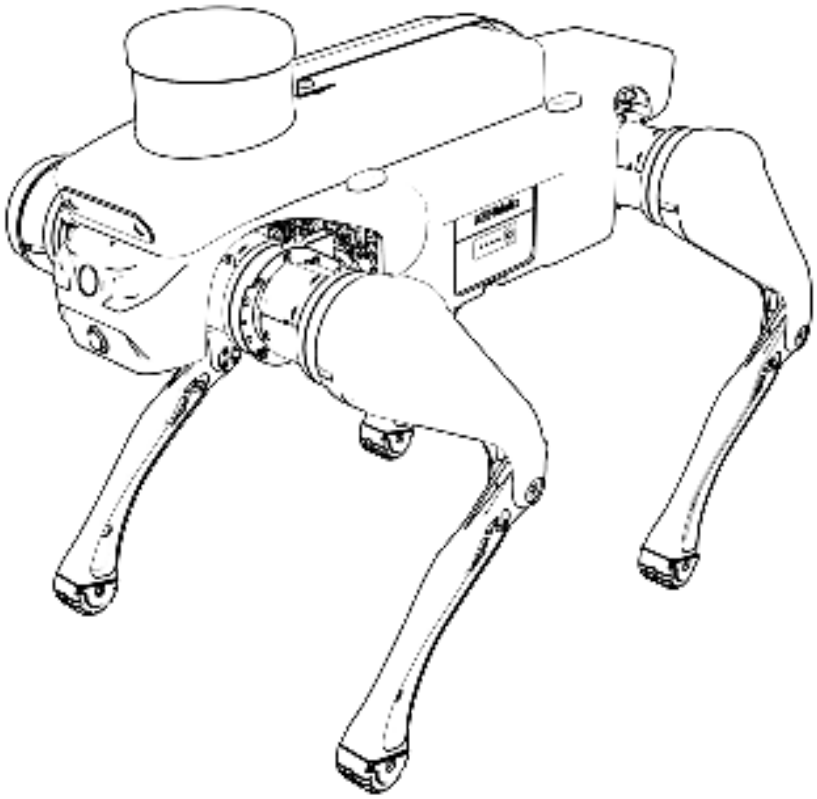
Jueying Lite3

Perception Development Manual(beta)

V2.2.2-0　2024.12.31



Content

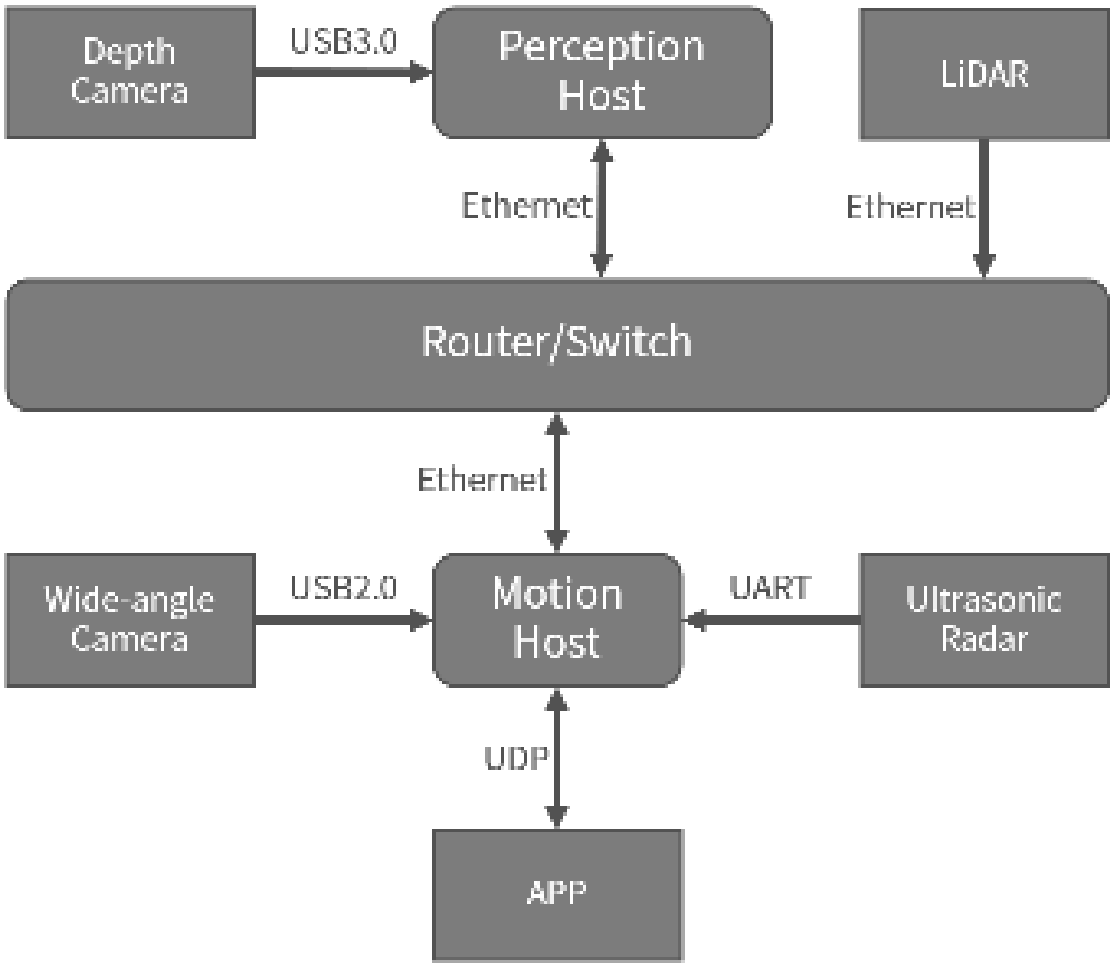
Document Description

This manual is for users who have some expertise and need to explore, develop and validate perception algorithms with Jueying Lite3. This manual version applies for Ubuntu20.

Version	Update Description	Release Date
V1.0.2-0	First release	2023/6/16
V2.0.1-0	Ubuntu 20	2024/5/15
V2.1.1-0	Mapping&HDMI	2024/7/26
V2.2.2-0	ROS2	2024/12/31

1 Perception System

Jueying Lite3 Pro/LiDAR uses **NVIDIA Jetson Xavier NX** as its perception host for perception algorithms calculation. And the robot also provides some perception development examples to facilitate user's development.



2 Preparatory Work

2.1 Remote Desktop

[Caution] If the HDMI connection has been configured according to Section 2.2 before using the remote desktop, please refer to Section 2.2.2 to restore the configuration and restart the robot, or the remote desktop cannot be used.

2.1.1 Connect

Users can remotely log in to the perception host through NoMachine.

1. Connect user's development host to the robot's WiFi.
2. Open NoMachine on your PC, and click "New" or "Add" to create a new connection.
   a) Select "NX" in "Protocol" option
   b) Enter "192.168.1.103" in the "Host" field
   c) Select "Password" in "Use password authentication" option
   d) Do not check "Use custom proxy configuration for this session" in "Proxy"
   e) The rest of the options remain the default
3. Then a new remote icon will appear, as shown in the following screenshot.

4. Click the icon, and enter the user name ysc and password ’ (a single quote), to make a remote connection.

[Caution] After logging in to the perception host, if desktop is locked or a terminal command requires you to enter a password, the password is ' (a single quote).

2.1.2 Troubleshooting

1. If the password is incorrect when you enter the password ’ , try to switch the IME to English and enter ’ again.
2. If you experience a white screen after connecting to the remote desktop, please click the "Settings" button of NoMachine. When the settings page pops out, click "Server Preferences", then "Updates". Click "Check now" to update the software. After the update is complete, try to connect again. If NoMachine shows "session negotiation failed" message after entering the password, you will need to connect to the perception host via SSH from your computer and repair it.

```
ssh ysc@192.168.1.103          # password is ‘ (a single quote)
sudo su
cd /usr/NX/var/db/limits/
ls                             # list files in /limits
rm xxxxx xxxxxx xxxxxx         # delete xxxxx xxxxxx xxxxxx
```

If still shows "session negotiation failed", repeat the operation and restart the perception host using `sudo reboot`.

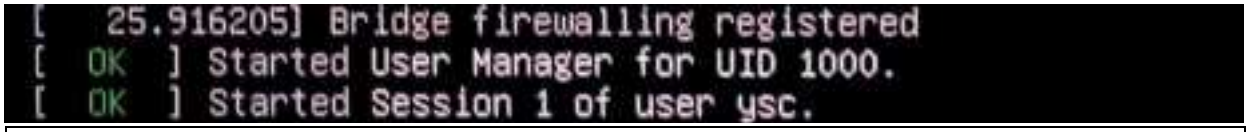
2.2 Connect Perception Host Via HDMI

Jueying Lite3 Pro/LiDAR supports connection to the perception host desktop via the HDMI port on the back. The perception host automatically boots into tty3 terminal by

default. If you want to enter the GUI(Graphic user interface) automatically after the host is started, you need to make related settings.

2.2.1 Start GUI Automatically on Boot

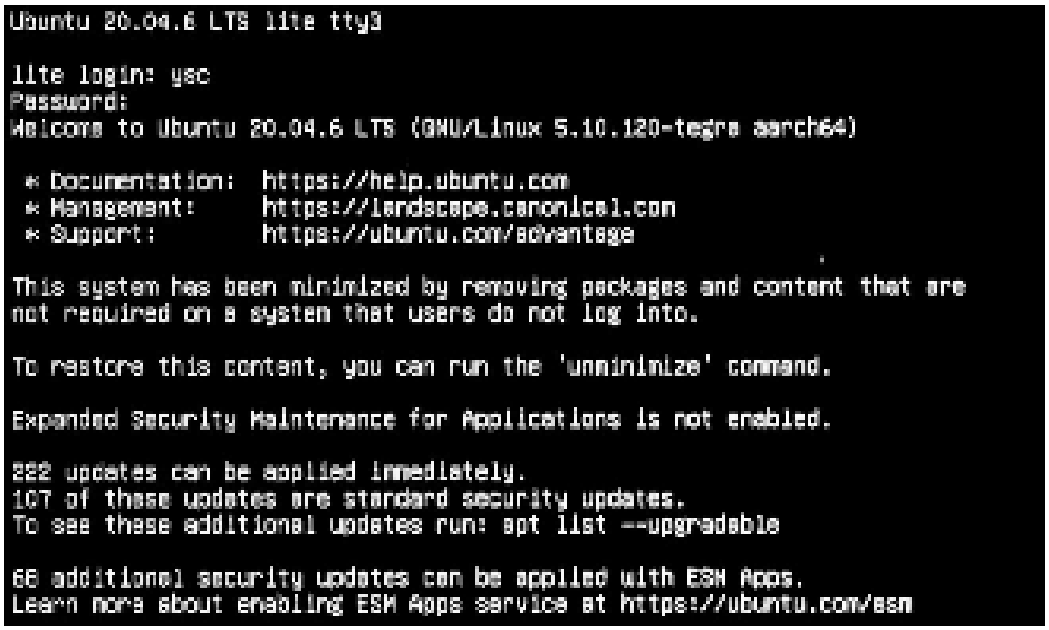
1. First, use an HDMI cable to connect the perception host to the monitor and boot the robot. The monitor will show the boot process. If the screen shows "[OK] Started Session 1 of user ysc. ", the system starts successfully.



[Caution] If the NVIDIA logo or "[OK] Started Session 1 of user ysc. " doesn't show on screen after the system starts, but " [Failed] " is displayed, the perception host hardware may be faulty. Please contact after-sales personnel for help.

2. After successfully booting the robot, use USB interface to connect keyboard, and press "Ctrl+Alt+F3" to enter the tty3 terminal.

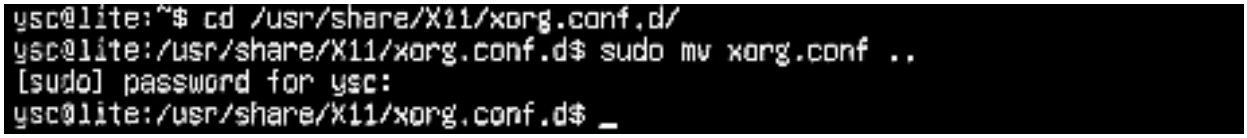
3. Then enter the user name ysc and password ' (a single quote) to log in.



4. After successfully logging in (pictured above), navigate to "/usr/share/X11/xorg conf.d" directory and move the "xorg.conf" file to other directory. Specific commands are as follows:

```
cd /usr/share/X11/xorg.conf.d/
sudo mv xorg.conf ..          #move the xorg.conf file up to /usr/share/X11
```

And enter the password ' (single quotes) after "[sudo] password for ysc:".



5. After rebooting the robot, the perception host will automatically enter the GUI. If the GUI does not appear, it may be because the system has not cleared the previously configured cache. Please reboot the robot again.

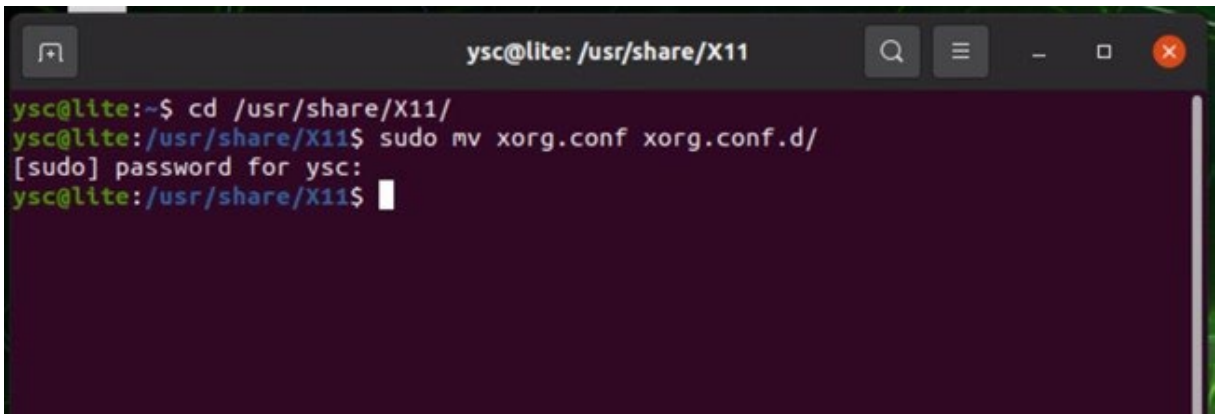
2.2.2 Start tty3 Automatically On Boot

Move "xorg.conf" back to the original path and reboot the robot to restore the tty3 to boot. Method is as follows:

1. Boot the robot, after entering the GUI, press "Alt+Ctrl+T" to open the terminal window and enter the following:

```
cd /usr/share/X11/
sudo mv xorg.conf xorg.conf.d/ # Move the xorg.conf file to xorg.conf.d/
```

2. Enter the password ' (single quotes) to move the file. If the move fails, it may be that the "xorg.conf" file is not under "/usr/share/X11/". Find the actual location of the "xorg.conf" file and move it to the "/usr/share/X11/xorg.conf.d/" directory.



3. Reboot the robot. The perception host will enter the tty3 terminal by default while booting.

3 Check and Change ROS version

Jueying Lite3 supports both ROS1 and ROS2 since V3.2.1.

1. Run the following command in the terminal to check the ROS version currently in use

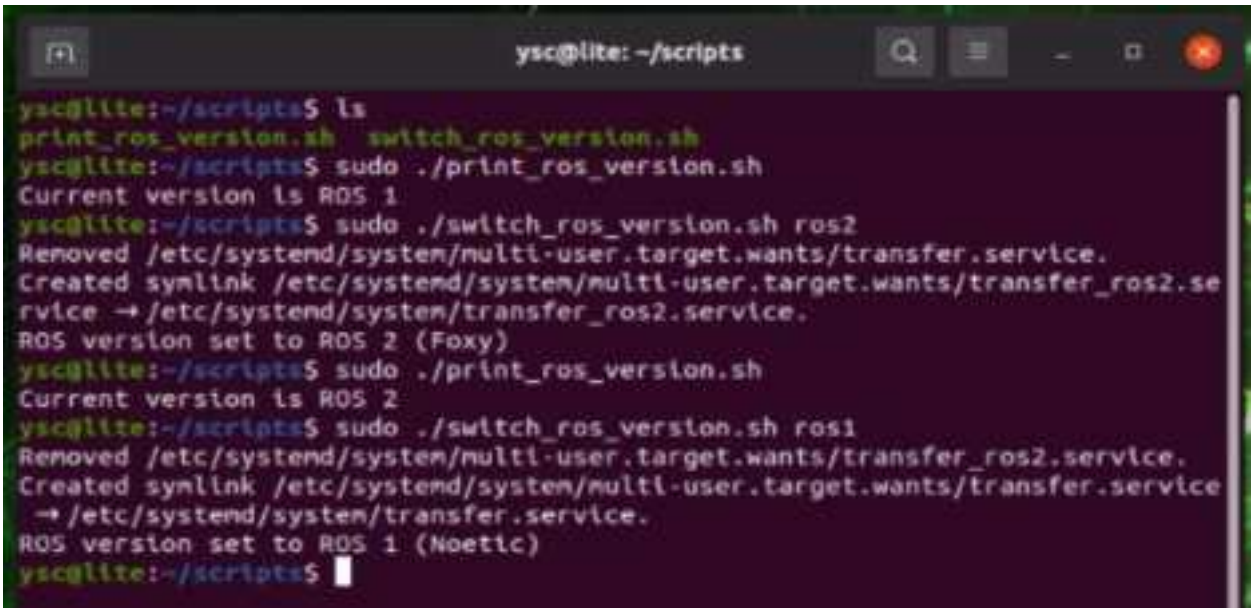
```
cd /home/ysc/scripts
sudo ./print_ros_version.sh
```

2. Run the following command in the terminal to switch to ROS1:

```
cd /home/ysc/scripts
sudo ./switch_ros_version.sh ros1
```

3. Run the following command in the terminal to switch to the ROS2:

```
cd /home/ysc/scripts
sudo ./switch_ros_version.sh ros2
```



4. Restart your robot.

> [Caution] Do not run ROS1 and ROS2 programs at the same time to avoid unnecessary resource usage and errors.

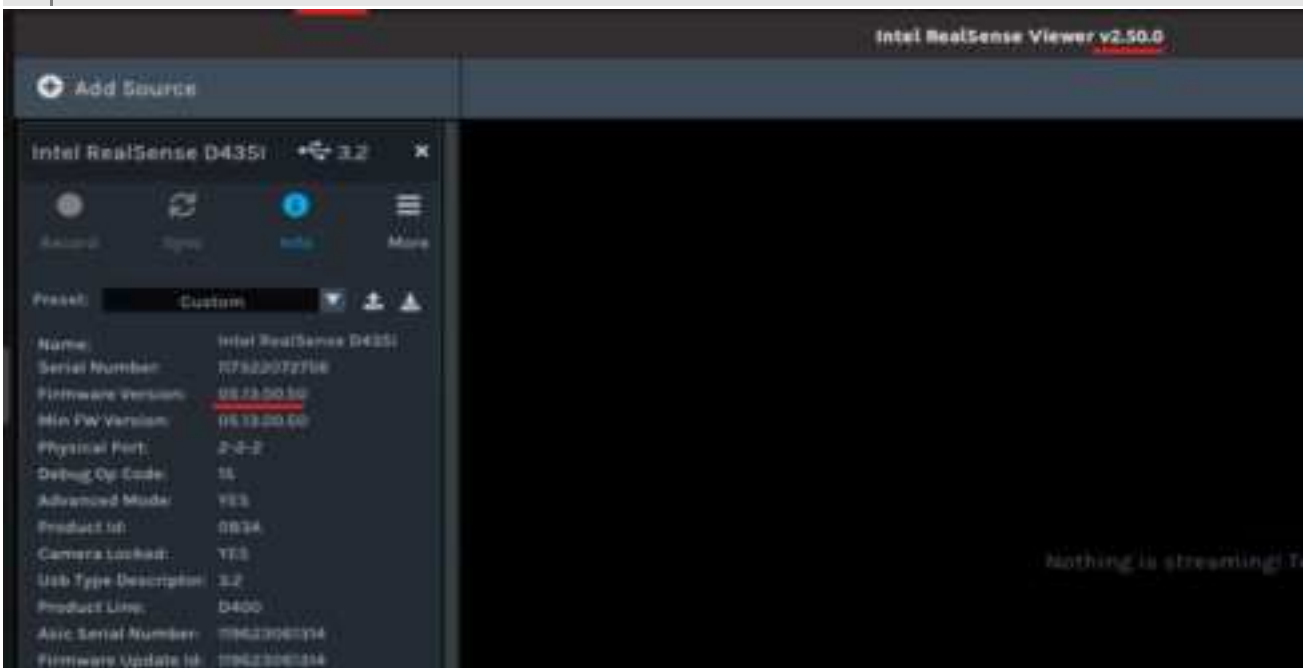
4 Depth Camera

Jueying Lite3 Pro/LiDAR is equipped with Intel RealSense D435i.

4.1 Camera Driver

The depth camera driver Intel RealSense SDK has been installed on the perception host of Jueying Lite3. Users can open the visualization tool provided by Intel by entering the following command line in the Terminal:

```
realsense-viewer
```



Click on the "Info" button for more detailed parameter information, such as the serial number, firmware version and so on.

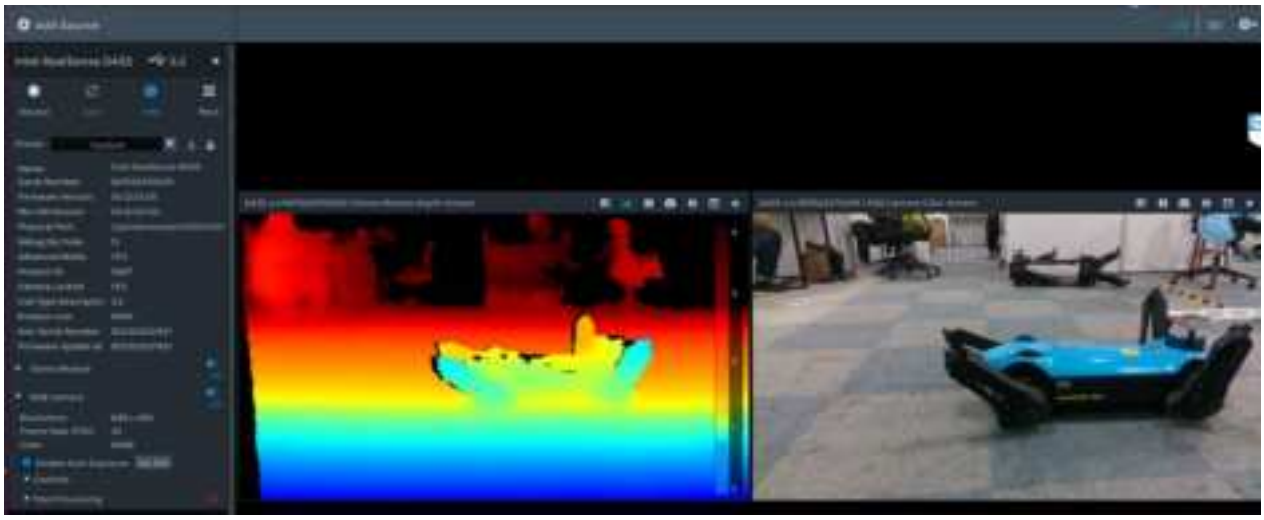
[Caution]The realsense-ros package depnds on *librealsense* v2.50.0, which corresponds to the Depth Camera firmware version 05.13.00.50. The Version of *librealsense* is displayed in the window title, and the Firmware Version is displayed by clicking the “Info” button.

Click the triangle to Expand "Stereo Module" or "RGB Camera" and you can configure camera parameters such as resolution and frame rate.

4.2 Camera Test

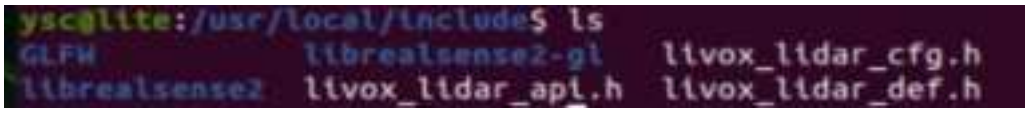
Before using the camera driver for development, first check whether the depth camera is connected normally:

1. Make sure that an Intel RealSense D435i is added;
2. Click the on/off switch of the stereo module and the RGB camera. If the depth map and color map are successfully displayed, it indicates that the depth camera is properly connected.



4.3 Camera Library *librealsense*

The Library *librealsense* and related libraries are compiled based on CUDA and have been installed in /usr/local/lib and /usr/local/include. When development, you can include corresponding header files and link corresponding libraries for compilation.



4.4 Realsense Development (ROS1)

The realsense-ros package of ROS1 is located in the /home/ysc/lite_cog/drivers /realsense_ws directory, and you can start/stop the relevant functions and check the status by system service, corresponding to the following commands:

```
sudo systemctl start realsense
sudo systemctl stop realsense
sudo systemctl status realsense
```

The commands will use the dr_camera.launch file in the ~/lite_cog/drivers/realsense_ws/src/realsense2_camera/launch folder to start the realsense camera.

If you need to modify the startup parameters of the camera, modify the launch file.

4.5 Realsense Development (ROS2)

The realsense-ros package of ROS2 is located in the /home/ysc/ lite_cog_ros2/drivers/ realsense_ws directory, and you can start/stop the relevant functions and check the status by system service, corresponding to the following commands:

```
sudo systemctl start realsense_ros2
sudo systemctl stop realsense_ros2
sudo systemctl status realsense_ros2
```

The commands will use dr_camera_launch.py file in the ~/lite_cog_ros2/drivers/realsense_ws /src/realsense2_camera/launch folder to start the realsense camera.

If you need to modify the startup parameters of the camera, modify the launch file

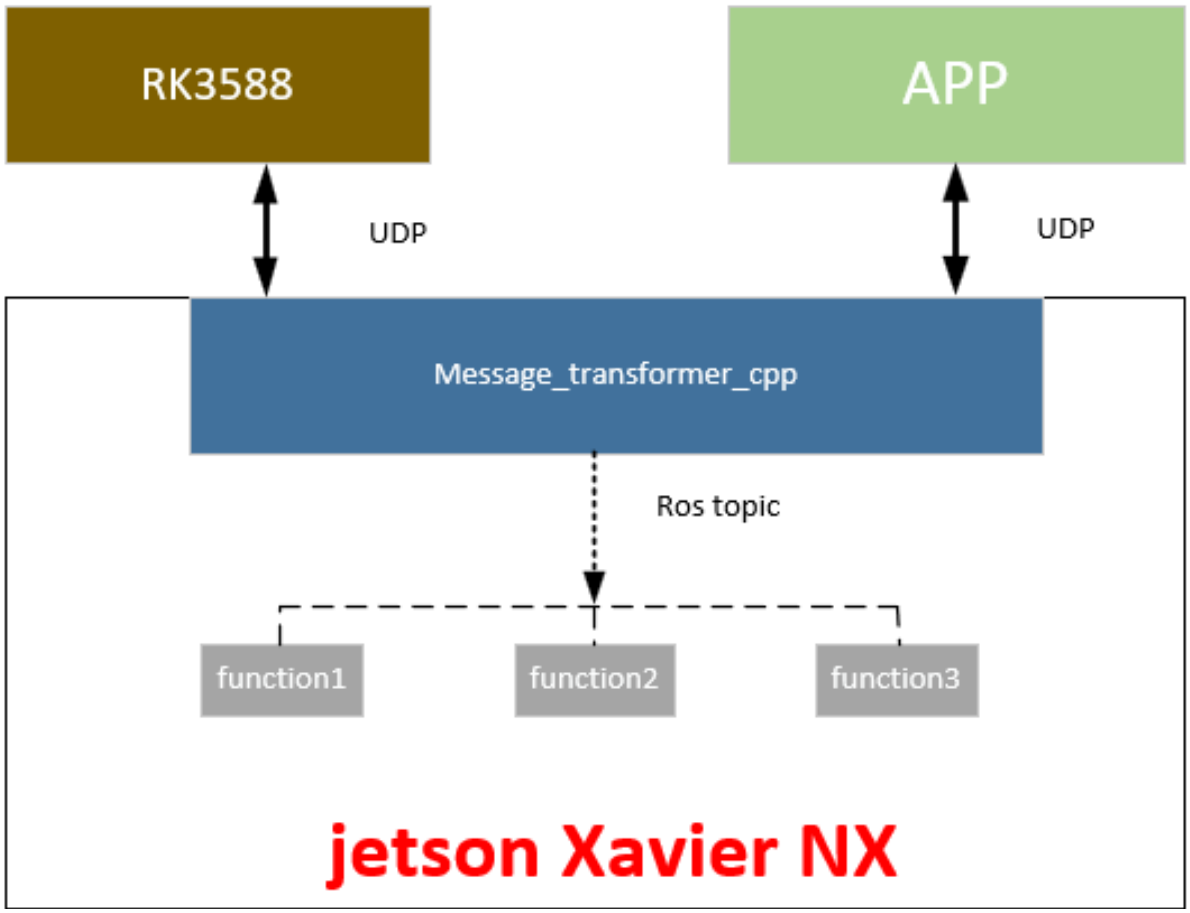
5 Message Transformer

5.1 Introduction

This package enables the conversion between ROS and UDP messages.

The data transmission between the perception host and the motion host or app is based on the UDP protocol. **Message_transformer** can be used as the following:

1. transform UDP messages sent by motion host into ROS topic messages and publish, and send motion control commands issued by perception host to motion host using UDP;
2. receive control commands from the app to turn on and off some AI functions on perception host.



ROS topics:

Message_transformer will receive the UDP messages from motion host and publish them to the following topics:

```
Leg Odometry Data:        /leg_odom       (nav_msgs::Odometry)
IMU Data:                 /imu/data       (sensor_msgs::Imu)
Joint Data:               /joint_states   (sensor_msgs::JointState)
```

Message_transformer will subscribe to the following topics and send the topic messages to motion host:

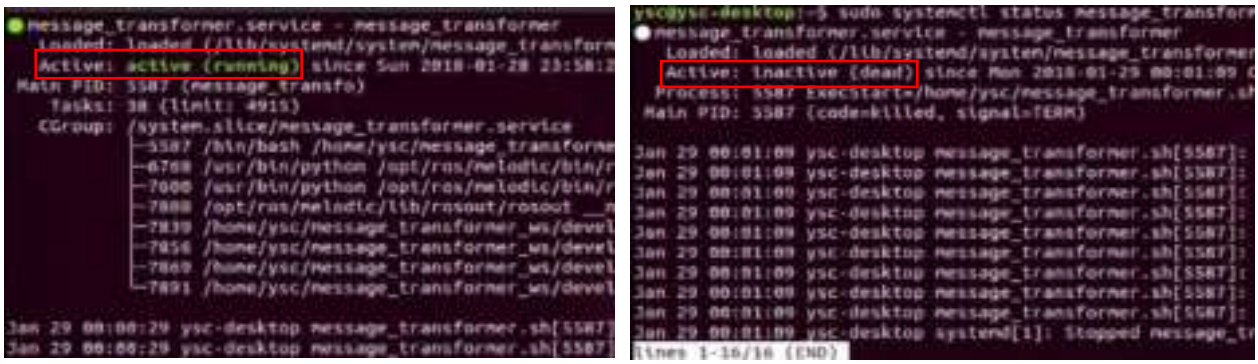
```
Velocity Command:         /cmd_vel        (geometry_msgs::Twist)
```

5.2 Usage

5.2.1 ROS1

1. Open a new terminal and enter the following codes to **check the status of message_transformer**:

```
1 sudo systemctl status message_transformer.service
```



a) If the status is active, message_transformer is running and can be used directly;

b) If the status is inactive, please enter the following command in a terminal to **start message_transformer:**

```
1 sudo systemctl start transfer
```

c) The command to **stop message_transformer:**

```
1 sudo systemctl stop transfer
```

d) The command to **view the real-time logs** of message_transformer:

```
journalctl -fu  transfer
```

2. Open a new terminal and use rostopic command to check the robot status:

```
rostopic info xxxxxx
rostopic echo xxxxxx
# xxxxxx refers to the topic name, and users can subscribe to the topic for development
```

3. Use the topic `/cmd_vel` to send velocity commands to motion host, in the format of `geometry_msgs/Twist`:

```
geometry_msgs/Vector3 linear   # Linear velocity (m/s)
float64 x                      # Longitudinal velocity: positive value when going
forward
float64 y                      # Lateral velocity: positive value when going left
float64 z                      # Invalid parameter
geometry_msgs/Vector3 angular  # Angular velocity (rad/s)
float64 x                      # Invalid parameter
float64 y                      # Invalid parameter
float64 z                      # Angular velocity: positive value when turning left
```

a) Users can publish to this topic in C++ or Python programs compiled based on ROS (refer to http://wiki.ros.org/ROS/Tutorials for learning about ROS). Users can also publish messages to the topic for debugging in terminal. Please first type the following codes in terminal:

```
rostopic pub /cmd_vel geometry_msgs/Twist
```

b) Before pressing Enter, add a space after the commands and press Tab key to automatically complement the message type as follows:

```
rostopic pub /cmd_vel geometry_msgs/Twist "linear:
x: 0.0
y: 0.0
z: 0.0
angular:
x: 0.0
y: 0.0
z: 0.0
"
```

c) Use the left/right arrow keys on the keyboard to move the cursor, modify the velocity values, and then add `-r 10` after `geometry_msgs/Twist` to specify the posting frequency (10Hz) as follows:

```
rostopic pub /cmd_vel geometry_msgs/Twist -r 10 "linear:
x: 0.2
y: 0.1
z: 0.0
angular:
x: 0.0
y: 0.0
z: 0.3
"
```

d) Press Enter key to run and publish the topic messages.

e) Message_transformer can subscribe to this topic, transform the topic messages into UDP messages and send them to motion host.

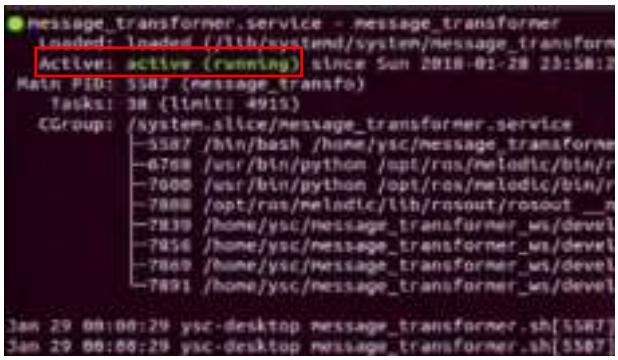
f) After the transmission process is normally opened, make the robot stand up and start the auto mode in the APP Settings page, and the robot can act at the above speed.

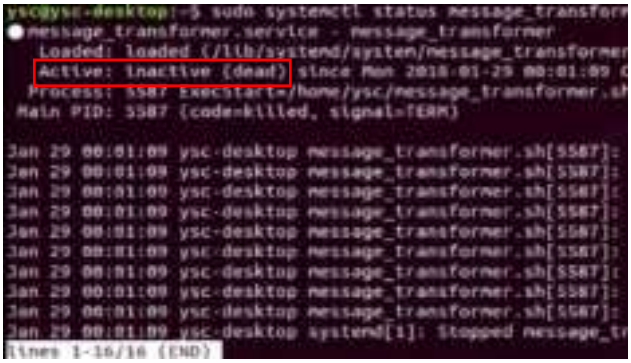
[Caution] Please debug in an open area to prevent damage to people or objects. In case of an emergency, press the STOP button in time, or turn off the auto mode.

5.2.2 ROS2

1. Open a new terminal and enter the following codes to **check the status of message_transformer**:

```
sudo systemctl status transfer_ros2
```





a) If the status is active, message_transformer is running and can be used directly;

b) If the status is inactive, please enter the following command in a terminal to **start message_transformer:**

```
sudo systemctl start transfer_ros2
```

c) The command to **stop message_transformer:**

```
sudo systemctl stop transfer_ros2
```

d) The command to **view the real-time logs** of message_transformer:

```
journalctl -fu  transfer_ros2
```

2. Open a new terminal and use ros2 topic command to check the robot status:

```
ros2 topic info xxxxxx
ros2 topic echo xxxxxx
# xxxxxx refers to the topic name, and users can subscribe to the topic for development
```

3. Use the topic `/cmd_vel` to send velocity commands to motion host, in the format of `geometry_msgs/Twist`:

```
geometry_msgs/msg/Vector3 linear   # Linear velocity (m/s)
float64 x                      # Longitudinal velocity: positive value when going forward
float64 y                      # Lateral velocity: positive value when going left
float64 z                      # Invalid parameter
geometry_msgs/mag/Vector3 angular  # Angular velocity (rad/s)
float64 x                      # Invalid parameter
float64 y                      # Invalid parameter
float64 z                      # Angular velocity: positive value when turning left
```

a) Users can publish to this topic in C++ or Python programs compiled based on ROS2 (refer to https://docs.ros.org/en/foxy/Tutorials.html for learning about

ROS). Users can also publish messages to the topic for debugging in terminal.

Please first type the following codes in terminal:

```
ros2 topic pub -r 10 /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.2, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

b) Press Enter key to run and publish the topic messages.

c) Message_transformer can subscribe to this topic, transform the topic messages into UDP messages and send them to motion host.

d) After the transmission process is normally opened, make the robot stand up and start the auto mode in the APP Settings page, and the robot can act at the above speed.

[Caution] Please debug in an open area to prevent damage to people or objects. In case of an emergency, press the STOP button in time, or turn off the auto mode.

5.3 Package Structure

5.3.1 ROS1

```
/home/ysc/lite_cog/transfer/src
├── CMakeLists.txt
└── message_transformer
    ├── CMakeLists.txt
    ├── include
    │   ├── protocol.h
    │   └── sensor_logger.h
    ├── launch
    │   └── message_transformer.launch
    ├── msg
    │   ├── SimpleCMD.msg
    │   └── ComplexCMD.msg
    ├── package.xml
    └── src
        ├── nx2app.cpp
        ├── qnx2ros.cpp
        ├── ros2qnx.cpp
```

```
        └── sensor_checker.cpp
```

1. ***nx2app.cpp*** is mainly used for UDP communication between perception host and app. The app sends command code to perception host and ***nx2app.cpp*** will execute tasks according to the received command. The commands sent by app are structured as follows:

```
class SimpleCMD{
public:
  int32_t cmd_code;
  int32_t cmd_value;
  int32_t type;
};
```

2. ***qnx2ros.cpp*** is used to receive the data sent by motion host and transform it into ROS topic messages.

```
leg_odom_pub_ = nh.advertise<geometry_msgs::PoseWithCovarianceStamped>("leg_odom", 1);
leg_odom_pub2_ = nh.advertise<nav_msgs::Odometry>("leg_odom2", 1);
joint_state_pub_ = nh.advertise<sensor_msgs::JointState>("joint_states", 1);
imu_pub_ = nh.advertise<sensor_msgs::Imu>("/imu/data", 1);
handle_pub_ = nh.advertise<geometry_msgs::Twist>("/handle_state", 1);
ultrasound_pub_ = nh.advertise<std_msgs::Float64>("/us_publisher/ultrasound_distance",
1);
```

3. ***ros2qnx.cpp*** can subscribe to the topic published by other nodes, transform the messages into UDP data and send them to motion host.

```
ros::Subscriber vel_sub = nh.subscribe("cmd_vel", 1, &ROS2QNX::CmdVelCallback,
&ros2qnx);
ros::Subscriber vel_sub2 = nh.subscribe("cmd_vel_corrected", 1,
&ROS2QNX::CmdVelCallback, &ros2qnx);
ros::Subscriber simplecmd_sub = nh.subscribe("simple_cmd", 1,
&ROS2QNX::SimpleCMDCallback, &ros2qnx);
ros::Subscriber complexcmd_sub = nh.subscribe("complex_cmd", 1,
&ROS2QNX::ComplexCMDCallback, &ros2qnx);
```

5.3.2 ROS2

```
ysc@lite:~/lite_cog_ros2/transfer/src$ tree .
├── transfer
|   ├── CMakeLists.txt
```

```
|   ├── include
|   |   └── protocol.hpp
|   ├── launch
|   |   └── transfer_launch.py
|   ├── package.xml
|   └── src
|       ├── Jetson2App.cpp
|       ├── Jetson2Motion.cpp
|       ├── SensorChecker.cpp
|       └── SensorsLogger.hpp
└── transfer_interfaces
    ├── CMakeLists.txt
    ├── msg
    |   ├── MotionComplexCMD.msg
    |   └── MotionSimpleCMD.msg
    └── package.xml
```

1. ***Jetson2App.cpp*** is mainly used for UDP communication between perception host and app. The app sends command code to perception host and ***Jetson2App.cpp*** will execute tasks according to the received command. The commands sent by app are structured as follows:

```
class SimpleCMD{
public:
  int32_t cmd_code;
  int32_t cmd_value;
  int32_t type;
};
```

2. The ***Jetson2Motion.cpp*** contains two classes: ***MotionReceiver*** and ***MotionSender***. The ***MotionReceiver*** is used to receive the data reported by the motion host and convert it into a ROS topic message for other function packages to use.

```
leg_odom_pub_ = create_publisher<geometry_msgs::msg::PoseWithCovarianceStamped>("leg_odom", 10);
leg_odom_pub2_ = create_publisher<nav_msgs::msg::Odometry>("leg_odom2", 10);
joint_state_pub_ = create_publisher<sensor_msgs::msg::JointState>("joint_states", 10);
imu_pub_ = create_publisher<sensor_msgs::msg::Imu>("/imu/data", 10);
handle_pub_ = create_publisher<geometry_msgs::msg::Twist>("/handle_state", 10);
ultrasound_pub_ = create_publisher<std_msgs::msg::Float64>("/us_publisher/ultrasound_distance", 10);
```

MotionSender subscribes to topics published by other feature pack nodes, converts them into UDP message packages and sends them to the motion host, and the topics subscribed to are as follows:

```
cmd_vel_sub_ = create_subscription<geometry_msgs::msg::Twist>(
    "cmd_vel", 10,
    std::bind(&MotionSender::CmdVelCallback, this, std::placeholders::_1));
cmd_vel_corrected_sub_ = create_subscription<geometry_msgs::msg::Twist>(
    "cmd_vel_corrected", 10,
    std::bind(&MotionSender::CmdVelCallback, this, std::placeholders::_1));
simplecmd_sub = create_subscription<transfer_interfaces::msg::MotionSimpleCMD>(
    "simple_cmd", 10,
    std::bind(&MotionSender::SimpleCMDCallback, this, std::placeholders::_1));
complexcmd_sub = create_subscription<transfer_interfaces::msg::MotionComplexCMD>(
    "complex_cmd", 10,
    std::bind(&MotionSender::ComplexCMDCallback, this, std::placeholders::_1));
```

6 People Tracking

6.1 Introduction

This case first utilizes DeepStream, YOLOv8 and TensorRT to recognize and track the target individuals in the scene and then calculates the target position and transmits it to the motion host to enable the robot to follow the target people. Hardware decoding based on DeepStream is used to obtain an h264-encoded 720p resolution rtsp video stream, and TensorRT is used to accelerate the Yolov8 human detection model to recognize people in open scenes, enabling it to recognize people at approximately 20 fps and track them at around 10 fps.

This case is divided into two parts: **recognition and tracking**.

1. **Recognition algorithm** performs deep learning neural network for visual recognition to find the position of human body in the picture. When multiple bodies appear in the picture, all the human bodies in the frame are first recognized. Then, the features of the human body identified in each frame of video are extracted based on deep learning and compared one by one to determine the trajectory of the same person in the previous and subsequent frames.
2. **Tracking algorithm** allows users to choose the target human they want the robot to follow in the screen. The robot can achieve real-time targeting and continuous tracking. The recognition algorithm can determine the direction and distance of people from the robot so that the robot can respond accordingly (translate or

rotate). Its velocity can also be adjusted in real-time depending on the distance between the robot and the person being tracked.

a) Too close: When the target is too close, the robot will stop to prevent a collision.

b) Close: When the target is close, the robot will dynamically slow down in real-time to get close to the target.

c) Far: The robot will move at maximum speed when the target is far away.

The source codes of yolov8 and sdk_hub used in this case are from ultralytics and hub-sdk. Also you can search materials about yolov8 on the Internet.

6.2 Usage

[Caution]When the program is started, the initialization of video decoding and deep learning inference environment are required, which takes about 40s. If the function cannot be started for a long time, connect the controller to the robot to check whether the video stream works properly.

1. Open a Terminal and enter the following command to start the program:

```
cd /home/ysc/lite_cog/track/src    # for ROS1
cd /home/ysc/lite_cog_ros2/track/src    # for ROS2
python3 run_tracker.py
```

2. Use the app to make the robot stand up and start the auto mode.
3. When people appear, the system will assign numbers to all the people who has been identified and displayed the numbers on the screen. Use the keyboard to enter the assigned number of the person you want to follow and press enter to confirm.

[Caution] When entering a target number, kindly ensure that the video window is on top.

4. The robot will then track and identify the target.

5. You can press Enter key to reset the target when tracking, or when the target is lost and "Miss Object" is displayed.
6. Press Ctrl+c to end the program.

6.3 Development

This case provides two packages in the ~/lite_cog/track and ~/lite_cog_ros2/track workspaces for different ROS versions, and the difference is only the `kUseRos1Transfer = True` line of RobotController.py in the /src/RobotController folder, which is used to specify whether to use ROS1 or ROS2 for message sending. ROS1 is used when the value is True, and ROS2 is used when the value is False.

The structure of package **people_tracking** is as below:

```
People_tracking
├── model
│   ├── export_engine.sh
│   ├── yolov8n_amd.engine
│   ├── yolov8n_arm.engine
│   ├── yolov8n.onnx
│   └── yolov8n.pt
└── src
    ├── GStreamerWrapper
    │   └── GStreamerWrapper.py
    ├── hub_sdk
 ├── RobotController
    │   ├── FpsCounter
    │   │   └── FpsCounter.py
    │   ├── RobotController.py
    │   ├── ROSTransfer
    │   │   ├── ROS1Transfer.py
    │   │   ├── ROS2Transfer.py
    │   │   └── TransferConstants.py
    │   └── YoloWrapper
    │       ├── CocoTypeId.py
    │       └── YoloWrapper.py
    ├── run_tracker.py
```

```
    ├── test
    │   ├── pull.py
    │   ├── pull.sh
    │   └── yolov8.py
    └── ultralytics
```

1. **run_tracker.py** is the main program.
2. **GStreamerWrapper** is a DeepStream-based GStreamer hardware decoder used to obtain RTSP video streams.
3. The main operation logic of RobotController.py is reflected in its Run() function, which is used to identify the human body in the image obtained from the video stream, and then send motion instructions.
4. **ultralytics** is an open source Yolov8 program package, which is used to reason and track image frames obtained from video streams, and is the operation dependency of YoloWrapper in RobotController. The **ultralytics/cfg** folder is the storage address of various Yolov8 configuration files. Each configuration file has been fully commented.
5. **sdk_hub** is the open source sdk_hub program package, which is the operation dependency of the Yolov8 package.

7 LiDAR-based SLAM and Navigation (ROS1)

[Caution] This section applies to ROS1, so please refer to “3 Check and Change ROS version” to check the ROS version currently in use.

This case uses LiDAR and imu to achieve mapping (indoor and outdoor scenes), localization, navigation, and obstacle avoidance on perception host. The robot can achieve real-time localization and online 3D mapping. When localizing in a map, by fusing IMU, it will not lose its location due to falling or high-speed rotation. Map-based navigation is achieved using the ***move_base*** package.

[Caution] Before mapping, please check "～/Desktop/version_log.txt" document. If the version is v3.1.04 or later, please refer to 7.1 for mapping. If the version is earlier than v3.1.04, please refer to 7.2 for mapping.

7.1 Mapping(for v3.1.04 and later)

7.1.1 Introduction

This case uses SLAM Mapping released on Github by Dr. Gao Xiang's team. The main operation process and data flow diagram are shown as below:

7.1.2 Package Structure

The mapping package is located in the "~/lite_cog/slam/src" path and contains three packages: ***faster-lio***, ***pcd_2_gridmap*** and ***map_server***. The ***faster-lio*** package is responsible for building 3D point cloud maps (.pcd). The ***pcd_2_gridmap*** package is

responsible for converting 3D point cloud maps (.pcd) to grid maps (.pgm) and publishing them. The ***map_server*** package is responsible for saving grid maps (.pgm).

7.1.3 Usage

> [Caution] Before mapping, please check whether there is a previously created map in the /home/ysc/lite_cog/system/map folder. If so, you can move it to another folder to avoid overwriting.
> [Caution] Mapping requires more computing resources, so please turn off all the AI options on the app first.

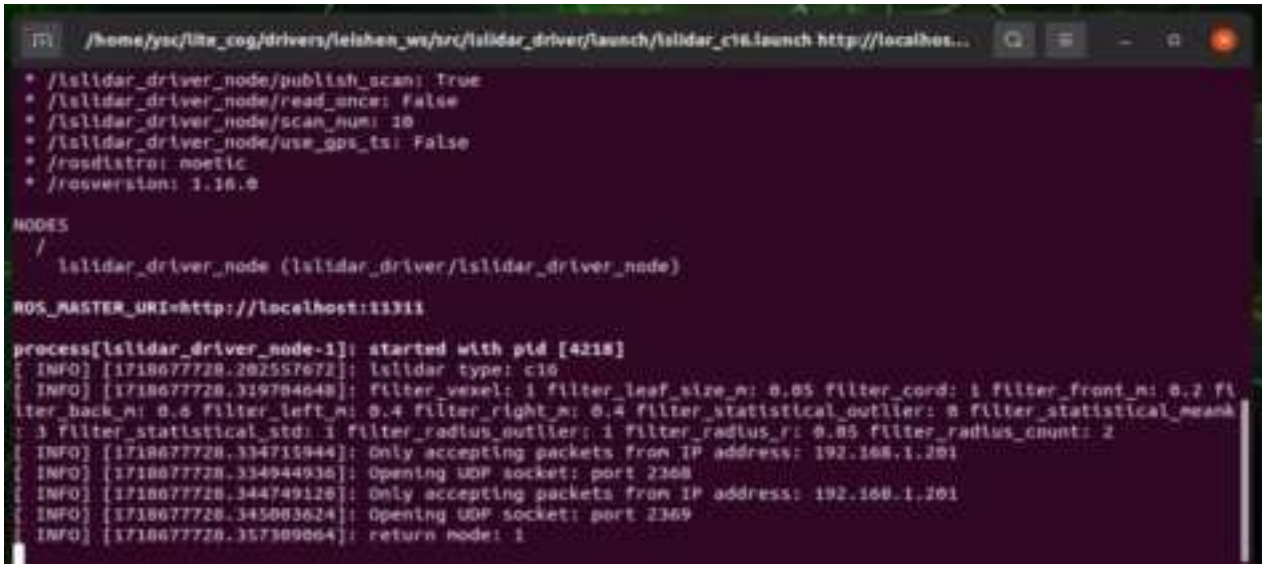
1. Open the Terminal and enter the following to start the LiDAR driver (Choose one script of the two):

```
cd /home/ysc/lite_cog/system/scripts/lidar
./start_lslidar.sh    #Leishen Lidar
./start_livox.sh      #livox lidar
```

If the LiDAR driver node fails to start, check whether the LiDAR has connected to the perception host using the following command:

```
ping 192.168.1.201
```

> [Caution] This terminal should be kept running during mapping.



2. Start the mapping program:

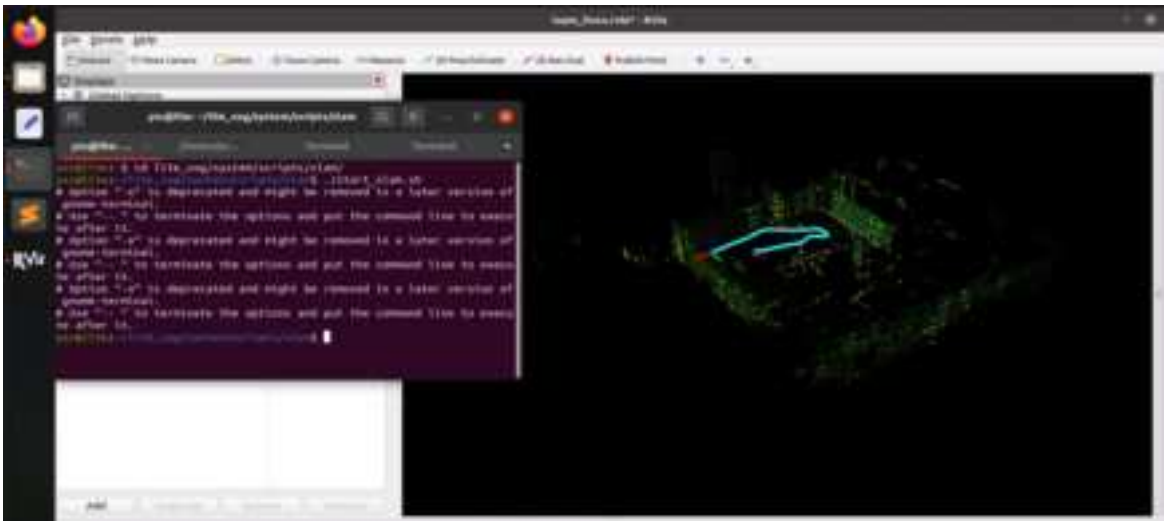
a) The script ***start_slam.sh***, which starts the mapping program, is in the path /home/ysc/lite_cog/system/scripts/slam and reads as follows:

```
#!/bin/sh

# Open the mapping program
gnome-terminal -x bash -c "source /home/ysc/lite_cog/slam/devel/setup.bash;
roslaunch faster_lio mapping_c16.launch; read -p 'Press any key to exit...'"

# open a terminal used for creating grid_map
gnome-terminal -x bash -c "bash /home/ysc/lite_cog/system/scripts/slam/gridmap.sh;
read -p 'Press any key to exit...'"

# open a terminal used for saving grid map
gnome-terminal -x bash -c "bash
/home/ysc/lite_cog/system/scripts/slam/save_map.sh; read -p 'Press any key to
exit...'"
```

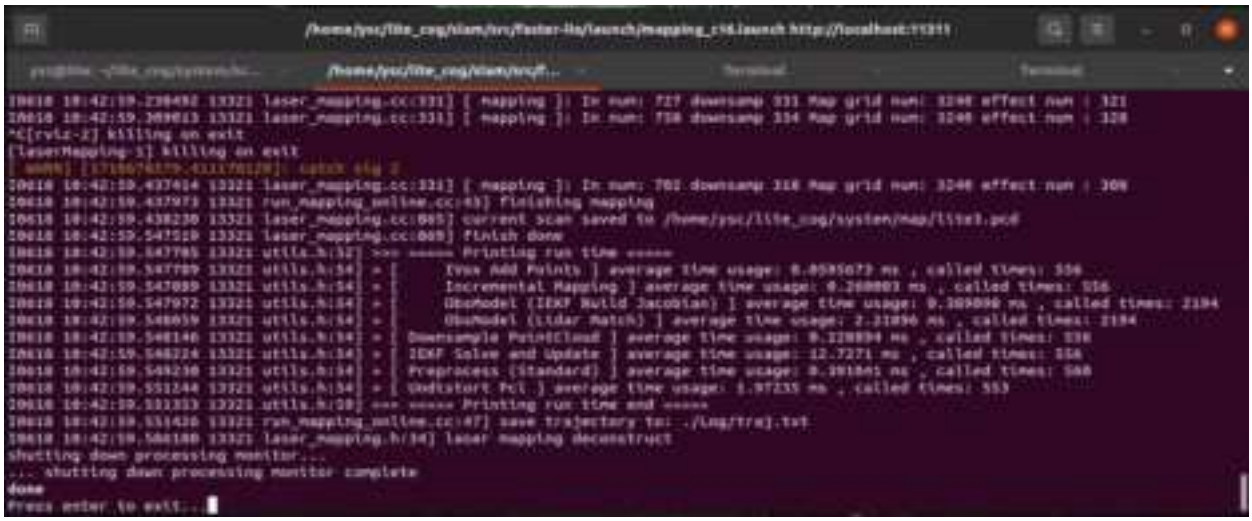
b) After logging into the perception host desktop using NoMachine and remotely controlling the robot to stand up, open a Terminal and enter the following command to start the mapping program using the script:

```
cd /home/ysc/lite_cog/system/scripts/slam
./start_slam.sh
```

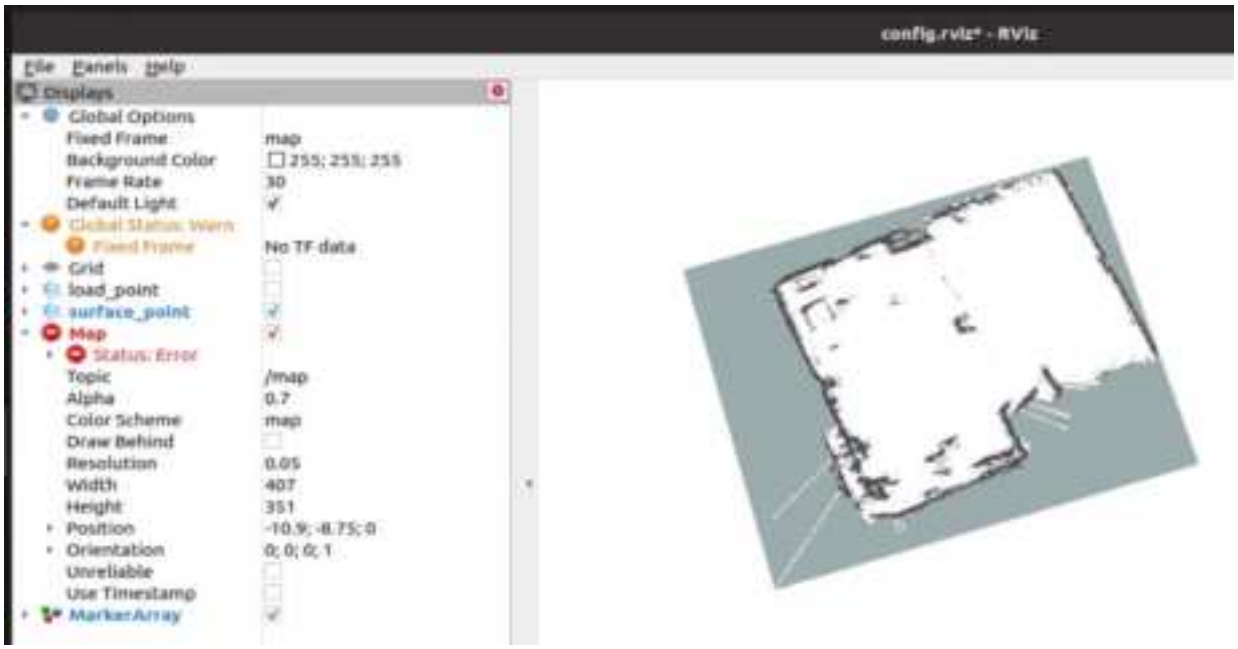
c) After executing the above command, the visualization tool RViz will be launched, and three terminal tabs will be generated in the terminal running the script start_slam.sh, respectively, to run ***fast-lio***, generate grid map and save grid map.



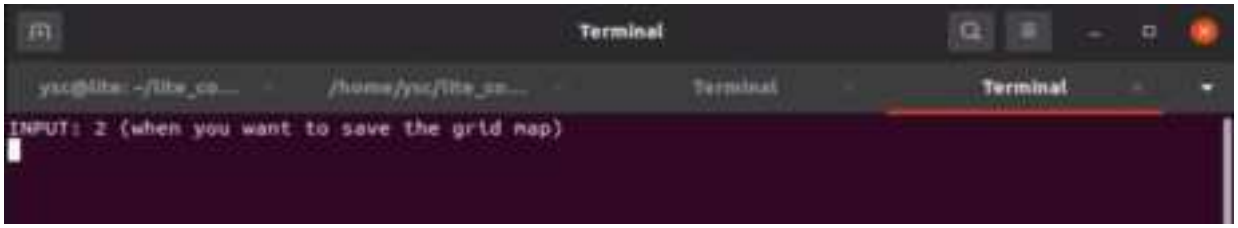
3. Operate the robot and guide it around the designated area to construct the map. When taking turns, please slow down. Also, be mindful of LiDAR's blind spots and keep the robot at a minimum distance of 0.5 meters from any walls.
4. After finishing scanning the designated area, check whether the point cloud map matches the real environment in RViz.
5. Find the corresponding tab page of faster-lio program after completing the map scanning, press "Ctrl+C" to stop mapping, the program will automatically save the 3D point cloud file (.pcd) to the ~/lite_cog/system/map directory, and display the average processing time (for reference only). Press Enter to close this tab page.



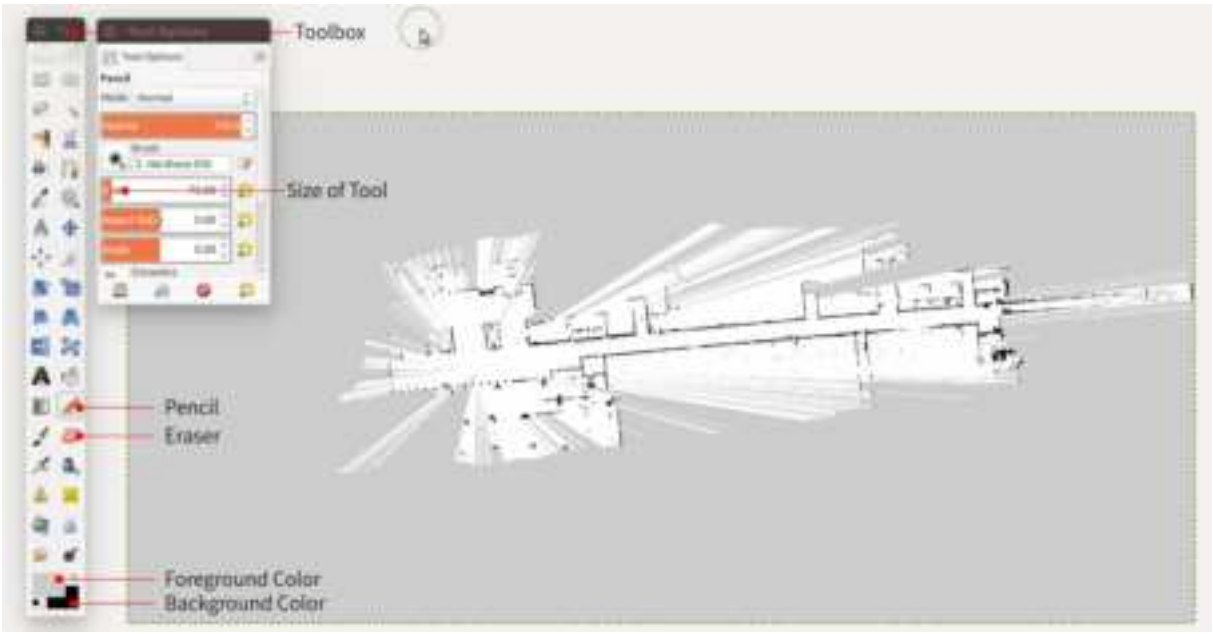
6. After saving the 3D point cloud file (.pcd) successfully, find the tab page as shown in the following figure, enter 1 and press Enter key. After a while, ***pcd_2_gridmap*** package will be called to convert the point cloud map into a grid map, and the next step can be carried out when the RViz window pops up and displays the grid map.



7. To save the grid map, first, select the Terminal that displays "when you want to save the grid map". Next, enter the number 2 and press Enter to call map_server. After that, map files will be saved to /home/ysc/lite_cog/system/map, including the .yaml file, .pgm file, and .pcd file.



8. If the grid map (.pgm) is not completely in line with the actual environment or users need to manually delimit passable areas, GIMP Image Editor can be used to edit it. Open a Terminal and type `gimp` to open GIMP Image Editor and drag the grid map (.pgm) into it.



a) *Toolbox* can be opened by choosing [Windows] – [New Toolbox] in the top menu bar if it is not displayed.

b) *Foreground Color* specifies the color of *Pencil* and *Background Color* specifies the color of *Eraser*. In the grid map, the black area is not passable, the white area is passable and the gray area is unknown. Users can erase the noise or add a virtual wall with Pencil or Eraser.

c) Save the modified map by clicking [File] – [Overwrite usr_map.pgm] to cover the origin file and it is not necessary to save it again when closing the editor.

9. Please close all Terminals with ctrl+c after completing all operations to avoid affecting subsequent processes.

10. The map files will by default be saved in /home/ysc/lite_cog/system/map directory. If the path or name of map files is changed, please follow the steps below to ensure that localization and navigation program can call the map correctly:

a) Update the path of map file(.pgm) in the corresponding .yaml file, and modify the file name of .yaml file to keep same with the .pgm file:

```
image:/home/ysc/lite_cog/system/map.pgm            // path of map file (.pgm)
resolution:0.050000
...
```

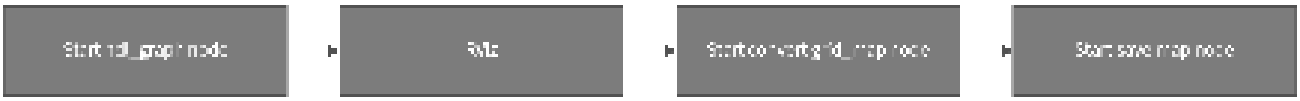
b) Go into the directory ~/lite_cog/nav/src/hdl_localization/launch and configure the name and path of map files in the file *local_rslidar_imu.launch*.

```
<arg name="map_name" default="lite3" />      //Define Map File Name
...
<node name="MapServer" pkg="map_server" type="map_server"
args="/home/ysc/lite_cog/system/map/$(arg map_name).yaml"/>

...
...
<param name="globalmap_pcd" value="/home/ysc/lite_cog/system/map/$(arg
map_name).pcd" />
...
```

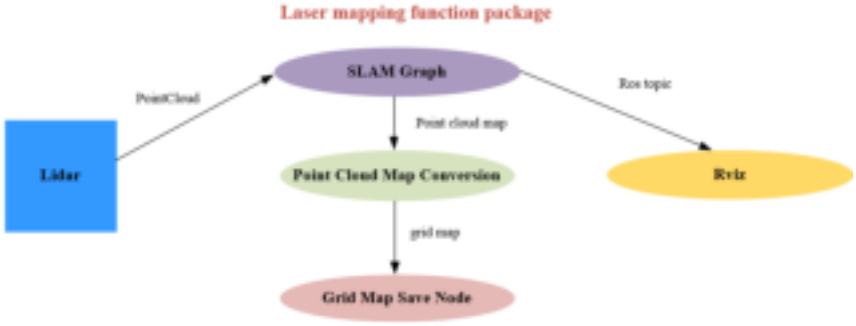
7.2 Mapping(for earlier than v3.1.04)

7.2.1 Introduction

This case uses 6DOF SLAM released on Github by Kenji Koide from Toyohashi University of Technology. The main operation process is shown below:



The corresponding data flow diagram is shown below:



7.2.2 Package Structure

```
/home/ysc/lite_cog/slam
├── build
├── devel
├── src
│   ├── CMakeLists.txt
│   ├── fast_gicp
│   ├── hdl_graph_slam
│   ├── map_server
│   └── ndt_omp
└── version
```

The *hdl_graph_slam* package builds the map.

7.2.3 Usage

[Caution] Before mapping, please check whether there is a previously created map in the /home/ysc/lite_cog/system/map folder. If so, you can move it to another folder to avoid overwriting.
[Caution] Mapping requires more computing resources, so please turn off all the AI options on the app first.

1. Open the Terminal and enter the following to start the LiDAR driver (Choose one script of the two):

```
cd /home/ysc/lite_cog/system/scripts/lidar
./start_lslidar.sh    #Leishen Lidar
./start_livox.sh      #Livox Lidar
```

If the LiDAR driver node fails to start, check whether the LiDAR has connected to the perception host using the following command:

```
ping 192.168.1.201   #Both lidar IPs are 192.168.1.201
```

2. Start the mapping program:

a) The script *start_slam.sh*, which starts the mapping program, is in the path /home/ysc/lite_cog/system/scripts/slam and reads as follows:

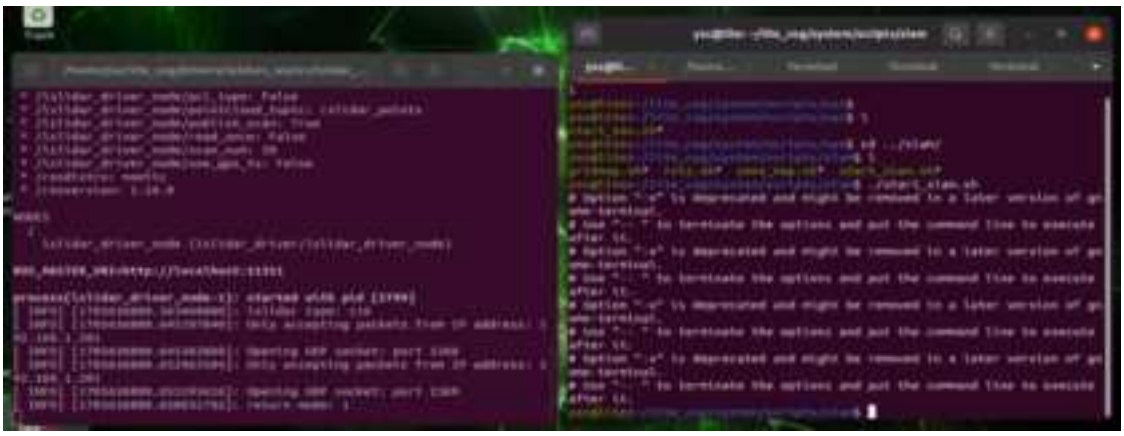
```
#!/bin/sh
```

```
2
3  # open rviz
4  gnome-terminal -x bash -c "cd /home/ysc/lite_cog/slam; source devel/setup.bash;
5  roslaunch hdl_graph_slam mapping_rslidar_indoor.launch;"
6
7  # open rviz
8  gnome-terminal -x bash -c "bash /home/ysc/lite_cog/system/scripts/slam/rviz.sh"
9
10 # open a terminal used for creating grid_map
11 gnome-terminal -x bash -c "bash /home/ysc/lite_cog/system/scripts/slam/gridmap.sh"
12
13 # open a terminal used for saving map
14 gnome-terminal -x bash -c "bash /home/ysc/lite_cog/system/scripts/slam/save_map.sh"
15
```

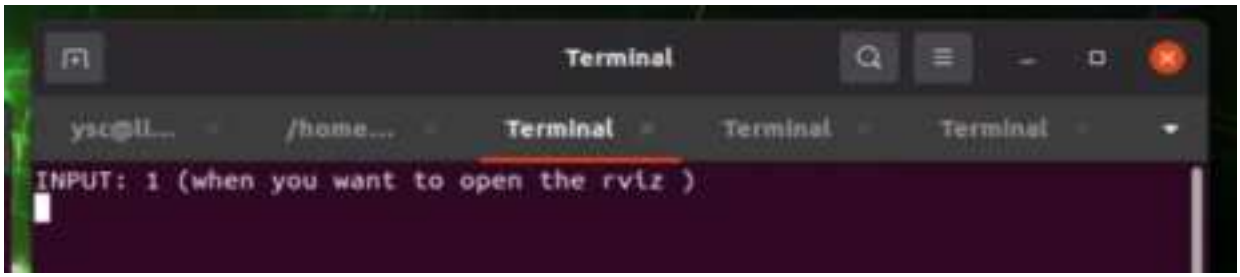
b) After logging into the perception host desktop using NoMachine and remotely controlling the robot to stand up, open a Terminal and enter the following command to start the mapping program using the script:

```
1  cd /home/ysc/lite_cog/system/scripts/slam
2  ./start_slam.sh
```

c) After executing the above command, five terminal Windows will be generated, respectively used for running the scripts, running the mapping program, opening Rviz, creating grid map, and saving map. (In the picture below, the window in left side is running the LiDAR driver window, the right side is running the mapping script) :

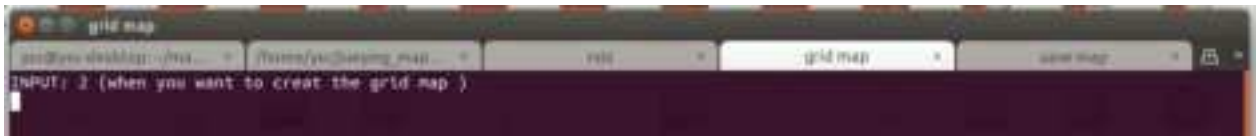


3. If you want to see the effect of mapping in real time, find the terminal used for opening RViz (as shown below), input the number 1 and press Enter, then the RViz visualization interface will be opened. Opening this interface will reduce the performance of mapping, and if you are not satisfied with the effect of mapping, please try not to open Rviz and close the NoMachine remote interface when mapping to save computing resources).

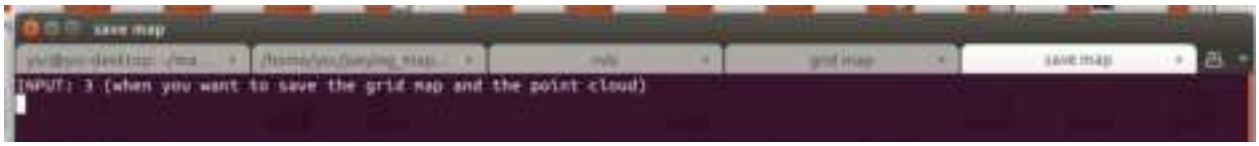


4. Operate the robot and guide it around the designated area to construct the map. When taking turns, please slow down. Also, be mindful of LiDAR's blind spots and keep the robot at a minimum distance of 0.5 meters from any walls.
5. After finishing scanning the designated area, check whether the point cloud map matches the real environment (if the RViz has not been opened before, open it at this time). If the area is large or there is a closed loop in the real environment (such as circling around a house), please check the map whether it is a closed loop consistent with the real environment. If it is not a closed loop, you can circle again to complete the loop-closure detection.
6. To convert the point cloud map into a grid map after completing the map scanning, first, select the Terminal that displays "when you want to create the grid map" as shown in the figure below. Then, enter the number 2 and press Enter to call

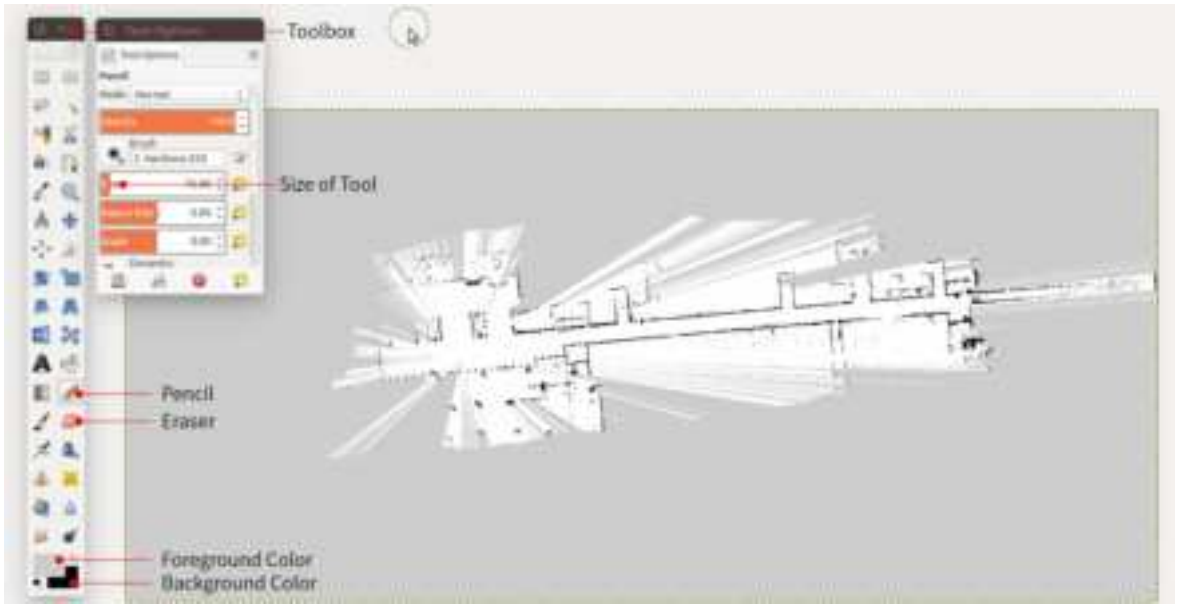
octomap. After this, remotely control the robot to walk a short distance. Doing so will convert the point cloud map into a grid map.



7. To save the grid map, first, select the Terminal that displays "when you want to save the grid map and the point cloud". Next, enter the number 3 and press Enter to call map_server. After that, remotely control the robot to walk a short distance. Map files will be saved to /home/ysc/lite_cog/system/map, including the .yaml file, .pgm file, and .pcd file.



8. If the grid map (.pgm) is not completely in line with the actual environment or users need to manually delimit passable areas, GIMP Image Editor can be used to edit it. Open a Terminal and type `gimp` to open it and drag the grid map (.pgm) into GIMP Image Editor.



a) *Toolbox* can be opened by choosing [Windows] – [New Toolbox] in the top menu bar if it is not displayed.

b) *Foreground Color* specifies the color of *Pencil* and *Background Color* specifies the color of *Eraser*. In the grid map, the black area is not passable, the white area is passable and the gray area is unknown. Users can erase the noise and add a virtual wall with Pencil or Eraser.

c) Save the modified map by clicking [File] – [Overwrite usr_map.pgm] to cover the origin file and it is not necessary to save it again when closing the editor.

9. Please close all Terminals with ctrl+c after completing all operations to avoid affecting subsequent processes.

10. The map files will by default be saved in /home/ysc/lite_cog/system/map. If the path or name of map files is changed, please follow the steps below to ensure that localization and navigation program can call the map correctly:

a) Update the path of map file(.pgm) in the corresponding .yaml file, and modify the file name of .yaml file to keep same with the .pgm file:

```
image:/home/ysc/lite_cog/system/map.pgm            // path of map file (.pgm)
resolution:0.050000
...
```

b) Go into the directory ~/lite_cog/nav/src/hdl_localization/launch and configure the name and path of map files in the file *local_rslidar_imu.launch*.

```
<arg name="map_name" default="lite3" />      //Define Map File Name
...
<node name="MapServer" pkg="map_server" type="map_server"
args="/home/ysc/lite_cog/system/map/$(arg map_name).yaml"/>

...
...
```

```
<param name="globalmap_pcd" value="/home/ysc/lite_cog/system/map/$(arg
map_name).pcd" />
...
```

7.3 Localization & Navigation

This case is based on LiDAR and IMU to implement localization and navigation. The localization algorithm used in this case is hdl_localization algorithm.

> [Caution] The LIDAR driver needs to be running during localization and navigation (refer to 7.1.2).

7.3.1 Usage of Point-to-point Navigation

1. Open a Terminal and enter the following codes to run the LiDAR driver(If the driver has already been started, there is no need to start it again):

```
cd /home/ysc/lite_cog/system/scripts/lidar
./start_lslidar.sh    #Leishen LiDAR
./start_livox.sh      #Livox LiDAR
```

2. Open a terminal and enter the following command to start the node.

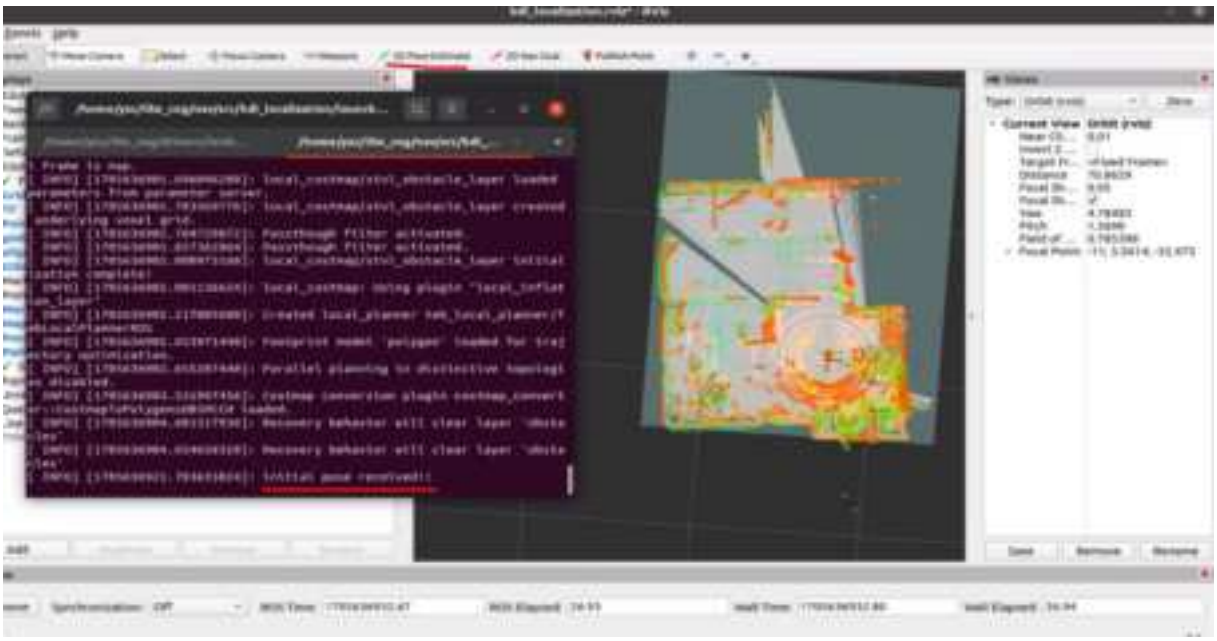
```
cd /home/ysc/lite_cog/system/scripts/nav
./start_nav.sh
```

3. After RViz is opened, initialize the robot location:
   a) Click the "2D Pose Estimate" button in the top toolbar;
   b) According to the actual location and orientation of the robot, press the mouse left button and drag to pull out an arrow at the corresponding location on the grid map;
   c) If the positioning initialization is successful, the point cloud and grid map will coincide, and the terminal print "initial pose received!!" ;

d) If the laser point cloud does not coincide with the grid map, the initial position is not correct, please re-operate;

e) If the point cloud does not appear on the map and the terminal prints "globalmap has not been received!", please close the program with ctrl+c and try again.

f) The base_link coordinate system is the robot coordinate system, and the x-axis (red) indicates the robot orientation.



[RViz Usage Tips] To manipulate the map on RViz, you can zoom in and out by using the mouse wheel. For rotation, you can drag the left mouse button. Meanwhile, to pan and drag the map, you need to hold Shift key and drag the left mouse button.

4. After initializing the location, a target point can be given according to a similar method:

a) Click the "2D Nav Goal" button in the top toolbar.

b) Press the left mouse button and drag on the grid map to specify a navigation goal and its orientation.

c) If successfully specifying a navigation goal, the planned path will be computed and shown. Else, start again from the first step.

5. Open the auto mode on the app and make the robot stand up, the robot will navigate along the route computed by the global planning, while using local planning to avoid dynamic obstacles, until it successfully arrives at the destination.

[Caution] To avoid the robot body being classified as an obstacle, only items that exceed a certain height will be identified as obstacles.

7.3.2 Usage of Multi-point Navigation

This case also provides the function to make the robot autonomously arrive at a series of waypoints in order.

[Caution] Before recording a new route, please check if there are any waypoint files saved before in the/home/ysc/lite_cog/pipeline/src/pipeline/data folder, and move them to other folders, to avoid the overwriting.

1. Refer to steps 1 to 3 in 7.2.1 to start the navigation program and initialize the localization, then open a terminal and run the following command to start the pipeline which used for recording a route (consists of many waypoints in sequence):

```
cd /home/ysc/lite_cog/pipeline/src/pipeline_tracking/tools
python3 location_record.py      # for opening a Chinese interface(left window below)
python3 location_record_en.py    # for opening an English interface(right window below)
```

2. First the robot should be controlled to arrive at the first waypoint and stand still. After the point cloud shown in the RViz stops moving, input 1 in the textbox of [location number]. Then click [get location] and the location and orientation information of the robot will be printed. Then click [record location] and a record file named 1.json will appear in /home/ysc/lite_cog/pipeline/src/pipeline/data. Then remote control the robot to the next waypoint, repeat the above operation until all the waypoints are recorded, and then close the window. If it is closed accidentally during recording, just open it again (referring to the first step).
3. Open a terminal, run the following command, and turn on auto mode on the app. The robot will go to the nearest waypoint and navigate in a loop according to the location number.

```
cd /home/ysc/lite_cog/pipeline
source devel/setup.bash
cd /home/ysc/lite_cog/pipeline/src/pipeline_tracking/scripts
python3 Task.py
```

4. Once the previous operation is finished, you can simply start the navigation program and initialize the localization referring to 7.2.1, and execute step 3 to make the robot follow the previously recorded waypoints for circular navigation when using it again.

7.4 Development

7.4.1 LiDAR Drivers Development

LiDAR drivers are located in the ~/lite_cog/driver/leishen_ws/ or ~/lite_cog/driver/mid360_ws/. Users can read the source code for development.

7.4.2 Development of SLAM Mapping

The SLAM mapping program is launched through the mapping_rslidar_indoor.launch file in the /home/ysc/lite_cog/slam/src/hdl_graph_slam/launch folder, which mainly contains point cloud preprocessing parameters, point cloud registration algorithm parameters, and g2o graph optimization algorithm parameters, which can be adjusted by the user.

7.4.3 Development of Localization and Navigation

1. Development of Localization: The localization program is started from the local_rslidar_imu.launch file in the ~/lite_cog/nav/src/hdl_localization/launch folder, which mainly contains the point cloud preprocessing parameters and point cloud registration parameters, which can be adjusted by the user.
2. Development of Navigation: In ~/lite_cog/nav/src/navigation/config directory, there are 6 .yaml files that can be adjusted.
    a) common_costmap_params.yaml: contains the parameters the local cost map and the global cost map used in common;
    b) global_costmap_params.yaml: contains the parameters of the global cost map;
    c) local_costmap_params.yaml: contains the parameters of the local cost map;
    d) global_planner_params.yaml: contains the parameters of the global planner;
    e) teb_ local_planner_params.yaml: contains the parameters of the teb local planner;
    f) move_base_params.yaml: contains the parameters of the move_base package.

Users can find the relevant description and source codes of some packages:

global_planner、teb_local_planner、costmap_2d、move_base.

8 LiDAR-based SLAM and Navigation(ROS2)

[Caution] This section applies to ROS2, so please refer to “3 Check and Change ROS version” to check the ROS version currently in use.

This case uses LiDAR and imu to achieve mapping (indoor and outdoor scenes), localization, navigation, and obstacle avoidance on perception host. The robot can achieve real-time localization and online 3D mapping. When localizing in a map, by fusing IMU, it will not lose its location due to falling or high-speed rotation. Map-based navigation is achieved using the ***bt_navigator*** package.

8.1 Mapping

8.1.1 Introduction

This case uses SLAM Mapping released on Github by Dr. Gao Xiang's team. The main operation process and data flow diagram are shown as below:

8.1.2 Package Structure

```
ysc@lite:~/lite_cog_ros2/slam/src$ tree .
.
├── faster-lio
├── map_server
└── pcd2grid
```

The ***faster-lio*** package is responsible for building PCD maps, the ***pcd2grid*** package is responsible for converting PCD maps into grid maps and publishing them, and the ***map_server*** package is responsible for saving grid maps.

8.1.3 Usage

[Caution] Before mapping, please check whether there is a previously created map in the /home/ysc/lite_cog_ros2/system/map folder. If so, you can move it to another folder to avoid overwriting.
[Caution] Mapping requires more computing resources, so please turn off all the AI options on the app first.

1. Open the Terminal and enter the following commands to start the LiDAR driver (Choose one scripts of the two):

```
cd ~/lite_cog_ros2/system/scripts/lidar
./start_lslidar.sh    #Leishen Lidar
./start_livox.sh      #Livox Lidar
```

If the LiDAR driver node fails to start, check whether the LiDAR has connected to the perception host using the following command:

```
ping 192.168.1.201   #Both lidar IPs are 192.168.1.201
```

2. Start the point cloud mapping program:

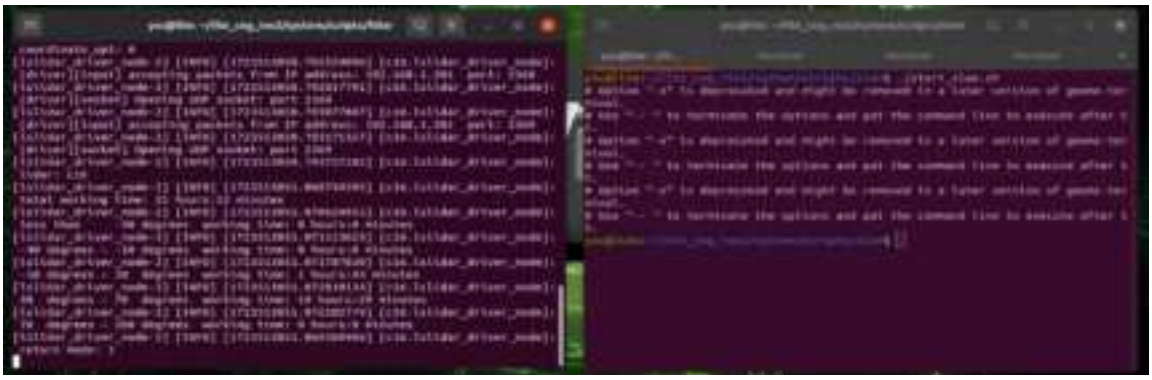
a) The script **start_slam.sh** to start the mapping program is located in the ~/lite_cog_ros2/system/scripts/slam directory and reads as follows:

```
#!/bin/sh

# start rviz
gnome-terminal -x bash -c "source /home/ysc/lite_cog_ros2/slam/install/setup.bash; ros2 launch faster_lio mapping_c16.launch.py; read -p 'Press Enter to exit...'"

#start a terminal for generating grid_map
gnome-terminal -x bash -c "bash /home/ysc/lite_cog_ros2/system/scripts/slam/gridmap.sh; read -p 'Press Enter to exit...'"

# Open a terminal for Saving Map
gnome-terminal -x bash -c "bash /home/ysc/lite_cog_ros2/system/scripts/slam/save_map.sh; read -p 'Press any key to exit...'; read -p 'Press Enter to exit...'"
```

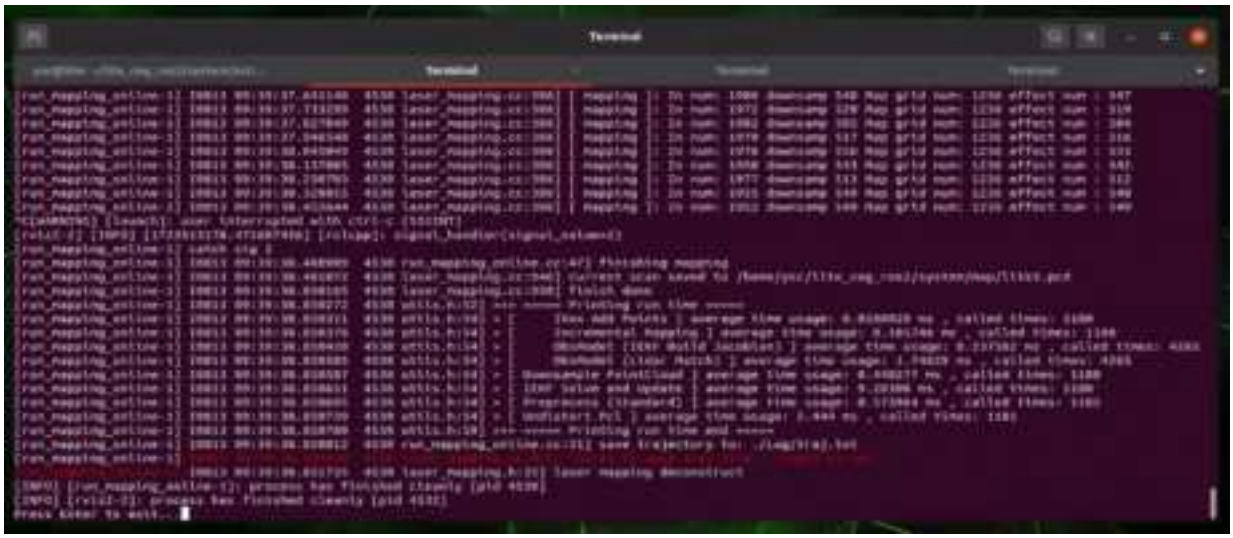
b) After logging into the perception host desktop using NoMachine and remotely controlling the robot to stand up, open a Terminal and enter the following command to start the mapping program using the script:

```
cd ~/lite_cog_ros2/system/scripts/slam
./start_slam.sh
```

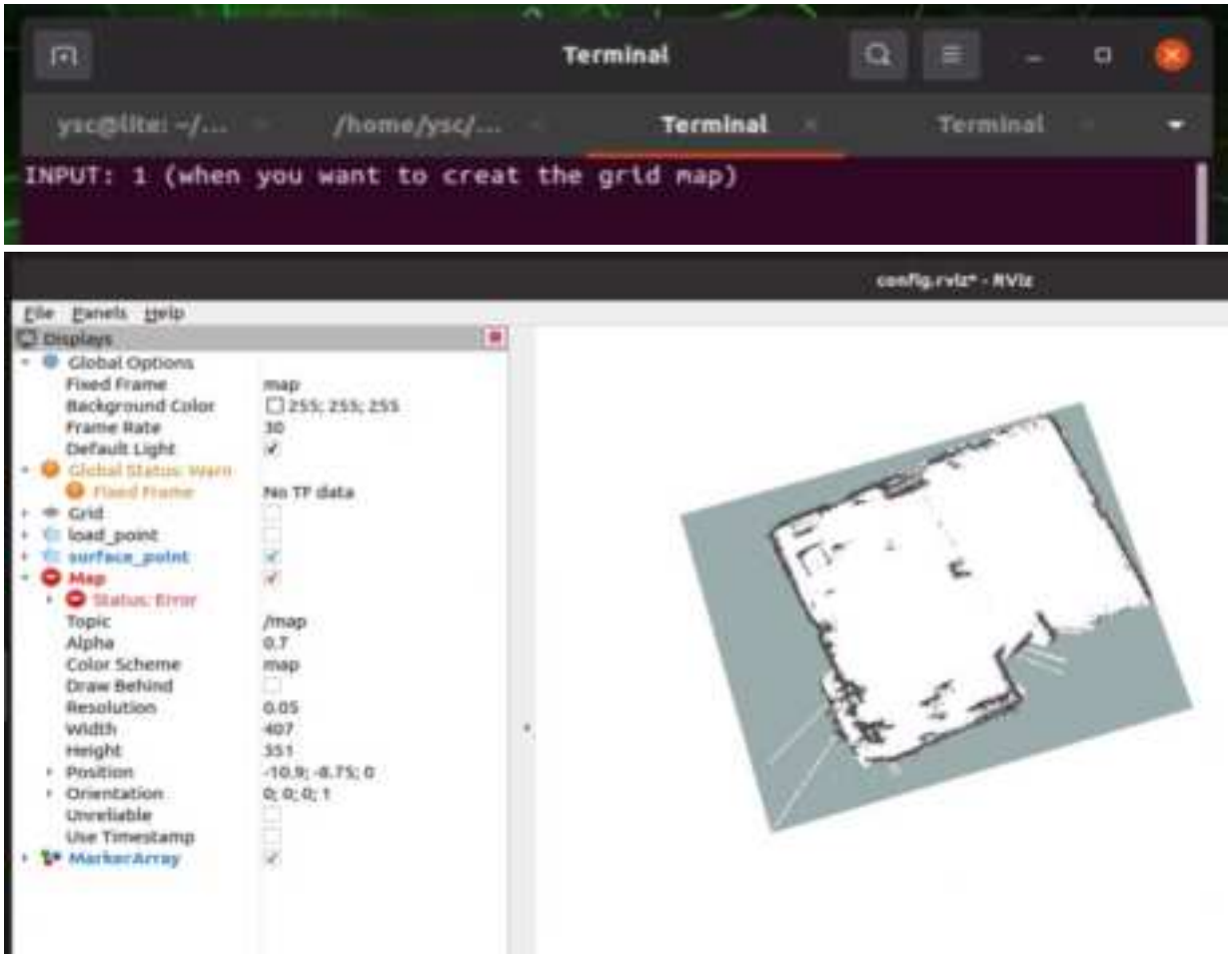
c) After executing the above command, four terminal windows will be generated, respectively for running the script of starting mapping, running the mapping program, generating the grid map, and saving the grid map (the left window below is running the LiDAR driver and the right is running the script of starting mapping):



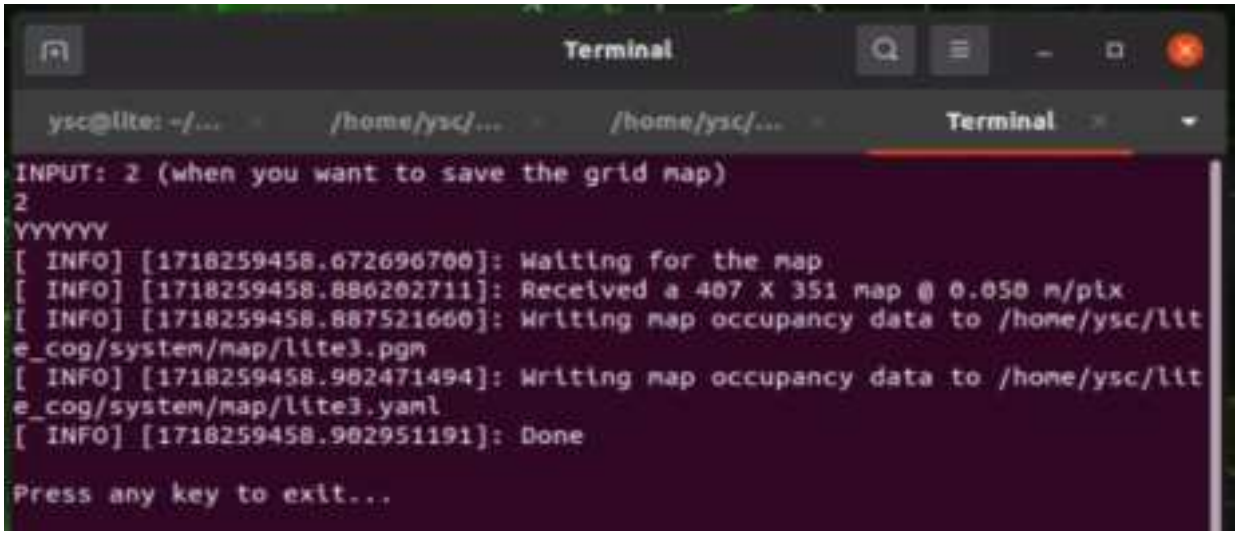
3. Operate the robot and guide it around the designated area to construct the map. When taking turns, please slow down. Also, be mindful of LiDAR's blind spots and keep the robot at a minimum distance of 0.5 meters from any walls.
4. After finishing scanning the designated area, check whether the point cloud map matches the real environment.
5. Find the corresponding tab page of faster-lio program after completing the map scanning, press "Ctrl+C" to stop mapping, the program will automatically save the 3D point cloud file (.pcd) to the ~/lite_cog_ros2/system/map directory, and display the average processing time (for reference only). Press Enter to close this tab page.



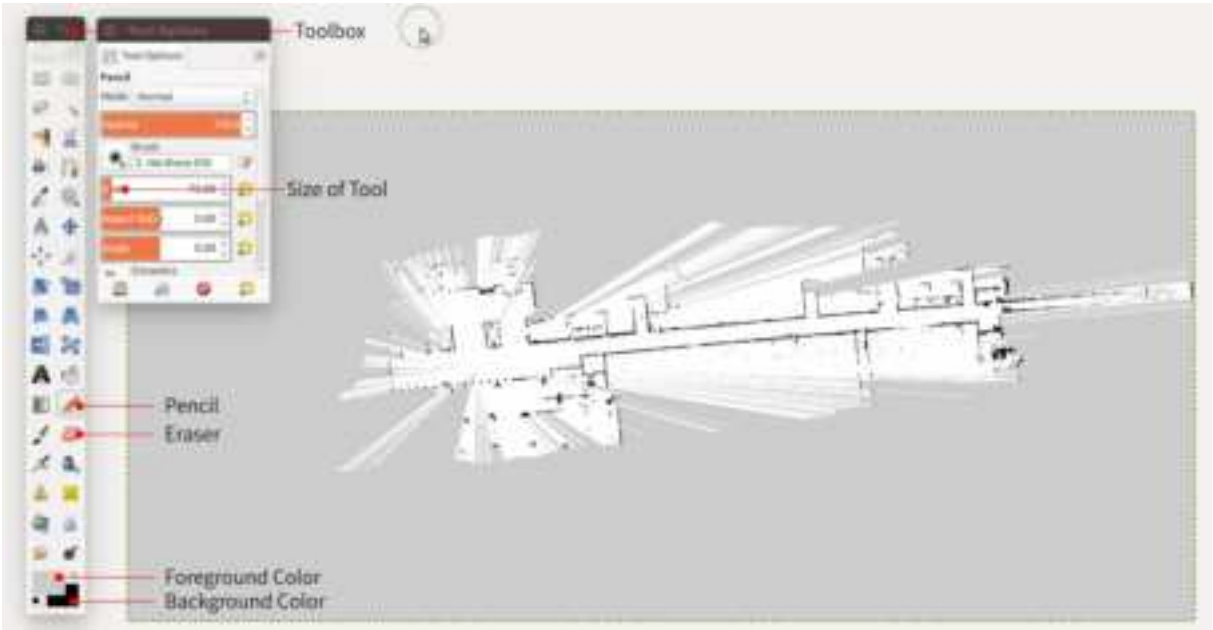
6. After saving the 3D point cloud file (.pcd) successfully, find the tab page as shown in the following figure, enter 1 and press Enter key. After a while, **pcd_2_gridmap** package will be called to convert the point cloud map into a grid map, and the next step can be carried out when the RViz window pops up and displays the grid map.



7. After the point cloud map is converted into a grid map, select the Terminal that displays "when you want to save the grid map". Next, enter the number 2 and press Enter to call map_server. After that, map files, including .yaml file, .pgm file and .pcd file, will be saved to ~/lite_cog_ros2/system directory.



8. If the grid map (.pgm) is not completely in line with the actual environment or users need to manually delimit passable areas, GIMP Image Editor can be used to edit it. Open a Terminal and type `gimp` to open GIMP Image Editor and drag the grid map (.pgm) into it.



a) *Toolbox* can be opened by choosing [Windows] – [New Toolbox] in the top menu bar if it is not displayed.

b) *Foreground Color* specifies the color of *Pencil* and *Background Color* specifies the color of *Eraser*. In the grid map, the black area is not passable, the white area is passable and the gray area is unknown. Users can erase the noise or add a virtual wall with Pencil or Eraser.

c) Save the modified map by clicking [File] – [Overwrite usr_map.pgm] to cover the origin file and it is not necessary to save it again when closing the editor.

9. Please close all Terminals with ctrl+c after completing all operations to avoid affecting subsequent processes.

10. The map files will by default be saved in /home/ysc/lite_cog_ros2/system/map directory. If the path or name of map files is changed, please follow the steps below to ensure that localization and navigation program can call the map correctly:

a) Update the path of map file(.pgm) in the corresponding .yaml file, and modify the file name of .yaml file to keep same with the .pgm file:

```
image:/home/ysc/lite_cog_ros2/system/map.pgm            // path of map file (.pgm)
resolution:0.050000
...
```

b) Go into the directory ~/lite_cog_ros2/nav/src/hdl_localization/launch and configure the name and path of map files in the file *lite_localization.launch.py*.

```
declare_map_server_config_file_cmd = DeclareLaunchArgument(
    'map_server_config_file',
    default_value=os.path.join(
        '/home/ysc/lite_cog_ros2/system/map/lite3.yaml'
    )
)

```

```
...

parameters=[
    {"globalmap_pcd": "/home/ysc/lite_cog_ros2/system/map/lite3.pcd"},

...
```

8.2 Localization & Navigation

This case is based on LiDAR and IMU to implement localization and navigation. The localization algorithm used in this case is hdl_localization algorithm.

[Caution] The LIDAR driver needs to be running during localization and navigation (refer to 8.1.3).

8.2.1 Usage of Point-to-point Navigation

1. Open a Terminal and enter the following codes to start LiDAR driver (If the lidar driver has already been started when mapping, there is no need to start it again):

```
cd ~/lite_cog_ros2/system/scripts/lidar
./start_lslidar.sh    #Leishen Lidar
./start_livox.sh      #Livox Lidar
```

2. Open a terminal and enter the following commands to start the node of navigation:

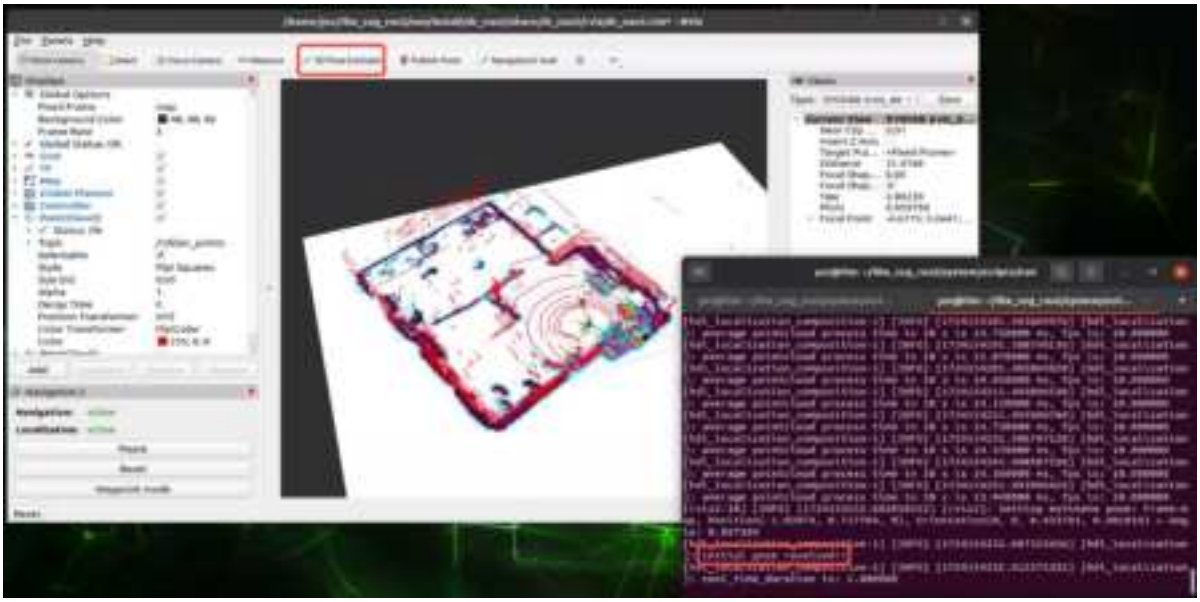
```
cd /home/ysc/lite_cog_ros2/system/scripts/nav
./start_nav.sh
```

3. After RViz is opened, initialize the robot location:
   a) Click the "2D Pose Estimate" button in the top toolbar;
   b) According to the actual location and orientation of the robot, press the mouse left button and drag to pull out an arrow at the corresponding location on the grid map;
   c) If the positioning initialization is successful, the point cloud and grid map will coincide, and the terminal print "initial pose received!!";

d) If the laser point cloud does not coincide with the grid map, the initial position is not correct, please re-operate;

e) If the point cloud does not appear on the map and the terminal prints "globalmap has not been received!", please close the program with ctrl+c and try again.

f) The base_link coordinate system is the robot coordinate system, and the x-axis (red) indicates the robot orientation.



[RViz Usage Tips] To manipulate the map on RViz, you can zoom in and out by using the mouse wheel. For rotation, you can drag the left mouse button. Meanwhile, to pan and drag the map, you need to hold Shift key and drag the left mouse button.

4. After initializing the location, a target point can be given according to a similar method:

a) Click the "Navigation2 Goal" button in the top toolbar.

b) Press the left mouse button and drag on the grid map to specify a navigation goal and its orientation.

c) If successfully specifying a navigation goal, the planned path will be computed and shown. Else, start again from the first step.

5. Open the auto mode on the app and make the robot stand up, the robot will navigate along the route computed by the global planning, while using local planning to avoid dynamic obstacles, until it successfully arrives at the destination.

[Caution] To avoid the robot body being classified as an obstacle, only items that exceed a certain height will be identified as obstacles.

8.2.2 Usage of Multi-point Navigation

This case also provides the function to make the robot autonomously arrive at a series of waypoints in order.

[Caution] Before recording a new route, please check if there are any waypoint files saved before in the/home/ysc/lite_cog_ros2/pipeline/src/pipeline/data folder, and move them to other folders, to avoid the overwriting.

1. Refer to steps 1 to 3 in 8.2.1 to start the navigation program and initialize the localization, then open a terminal and run the following commands to start the pipeline which used for recording a route:

```
cd ~/lite_cog_ros2/pipeline/src/pipeline
python3 LocationRecorder.py      # for opening a Chinese interface(left window below)
python3 LocationRecorder_EN.py    # for opening an English interface(right window below)
```

2. First the robot should be controlled to arrive at the first waypoint and stand still. After the point cloud shown in the RViz stops moving, input 1 in the textbox of [location number]. Then click [get location] and the location and orientation information of the robot will be printed. Then click [record location] and a record file named ***1.json*** will appear in /home/ysc/lite_cog_ros2/pipeline/src /data. Then remotely control the robot to the next waypoint, repeat the above operations until all the waypoints are recorded, and then close the window. If it is closed accidentally during recording, just open it again (referring to the first step).
3. Open a terminal, run the following commands, and turn on auto mode on the app. The robot will go to the nearest waypoint and navigate in a loop according to the location number.

```
cd /home/ysc/lite_cog_ros2/pipeline/src/pipeline
python3 Task.py
```

4. Thereafter, when used again, only steps 1 and 3 need to be performed to make the robot navigate in a loop according to the previously recorded target points.

8.3 Development

8.3.1 LiDAR Drivers Development

LiDAR drivers are located in the ~/lite_cog_ros2/driver/leishen_ws and ~/lite_cog_ros2/driver/mid360_ws. Users can read the source code for secondary development.

8.3.2 Development of SLAM Mapping

The SLAM mapping program is launched through the file **hdl_graph_slam_launch.py** in the /home/ysc/lite_cog_ros2/slam/src/hdl_graph_slam/launch folder, which mainly includes point cloud preprocessing parameters, point cloud registration algorithm parameters, and g2o graph optimization algorithm parameters, which can be adjusted by the user.

8.3.3 Development of Positioning and Navigation

1. **Development of Positioning:** The localization program is started from the file lite_localization.launch.py in the ~/lite_cog_ros2/nav/src/hdl_localization/launch folder, which mainly contains the point cloud preprocessing parameters and point cloud registration parameters. Users can modify them.
2. **Development of Navigation:** Users can adjust the parameters in the lite_nav2.yaml in the ~/lite_cog_ros2/nav/src/dr_nav2/config directory, and the functions of each parameter can be found in the navigation2.

Generation ROBOTS

Brand of NGX ROBOTICS

+33 (0)5 56 39 37 05

contact@generationrobots.com

1 rue Pierre-Georges Latécoère 33700 Mérignac, France

www.generationrobots.com