of the robot (base_link) relative to the starting point (odom), it can be noticed that there is a minimal difference in direction between "base_link" and "odom" in rviz2 after returning to

the starting point. There might be slight deviations in the X and Y axis positions, but it can be used for mapping purposes with gmapping and cartographer.

| ros2 launch serial_imu imu_spec_msg.launch.py |
|---|

```
yhs@yhs-ros2:~$ ros2 launch serial_imu imu_spec_msg.launch.py
[INFO] [launch]: All log files can be found below /home/yhs/.ros/log/2023-12-06-14-22-09-081742-yhs-ros2-3350
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [serial_imu-1]: process started with pid [3351]
[INFO] [static_transform_publisher-2]: process started with pid [3353]
[static_transform_publisher-2] [WARN] [1701843729.360508850] []: Old-style arguments are deprecated; see --help for n
ew-style arguments
[serial_imu-1] isatty success!
[serial_imu-1]
[static_transform_publisher-2] [INFO] [1701843729.423529699] [static_imu_transform_publisher]: Spinning until stopped
 - publishing transform
[static_transform_publisher-2] translation: ('0.000000', '0.000000', '0.000000')
[static_transform_publisher-2] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-2] from 'base_link' to 'imu_link'
```



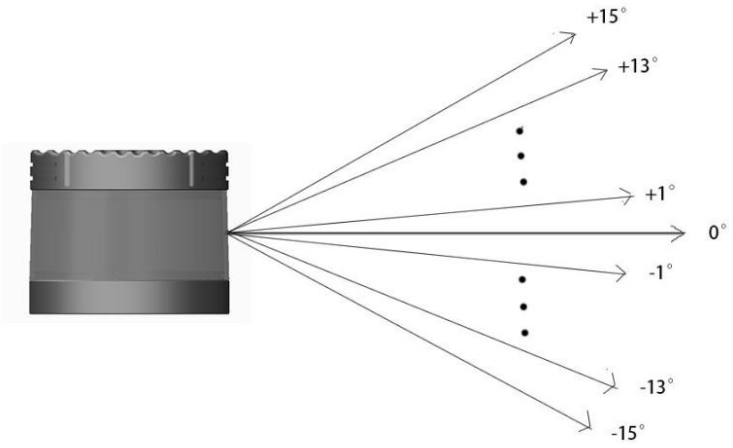




4. Here, the IMU heading angle is directly used as the robot's heading angle. Additionally, you can use the "robot_localization" package to fuse and calculate the odometry data. If you are interested, you can refer to the documentation or resources related to "robot_localization" for more information.

“https://navigation.ros.org/setup_guides/odom/setup_odom.html”。

# Experiment 5: 3D LiDAR

Lidar, specifically the 16-line Lidar, is a commonly used sensor in ground mobile robots. We utilize the pulsed time-of-flight (Pulsed ToF) method for distance measurements. The 16-line Lidar is equipped with 16 pairs of laser emitter-receiver modules. The motor driving these modules typically rotates at a speed of 10 Hz to enable a 360-degree scan.



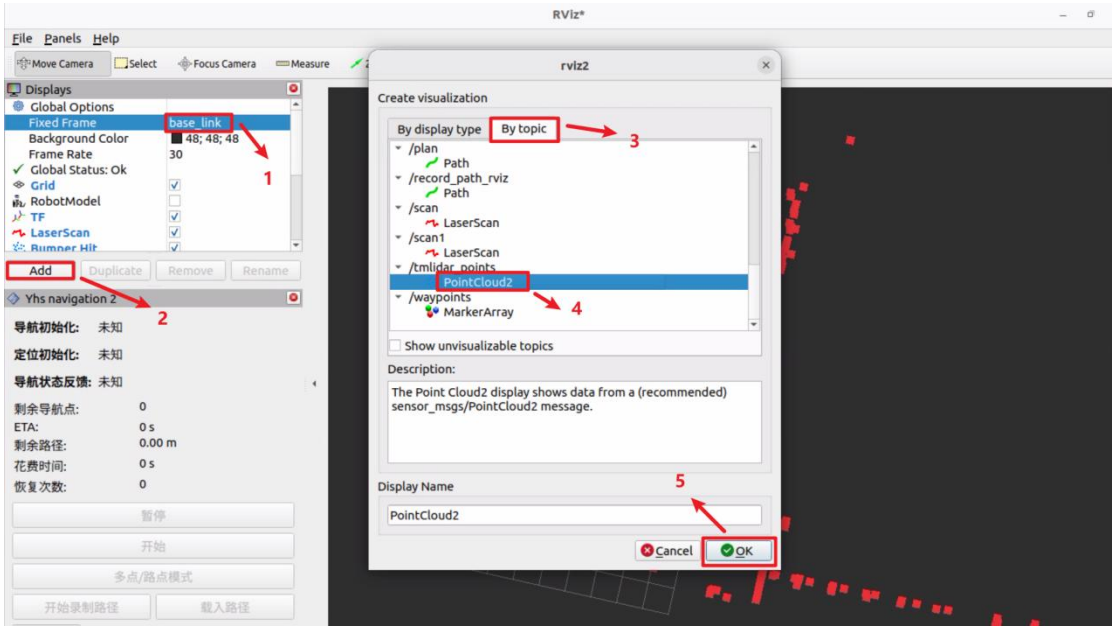

1. First, let's get a rough experience of the sensor characteristics of the lidar through a simple experiment.

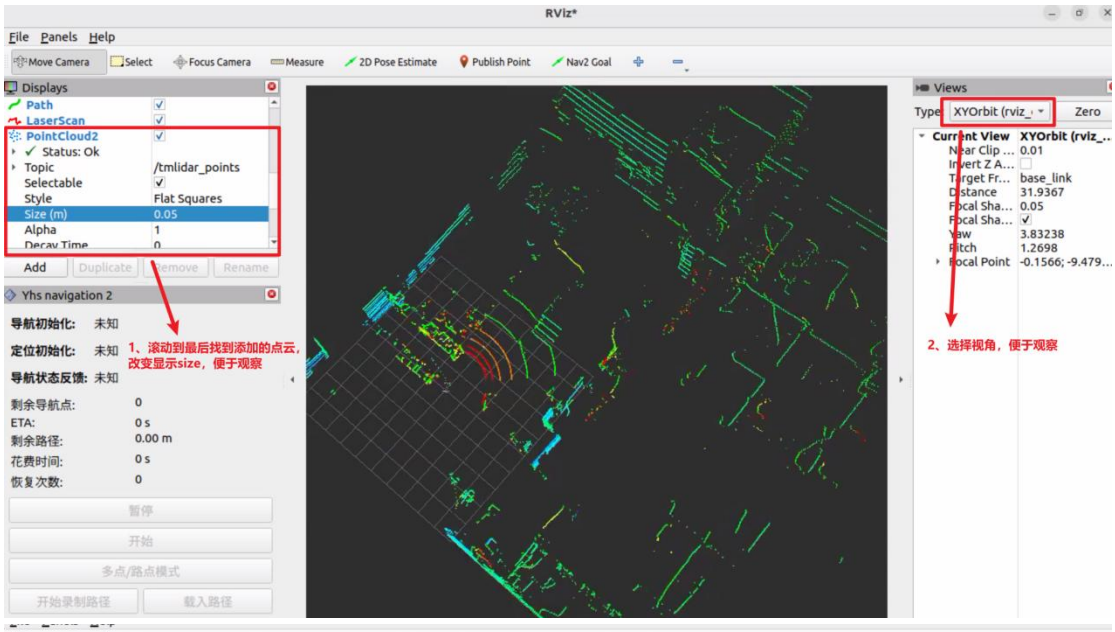
(1) Open a terminal and enter the command, then press Enter to start the lidar node.

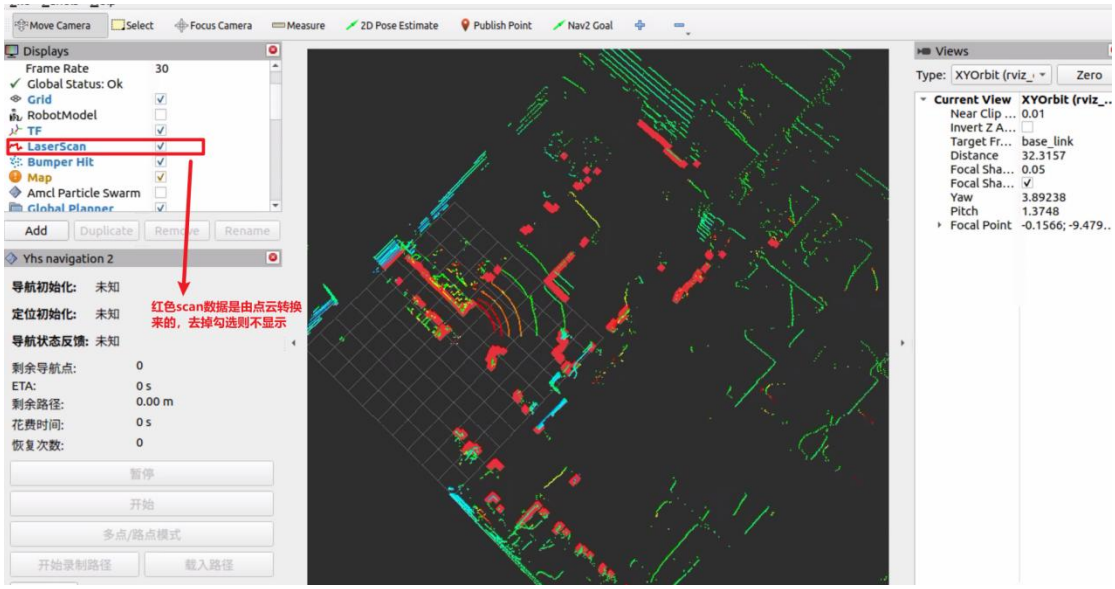
```
ros2 launch tmlidar_sdk start.launch.py
```

```
yhs@yhs-ros2:~$ ros2 launch tmlidar_sdk start.launch.py
[INFO] [launch]: All log files can be found below /home/yhs/.ros/log/2024-02-21-16-20-15-298789-yhs-ros2-27846
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [tmlidar_sdk_node-1]: process started with pid [27847]
[INFO] [pointcloud_to_laserscan_node-2]: process started with pid [27849]
[INFO] [static_transform_publisher-3]: process started with pid [27851]
[tmlidar_sdk_node-1] ******************************************************
[tmlidar_sdk_node-1] **********    TIMOOTECH LIDAR SDK Version: v1.3.0     **********
[tmlidar_sdk_node-1] ******************************************************
[static_transform_publisher-3] [WARN] [1708503615.440331534] []: Old-style arguments are deprecated; see --help for new-style arguments
[tmlidar_sdk_node-1] ------------------------------------------------------
[tmlidar_sdk_node-1] Receive Packets From : Online LiDAR
[tmlidar_sdk_node-1] Msop Port: 2368
[tmlidar_sdk_node-1] Difop Port: 8603
[tmlidar_sdk_node-1] ------------------------------------------------------
[tmlidar_sdk_node-1] driver_param.decoder_param.left_min_angle:0
[tmlidar_sdk_node-1] driver_param.decoder_param.left_min_angle:4500
[tmlidar_sdk_node-1] driver_param.decoder_param.left_min_angle:32500
[tmlidar_sdk_node-1] driver_param.decoder_param.left_min_angle:36000
[tmlidar_sdk_node-1] cut_start_angle:13500
[tmlidar_sdk_node-1] cut_end_angle:22500
[tmlidar_sdk_node-1] ------------------------------------------------------
[tmlidar_sdk_node-1] Send PointCloud To : ROS
[tmlidar_sdk_node-1] PointCloud Topic: /tmlidar_points
[tmlidar_sdk_node-1] ------------------------------------------------------
[static_transform_publisher-3] [INFO] [1708503615.454769241] [lidar_tf]: Spinning until stopped - publishing transform
[static_transform_publisher-3] translation: ('0.320000', '0.000000', '0.480000')
[static_transform_publisher-3] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-3] from 'base_link' to 'laser_link'
[tmlidar_sdk_node-1] Timoo-LiDAR-Driver is running.....
```

(2) Open rviz2 in another terminal. In rviz2, you will be able to see colored point clouds around the robot. These point clouds represent the obstacle data collected by the lidar. The ground reference in rviz2 is represented by a grid pattern, where each grid represents a distance of 1 meter. You can try observing the relationship between these points and the surrounding obstacles. Due to measurement errors typically present in sensors, these point clouds will exhibit small fluctuations within a narrow range. You can move around the robot and observe the changes in the lidar point clouds.

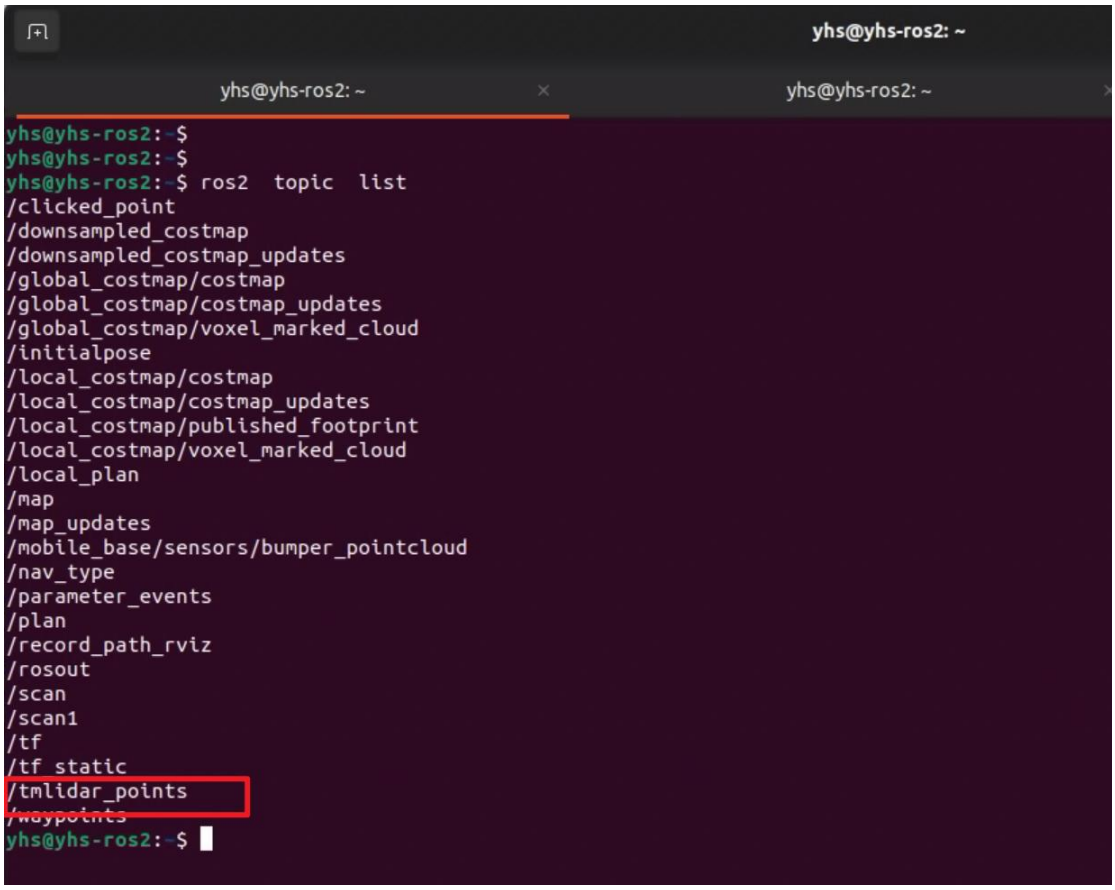

RViz*
File Panels Help
Move Camera
Select
Focus Camera
Measure
Displays
Global Options
Fixed Frame
base_link
Background Color
48; 48; 48
Frame Rate
30
Global Status: Ok
Grid
RobotModel
TF
LaserScan
Bumper Hit
Add
Duplicate
Remove
Rename
Yhs navigation 2
导航初始化: 未知
定位初始化: 未知
导航状态反馈: 未知
剩余导航点: 0
ETA: 0 s
剩余路径: 0.00 m
花费时间: 0 s
恢复次数: 0
暂停
开始
多点/路点模式
开始录制路径
载入路径
1
2
rviz2
Create visualization
By display type
By topic
3
/plan
Path
/record_path_rviz
Path
/scan
LaserScan
/scan1
LaserScan
/tmlidar_points
PointCloud2
/waypoints
MarkerArray
4
Show unvisualizable topics
Description:
The Point Cloud2 display shows data from a (recommended) sensor_msgs/PointCloud2 message.
Display Name
5
PointCloud2
Cancel
OK


RViz*
File Panels Help
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
Publish Point
Nav2 Goal
Displays
Path
LaserScan
PointCloud2
Status: Ok
Topic
/tmlidar_points
Selectable
Style
Flat Squares
Size (m)
0.05
Alpha
1
Decay Time
0
Add
Duplicate
Remove
Rename
Yhs navigation 2
导航初始化: 未知
定位初始化: 未知
导航状态反馈: 未知
剩余导航点: 0
ETA: 0 s
剩余路径: 0.00 m
花费时间: 0 s
恢复次数: 0
暂停
开始
多点/路点模式
开始录制路径
载入路径
1、滚动到最后找到添加的点云，改变显示size，便于观察
Views
Type: XYOrbit (rviz_)
Zero
Current View
XYOrbit (rviz_...
Near Clip ... 0.01
Invert Z A...
Target Fr... base_link
Distance 31.9367
Focal Sha... 0.05
Focal Sha...
Yaw 3.83238
Pitch 1.2698
Focal Point -0.1566; -9.479...
2、选择视角，便于观察


Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
Publish Point
Nav2 Goal
Displays
Frame Rate
30
Global Status: Ok
Grid
RobotModel
TF
LaserScan
Bumper Hit
Map
Amcl Particle Swarm
Global Planner
Add
Duplicate
Remove
Rename
Yhs navigation 2
导航初始化: 未知
定位初始化: 未知
导航状态反馈: 未知
剩余导航点: 0
ETA: 0 s
剩余路径: 0.00 m
花费时间: 0 s
恢复次数: 0
暂停
开始
多点/路点模式
开始录制路径
载入路径
红色scan数据是由点云转换来的，去掉勾选则不显示
Views
Type: XYOrbit (rviz_)
Zero
Current View
XYOrbit (rviz_...
Near Clip ... 0.01
Invert Z A...
Target Fr... base_link
Distance 32.3157
Focal Sha... 0.05
Focal Sha...
Yaw 3.89238
Pitch 1.3748
Focal Point -0.1566; -9.479...

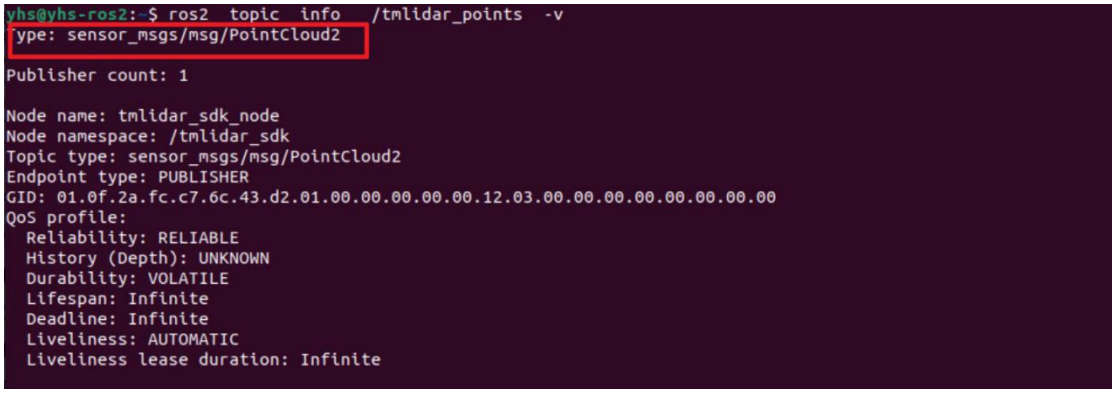
(3) To view the topic published by the lidar node, enter the command and press Enter. Locate the "/tmlidar_points" topic, which corresponds to the lidar data.

```
ros2 topic list
```




(4) To view the information about the "/tmlidar_points" topic, enter the command and press Enter. You will see that the message type for the "/tmlidar_points" topic is "sensor_msgs/msg/PointCloud2".

```
ros2 topic info /scan -v
```



(5) To view the contents of "sensor_msgs/msg/PointCloud2", enter the command and press Enter.

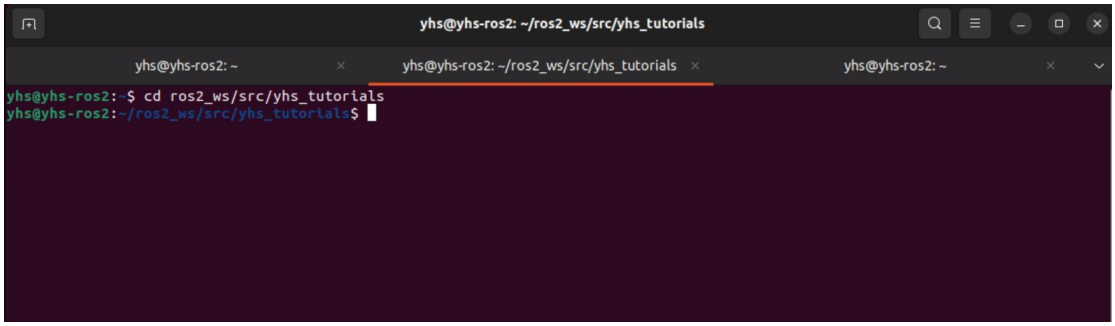
```
ros2 interface show sensor_msgs/msg/PointCloud2
```

```
yhs@yhs-ros2:~$ ros2 interface show sensor_msgs/msg/PointCloud2
# This message holds a collection of N-dimensional points, which may
# contain additional information such as normals, intensity, etc. The
# point data is stored as a binary blob, its layout described by the
# contents of the "fields" array.
#
# The point cloud data may be organized 2d (image-like) or 1d (unordered).
# Point clouds organized as 2d images may be produced by camera depth sensors
# such as stereo or time-of-flight.

# Time of sensor data acquisition, and the coordinate frame ID (for 3d points).
std_msgs/Header header
        builtin_interfaces/Time stamp
                int32 sec
                uint32 nanosec
        string frame_id

# 2D structure of the point cloud. If the cloud is unordered, height is
# 1 and width is the length of the point cloud.
uint32 height
uint32 width

# Describes the channels and their layout in the binary data blob.
PointField[] fields
        uint8 INT8    = 1
        uint8 UINT8   = 2
        uint8 INT16   = 3
        uint8 UINT16  = 4
        uint8 INT32   = 5
        uint8 UINT32  = 6
        uint8 FLOAT32 = 7
        uint8 FLOAT64 = 8
        string name      #
        uint32 offset    #
        uint8  datatype  #
        uint32 count     #

bool    is_bigendian # Is this data bigendian?
uint32  point_step   # Length of a point in bytes
uint32  row_step     # Length of a row in bytes
uint8[] data         # Actual point data, size is (row_step*height)

bool is_dense        # True if there are no invalid points
```

2. Now, let's obtain the specific measurement values from the lidar by writing code.

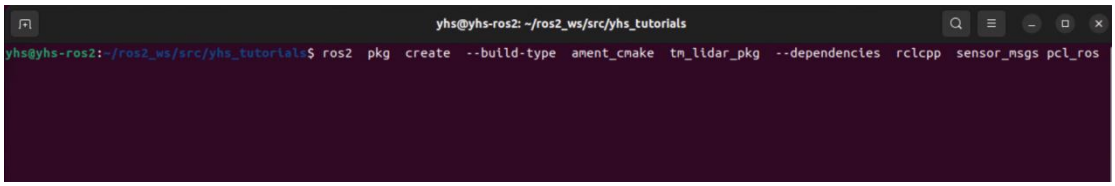
(1) Open a terminal program and enter the command to navigate to the "cd ros2_ws/src/yhs_tutorials" directory.

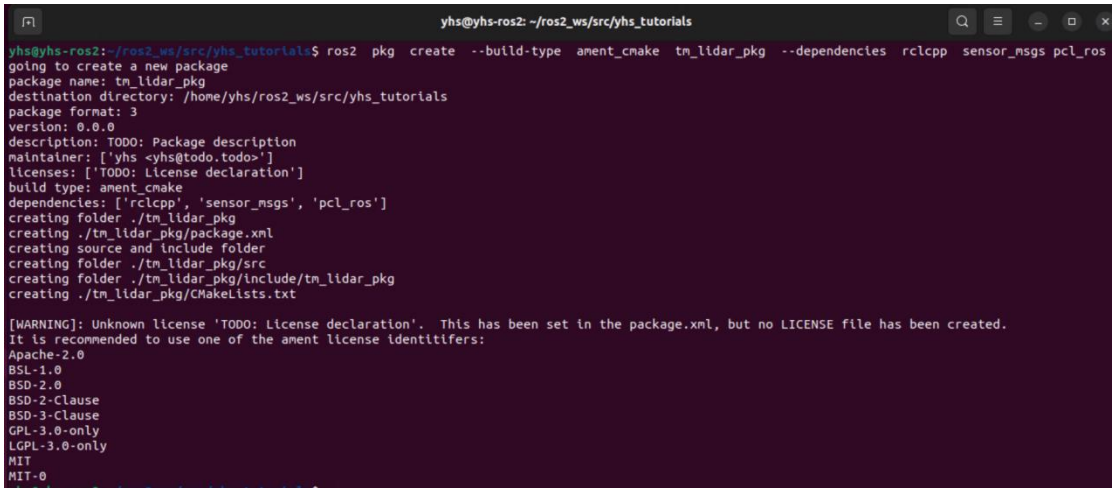
```
cd ros2_ws/src/yhs_tutorials
```



(2) Use the following command to create a new ROS2 package:

```
ros2 pkg create --build-type ament_cmake tm_lidar_pkg --dependencies rclcpp sensor_msgs pcl_ros
```

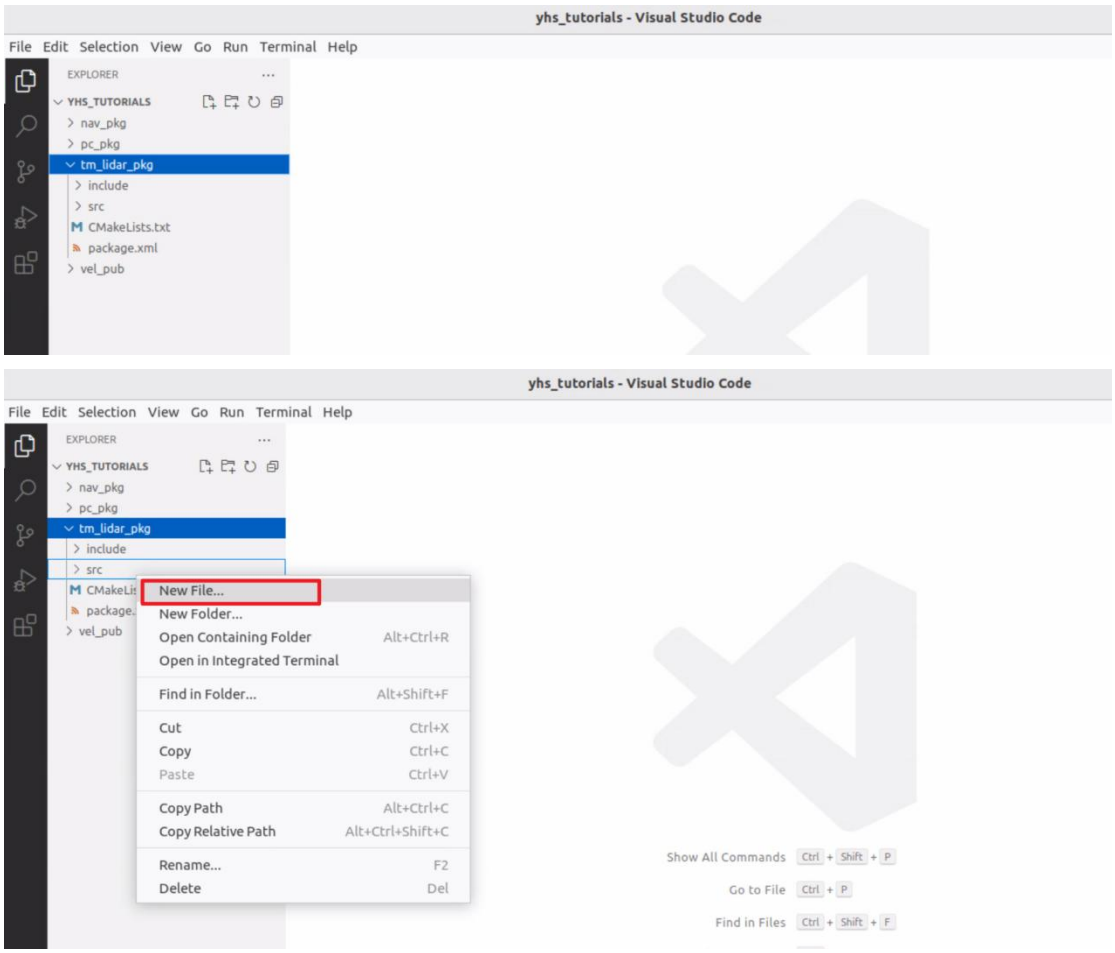


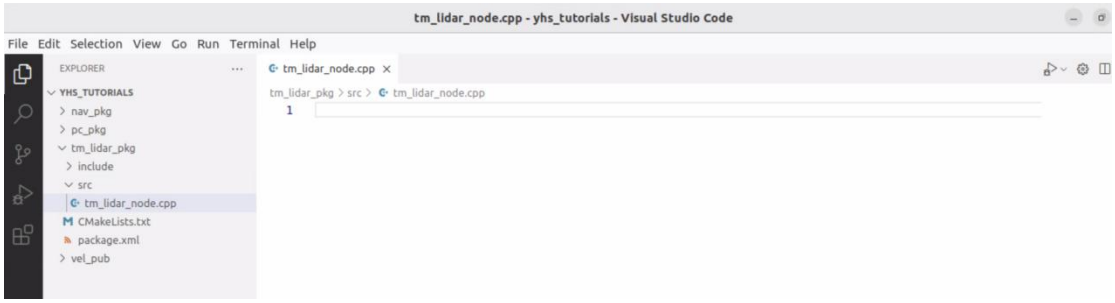

The meaning of this command

| Command | Meaning |
|---|---|
| ros2 pkg create | creating ROS2 source code package |
| --build-type ament_cmake | creating and managing package by using ament_cmake |
| tm_lidar_pkg | the name of new creating ROS2 source code package |
| --dependencies | designated dependence |
| sensor_msgs | Sensor message dependencies require the point cloud data format within them. |
| pcl_ros | PCL dependencies of open source point cloud library within ROS2 |

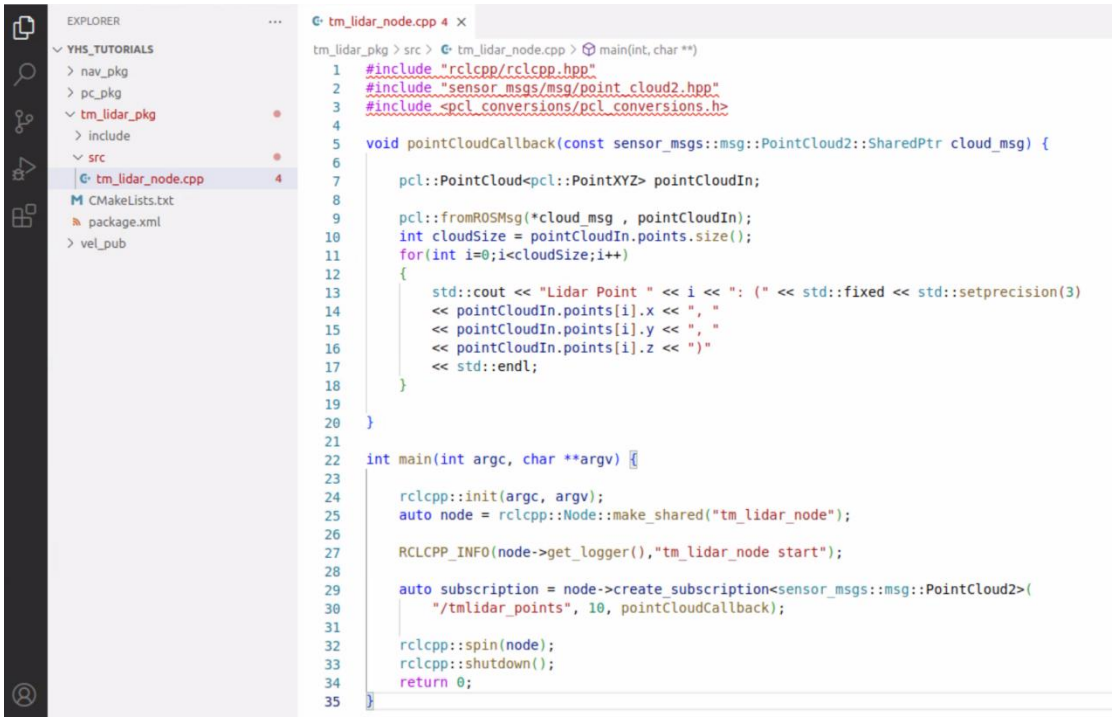
(3) After creating the "tm_lidar_pkg" package, open the "yhs_tutorials" directory in your IDE. You will notice a new folder named "tm_lidar_pkg" within it. Right-click on the "src" subfolder and select "New File" to create a new code file. (Here, we are using Visual Studio Code, but you can also use other text editing tools such as Vim or Gedit to open or create the file.)




(4) The newly created code file should be named "tm_lidar_node.cpp".




(5) Once named, you can start writing the code for "tm_lidar_node.cpp" on the right side of your IDE. The content of the file is as follows:




```
#include "rclcpp/rclcpp.hpp"
#include "sensor_msgs/msg/point_cloud2.hpp"
#include <pcl_conversions/pcl_conversions.h>

void pointCloudCallback(const sensor_msgs::msg::PointCloud2::SharedPtr cloud_msg) {

    pcl::PointCloud<pcl::PointXYZ> pointCloudIn;

    pcl::fromROSMsg(*cloud_msg , pointCloudIn);
    int cloudSize = pointCloudIn.points.size();
    for(int i=0;i<cloudSize;i++)
    {
        std::cout << "Lidar Point " << i << ": (" << std::fixed << std::setprecision(3)
        << pointCloudIn.points[i].x << ", "
        << pointCloudIn.points[i].y << ", "
        << pointCloudIn.points[i].z << ")"
        << std::endl;
    }

}

int main(int argc, char **argv) {

    rclcpp::init(argc, argv);
    auto node = rclcpp::Node::make_shared("tm_lidar_node");
```

```
    RCLCPP_INFO(node->get_logger(),"tm_lidar_node start");

    auto subscription = node->create_subscription<sensor_msgs::msg::PointCloud2>(
        "/tmlidar_points", 10, pointCloudCallback);

    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

1) The code begins by including three header files: "rclcpp.hpp" for ROS2 system, "sensor_msgs/msg/point_cloud2.hpp" for the point cloud type in ROS2, and "pcl_conversions.h" for point cloud conversion types in PCL.

2) A callback function, "void pointCloudCallback()", is defined to handle the 3D point cloud data. This callback function is automatically called by ROS2 every time a frame of 3D lidar point cloud data is received. The 3D point cloud data is passed as a parameter to this callback function.

3) The parameter "cloud_msg" of the callback function "void pointCloudCallback()" is a pointer to the memory area that holds the 3D point cloud in the "sensor_msgs::msg::PointCloud2" format. In practical development, it is common to convert this point cloud format to the PCL point cloud format, allowing the use of various PCL functions for point cloud data processing.

4) Inside the callback function "void pointCloudCallback()", a point cloud container of type "pcl::PointXYZ" named "pointCloudIn" is defined. The function "pcl::fromROSMsg()" is called to convert the point cloud data from the ROS2 format to the PCL format, which is then stored in the "pointCloudIn" container.。

5) The number of 3D points in the converted point cloud array, "pointCloudIn.points", is obtained and stored in a variable called "cloudSize". A for loop is used to display the x, y, and z values of all the points in "pointCloudIn.points" using "std::cout" in the terminal program. Typically, the raw coordinate values in "pointCloudIn.points" are not directly used but need to be transformed into the robot's coordinate system before processing them with PCL point cloud library functions.

6) In the main function, "main()", the "rclcpp::init()" function is called to initialize the node.

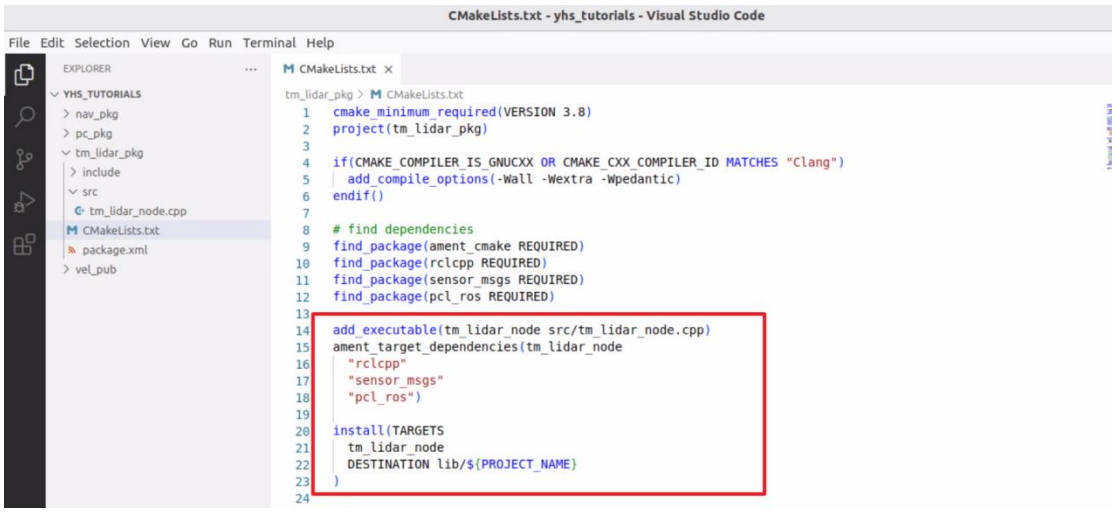
7) The "RCLCPP_INFO()" function is called to output a string message to the terminal program, indicating that the node has started successfully.

8) A node handle, "rclcpp::Node node", is defined in the main function, and this handle is used to subscribe to the data of the topic "/tmlidar_points" from the ROS2 core node. The callback function is set to the previously defined "pointCloudCallback()". The topic "/tmlidar_points" is the name of the topic where the 3D lidar node publishes the 3D point cloud. The 3D point cloud collected by the 3D lidar will be sent to this topic, and our own node, "tm_lidar_node", only needs to subscribe to it to receive the 3D lidar point cloud data.

9) The "rclcpp::spin()" function is called to block the execution of the main() function, ensuring that the node program does not end or exit.

(6) After finishing the code, the filename needs to be added to the compilation file to proceed with the compilation. The compilation file is located in the "tm_lidar_pkg" directory and named "CMakeLists.txt". Click on this file on the left side of the IDE interface, and its contents will be displayed on the right side. In the "CMakeLists.txt" file, add a new compilation rule for "tm_lidar_node.cpp". The content should be as follows:

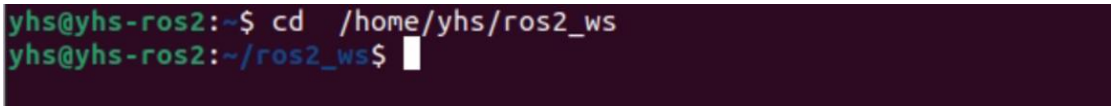
```
add_executable(tm_lidar_node src/tm_lidar_node.cpp)
ament_target_dependencies(tm_lidar_node
  "rclcpp"
  "sensor_msgs"
  "pcl_ros")

install(TARGETS
  tm_lidar_node
  DESTINATION lib/${PROJECT_NAME}
)
```




(7) Similarly, after making the modifications, press the keyboard shortcut Ctrl+S to save the changes. The small black dot on the right side of the filename above the code will turn into an "X," indicating that the file has been successfully saved. Now, let's proceed with

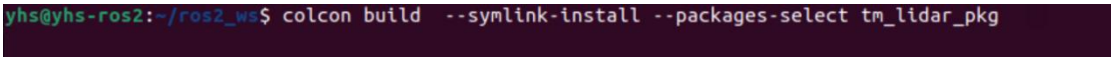
the compilation of the code file. Start a terminal program and enter the following command to navigate to the ROS2 workspace:

```
cd /home/yhs/ros2_ws
```

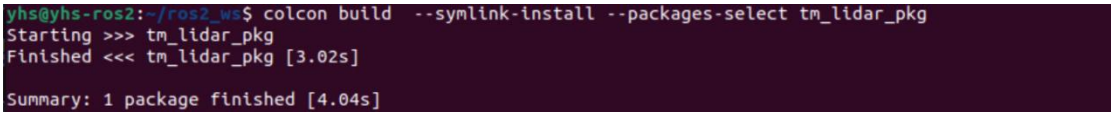



(8)Execute below command to compile

```
colcon build --symlink-install --packages-select tm_lidar_pkg
```



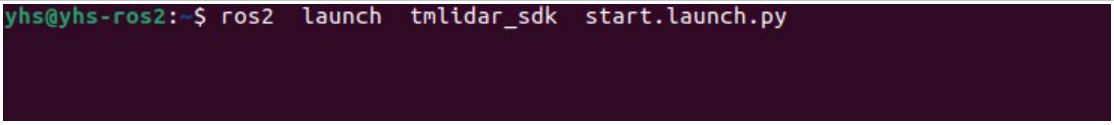

After executing this command, you will see scrolling compilation information until you encounter the message "Summary: 1 package finished". This message indicates that the new tm_lidar_node node has been successfully compiled.




3. Running the "tm_lidar_node" node

(1) Start by running the 3D lidar node. Open a terminal and enter the command to start the lidar node by pressing Enter.

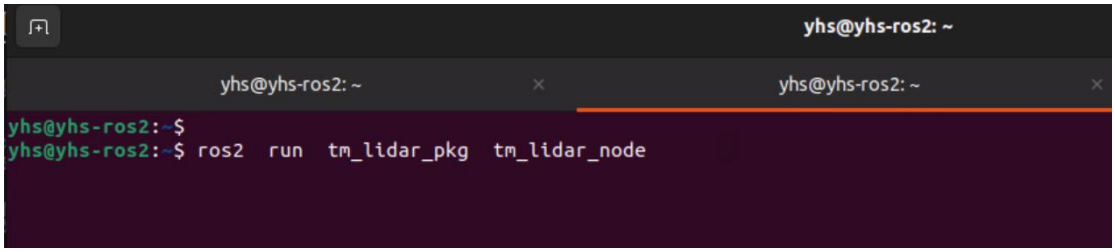
```
ros2 launch tmlidar_sdk start.launch.py
```





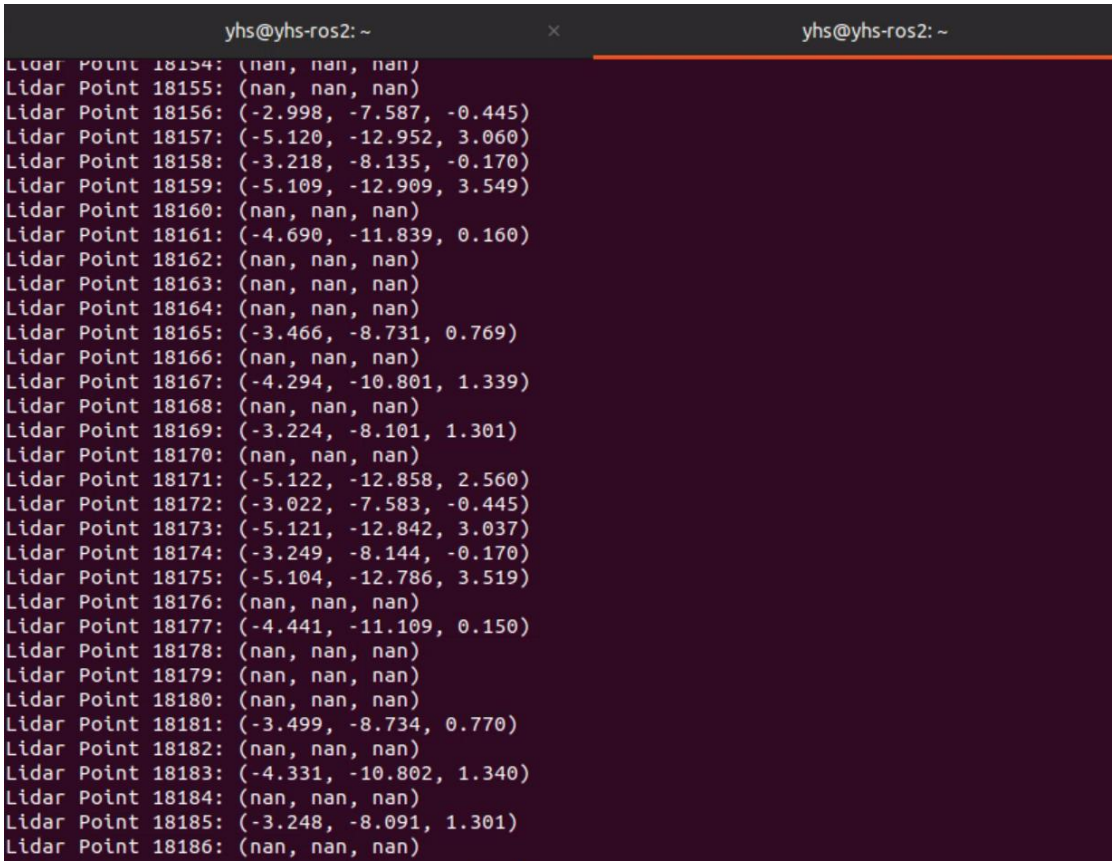
(2) Now we can run the previously written tm_lidar_node. Open another terminal and enter the command, then press Enter.

```
ros2 run tm_lidar_pkg tm_lidar_node
```




This command will start our written tm_lidar_node. According to the program logic, it will continuously retrieve point cloud data packets from the 3D lidar's point cloud topic "/tmlidar_points". It will then convert the ROS-formatted 3D point cloud to the PCL format and display the x, y, and z coordinate values of all the points in the terminal.

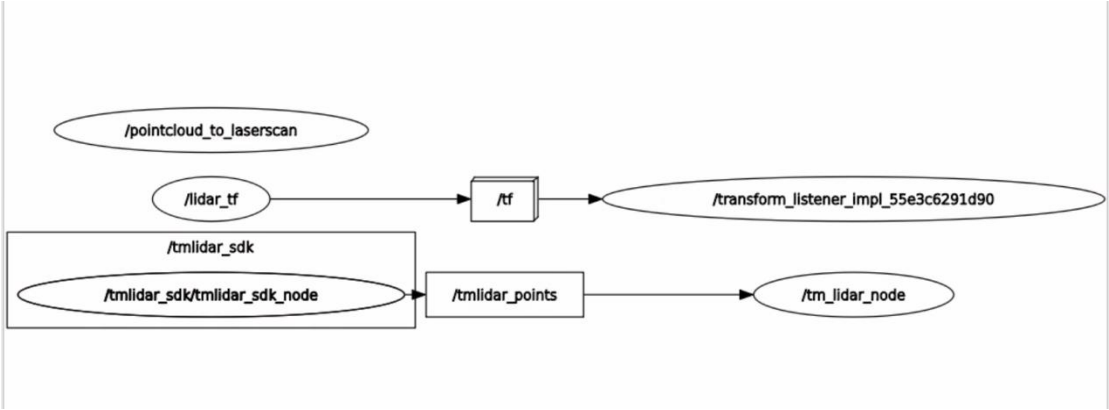

In the terminal, the message "tm_lidar_node start" will be displayed, indicating that the tm_lidar_node has started successfully. Subsequently, the terminal will continuously refresh and display the coordinate values of all the points in the point cloud.



4. While keeping the previous terminal program running, we can check the node network relationships in the ROS2 system. Start another terminal program and enter the following command:

```
rqt_graph
```

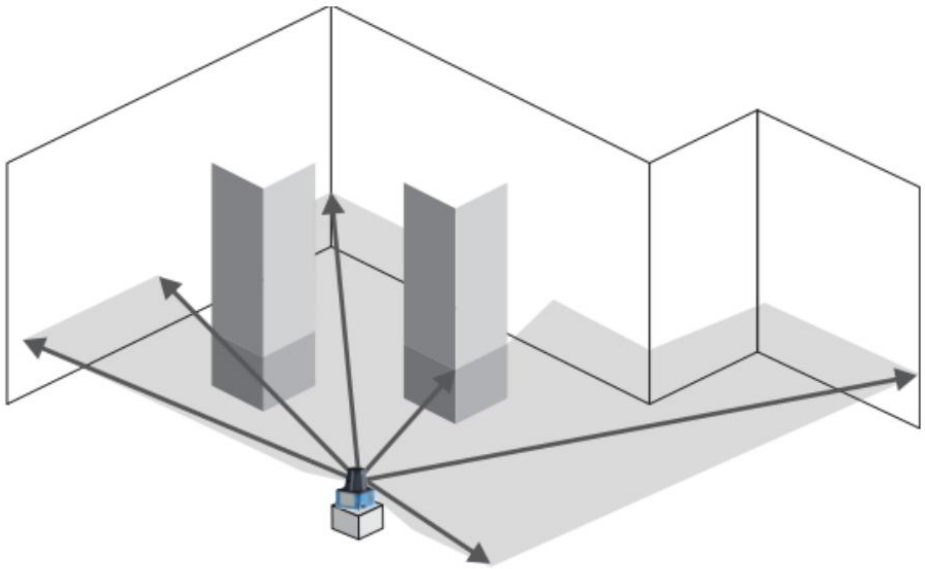

After pressing the Enter key, a new window will pop up displaying a series of interconnected boxes, which represent the node network relationships in the current ROS2 system.



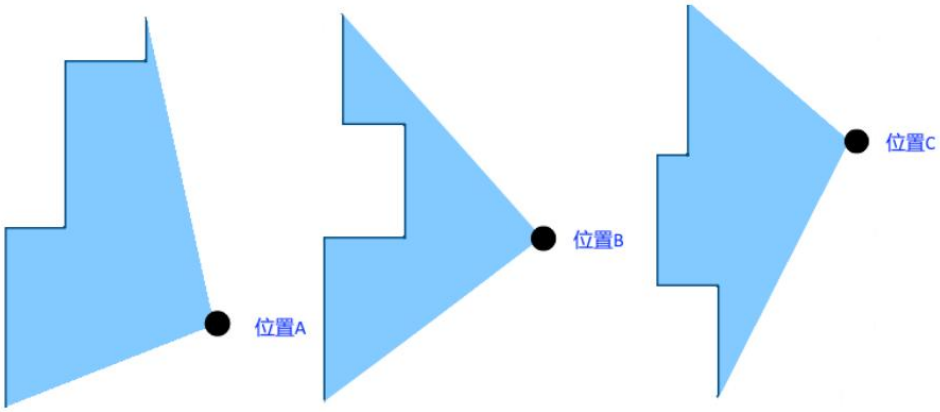
# Experiment 6: 2D Mapping

The "pointcloud_to_laserscan" node can be used to convert 3D lidar point cloud data into 2D laser data format, which is used for 2D mapping.

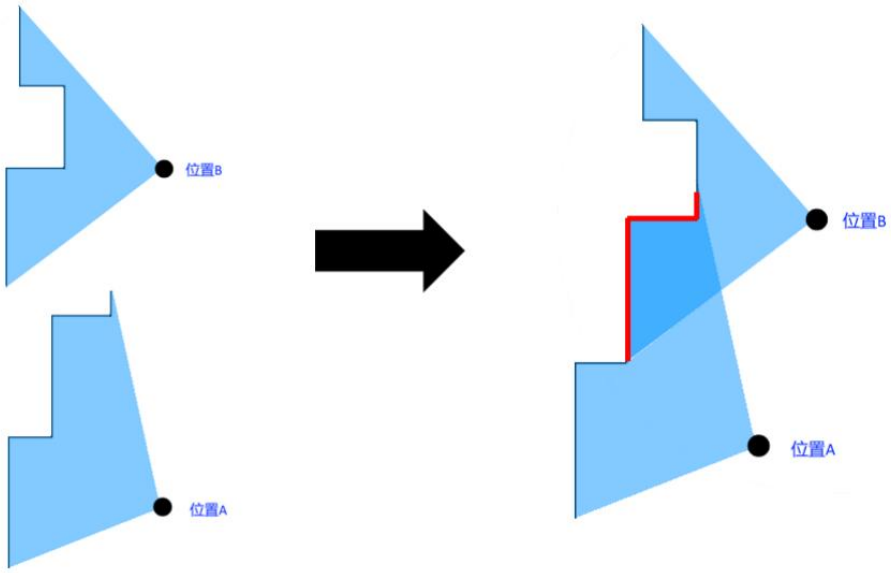
Now, let's have a preliminary understanding of the mapping process through 2D mapping. After familiarizing ourselves with the data format of lidar, we will begin applying lidar data. In this section, we will encounter a term called "SLAM," which stands for "Simultaneous Localization And Mapping." SLAM was originally proposed by Smith, Self, and Cheeseman in 1988 and is considered a key technology for achieving true autonomous mobile robots due to its significant theoretical and practical value. To understand SLAM, it is necessary to understand the characteristics of lidar data. Lidar scan data can be viewed as a cross-sectional representation of obstacle distribution, reflecting the shape and distribution of obstacle edges facing the lidar at a specific height.



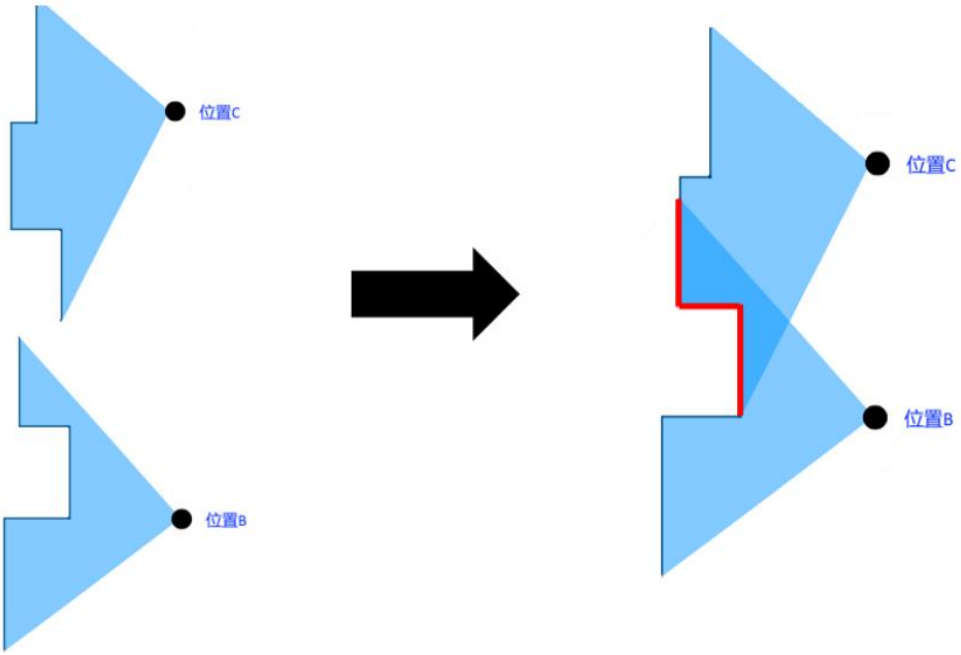
Therefore, when a robot equipped with a lidar sensor moves in an environment, at any given moment, it can only perceive partial contours of obstacles within a limited range and their relative positions in the robot's coordinate system. For example, in the following figure, it depicts the obstacle contours captured by the lidar sensor at three adjacent and relatively close positions A, B, and C.



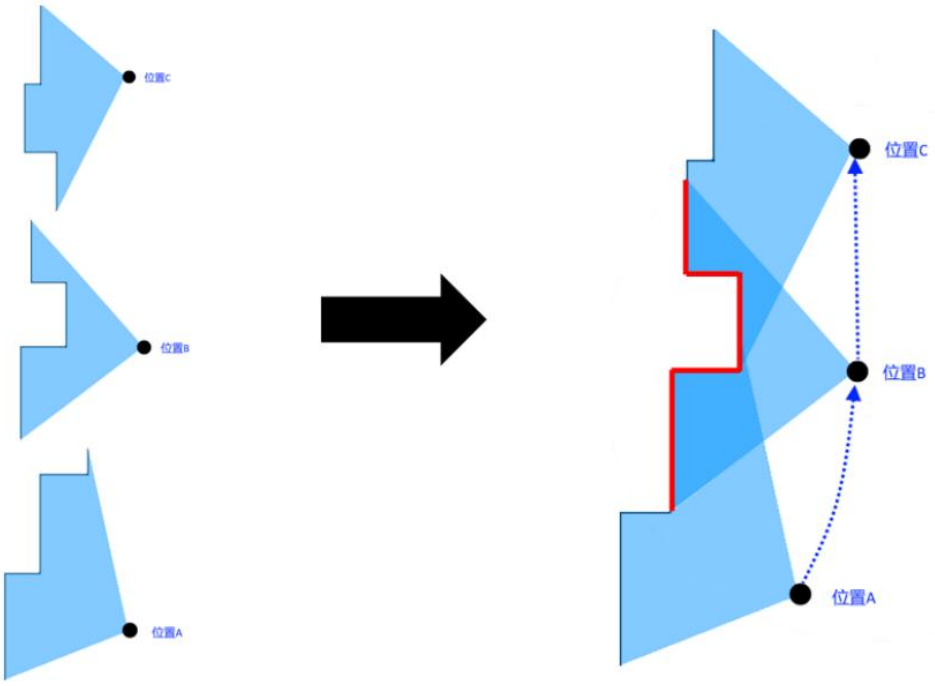

Although we may not know the exact spatial relationship between positions A, B, and C yet, through careful observation, we can notice that certain parts of the obstacle contours scanned at positions A, B, and C can be matched and overlapped. Since these three positions are relatively close to each other, we can assume that the similar parts of the obstacle contours belong to the same obstacle. Based on this assumption, we can attempt to overlay and align the similar parts of the obstacle contours to create a larger pattern of the obstacle contours. For example, the overlapped obstacle contours from position A and position B would look like the following:




Similarly, when we overlay the obstacle contours from position B and position C, it would look like the following:




By following the aforementioned method and combining the consecutive obstacle contours scanned by the lidar sensor from multiple positions, a relatively complete 2D map can be formed. This map is a representation of the obstacle contours and their distribution on a 2D plane, reflecting the environment as seen from the scanning plane of the lidar sensor. During the map construction process, based on the overlapping relationships of the obstacle contours, it is possible to infer the spatial relationships between the positions that the robot has traversed and determine the robot's own position within the map. This simultaneous achievement of map construction and real-time self-localization of the robot constitutes the essence of SLAM (Simultaneous Localization And Mapping). Taking the previous example of positions A, B, and C, by combining the lidar scan contours from these three positions, a relatively complete 2D map can be obtained, and the positions of A, B, and C within this map can be determined.




Next, we will use Gmapping and Cartographer for mapping.

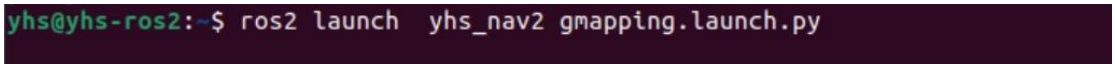
The mapping approach used in Gmapping is based on the PBRF (Particle-Based Recursive Filtering) algorithm, where each particle carries a map. Gmapping effectively utilizes odometry information to reduce the requirements on the frequency of the used lidar sensor. The robot uses odometry to calculate the next pose based on its current pose, and then each particle's laser sensor data is compared with the built map features. The particle with the best match to the map features is selected as the current pose, and this process is repeated to construct the map.

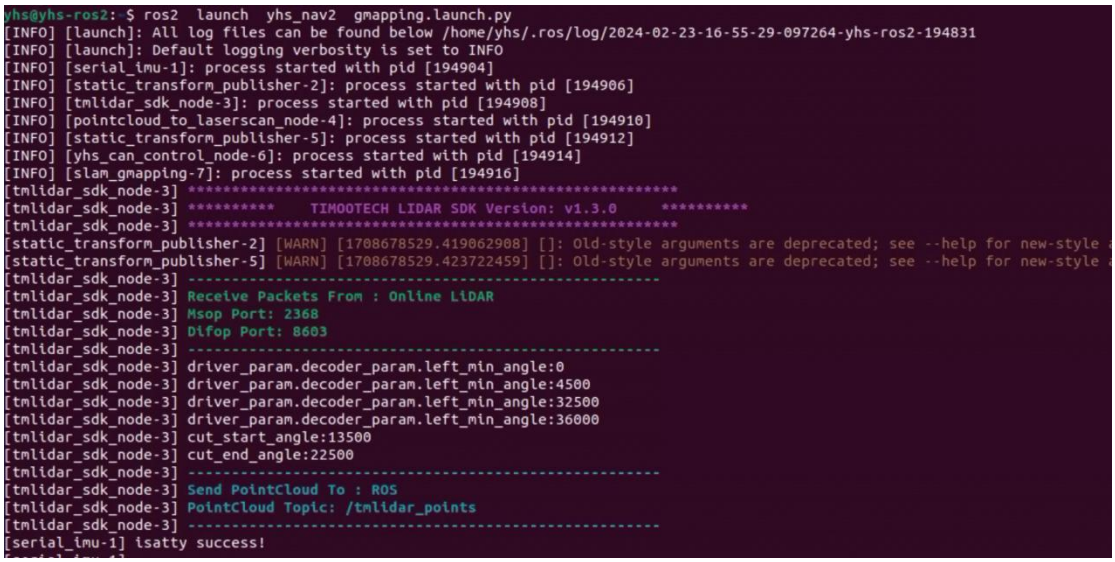
Cartographer, on the other hand, is a graph optimization-based SLAM algorithm with the mapping approach known as GraphSLAM. It uses graph optimization to efficiently compute highly accurate maps. Unlike online pose correction methods, Cartographer records all the data and performs a final batch computation, resulting in efficient map processing. Additionally, Cartographer incorporates loop closure detection, which aids in identifying and correcting errors in the map caused by revisiting previously seen locations.

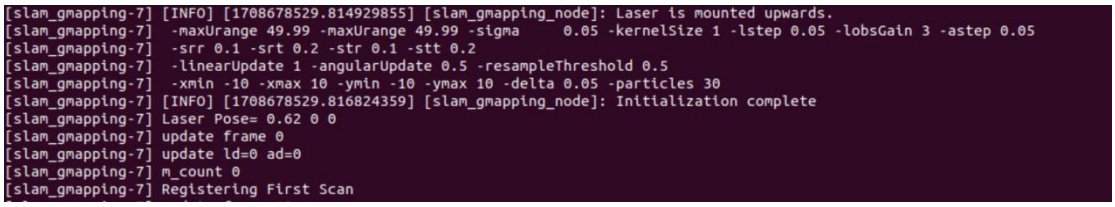
## 1. Gmapping mapping process

(1) Open the terminal and enter the command, then press Enter. If you see "Registering Scans: Done," it indicates that the Gmapping mapping process has been successfully initiated.

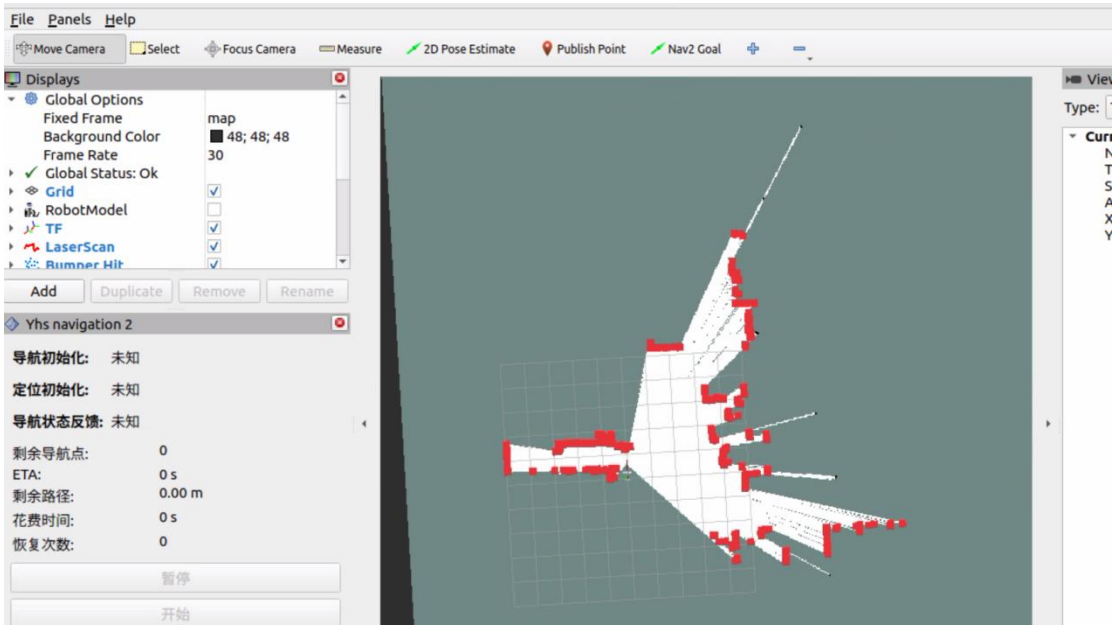
```
ros2 launch yhs_nav2 gmapping.launch.py
```



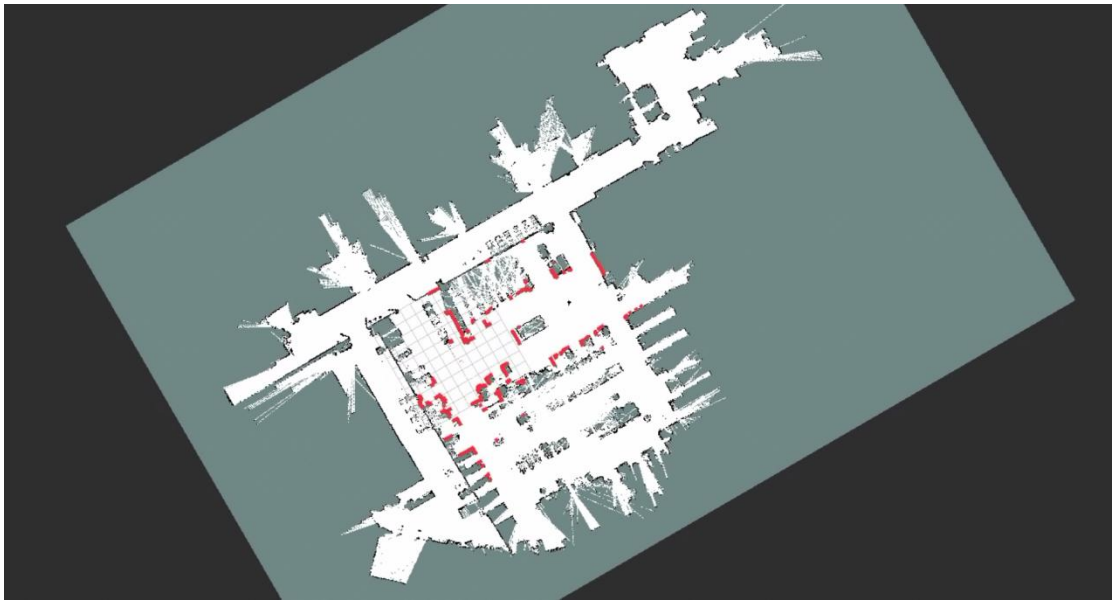

(2) At this point, in rviz2, you will see the ground around the robot turning into a dark gray color, while underneath the robot, there will be a white pattern. This pattern is formed by overlaying multiple line segments, which represent the trajectories of the distance measurement lasers flying from the center of the robot's base to each red obstacle point detected by the lidar sensor. These line segments indicate the absence of obstacles within their span.




(3) At this point, you can use a remote controller to control the robot's chassis movement for mapping. The mapping process should follow the following two principles:

- Avoid rapid rotations.
- In scenarios with both corridors and rooms, first map the outlines of the rooms before mapping the corridors.

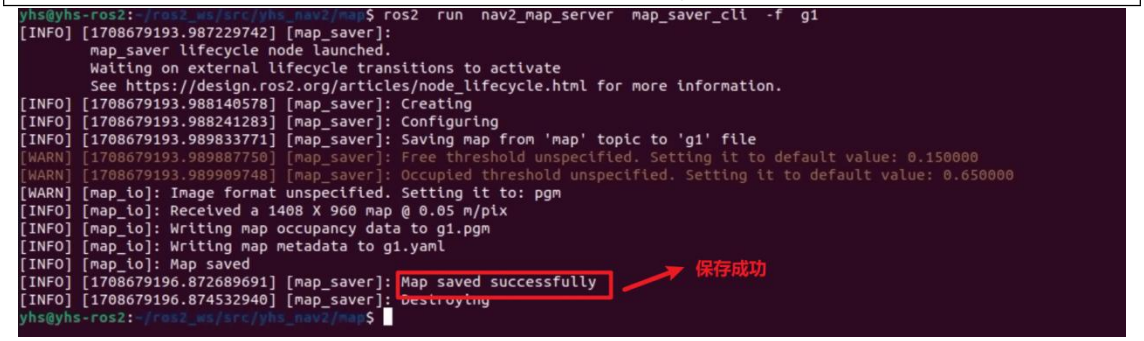
During the mapping process, you can observe the map in real-time on rviz2. If there is a significant discrepancy between the map and the actual environment, stop mapping and restart the process following the two principles mentioned above.



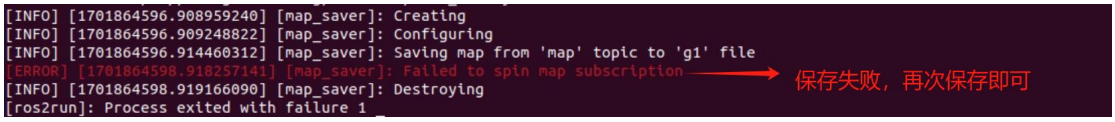
(4) Use the remote controller to maneuver the robot's chassis and cover the desired mapping area until the mapping result meets the requirements. At this point, you can save the map. Open another terminal and navigate to the

"/home/yhs/ros2_ws/src/yhs_nav2/map" directory. Enter the command to save the map. The map name is specified at the end of the command, for example, "g1". If you choose the same name as a previously saved map, it will overwrite the previous map.

```
ros2 run nav2_map_server map_saver_cli -f g1
```



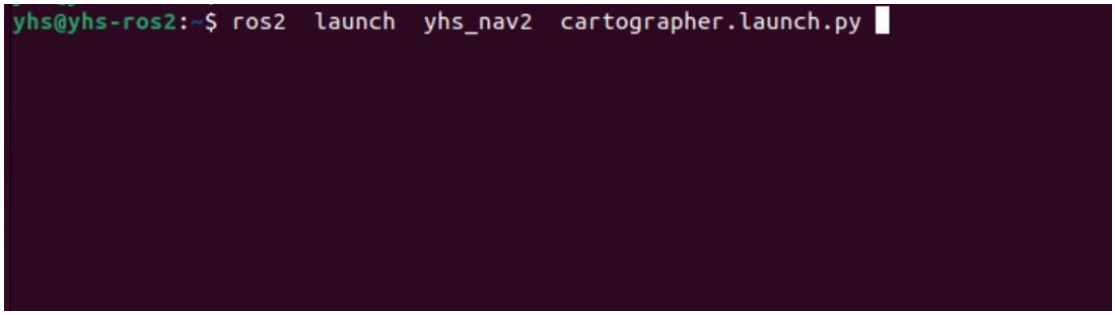

If the saving process fails, you can simply retry the command to save the map.

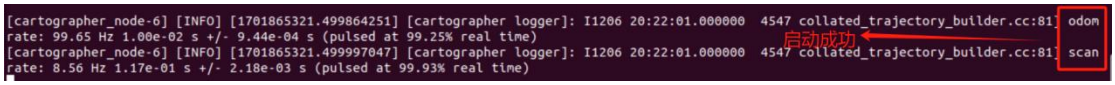



## 2. Cartographer mapping process

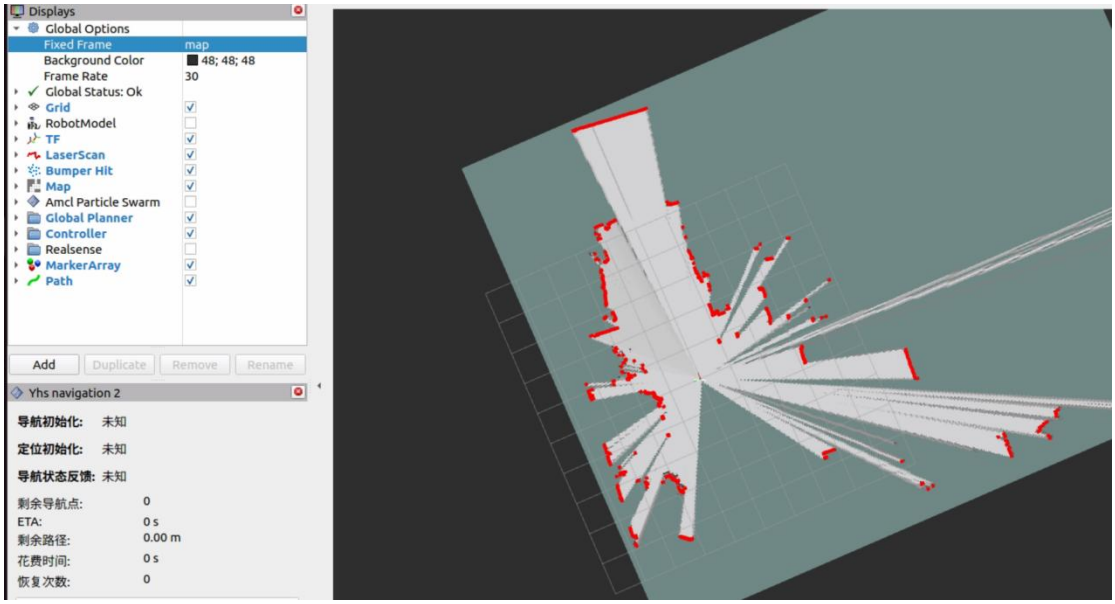
(1)Open terminal, entering below command:

```
ros2 launch yhs_nav2 cartographer.launch.py
```

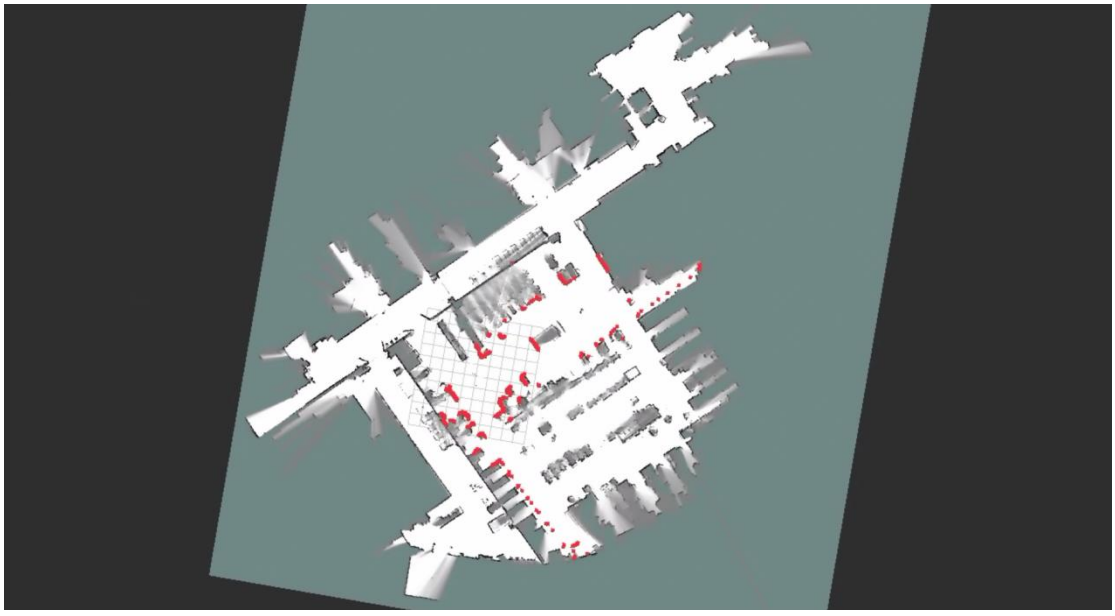



(2) At this moment, in rviz2, you will notice that the ground around the robot appears as a dark gray color, while underneath the robot, there is a white pattern. This pattern is formed by overlaying multiple line segments. These segments represent the lines connecting the center of the robot's base to each red obstacle point detected by the lidar sensor. They represent the trajectory of the distance measurement lasers and indicate that there are no obstacles within the span of each line segment.



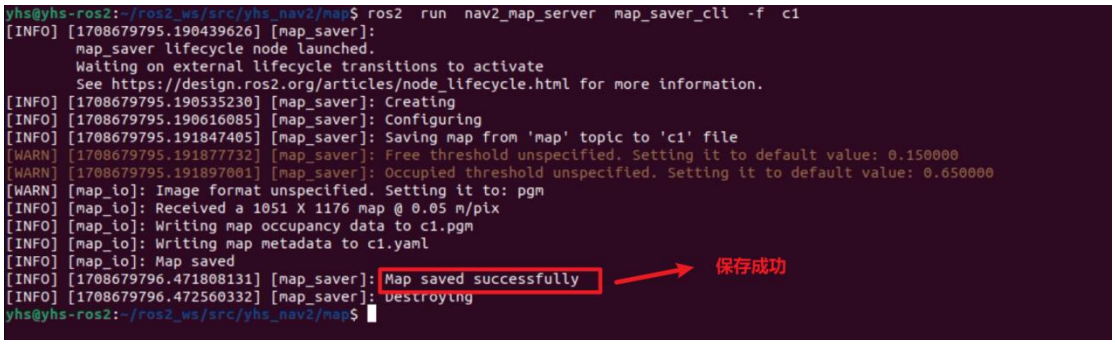

(3) At this point, you can use a remote controller to control the chassis and start mapping. During the mapping process, it is important to avoid rapid rotations of the chassis. You can observe the map in real-time on rviz2. If you notice a significant discrepancy between the map and the actual environment, it is recommended to stop the mapping process. Restart the mapping process again, this time following the principle of avoiding rapid rotations.



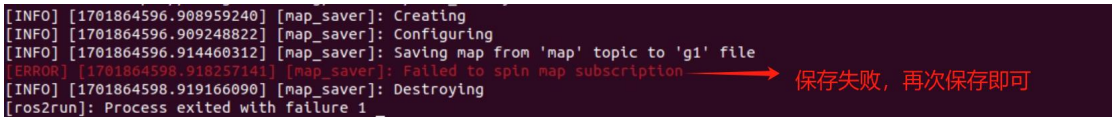
(4) Use the remote controller to navigate the chassis throughout the desired mapping area and return to the starting point. Once you have stopped, observe the map in rviz2. If there are overlapping areas in the map, the mapping program will perform optimization and correction. Be patient and wait for the mapping optimization process to complete. Once the desired mapping result is achieved, you can proceed to save the map. Open another terminal and navigate to the "/home/yhs/ros2_ws/src/yhs_nav2/map" directory. Enter the command to save the map, specifying the map name at the end of the command, for

example, "c1". If you choose the same name as a previously saved map, it will overwrite the previous map.

```
ros2 run nav2_map_server map_saver_cli -f c1
```




If the saving process fails, you can simply retry the command to save the map.



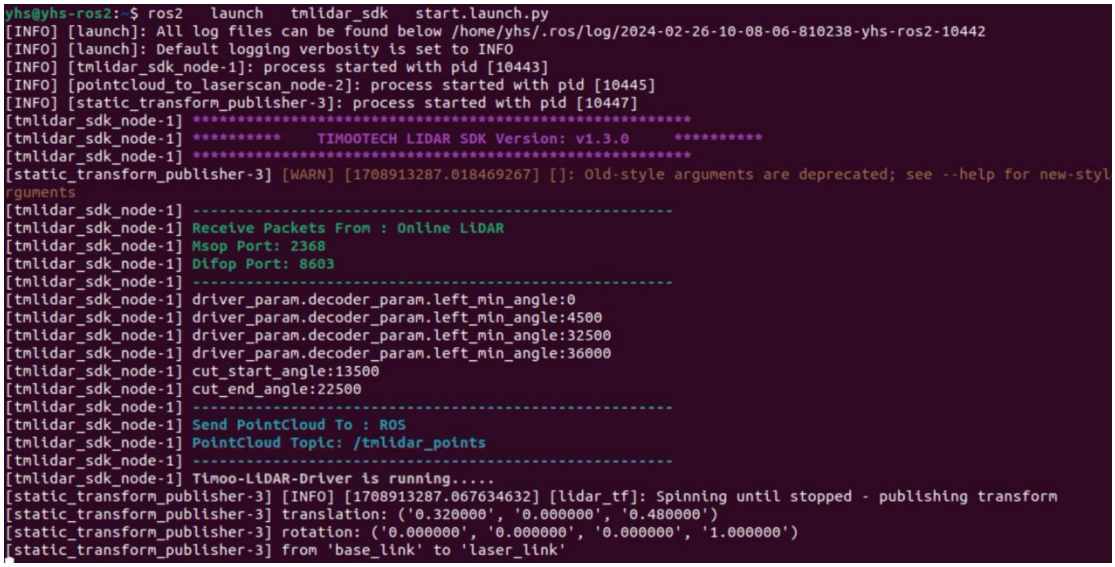
# Experiment 7: 3D Mapping

LIO-SAM, which stands for Laser Inertial Odometry and Mapping with Smoothing and Mapping, is a tightly-coupled framework for laser odometry that integrates laser and inertial measurements to achieve high-precision, real-time mobile robot trajectory estimation and map construction. LIO-SAM formulates the laser odometry on a factor graph, allowing a large number of relative and absolute measurements from different sources, including loop closures, to be integrated as factors in the system. The estimated motion from the pre-integrated inertial measurement unit (IMU) helps mitigate point cloud biases and generates an initial guess for laser odometry optimization. The laser odometry solution is then used to estimate IMU biases. To ensure real-time performance, old laser scans are marginalized for pose optimization instead of matching the laser scans to a global map. Performing scan-to-map matching at a local scale rather than a global one significantly improves the system's real-time performance. Keyframes are selectively introduced, and an efficient sliding window method registers new keyframes to a fixed-size set of previous "sub-keyframes" to further enhance efficiency.

**LIO-SAM mapping process:**

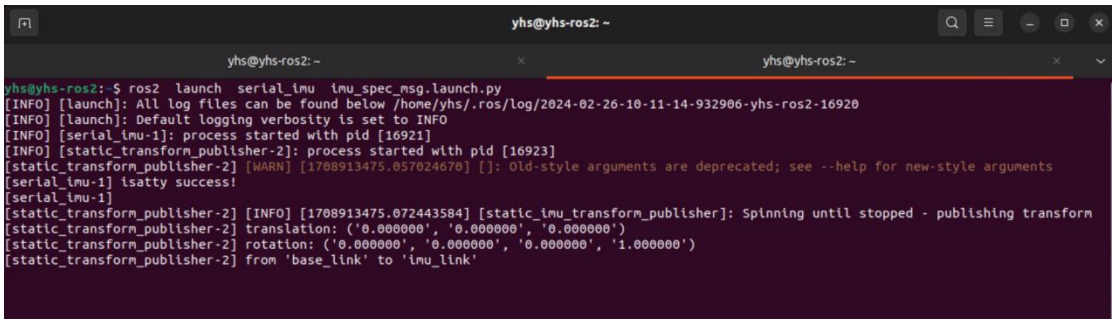
(1) Open the terminal and run the 3D lidar driver program.

```
ros2 launch tmlidar_sdk start.launch.py
```



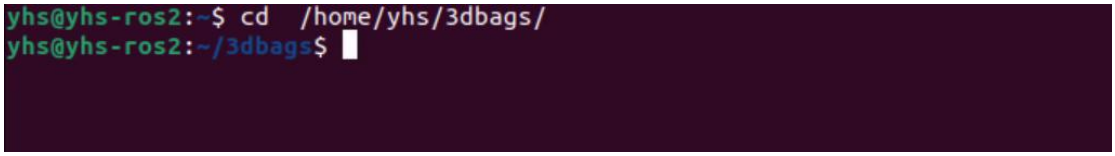

(2) Open another terminal and run the IMU driver program.

```
ros2 launch serial_imu imu_spec_msg.launch.py
```



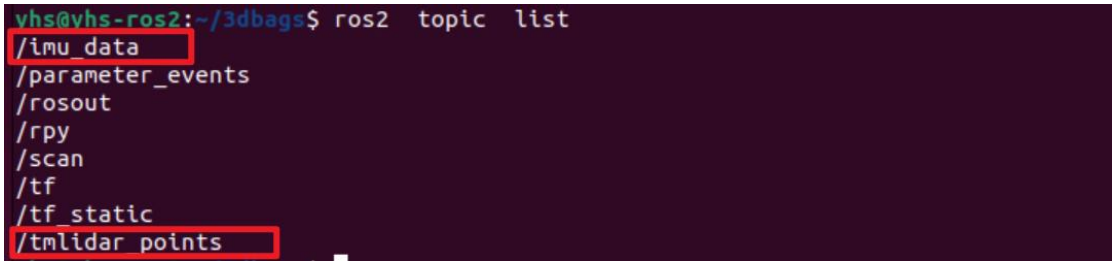

(3) Open another terminal and navigate to the "3dbags" directory.

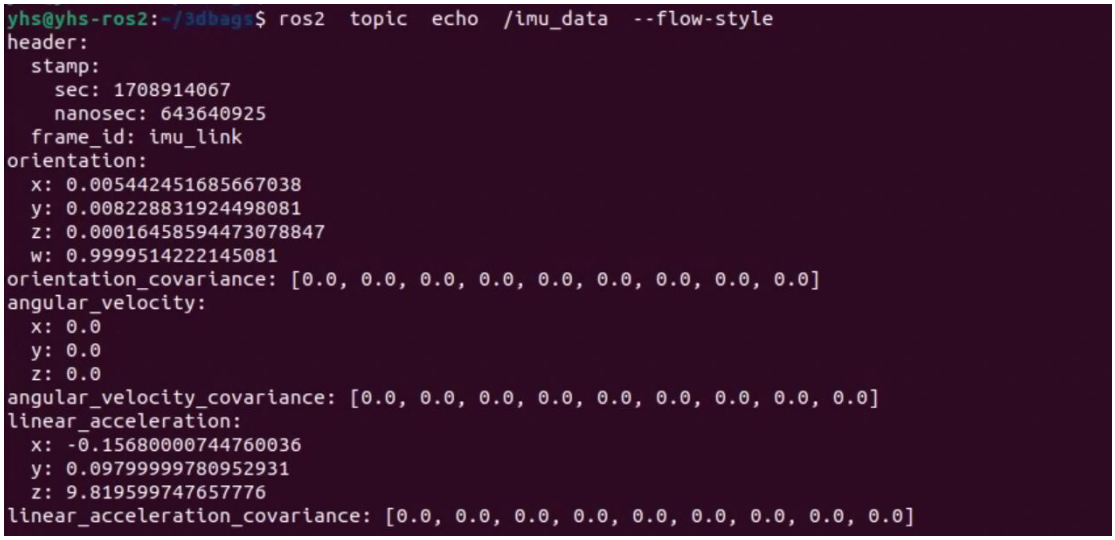
| cd /home/yhs/3dbags |
|---|




(4) Check the topics to confirm that the 3D lidar point cloud topic "/tmlidar_points" and the IMU data topic "/imu_data" are both outputting data.

| ros2 topic list |
|---|
| ros2 topic echo /imu_data --flow-style |
| ros2 topic echo /tmlidar_points --flow-style |



```
yhs@yhs-ros2:~/3dbags$ ros2 topic echo /tmlidar_points --flow-style
header:
  stamp:
    sec: 1708914142
    nanosec: 307362556
  frame_id: laser_link
height: 16
width: 2040
fields: [{name: x, offset: 0, datatype: 7, count: 1}, {name: y, offset: 4, datatype: 7, count: 1}, {name: z, offset: 8, datatype: 7, count: 1}
, {name: intensity, offset: 16, datatype: 2, count: 1}, {name: ring, offset: 18, datatype: 4, count: 1}, {name: timestamp, offset: 24, datatyp
e: 8, count: 1}]
is_bigendian: false
point_step: 32
row_step: 65280
data: [0, 0, 192, 127, 0, 0, 192, 127, 0, 0, 192, 127, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 41, 64, 141, 119, 253, 118, 217, 65, 62, 21
9, 115, 190, 103, 34, 39, 191, 134, 22, 15, 60, 255, 255, 255, 255, 49, 0, 8, 0, 0, 0, 0, 0, 54, 64, 141, 119, 253, 118, 217, 65, 0, 0, 192, 1
27, 0, 0, 192, 127, 0, 0, 192, 127, 255, 255, 255, 255, 0, 0, 1, 0, 0, 0, 0, 0, 67, 64, 141, 119, 253, 118, 217, 65, 86, 82, 112, 190, 133, 13
6, 36, 191, 64, 32, 6, 61, 255, 255, 255, 255, 52, 0, 9, 0, 0, 0, 0, 0, 80, 64, 141, 119, 253, 118, 217, 65, '...']
is_dense: true
```

(5) Start recording data by executing the command in the "/home/yhs/3dbags" directory.

| ros2 bag record /imu_data /tmlidar_points -o in1 |
|---|

```
yhs@yhs-ros2:~/3dbags$ ros2 bag record /imu_data /tmlidar_points -o in1
[INFO] [1708914511.925583384] [rosbag2_recorder]: Press SPACE for pausing/resuming
[INFO] [1708914511.939362855] [rosbag2_storage]: Opened database 'in1/in1_0.db3' for READ_WRITE.
[INFO] [1708914511.946524448] [rosbag2_recorder]: Listening for topics...
[INFO] [1708914511.946613800] [rosbag2_recorder]: Event publisher thread: Starting
[INFO] [1708914511.969890230] [rosbag2_recorder]: Subscribed to topic '/tmlidar_points'
[INFO] [1708914511.974112330] [rosbag2_recorder]: Subscribed to topic '/imu_data'
[INFO] [1708914511.974305887] [rosbag2_recorder]: Recording...
[INFO] [1708914511.976796734] [rosbag2_recorder]: All requested topics are subscribed. Stopping discovery...
```

"In1" is the name of the folder where the data will be stored. You can choose any name you prefer. Once the recording starts, you can use the remote controller to navigate the chassis around the desired mapping area. During the movement, avoid rotating in place or making rapid rotations. Finally, return to the starting point. To stop the recording, make sure to press "Ctrl + C" on the terminal.

```
yhs@yhs-ros2:~/3dbags$ ros2 bag record /imu_data /tmlidar_points -o in1
[INFO] [1708914511.925583384] [rosbag2_recorder]: Press SPACE for pausing/resuming
[INFO] [1708914511.939362855] [rosbag2_storage]: Opened database 'in1/in1_0.db3' for READ_WRITE.
[INFO] [1708914511.946524448] [rosbag2_recorder]: Listening for topics...
[INFO] [1708914511.946613800] [rosbag2_recorder]: Event publisher thread: Starting
[INFO] [1708914511.969890230] [rosbag2_recorder]: Subscribed to topic '/tmlidar_points'
[INFO] [1708914511.974112330] [rosbag2_recorder]: Subscribed to topic '/imu_data'
[INFO] [1708914511.974305887] [rosbag2_recorder]: Recording...
[INFO] [1708914511.976796734] [rosbag2_recorder]: All requested topics are subscribed. Stopping discovery...
[INFO] [1708914818.547375793] [rosbag2_cpp]: Writing remaining messages from cache to the bag. It may take a while
[INFO] [1708914818.552764806] [rosbag2_recorder]: Event publisher thread: Exiting
[INFO] [1708914818.553506372] [rosbag2_recorder]: Recording stopped
[INFO] [1708914818.563457362] [rosbag2_recorder]: Recording stopped
yhs@yhs-ros2:~/3dbags$
```

(6) Parameter modification: In an indoor, flat, long corridor scenario, change the values of three parameters in the "/home/yhs/ros2_ws/src/LIO-SAM/config/params.yaml" file to the following values. No changes are required for other scenarios; you can use the default values.

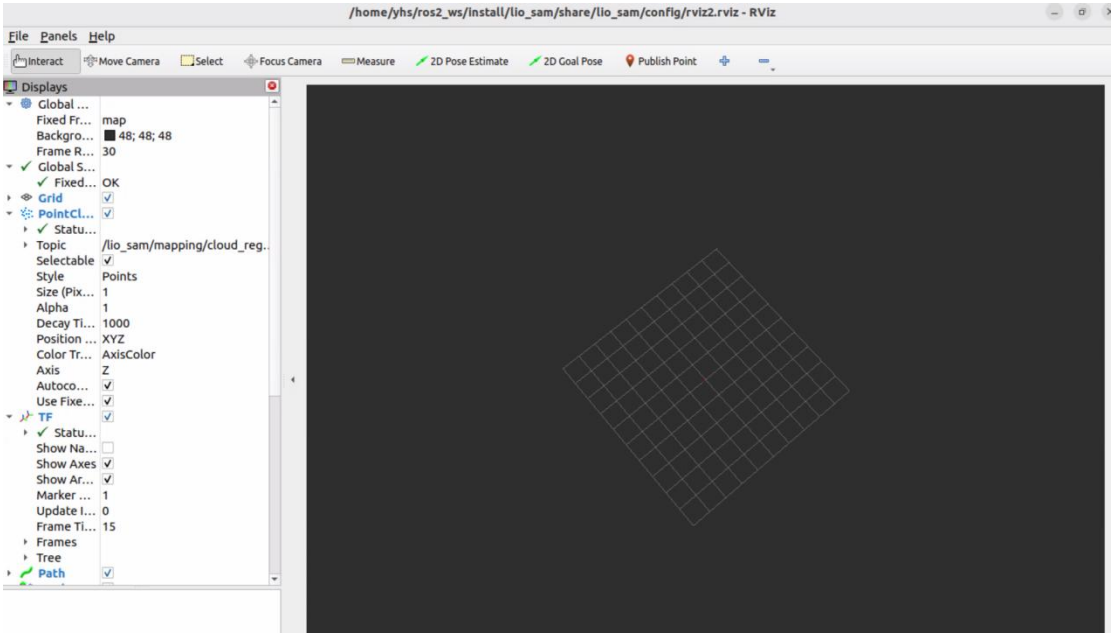
| | |
|---|---|
| z_tollerance | 0.0 |
| rotation_tollerance | 0.0 |
| historyKeyframeFitnessScore | 0.1 |

(7) Run the LIO-SAM node by first closing the previously opened 3D lidar and IMU driver program nodes. Enter the command.

| ros2 launch lio_sam run.launch.py |
|---|

```
yhs@yhs-ros2:~$ ros2  launch  lio_sam  run.launch.py
[INFO] [launch]: All log files can be found below /home/yhs/.ros/log/2024-02-26-10-58-05-646367-yhs-ros2-109478
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [static_transform_publisher-1]: process started with pid [109480]
[INFO] [lio_sam_imuPreintegration-2]: process started with pid [109482]
[INFO] [lio_sam_imageProjection-3]: process started with pid [109484]
[INFO] [lio_sam_featureExtraction-4]: process started with pid [109486]
[INFO] [lio_sam_mapOptimization-5]: process started with pid [109488]
[INFO] [rviz2-6]: process started with pid [109490]
[static_transform_publisher-1] [WARN] [1708916285.838180238] []: Old-style arguments are deprecated; see --help for new-style arguments
[lio_sam_featureExtraction-4] [INFO] [1708916285.854800450] [rclcpp]: ----> Feature Extraction Started.
[lio_sam_imageProjection-3] [INFO] [1708916285.863780628] [rclcpp]: ----> Image Projection Started.
[static_transform_publisher-1] [INFO] [1708916285.867494543] [static_transform_publisher_0x3U739Hh87WSDaZ]: Spinning until stopped - publishin
g transform
[static_transform_publisher-1] translation: ('0.000000', '0.000000', '0.000000')
[static_transform_publisher-1] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-1] from 'map' to 'odom'
[lio_sam_imuPreintegration-2] [WARN] [1708916285.875742020] [rcl.logging_rosout]: Publisher already registered for provided node name. If this
 is due to multiple nodes with the same name then all logs for that logger name will go out over the existing publisher. As soon as any node w
ith that name is destructed it will unregister the publisher, preventing any further logs for that name from being published on the rosout top
ic.
[lio_sam_imuPreintegration-2] [INFO] [1708916285.887140652] [rclcpp]: ----> IMU Preintegration Started.
[lio_sam_mapOptimization-5] [INFO] [1708916285.926614905] [rclcpp]: ----> Map Optimization Started.
[rviz2-6] [INFO] [1708916286.069154270] [rviz2]: Stereo is NOT SUPPORTED
[rviz2-6] [INFO] [1708916286.069340632] [rviz2]: OpenGl version: 4.5 (GLSL 4.5)
[rviz2-6] [INFO] [1708916286.084056751] [rviz2]: Stereo is NOT SUPPORTED
```

After executing the command, the Rviz2 interface will automatically pop up.




(8) After running the LIO-SAM node, play the recorded "bag" file by opening the terminal and navigating to the "/home/yhs/3dbags" directory.

```
cd  /home/yhs/3dbags
```

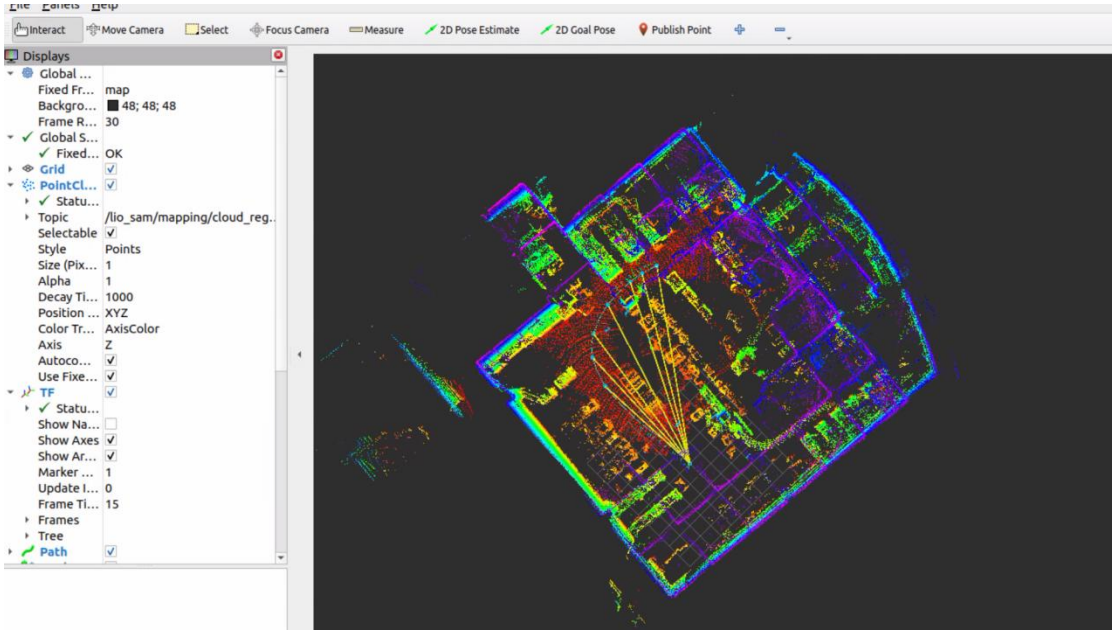
```
yhs@yhs-ros2:~$ cd  3dbags/
yhs@yhs-ros2:~/3dbags$
```

Enter the following command, where the "-r" option followed by the playback speed factor. It is common to play at 3 to 4 times the speed. During playback, you can press the "Space" key to pause and press it again to resume.

```
ros2  bag  play  -r  3  in1/
```

```
yhs@yhs-ros2:~/3dbags$ ros2 bag play -r 3 in1/
[INFO] [1708917345.594925814] [rosbag2_storage]: Opened database 'in1/in1_0.db3' for READ_ONLY.
[INFO] [1708917345.594980770] [rosbag2_player]: Set rate to 3
[INFO] [1708917345.604082637] [rosbag2_player]: Adding keyboard callbacks.
[INFO] [1708917345.604136390] [rosbag2_player]: Press SPACE for Pause/Resume
[INFO] [1708917345.604154152] [rosbag2_player]: Press CURSOR_RIGHT for Play Next Message
[INFO] [1708917345.604166621] [rosbag2_player]: Press CURSOR_UP for Increase Rate 10%
[INFO] [1708917345.604180309] [rosbag2_player]: Press CURSOR_DOWN for Decrease Rate 10%
[INFO] [1708917345.604990075] [rosbag2_storage]: Opened database 'in1/in1_0.db3' for READ_ONLY.
[INFO] [1708917347.925469925] [rosbag2_player]: Pausing play.
[INFO] [1708917369.106185252] [rosbag2_player]: Resuming play.
```

At this point, you will be able to see the point cloud, trajectory, and other data on Rviz2.




Wait for the playback to finish, and in the terminal where the LIO-SAM node is running, press "Ctrl + C" to save the map.

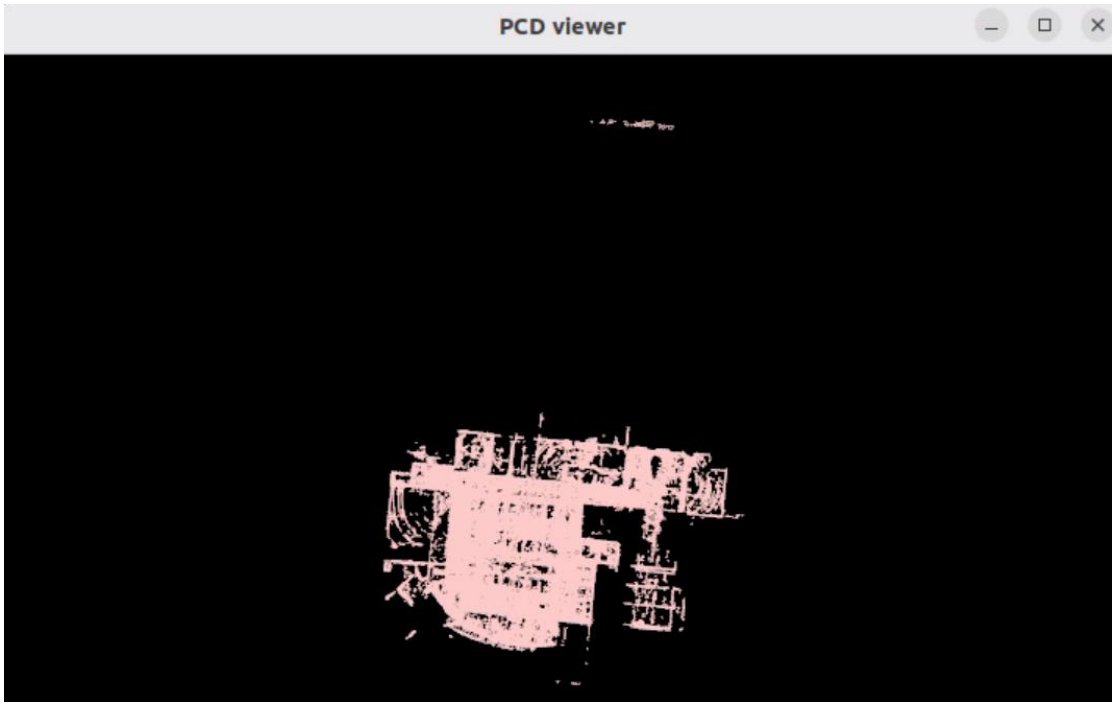
```
^C[WARNING] [launch]: user interrupted with ctrl-c (SIGINT)
[lio_sam_mapOptimization-5] [INFO] [1708917645.579579695] [rclcpp]: signal_handler(signum=2)
[lio_sam_featureExtraction-4] [INFO] [1708917645.579580665] [rclcpp]: signal_handler(signum=2)
[rviz2-6] [INFO] [1708917645.579570165] [rclcpp]: signal_handler(signum=2)
[lio_sam_imageProjection-3] [INFO] [1708917645.579603179] [rclcpp]: signal_handler(signum=2)
[static_transform_publisher-1] [INFO] [1708917645.579662451] [rclcpp]: signal_handler(signum=2)
[lio_sam_imuPreintegration-2] [INFO] [1708917645.579669477] [rclcpp]: signal_handler(signum=2)
[lio_sam_mapOptimization-5] ***************************************************
[lio_sam_mapOptimization-5] Saving map to pcd files ...
[INFO] [static_transform_publisher-1]: process has finished cleanly [pid 109480]
[INFO] [lio_sam_featureExtraction-4]: process has finished cleanly [pid 109486]
[INFO] [lio_sam_imageProjection-3]: process has finished cleanly [pid 109484]
[INFO] [lio_sam_imuPreintegration-2]: process has finished cleanly [pid 109482]
[INFO] [rviz2-6]: process has finished cleanly [pid 109490]
Processing feature cloud 209 of 210 ...***************************************************
[lio_sam_mapOptimization-5] Saving map to pcd files completed
[INFO] [lio_sam_mapOptimization-5]: process has finished cleanly [pid 109488]
yhs@yhs-ros2:~$
```

(9) To view the generated PCD point cloud map, the map is saved by default in the "/home/yhs/Downloads/LOAM" directory. Please note that each time you save, it will overwrite the previous map. Navigate to the "/home/yhs/Downloads/LOAM" directory, open the terminal, and use the "pcl_viewer" command to view the "cloudGlobal.pcd" map.

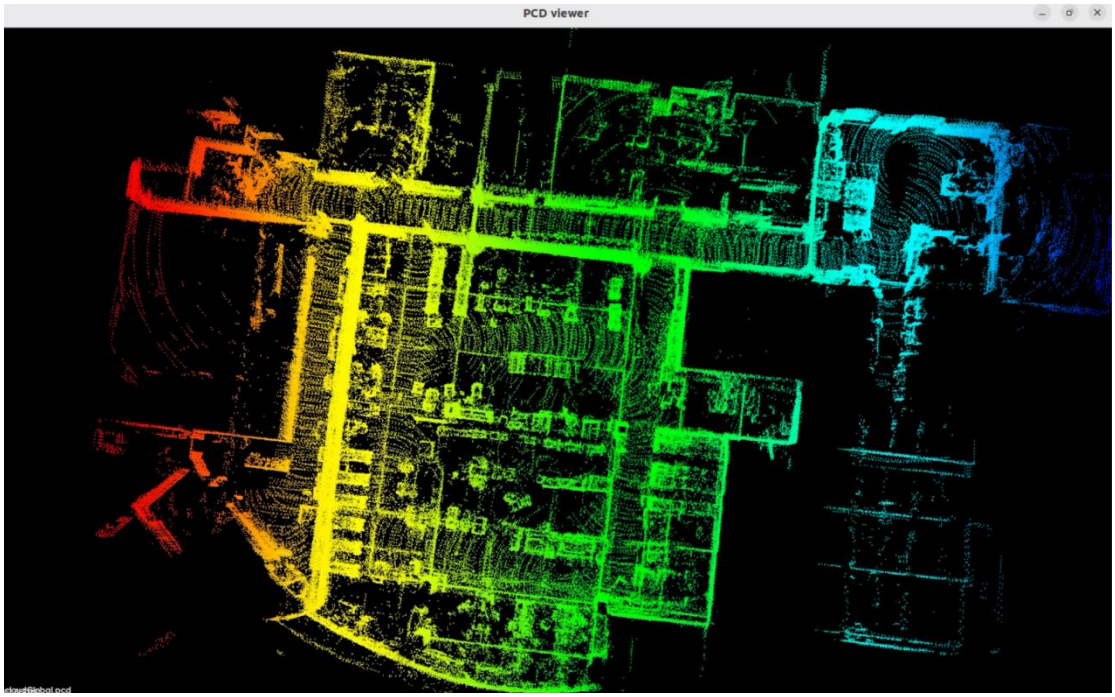
```
pcl_viewer  cloudGlobal.pcd
```

```
yhs@yhs-ros2:~/Downloads/LOAM$ pcl_viewer  cloudGlobal.pcd
2024-02-26 11:32:43.096 (   0.002s) [        8AB75880] vtkContextDevice2D.cxx:32    WARN| Error: no override found for 'vtkContextDevice2D'.
The viewer window provides interactive commands; for help, press 'h' or 'H' from within the window.
> Loading cloudGlobal.pcd [PCLVisualizer::setUseVbos] Has no effect when OpenGL version is ≥ 2
[done, 682.302 ms : 404116 points]
Available dimensions: x y z intensity
```

The "PCD viewer" interface will pop up.




Use the mouse scroll wheel to zoom in or out. Hold down the mouse scroll wheel and drag to pan the view. Press the number keys "1, 2, 3, 4, 5" to change the color scheme. Press the "f" key to zoom in on the location pointed by the mouse cursor. Observe the point cloud map to check if it forms a closed loop. If there is no closed loop, adjust the parameters and rebuild the map.




(10) Convert the PCD point cloud map to a grid map and save it. Enter the following command:

```
ros2 launch pcl_csf run.launch.py
```

```
yhs@yhs-ros2:~$ ros2 launch pcl_csf run.launch.py
[INFO] [launch]: All log files can be found below /home/yhs/.ros/log/2024-02-26-11-41-32-482187-yhs-ros2-196594
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [pcl_csf_node-1]: process started with pid [196595]
[pcl_csf_node-1] [INFO] [1708918893.255873950] [pcl_csf_node]: 载入点云地图 "/home/yhs/Downloads/LOAM/cloudGlobal.pcd" 成功。。。。。。
[pcl_csf_node-1] [INFO] [1708918893.256063488] [pcl_csf_node]: 开始转换，稍等片刻。。。。。。
[pcl_csf_node-1] [0] Configuring terrain...
[pcl_csf_node-1] [0] Configuring cloth...
[pcl_csf_node-1] [0]  - width: 127 height: 196
[pcl_csf_node-1] [0] Rasterizing...
[pcl_csf_node-1] [0] Simulating...
[pcl_csf_node-1] [0]  - post handle...
[pcl_csf_node-1] [INFO] [1708918898.969791157] [pcl_csf_node]: 转换完成，可以保存栅格地图了。。。。。。
```

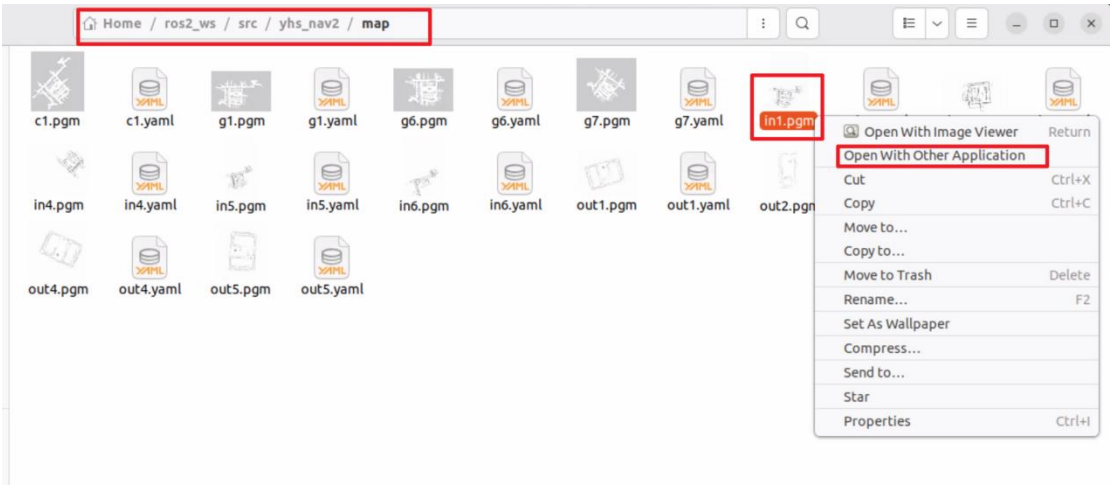
Wait for the conversion to complete. The larger the map, the longer the conversion time. After the conversion is done, do not close the conversion node yet. Open another terminal and navigate to the "/home/yhs/ros2_ws/src/yhs_nav2/map/" directory. Enter the following command to save the map, where "in1" is the name of the map and can be chosen freely. If the name is the same, it will overwrite the previous map.

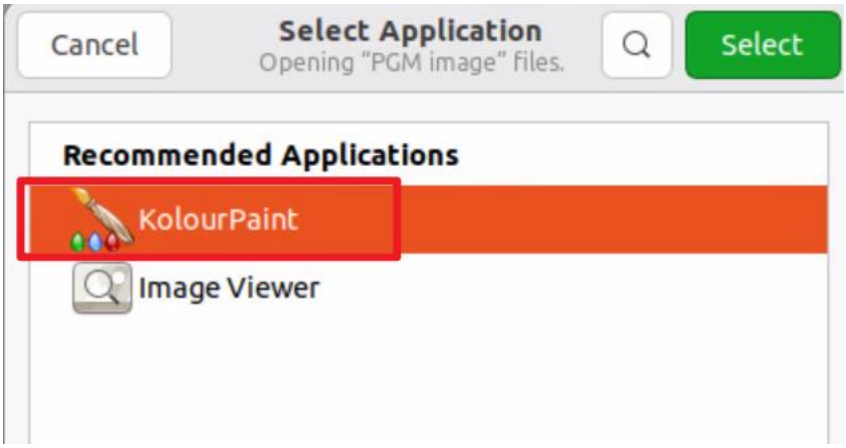
```
ros2 run nav2_map_server map_saver_cli -f in1
```

```
yhs@yhs-ros2:~/ros2_ws/src/yhs_nav2/map$ ros2 run nav2_map_server map_saver_cli -f in1
[INFO] [1708919161.208222120] [map_saver]:
       map_saver lifecycle node launched.
       Waiting on external lifecycle transitions to activate
       See https://design.ros2.org/articles/node_lifecycle.html for more information.
[INFO] [1708919161.208938946] [map_saver]: Creating
[INFO] [1708919161.209009893] [map_saver]: Configuring
[INFO] [1708919161.212799340] [map_saver]: Saving map from 'map' topic to 'in1' file
[WARN] [1708919161.212820762] [map_saver]: Free threshold unspecified. Setting it to default value: 0.150000
[WARN] [1708919161.212835157] [map_saver]: Occupied threshold unspecified. Setting it to default value: 0.650000
[WARN] [map_io]: Image format unspecified. Setting it to: pgm
[INFO] [map_io]: Received a 1239 X 1929 map @ 0.05 m/pix
[INFO] [map_io]: Writing map occupancy data to in1.pgm
[INFO] [map_io]: Writing map metadata to in1.yaml
[INFO] [map_io]: Map saved
[INFO] [1708919161.903273239] [map_saver]: Map saved successfully
[INFO] [1708919161.905159036] [map_saver]: Destroying
```

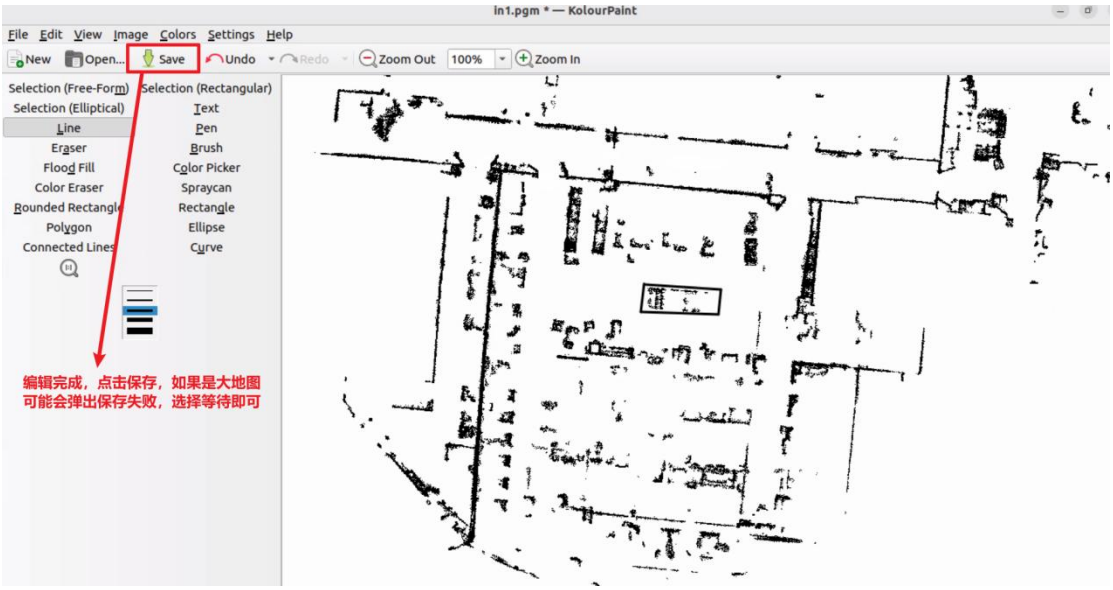
保存成功

(11) Grid map editing: After obtaining the grid map, you may need to clear certain areas or add obstacles to the map. Navigate to the "/home/yhs/ros2_ws/src/yhs_nav2/map/" directory and locate the map you want to edit. Right-click and select "KolourPaint" software to open it.




Use the left mouse button to draw lines and erase. After finishing the editing, click on the save button to save the changes. Once the saving is complete, you can close the software.



# Experiment 8: Navigation2 Framework Instruction

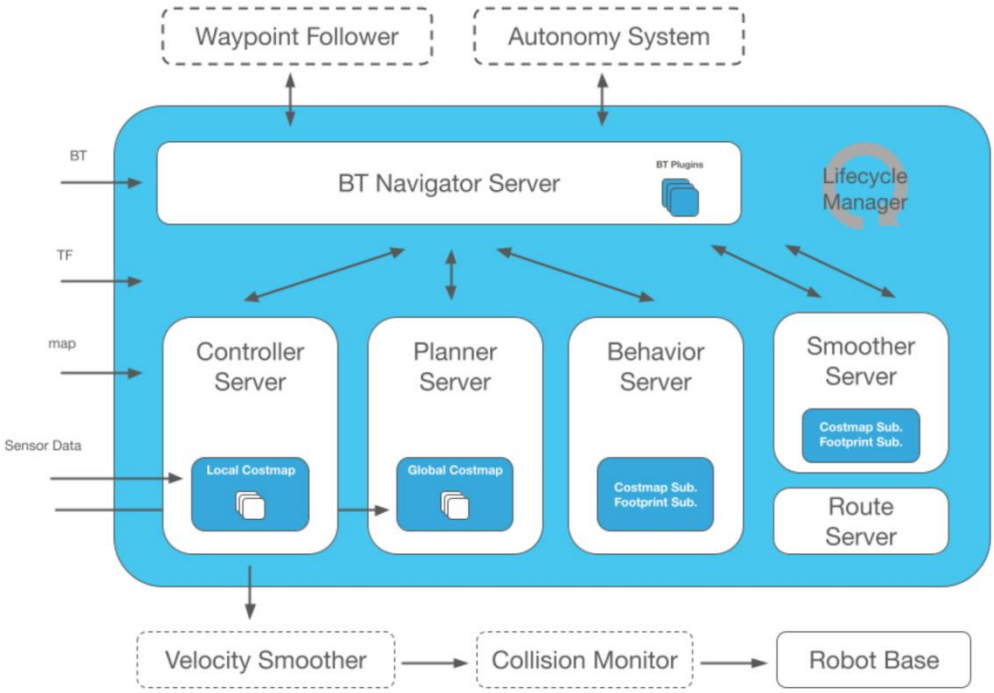
1. Comparison of Navigation Frameworks between ROS1 and ROS2

In Nav2, the "move_base" component in ROS1 has been split into multiple components. Unlike a single state machine in ROS1, Nav2 utilizes action servers and the low-latency, reliable communication of ROS2 to separate the concepts. Behavior Trees (BT) are used to orchestrate and organize these tasks, allowing Nav2 to have highly configurable navigation behavior without the need for programming, thanks to the BT XML file.

The "nav2_bt_navigator" replaces "move_base" at the top level in Nav2. It uses an Action interface to accomplish navigation tasks based on a tree-based action model. BT is used to implement more complex state machines and add recovery behavior as additional Action Servers. These behavior trees are configurable via XML.

The planning, recovery, and controller servers, responsible for path planning, recovery, and control, are also implemented as action servers. The BT Navigator can invoke these servers for computations. All three servers can host multiple plugins for various algorithms, and each plugin can be individually called from the navigation behavior tree to perform specific actions. These servers are invoked by the BT Navigator through their action servers to compute results or complete tasks.

2. Navigation2 frame work as below image:



Nav2 provides the following tools:

- Map loading, serving, and storage (Map Server)
- Robot localization on the map (AMCL)
- Path planning from point A to point B around obstacles on the map (Nav2 Planner)
- Robot control along the planned path (Nav2 Controller)
- Conversion of sensor data to costmap representation of the world (Nav2 Costmap 2D)
- Construction of complex robot behaviors using behavior trees (Nav2 Behavior and BT Navigator)
- Recovery behaviors to handle failures (Nav2 Recoveries)
- Waypoint following along a series of path points (Nav2 Waypoint Follower)
- Lifecycle management of the servers (Nav2 Lifecycle Manager)
- Customization of algorithms and behaviors through plugins (Nav2 Core)

# Experiment 9: Point-Based Autonomous Navigation in Rviz2

## 1. 2D Navigation

### 1.1. Map Selection and Initialization

(1) When you are in a new area and have successfully built a new map and saved it, if you want to use this map for navigation, you need to make modifications in the launch file. Navigate to the "/home/yhs/ros2_ws/src/yhs_nav2/launch" directory and locate the "yhs_nav2_2d.launch.py" file.

```
yhs@yhs-ros2:~/ros2_ws/src/yhs_nav2/launch$ ls
cartographer.launch.py  localization_2d_launch.py  navigation_launch.py   yhs_nav2.launch.py
gmapping.launch.py      localization_launch.py      yhs_nav2_2d.launch.py
yhs@yhs-ros2:~/ros2_ws/src/yhs_nav2/launch$
```

(2) You can open the file using either "gedit" or "vim". Once the file is open, locate the position shown in the image below, and make the necessary modifications following the instructions in the image. After making the changes, save the file.

```
67
68      #选择，设置地图
69      declare_map_yaml_cmd = DeclareLaunchArgument(
70          'map',
71          default_value=os.path.join(bringup_dir, 'map', 'g1.yaml'),
72          description='地图名称')
73
```

选择地图，保存地图会生成两个文件，一个是"地图名.pgm"和“地图名.yaml”，这里修改为“地图名.yaml”

(3) Once you have selected the map, you can proceed to start the navigation. Open a terminal and enter the command, then press Enter. If you see the message "Navigation started successfully..." after a while, it means that the navigation has been successfully started. If not, you should check the information displayed in the terminal output to identify and troubleshoot any issues.

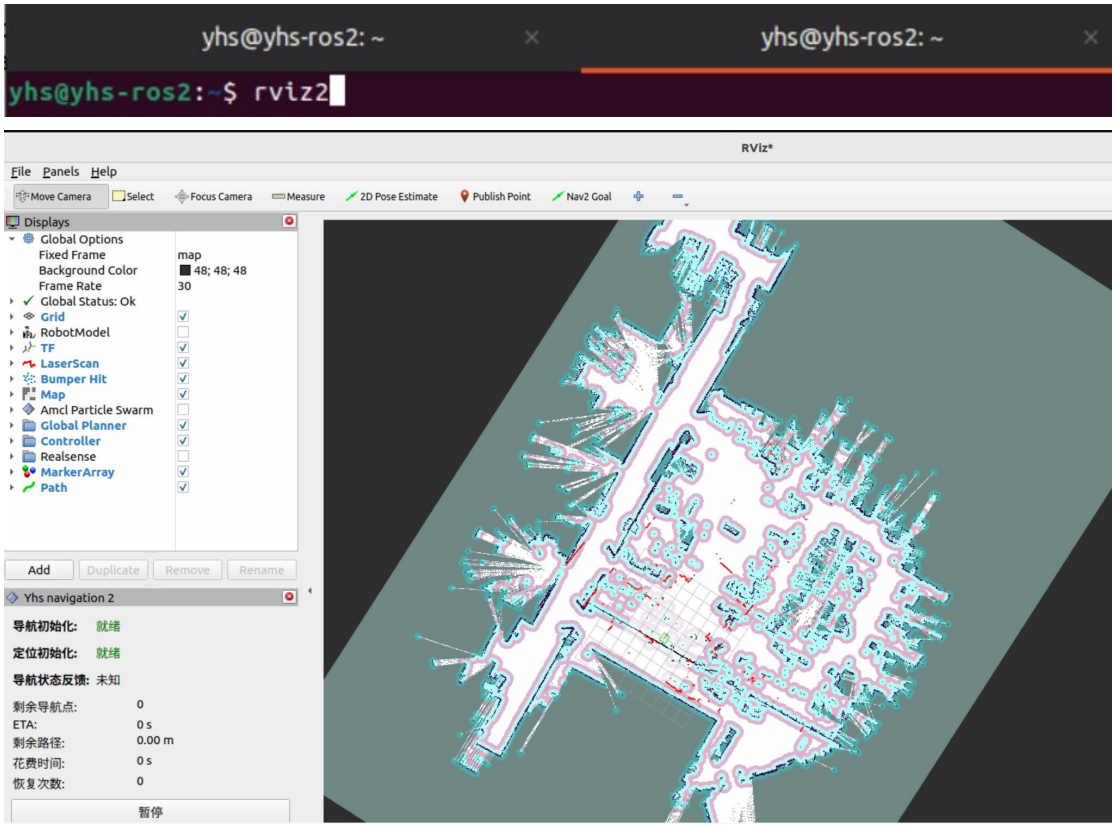
```
ros2 launch yhs_nav2 yhs_nav2_2d.launch.py
```

```
yhs@yhs-ros2:~$ ros2 launch yhs_nav2 yhs_nav2_2d.launch.py
```

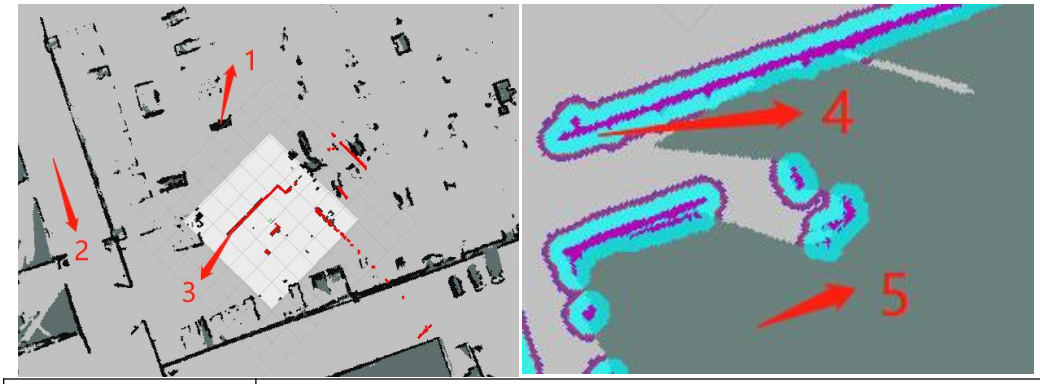
```
yhs@yhs-ros2: ~
[lifecycle_manager-20] [INFO] [1701917397.487586143] [lifecycle_manager_navigation]: Server behavior_server connected with bond.
[lifecycle_manager-20] [INFO] [1701917397.487696720] [lifecycle_manager_navigation]: Activating bt_navigator
[lifecycle_manager-20] [INFO] [1701917397.488012005] [lifecycle_manager_navigation]: bt_navigator/change_state service client: send async reque
[bt_navigator-17] [INFO] [1701917397.488507100] [bt_navigator]: Activating
[bt_navigator-17] load tree xml:  /home/yhs/ros2_ws/install/nav2_bt_navigator/share/nav2_bt_navigator/behavior_trees/navigate_follow_record_pat
and_recovery.xml
[bt_navigator-17] load tree xml:  /home/yhs/ros2_ws/install/nav2_bt_navigator/share/nav2_bt_navigator/behavior_trees/navigate_through_poses_w_r
ecovery.xml
[bt_navigator-17] load tree xml:  /home/yhs/ros2_ws/install/nav2_bt_navigator/share/nav2_bt_navigator/behavior_trees/navigate_to_pose_w_replann
y.xml
[bt_navigator-17] [INFO] [1701917397.904272858] [bt_navigator]: Creating bond (bt_navigator) to lifecycle manager.
[lifecycle_manager-20] [INFO] [1701917398.020475464] [lifecycle_manager_navigation]: Server bt_navigator connected with bond.
[lifecycle_manager-20] [INFO] [1701917398.020539235] [lifecycle_manager_navigation]: Activating waypoint_follower
[lifecycle_manager-20] [INFO] [1701917398.020825507] [lifecycle_manager_navigation]: waypoint_follower/change_state service client: send async
[waypoint_follower-18] [INFO] [1701917398.021210121] [waypoint_follower]: Activating
[waypoint_follower-18] [INFO] [1701917398.021265015] [waypoint_follower]: Creating bond (waypoint_follower) to lifecycle manager.
[lifecycle_manager-20] [INFO] [1701917398.135354472] [lifecycle_manager_navigation]: Server waypoint_follower connected with bond.
[lifecycle_manager-20] [INFO] [1701917398.135408550] [lifecycle_manager_navigation]: Activating velocity_smoother
[lifecycle_manager-20] [INFO] [1701917398.135720781] [lifecycle_manager_navigation]: velocity_smoother/change_state service client: send async
[velocity_smoother-19] [INFO] [1701917398.136026881] [velocity_smoother]: Activating
[velocity_smoother-19] [INFO] [1701917398.136113528] [velocity_smoother]: Creating bond (velocity_smoother) to lifecycle manager.
[lifecycle_manager-20] [INFO] [1701917398.253592096] [lifecycle_manager_navigation]: Server velocity_smoother connected with bond.
[lifecycle_manager-20] [INFO] [1701917398.253680324] [lifecycle_manager_navigation]: Managed nodes are active
[lifecycle_manager-20] [INFO] [1701917398.253708135] [lifecycle_manager_navigation]: Creating bond timer...
[lifecycle_manager-20] [INFO] [1701917398.253732324] [lifecycle_manager_navigation]: 导航启动成功。。。。。。
```

(4) Open another terminal and enter the command, then press Enter. This will launch the rviz2 interface, where you will be able to visualize the map, laser data, and other relevant information.

| rviz2 |
|---|



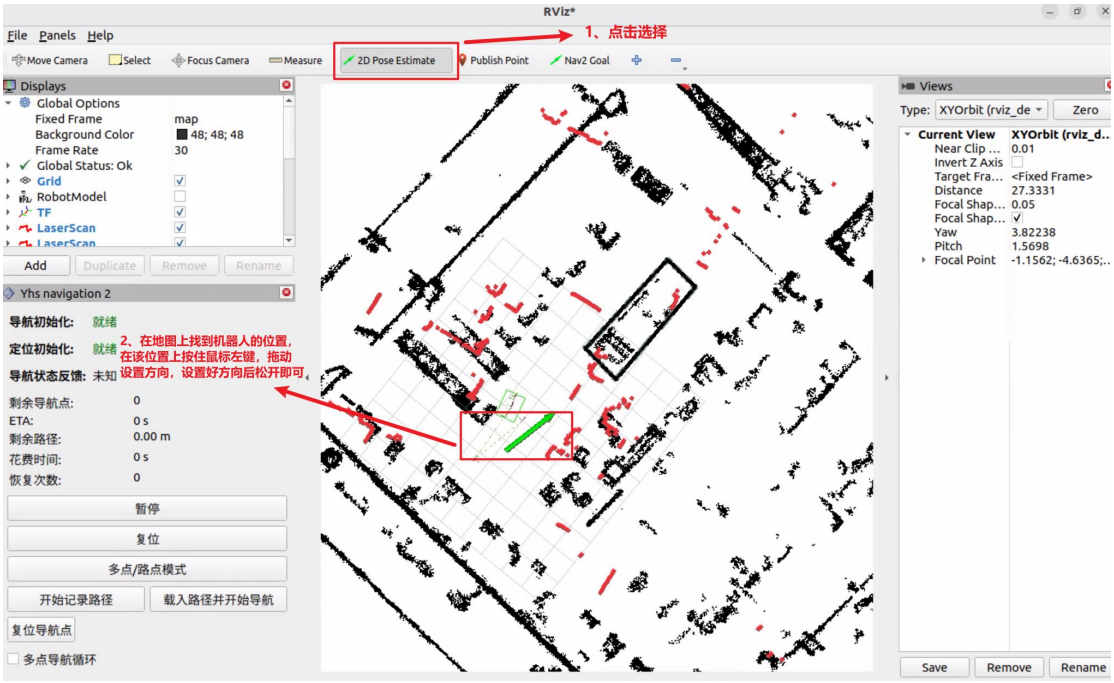

The meanings of various colors on the map are as follows:

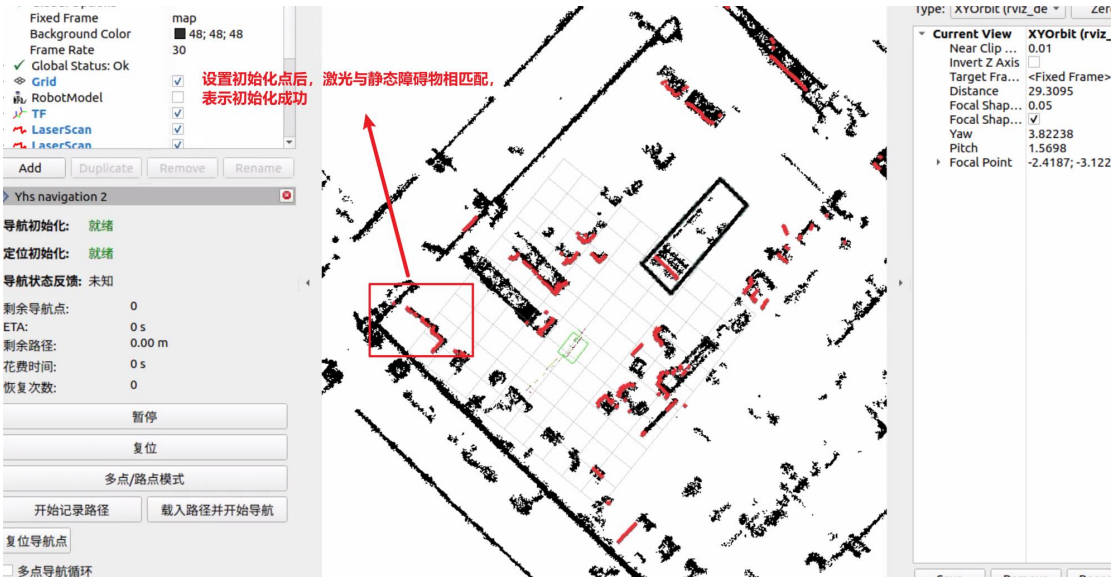



| | |
|---|---|
| 1 | Black area: Represents static obstacles. |
| 2 | Light gray area: Represents the traversable or free space. |
| 3 | Red color: Represents laser data. |
| 4 | Blue area: Represents the inflated or expanded region. |
| 5 | Dark gray area: Represents unknown or unexplored regions. |

For the upcoming navigation, you can configure Rviz2 to hide the inflated area layer.

(5) Setting an initial pose: Before starting the navigation, the robot's position is typically not at the starting point of the map. Therefore, it is necessary to set an initial pose. Follow the steps shown in the image below to set the initial pose. Once set, observe if the red laser points roughly correspond to the surrounding black static obstacles. If there is a significant discrepancy, you can set the initial pose again.

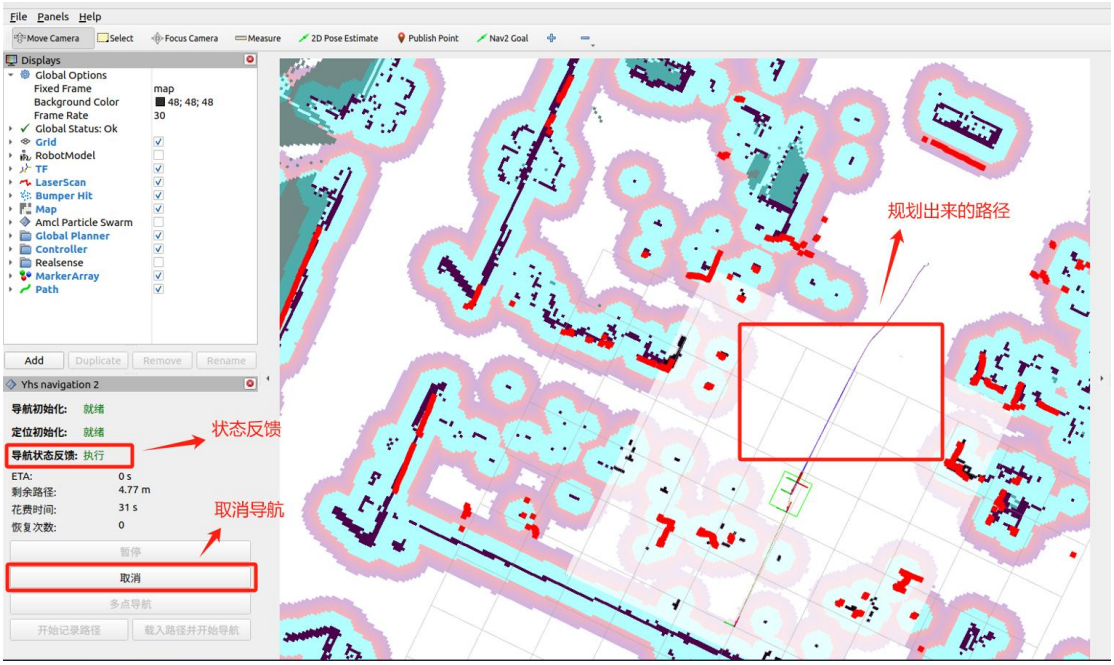


## 1.2. Single Point Navigation

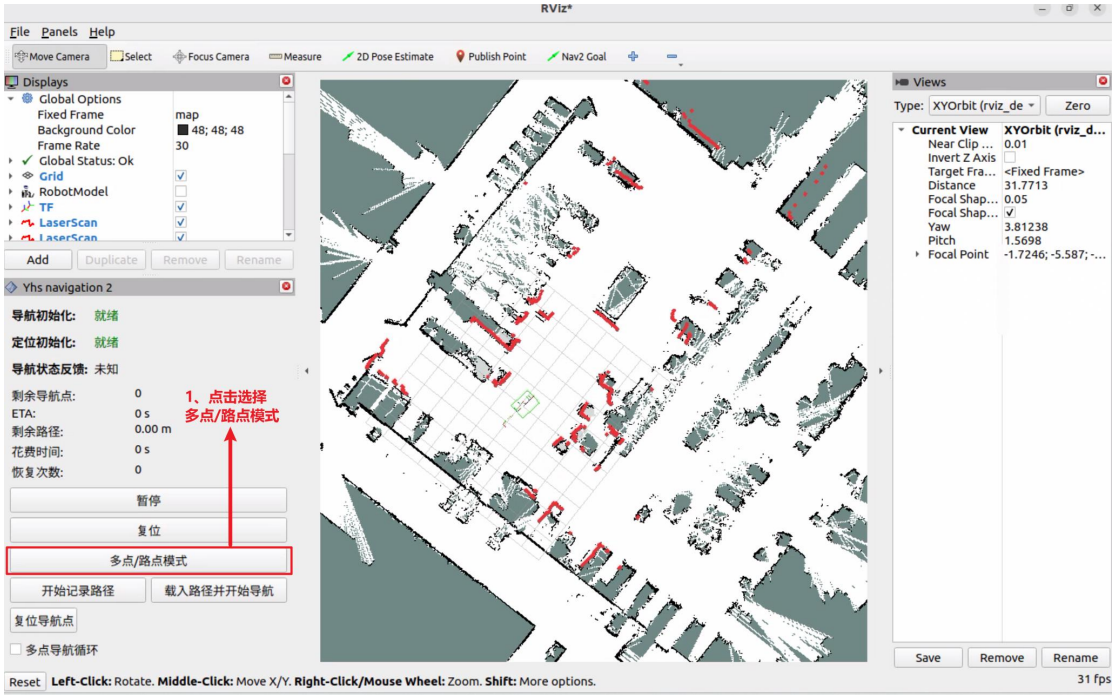
(1)Operation according to below image

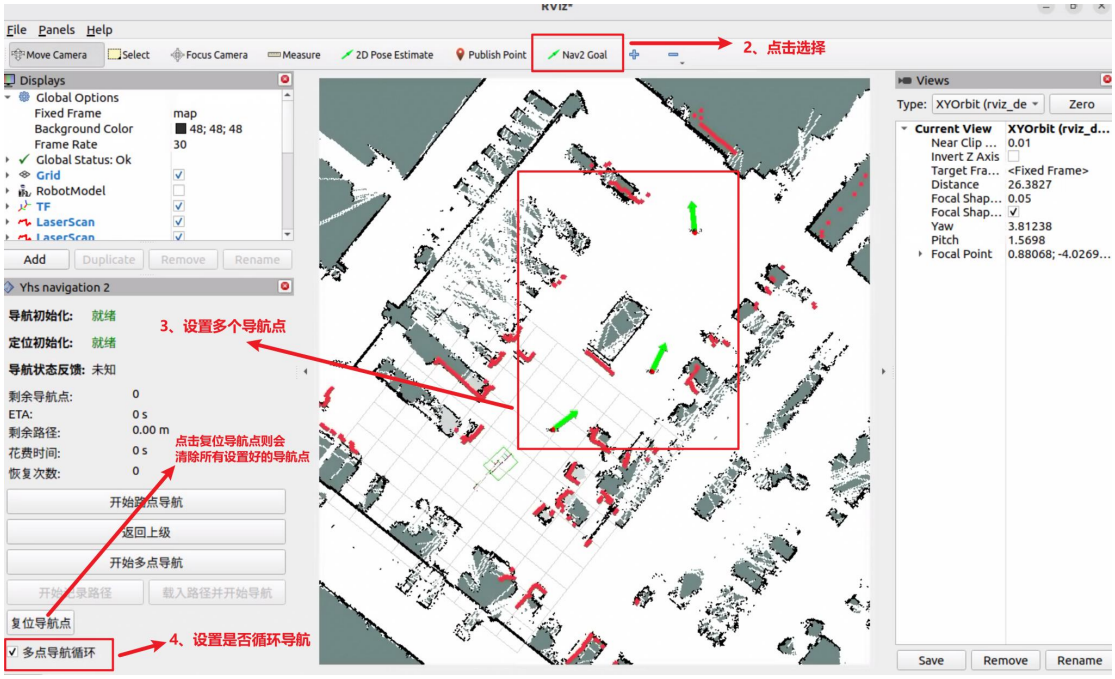

(3) After releasing the mouse, you will be able to see the planned path. Switch the remote controller to the command mode, and the robot will follow the path to reach the target point. The bottom-left corner will provide navigation status feedback. You can also cancel the navigation if needed.



### 1.3. Multiple Points Navigation

(1)Operation according to below image



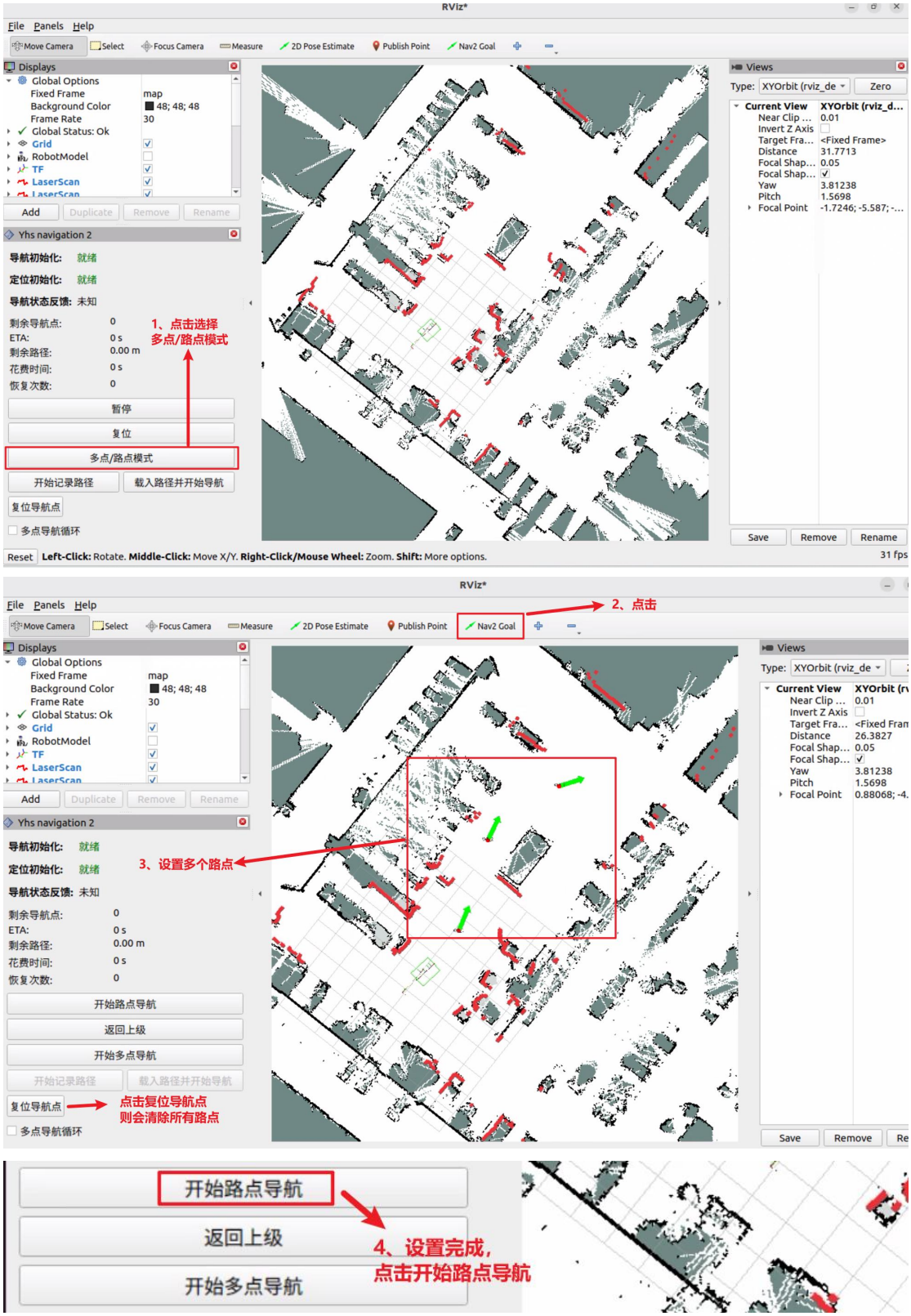





(2)After clicking on "Start Multi-Point Navigation," the robot will sequentially reach each navigation point. Once it reaches the last navigation point, if the option for "Loop Navigation" is selected, the robot will start the next round of navigation. During the navigation process, clicking on "Cancel" will stop the navigation.

### 1.4. Road Points Navigation

(1)Operation according to below image

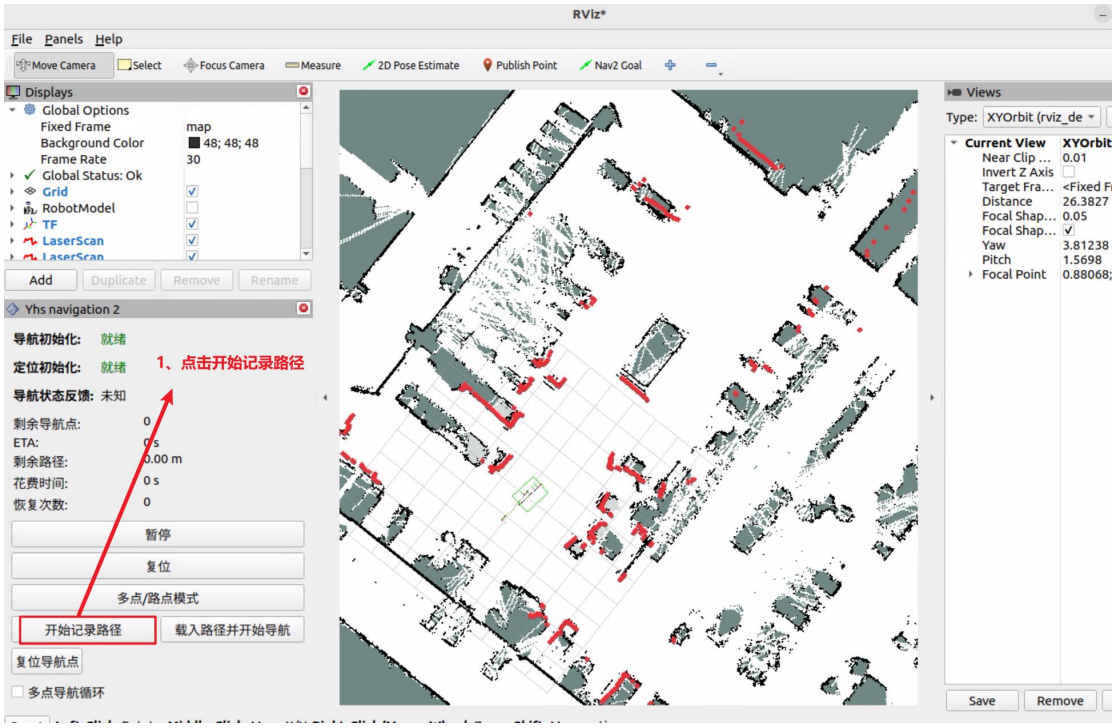



(2) After setting multiple waypoints, click on "Start Waypoint Navigation." The robot will navigate from the first point to the last point, planning paths between adjacent points to

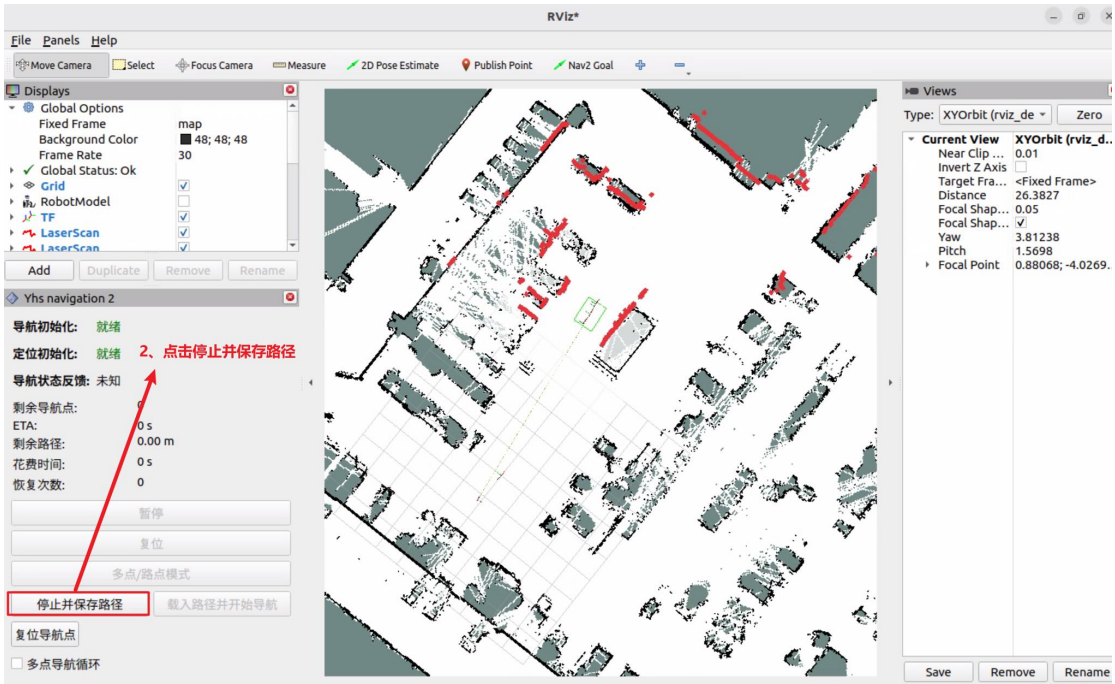
form a global path. The robot will then follow this global path during navigation. Clicking on "Cancel" during the navigation process will stop the navigation.

### 1.5. Path Recording Navigation

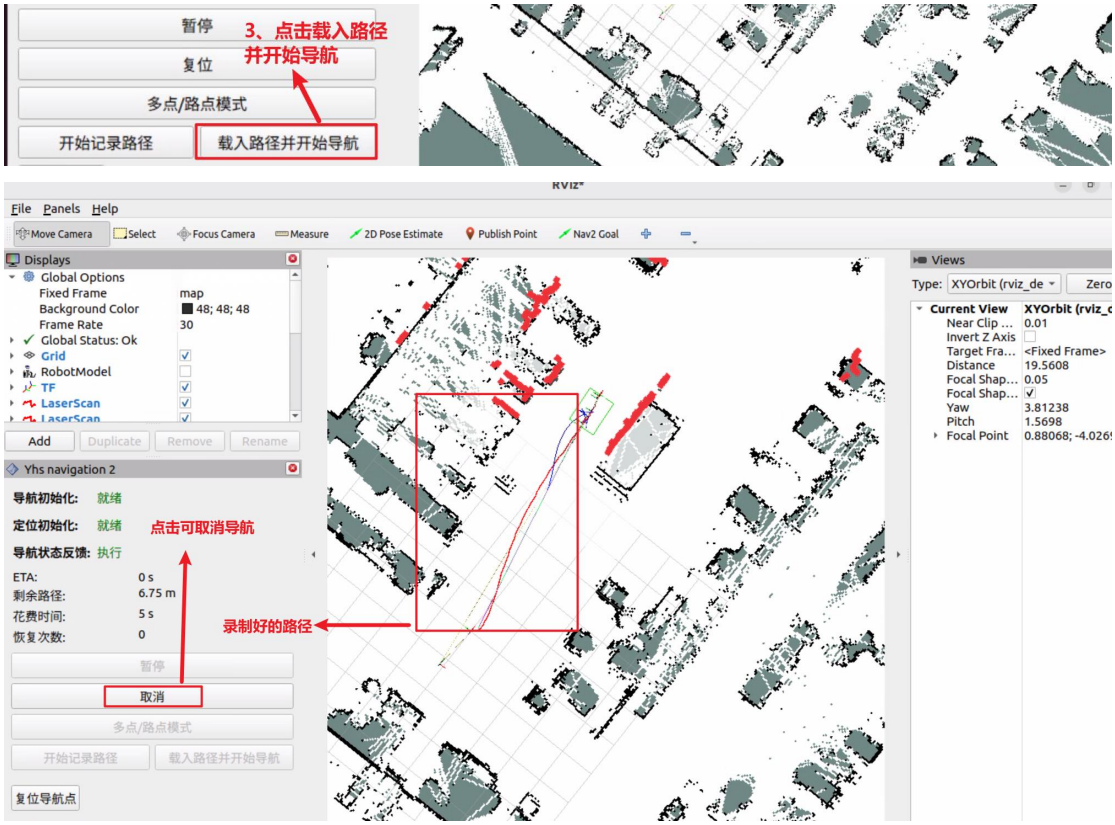
(1)Operation according to below image




(2) Now you can remotely control the robot's movement. The program will continuously record the path in real-time. It is important to ensure that the path does not have any intersections. To save the path and overwrite the previous one, click on "Stop and Save Path."



(3) After saving the path, click on "Load Path and Start Navigation." The robot will first navigate to the starting point of the path. Then, it will follow the recorded path as the global path, proceeding towards the last point. During this navigation, it is possible to cancel the process at any point.



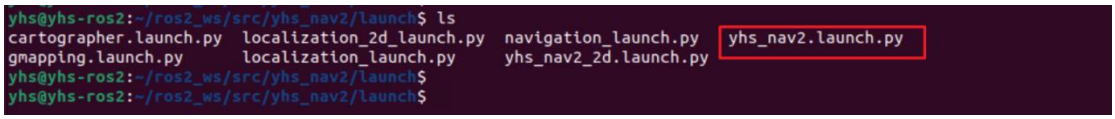
# 2. 3D Navigation

## 2.1. Remote Control and Rviz2 Display Configuration

When loading a large-area map for navigation, it is recommended to use another computer to open Rviz2 and send navigation points. Please refer to Appendix 1 for more details. On the computer running Rviz2, the PCD point cloud map and global inflation layer are not displayed to reduce the CPU usage on the industrial computer.

## 2.2. Map Selection and Initialization

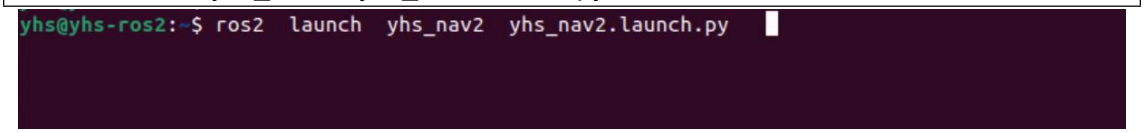
(1) If you have created and saved a new map in a new area and want to use it for navigation, you need to make modifications in the launch file. Navigate to the "/home/yhs/ros2_ws/src/yhs_nav2/launch" directory and locate the "yhs_nav2.launch.py" file.

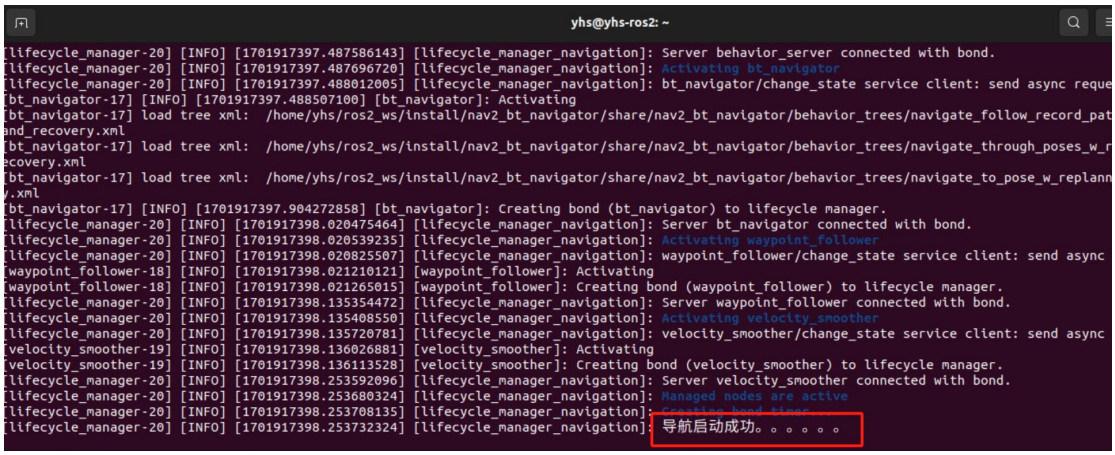



(2) You can open the file using either "gedit" or "vim." Once the file is open, locate the position as shown in the image below. Modify the file according to the instructions in the image and save the changes.


(3) After selecting the map, you can start the navigation process. Open a terminal and enter the command, then press Enter. If you see the message "Navigation started successfully..." after that, it means the navigation has been successfully initiated. Otherwise, you need to check the information displayed in the terminal and investigate any potential issues.

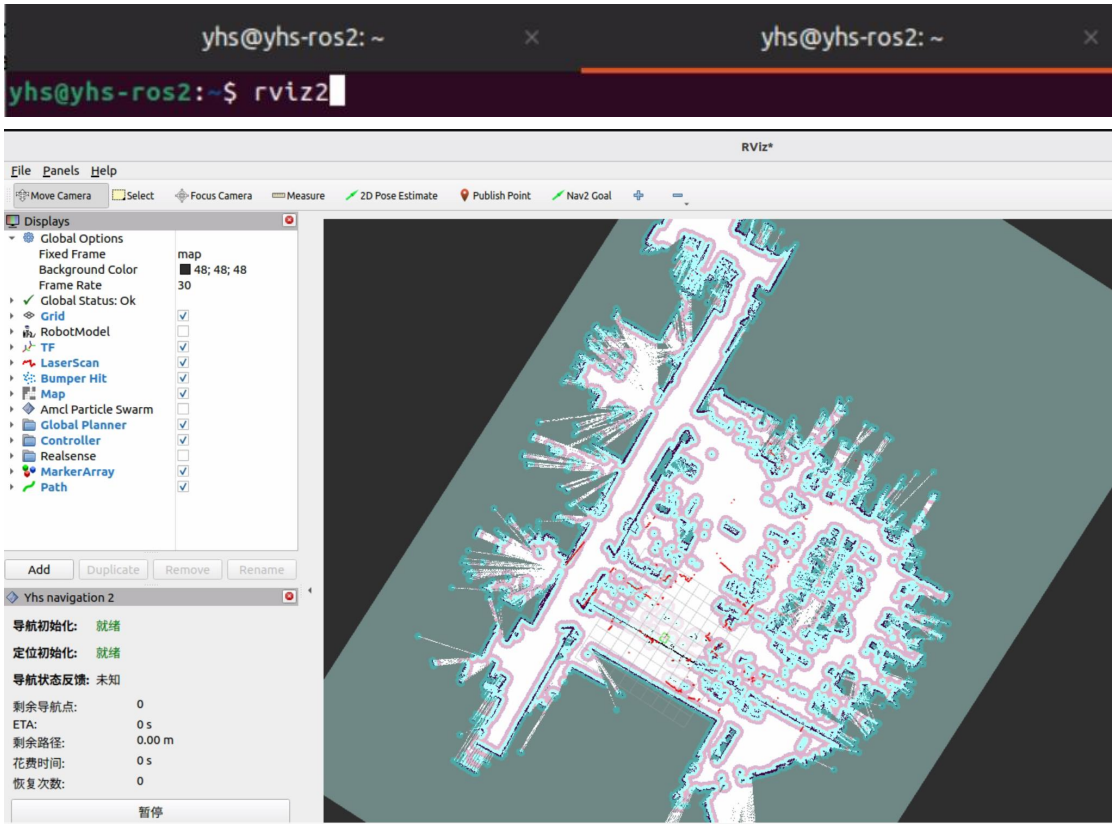
```
ros2 launch yhs_nav2 yhs_nav2.launch.py
```



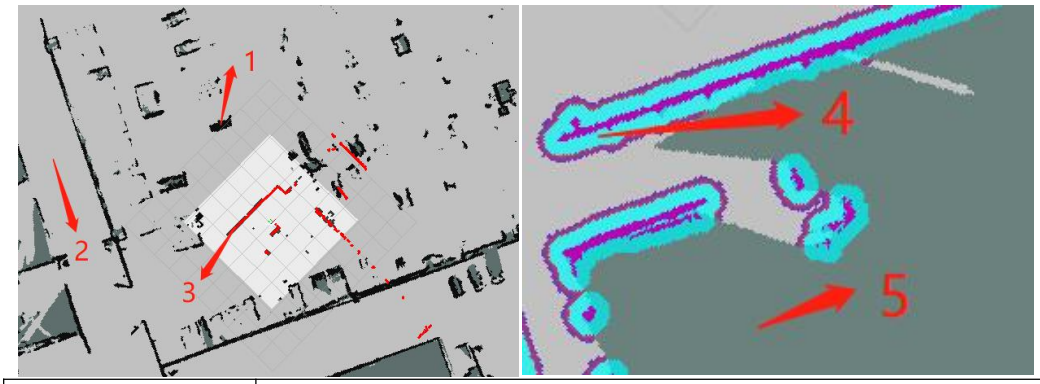



(4) Open another terminal and enter the command, then press Enter. This will launch the Rviz2 interface, where you can view the map, laser data, and other information.

```
rviz2
```



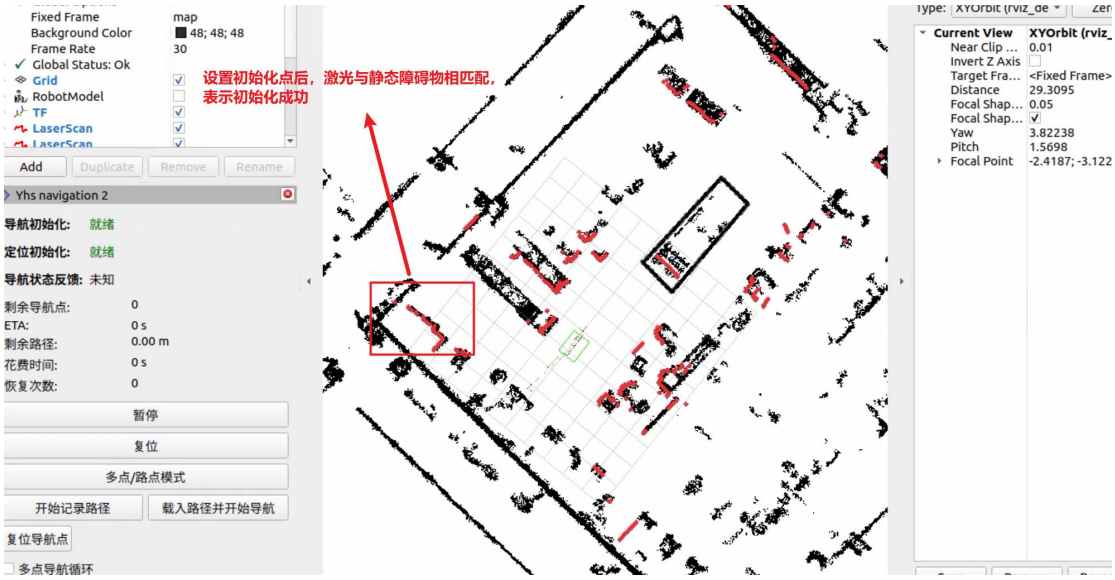

The meanings of various colors on the map are as follows:




| | |
|---|---|
| 1 | Black color: Represents static obstacles. |
| 2 | Light gray color: Represents traversable or free space. |
| 3 | Red color: Represents laser data. |
| 4 | Blue color: Represents inflated or expanded areas. |
| 5 | Dark gray color: Represents unknown or unexplored areas. |

For the upcoming navigation, you can configure Rviz2 to hide the inflated area layer.

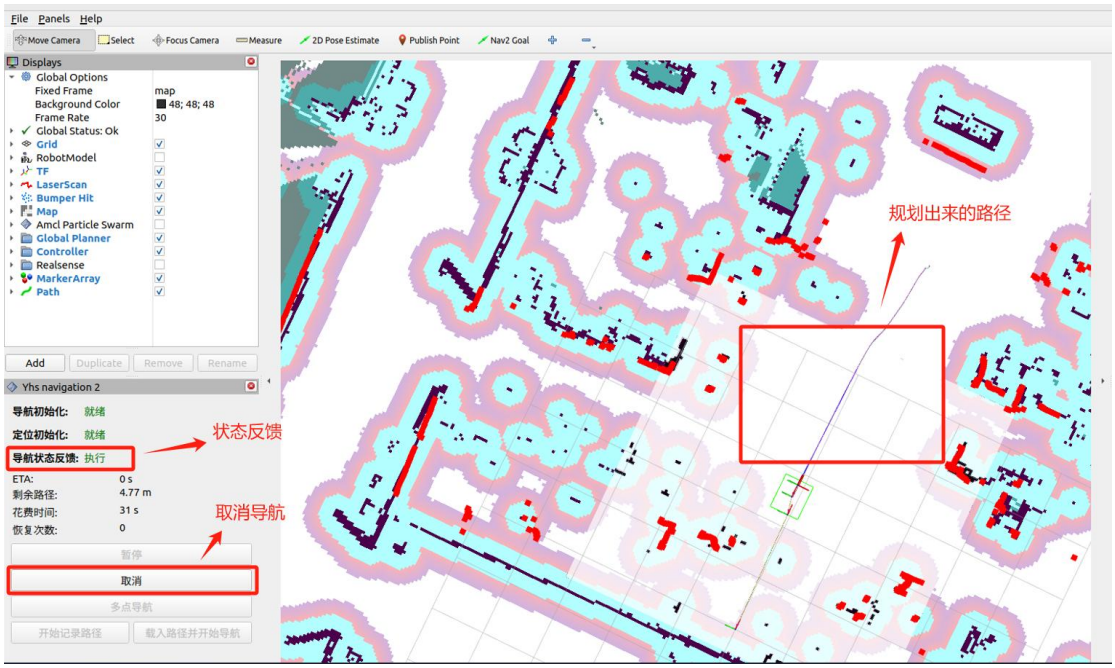
(5) Setting an initial pose: Before starting the navigation, the robot's position is typically not at the starting point of the map. Therefore, it is necessary to set an initial pose. Follow the steps shown in the image below to set the initial pose. After setting it, observe if the red laser points match the surrounding black static obstacles. If they do not match, set the initial pose again.




## 2.3. Single Point Navigation

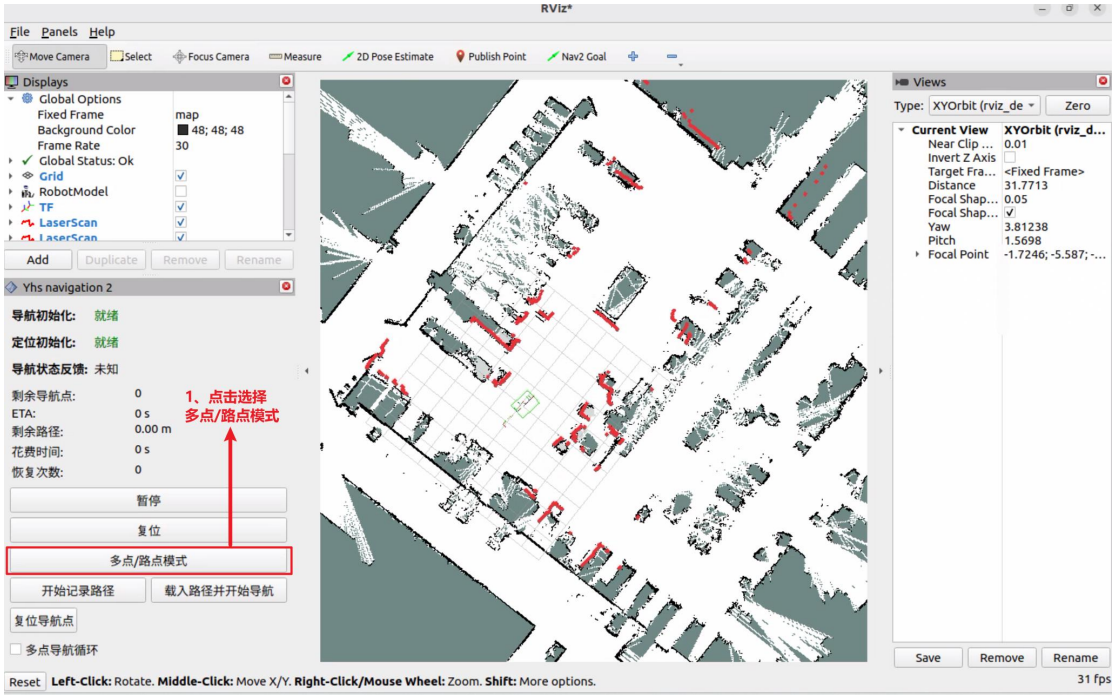
(1) Follow the steps shown in the image below.

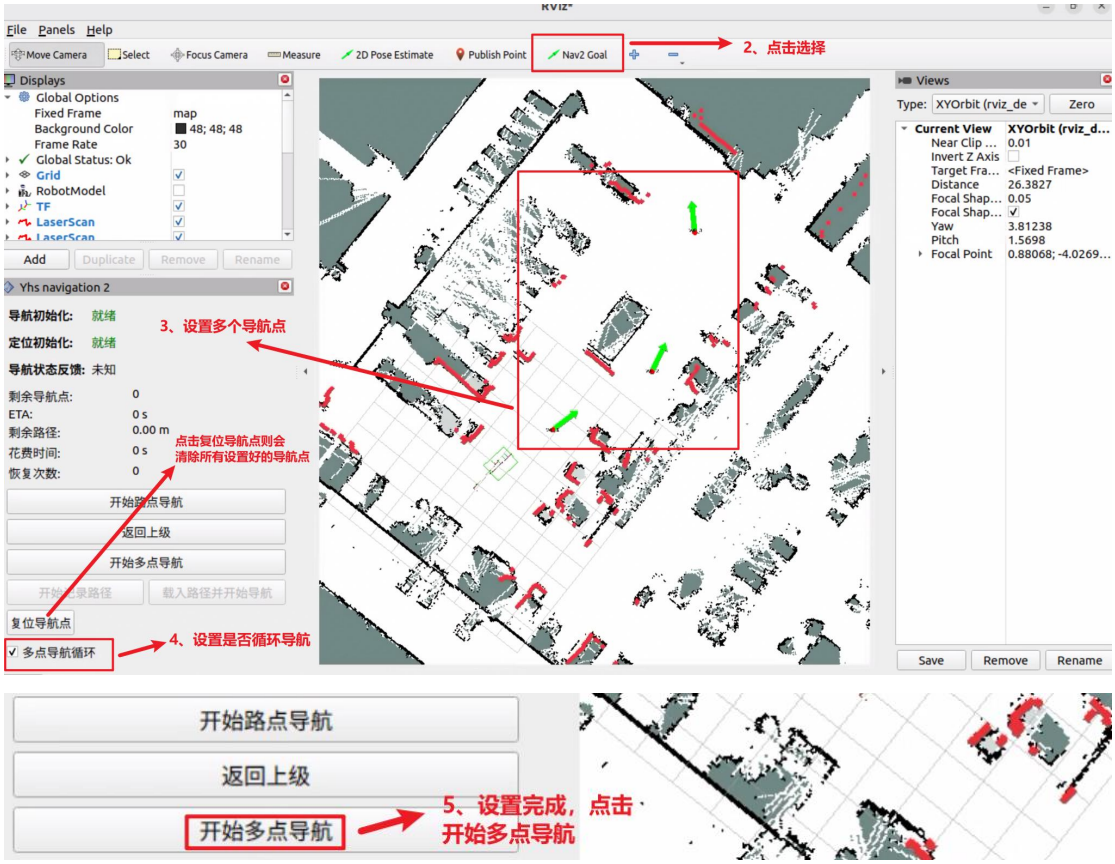

(2) After releasing the mouse control, you will be able to see the planned path. Switch the remote controller to the command mode, and the robot will follow the path to reach the target point. There will be navigation status feedback in the bottom-left corner, and you can also cancel the navigation if needed.




## 2.4. Multiple Points Navigation

(1) Follow the steps shown in the image below.





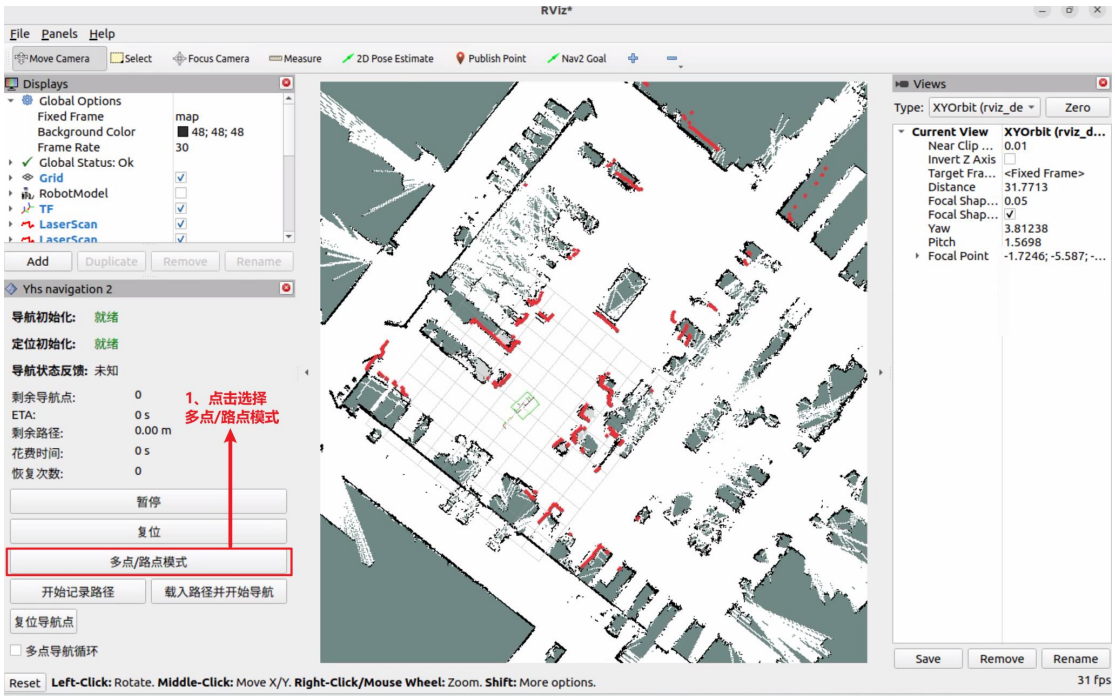




(2) After clicking on "Start Multi-Point Navigation," the robot will sequentially reach each navigation point. Once it reaches the last navigation point, if the option for "Loop Navigation" is selected, the robot will start the next round of navigation. During the navigation process, clicking on "Cancel" will stop the navigation.

## 2.5. Road Points Navigation

(1) Follow the steps shown in the image below.

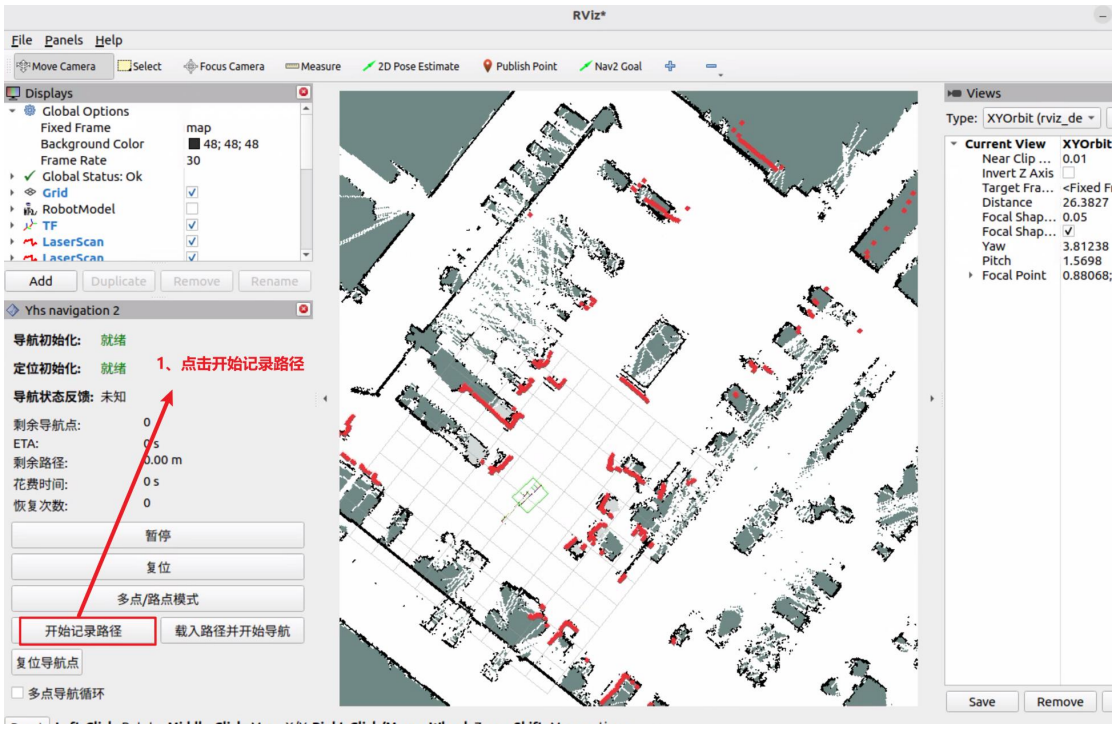




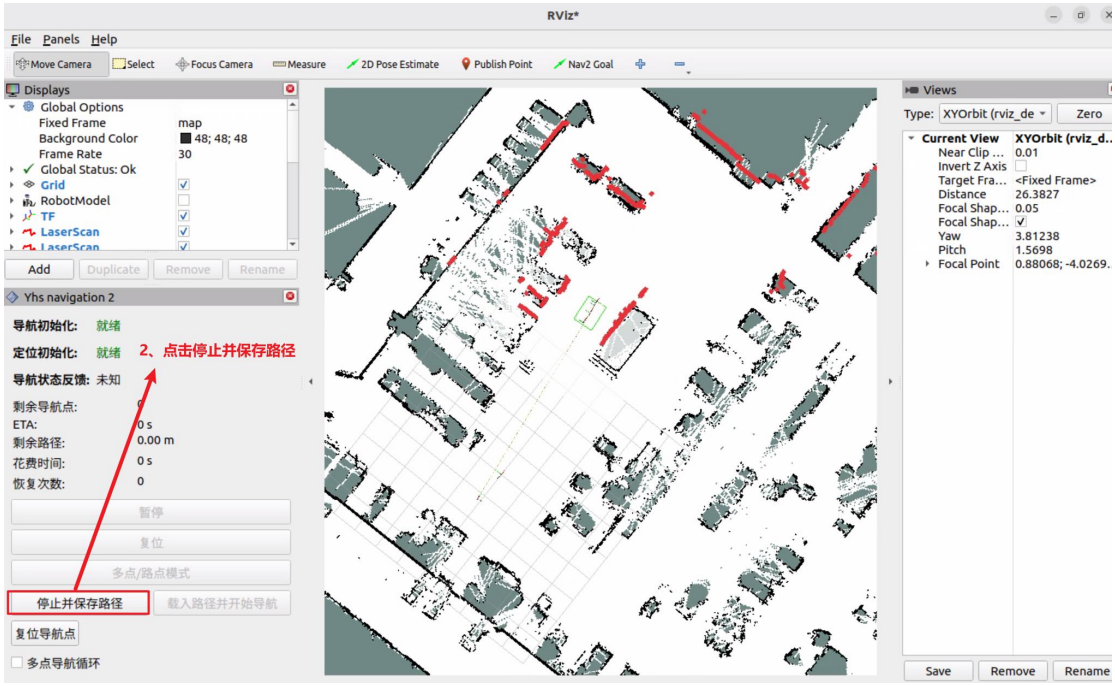

(2) After setting multiple waypoints, click on "Start Waypoint Navigation." The robot will navigate from the first point to the last point by planning paths between adjacent points. These paths will be combined to form a global path. The robot will follow this global path during navigation. Clicking on "Cancel" during the navigation process will stop the navigation.

## 2.6. Path Recording Navigation

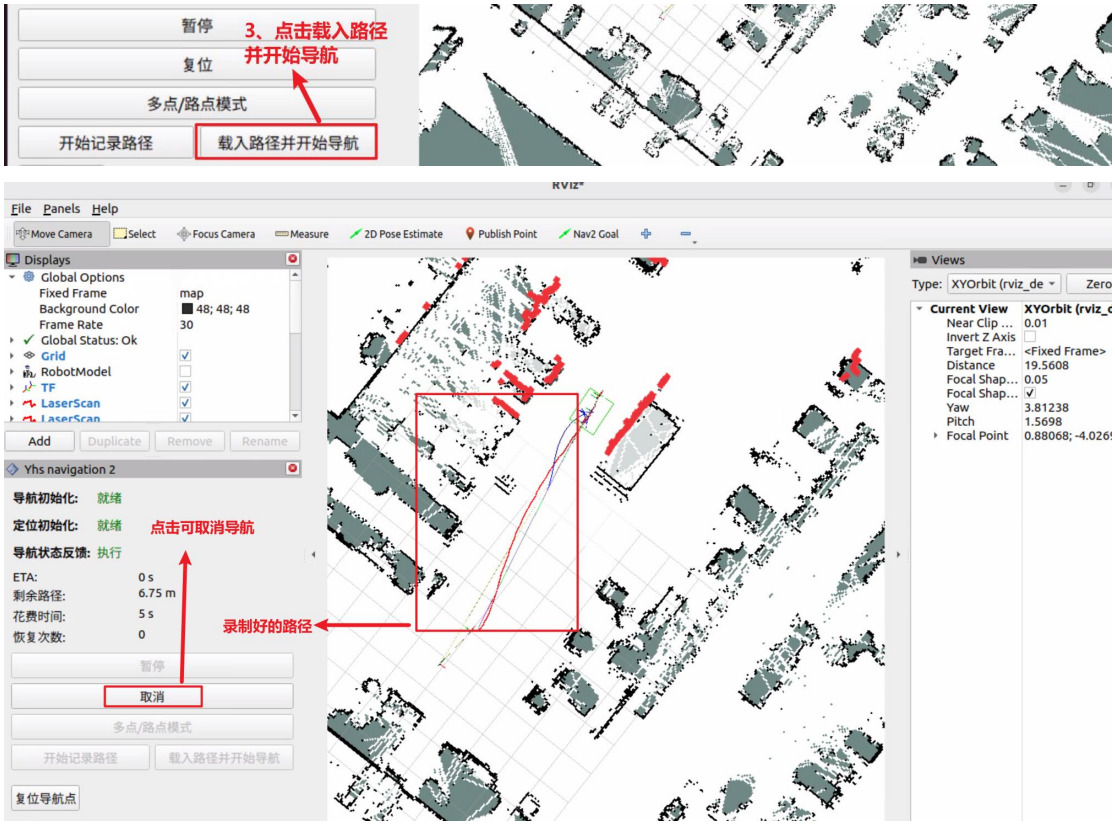
(1) Follow the steps shown in the image below.




(2) Now you can remotely control the robot's movement. The program will record the path in real-time, ensuring that the path does not have any intersections. After clicking on "Stop and Save Path," you can save the path, overwriting the previous one.



(3) After saving the path, click on "Load Path and Start Navigation." The robot will first navigate to the starting point of the path. Then, it will use the recorded path as the global path and follow it to the last point. During this navigation, it is possible to cancel the process at any point.



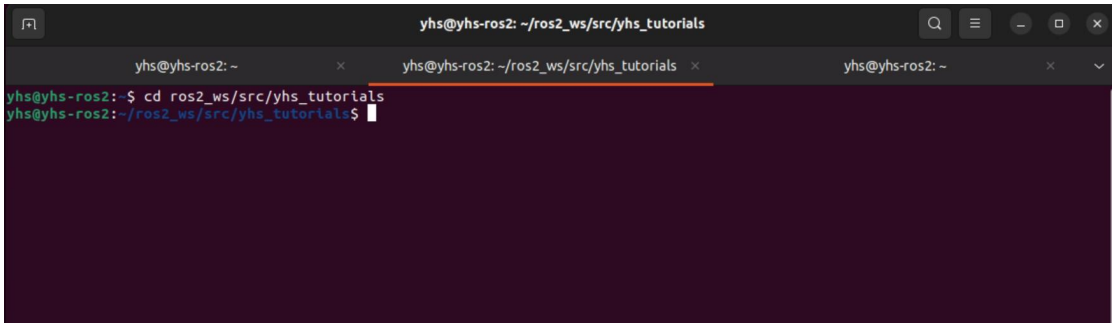
# Experiment 10: Programming Autonomous Navigation Nodes

In the previous section, we used graphical operations in Rviz2 to navigate the robot. In this section, we will write code to achieve the same process.

1. Creating a ROS2 source package

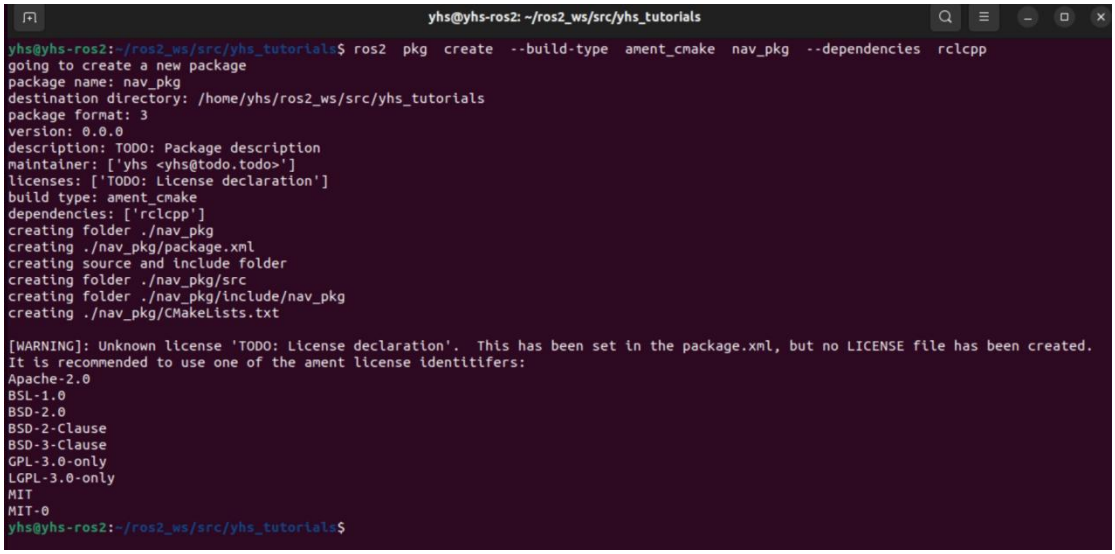
(1) Open a terminal and enter the command to navigate to the directory "cd ros2_ws/src/yhs_tutorials".

```
cd  ros2_ws/src/yhs_tutorials
```




(2) Use the following command to create a new ROS2 package:

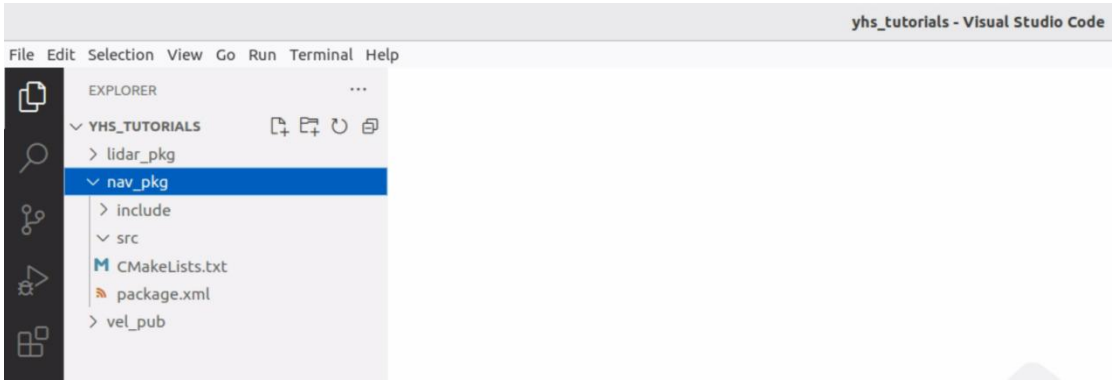
```
ros2  pkg  create  --build-type  ament_cmake  nav_pkg  --dependencies  rclcpp
```

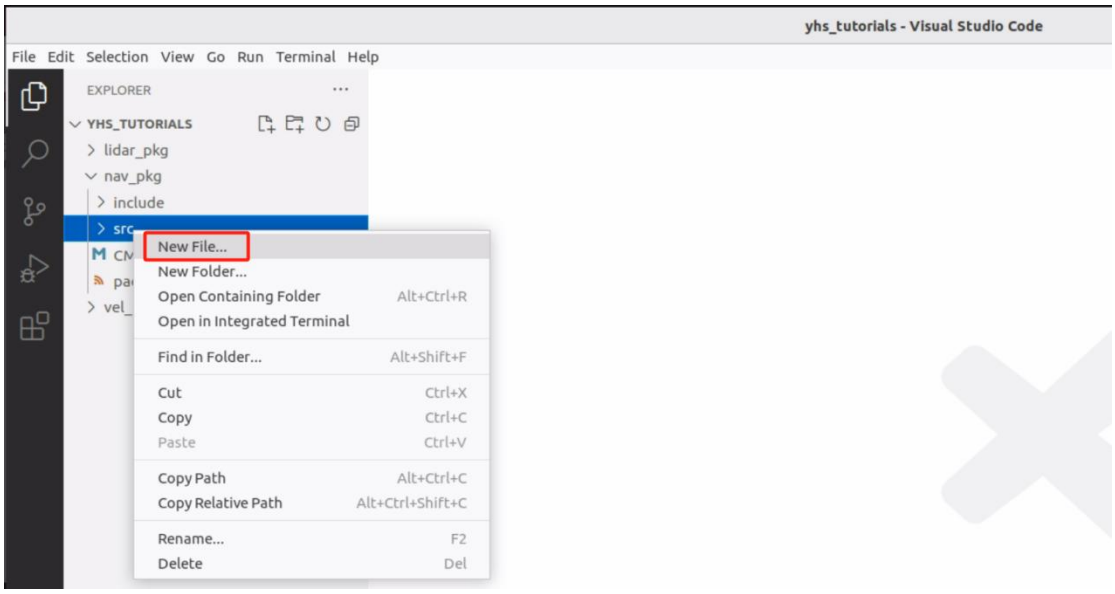



The meaning of this command:

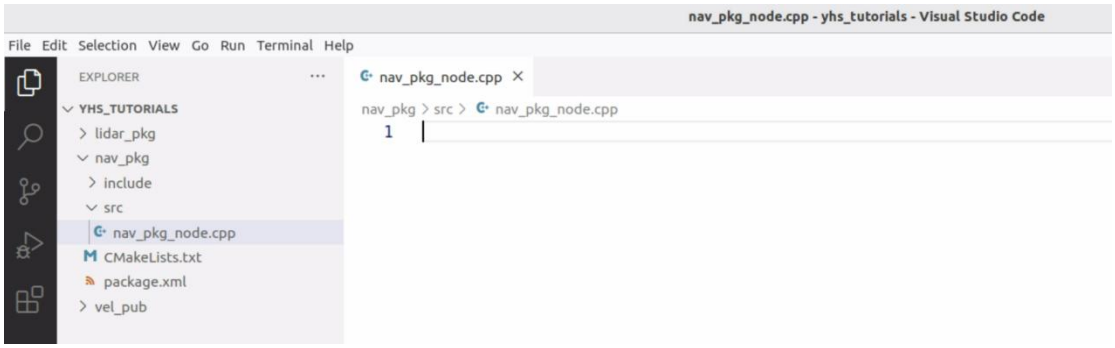
| Command | Meaninh |
|---|---|
| ros2 pkg create | creating source code package of ROS2 |
| --build-type ament_cmake | package creating and managing package by using ament_cmake |
| nav_pkg | the name of new creating ROS2 source code package |
| --dependencies | designated dependencies |
| roscpp | C++ dependencies: This example is written in C++, so this dependency is required. |

(3) After creating the "nav_pkg" package, open the "yhs_tutorials" directory in an IDE. You will notice that a new folder named "nav_pkg" has been added under the "yhs_tutorials" directory. Right-click on the "src" subfolder and select "New File" to create a new code file. (In this case, we are using Visual Studio Code, but you can also use other text editing tools such as Vim or Gedit to open or create the file.)




(4) The newly created code file should be named "nav_pkg_node.cpp".



(5) After naming the file, you can start writing the code for "nav_pkg_node.cpp" on the right side of the IDE. The content of the file should be as follows:

```
#include "rclcpp/rclcpp.hpp"
#include "rclcpp_action/rclcpp_action.hpp"
#include "nav2_msgs/action/navigate_to_pose.hpp"

#include "tf2/utils.h"
#include "tf2_geometry_msgs/tf2_geometry_msgs.hpp"
#include "tf2_ros/buffer.h"
#include "tf2_ros/transform_broadcaster.h"
#include "tf2_ros/transform_listener.h"

class NavigationClient : public rclcpp::Node
{
public:
  NavigationClient() : Node("send_goal")
  {
    // Create a timer that executes the callback function every 200 milliseconds.
    timer_ = this->create_wall_timer(std::chrono::milliseconds(200),
std::bind(&NavigationClient::timerCallback, this));

    // Create a TF2 listener.
    tf_buffer_ = std::make_shared<tf2_ros::Buffer>(this->get_clock());
    tf_listener_ = std::make_shared<tf2_ros::TransformListener>(*tf_buffer_);
  }

  // Used to send a navigate_to_pose request.
  void sendNavigateToPoseRequest()
  {
    // Create a navigate_to_pose client.
    action_client_ =
rclcpp_action::create_client<nav2_msgs::action::NavigateToPose>(this,
"navigate_to_pose");

    // Wait for the navigate_to_pose server connection.
    auto is_action_server_ready =
action_client_->wait_for_action_server(std::chrono::seconds(5));
    if (!is_action_server_ready)
    {
      RCLCPP_ERROR(
          this->get_logger(),
          "navigate_to_pose action server is not available.");
      return;
    }
```

```
    // creating request message
    auto goal_msg = nav2_msgs::action::NavigateToPose::Goal();
    // Set the target point as 1 meter ahead of the base_link position.
    goal_msg.pose.header.stamp = this->now();
    goal_msg.pose.header.frame_id = "map"; // Choose between base_link or map. If you select base_link, after running the program, immediately switch the remote controller to command mode.
    goal_msg.pose.pose.position.x = 1.0;
    goal_msg.pose.pose.position.y = 0.0;
    goal_msg.pose.pose.orientation.w = 1.0;

    navigation_goal_ = goal_msg;

    auto send_goal_options = rclcpp_action::Client<nav2_msgs::action::NavigateToPose>::SendGoalOptions();
    send_goal_options.result_callback = [this](auto)
    {
      RCLCPP_INFO(this->get_logger(), "arrive");
      navigation_goal_handle_.reset();
      is_running_ = false;
    };

    // send
    auto future_goal_handle = action_client_->async_send_goal(navigation_goal_, send_goal_options);
    if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), future_goal_handle, std::chrono::seconds(2)) !=
        rclcpp::FutureReturnCode::SUCCESS)
    {
      RCLCPP_ERROR(this->get_logger(), "Send goal call failed");
      return;
    }

    navigation_goal_handle_ = future_goal_handle.get();
    if (!navigation_goal_handle_)
    {
      RCLCPP_ERROR(this->get_logger(), "Goal was rejected by server");
      return;
    }

    is_running_ = true;
  }
```

```
  bool isRunning()
  {
    return is_running_;
  }

  void cancelGoal()
  {
    auto status = navigation_goal_handle_->get_status();
    if ((status == action_msgs::msg::GoalStatus::STATUS_ACCEPTED ||
         status == action_msgs::msg::GoalStatus::STATUS_EXECUTING))
    {
      action_client_->async_cancel_goal(navigation_goal_handle_);

      RCLCPP_INFO(this->get_logger(), "cancel navigation");
      sleep(2);
      navigation_goal_handle_.reset();
    }
  }

private:
  // Timer callback function used to retrieve the real-time coordinates of the robot.
  void timerCallback()
  {
    if (rclcpp::ok() && navigation_goal_handle_)
    {
      try
      {
        geometry_msgs::msg::TransformStamped transform_stamped =
tf_buffer_->lookupTransform("map", "base_link", tf2::TimePointZero);

        RCLCPP_INFO(this->get_logger(), "the real-time coordinates of the
robot(x,y,yaw)=>(%.3f,%.3f,%.3f)", transform_stamped.transform.translation.x,
                    transform_stamped.transform.translation.y,
tf2::getYaw(transform_stamped.transform.rotation));
      }
      catch (tf2::TransformException &ex)
      {
        //          RCLCPP_ERROR(this->get_logger(), "Tf_listener Exception: %s",
ex.what());
      }
    }
  }

  rclcpp::TimerBase::SharedPtr timer_;
```

```
    rclcpp_action::Client<nav2_msgs::action::NavigateToPose>::SharedPtr action_client_;
    nav2_msgs::action::NavigateToPose::Goal navigation_goal_;
    rclcpp_action::ClientGoalHandle<nav2_msgs::action::NavigateToPose>::SharedPtr
navigation_goal_handle_;
    std::shared_ptr<tf2_ros::Buffer> tf_buffer_;
    std::shared_ptr<tf2_ros::TransformListener> tf_listener_;
    bool is_running_{false};
};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<NavigationClient>();
    node->sendNavigateToPoseRequest(); // Invoke the NavigateToPose request.

    rclcpp::executors::SingleThreadedExecutor executor;
    executor.add_node(node);

    // Main loop that continues until the goal is reached or the node is closed.

    while (rclcpp::ok() && node->isRunning())
    {
        RCLCPP_INFO_ONCE(node->get_logger(), "start navigation");
        executor.spin_some();
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    //cancel
    if(node->isRunning())
    {
        node->cancelGoal();
    }

    rclcpp::shutdown();
    return 0;
}
```

Code functionality: Set navigation point data, create a "navigate_to_pose" action client, send the navigation point through the client to make the robot move forward by 1 meter. The robot continuously prints its own coordinates on the map during the navigation process.

Code analysis: Please refer to the comments in the code.

(6) After finishing the code, you need to add the file name to the compilation file in order to compile it. The compilation file is located in the "nav_pkg" directory and named "CMakeLists.txt". In the IDE interface, click on that file on the left side, and the content of the file will be displayed on the right side. In the "CMakeLists.txt" file located in the "nav_pkg" directory, add a new compilation rule for "nav_pkg_node.cpp". The content should be as follows:

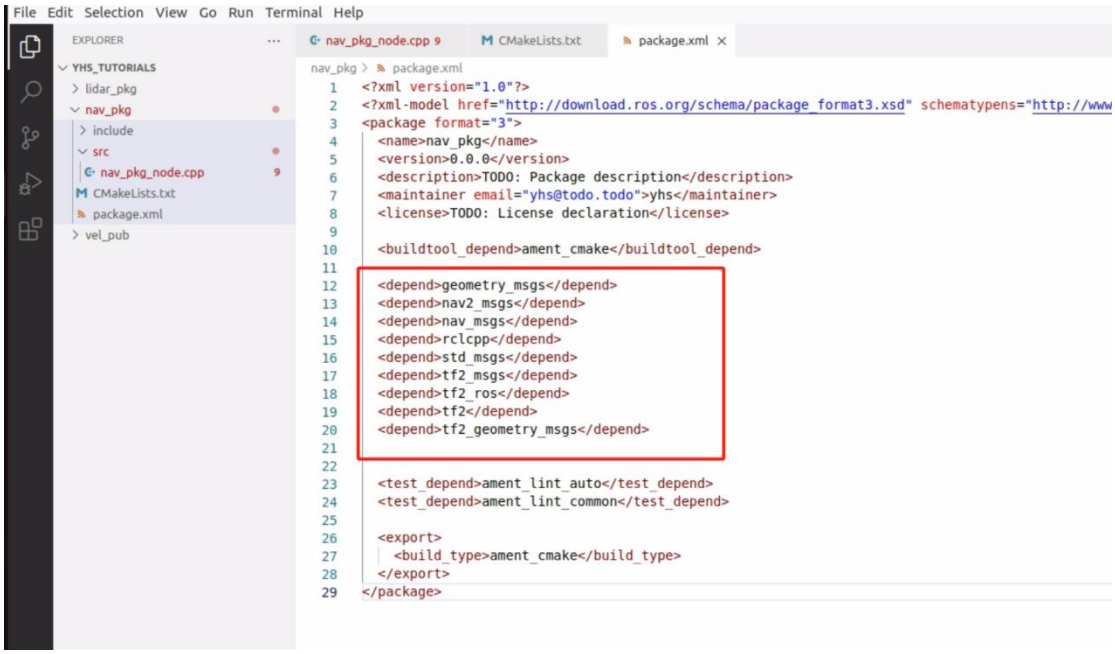
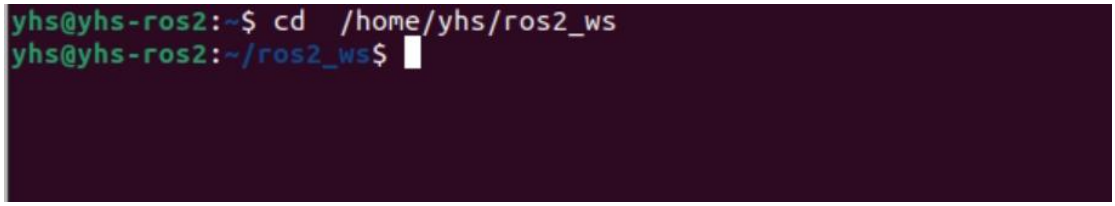
```
cmake_minimum_required(VERSION 3.8)
project(nav_pkg)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(nav2_util REQUIRED)
find_package(nav2_msgs REQUIRED)
find_package(tf2_ros REQUIRED)
find_package(tf2 REQUIRED)
find_package(tf2_msgs REQUIRED)
find_package(nav_msgs REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(tf2_geometry_msgs REQUIRED)

add_executable(nav_pkg_node src/nav_pkg_node.cpp)

ament_target_dependencies(nav_pkg_node
  "geometry_msgs"
  "nav2_msgs"
  "tf2_ros"
  "tf2"
  "tf2_msgs"
  "nav_msgs"
  "rclcpp"
  "std_msgs"
  "tf2_geometry_msgs"
)

install(TARGETS
  nav_pkg_node
  DESTINATION lib/${PROJECT_NAME}
```

```
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source files
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```


(7)Next, modify the "package.xml" file in the "nav_pkg" directory. The content should be as follows:

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>nav_pkg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="yhs@todo.todo">yhs</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>
  <depend>geometry_msgs</depend>
```

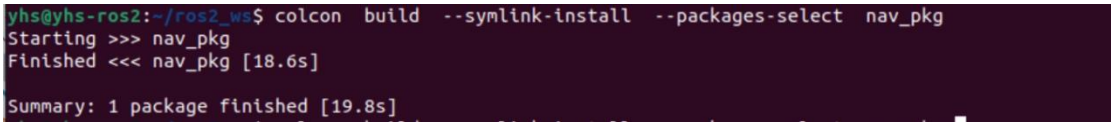
```
  <depend>nav2_msgs</depend>
  <depend>nav_msgs</depend>
  <depend>rclcpp</depend>
  <depend>std_msgs</depend>
  <depend>tf2_msgs</depend>
  <depend>tf2_ros</depend>
  <depend>tf2</depend>
  <depend>tf2_geometry_msgs</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>
  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```


File Edit Selection View Go Run Terminal Help
EXPLORER
YHS_TUTORIALS
lidar_pkg
nav_pkg
include
src
nav_pkg_node.cpp
CMakeLists.txt
package.xml
vel_pub
nav_pkg_node.cpp
CMakeLists.txt
package.xml
nav_pkg > package.xml
1 <?xml version="1.0"?>
2 <?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www
3 <package format="3">
4 <name>nav_pkg</name>
5 <version>0.0.0</version>
6 <description>TODO: Package description</description>
7 <maintainer email="yhs@todo.todo">yhs</maintainer>
8 <license>TODO: License declaration</license>
9
10 <buildtool_depend>ament_cmake</buildtool_depend>
11
12 <depend>geometry_msgs</depend>
13 <depend>nav2_msgs</depend>
14 <depend>nav_msgs</depend>
15 <depend>rclcpp</depend>
16 <depend>std_msgs</depend>
17 <depend>tf2_msgs</depend>
18 <depend>tf2_ros</depend>
19 <depend>tf2</depend>
20 <depend>tf2_geometry_msgs</depend>
21
22
23 <test_depend>ament_lint_auto</test_depend>
24 <test_depend>ament_lint_common</test_depend>
25
26 <export>
27 <build_type>ament_cmake</build_type>
28 </export>
29 </package>
```

(8) Similarly, after making the modifications, press the keyboard shortcut Ctrl+S to save the file. The small black dot on the right side of the file name at the top of the code should change to an "X," indicating that the file has been successfully saved. Now, let's proceed with the compilation of the code file. Start a terminal program and enter the following command to navigate to the ROS2 workspace:

```
cd  /home/yhs/ros2_ws
```


```

(9) Then execute the following command to start the compilation:

```
colcon build --symlink-install --packages-select nav_pkg
```

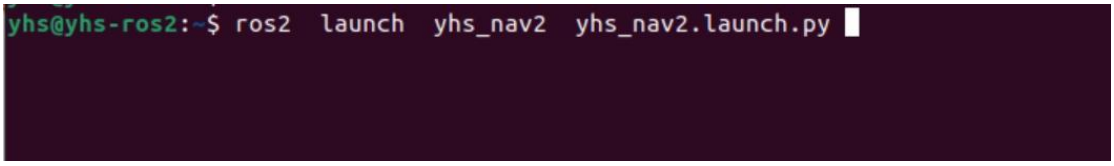

After executing this command, you will see scrolling compilation information. It will continue until you see the message "Summary: 1 package finished," indicating that the new nav_pkg_node node has been successfully compiled.

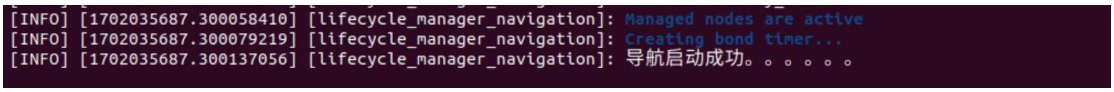



2. Run the "nav_pkg_node" node.

(1) Before launching our nav_pkg_node node, we need to start the navigation service of the Yuhesen robot. Open a terminal and enter the following command, then press Enter. If you see the message "Navigation started successfully...," it means that the navigation has been successfully launched.

```
ros2 launch yhs_nav2 yhs_nav2.launch.py
```

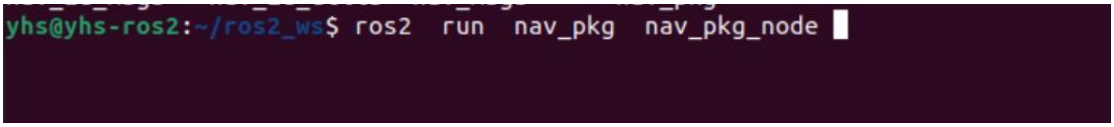



(2) After successfully launching the navigation, initialize the robot following the instructions provided in the previous section.

(3) After initialization, open another terminal and enter the command, then press Enter.

```
ros2 run nav_pkg nav_pkg_node
```




(4) Switch the remote controller to command mode. The robot will navigate forward for approximately 1 meter while continuously printing its own coordinates during the process.

```
yhs@yhs-ros2:~/ros2_ws$ ros2  run  nav_pkg  nav_pkg_node
[INFO] [1702038945.131627586] [nav_pkg_node]: 导航中。。。
[INFO] [1702038945.631501249] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(3.214,-0.035,0.007)
[INFO] [1702038946.131477660] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(3.214,-0.035,0.007)
[INFO] [1702038946.631472118] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(3.214,-0.035,0.007)
[INFO] [1702038947.131476485] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(3.214,-0.035,0.007)
[INFO] [1702038947.631448435] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(3.345,-0.034,0.006)
[INFO] [1702038948.131815725] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(3.592,-0.032,0.008)
[INFO] [1702038948.631502362] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(3.841,-0.040,0.001)
[INFO] [1702038949.131494066] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(4.030,-0.047,0.017)
[INFO] [1702038949.631478247] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(4.131,-0.051,0.025)
[INFO] [1702038950.131481887] [nav_pkg_node]: 机器人实时坐标(x,y,yaw)=>(4.126,-0.034,0.040)
[INFO] [1702038950.131635375] [nav_pkg_node]: 导航完成。
```

(5) Try assigning different values to pose.pose.position's x and y in the code and observe the robot's navigation behavior.

(6) Try changing pose.header.frame_id to another coordinate system (e.g., "map") and observe if the robot's navigation destination changes.

3. Please refer to the file in the directory for the Python version of the code.

| /home/yhs/ros2_ws/src/navigation2/nav2_system_tests/src/system |
|---|

# Experiment 11: Global Path Planning

In autonomous driving, global path planning algorithms commonly use A* or Dijkstra's algorithm. For detailed algorithm principles and implementations, you can refer to (https://www.redblobgames.com/pathfinding/a-star/introduction.html). Let's take a look at how these two algorithms are used in navigation.

1. View the navigation configuration file

(1) Open a terminal and navigate to the "/home/yhs/ros2_ws/src/yhs_nav2/param" directory. You will find the "yhs_nav2.yaml" file, where all navigation-related parameters are set and can be modified.

```
cd  ros2_ws/src/yhs_nav2/param
```

```
yhs@yhs-ros2:~/ros2_ws/src/yhs_nav2/param$ ls
yhs_nav2.yaml
```

(2)Open the file and look for the content related to global path planning.

```
348 planner_server:
349   ros__parameters:
350     expected_planner_frequency: 20.0
351     use_sim_time: False
352     planner_plugins: ["GridBased"]
353     GridBased:
354       plugin: "nav2_navfn_planner/NavfnPlanner"
355       tolerance: 0.5
356       use_astar: true
357       allow_unknown: false
358
```

(3) You may need to modify the debugging parameters.

| | |
|---|---|
| plugin | Planning plugin: nav2_navfn_planner/NavfnPlanner |
| expected_planner_frequency | The desired planning frequency, map size, target distance, and industrial computer performance can all affect the actual planning time. |
| use_astar | To select the planning algorithm, set it to true for A* or false for Dijkstra. |
| allow_unknown | By default, path planning is not possible in unknown areas of the map. |

2. View the global planned path

(1) Using the default global planning parameters, after starting the navigation, initialize the robot's position.

```
ros2  launch  yhs_nav2  yhs_nav2.launch.py
```

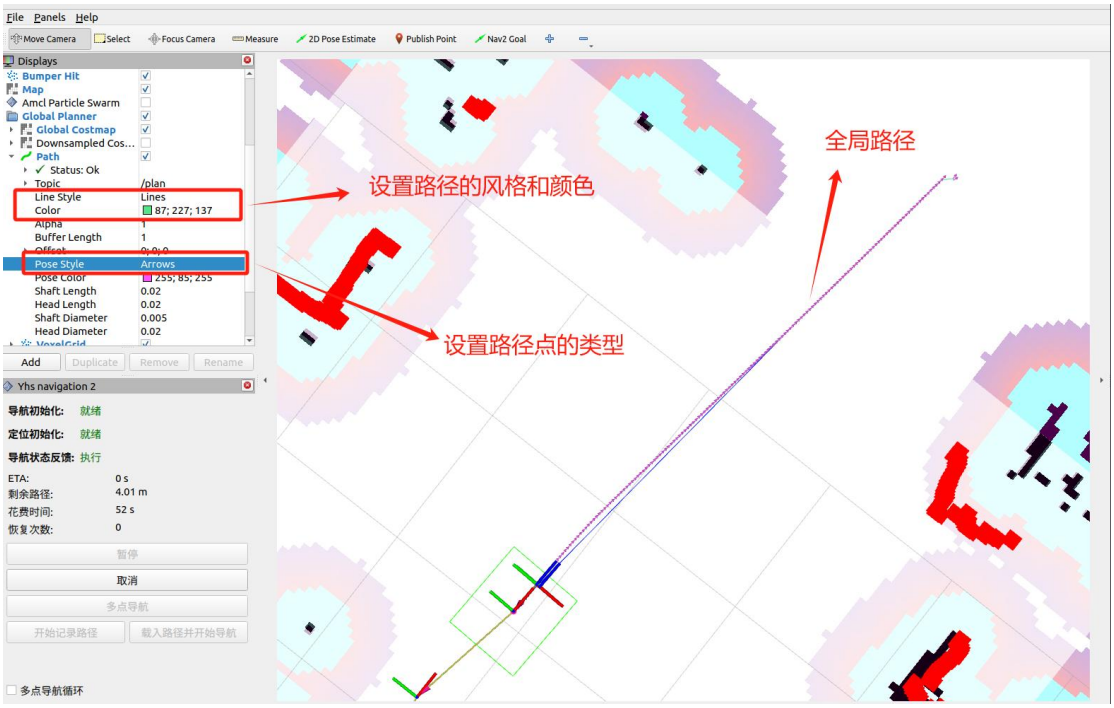
```
yhs@yhs-ros2:~$ ros2  launch  yhs_nav2  yhs_nav2.launch.py
```

```
[INFO] [1702106224.915899532] [lifecycle_manager_navigation]: Managed nodes are active
[INFO] [1702106224.915919776] [lifecycle_manager_navigation]: Creating bond timer...
[INFO] [1702106224.915969843] [lifecycle_manager_navigation]: 导航启动成功。。。。。。
```

(2) Open RViz2 and send the navigation point to visualize the global path. The size and color of the path can be configured.

rviz2

```
yhs@yhs-ros2:~$ rviz2
```




(3) Try using other planning algorithms by uncommenting the relevant content in "yhs_nav2.yaml" file, which utilizes a hybrid A* algorithm. For detailed information, please refer to "https://github.com/ros-planning/navigation2/tree/humble/nav2_smac_planner".

```
#planner_server:
#  ros__parameters:
#    expected_planner_frequency: 20.0
#    use_sim_time: False
#    planner_plugins: ["GridBased"]
#
#    GridBased:
#      plugin: "nav2_smac_planner/SmacPlannerHybrid"  # SmacPlannerHybrid SmacPlannerLattice SmacPlanner2D
#      tolerance: 0.5
#      downsample_costmap: false
#      downsampling_factor: 5
#      allow_unknown: false
#      max_iterations: 1000000
#      max_on_approach_iterations: 1000
#      max_planning_time: 3.5
#      motion_model_for_search: "DUBIN"    # MOORE, VON_NEUMANN, DUBIN, REEDS_SHEPP, STATE_LATTICE
#      cost_travel_multiplier: 2.0
#      angle_quantization_bins: 64
#      analytic_expansion_ratio: 3.5
#      analytic_expansion_max_length: 3.0
#      minimum_turning_radius: 0.05
#      reverse_penalty: 2.1
#      change_penalty: 0.0
#      non_straight_penalty: 1.20
#      cost_penalty: 2.0
#      retrospective_penalty: 0.025
#      rotation_penalty: 5.0
#      lookup_table_size: 20.0
#      cache_obstacle_heuristic: True
#      allow_reverse_expansion: True
#      smooth_path: True
#      debug_visualizations: True
#      smoother:
#        max_iterations: 1000
#        w_smooth: 0.3
#        w_data: 0.2
#        do_refinement: true

```

# Experiment 12: Local Path Planning

In the previous experiment, we selected a global path planning algorithm and sent the navigation point to generate the global path. Then, we used the local path planner to generate specific action strategies for the robot based on this path and the information from the costmap. The commonly used local path planning algorithms are the Dynamic Window Approach (DWA) and the Time Elastic Band (TEB) algorithm.

Dynamic Window Approach (DWA): The DWA algorithm samples multiple sets of velocities in the velocity space (v, w) and simulates the motion trajectories of these velocities over a certain period of time. It evaluates these trajectories using a cost function and selects the optimal trajectory to drive the robot's movement.

Time Elastic Band (TEB): The TEB algorithm is a nonlinear optimization algorithm. It utilizes the global path as the initial trajectory to obtain a series of discrete robot poses (pose) and corresponding time differences (timediff) within a local range. The poses and time differences on the trajectory need to satisfy constraints such as velocity, acceleration, robot kinematics, and distance to obstacles. These constraints can be modeled as a nonlinear optimization problem to obtain an optimized trajectory. The velocity can be derived from the poses and time differences, which is then sent to the robot for control.

Comparison of advantages and disadvantages:

| | Advantages | Disadvantages |
|---|---|---|
| DWA | Simple and computationally efficient, suitable for real-time applications. Can handle dynamic obstacles and robot constraints. | May result in suboptimal trajectories in complex environments. Limited ability to handle long-term planning. |
| | 适用于差分和全向车模 suitable for differential and omnidirectional chassis | Not suitable for Ackermann steering chassis |
| TEB | Suitable for most chassis | Requires more computational resources compared to DWA. |
| | Provides smooth and dynamically feasible trajectories. | Significant fluctuations in speed and angle, as well as unstable control. |
| | It has good obstacle avoidance capabilities for dynamic obstacles. | Suboptimal (non-global) solution. |

1. View the navigation configuration file

(1) Open a terminal and navigate to the "/home/yhs/ros2_ws/src/yhs_nav2/param" directory. You will find the "yhs_nav2.yaml" file where all the navigation-related parameters are set and can be modified.

```
cd  ros2_ws/src/yhs_nav2/param
```

```
yhs@yhs-ros2:~/ros2_ws/src/yhs_nav2/param$ ls
yhs_nav2.yaml
```

(2) Open the file and look for the content related to local path planning.

```
# TEB parameters
FollowPath:
  plugin: teb_local_planner::TebLocalPlannerROS

  teb_autosize: 1.0
  dt_ref: 0.3
  dt_hysteresis: 0.1
  max_samples: 500
  global_plan_overwrite_orientation: True
  allow_init_with_backwards_motion: False
  max_global_plan_lookahead_dist: 3.0
  global_plan_viapoint_sep: 0.0
  global_plan_prune_distance: 1.0
  control_look_ahead_poses: 2
  exact_arc_length: False
  feasibility_check_no_poses: 2
  publish_feedback: False

  # Robot
  max_vel_x: 0.5
  max_vel_theta: 0.3
  acc_lim_x: 0.3
  acc_lim_theta: 0.3

  footprint_model: # types: "point", "circular", "two_circles", "line", "polygon"
    type: "circular"
    radius: 0.1 # for type "circular"

  min_turning_radius: 0.0 # diff-drive robot (can turn on place!)

  # GoalTolerance
  xy_goal_tolerance: 0.1
  yaw_goal_tolerance: 0.1


   # Obstacles
  min_obstacle_dist: 0.45
  inflation_dist: 0.6
  include_costmap_obstacles: True
  costmap_obstacles_behind_robot_dist: 3.0
  obstacle_poses_affected: 15

  dynamic_obstacle_inflation_dist: 0.6
  include_dynamic_obstacles: True

  costmap_converter_plugin: "costmap_converter::CostmapToPolygonsDBSMCCH"
  costmap_converter_spin_thread: True
  costmap_converter_rate: 5

  # Optimization
  no_inner_iterations: 5
  no_outer_iterations: 4
  optimization_activate: True
  optimization_verbose: False
  penalty_epsilon: 0.1
  obstacle_cost_exponent: 4.0
  weight_max_vel_x: 2.0
  weight_max_vel_theta: 1.0
  weight_acc_lim_x: 1.0
  weight_acc_lim_theta: 10.5
  weight_kinematics_nh: 1000.0
  weight_kinematics_forward_drive: 1000.0
  weight_kinematics_turning_radius: 1.0
  weight_optimaltime: 1.0 # must be > 0
  weight_shortest_path: 0.0
  weight_obstacle: 100.0
  weight_inflation: 0.2
  weight_dynamic_obstacle: 10.0 # not in use yet
  weight_dynamic_obstacle_inflation: 0.2
  weight_viapoint: 50.0
  weight_adapt_factor: 2.0

```

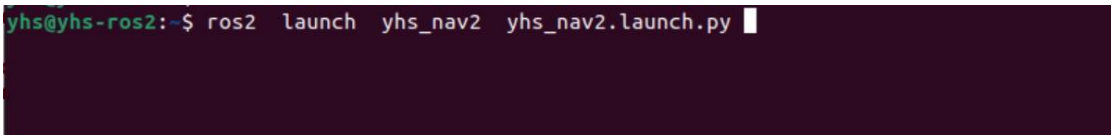
(3) You may need to modify the debugging parameters.

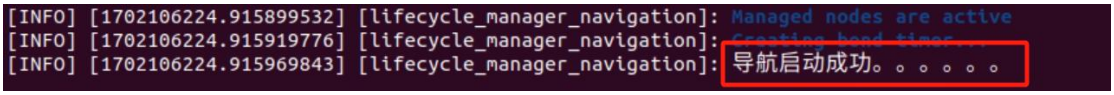
| | |
|---|---|
| dt_ref | Desired trajectory time resolution |
| dt_hysteresis | Automatic adjustment of size based on current time resolution, typically around. It is recommended to use 10% of dt_ref. |
| feasibility_check_no_poses | Feasibility analysis of poses for each sampling interval |
| max_vel_x | Maximum linear velocity of the robot |
| max_vel_x_backwards | Maximum backward linear velocity of the robot |
| acc_lim_x | Linear acceleration of the robot |
| max_vel_theta | Angular velocity of the robot |
| acc_lim_theta | Angular acceleration of the robot |
| weight_kinematics_forward_drive | Force the robot to only select optimization weights for forward (positive translational velocity) motions |
| weight_viapoint | Weight for tracking the reference path |
| min_obstacle_dist | Minimum desired distance to obstacles |

2. View debug information for local path planning

(1) Using the default local planning parameters, after starting the navigation, initialize the robot's position.

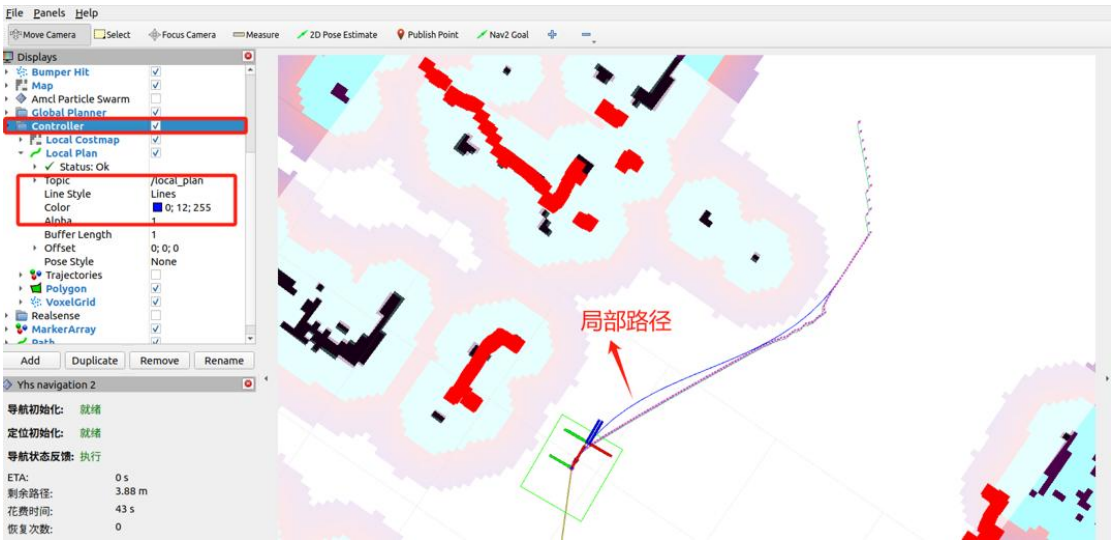
```
ros2 launch yhs_nav2 yhs_nav2.launch.py
```



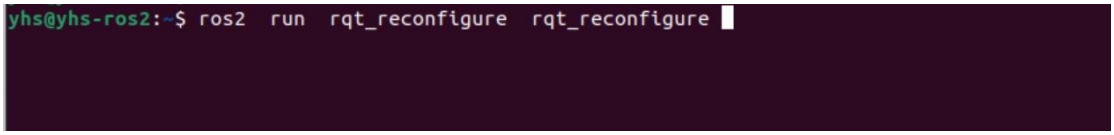

(2) Open RViz2, send the navigation point, and visualize the global path. The size and color of the path can be configured.

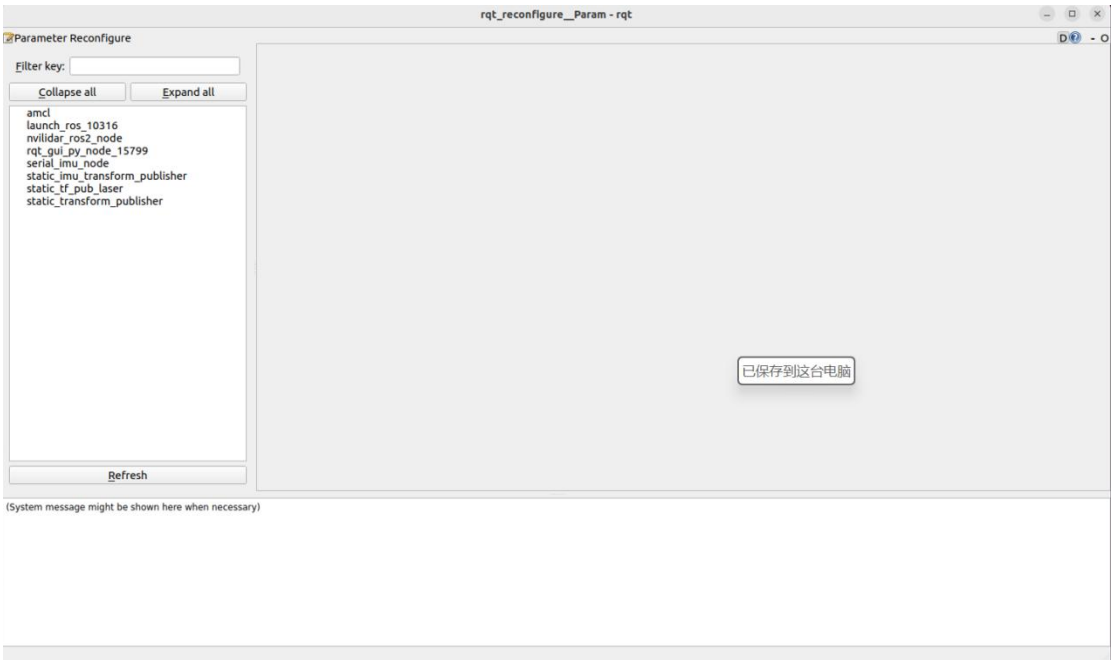
```
rviz2
```



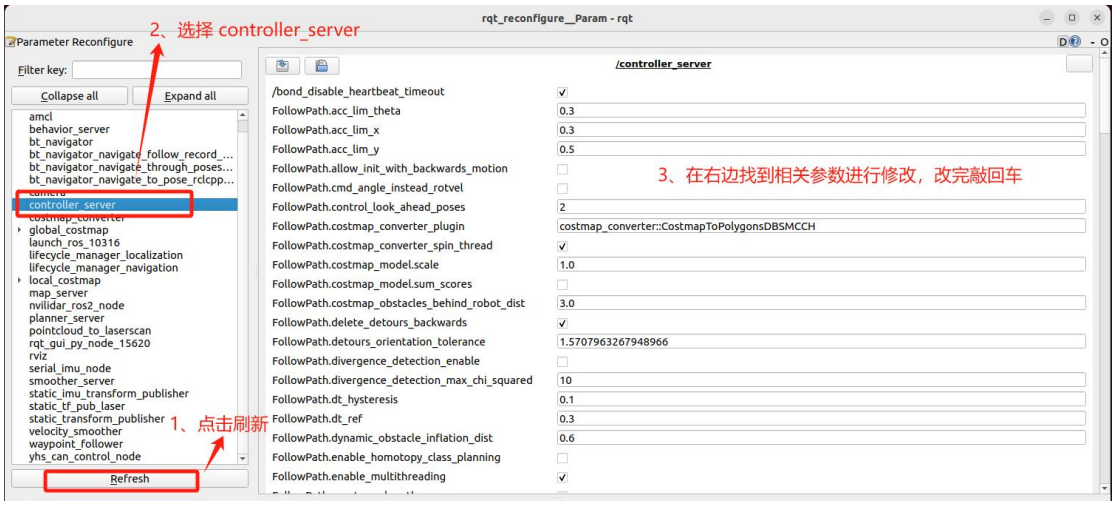

(3) During navigation, you can dynamically adjust the parameters of local path planning using "rqt_reconfigure" to facilitate observation. This eliminates the need to modify parameters and restart navigation each time, improving efficiency. Open another terminal and enter the following command, then press Enter to launch the "rqt_reconfigure" interface.

```
ros2 run rqt_reconfigure rqt_reconfigure
```



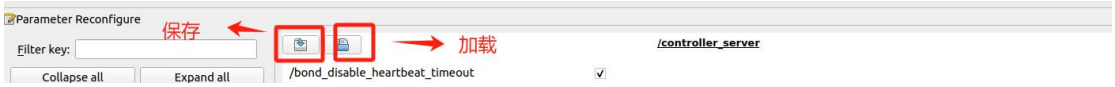




(4) Follow the steps in the image below for debugging.




(5) After modifying the parameters, you can directly observe the effects in the RViz2 interface.

(6) Parameter saving and loading can be done by clicking on the "Load" and "Save" buttons located at the top left corner of the parameter interface on the right side.




3. Regulated_pure_pursuit_controller Calculating velocity in the controller.

(1) Originally, the Teb controller directly calculated the velocity based on the path. After modifying the code, the local path generated by Teb is passed to the Regulated_pure_pursuit_controller controller, which is responsible for computing the velocity.

(2) The parameters for the Regulated_pure_pursuit_controller controller are also located in the "yhs_nav2.yaml" file.

```
# R_P_P_C parameters
Regulated_pure_pursuit_controller:
  plugin: "nav2_regulated_pure_pursuit_controller::RegulatedPurePursuitController"
  desired_linear_vel: 0.5
  lookahead_dist: 0.6
  min_lookahead_dist: 0.3
  max_lookahead_dist: 0.9
  lookahead_time: 1.5
  rotate_to_heading_angular_vel: 0.3
  transform_tolerance: 0.1
  use_velocity_scaled_lookahead_dist: false
  min_approach_linear_velocity: 0.1
  approach_velocity_scaling_dist: 1.0
  use_collision_detection: true
  max_allowed_time_to_collision_up_to_carrot: 1.0
  use_regulated_linear_velocity_scaling: true
  use_cost_regulated_linear_velocity_scaling: false
  regulated_linear_scaling_min_radius: 0.9
  regulated_linear_scaling_min_speed: 0.25
  use_fixed_curvature_lookahead: false
  curvature_lookahead_dist: 1.0
  use_rotate_to_heading: true
  rotate_to_heading_min_angle: 0.785
  allow_reversing: true
  max_angular_accel: 0.5
  max_robot_pose_search_dist: 10.0
  cost_scaling_dist: 0.3
  cost_scaling_gain: 1.0
  inflation_cost_scaling_factor: 3.0
```

# Experiment 13: RGB-D Camera

The Yuhesen ROS robot is equipped with a vision sensor on its front end, which is a typical RGB-D camera. RGB-D stands for "RGB-Depth," where "RGB" refers to the colors red, green, and blue, representing the color image, and "Depth" refers to the depth (distance) image. Therefore, an RGB-D camera is a composite sensor that can perceive object color and detect object distance. In terms of data acquisition, it provides a color image and a point cloud containing distance information. In this experiment, we will have a general understanding of the camera's principle of operation and the data it captures.

First, let's take a look at the external components of the depth camera. The main body is in the shape of a rectangular prism. On the front panel of the main body, several key sensors are arranged:

In this experiment, we primarily utilize the color camera.

1. Start the camera

(1) Open a terminal and enter the command, then press Enter to start the camera node.

| | | | | |
|---|---|---|---|---|
| 单相机： | ros2 | launch | ascamera_listener | hp60c.launch.py |
| 双相机： | ros2 | launch | ascamera_listener | hp60c_multiple.launch.py |

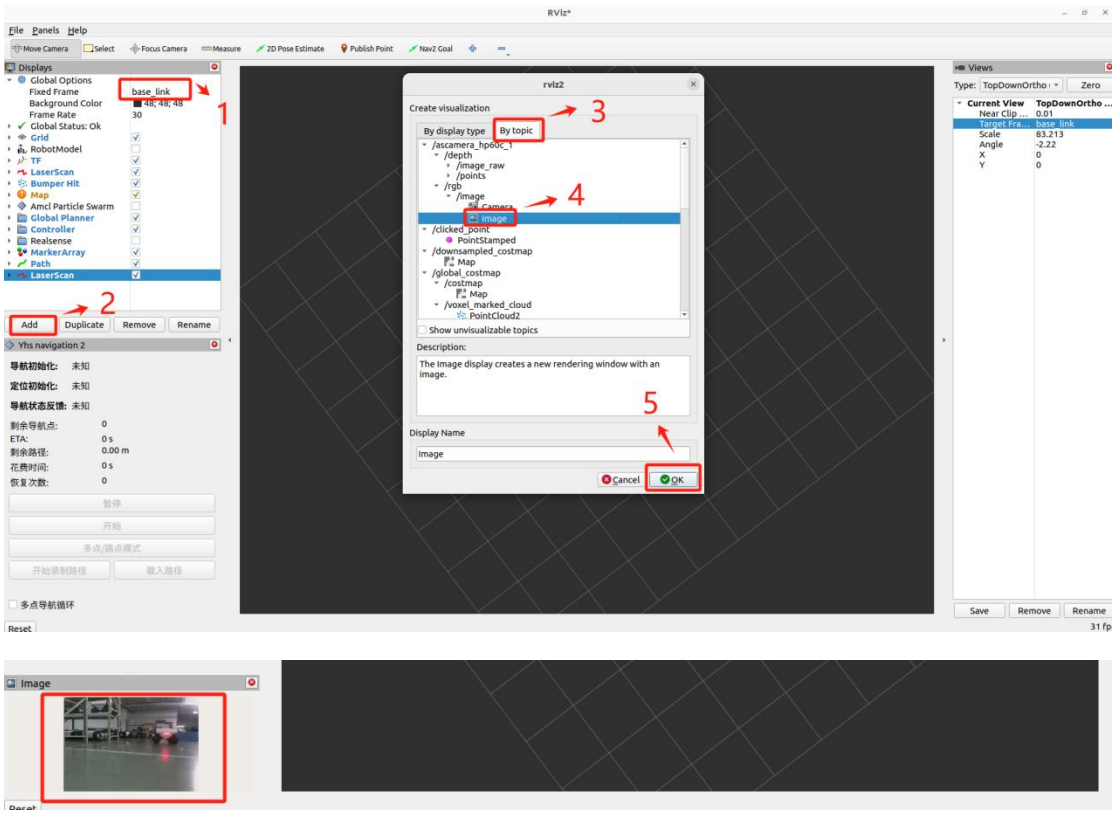
```
yhs@yhs-ros2:~$ ros2  launch  ascamera_listener   hp60c.launch.py
```

```
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [CameraHp60c.cpp] [250] [startStreaming] start streaming
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [CameraSrv.cpp] [175] [onAttached] attached end
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [120] [backgroundThread] SN [ ASC60CD21000104 ]'s parameter:
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [121] [backgroundThread] irfx: 207.71
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [122] [backgroundThread] irfy: 207.182
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [123] [backgroundThread] ircx: 154.072
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [124] [backgroundThread] ircy: 119.499
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [125] [backgroundThread] rgbfx: 291.259
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [126] [backgroundThread] rgbfy: 291.029
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [127] [backgroundThread] rgbcx: 163.26
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [128] [backgroundThread] rgbcy: 120.743
[ascamera_listener_node-1]
```

(2) Open another terminal and enter the command to launch RViz2.

```
rviz2
```

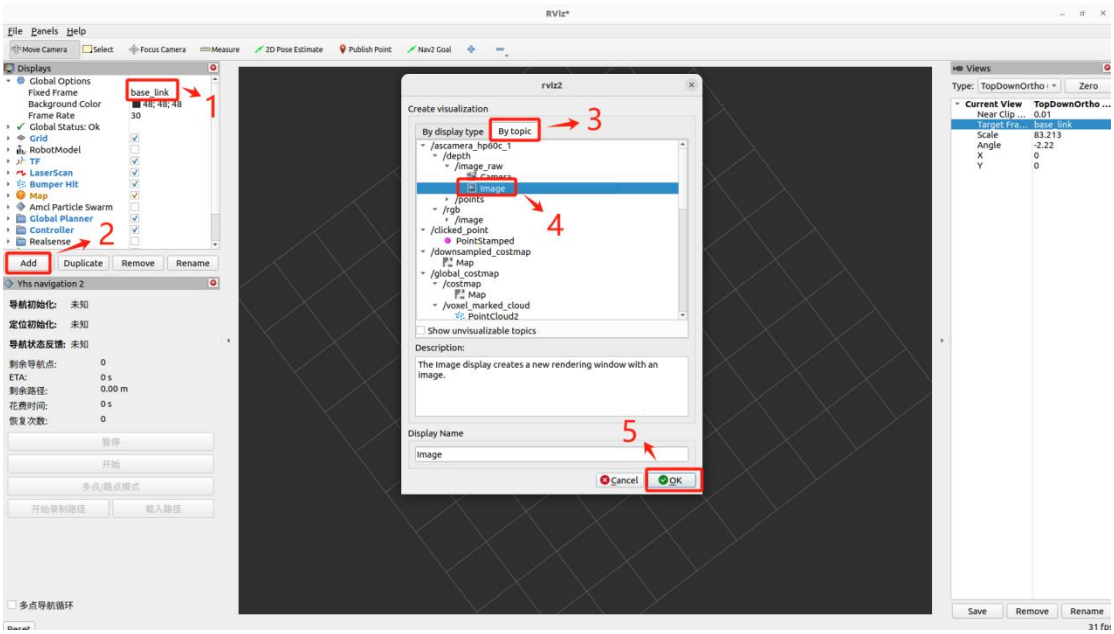

(3) Follow the steps below to add the display of the color image in RViz2:







(4) Follow the steps below to add the display of the depth image in RViz2:


RViz*
File Panels Help
Move Camera Select Focus Camera Measure 2D Pose Estimate Publish Point Nav2 Goal
Displays
Global Options
Fixed Frame base_link
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Grid
RobotModel
TF
LaserScan
Bumper Hit
Map
Amcl Particle Swarm
Global Planner
Controller
Realsense
Add Duplicate Remove Rename
Yhs navigation 2
导航初始化: 未知
定位初始化: 未知
导航状态反馈: 未知
剩余导航点: 0
ETA: 0 s
剩余路径: 0.00 m
花费时间: 0 s
恢复次数: 0
暂停
开始
多点/路点模式
开始录制路径
载入路径
多点导航循环
rviz2
Create visualization
By display type
By topic
/ascamera_hp60c_1
/depth
/image_raw
Camera
Image
/points
/rgb
/image
/clicked_point
PointStamped
/downsampled_costmap
Map
/global_costmap
/costmap
Map
/voxel_marked_cloud
PointCloud2
Show unvisualizable topics
Description:
The Image display creates a new rendering window with an image.
Display Name
Image
Cancel
OK
Views
Type: TopDownOrtho
Zero
Current View TopDownOrtho ...
Near Clip ... 0.01
Target Fra... base_link
Scale 83.213
Angle -2.22
X 0
Y 0
Save Remove Rename
1
2
3
4
5

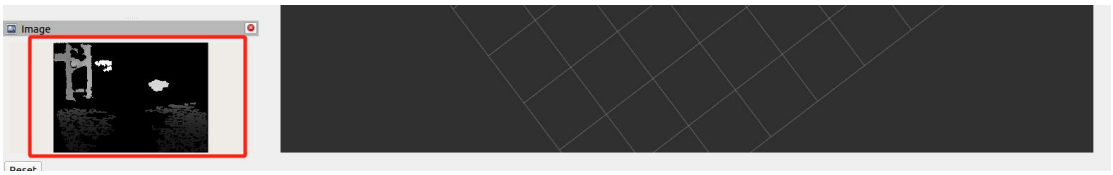


Image
Reset

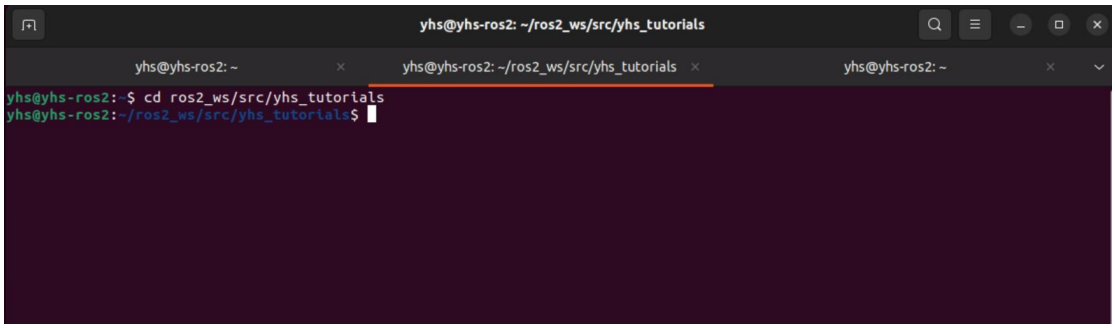
# Experiment 14: 3D Point Cloud from RGB-D Camera

In the previous experiment, we have already obtained the color image captured by the camera. In this experiment, we will learn how to retrieve the 3D point cloud captured by the RGB-D camera in the code.

1. Create a ROS2 package

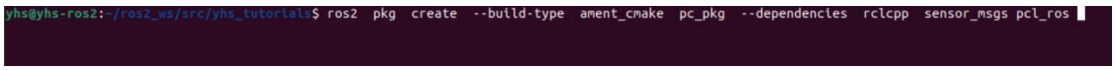
(1) Open a terminal and enter the command "cd ros2_ws/src/yhs_tutorials" to navigate to the "yhs_tutorials" directory.

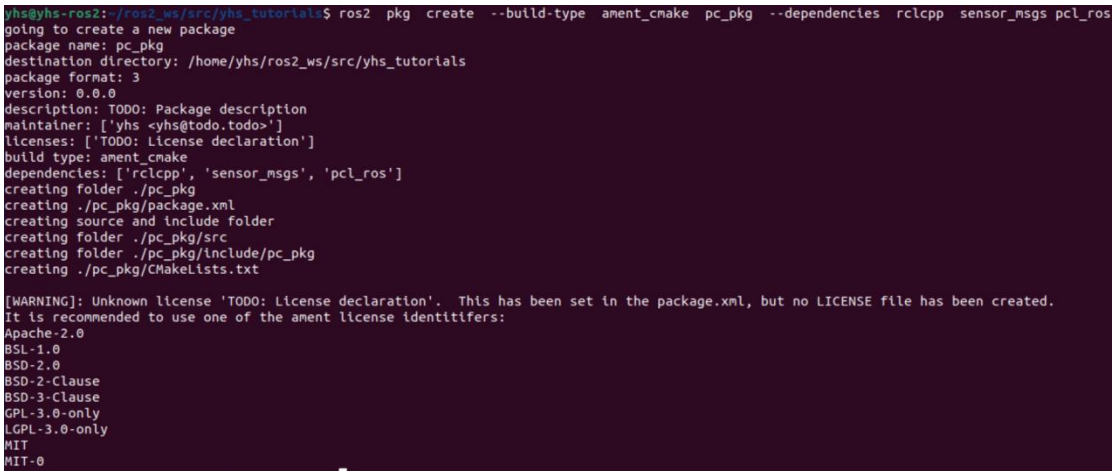
```
cd  ros2_ws/src/yhs_tutorials
```




(2) Use the following command to create a new ROS2 package:

```
ros2 pkg create --build-type ament_cmake pc_pkg --dependencies rclcpp sensor_msgs pcl_ros
```



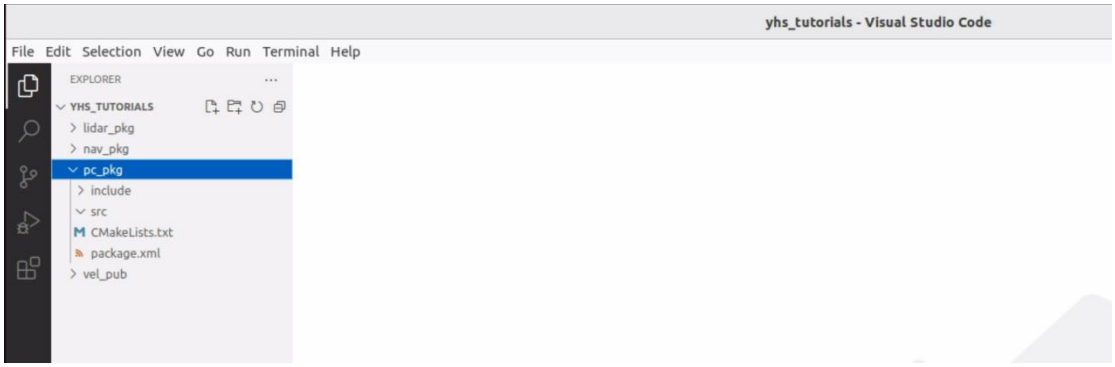



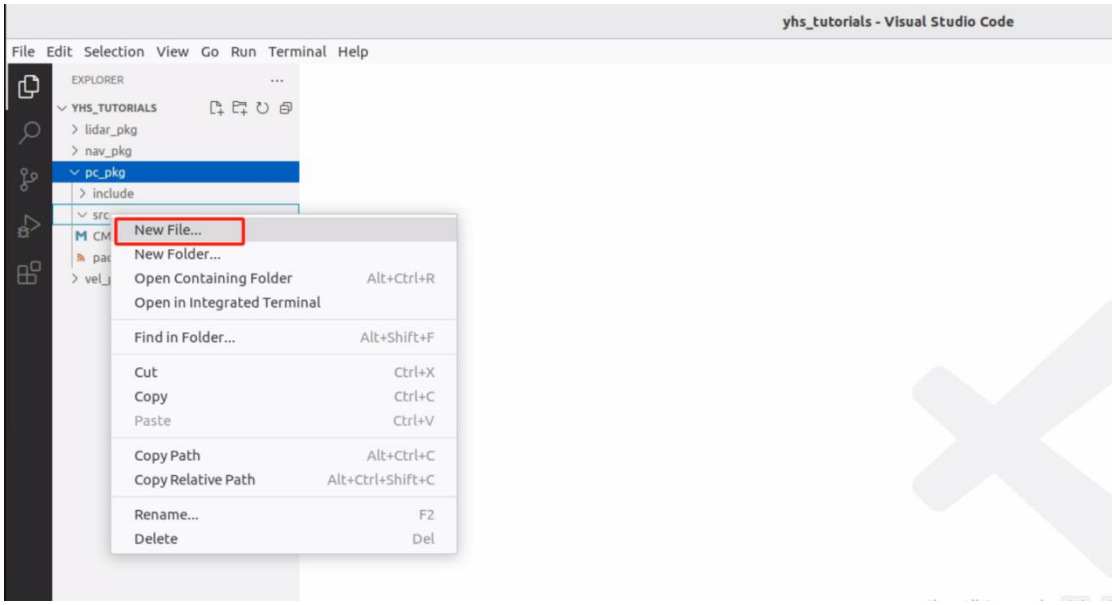

The meaning of this command:

| Command | Meaning |
|---|---|
| ros2 pkg create | creating source code package of ROS2 |
| --build-type ament_cmake | Use ament_cmake as the build type for package management. |
| pc_pkg | the name of new creating ROS2 source code package |
| --dependencies | Designated dependencies |

| | |
|---|---|
| sensor_msgs | For sensor message dependencies, include the point cloud data format. |
| pcl_ros | For the dependency on the open-source point cloud library (PCL) in ROS2. |

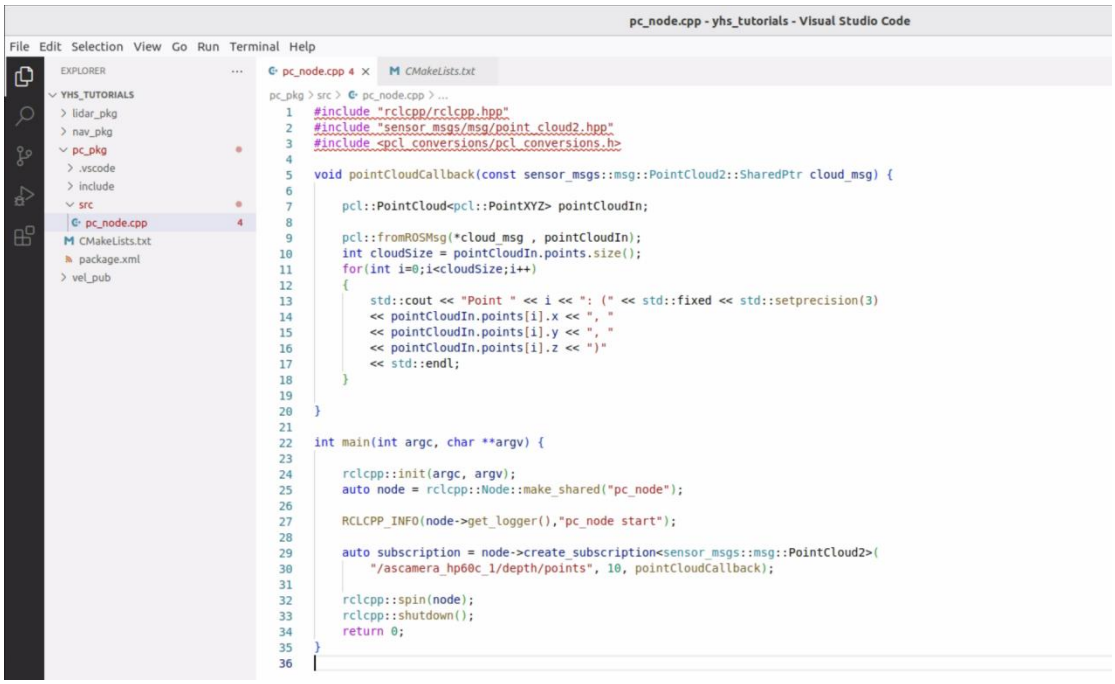
(3) After creating the "pc_pkg" package, open the "yhs_tutorials" directory in your IDE. You will notice that a new folder called "pc_pkg" has been added under it. Right-click on the "src" subfolder and select "New File" to create a new code file.




(4) The newly created code file should be named "pc_node.cpp".

(5) After naming the file, you can start writing the code for "pc_node.cpp" on the right side of the IDE. The code content is as follows:




```
#include "rclcpp/rclcpp.hpp"
#include "sensor_msgs/msg/point_cloud2.hpp"
#include <pcl_conversions/pcl_conversions.h>

void pointCloudCallback(const sensor_msgs::msg::PointCloud2::SharedPtr cloud_msg) {

    pcl::PointCloud<pcl::PointXYZ> pointCloudIn;

    pcl::fromROSMsg(*cloud_msg , pointCloudIn);
    int cloudSize = pointCloudIn.points.size();
    for(int i=0;i<cloudSize;i++)
    {
        std::cout << "Point " << i << ": (" << std::fixed << std::setprecision(3)
        << pointCloudIn.points[i].x << ", "
        << pointCloudIn.points[i].y << ", "
        << pointCloudIn.points[i].z << ")"
        << std::endl;
    }

}

int main(int argc, char **argv) {
```

```
    rclcpp::init(argc, argv);
    auto node = rclcpp::Node::make_shared("pc_node");

    RCLCPP_INFO(node->get_logger(),"pc_node start");

    auto subscription = node->create_subscription<sensor_msgs::msg::PointCloud2>(
        "/ascamera_hp60c_1/depth/points", 10, pointCloudCallback);

    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

1) At the beginning of the code, three header files are included: "rclcpp.hpp" is the ROS2 system header file, "sensor_msgs/msg/point_cloud2.hpp" is the header file for point cloud type in ROS2, and "pcl_conversions.h" is the header file for point cloud conversion types in PCL.

2) Define a callback function named "void pointCloudCallback()" to handle the 3D point cloud data. Every time ROS2 receives a frame of depth image, it automatically converts it into a 3D point cloud and calls this callback function. The 3D point cloud data is passed as a parameter to this callback function.

3) The parameter "cloud_msg" of the callback function "void pointCloudCallback()" is a pointer to the memory area that stores the 3D point cloud in the sensor_msgs::msg::PointCloud2 format. In practical development, this format of point cloud is usually not directly used. Instead, it is converted to the PCL point cloud format, allowing the use of the rich functions provided by PCL for point cloud data processing.

4) Inside the callback function "void pointCloudCallback()", a point cloud container of type "pcl::PointXYZ" named "pointCloudIn" is defined. The pcl::fromROSMsg() function is called to convert the point cloud data in ROS2 format from the parameter to the PCL format, and it is stored in the container "pointCloudIn".

5) The number of 3D points in the converted point cloud array "pointCloudIn.points" is obtained and stored in a variable named "cloudSize". A for loop is used to display the x, y, and z values of all points in "pointCloudIn.points" using std::cout in the terminal program. Typically, the raw coordinate values in "pointCloudIn.points" are not used directly; they need to be transformed to the robot's coordinate system before being processed using functions from the PCL point cloud library.

6) In the main function "main()", rclcpp::init() is called to initialize the node.

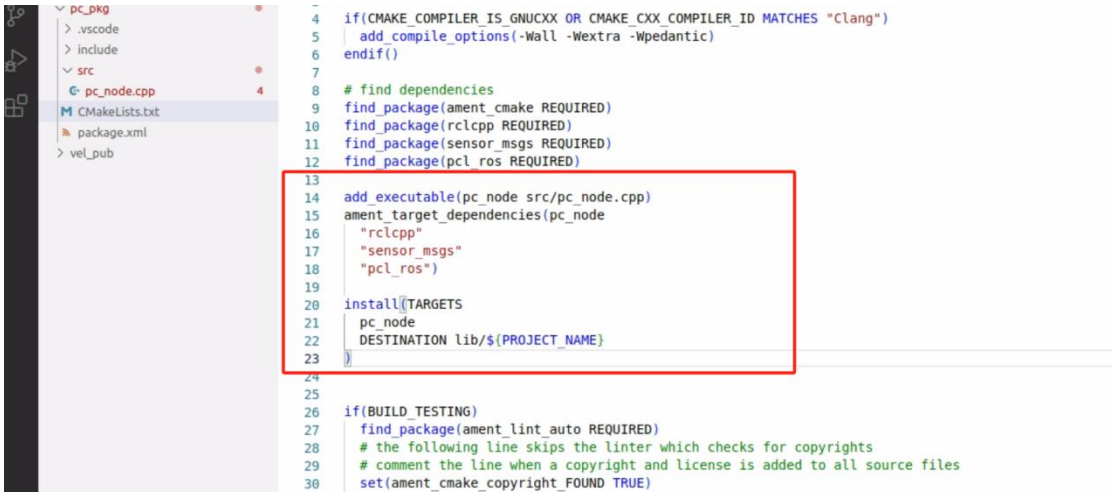
7) RCLCPP_INFO() is called to output a string message to the terminal program, indicating that the node has started successfully.

8) A rclcpp::Node object named "node" is defined as a node handle, and using this handle, data is subscribed from the ROS2 core node with the topic name "/ascamera_hp60c_1/depth/points", and the callback function is set as the previously defined "pointCloudCallback()". "/ascamera_hp60c_1/depth/points" is the topic name where the camera's ROS2 node publishes the 3D point cloud. The 3D point cloud captured by the camera will be sent to this topic, and our own node "pc_node" will subscribe to it to receive the 3D point cloud data.

9) rclcpp::spin() is called to block the main() function, keeping the node program from ending or exiting.

(6) After finishing the code, the filename needs to be added to the compilation file in order to compile it. The compilation file is located in the directory of "pc_pkg" and named "CMakeLists.txt". Click on this file on the left side of the IDE interface, and its contents will be displayed on the right side. In the "CMakeLists.txt" file, add a new compilation rule for "pc_node.cpp". The content should be as follows:

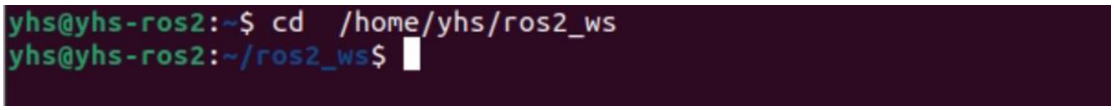
```
add_executable(pc_node src/pc_node.cpp)
ament_target_dependencies(pc_node
  "rclcpp"
  "sensor_msgs"
  "pcl_ros")

install(TARGETS
  pc_node
  DESTINATION lib/${PROJECT_NAME}
)
```




(7) Similarly, after making the modifications, press the keyboard shortcut Ctrl+S to save the changes. The small black dot on the right side of the filename at the top of the code

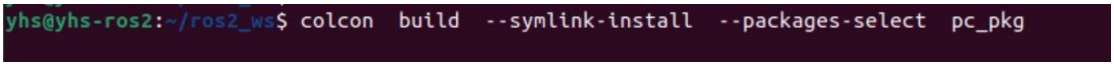
will turn into an "X", indicating that the file has been successfully saved. Now, let's proceed with the compilation of the code file. Launch a terminal program and enter the following command to navigate to the ROS2 workspace:

```
cd  /home/yhs/ros2_ws
```

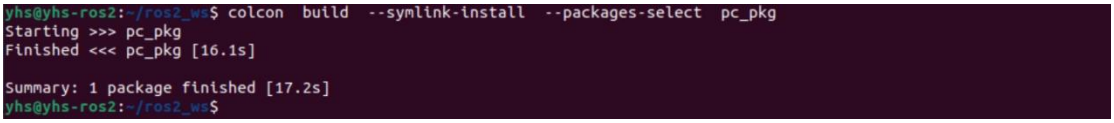



(8) Then, execute the following command to start the compilation:

```
colcon  build  --symlink-install  --packages-select  pc_pkg
```



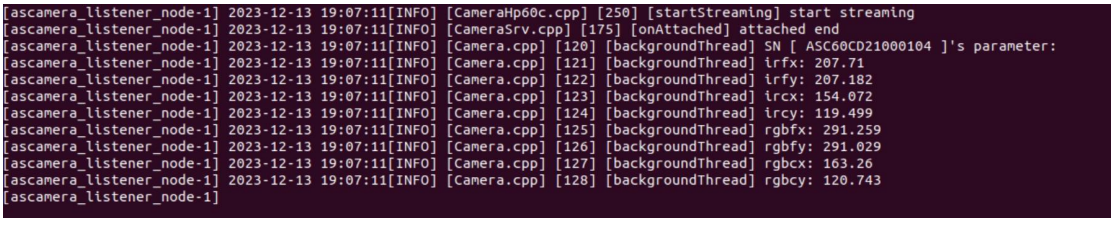

After executing this command, you will see scrolling compilation messages until you see the message "Summary: 1 package finished," indicating that the new "pc_node" node has been successfully compiled.




2. Running the "pc_node" node

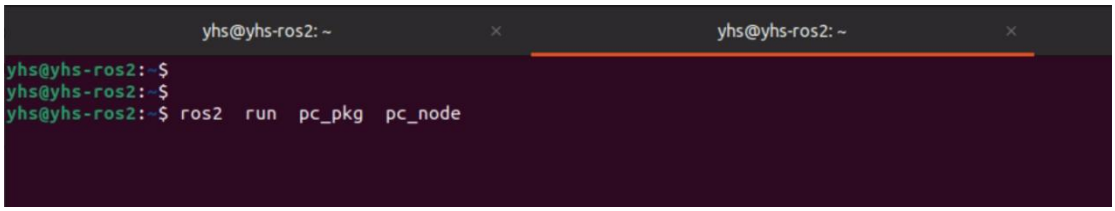
(1) First, run the camera node by opening a terminal and entering the command, then press Enter to start the camera node.

```
单相机：ros2  launch  ascamera_listener  hp60c.launch.py
双相机：ros2  launch  ascamera_listener  hp60c_multiple.launch.py
```


(2) Now, we can run the "pc_node" that we just wrote. Open another terminal and enter the command, then press Enter.

```
ros2  run  tm_lidar_pkg  pc_node
```




This command will start our written "pc_node". According to the program logic, it will continuously retrieve point cloud data packets from the 3D point cloud topic "/ascamera_hp60c_1/depth/points" of the camera. It will convert the ROS-formatted 3D point cloud into the PCL format and then display the x, y, and z coordinate values of all points in the terminal.

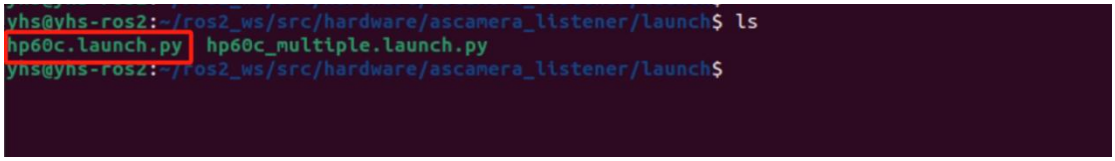

In the terminal, the message "pc_node start" will be displayed, indicating that the pc_node has started successfully. It will continuously refresh and display the coordinate values of all points in the point cloud only when obstacles within the specified distance are detected.

# Experiment 15: Incorporating Obstacle Layer with Camera LaserScan Data

During robot navigation, 2D laser data is used for dynamic global and local path planning to avoid obstacles along the path. However, the scanning range of the 2D laser is limited to a plane, and it cannot detect objects that are positioned higher or lower than the laser installation location. Therefore, we need to add the point cloud data from the depth camera to the navigation obstacle layer.

1. Converting Point Cloud Data to LaserScan Data

(1) To convert point cloud data to LaserScan data, we need to use the "pointcloud_to_laserscan" package. Let's take a look at the "hp60c.launch.py" file, which is located in the "/home/yhs/ros2_ws/src/hardware/ascamera_listener/launch" directory.




```
36
37    pointcloud_to_laserscan_1 = Node(
38        package='pointcloud_to_laserscan', executable='pointcloud_to_laserscan_node',
39        remappings=[('cloud_in', '/ascamera_hp60c_1/depth/points'),
40                    ('scan', '/scan1')],
41        parameters=[{
42            'target_frame': 'ascamera_hp60c_1',
43            'transform_tolerance': 0.01,
44            'min_height': -0.1,
45            'max_height': 2.0,
46            'angle_min': -1.5708,
47            'angle_max': 1.5708,
48            'angle_increment': 0.0087,
49            'scan_time': 0.3333,
50            'range_min': 0.05,
51            'range_max': 1.0,
52            'use_inf': True,
53            'inf_epsilon': 1.0
54        }],
55        name='pointcloud_to_laserscan_1'
56    )
57
```

The possible content that we may need to modify is as follows:

| | |
|---|---|
| /ascamera_hp60c_1/depth/points | Subscribed point cloud topic: The camera's published point cloud topic is used here. If it is a different topic, you can modify this parameter accordingly. |
| scan1 | Published topic after converting point cloud to LaserScan: Since the topic for 2D laser is typically "scan", it should not have the same name. Here, let's change it to "scan1". |
| min_height | Minimum height for point cloud data cropping: If the minimum height is set too low, it may convert ground point cloud data into LaserScan data, affecting path planning. You can modify and test this value repeatedly to find a suitable value. |
| max_height | Maximum height for point cloud data cropping: If the maximum height is set too high, it may affect path planning. You can modify and test this value repeatedly to find a suitable value. |

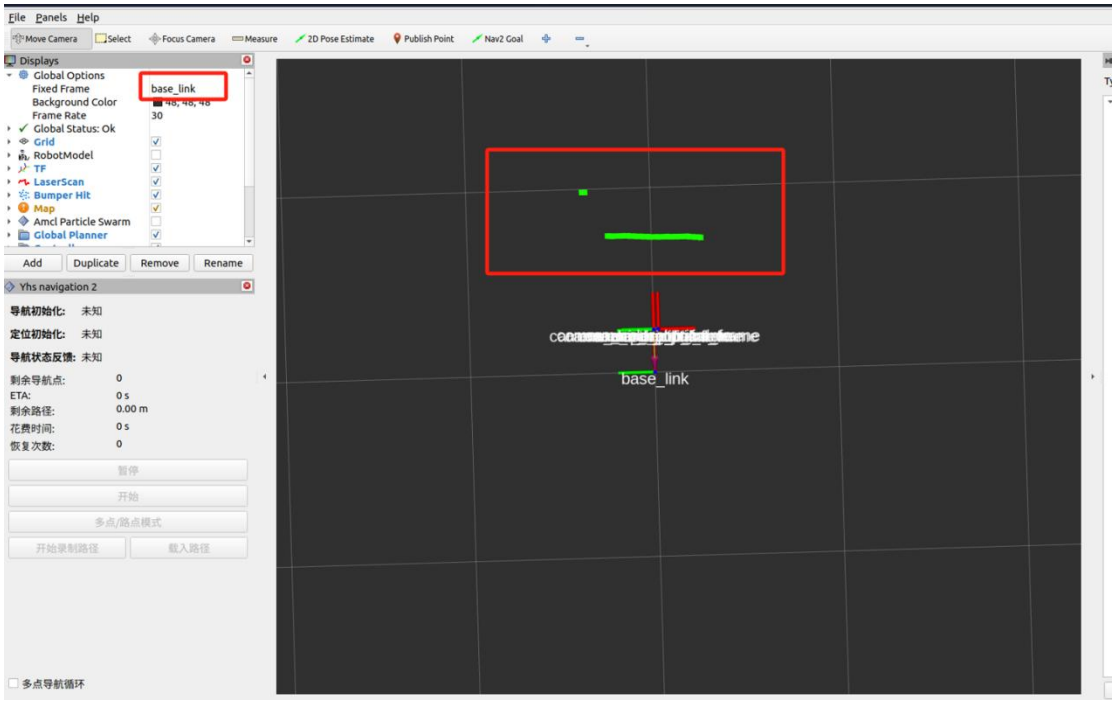
(2) Open a terminal and enter the command to start the camera node.

| Single camera：ros2 launch ascamera_listener hp60c.launch.py |
| --- |
| Dual camera：ros2 launch ascamera_listener hp60c_multiple.launch.py |

```
yhs@yhs-ros2:~$ ros2 launch ascamera_listener hp60c.launch.py
```

```
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [CameraHp60c.cpp] [250] [startStreaming] start streaming
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [CameraSrv.cpp] [175] [onAttached] attached end
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [120] [backgroundThread] SN [ ASC60CD21000104 ]'s parameter:
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [121] [backgroundThread] irfx: 207.71
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [122] [backgroundThread] irfy: 207.182
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [123] [backgroundThread] ircx: 154.072
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [124] [backgroundThread] ircy: 119.499
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [125] [backgroundThread] rgbfx: 291.259
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [126] [backgroundThread] rgbfy: 291.029
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [127] [backgroundThread] rgbcx: 163.26
[ascamera_listener_node-1] 2023-12-13 19:07:11[INFO] [Camera.cpp] [128] [backgroundThread] rgbcy: 120.743
[ascamera_listener_node-1]
```

(3) To view the data of the depth camera point cloud converted to LaserScan in rviz2, let's set the color of "scan1" to green.




2. Adding "scan1" Topic Data to the Obstacle Layer

(1) Navigate to the "/home/yhs/ros2_ws/src/yhs_nav2/param" directory, where you can find the "yhs_nav2.yaml" file.

| cd /home/yhs/ros2_ws/src/yhs_nav2/param |
| --- |

```
yhs@yhs-ros2:~/ros2_ws/src/yhs_nav2/param$ ls
yhs_nav2.yaml
```


(2) Open the "yhs_nav2.yaml" file and add the "scan1" data to the obstacle layer of the global costmap as shown in the following image.

```
283 global_costmap:
284   global_costmap:
285     ros__parameters:
286       update_frequency: 1.0
287       publish_frequency: 1.0
288       global_frame: map
289       robot_base_frame: base_link
290       use_sim_time: False
291 #      robot_radius: 0.22
292       footprint: "[[0.31,0.26],[0.31,-0.26],[-0.31,-0.26],[-0.31,0.26]]"
293       resolution: 0.05
294       track_unknown_space: true
295       plugins: ["static_layer", "obstacle_layer", "inflation_layer"]
296       obstacle_layer:
297         plugin: "nav2_costmap_2d::ObstacleLayer"
298         enabled: True
299         observation_sources: scan scan1
300         scan:
301           topic: /scan
302           max_obstacle_height: 2.0
303           clearing: True
304           marking: True
305           data_type: "LaserScan"
306           raytrace_max_range: 3.0
307           raytrace_min_range: 0.0
308           obstacle_max_range: 2.5
309           obstacle_min_range: 0.0
310         scan1:
311           topic: /scan1
312           max_obstacle_height: 2.0
313           clearing: True
314           marking: True
315           data_type: "LaserScan"
316           raytrace_max_range: 3.0
317           raytrace_min_range: 0.0
318           obstacle_max_range: 2.5
319           obstacle_min_range: 0.0
```

1

2

(3) Open the "yhs_nav2.yaml" file and add the "scan1" data to the obstacle layer of the local costmap as shown in the following image.

```
local_costmap:
  local_costmap:
    ros__parameters:
      update_frequency: 5.0
      publish_frequency: 2.0
      global_frame: odom
      robot_base_frame: base_link
      use_sim_time: False
      rolling_window: true
      width: 6
      height: 6
      resolution: 0.05
#      robot_radius: 0.22
      footprint: "[[0.31,0.26],[0.31,-0.26],[-0.31,-0.26],[-0.31,0.26]]"
      plugins: ["static_layer", "obstacle_layer"]
      inflation_layer:
        plugin: "nav2_costmap_2d::InflationLayer"
        inflation_radius: 1.0
        cost_scaling_factor: 3.0
      obstacle_layer:
        plugin: "nav2_costmap_2d::ObstacleLayer"
        enabled: True
        observation_sources: scan scan1
        scan:
          topic: /scan
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
        scan1:
          topic: /scan1
          max_obstacle_height: 2.0
          clearing: True
          marking: True
          data_type: "LaserScan"
      static_layer:
        plugin: "nav2_costmap_2d::StaticLayer"
        map_subscribe_transient_local: True
      always_send_full_costmap: True
```

1

2

(4) Open a terminal and enter the following command, then press Enter. If you see the message "Navigation started successfully...," it means that the navigation has been successfully launched.

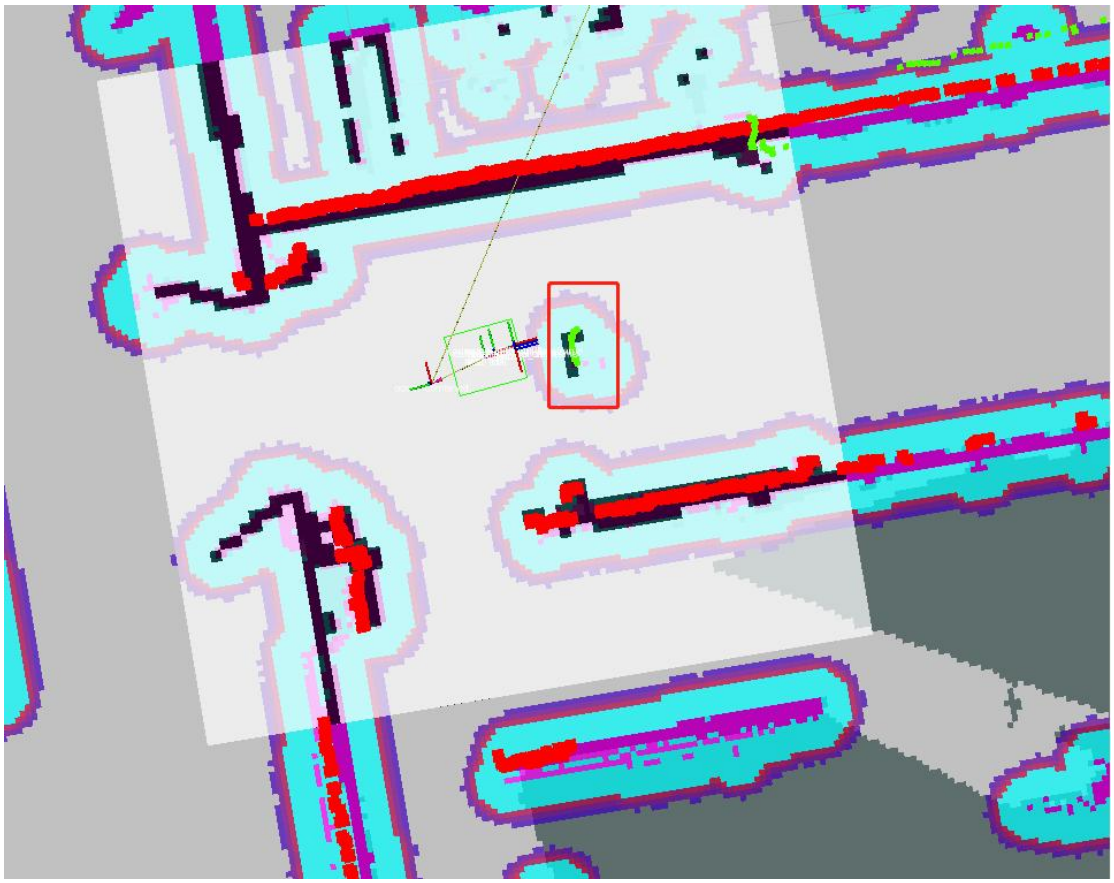
```
ros2 launch yhs_nav2 yhs_nav2.launch.py
```

```
yhs@yhs-ros2:~$ ros2 launch yhs_nav2 yhs_nav2.launch.py
```

```
[INFO] [1702035687.300058410] [lifecycle_manager_navigation]: Managed nodes are active
[INFO] [1702035687.300079219] [lifecycle_manager_navigation]: Creating bond timer...
[INFO] [1702035687.300137056] [lifecycle_manager_navigation]: 导航启动成功。。。。。
```

(5) After the successful launch of navigation, initialize the robot following the instructions provided in the previous section.

(6) As shown in the image below, the laser is mounted higher than the camera. The camera can detect relatively low objects, while the laser cannot.



## Common Issue Troubleshooting Solutions

First, check the error and warning messages printed on the terminal, as well as the robot's localization status and the positions of obstacles in Rviz2. In most cases, the root causes of the problems can be identified.

### Problem 1: Mapping or navigation fails to start.

Check if the commands are entered correctly.

### Problem 2: Unable to find the saved map.

Verify if the directory for saving the map is correct.

### Problem 3: Unsuccessful 3D localization.

Check if the saved 2D map matches the 3D point cloud map.

### Problem 4: Robot doesn't move after sending navigation goals.

Check if the remote controller is switched to command mode, examine the terminal output, and inspect the surrounding obstacles of the robot in Rviz2.

### Problem 5: Mapping or navigation fails to start after rebooting the industrial computer.

If you used the "reboot" command to restart, it may cause the CAN card to fail to initialize, resulting in the inability to receive or send data. In such cases, perform a power cycle restart instead of using the "reboot" command.

# Appendix 1: Remote Control

It is recommended to use SSH for remote access. If a graphical interface is necessary, NOMACHINE can be used.

## 1. SSH Remote Connection to the Server Prerequisites:

- A laptop computer
- Ubuntu 22.04 installed
- ROS2 installed with the Humble version

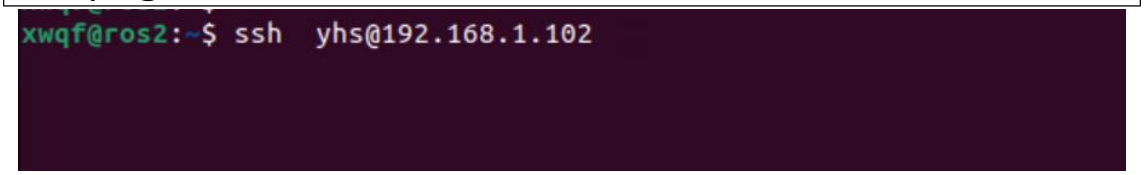
(1) Install ssh

```
sudo apt install ssh
```

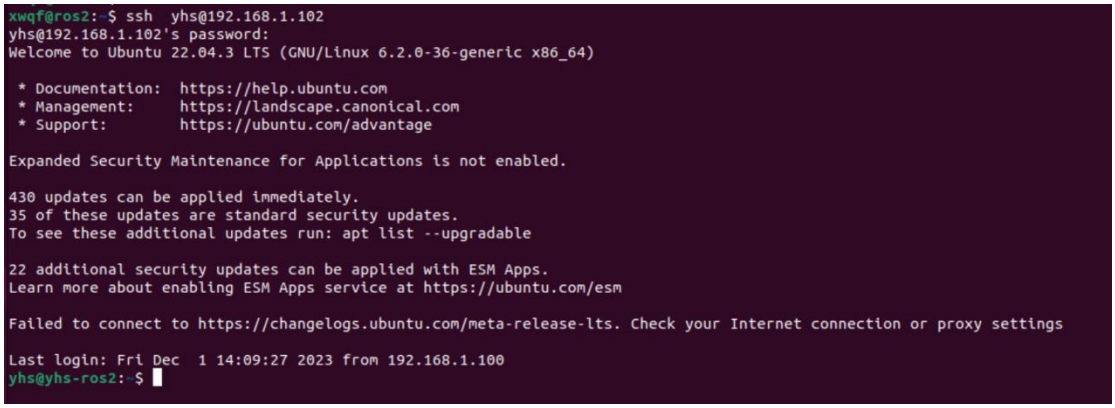
(2) To connect to the robot's Wi-Fi network and remotely log in, follow these steps:

1) Connect to the car's Wi-Fi network using your laptop or device.
2) Open a terminal on your laptop.
3) Enter the following command and press Enter.
4) When prompted, enter the password "123456". Note that the password input will not be displayed as you type.
5) Once you have entered the correct password, you will be remotely logged in to the robot's system.

```
ssh yhs@192.168.1.102
```



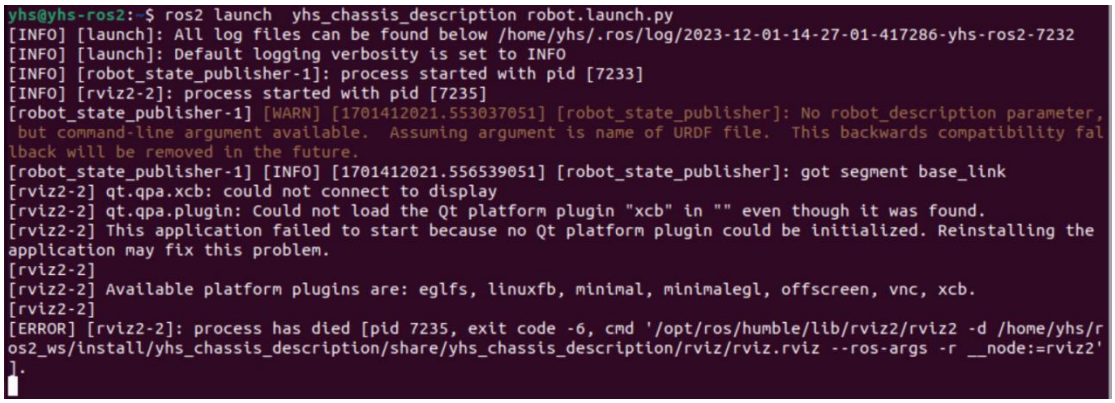





(3) Sometimes, it is necessary to have multiple terminals on the robot to enter commands. In such cases, you can open another terminal on your laptop and establish a remote login.。

(4) After remote login, when opening rviz2 on the robot's industrial computer, you may encounter errors, such as the following:

```
ros2 launch yhs_chassis_description robot.launch.py
```




(5) In ROS2, as long as the nodes are connected to the same local network, they can communicate with each other. This allows you to open rviz2 on your laptop. Before doing so, you need to copy the default configuration file of rviz2 from the robot's industrial computer to your laptop.

1) Open a terminal on your laptop.

2) Use the scp command to securely copy the file from the car's industrial computer to your laptop. The command syntax is as follows:

3) Enter the password for the car's industrial computer when prompted. The file will be securely copied to the specified destination on your laptop.

4) Once the copy is complete, you can open rviz2 on your laptop and use the copied default configuration file for visualization and configuration.

```
scp yhs@192.168.1.102:~/.rviz2/default.rviz ~/.rviz2/
```


(6) Assuming that there is a "ros2_ws/src" directory on your laptop, you can copy the folders "yhs_can_interfaces" and "nav2_rviz_plugins" from the car's industrial computer to the "ros2_ws/src" directory on your laptop.

```
xwqf@ros2:~/ros2_ws/src$ scp  -r  yhs@192.168.1.102:/home/yhs/ros2_ws/src/yhs_can_interfaces .
yhs@192.168.1.102's password:
package.xml                                   100%  837   323.5KB/s   00:00
CMakeLists.txt                                100% 1390   518.6KB/s   00:00
CtrlCmd.msg                                   100%   75    29.7KB/s   00:00
LWheelFb.msg                                  100%   60    26.5KB/s   00:00
IoFb.msg                                      100%  624   268.9KB/s   00:00
BmsFb.msg                                     100%   81     7.7KB/s   00:00
ChassisInfoFb.msg                             100%  260    75.8KB/s   00:00
RWheelFb.msg                                  100%   60    24.2KB/s   00:00
CtrlFb.msg                                    100%   79    30.8KB/s   00:00
IoCmd.msg                                     100%  251   100.6KB/s   00:00
FreeCtrlCmd.msg                               100%   92    41.3KB/s   00:00
BmsFlagFb.msg                                 100%  466   218.2KB/s   00:00
RecordPath.action                             100%  271   124.1KB/s   00:00
```

```
xwqf@ros2:~/ros2_ws/src$ scp  -r  yhs@192.168.1.102:/home/yhs/ros2_ws/src/navigation2/nav2_rviz_plugins  .
yhs@192.168.1.102's password:
recorded_path (copy).txt                      100%  689   277.6KB/s   00:00
package.xml                                   100% 1477   583.4KB/s   00:00
CMakeLists.txt                                100% 3443     1.4MB/s   00:00
goal_pose_updater.hpp                         100% 1201   503.6KB/s   00:00
goal_tool.hpp                                 100% 1356   583.3KB/s   00:00
ros_action_qevent.hpp                         100% 1731   199.9KB/s   00:00
particle_cloud_display.hpp                    100% 5392     1.5MB/s   00:00
flat_weighted_arrows_array.hpp                100% 3650     1.6MB/s   00:00
nav2_panel.hpp                                100%   11KB   3.8MB/s   00:00
goal_common.hpp                               100%  879   208.7KB/s   00:00
nav2_panel.cpp                                100%   53KB   8.8MB/s   00:00
flat_weighted_arrows_array.cpp                100% 5103     2.1MB/s   00:00
particle_cloud_display.cpp                    100%   12KB   3.2MB/s   00:00
goal_tool.cpp                                 100% 1498   539.9KB/s   00:00
settings.json                                 100%  110    47.6KB/s   00:00
plugins_description.xml                       100%  708   314.8KB/s   00:00
```

```
xwqf@ros2:~/ros2_ws/src$ ls
nav2_rviz_plugins  yhs_can_interfaces
```

(7) After copying the required files to your laptop, you can proceed with the compilation process. During compilation, you may encounter errors indicating missing dependencies. In such cases, you can switch to a Wi-Fi network with internet access (switch back to the robot's Wi-Fi network after installation) and install the dependencies one by one. Once all the dependencies are installed, you can proceed with the compilation.

```
xwqf@ros2:~/ros2_ws$ colcon  build  --symlink-install
```

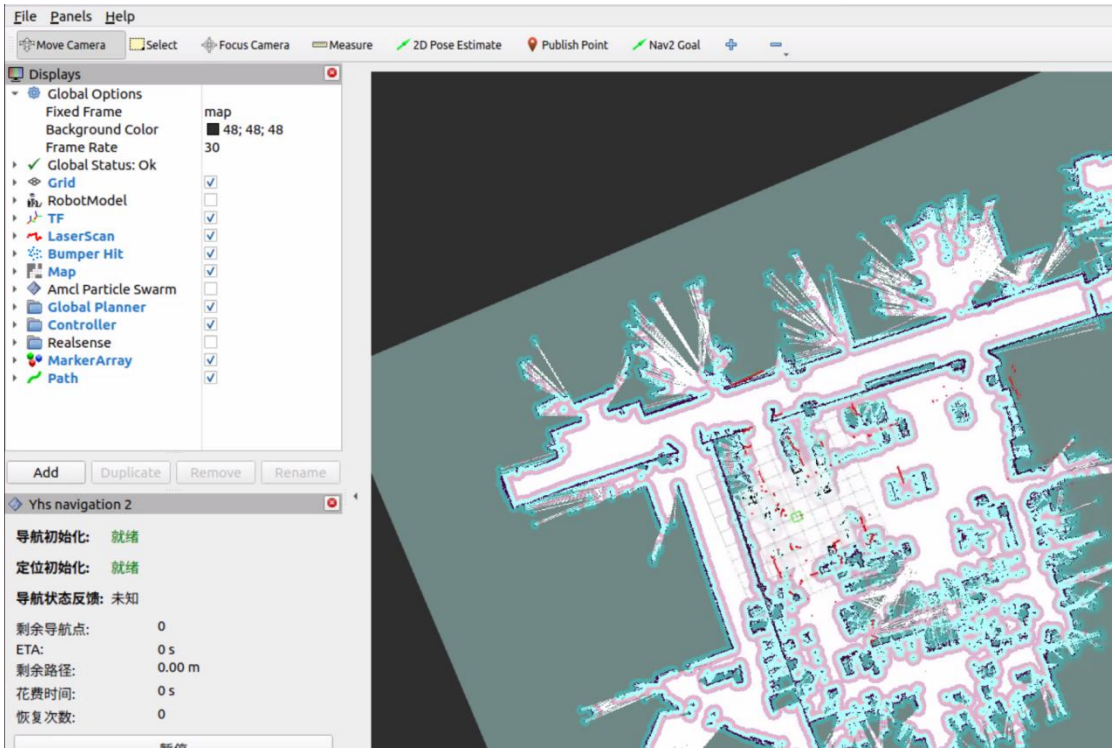
```
xwqf@ros2:~/ros2_ws$ colcon  build  --symlink-install
[0.695s] WARNING:colcon.colcon_core.package_selection:Some selected packages are already built in one or more un
derlay workspaces:
        'nav2_rviz_plugins' is in: /home/xwqf/ros2/install/nav2_rviz_plugins, /opt/ros/humble
        'yhs_can_interfaces' is in: /home/xwqf/ros2/install/yhs_can_interfaces
If a package in a merged underlay workspace is overridden and it installs headers, then all packages in the over
lay must sort their include directories by workspace order. Failure to do so may result in build failures or und
efined behavior at run time.
If the overridden package is used by another package in any underlay, then the overriding package in the overlay
 must be API and ABI compatible or undefined behavior at run time may occur.

If you understand the risks and want to override a package anyways, add the following to the command line:
        --allow-overriding nav2_rviz_plugins yhs_can_interfaces

This may be promoted to an error in a future release of colcon-override-check.
Starting >>> yhs_can_interfaces
Finished <<< yhs_can_interfaces [14.1s]
Starting >>> nav2_rviz_plugins
[Processing: nav2_rviz_plugins]
Finished <<< nav2_rviz_plugins [37.9s]

Summary: 2 packages finished [52.2s]
```

(8) After a successful compilation, you can run the navigation program on the industrial computer through remote login. On the terminal you opened in step (7) on your laptop, you can open rviz2. Please make sure to follow the steps in the image below (note the green text in the top left corner of the image) and be careful not to confuse the terminals.

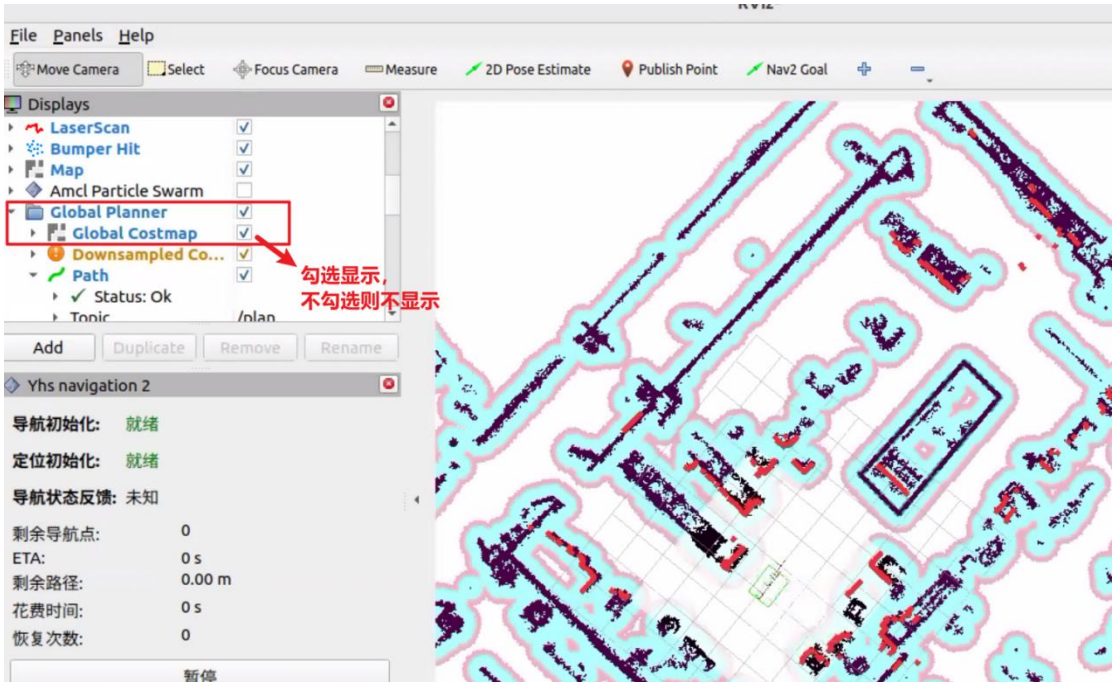
```
yhs@yhs-ros2: $ ros2  launch   yhs_nav2   yhs_nav2.launch.py
[INFO] [launch]: All log files can be found below /home/yhs/.ros/log/2023-12-01-16-03-43-668794-yhs-ros2-10068
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [nvilidar_ros2_node-4]: process started with pid [10086]
```

```
xwqf@ros2:~/ros2_ws$ source  install/setup.bash
xwqf@ros2:~/ros2_ws$
```

```
xwqf@ros2:~/ros2_ws$ rviz2
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
[INFO] [1701442923.107710528] [rviz2]: Stereo is NOT SUPPORTED
[INFO] [1701442923.107760747] [rviz2]: OpenGl version: 4.6 (GLSL 4.6)
[INFO] [1701442923.125054555] [rviz2]: Stereo is NOT SUPPORTED
[INFO] [1701442923.847406133] [rviz2]: Trying to create a map of size 1536 x 896 using 1 swatches
[INFO] [1701442923.852573258] [rviz2]: Trying to create a map of size 1536 x 896 using 1 swatches
[ERROR] [1701442923.859433989] [rviz2]: Vertex Program:rviz/glsl120/indexed_8bit_image.vert Fragment Program:rvi
z/glsl120/indexed_8bit_image.frag GLSL link result :
active samplers with a different type refer to the same texture image unit
[INFO] [1701442924.374924282] [rviz2]: Trying to create a map of size 120 x 120 using 1 swatches
[INFO] [1701442924.467995371] [rviz]: Message Filter dropping message: frame 'laser_link' at time 1701418004.949
 for reason 'discarding message because the queue is full'
[INFO] [1701442924.564127033] [rviz]: Message Filter dropping message: frame 'laser_link' at time 1701418005.048
 for reason 'discarding message because the queue is full'
[ERROR] [1701442926.365504367] [rviz2]: Lookup would require extrapolation into the future.  Requested time 1701
418007.946441 but the latest data is at time 1701418007.943432, when looking up transform from frame [laser_link
] to frame [map]
```




(9) To configure whether to display the global costmap in rviz2:




(10) After opening rviz2 on your laptop, you can perform various navigation operations. For detailed instructions on these operations, please refer to the navigation section mentioned above.

## 2. NOMACHINE remote control software

Prerequisites:

- A laptop
- NOMACHINE software downloaded and installed. There are respective versions available for both Windows and Ubuntu systems.

(1) Connect to the Wi-Fi network on the car first. Before proceeding with the following steps, SSH into the industrial computer from your laptop and enter the following two commands.

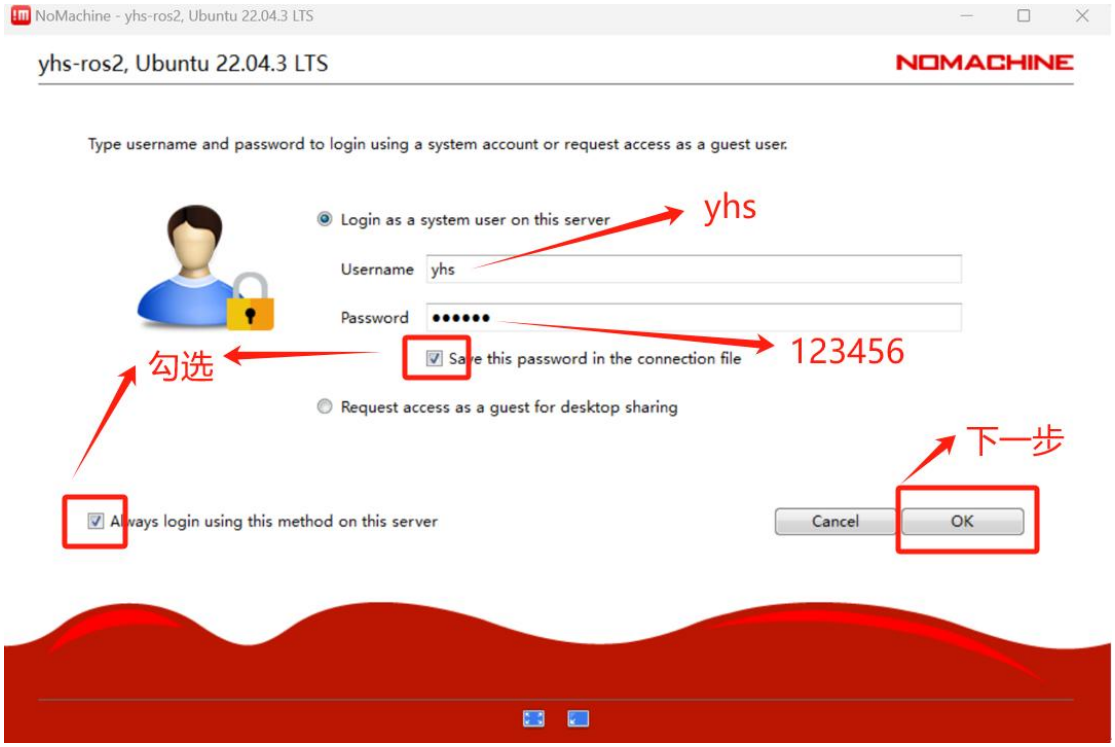
| |
|---|
| sudo systemctl stop gdm3 |
| sudo systemctl restart nxserver.service |

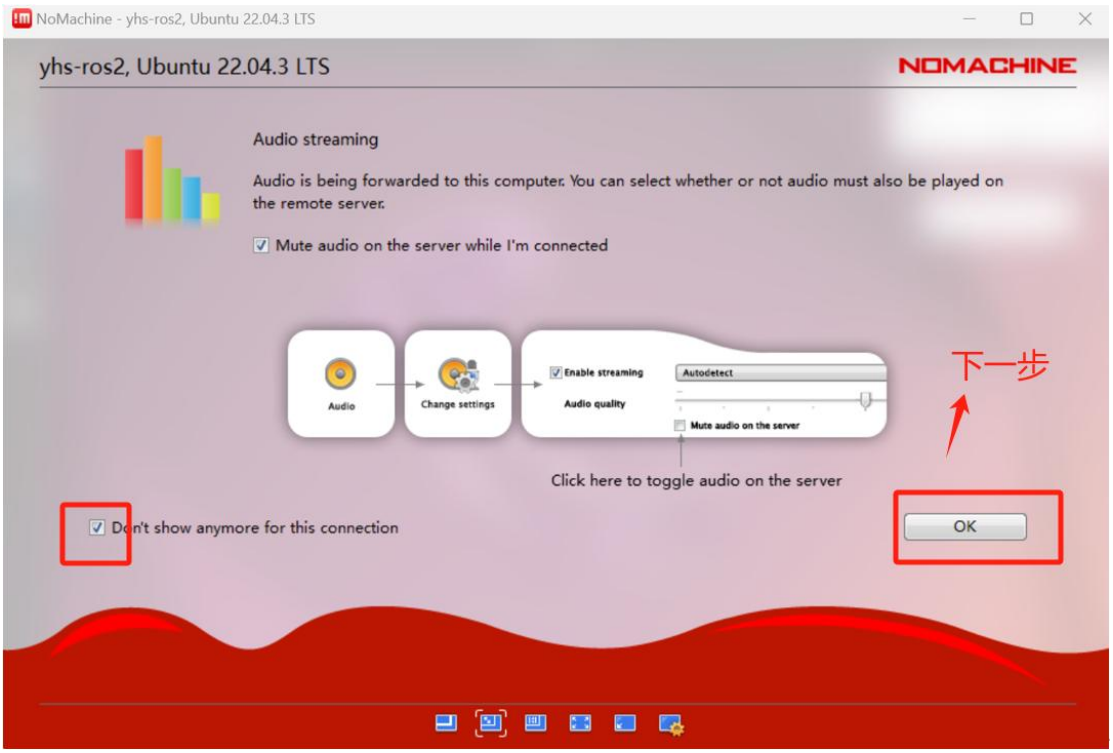
```
yhs@yhs-ros2:~$ sudo systemctl stop gdm3
```

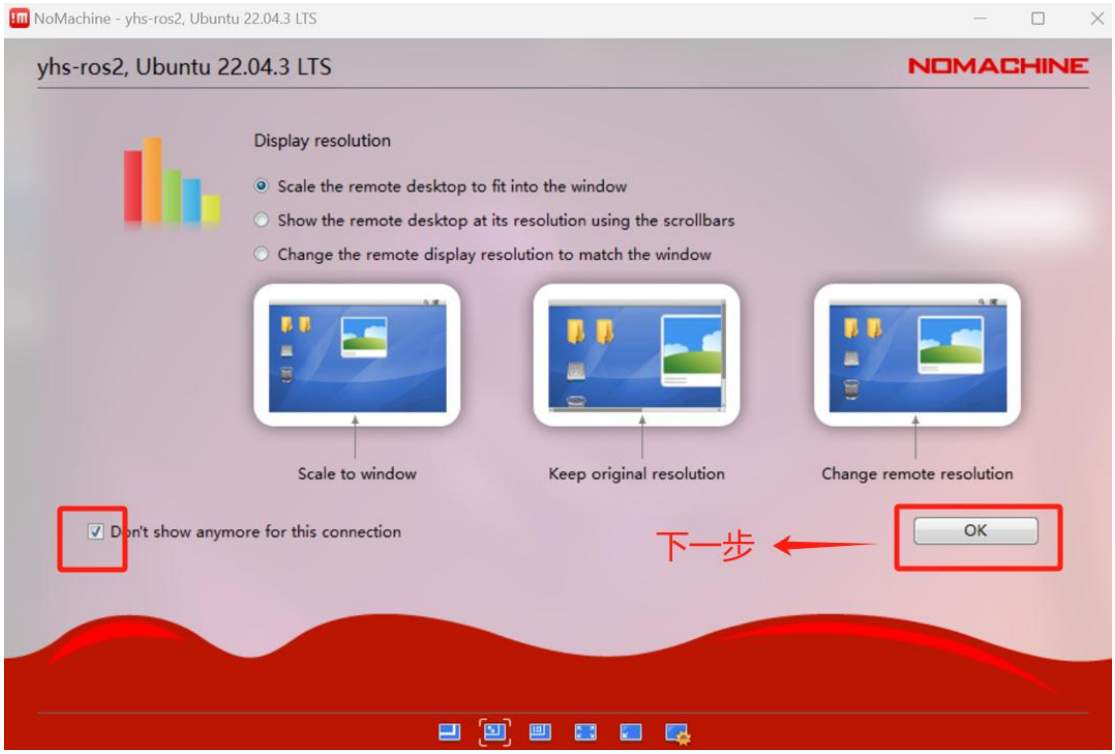
```
yhs@yhs-ros2:~$ sudo systemctl restart nxserver.service
```

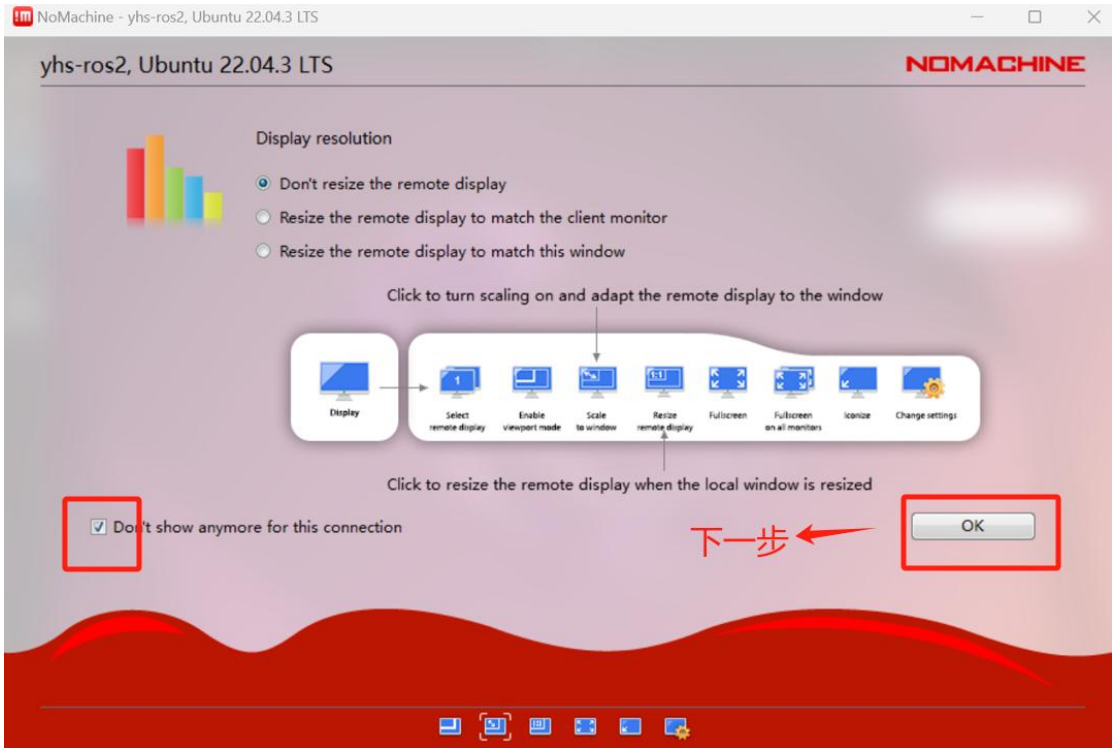
(2) Open the NOMACHINE software, and once you are on the main interface, you will be able to see the information about the Ubuntu system installed on the industrial computer on the robot.


(3) Double-click on the icon to initiate the connection. The process is as follows:

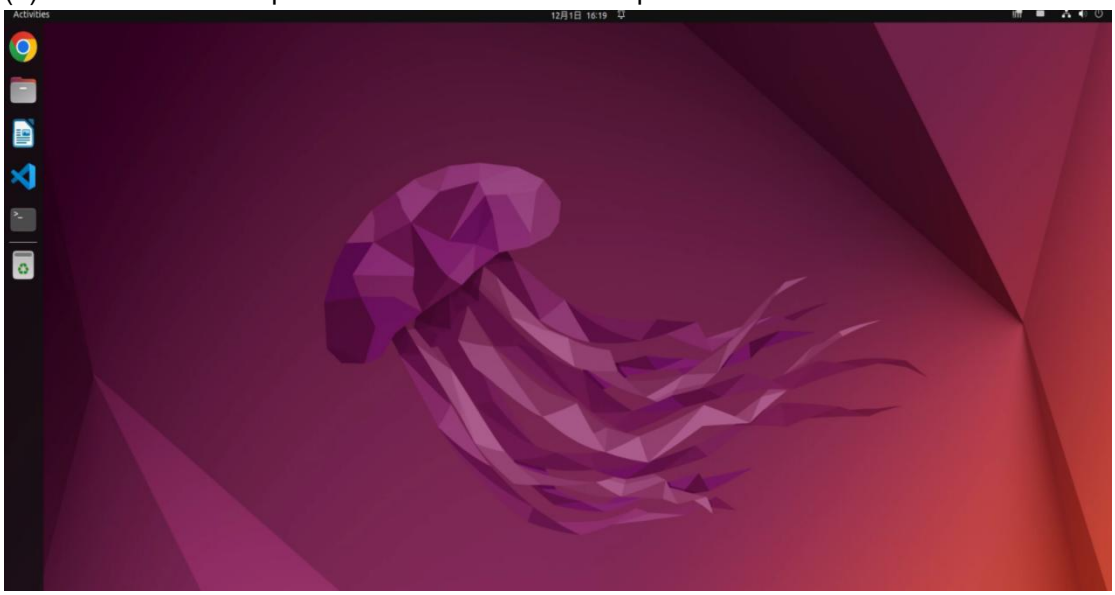



NoMachine - yhs-ros2, Ubuntu 22.04.3 LTS
yhs-ros2, Ubuntu 22.04.3 LTS
NOMACHINE
Audio streaming
Audio is being forwarded to this computer. You can select whether or not audio must also be played on the remote server.
Mute audio on the server while I'm connected
Audio
Change settings
Enable streaming
Autodetect
Audio quality
Mute audio on the server
Click here to toggle audio on the server
下一步
Don't show anymore for this connection
OK


NoMachine - yhs-ros2, Ubuntu 22.04.3 LTS
yhs-ros2, Ubuntu 22.04.3 LTS
NOMACHINE
Display resolution
Scale the remote desktop to fit into the window
Show the remote desktop at its resolution using the scrollbars
Change the remote display resolution to match the window
Scale to window
Keep original resolution
Change remote resolution
Don't show anymore for this connection
下一步
OK



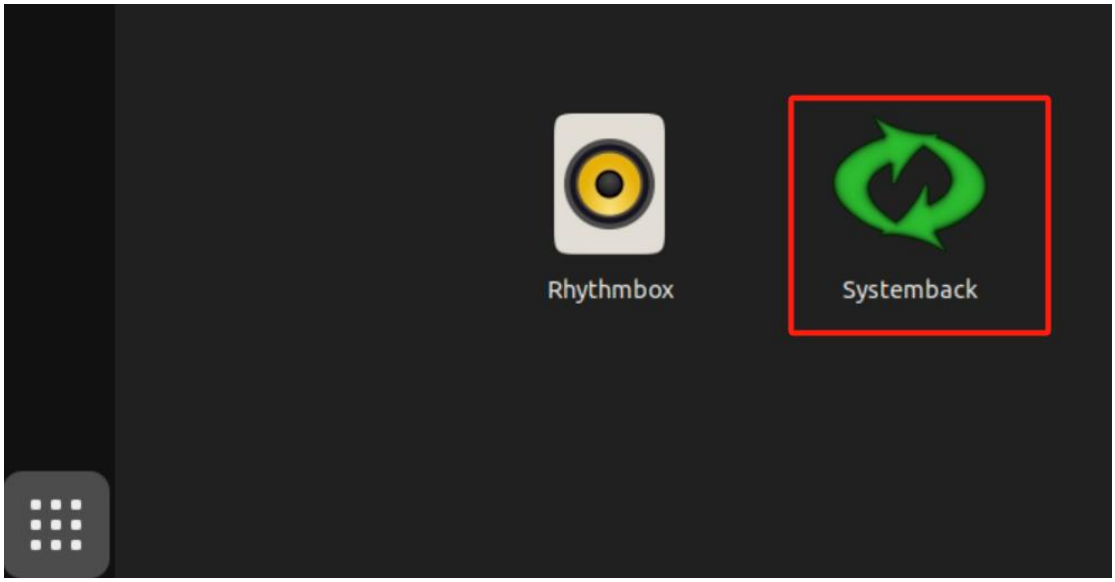

(4) Access the desktop of the robot's industrial computer.

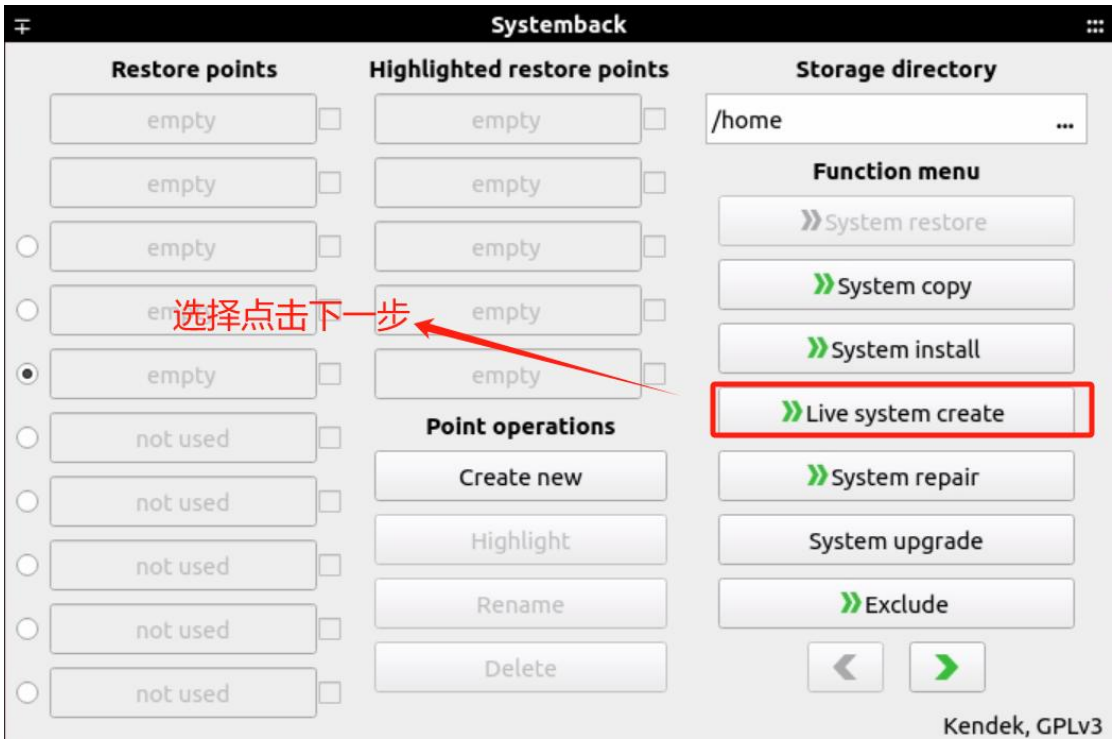


(5) Once you are on the desktop, the operations are the same as using the industrial computer directly connected to a screen. However, please note that certain keyboard shortcuts may not be available in this remote setup.

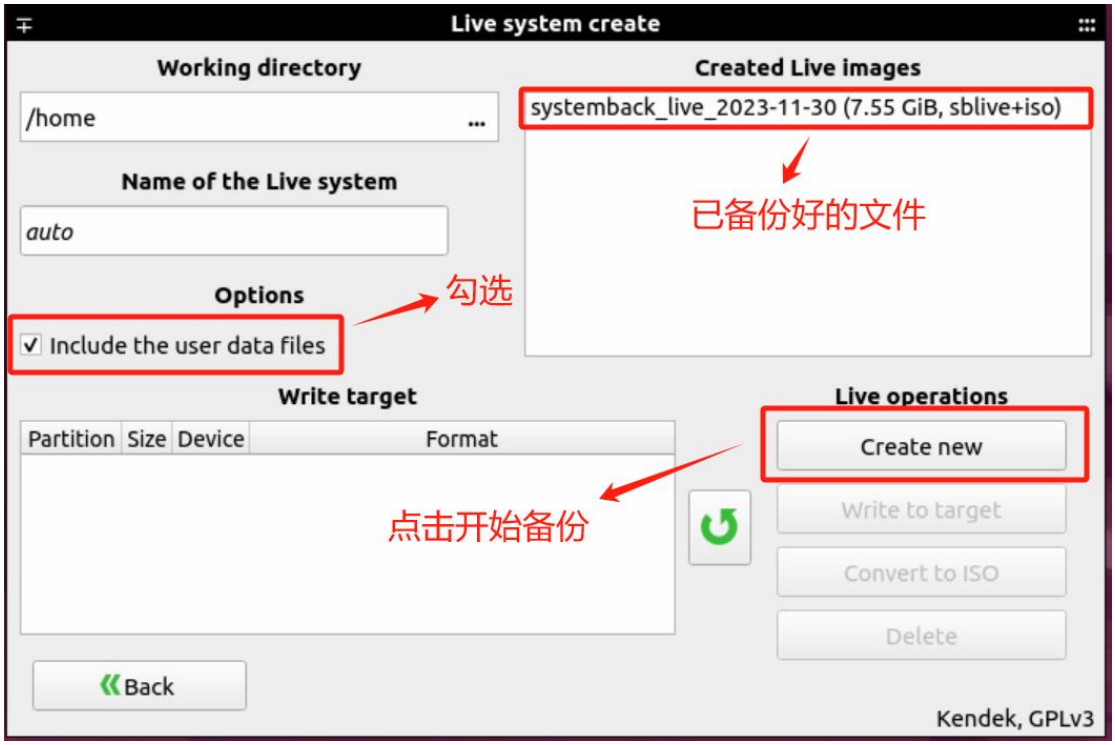
# Appendix 2: System Backup

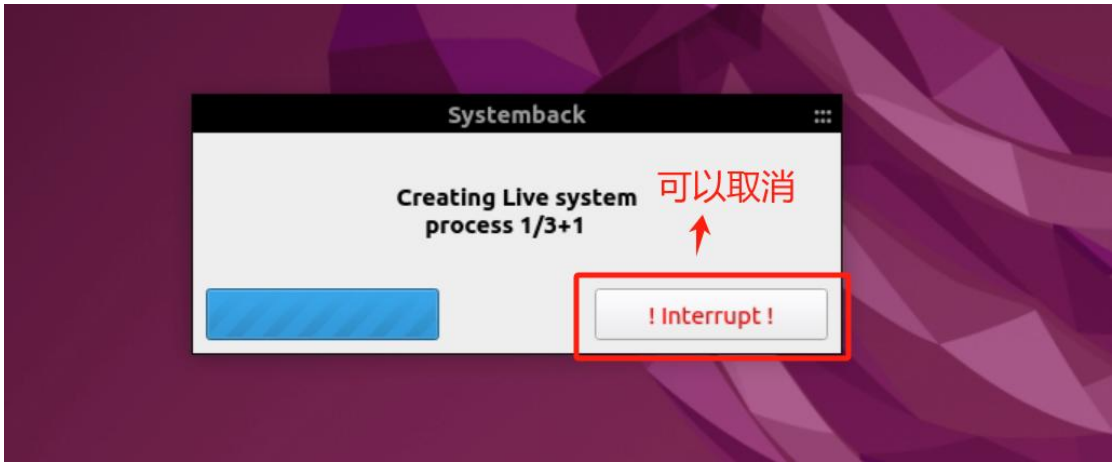
(1) On the desktop, search and select "Systemback" located at the bottom left corner. Enter your password and choose "Live system create" to proceed.



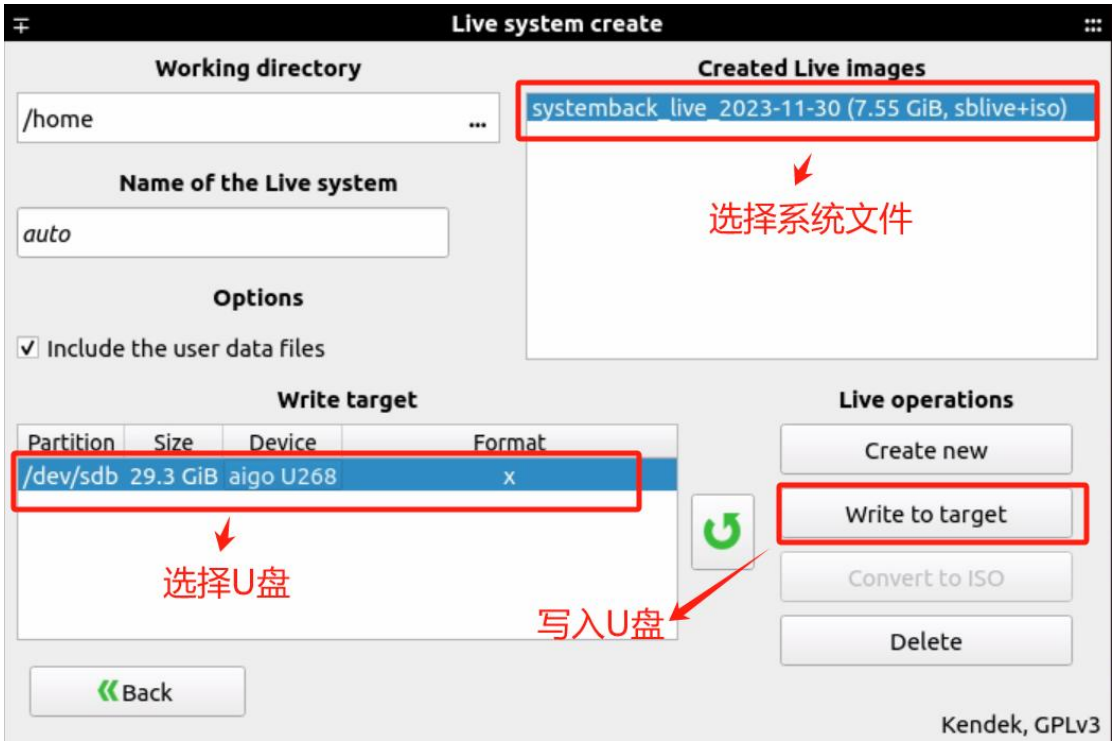




(2) Start the system backup. The backup process may take a long time, so please wait for it to complete. During the backup process, avoid running programs or performing other operations.






(3) After the backup is complete, insert a USB flash drive into the industrial computer. The system will be written to the USB flash drive (note that the USB flash drive will be formatted and cleared before the write process). The writing process may take a long time, so please wait for it to complete.



# Appendix 3: Quick Guide of Groot Software

During the navigation startup process, a Behavior Tree (BehaviorTree) XML format file is loaded. The Behavior Tree XML file can be edited and monitored using Groot software. For detailed instructions and operations, please refer to "https://www.behaviortree.dev". Here is a brief explanation:

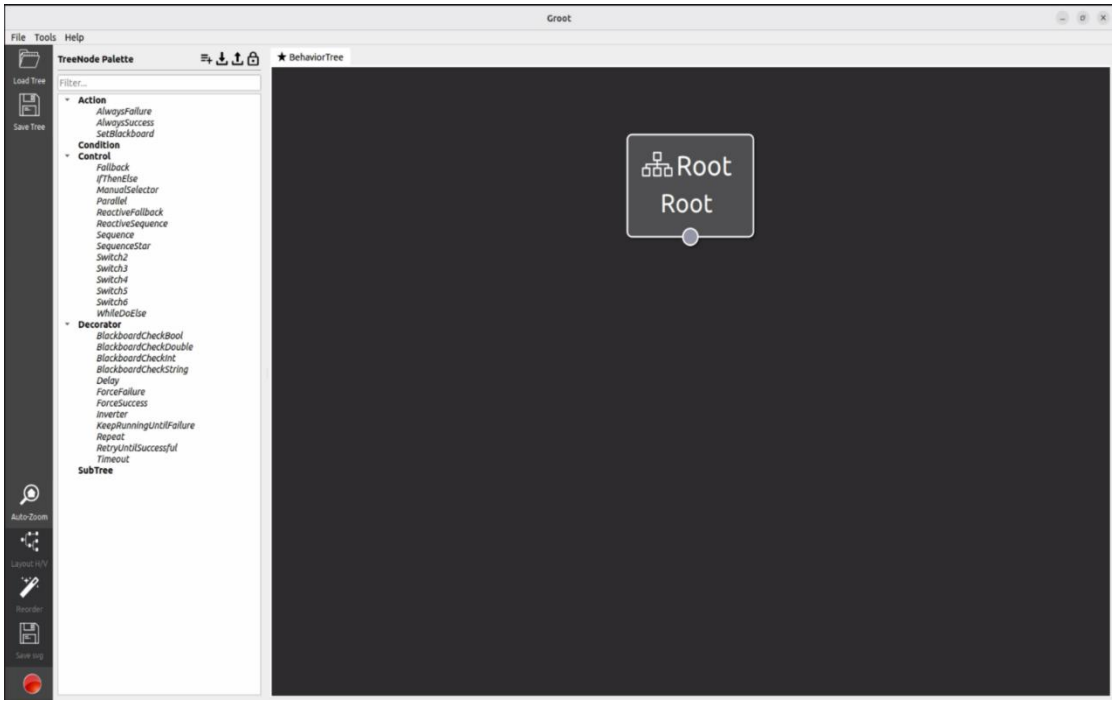
1. Edit Mode:

(1) Open a terminal and navigate to "/home/yhs/Groot/build".

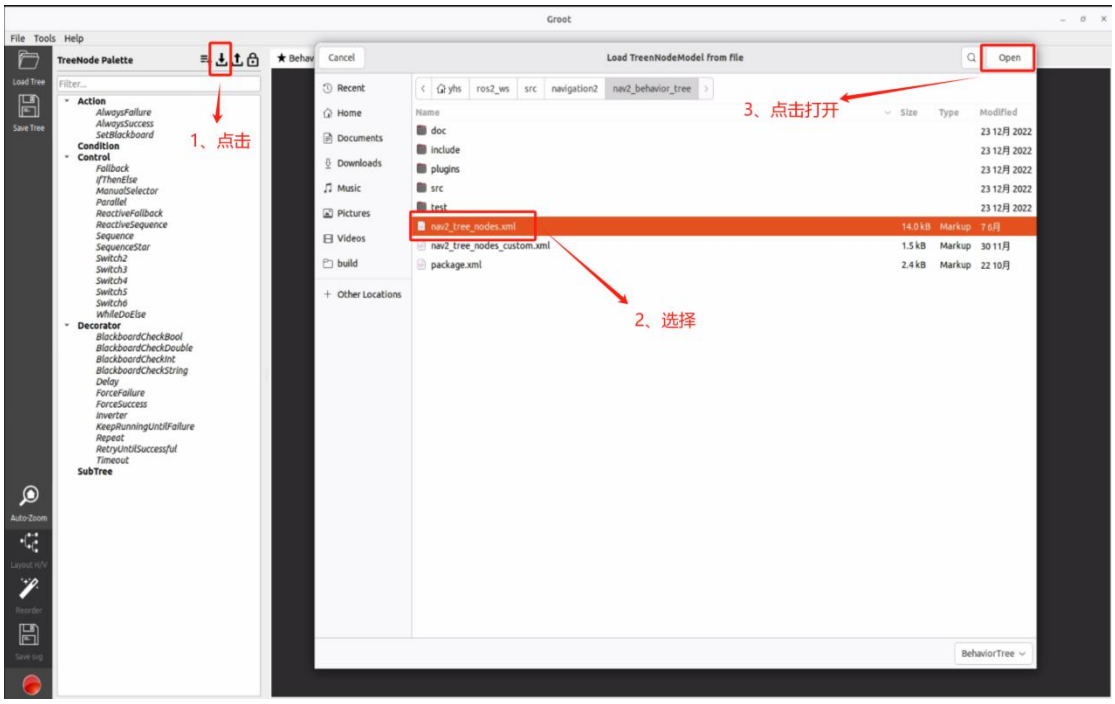
```
cd  /home/yhs/Groot/build
```

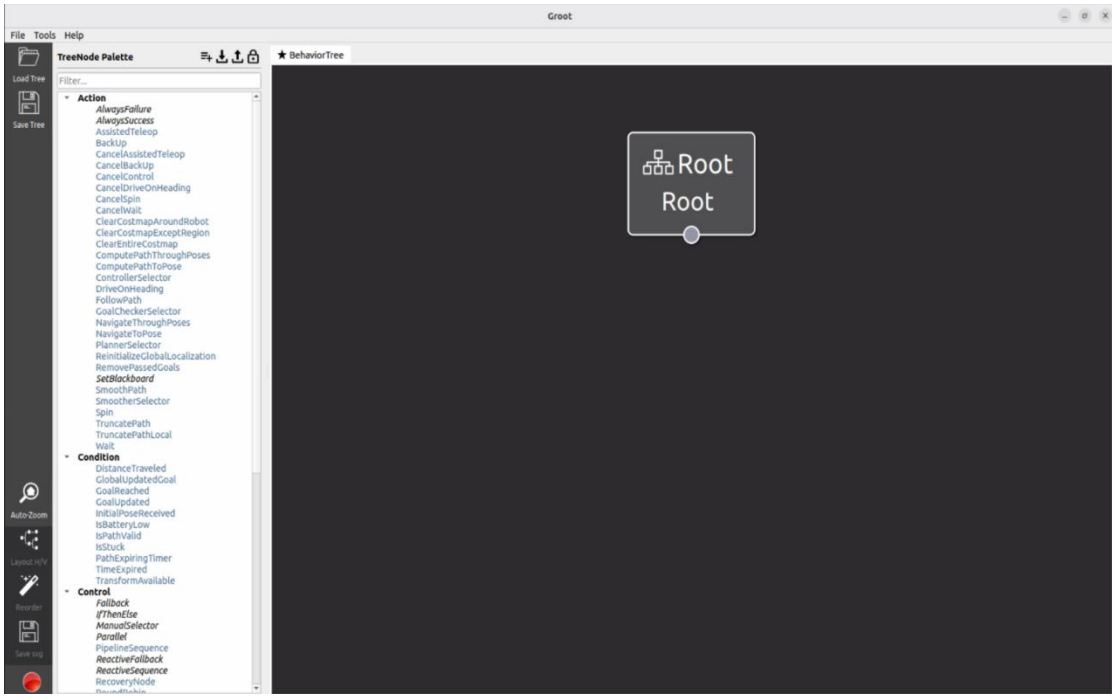

(2) Run the Groot software by entering the command.

```
./Groot
```

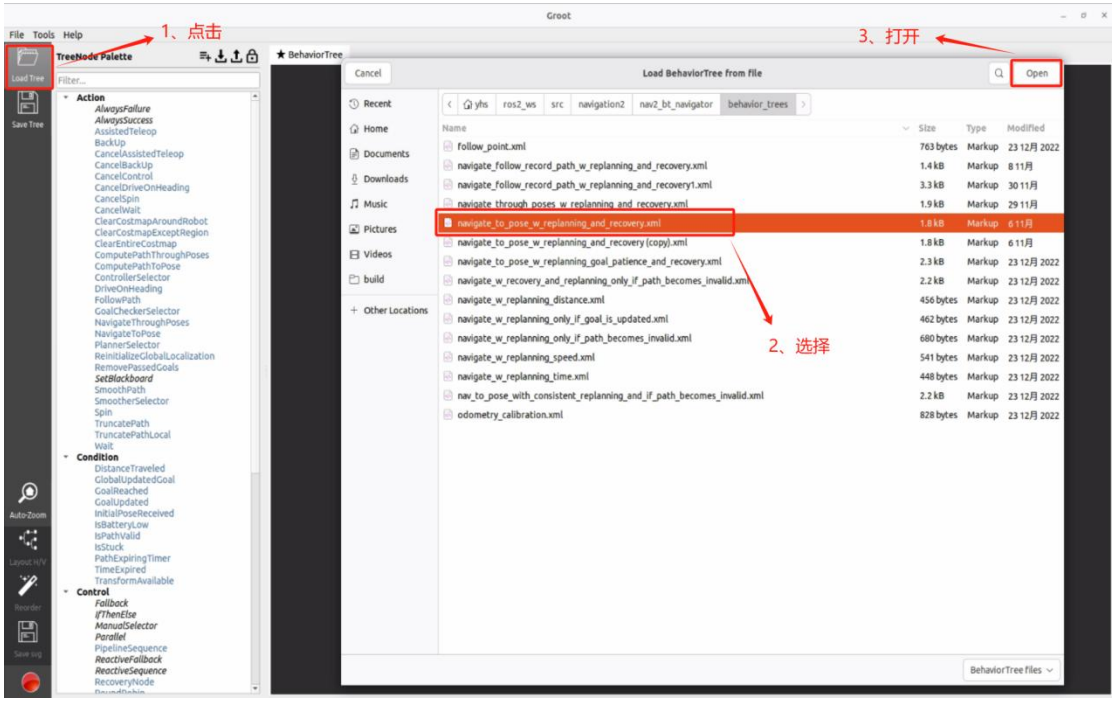


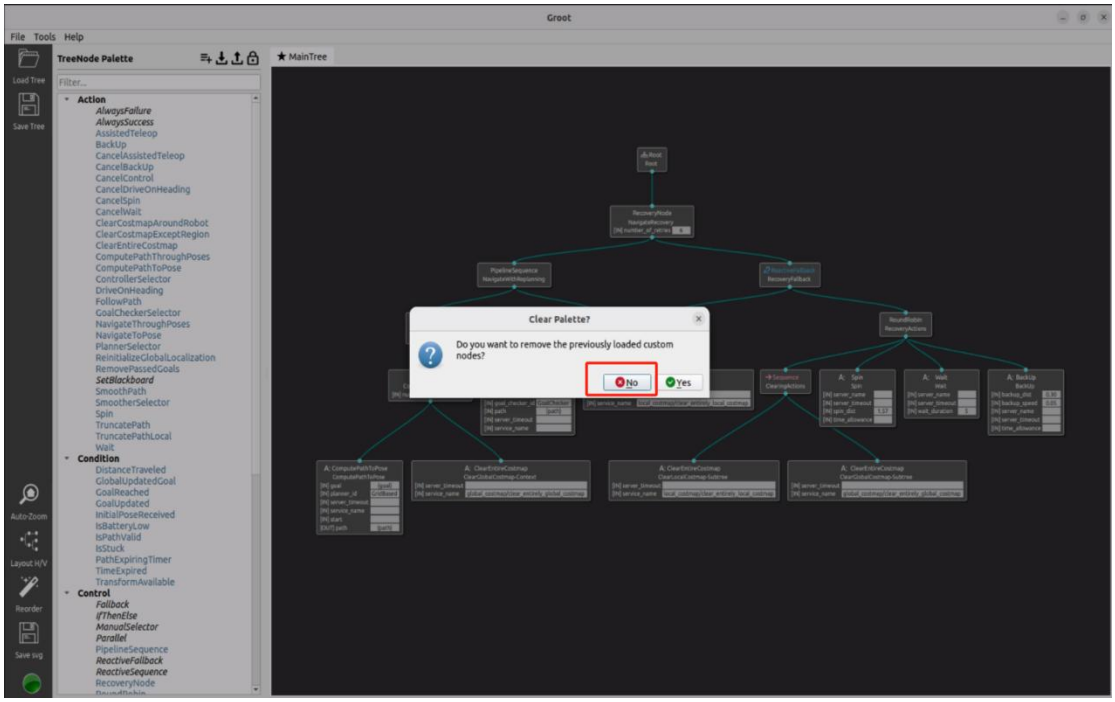

(3) Open the "nav2_tree_nodes.xml" file as shown in the following image.



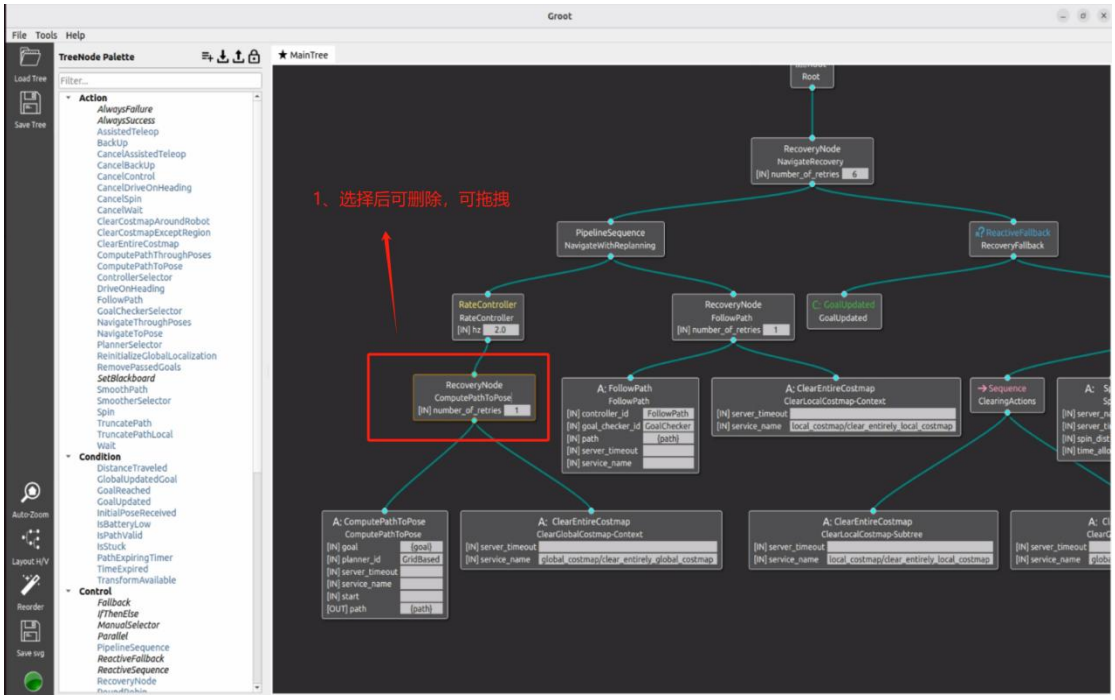


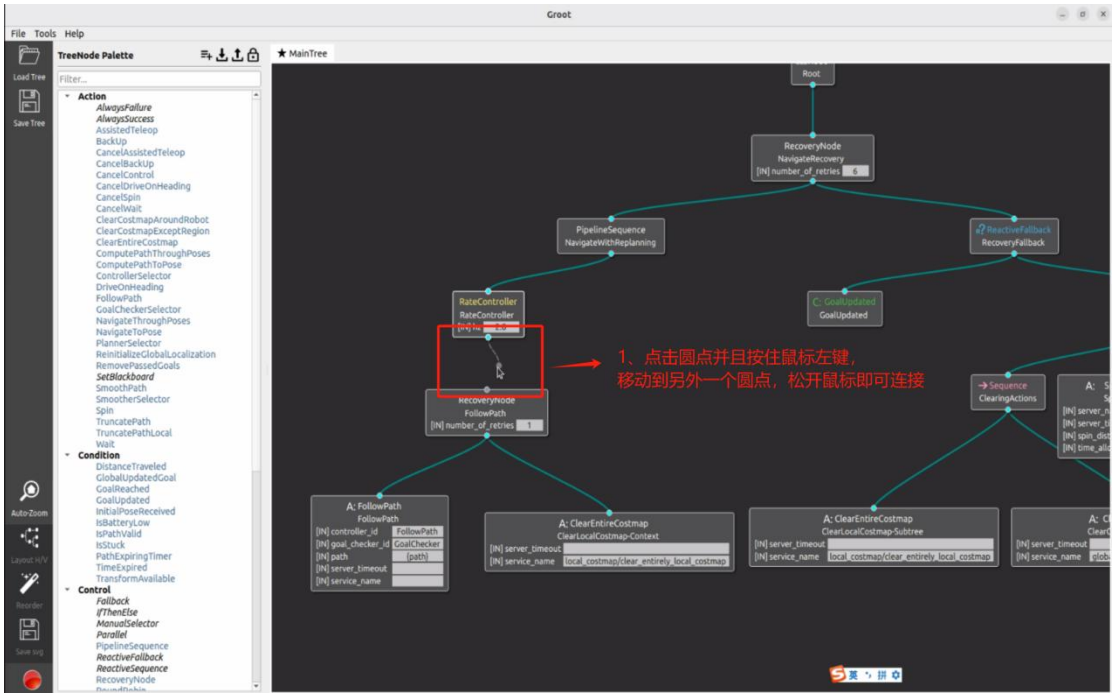

(4) Open the "navigate_to_pose_w_replanning_and_recovery.xml" file located in the directory "/home/yhs/ros2_ws/src/navigation2/nav2_bt_navigator/behavior_trees" for reference.



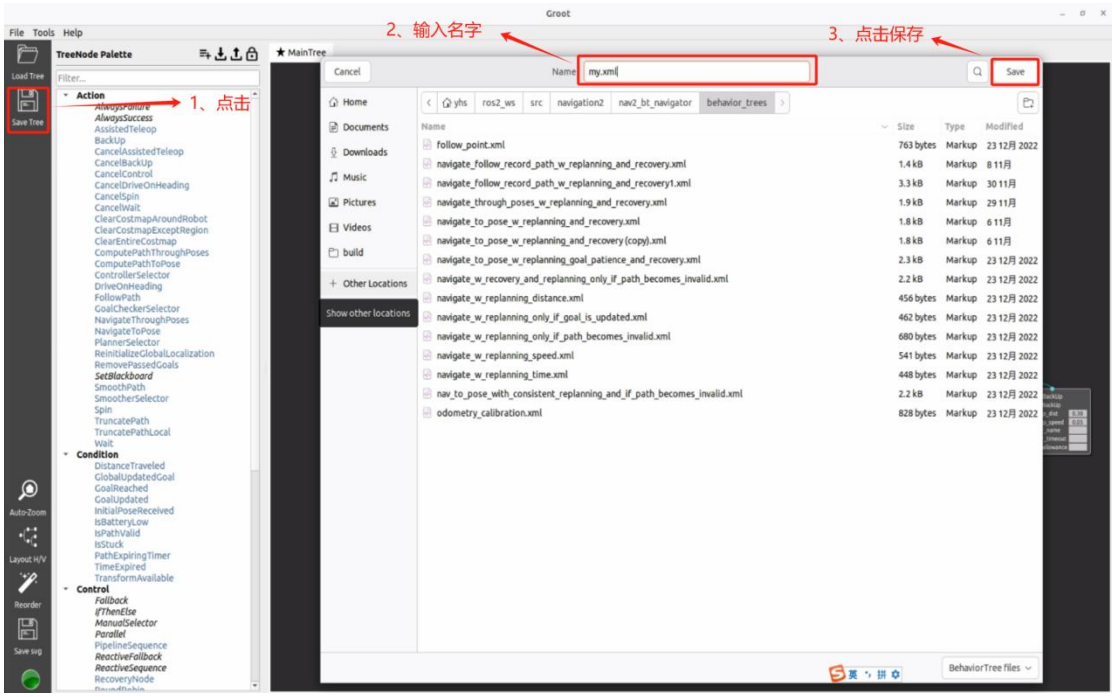



(5) Edit the file according to the navigation requirements.





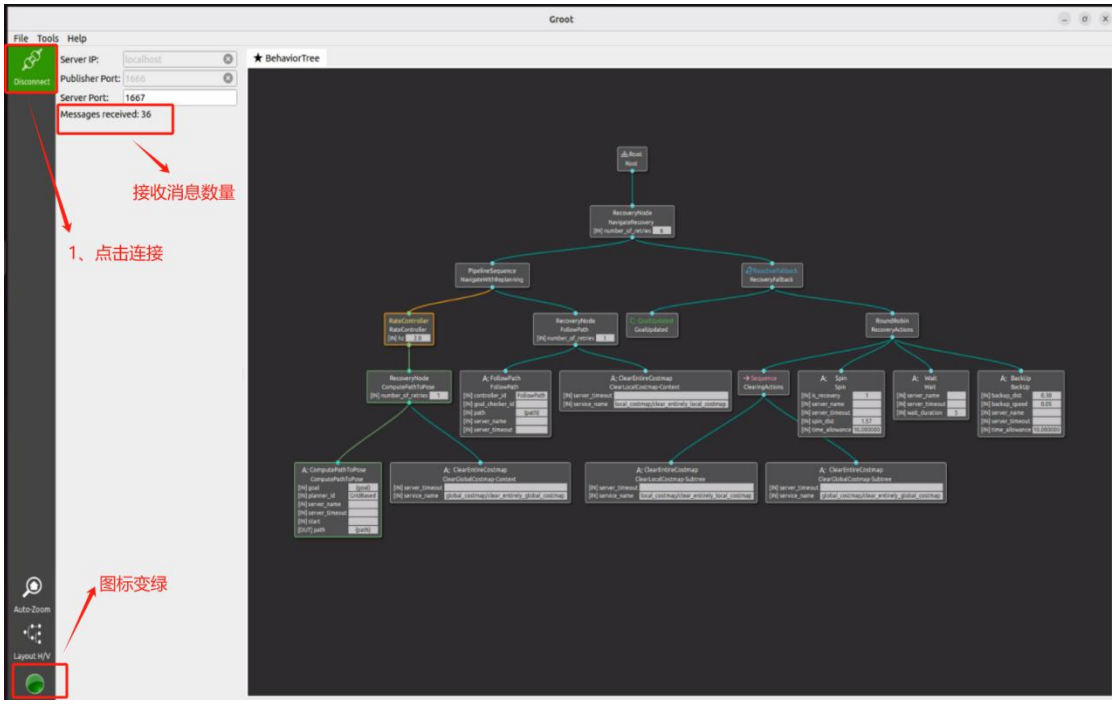

(6) Save the XML file.



2. Monitoring Mode:

(1) Open Groot and select the monitoring mode.


(2) Start the navigation and send the navigation points. Connect to Groot for monitoring.

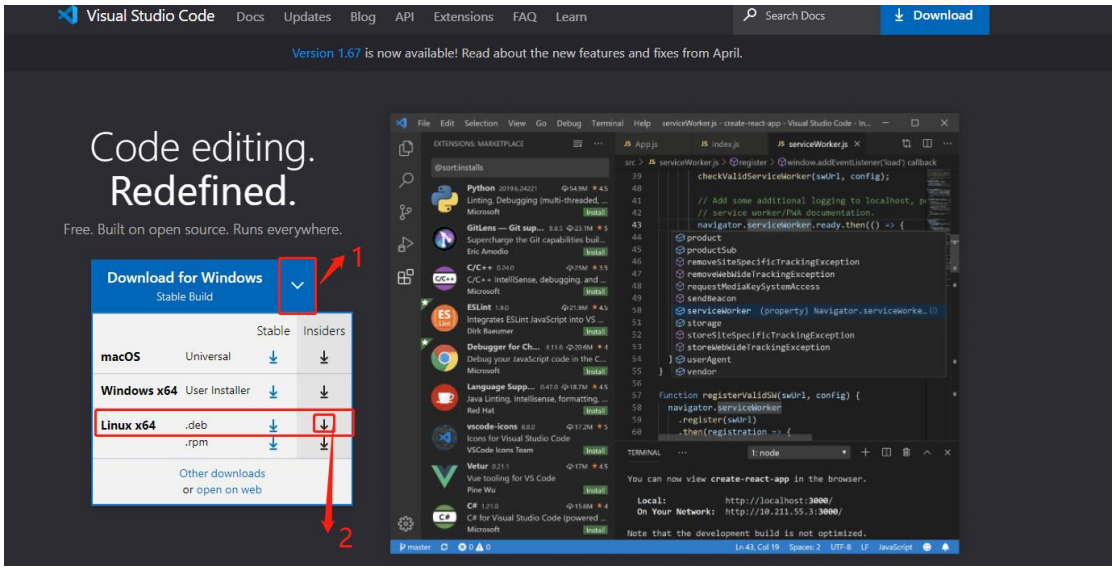



(3) For different types of navigation, you may need to restart the navigation program to enable monitoring.

# Appendix 4: Installing VSCode

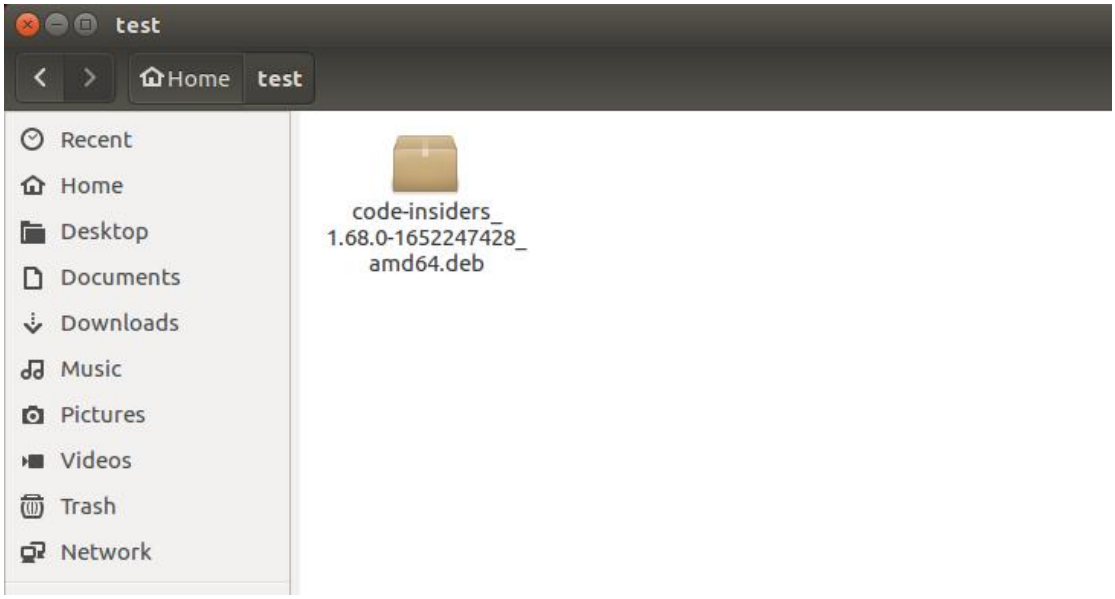
The VScode is already installed in robot's industrial computer, and the following steps are for demonstration purposes only.

1. Go to the official website (https://code.visualstudio.com/) and download the Linux x64 deb installation package.



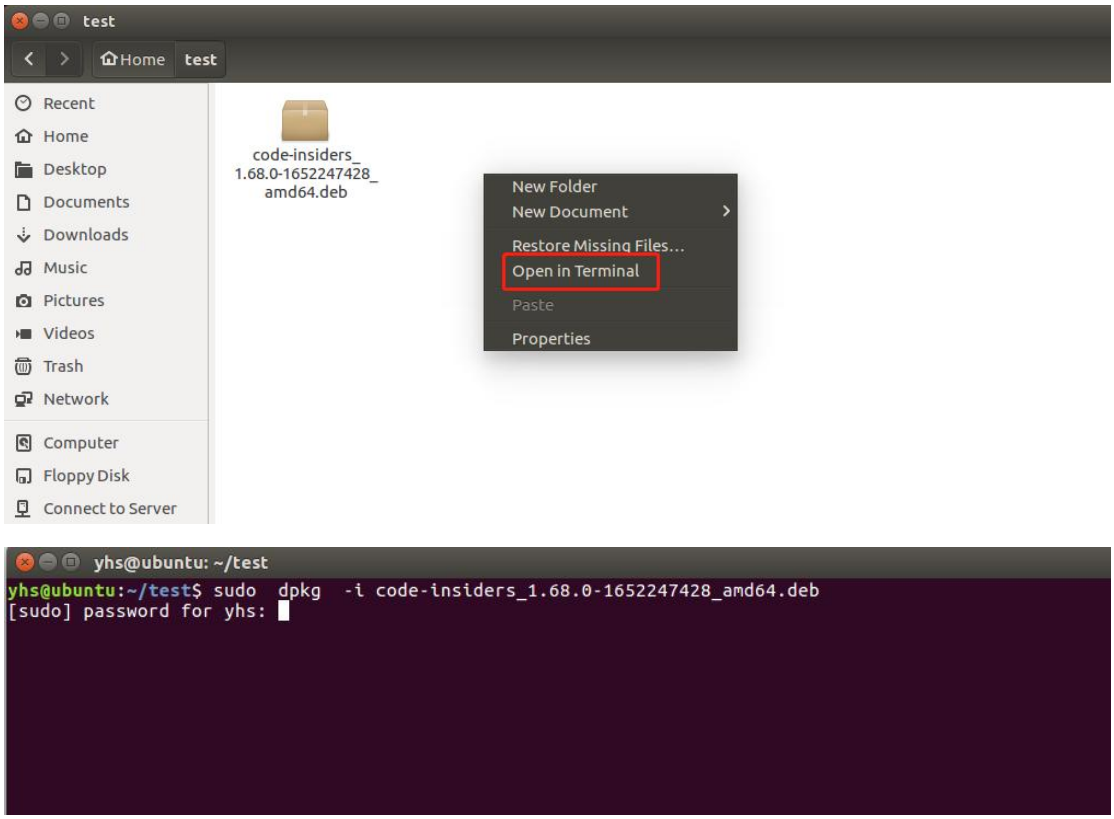

2. Let's assume that we have downloaded "code-insiders_1.68.0-1652247428_amd64.deb".

cc



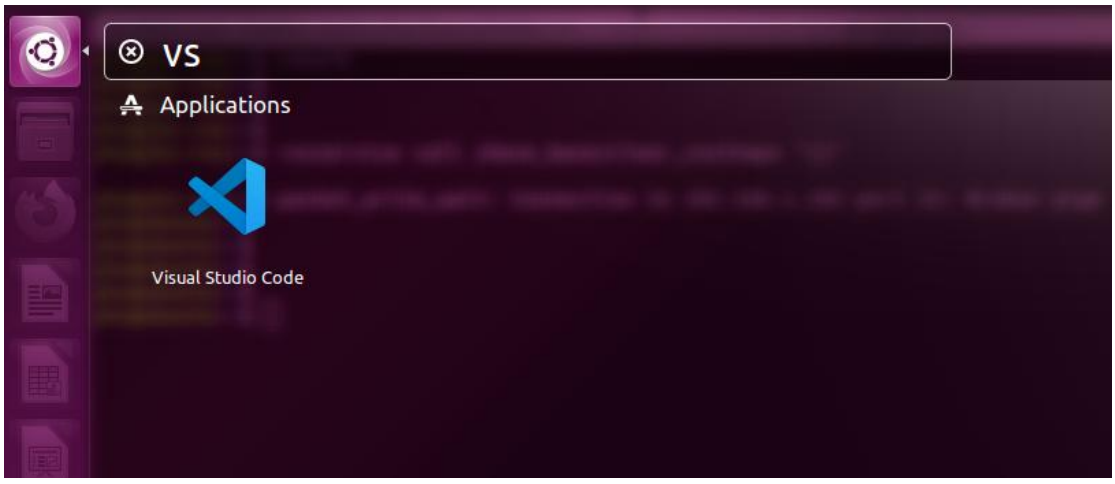
3. Right-click on the directory where the installation package is located and select "Open in Terminal." Enter the command and press Enter.

```
sudo  dpkg  -i code-insiders_1.68.0-1652247428_amd64.deb
```




4. Enter the root password and wait for the installation to complete.

5. Open VS Code and click on the search bar. Type "vs" and you will see the software icon. Click on the icon to open the software. At this point, the software installation is complete.



电话：0755-28468956

传真：0755-28468956

网址 www.yuhesen.com 邮件：sales01@yuhesen.com

地址：深圳市龙岗区宝龙六路 1 号创维群欣科技园 1 栋 5 楼

YUHESEN

深圳煜禾森科技有限公司

Shenzhen yuhesen Technology Co., Ltd.