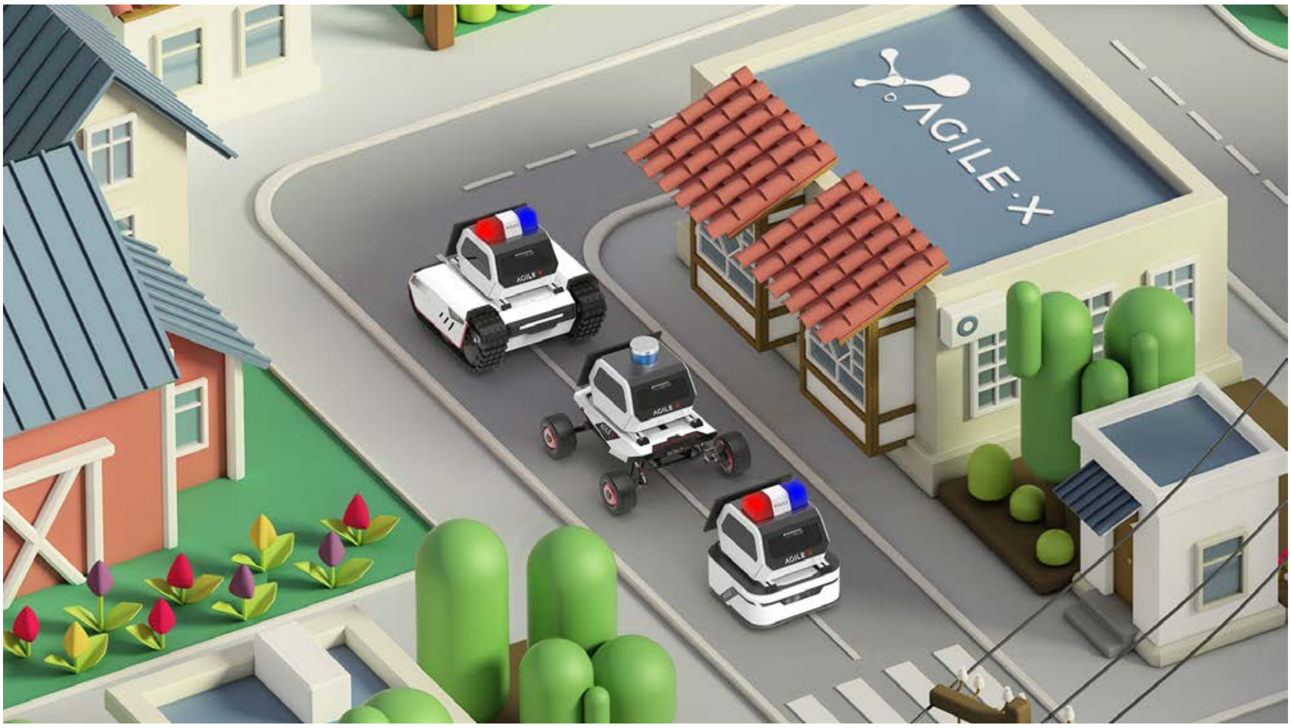
AGILE·X
Robotics

ROS2 EDU KIT

Génération
ROBOTS



1. Kit instruction

ROS2 NAV Kit is a customized ROS2 developer entry version and advanced kit, developed by Songling Robotics for ROS2 scientific research and education applications.

This kit is based on Songling Robotics' ROS ecosystem with integration of high performance, high precision control in the LiDAR and multi-sensor collocation, which can realize the mobile robot motion control, communication, navigation, map building and so on. We provide perfect developer documentation and DEMO resources.

It's light and portable, fully science and technology of industrial design, exclusive custom sensor bracket, that provides the best experimental platform of rapid ROS secondary development for education and scientific research, product pre-research, subject, product demonstration and other multi-direction applications .

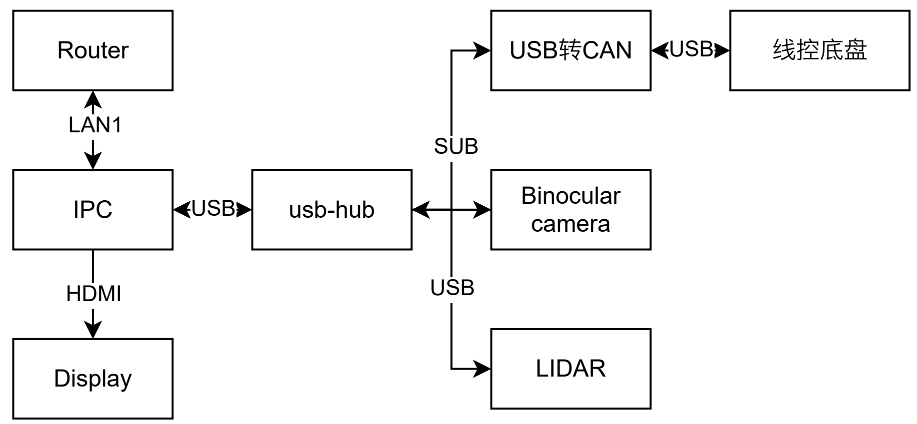
2. Hardware configuration

2.1 Upload configuration list

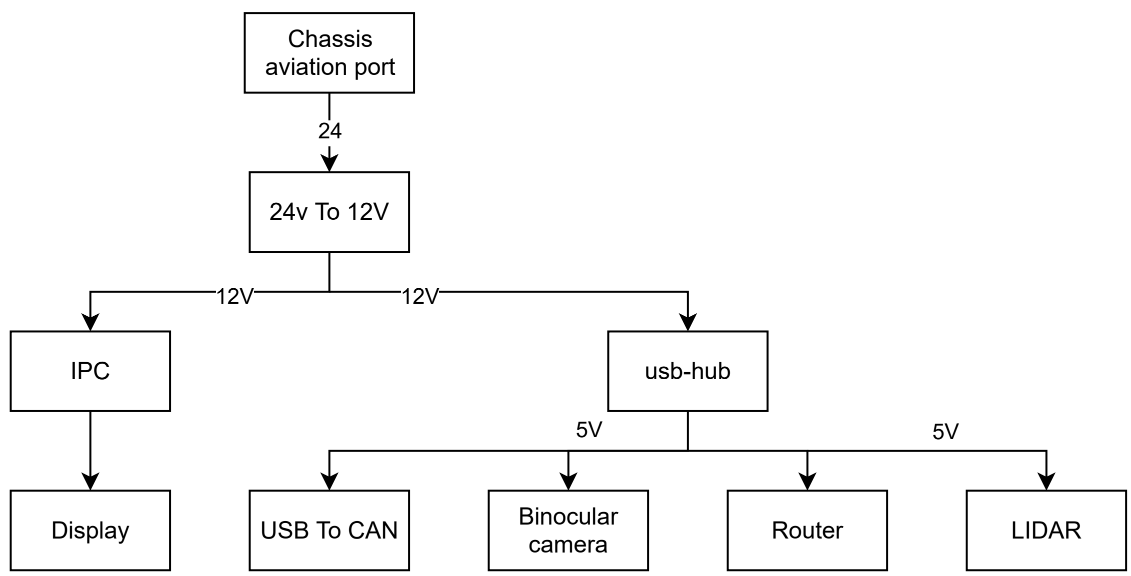
Lite:

No.	Name	Items	Description
1	Lite	IPC	minipc i5 16G 256
2		Single line LIDAR	G4
3		Binocular camera	RealSense D435
5		Displayer	14inch IPS portable 1920*1080 HDMI
6		Keyboard	k400 Plus
7		Router	GL.iNet AR750s
8		USB HUB	USB-HUB 12V-Power , divided to 7 USB ports
9		Bracket	
11		Power regulator1	12V to 5V, 15A /Aluminum
10		Power regulator2	24V（10-40）V to 12V，15A/Aluminum

Data flow for Lite version:



Power supply for Lite version:

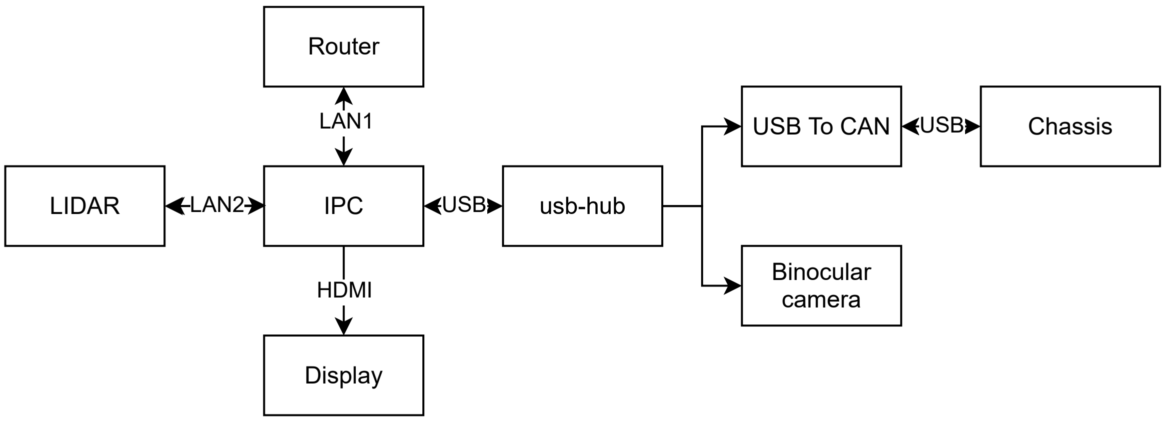


Lite Pro version configuration:

No.	Name	Items	Description
1	Lite Pro	IPC	minipc i7 16G 512G

8		Keyboard	k400 Plus
7		Router	GL.iNet AR750s
9		USB HUB	USB-HUB 12V-Power , divided to 7 USB ports
10		Bracket	Exclusive bracket
11		Power regulator	24V（10-40）V to 12V, 15A/Aluminum

Lite Pro version data flow:



Note： Lite Pro version LIDAR must be plugged to the IPC's LAN0 port.

Lite Pro version power supply:

SCOUT MINI is a four-wheel drive smart mobile chassis , with strong off-road performance。Its small size truly achieves nimble driving. SCOUT MINI inherits the advantages of SCOUT four-wheel differential chassis series, such as four-wheel drive, independent suspension and in-situ spin, and has made innovations in the design of hub motor. The minimum turning radius of the chassis is 0M, and the climbing Angle is close to 30 degrees. The SCOUT MINI is half the size of the SCOUT, while still providing excellent off-road performance and achieving a breakthrough 10.8km /h high-speed, accurate, stable and controllable power control system.

SCOUT MINI development platform has its own control core, supports standard CAN bus communication, as well as various external devices.On this basis, it supports ROS and other secondary development and more advanced robot development system to get access. Equipped with standard model airplane remote control and 24V15AH lithium battery power supply, the endurance can be up to 10KM. Stereo camera, liDAR, GPS, IMU, manipulator and other equipment can be optionally added to SCOUT MINI as extended applications. So it can be applied to unmanned inspection, security, scientific research, exploration, logistics and other fields.

Parameter Types	Items	Values
Mechanical specifications	L × W × H (mm)	615x580x245
	Wheelbase (mm)	452
	Front/rear wheel base (mm)	490
	Weight of vehicle body (kg)	22.5
	Battery type	Lithium battery 24V 15AH
	Motor	DC brushless 4 X 150W
	Drive type	Independent four-wheel drive
	Suspension	Independent suspension with rocker arm
	Steering	Four-wheel differential steering
	Safety equipment	Servo brake/anti-collision tube
Motion	No-load highest speed (km/h)	10.8
	Minimum turning radius	Be able to turn on a pivot
	Maximum climbing capacity	30°
	Minimum ground clearance (mm)	115
Control parameter	Control mode	Remote control Command control mode
	RC transmitter	2.4G/extreme distance 1km
	Communication interface	CAN

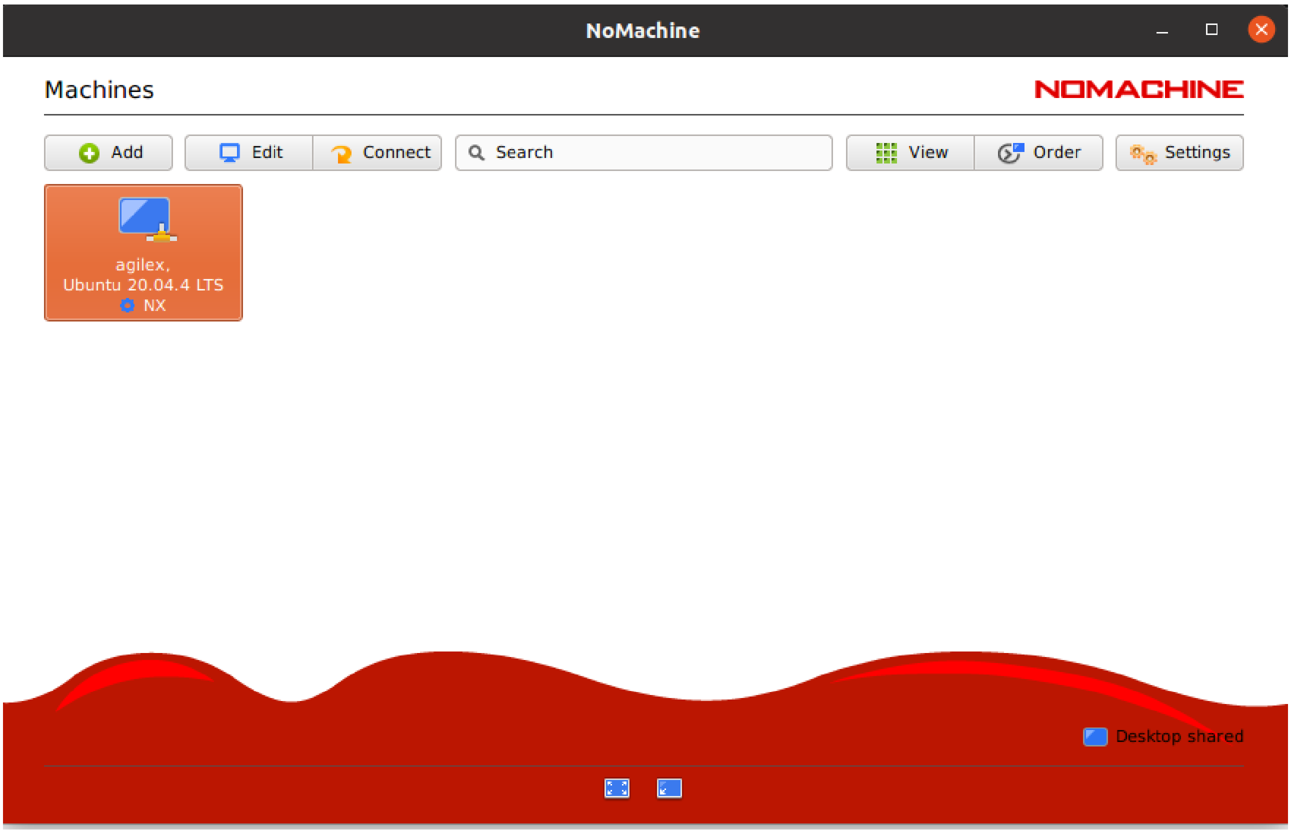
3. Software introduction

System login information:

Username	agilex
Password	agx
Router password	12345678

It can be developed through the keyboard and displayer of the IPC that come with the kit, or remotely through the LAN configuration and Nomachine software.

Configure the remote development and download Nomachine software on your computer, link: https://www.nomachine.com/. Connect to the router in the vehicle. Wifi name: Gl-ar750-xx. Then enter the password to open Nomachine software, click Connect, input the user name and password to achieve remote login.



This kit is developed on ros-foxy version. The workspace locates at: /home/agilex/agilex_ws. The main function packages include:

Package name	Function
navigation2	Navigation stack
rtabmap_ros rtabmap	Visual SLAM package, provide visual mapping and positioning.
scout_base, ugv_sdk	ROS2 and Scout MINI chassis communication

	package.
ydlidar_ros2	Drive for EAI single line LIDAR（Lite）.
rslidar_msg rslidar_sdk	Drive for 16-lines LIDAR (Pro).
pointcloud_to_laserscan	Point cloud data converted to LIDAR (Pro).
vision_opencv	OpenCV and ROS2 image data conversion.
realsense-ros	ROS2 driving package for Realsense-d435 camera.

Note: This kit has two versions: Lite and Pro. The features marked with Lite or Pro only exist in the related version.Users shall need to pay attention to the fact that two version are not compatible in the use.

The main launch file for this kit is in the scout_bringup folder:

```
cd /home/agilex/agilex_ws/src/scout_ros2/scout_bringup/launch
```

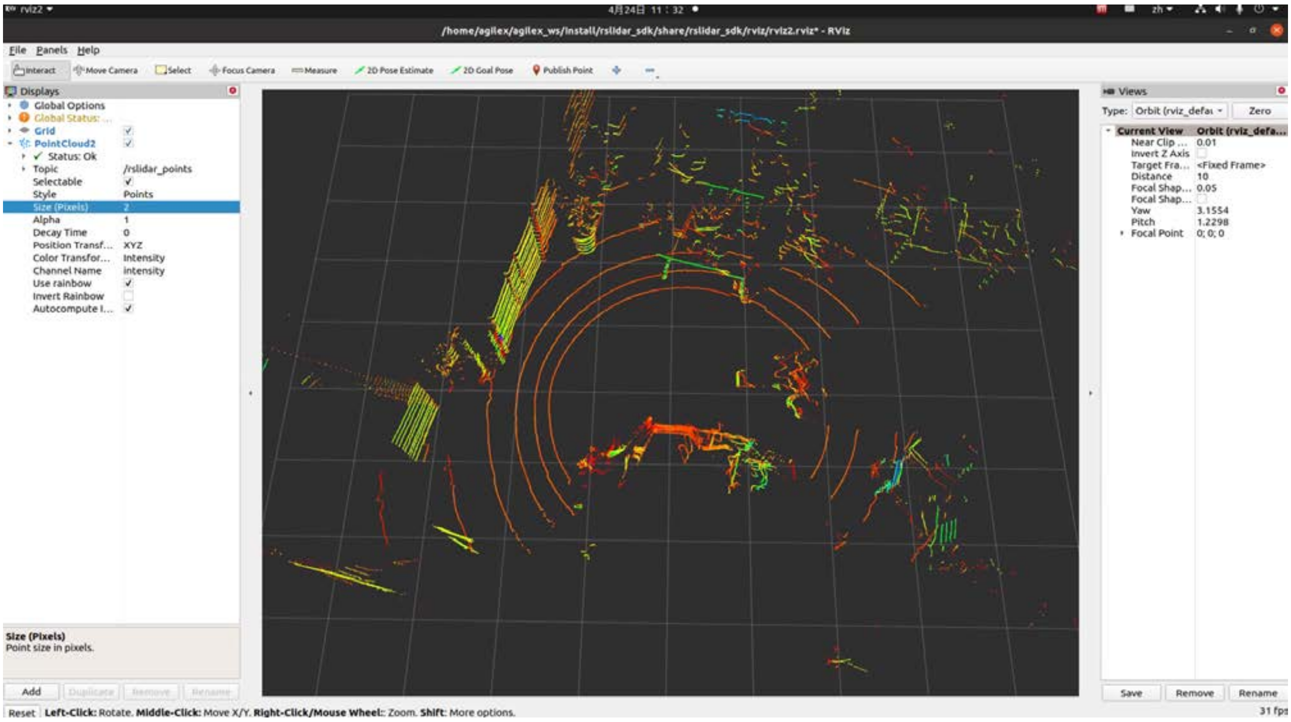
lanunch files	Function
rtab_slam.launch.py	Rtabmap visual mapping.
rtab_navigation.launch.py	Rtabmap positioning and start the navigation.
cartographer.launch.py	Cartographer mapping
open_rslidar.launch.py	16-lines LIDAR start up script and covert the point clou data to LIDAR data，then publish the tf of base_link->rslidar.
open_ydlidar.launch.py	EAI single line LIDAR start up script and publish the tf base_link->laser_frame.
localization_launch.py	Start amcl positioning and map_server.
navigation_launch.py	Start the navigation core plugin.
navigation2.launch.py	amcl positioning method and start the navigation.
slam_toolbox.launch.py	Use slam_toolbox for mapping.

Start LIDAR:

Open the terminal and input the below command:

For Pro version:

```
ros2 launch rslidar_sdk start.py
```



Note: The LIDAR is connected to LAN2 by default and the IP is set to 192.168.1.102.

For Lite version:

```
ros2 launch ydlidar ydlidar_launch.py
```

Start Scout MINI chassis. Open the terminal and input command:

```
ros2 launch scout_base scout_mini_base.launch.py
```

Input password: agx

```
agilex@agilex:~$ ros2 launch scout_base scout_mini_base.launch.py
[INFO] [launch]: All log files can be found below /home/agilex/.ros/log/2022-04-24-13-53-10-642127-agilex-2780
[INFO] [launch]: Default logging verbosity is set to INFO
[sudo] agilex 的密码:
[INFO] [scout_base_node-1]: process started with pid [2786]
[scout_base_node-1] Loading parameters:
[scout_base_node-1] - port name: can0
[scout_base_node-1] - odom frame name: odom
[scout_base_node-1] - base frame name: base_link
[scout_base_node-1] - odom topic name: odom
[scout_base_node-1] - is scout mini: true
[scout_base_node-1] - is omni wheel: false
[scout_base_node-1] - simulated robot: false
[scout_base_node-1] - control rate: 10
[scout_base_node-1] ----------------------------
[scout_base_node-1] Robot base: Scout Mini
[scout_base_node-1] Start listening to port: can0
[scout_base_node-1] Detected protocol: AGX_V2
[scout_base_node-1] Creating interface for Scout with AGX_V2 Protocol
[scout_base_node-1] Robot initialized, start running ...
[scout_base_node-1] Start listening to port: can0
[scout_base_node-1] Using CAN bus to talk with the robot
```

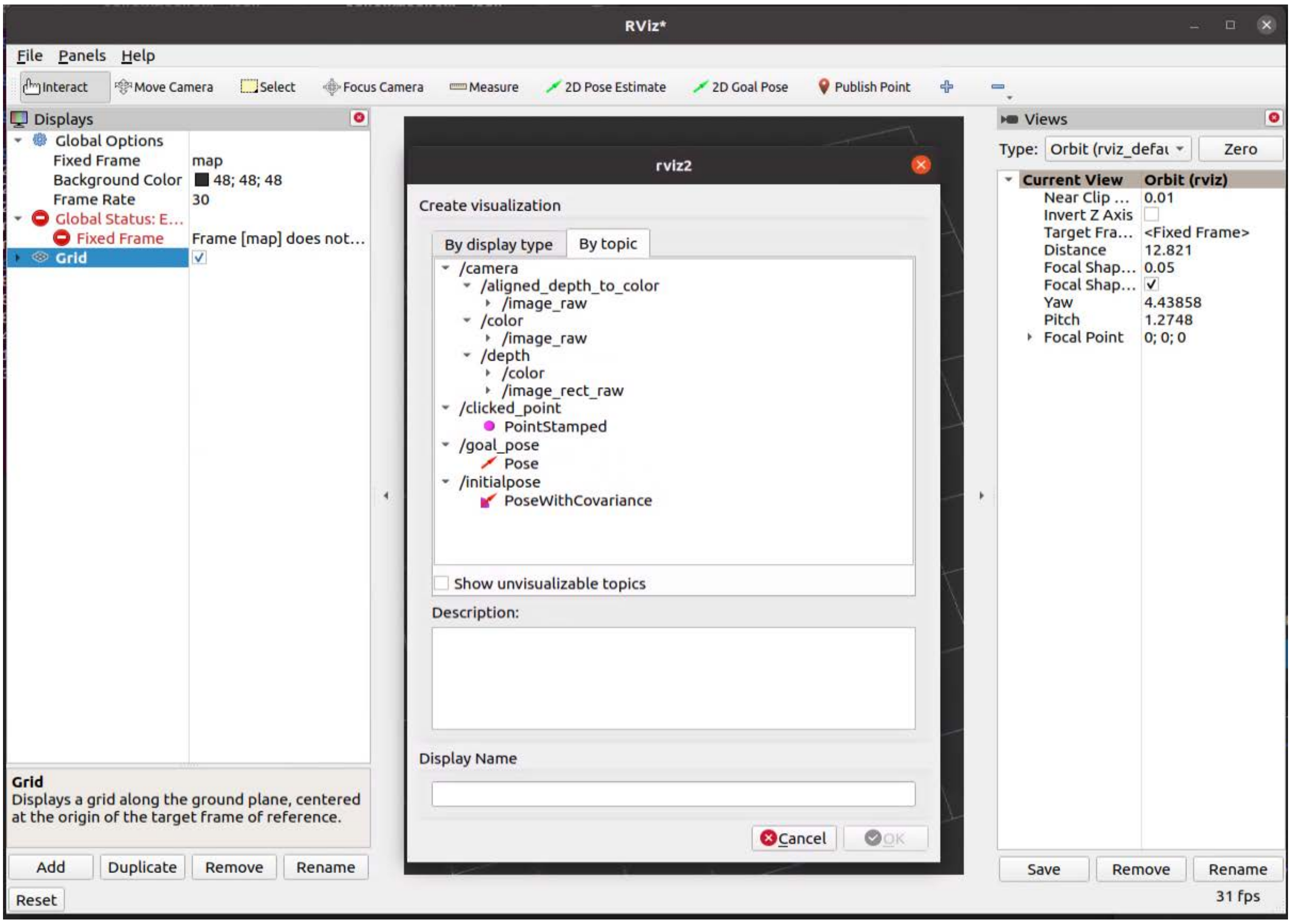
Then the chassis communicates with the ROS successfully. We can use the keyboard to control the chassis by using the following command:

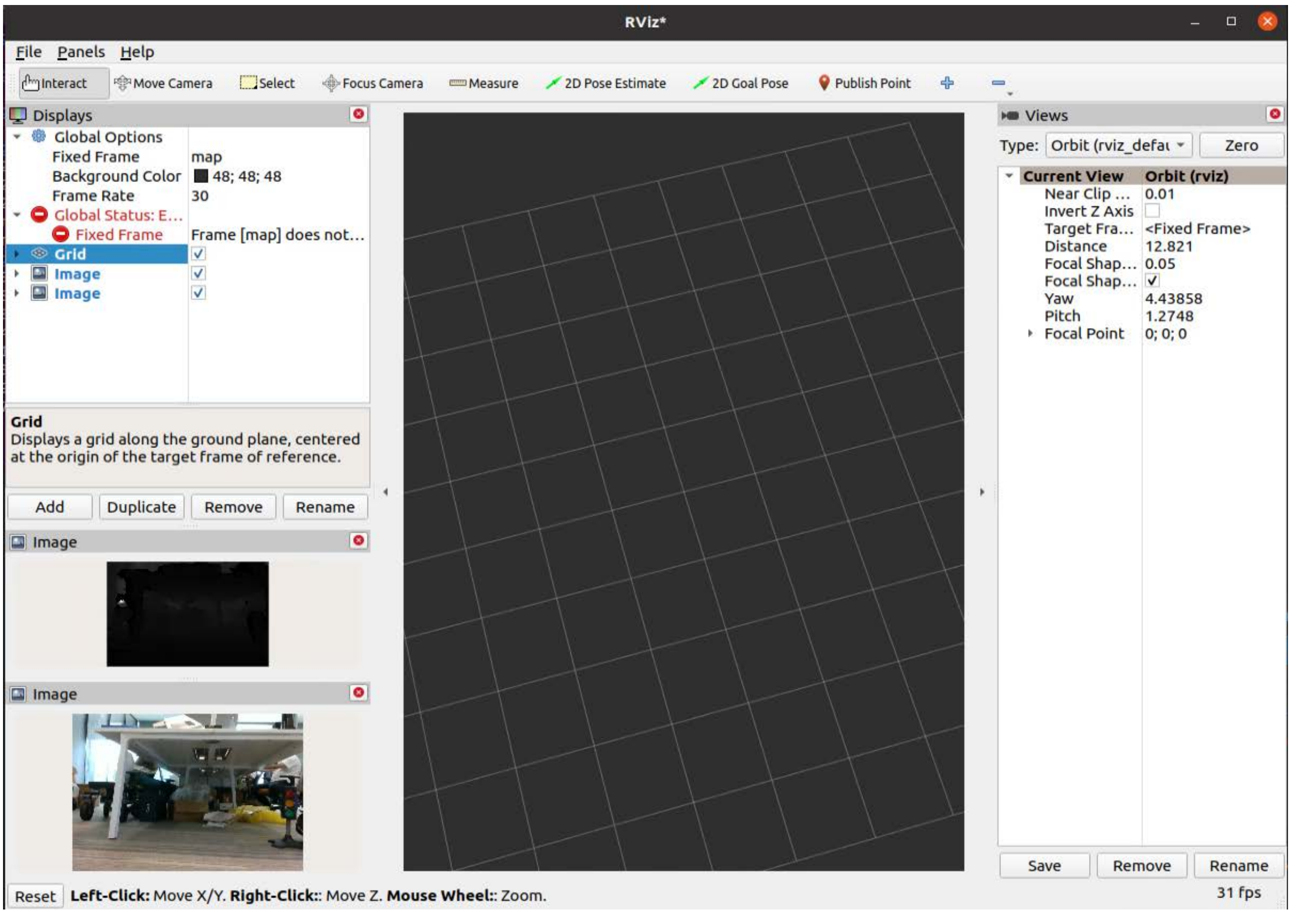
```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

Start the camera:

```
ros2 launch realsense2_camera rs_launch.py
```

Then start RVIZ2 and click Add icon to add the corresponding topic to view the depth and image information of the camera:


RViz*
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
Publish Point
Displays
Global Options
Fixed Frame
map
Background Color
48; 48; 48
Frame Rate
30
Global Status: E...
Fixed Frame
Frame [map] does not...
Grid
Grid
Displays a grid along the ground plane, centered at the origin of the target frame of reference.
Add
Duplicate
Remove
Rename
Reset
rviz2
Create visualization
By display type
By topic
/camera
/aligned_depth_to_color
/image_raw
/color
/image_raw
/depth
/color
/image_rect_raw
/clicked_point
PointStamped
/goal_pose
Pose
/initialpose
PoseWithCovariance
Show unvisualizable topics
Description:
Display Name
Cancel
OK
Views
Type:
Orbit (rviz_defau
Zero
Current View
Orbit (rviz)
Near Clip ...
0.01
Invert Z Axis
Target Fra...
<Fixed Frame>
Distance
12.821
Focal Shap...
0.05
Focal Shap...
Yaw
4.43858
Pitch
1.2748
Focal Point
0; 0; 0
Save
Remove
Rename
31 fps


RViz*
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
Publish Point
Displays
Global Options
Fixed Frame
map
Background Color
48; 48; 48
Frame Rate
30
Global Status: E...
Fixed Frame
Frame [map] does not...
Grid
Image
Image
Grid
Displays a grid along the ground plane, centered at the origin of the target frame of reference.
Add
Duplicate
Remove
Rename
Image
Image
Views
Type:
Orbit (rviz_defau
Zero
Current View
Orbit (rviz)
Near Clip ...
0.01
Invert Z Axis
Target Fra...
<Fixed Frame>
Distance
12.821
Focal Shap...
0.05
Focal Shap...
Yaw
4.43858
Pitch
1.2748
Focal Point
0; 0; 0
Save
Remove
Rename
Reset
Left-Click: Move X/Y. Right-Click:: Move Z. Mouse Wheel:: Zoom.
31 fps

4 Mapping and navigation demonstration

4.1 Mapping

The whole mapping process is mainly divided into three parts: first, start the sensor used in mapping, start the mapping algorithm, remote control the vehicle to complete the mapping, and finally save the constructed map.

4.1.1 Slam-toolbox mapping

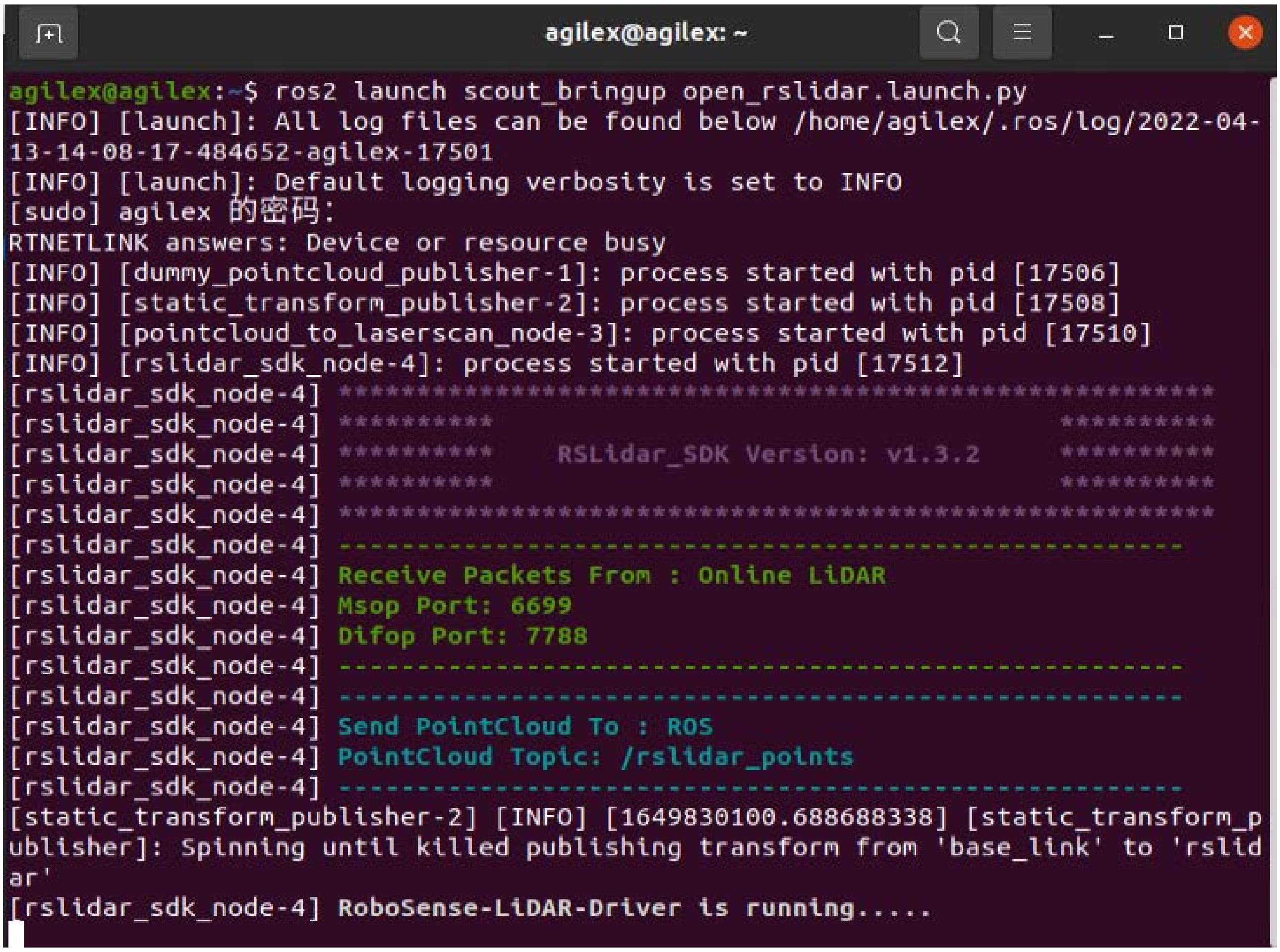
Start the LIDAR node and open the terminal to input:

For Pro version:

```
ros2 launch scout_bringup open_rslidar.launch.py
```

For Lite version:

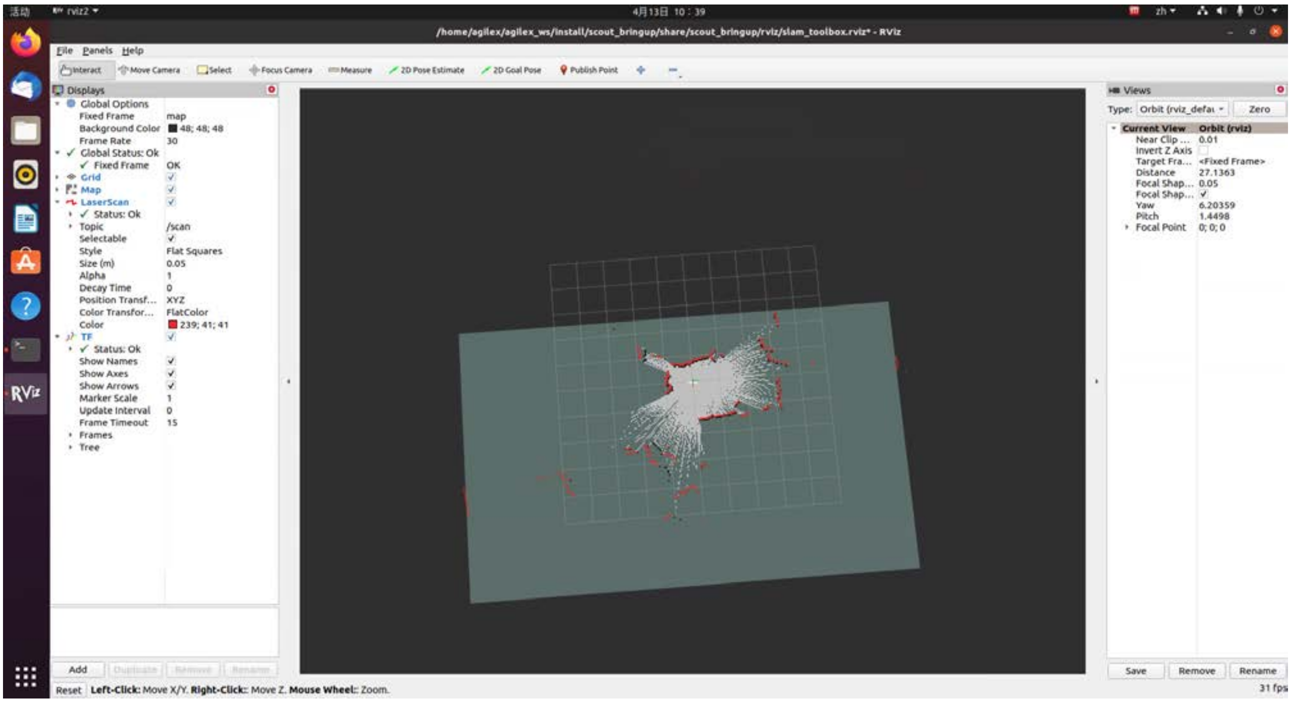
```
ros2 launch scout_bringup open_ydlidar.launch.py
```



Open the other Terminal and start the slam-toolbox:

```
ros2 launch scout_bringup slam_toolbox.launch.py
```

After the startup, RVIZ2 will display the map building interface, and the whole map can be scanned by remote control the vehicle:



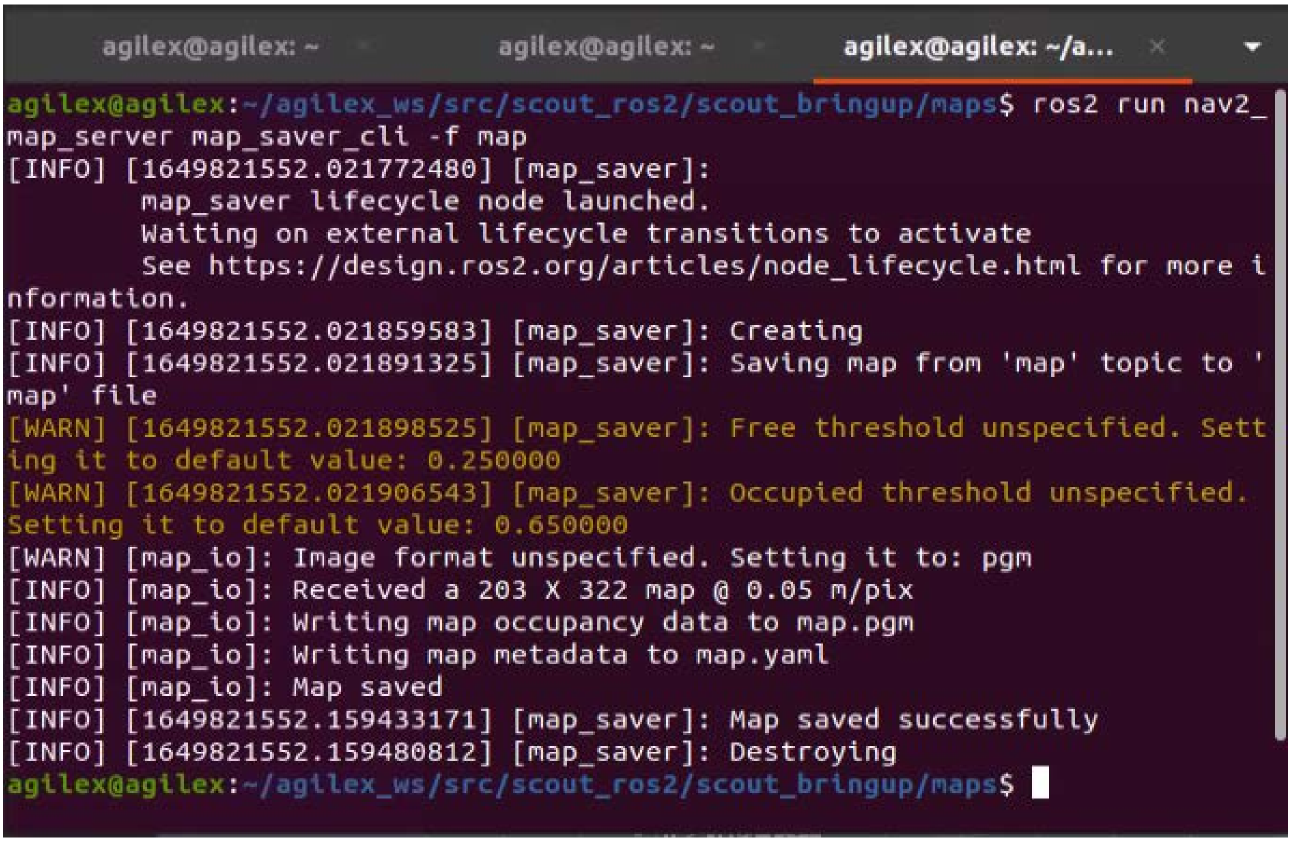
After complete the whole mapping, save the map and enter the directory:

```
cd /home/agilex/agilex_ws/src/scout_ros2/scout_bringup/maps
```

Run the save map command:

```
ros2 run nav2_map_server map_saver_cli -f map
```

When the Terminal pops up the message: Map saved successfully! , the map is saved successfully. As is shown in the following:



```
agilex@agilex: ~        agilex@agilex: ~        agilex@agilex: ~/a...
agilex@agilex:~/agilex_ws/src/scout_ros2/scout_bringup/maps$ ros2 run nav2_
map_server map_saver_cli -f map
[INFO] [1649821552.021772480] [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 i
nformation.
[INFO] [1649821552.021859583] [map_saver]: Creating
[INFO] [1649821552.021891325] [map_saver]: Saving map from 'map' topic to '
map' file
[WARN] [1649821552.021898525] [map_saver]: Free threshold unspecified. Sett
ing it to default value: 0.250000
[WARN] [1649821552.021906543] [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 203 X 322 map @ 0.05 m/pix
[INFO] [map_io]: Writing map occupancy data to map.pgm
[INFO] [map_io]: Writing map metadata to map.yaml
[INFO] [map_io]: Map saved
[INFO] [1649821552.159433171] [map_saver]: Map saved successfully
[INFO] [1649821552.159480812] [map_saver]: Destroying
agilex@agilex:~/agilex_ws/src/scout_ros2/scout_bringup/maps$
```

4.1.2 Cartographer mapping

Start the LIDAR node and open the Terminal to input the following command:

For Pro version:

```
ros2 launch scout_bringup open_rslidar.launch.py
```

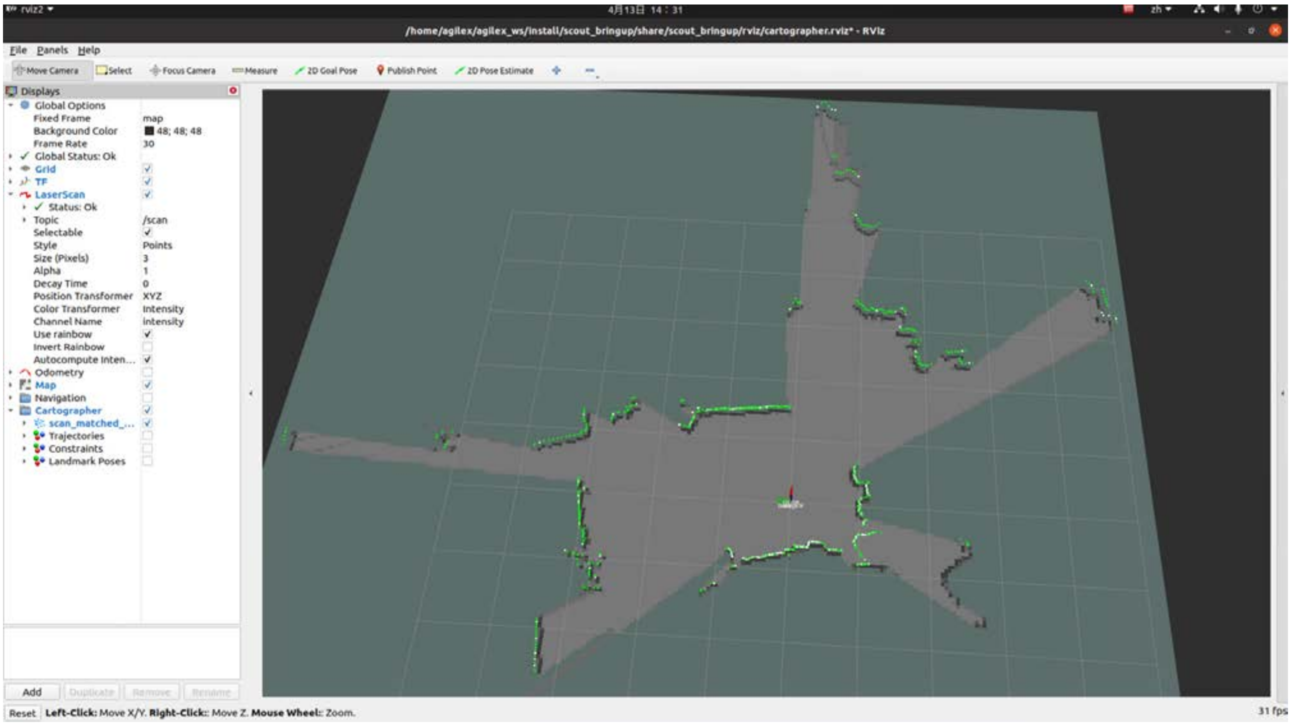
For Lite version:

```
ros2 launch scout_bringup open_ydlidar.launch.py
```

Open a new Terminal to start the Slam-toolbox:

```
ros2 launch scout_bringup cartographer.launch.py
```

After the startup, RVIZ2 will display the map building interface, and the whole map can be scanned by remote control the vehicle:

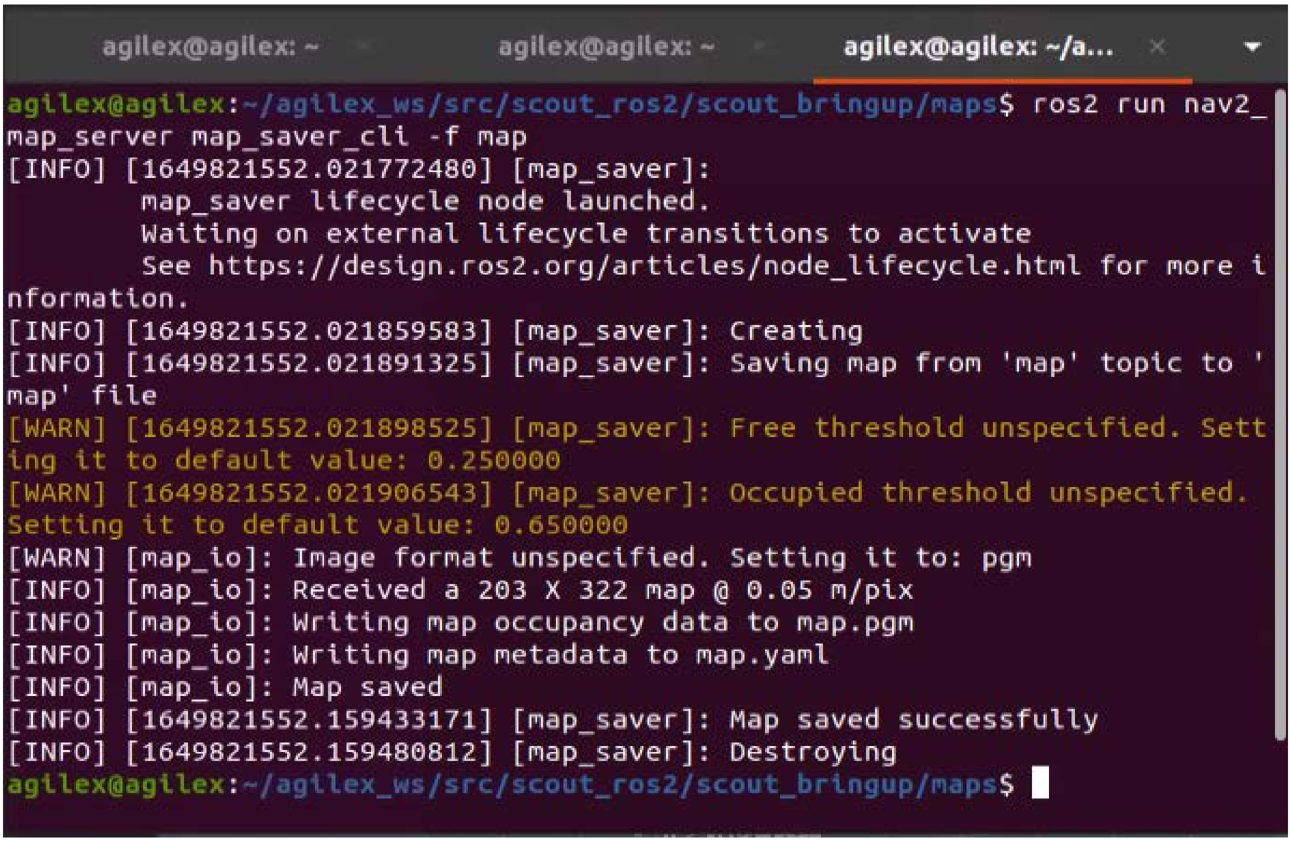


After complete the whole mapping, save the map and enter the directory:

```
cd /home/agilex/agilex_ws/src/scout_ros2/scout_bringup/maps
```

Run the save map command:

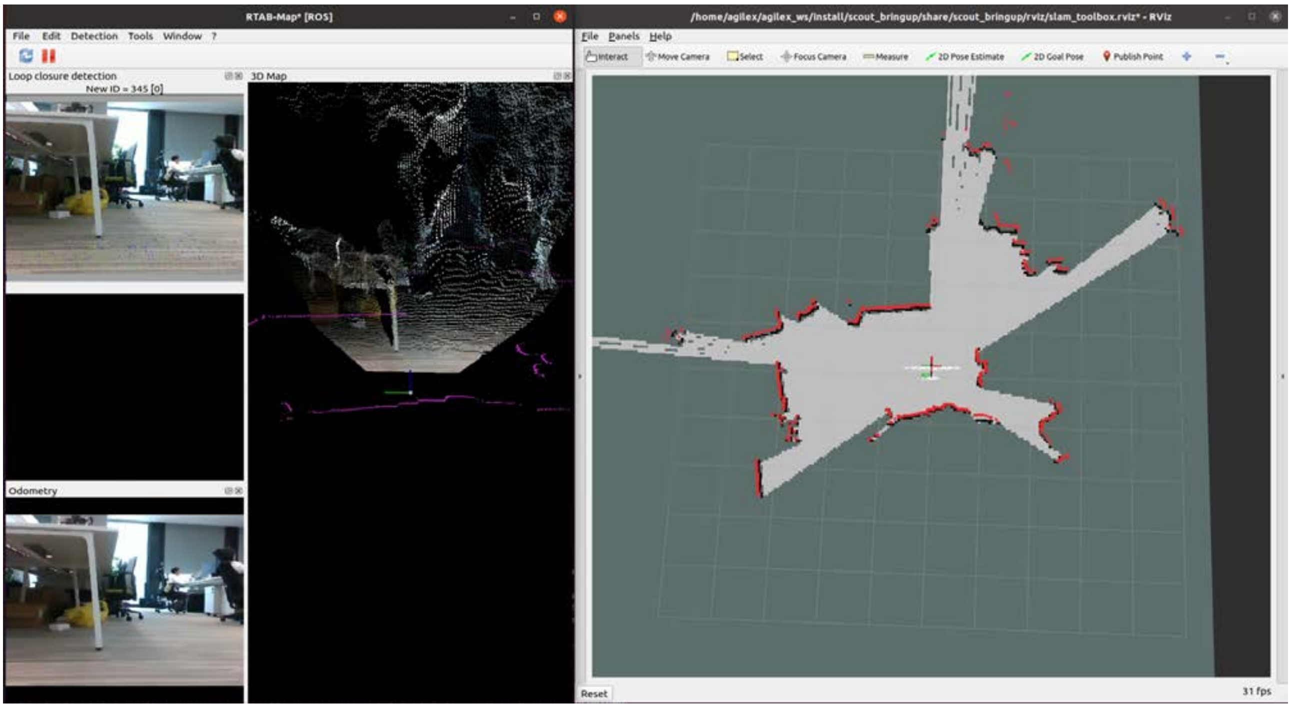
```
ros2 run nav2_map_server map_saver_cli -f map
```



4.1.3 Rtabmap mapping

Start Rtabmap mapping, open the Terminal and input:

```
ros2 launch scout_bringup rtab_slam.launch.py
```



Rtabmap automatically saves the rtabMap file to the ~ /. ros/ directory upon exit .

4.2 Navigation

Navigation operation process is mainly divided into: start sensor, calibrate initial pose, give target point and open navigation.
Open the Terminal and start the Lidar,
For Pro version:

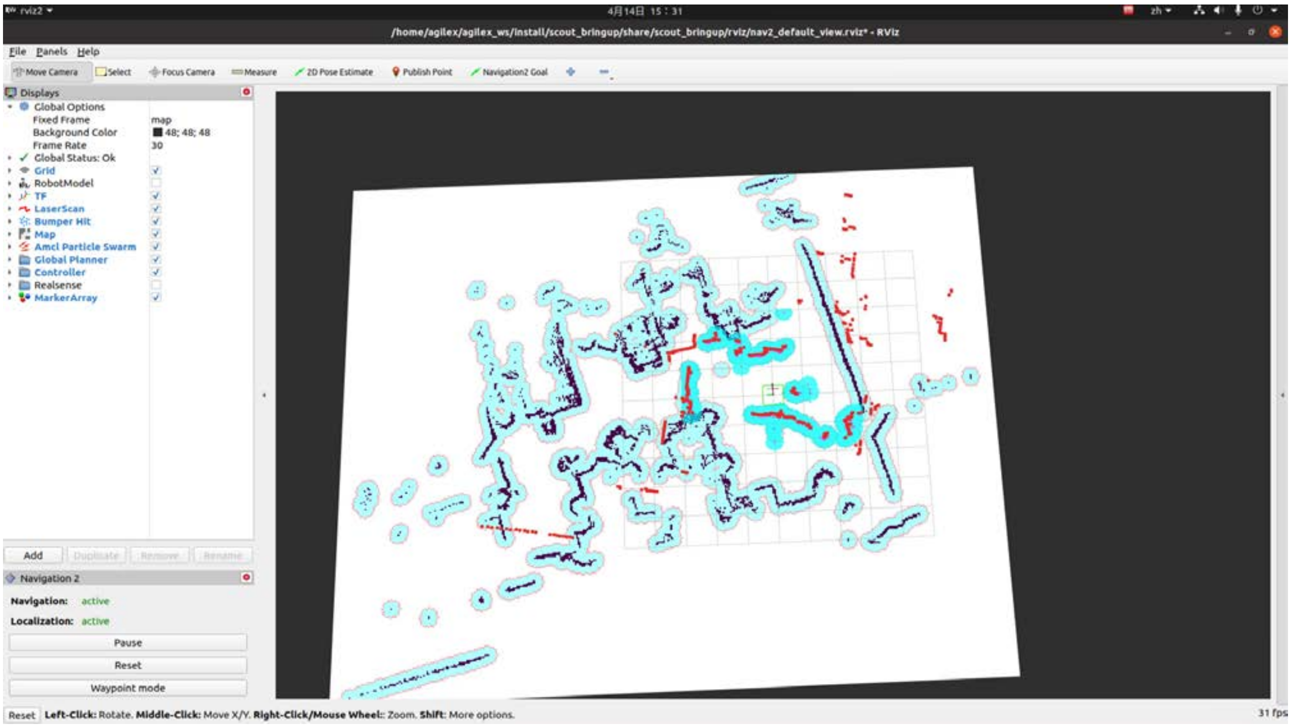
```
ros2 launch scout_bringup open_rslidar.launch.py
```

For Lite version:

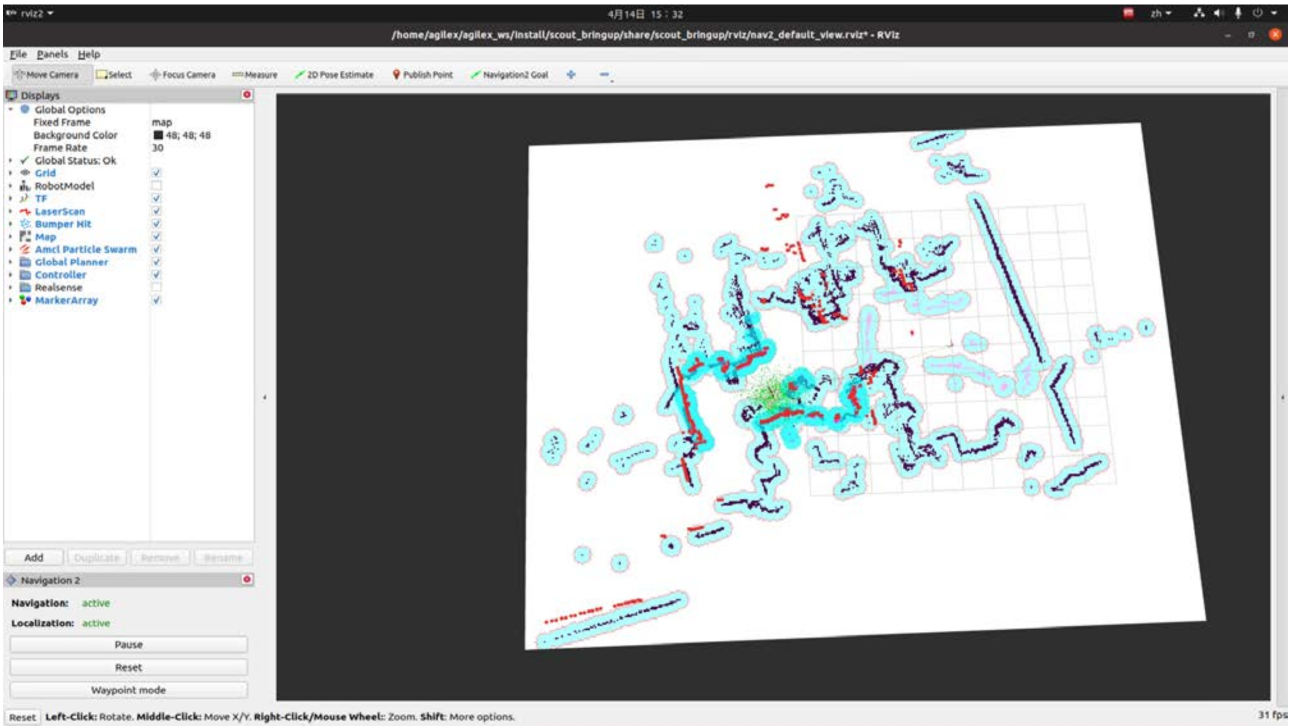
```
ros2 launch scout_bringup open_ydlidar.launch.py
```

Open a new Terminal, and start the navigation stack:

```
ros2 launch scout_bringup navigation2.launch.py
```

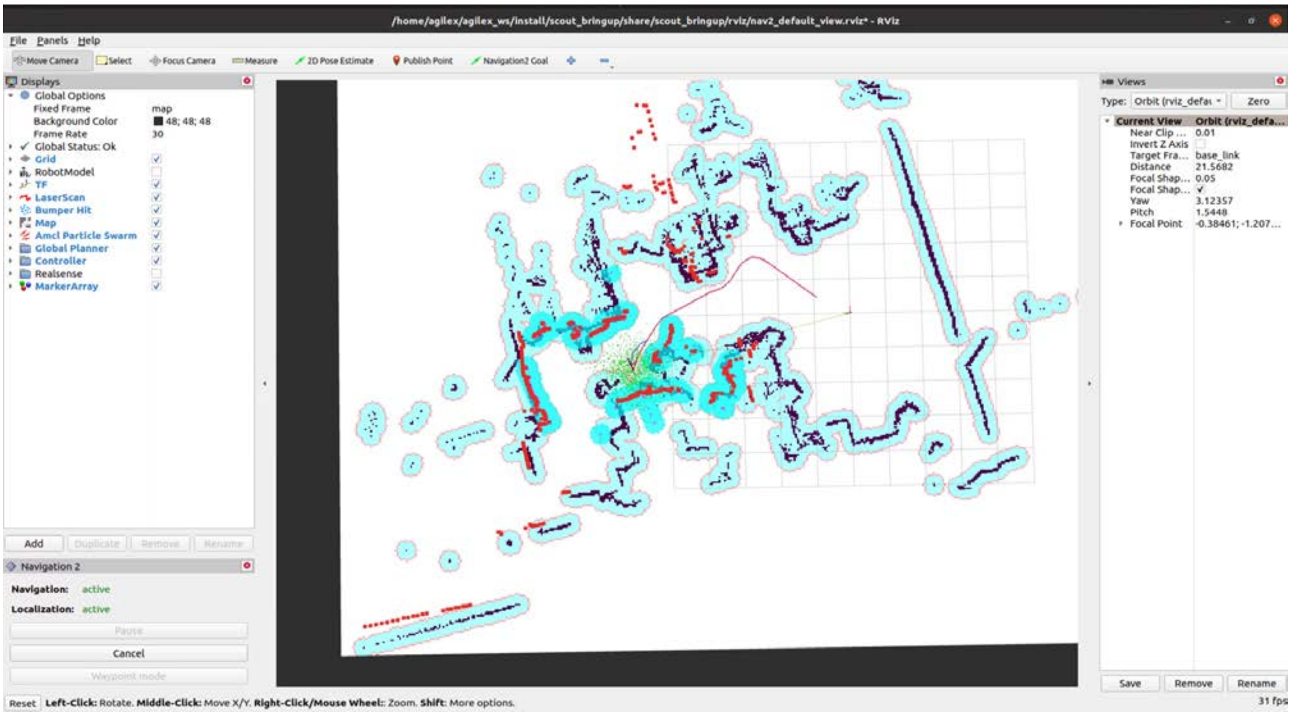


Click and select 2D Pose Estimate to set the original pose of robot:



Single point navigation mode:

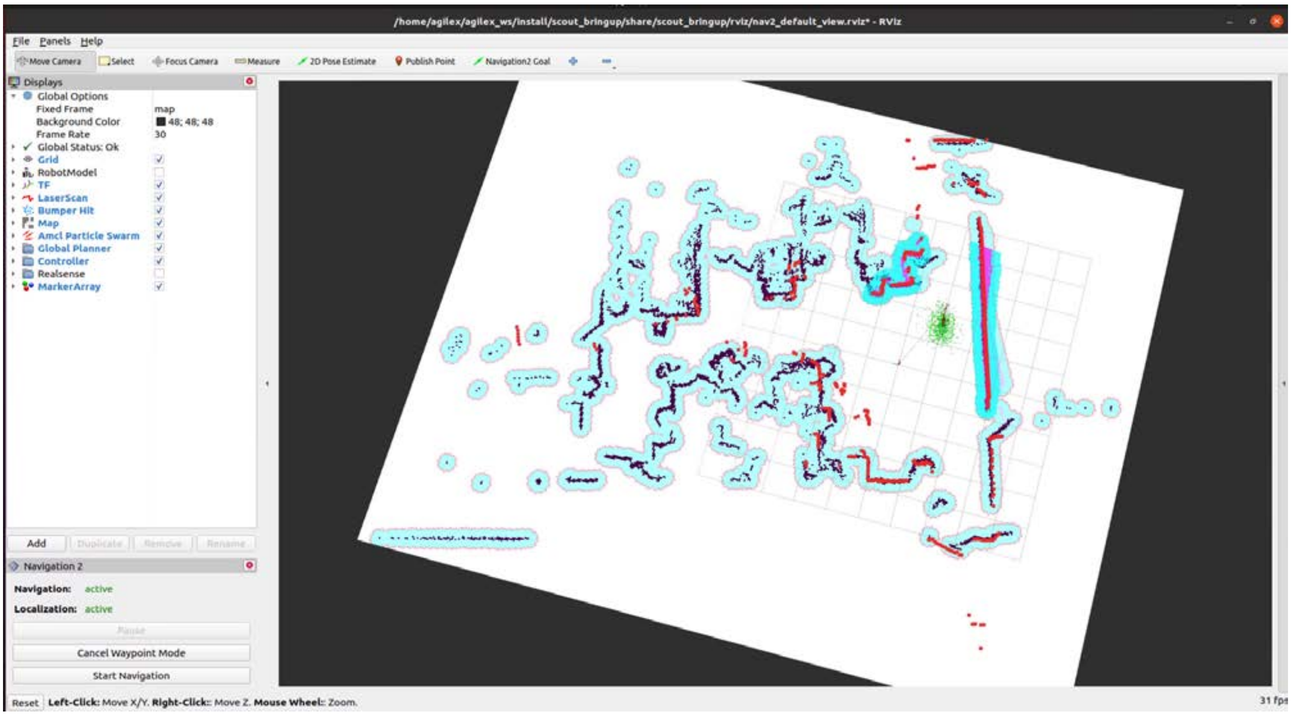
Click Navigation2 Goal to set the navigation destination point. Then the planning path point will be generated on the map.



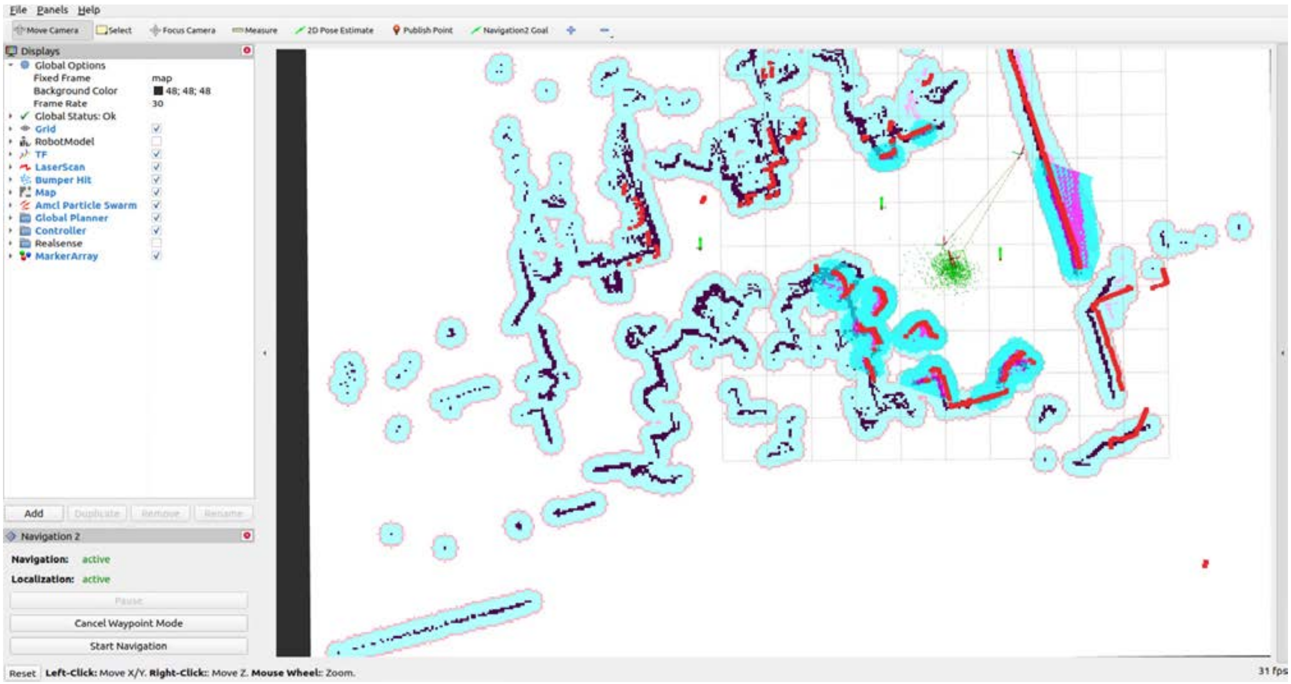
Finally, switch the remote controller to COMMAND mode. The robot will move automatically by following the planning path.

Waypoint mode:

Click Waypoint mode to switch to this mode.



Click Navigation2 Goal to set multiple the navigation destination points:



After setting the target point, click Start Navigation and switch the robot to command control mode, and the robot will move to the target point in sequence.

Note: You may need to enter a password during navigation or mapping. The password is: **agx**

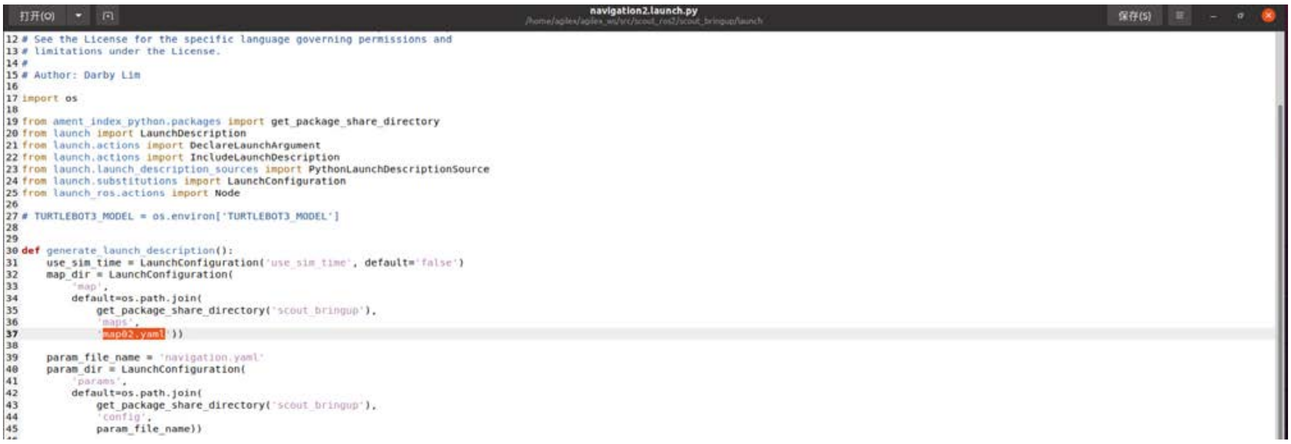
When saving maps with cartographer and slam_Toolbox, please confirm the directory to save the maps: /home/agilex/agilex_ws/ SRC /scout_ros2/scout_bringup/maps.

If you want to save the map as the navigation one,please:

```
cd /home/agilex/agilex_ws/src/scout_ros2/scout_bringup/launch
sudo gedit navigation2.launch.py
```

Replace map02.yaml on line 37 with the name of the saved map. Go to the workspace directory and compile and source to load the saved map correctly in the navigation

If the navigation startup fails, close the terminal and restart the corresponding node.



```
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14 #
15 # Author: Darby Lim
16
17 import os
18
19 from ament_index_python.packages import get_package_share_directory
20 from launch import LaunchDescription
21 from launch.actions import DeclareLaunchArgument
22 from launch.actions import IncludeLaunchDescription
23 from launch.launch_description_sources import PythonLaunchDescriptionSource
24 from launch.substitutions import LaunchConfiguration
25 from launch_ros.actions import Node
26
27 # TURTLEBOT3_MODEL = os.environ['TURTLEBOT3_MODEL']
28
29
30 def generate_launch_description():
31     use_sim_time = LaunchConfiguration('use_sim_time', default='false')
32     map_dir = LaunchConfiguration(
33         'map',
34         default=os.path.join(
35             get_package_share_directory('scout_bringup'),
36             'maps',
37             'map02.yaml'))
38
39     param_file_name = 'navigation.yaml'
40     param_dir = LaunchConfiguration(
41         'params',
42         default=os.path.join(
43             get_package_share_directory('scout_bringup'),
44             'config',
45             param_file_name))
```

5 ROS2

5.1 ROS1&ROS2 comparison

5.1.1 OS

The ROS2 supporting system can be Windows,Mac,Embedded RTOS platform or even computer without operation system while ROS only supports Linux.

5.1.2 Middleware

Decentration: ROS differs from ROS2 middleware in that ROS2 cancels the master node. After decentralization, all nodes can discover each other through DDS nodes, all nodes are equal, and can communicate with each other 1-to-1, 1-to-N, and N-to-N. Introduction of DDS communication: the real-time, reliability and continuity of ROS2 are enhanced.

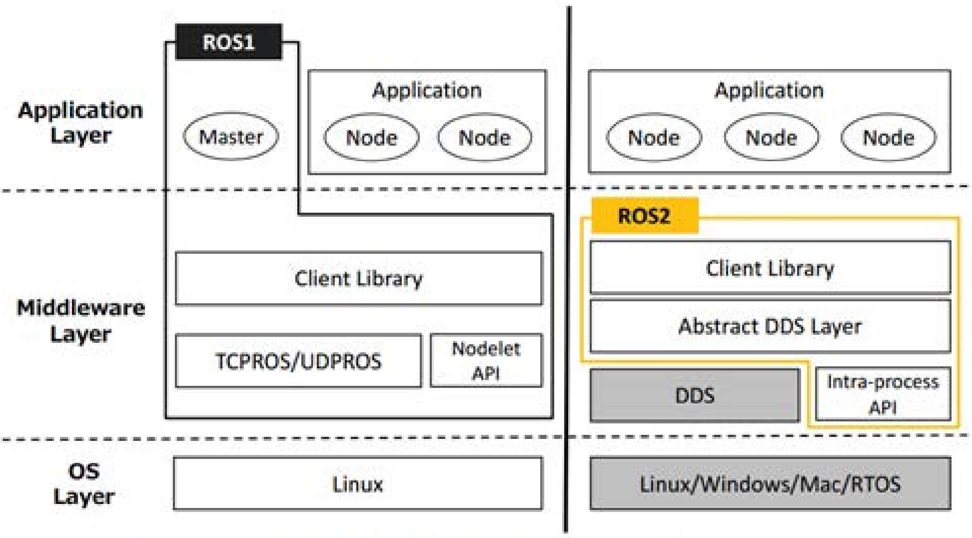


Figure 1: ROS1/ROS2 architecture.

5.1.3 Nodelet与Intra-process

In ROS1 architecture, Nodelet and TCPROS/UDPROS are parallel layers that can provide a more optimized way to transfer data for multiple nodes in the same process. ROS2 retains a similar data transfer method, called "intra-process," which is also independent of DDS.

5.1.4 Application layer

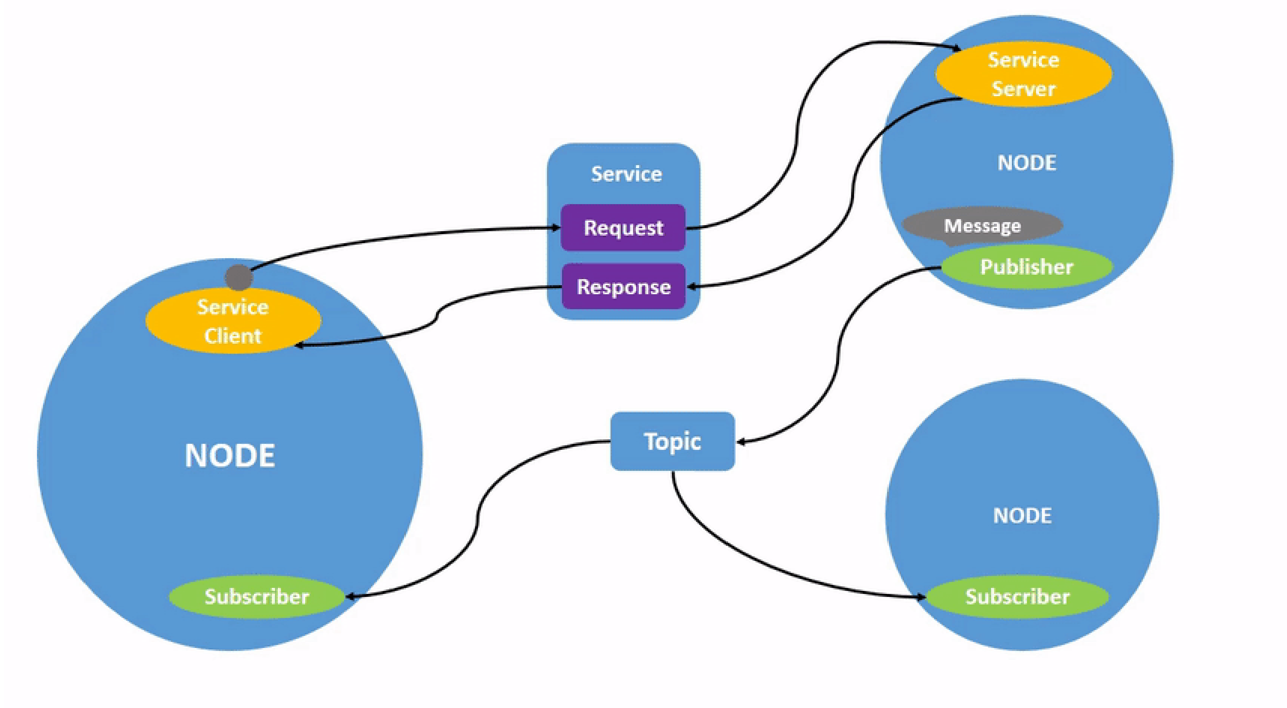
ROS1 relies heavily on ROS Masters, and you can imagine that if the Master goes down, the entire system will crash and fail. But in ROS2 architecture, Master is removed and nodes use a discovery mechanism called "Discovery" to help establish connections with each other.

5.1.5 Other features

1) Launch files written in Python are available
2) Multi-robot collaborative communication support
3) Support secure encrypted communication
4) The same process supports multiple nodes
5) Use ament for package management
6) Support Qos quality of service
7) Support node life cycle management
8) Efficient interprocess communication

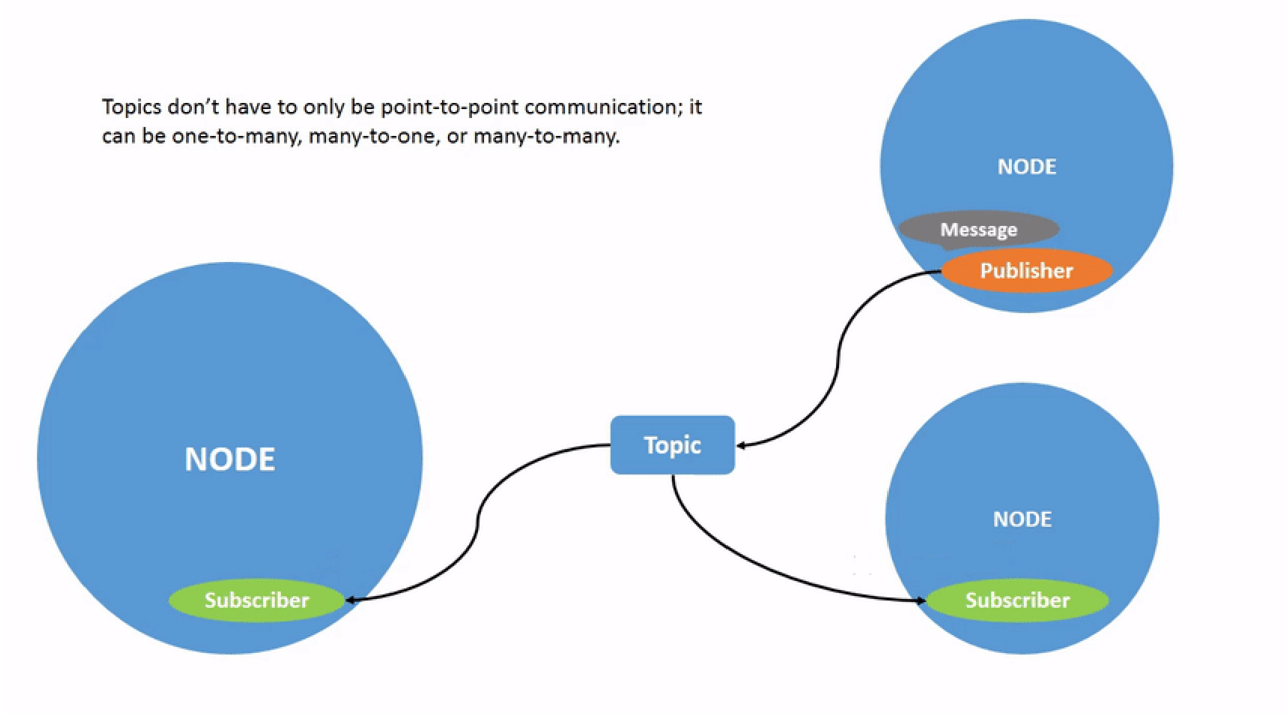
5.2 ROS2 basic concepts

ROS2 decomposes complex systems into many modular nodes. Each node in ROS should be responsible for a single modular purpose (for example, one node for controlling the chassis, one node for controlling the lidar, etc.). Each node can send and receive data to and from other nodes by topic, service, action, or parameter.



5.2 .2 Topic

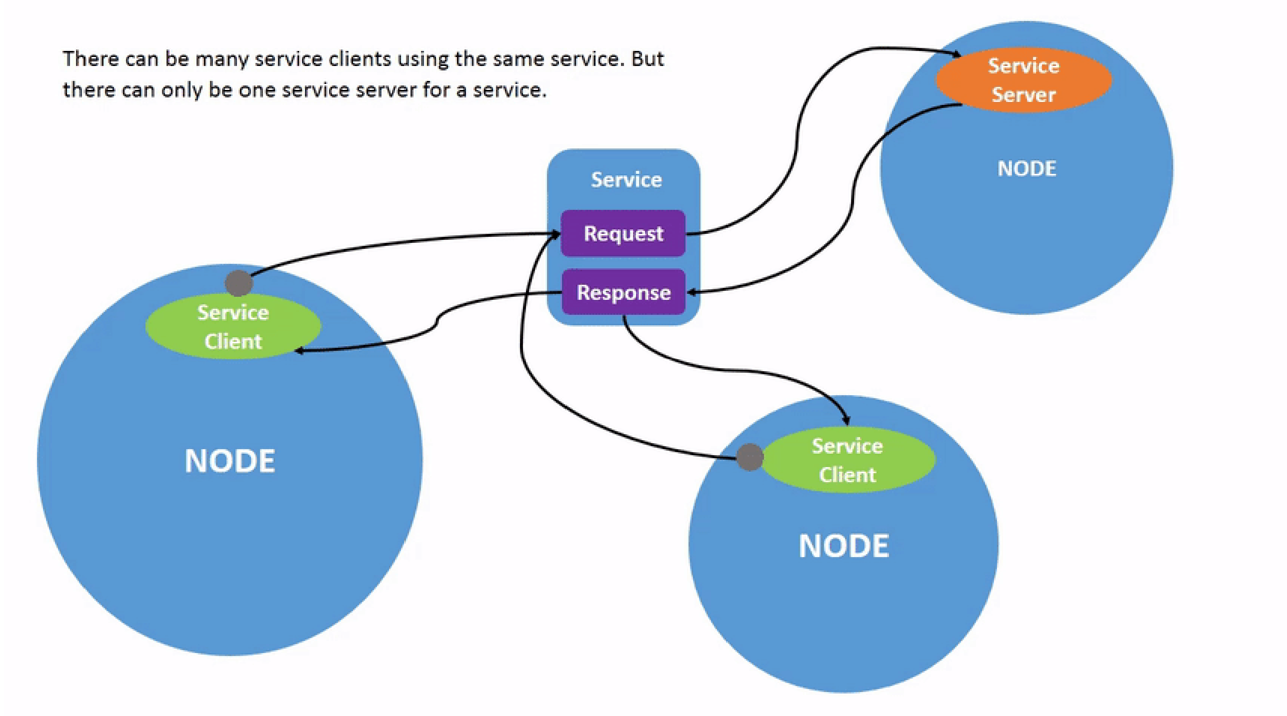
Topic is an important element of ROS2 and acts as a bus for nodes to exchange messages. A node can publish data to any number of topics and subscribe to any number of topics simultaneously. Topic is one of the main ways that data is moved between nodes and between different parts of the system.



Relevant commands of Topic	Function
ros2 topic list	Return the list of all topics currently active in the system.
ros2 topic echo <topic_name>	Check data of topic.
ros2 topic info <topic_name> -v	Check the publisher and subscriber o other information of topic .
ros2 topic pub <topic_name> <msg_type> '<args>'	Publish the specified topic message.
ros2 topic hz <topic_name>	Check the frequency of topic publishing.

5.2.3 Service

Service is the other way for nodes communication in the below ROS diagram. Services are based on the call-response model, not the publisher-subscriber model of a topic. Although topics allow nodes to subscribe to data streams and get continuous updates, services provide data only when clients specifically invoke them.



Relevant commands of Service	Function
ros2 service list	Return the list of all topics currently active in the system.
ros2 service type <service_name>	Check the service type.
ros2 service find <service_name>	Check all the services of specified type.
ros2 service call <service_name> <service_type> <arguments>	Check specified service type and its parameter structure.

5.2.4 Parameter

The parameter is the configuration value of node. The parameters can be treated as node settings. Nodes can store parameters as integers, floating point numbers, bools, strings, and lists. In ROS2, each node maintains its own parameters.

Parameters in ROS2 are associated with each node. Parameters are used to configure the node at startup (and run time) without changing the code. The life cycle of the parameter is related to the life cycle of the node (although the node can implement some kind of persistence to reload the value after a restart).

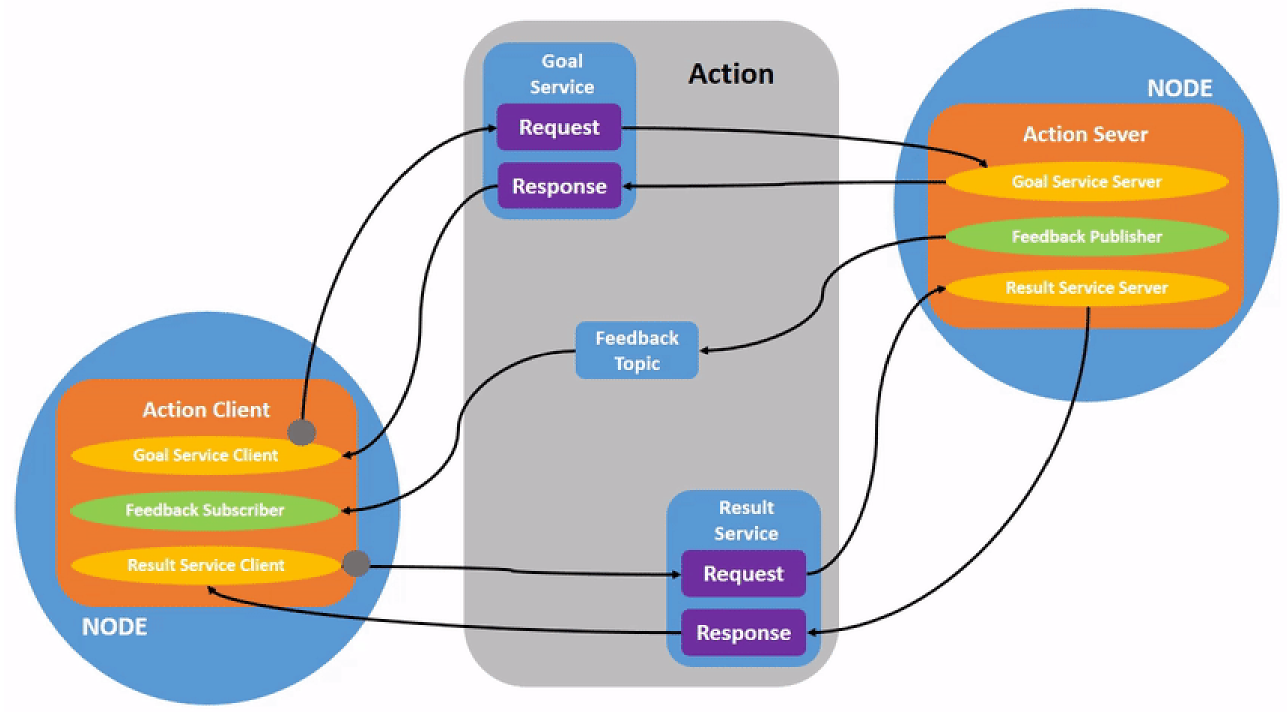
Parameters are addressed by the node name, node namespace, parameter name, and parameter namespace. Providing a parameter namespace is optional.

Each parameter consists of a key, a value, and a descriptor. The key is a string and the value is one of the following types: bool, INT64, FLOAT64, String, Byte [], bool[], INT64 [], FLOAT64

[], or String []. By default, all descriptors are empty, but can contain parameter descriptions, value ranges, type information, and other constraints.

5.2.4 Action

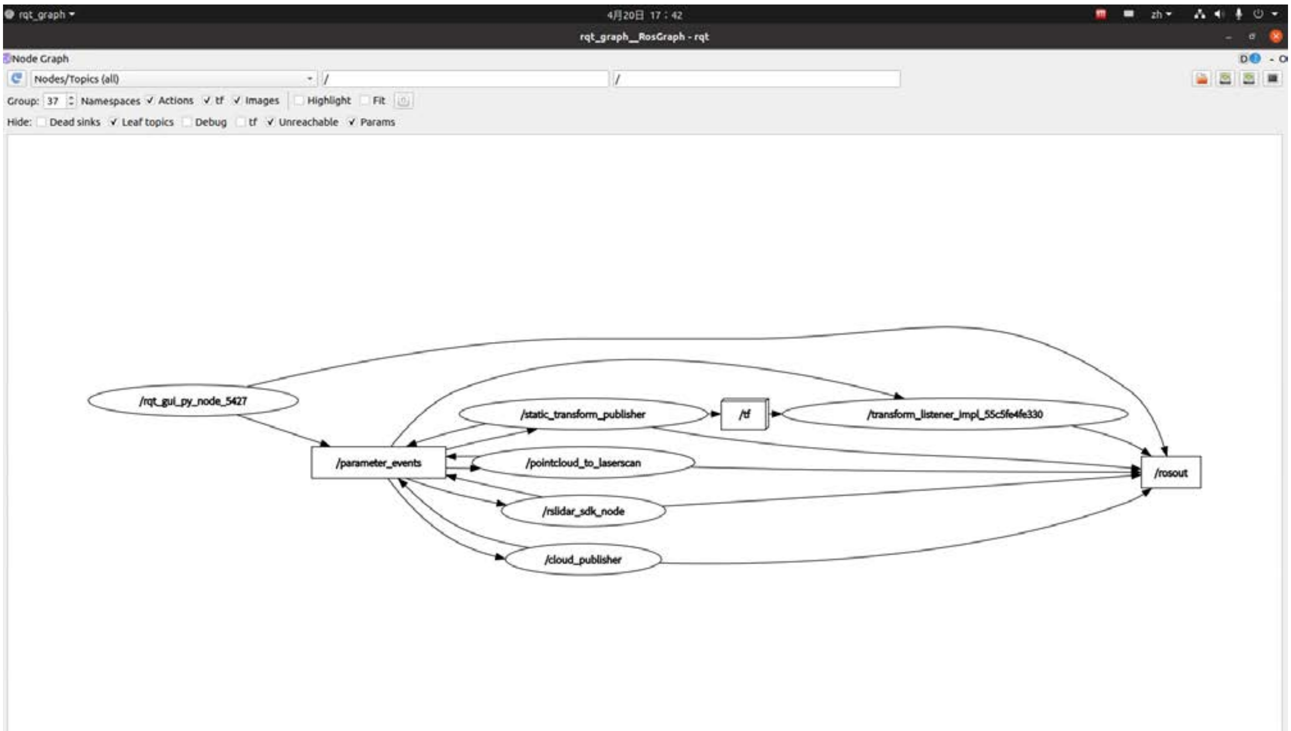
Actions is a type of communication in ROS2 that is suitable for long-running tasks. They consist of three parts: goals, feedback, and results. Action operation is built on themes and services. They function like services, except that operations are preemptable (Users can cancel them on execution). They also provide stable feedback rather than a single response service. Actions uses a client-server model, similar to the publisher-subscriber model (described in the topic tutorial). The Action Client node sends the target to the Action Server node, which validates the target and returns the feedback flow and results.



5.2.6 Tools

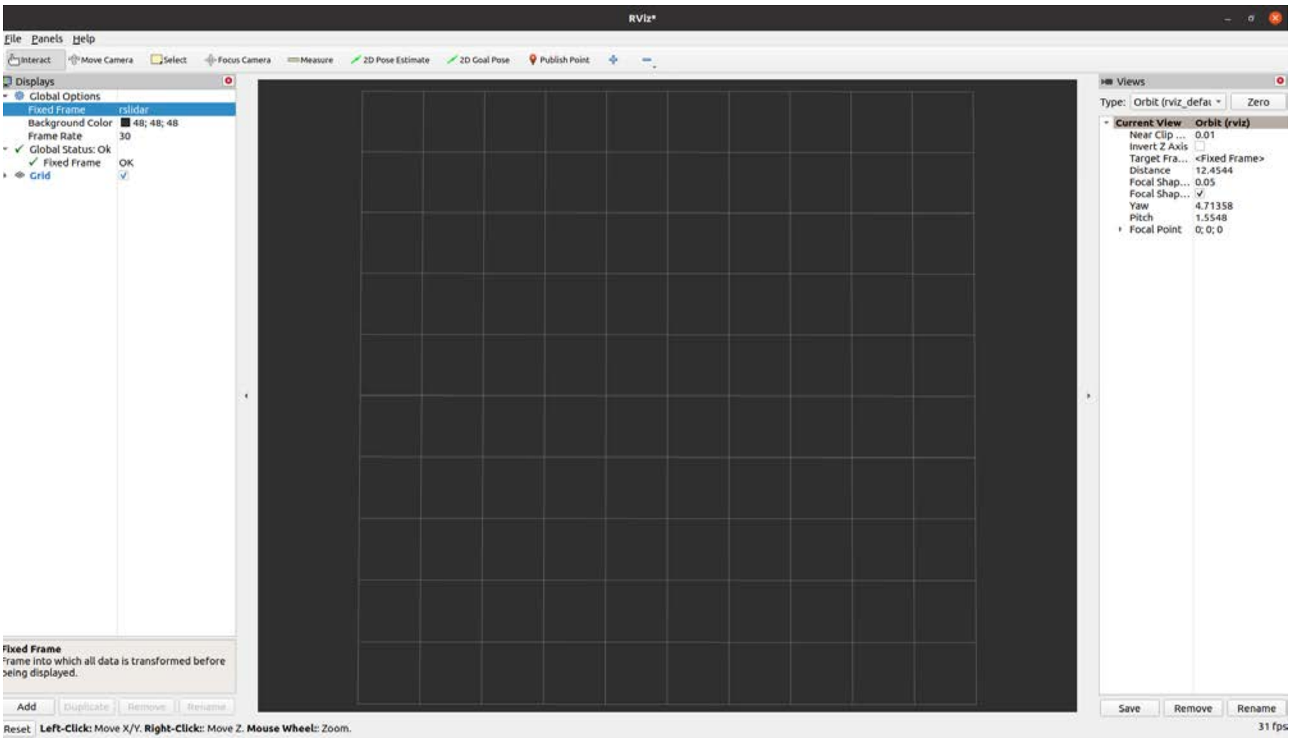
rqt_graph:

While the system is still running, we can open a new terminal and run rqt_graph to get better understanding of the relationship between the started nodes.



Rviz2:

Rviz2 is a graphical chemical tool of ROS2, which can be very convenient for users to develop and debug ROS2 through graphical interface. The operation interface is also very simple, which is mainly divided into the upper menu area, the left display content setting area, the middle display area, the right display angle setting area, and the lower area for ROS2 status.



5.3 Build ROS packages

Create workspace:

```
mkdir -p ros2_ws/src
cd ros2_ws/
colcon build --symlink-install
```

Create our own ROS packages:

```
cd ~/ros2_ws/src/
ros2 pkg create --build-type ament_cmake agx
```

```
agilex@agilex:~/ros2_ws/src$ ros2 pkg create --build-type ament_cmake agx
going to create a new package
package name: agx
destination directory: /home/agilex/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['agilex <agilex@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: []
creating folder ./agx
creating ./agx/package.xml
creating source and include folder
creating folder ./agx/src
creating folder ./agx/include/agx
creating ./agx/CMakeLists.txt
```

Enter agx folder will see the automatically created CMakelist.txt, package.xml, src, include, etc INCLUDE folder contains header files, SRC contains source files, and package.xml describes the package name, version number, author, maintainer, and dependency on ROS packages. TheCMakelist.txt file is the inputof CMake build system and is used to build packages.

```
agilex@agilex:~/ros2_ws/src$ cd agx/
agilex@agilex:~/ros2_ws/src/agx$ tree
.
├── CMakeLists.txt
├── include
│   └── agx
├── package.xml
└── src

3 directories, 2 files
```

Create a publisher:

```
agilex@agilex:~$ cd ~/ros2_ws/src/agx/src/
agilex@agilex:~/ros2_ws/src/agx/src$ sudo gedit publisher.cpp
```

```
#include <chrono>
#include <functional>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using namespace std::chrono_literals;

/* This example creates a subclass of Node and uses std::bind() to register
* member function as a callback from the timer. */

class MinimalPublisher : public rclcpp::Node
{
  public:
    MinimalPublisher()
    : Node("minimal_publisher"), count_(0)
    {
      publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10
      timer_ = this->create_wall_timer(
      500ms, std::bind(&MinimalPublisher::timer_callback, this));
    }

  private:
    void timer_callback()          //时间回调函数
    {
      auto message = std_msgs::msg::String();        //消息数据类型
```

Create a subscriber:

```
agilex@agilex:~$ cd ~/ros2_ws/src/agx/src/
agilex@agilex:~/ros2_ws/src/agx/src$ sudo gedit subscriber.cpp
```

```
#include <memory>
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
using std::placeholders::_1;

class MinimalSubscriber : public rclcpp::Node
{
  public:
    MinimalSubscriber()
    : Node("minimal_subscriber")
    {
    //创建订阅者
      subscription_ = this->create_subscription<std_msgs::msg::String>(
      "topic", 10, std::bind(&MinimalSubscriber::topic_callback, this, _1))
    }

  private:
  //订阅回调函数
    void topic_callback(const std_msgs::msg::String::SharedPtr msg) const
    {
      RCLCPP_INFO(this->get_logger(), "I heard: '%s'", msg->data.c_str());
    }
    //消息类型
    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;
};

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

Modify configuration file:

```
agilex@agilex:~$ cd ~/ros2_ws/src/agx/
agilex@agilex:~/ros2_ws/src/agx$ sudo gedit CMakeLists.txt
```

```
cmake_minimum_required(VERSION 3.5)
project(agx)

# Default to C++14
if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 14)
endif()

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

find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
add_executable(talker src/publisher.cpp)
ament_target_dependencies(talker rclcpp std_msgs)

add_executable(listener src/subscriber.cpp)
ament_target_dependencies(listener rclcpp std_msgs)

install(TARGETS
  talker
  listener
  DESTINATION lib/${PROJECT_NAME})

ament_package()
```

Save and exit:

```
agilex@agilex:~$ cd ~/ros2_ws/
agilex@agilex:~/ros2_ws$ colcon build
Starting >>> agx
Finished <<< agx [0.20s]

Summary: 1 package finished [0.31s]
```

This completes the creation of the workspace, the ROS2 package.

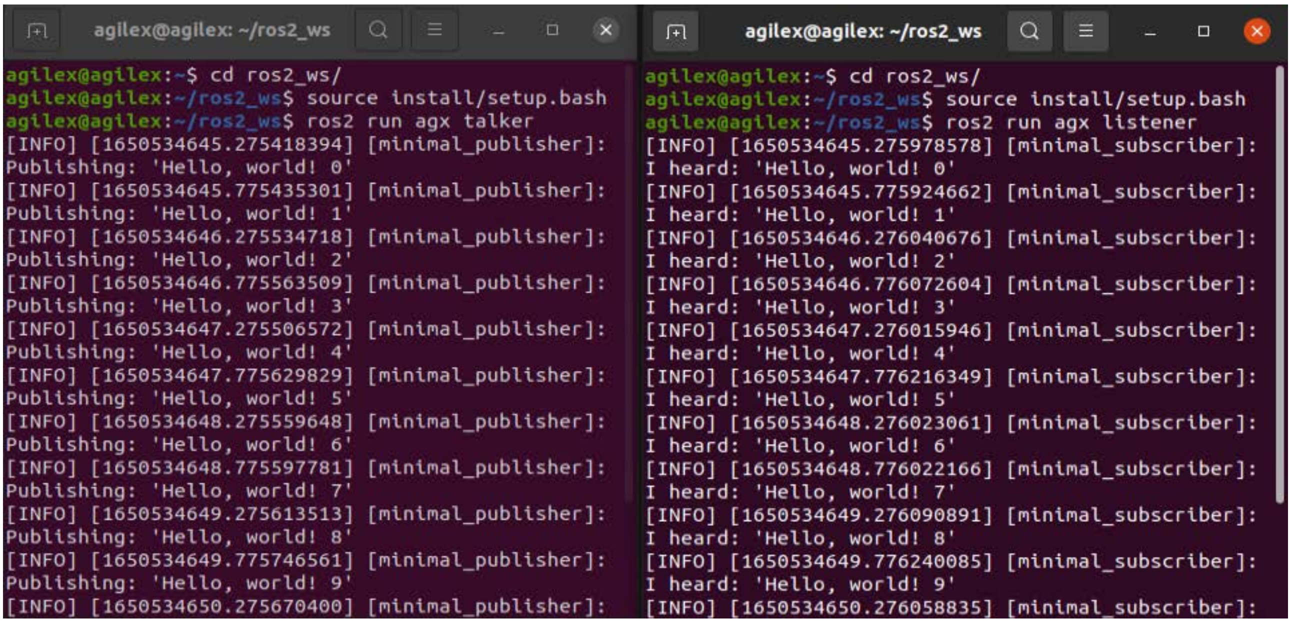
We will verify whether the publishers and subscribers are working properly below:

Publisher:

```
gilex@agilex:~$ cd ros2_ws/
agilex@agilex:~/ros2_ws$ source install/setup.bash
agilex@agilex:~/ros2_ws$ ros2 run agx talker
```

Subscriber:

```
gilex@agilex:~$ cd ros2_ws/
agilex@agilex:~/ros2_ws$ source install/setup.bash
agilex@agilex:~/ros2_ws$ ros2 run agx listener
```



As we can see that the publisher publishes the topic once in 500ms, and subscribers receive the topic and print it out.

Create our own launch file:

```
cd ~/ros2_ws/src/agx/
mkdir -p launch
sudo gedit agx.launch.py
```

```
from launch import LaunchDescription
from launch_ros.actions import Node
def generate_launch_description():
    return LaunchDescription([
        Node(
            package='agx',
            namespace='',
            executable='talker',
            output='screen',
            name='talker'
            ),
    Node(
            package='agx',
            namespace='',
            executable='listener',
            output='screen',
            name='listener'
            ),
    ])
```

Save and exit. Then modify CMakeList.txt .

```
cd ~/ros2_ws/src/agx/
sudo gedit CMakeLists.txt
```

Add the following information in the CMakeList.txt:

```
install(DIRECTORY launch DESTINATION share/${PROJECT_NAME})
```

Save and exit. Then modify the package.xml:

```
sudo gedit package.xml
```

Add the following information in the package.xml:

```
<exec_depend>ros2launch</exec_depend>
```

Save and exit. Then build the whole project:

```
cd ~/ros2_ws/
colcon build
source install/setup.bash
ros2 launch agx agx.launch.py
```

As we can see the terminal shows the information printed by the publisher and subscriber , and we implemented a launch file to launch multiple nodes:

```
[INFO] [launch]: All log files can be found below /home/agilex/.ros/log/2022-
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [talker-1]: process started with pid [109589]
[INFO] [listener-2]: process started with pid [109591]
[talker-1] [INFO] [1650543038.077415868] [talker]: Publishing: 'Hello, world!
[listener-2] [INFO] [1650543038.078129805] [listener]: I heard: 'Hello, world
[talker-1] [INFO] [1650543038.577421428] [talker]: Publishing: 'Hello, world!
[listener-2] [INFO] [1650543038.578005113] [listener]: I heard: 'Hello, world
```

launch:
ROS2 recommend to use python to write the launch files. The name suffix can be .launch.py or _launch.py.
The launch files usually in the directory: ~agilex_ws/src/package/
DeclareLaunchArgument is used to define launching parameters transmitted from launch files or control console.
Start a node:

```
Node(
    package='tf2_ros',
  executable='static_transform_publisher',
    name='static_transform_publisher',
  namespace='',
  arguments='['0','0','0.23','0','0','0','1','base_link','rslidar']',
)
```

executable	The nodes can be executed.
package	Package name for which can be found of the node's executive file.
name	Point the node name.
namespace	ROS namespace for nodes.
parameters	Parameter dictionary or the list for the yaml files including parameter rules.
remappings	ROS remapping rule for transmiting to node.
arguments	Additional parameters list for nodes.

Start other package's launch files:

```
IncludeLaunchDescription(
  PythonLaunchDescriptionSource([os.path.join(
  get_package_share_directory('scout_base'), 'launch'),
  '/scout_mini_base.launch.py'])
  ),
```

Call the parameter configuration files:

```
DeclareLaunchArgument(
   default_value=os.path.join(
   get_package_share_directory("scout_bringup"),
   'config', 'slam_toolbox.yaml'),
```

5.4 Navigation2

The Nav2 project is the successor to ROS Navigation. The project aims to find a safe way to get the robot moved from A point B. It can also be used for other applications involving robot navigation, such as following dynamic points, completing dynamic path planning, calculating motion speed, avoiding obstacles, and recovering behavior.

Nav2 uses the behavior tree to invoke the modular server to complete an action. The action can be calculating path, control, restore, or any other navigation-related action. These are individual nodes that communicate with the behavior tree (BT) through the ROS operation server.

Main plug-ins:

Loading, serving, and storing maps (Map Server)

Locating robots on a Map (AMCL)

Planning a path from A to B around obstacles (Nav2 Planner)

Control the robot by path (Nav2 Controller)

Make the planned path smoother, continuous and workable (Nav2 Smoother)

Convert sensor data into a costmap representation of the world (Nav2 Costmap 2D)

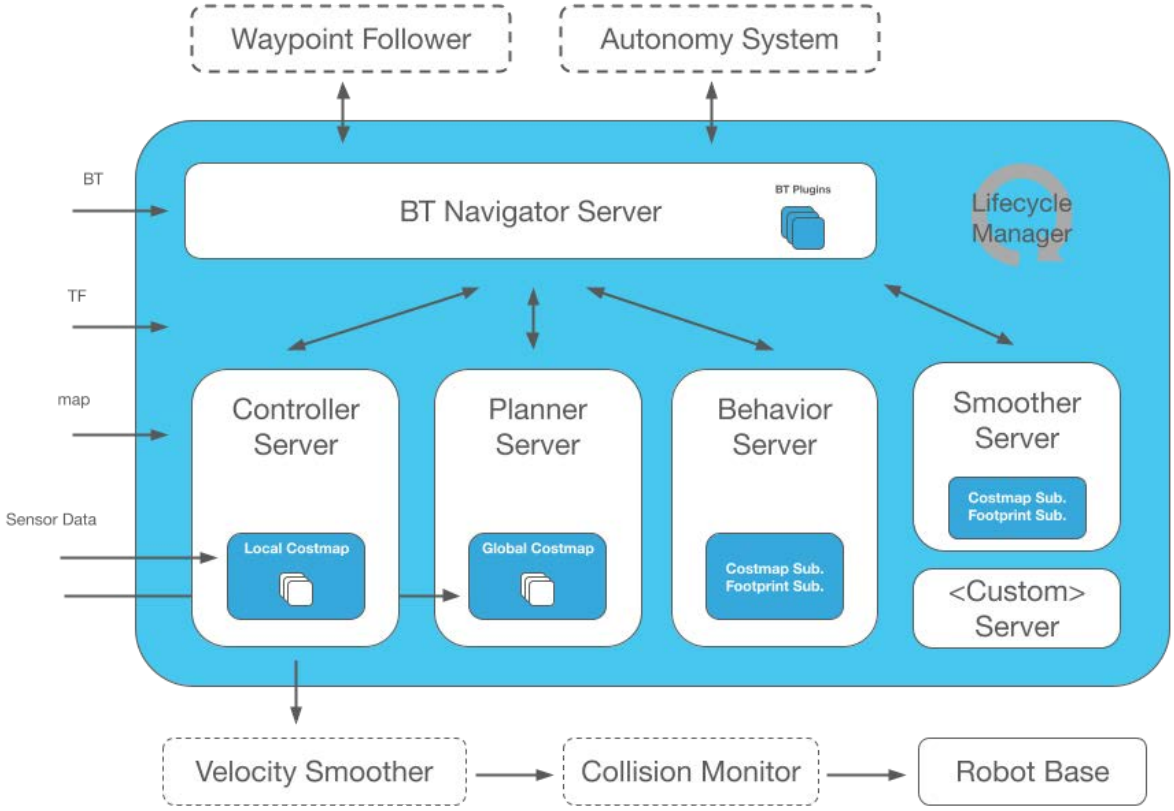
Use Behavior Trees to build the robot's behavior（Nav2 Behavior Trees and BT Navigator)

Calculate the recovery behavior when a fault occurs (Nav2 Recoveries)

Following sequence Waypoint（Nav2 Waypoint Follower)

Manage the server Lifecycle (Nav2 Lifecycle Manager)

Enable your own plugins for custom algorithms and behaviors (Nav2 Core)

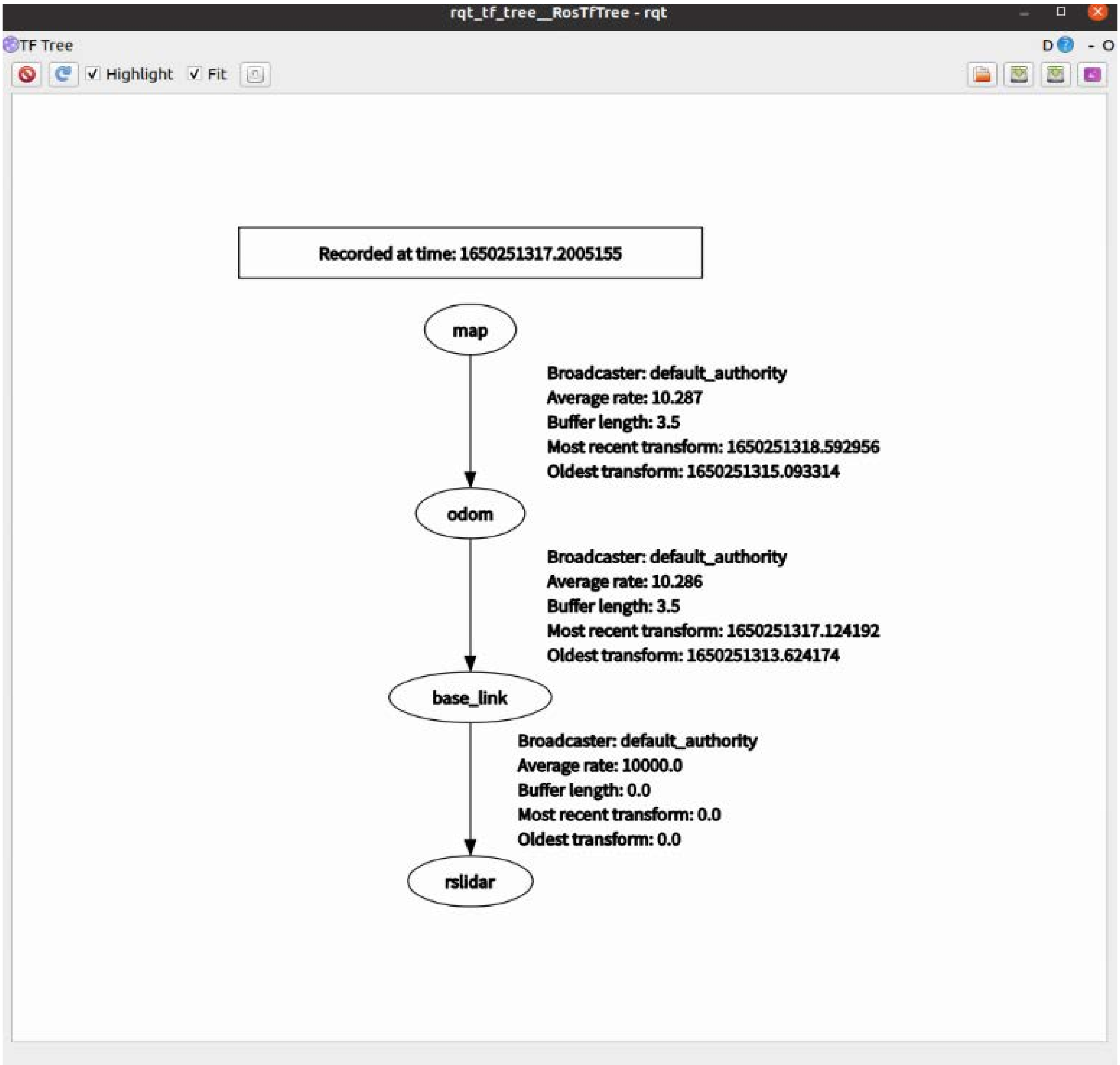


TF tree:

The necessary tf transformation when starting navigation2: map ->odom->base_link->rslidar

We can install the rqt-tf-tree tool to review the real-time TF2-tree.

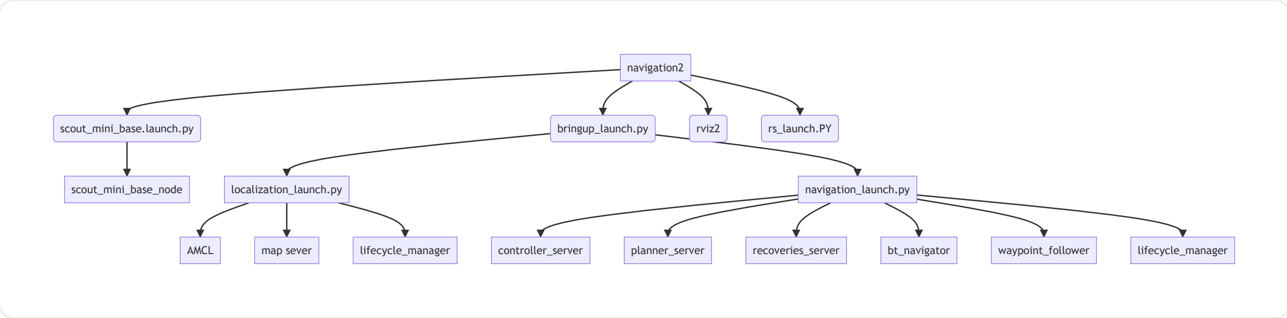
```
sudo apt install ros-foxy-rqt-tf-tree
ros2 run rqt_tf_tree rqt_tf_tree
```



base_link->rslidar is the static transformation. In file open_rslidar.launch.py, it's published by node static_transform_publisher. odom->base_link is dynamic coordinate transformation and it's published by node scout_mini_base_node in navigation2. map->odom is dynamic coordinate transformation and generally provided by AMCL.

launch

when navigation2.launch.py runs, the node relationship mainly started through launch files is shown in the following figure. And users can modify the launch files accordingly.



Create a Navigation2 plug-in:

One of the major changes in Navigation2 is the ability to split the navigation process into modules, which can be added in ros packages as needed. Realize fast iteration of navigation stack function and speed up product development progress.

We will create a line planner plug-in and add it to the navigation stack.

This source code is in directory: /home/agilex/agilex_ws/ SRC/Navigation2 / nav2_straightline_Planner /.

First we go to the workspace and created a ROS package named Nav2_straightline_Planner

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_cmake nav2_straightline_planner
```

In src/ folder, add a straight_line_planner.cpp file and then add the codes below:

```cpp
#include <cmath>
#include <string>
#include <memory>
#include "nav2_util/node_utils.hpp"

#include "nav2_straightline_planner/straight_line_planner.hpp"

namespace nav2_straightline_planner
{

void StraightLine::configure(
  rclcpp_lifecycle::LifecycleNode::SharedPtr parent,
  std::string name, std::shared_ptr<tf2_ros::Buffer> tf,
  std::shared_ptr<nav2_costmap_2d::Costmap2DROS> costmap_ros)
{
  node_ = parent;
  name_ = name;
  tf_ = tf;
  costmap_ = costmap_ros->getCostmap();
  global_frame_ = costmap_ros->getGlobalFrameID();

  // Parameter initialization
  nav2_util::declare_parameter_if_not_declared(
    node_, name_ + ".interpolation_resolution", rclcpp::ParameterValue(
      0.1));
  node_->get_parameter(name_ + ".interpolation_resolution", interpolation_r
}

```

The next step is to create the plug-in's description file in the package's root directory. Create the global_planner_plugin.xml file and add the following code to the file:

```xml
<library path="nav2_straightline_planner_plugin">
  <class name="nav2_straightline_planner/StraightLine" type="nav2_straightlin
    <description>This is an example plugin which produces straight path.</des
  </class>
</library>
```

library path	Name and location for plugin library.
class name	Class name

class type	Class type
base class	Base class name
description	Plugin instruction

The next step is to modify cmakelists.txt, which installs the plug-in description file into a shared directory and sets the directory index to make it discoverable. Open cmakelists.txt and add at the end :

```
cmake_minimum_required(VERSION 3.5)
project(nav2_straightline_planner)

# Default to C99
set(CMAKE_C_STANDARD 99)

# Default to C++14
set(CMAKE_CXX_STANDARD 14)

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(rclcpp_action REQUIRED)
find_package(rclcpp_lifecycle REQUIRED)
find_package(std_msgs REQUIRED)
find_package(visualization_msgs REQUIRED)
find_package(nav2_util REQUIRED)
find_package(nav2_msgs REQUIRED)
find_package(nav_msgs REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(builtin_interfaces REQUIRED)
find_package(tf2_ros REQUIRED)
find_package(nav2_costmap_2d REQUIRED)
find_package(nav2_core REQUIRED)
find_package(pluginlib REQUIRED)

include_directories(
  include
```

Then modify the package.xml and add the plugin description files to it:

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schema
<package format="3">
  <name>nav2_straightline_planner</name>
  <version>1.0.0</version>
  <description>Simple straight line planner.</description>
  <maintainer email="shivaang14@gmail.com">Shivang Patel</maintainer>
  <license>BSD-3-Clause</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>rclcpp_action</depend>
  <depend>rclcpp_lifecycle</depend>
  <depend>std_msgs</depend>
  <depend>visualization_msgs</depend>
  <depend>nav2_util</depend>
  <depend>nav2_msgs</depend>
  <depend>nav_msgs</depend>
  <depend>geometry_msgs</depend>
  <depend>builtin_interfaces</depend>
  <depend>tf2_ros</depend>
  <depend>nav2_costmap_2d</depend>
  <depend>nav2_core</depend>
  <depend>pluginlib</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>
```

Switch to ~/ros2_ws/,build to complete the plugin:

```
cd ~/ros2_ws/
colcon build
```

Now that we have completed to make the plug-in, we'll use our own plug-in in the navigation stack.
Add our own plug-in into the configuration file that launches the navigation stack:

```
cd ~/agilex_ws/src/navigation2/nav2_bringup/bringup/params/
```

Open the configuration file:

```
sudo gedit nav2_params.yaml
```

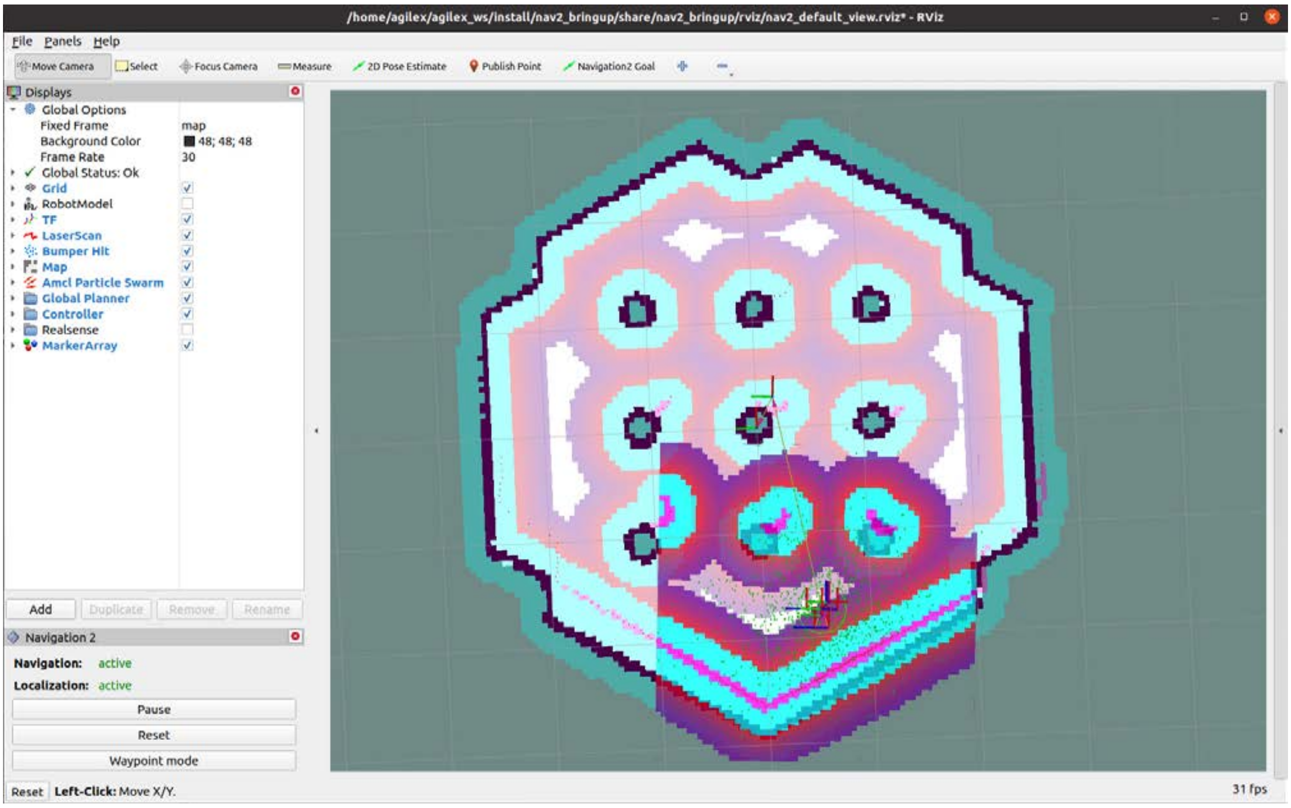
Modify the planner_server to the belwo:

```
planner_server:
  ros__parameters:
    expected_planner_frequency: 20.0
    use_sim_time: True
    planner_plugins: ["GridBased"]
    GridBased:
      plugin: "nav2_straightline_planner/StraightLine"  #我们自己创建的插件
      interpolation_resolution: 0.1
```

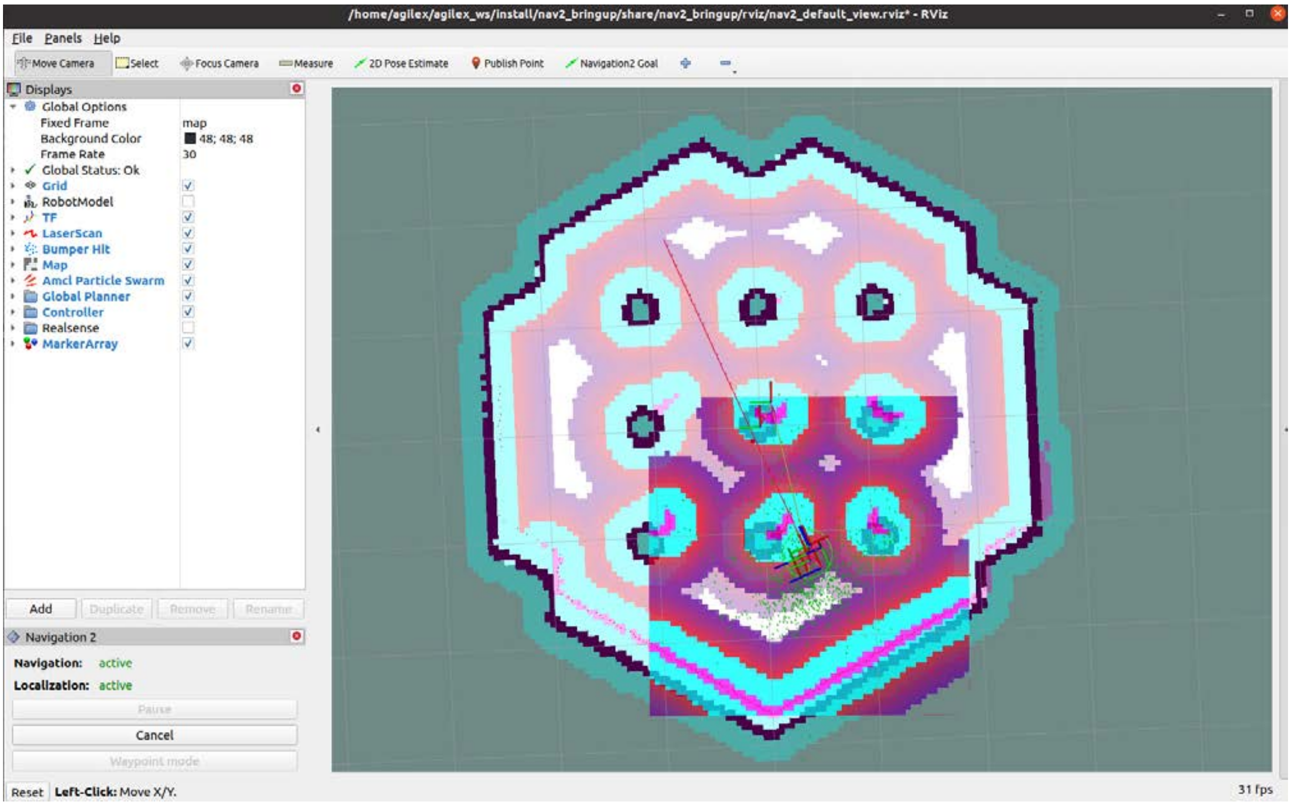
Save and exit. Then go to the workspace:

```
cd ~/agilex_ws
source install/setup.bash
ros2 launch nav2_bringup tb3_simulation_launch.py
```

Click 2D Pose Estimate to set the robot's original position:



Click Navigation2 Goal to set the navigation point：



As we can see that a straight path has been generated, proving that our plug-in is working properly. But in practice such a planner would not be available.

Note: The source of the add-ons is in the directory: /home/agilex/agilex_ws/ SRC/ Navigation2/nav2_Straightline_planner/, which can be modified to suit our own needs. After the plug-in is complete, be sure to remember to source it and change the configuration file that the navigation stack starts to enable the plug-in.

6 QA

Official distributor

gr@generationrobots.com

+33 5 56 39 37 05

www.generationrobots.com