3D Laser SLAM Development Guide

[toc]

Introduction

3D laser SLAM algorithm

This program uses the `LIO-SAM` algorithm as the LIDAR SLAM algorithm, which closely couples the LIDAR data and the `IMU` data fed by the robot dog itself to achieve simultaneous localization and map building. For more information about the `LIO-SAM` algorithm and development based on it, please refer to the introduction in original paper and Github repository.

3D map-based positioning

The patrol will load the map built during the build, and use the `NDT` algorithm for 3D point cloud matching. This algorithm needs to provide the approximate initial position information (all values are 0 by default, so the initial position of the robot dog when starting the patrol needs to be the same as the initial position when starting the build), and the `LIO-SAM` algorithm will not be run during the patrol.

Path planning and obstacle avoidance

The ROS `navigation` package is used for route planning and obstacle avoidance, and `teb_local_planner` is used as the local route planner.

Also `gmapping` is used to build 2D global maps for global path planning (LIO-SAM also generates global maps and is used for positioning during patrols, but it is not used for global path planning in environments with height differences).

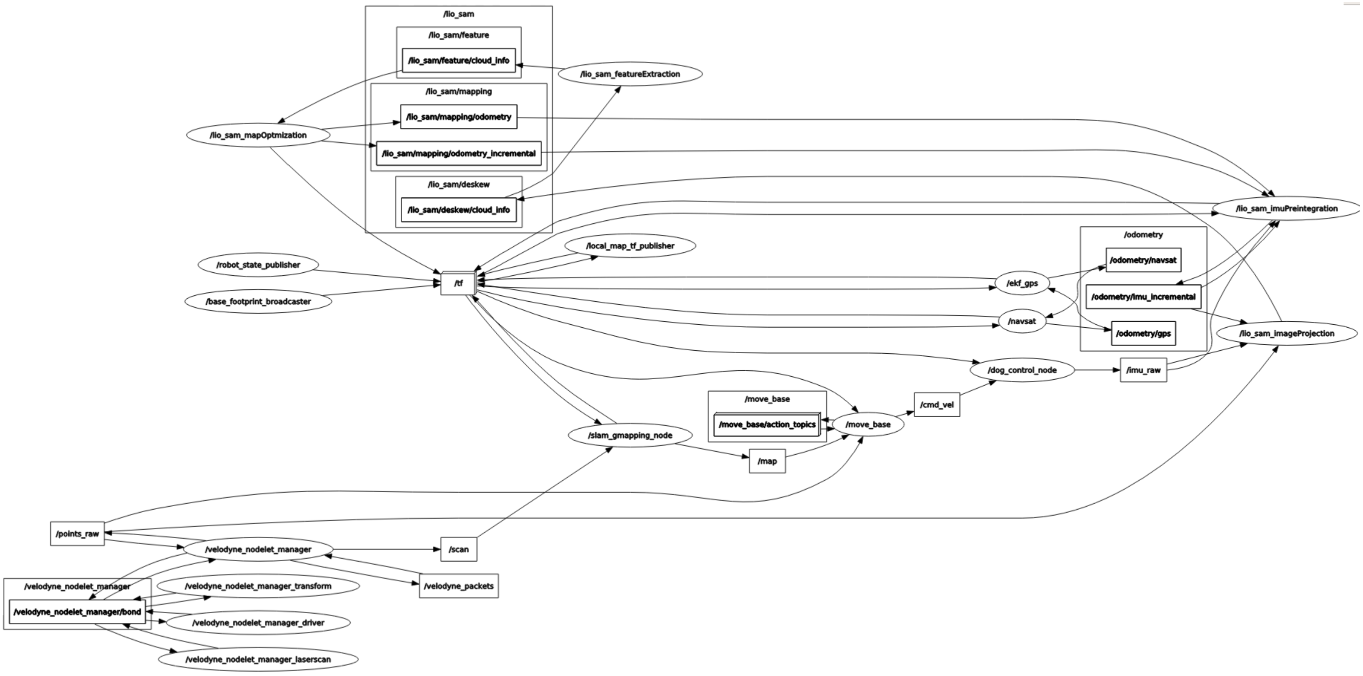
Interpretation of each software package

Package	Function
gmapping	Build a 2D global map for global path planning. Unlike the source program, the source code is modified and the odom of gmapping uses the odom of lio-sam.
lio_sam	SLAM algorithm, parameters are modified under config/params.yaml, source code is modified.
navigation	The launch file of move_base is called and the parameters are configured; the robot's motion performance depends heavily on the configuration here.
ndt_localization	Positioning algorithm during patrols.
start	Launch files for launching tasks, as well as applets that act as "glue", such as the release of patrol points.
a2_ros2udp	The node that communicates with the robot dog motion program SDK.
velodyne	Start the velodyne lidar driver.

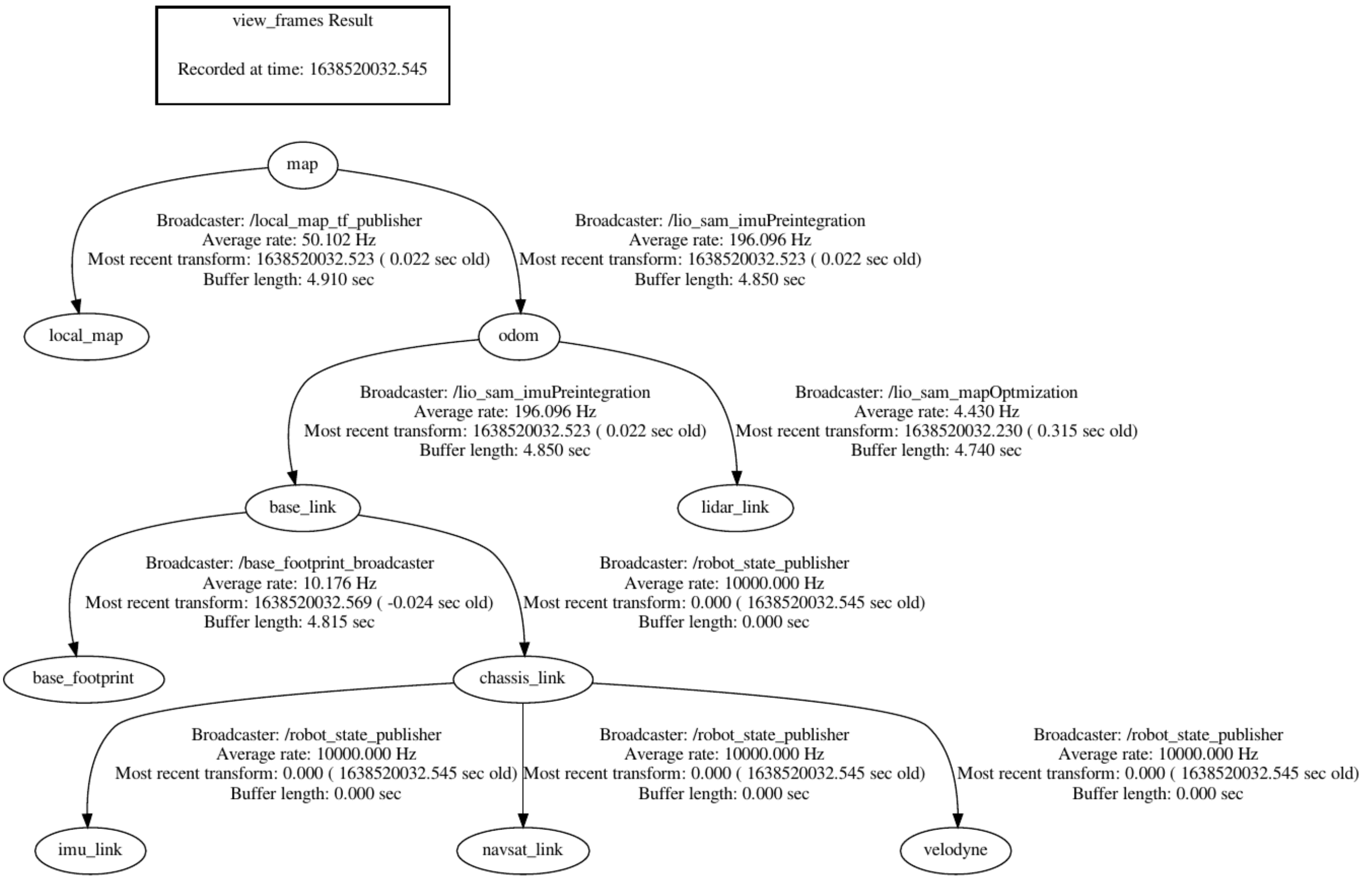
Package	Function
rslidar	Start the driver for robosense lidar.

Operational node diagram (using velodyne-16-line LIDAR)

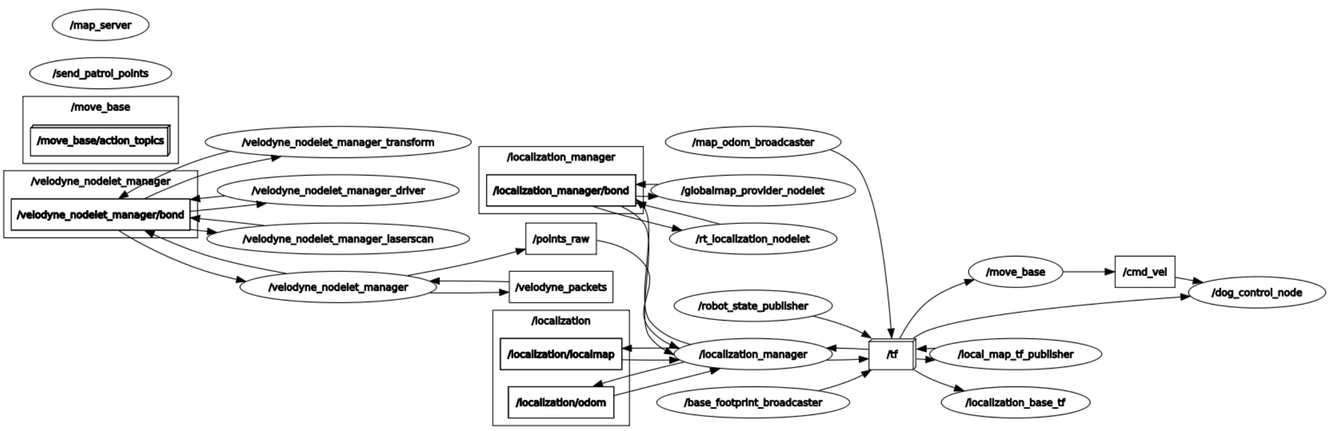
ROS node diagram at build time.



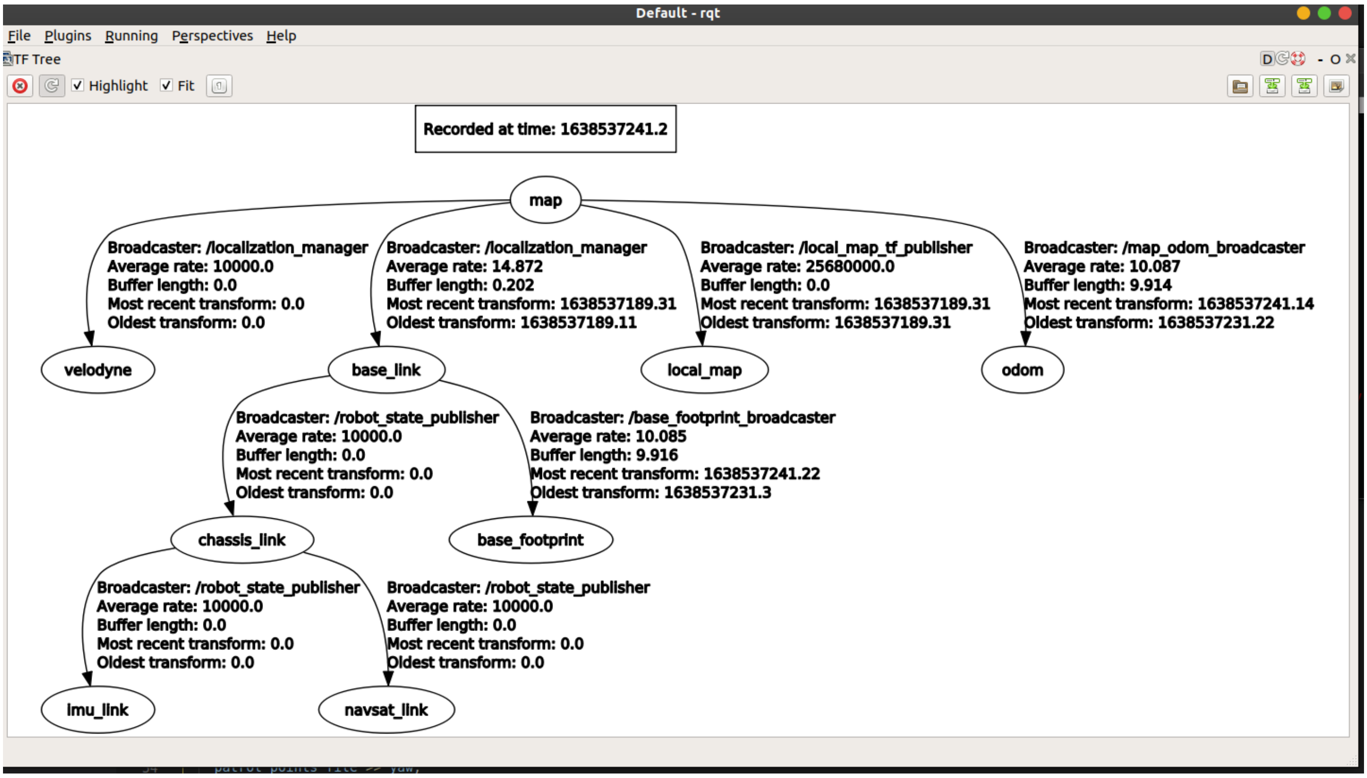
ROS TF tree at the time of graph building.



ROS node diagram during patrol.



ROS TF tree during patrol.



Platforms and Sensors

The robot platforms supported by this software for adaptation are

- `Unitree A1`
- `Unitree Go1`
- `Unitree Aliengo`

The sensors supported by this software for use are.

- Velodyne Formula's `Vledyne VLP-16` Lidar.
- Speedy Polytron's `RS-Lidar-16` LIDAR.

Dependencies (no user installation required by default)

Attention.

- On the shipped version of MachineDog, all dependencies are already configured by default and do not need to be configured by the user.

ROS

Official installation steps reference.

- http://wiki.ros.org/melodic/Installation/Ubuntu

Installation steps using domestic sources are as follows.

```
sudo sh -c '. /etc/lsb-release && echo "deb
http://mirrors.tuna.tsinghua.edu.cn/ros/ubuntu/ $DISTRIB_CODENAME main" >
/etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

sudo apt update

sudo apt install ros-melodic-desktop-full

echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc

source ~/.bashrc

sudo apt install python-rosdep python-rosinstall python-rosinstall-generator
python-wstool build-essential

sudo apt install python-rosdep

sudo rosdep init

rosdep update
```

gtsam-4.0.2

The official website of `GTSAM` is here:

- GTSAM

Note:

- here we need to install the specified version of `gtsam-4.0.2`, which is a prerequisite of LIOSAM.
- The newest version of gtsam, such as 4.1 will cause compiling problem `/usr/bin/ld: cannot find -lBoost::timer`.

Steps to install gtsam-4.0.2 are as follows:

```
wget -O gtsam.zip https://github.com/borglab/gtsam/archive/4.0.2.zip

unzip gtsam.zip

cd gtsam-4.0.2/

mkdir build && cd build

cmake -DGTSAM_BUILD_WITH_MARCH_NATIVE=OFF -DGTSAM_USE_SYSTEM_EIGEN=ON ..

make -j4

sudo make install
```

unitree_legged_sdk

Download address of the newest `unitree_legged_sdk`:

- https://github.com/unitreerobotics/unitree_legged_sdk/releases

Note that you should use the specific version of `unitree_legged_sdk` corresponding to the current dog model and the current `lidar_slam_3d` version.

Defaultly, the right version of `unitree_legged_sdk` is already configured in this project with corresponding dog model. So you don't have to configure it agian!

controller-manager

```
sudo apt install ros-melodic-controller-manager
```

libpcap-dev

```
sudo apt install libpcap-dev
```

openslam_gmapping

```
sudo apt install ros-melodic-openslam-gmapping
```

pcl_ros

```
sudo apt install ros-melodic-pcl-ros
```

tf_conversions

```
sudo apt install ros-melodic-tf-conversions
```

libmetis

```
sudo apt install libmetis-dev
```

robot_state_publisher

```
sudo apt install ros-melodic-robot-state-publisher
```

robot_localization

```
sudo apt install ros-melodic-robot-localization
```

teb_local_planner

```
sudo apt install ros-melodic-teb-local-planner
```

Compile and install (no user compilation required by default)

Increase swap space

The default memory of `miniPC` is only 4G, which is a bit small for compilation, so you need to increase swap space to get through the compilation quickly. The `swap` space will be deleted after every reboot, you can check it with `free -h`

Adding 16GB of swap space is done as follows.

```
sudo dd if=/dev/zero of=/swapfile bs=64M count=256 status=progress

sudo chmod 600 /swapfile

sudo mkswap /swapfile

sudo swapon /swapfile

free -h
```

Compilation

Place the folder `catkin_lidar_slam_3d` under the path `~/UnitreeSLAM`.

Depending on the current type of robot dog, select the SDK version of the corresponding motion program, refer to here.

- unitree_legged_sdk

For example, for the `v2.0.2` version of this program, on the `Go1` robot, you need to modify the following two lines of the `CMakeLists` file under the `a2_ros2udp` package.

```
### For Go1
include_directories(~/UnitreeSLAM/sdk/unitree_legged_sdk-v20220117/include)
link_directories(~/UnitreeSLAM/sdk/unitree_legged_sdk-v20220117/lib)
```

compile：

```
cd catkin_lidar_slam_3d

catkin_make
```

Possible compilation problems encountered

Error 1: Conflict of `PCL` and `OpenCV`

If the following error occurs at compile time, it may be due to a conflict between PCL and OpenCV.

```
error: field ‘param_k_’ has incomplete type ‘flann::SearchParams’
```

For the solution, refer to the following link.

- PCL-OpenCV conflict resolution
- https://github.com/strands-project/strands_3d_mapping/issues/67

Error 2: `cv_bridge`

Error:

```
CMake Error at /opt/ros/melodic/share/cv_bridge/cmake/cv_bridgeConfig.cmake:113
(message):
  Project 'cv_bridge' specifies '/usr/include/opencv' as an include dir,
  which is not found.  It does neither exist as an absolute directory nor in
  '${{prefix}}//usr/include/opencv'.
```

Solution:

- Because NVIDIA's 32.3.1.img file names the opencv file as opencv4
- So just modify the cv_bridgeconfig.cmke file in the above path, change `/usr/include/opencv` to `/usr/include/opencv4`.

Reference:

- https://blog.csdn.net/qq_34213260/article/details/106226837

Configuration (no user configuration required by default)

The robot dog is shipped with the correct parameters configured by default. However, to ensure that the robot's parameters are configured correctly, it is best for the user to check if the following configuration is correct after receiving the robot dog.

Configure the static IP of the computer

For LIDARs manufactured by Sprint Polytron (RoboSense), such as `RS-LIDAR-16` and `RS-HELIOS_16p`.

- The factory default LIDAR IP are: `192.168.1.200`
- The default network configuration of the target receiver computer are.
  - Static IP address: `192.168.1.102`
  - Subnet mask: `255.255.255.0`

On the computer that needs to run this program, add a static IP consistent with the LIDAR target IP:.

- Open the file

```
$ sudo vim /etc/network/interfaces
```

- Add the following (where `eth0` is the name of the current NIC and needs to be confirmed with `ifconfig`)

```
auto eth0:1
iface eth0:1 inet static
name For Robosense Lidar
address 192.168.1.102
netmask 255.255.255.0
broadcast 192.168.1.255
```

- Reboot this computer and then check to see if it contains the static IP.

```
$ ifconfig
```

- Check if the IP address of the LIDAR can be pinged from this computer, if it can be pinged, the configuration is correct.

```
$ ping 192.168.1.200

PING 192.168.1.200 (192.168.1.200) 56(84) bytes of data.
64 bytes from 192.168.1.200: icmp_seq=1 ttl=64 time=0.057 ms
64 bytes from 192.168.1.200: icmp_seq=2 ttl=64 time=0.044 ms
```

Configuring the robot's footprint

The robot's footprint is used to represent the shape and size of the robot's profile, which is modeled as a polygon. The footprint parameters vary from robot to robot, so it is important to configure the robot's footprint parameters according to the actual size and shape of the robot, so that the robot can plan a more reasonable path according to its shape and avoid colliding with objects in its path.

For example, for B1 machine dog, under the folder `lidar_slam_3d/navigation/param/b1`, the user needs to check the configuration file `costmap_common_params.yaml` first.

- The sample content of this file is as follows.

```
# General parameter settings for global maps
global_frame: local_map
robot_base_frame: base_link

# The shape of the robot in the LIDAR coordinate system, with all points connected together to form a closed polygon representing the shape of the robot
footprint: [[0.50, 0.30], [-0.8, 0.30], [-0.8, -0.30], [0.50, -0.30]]

footprint_padding: 0.0
```

- where the parameter after the footprint represents the coordinates of the four vertices of the rectangular shape formed by the current robot under the robot coordinate system `base_link`.

Running

The run needs to be run on the NX of the machine dog, whose ip address is

- `192.168.123.15` for `Go1`
- `192.168.123.12` for `A1`
- `192.168.123.220` for `Aliengo`
- `192.168.123.24` for `B1`

The code for this software is located in the path`~/UnitreeSLAM/catkin_lidar_slam_3d`。

All of the following operations require that the folder be accessed first.

```
cd ~/UnitreeSLAM/catkin_lidar_slam_3d
```

Before running a build and patrol, you need to ensure that the following conditions are met.

- The robot is in motion mode
- The UDP connection to the robot motion program SDK `unitree_legged_sdk` is not occupied by other ports, such as `~/RobotVisionSystem` and `2D SLAM`. If this port is occupied, we need to close the program that occupies it, otherwise we will not be able to send control commands to the robot motion program.

Building Map

Open a command line window and start the build task.

```
$ sudo su

$ source devel/setup.bash

$ roslaunch start build_map.launch map_name:=my_map_name
```

Open a second command line window to start RVIZ visualization:.

```
$ rosrun rviz rviz -d src/lidar_slam_3d/start/rviz/build_map.rviz
```

The patrol points are saved in the `txt` file under the folder `start/maps/gmapping`. Each row has one patrol point, and the three values in each row are: `x`, `y`, `yaw` (angle), and `time` (dwell time).

Patrol

When patrolling, you need to start the patrolling task at the starting position of the map (otherwise it cannot be accurately positioned), and the robot dog will patrol in order according to the patrol points set at the time of map building.

- First, move the dog to the initial position and orientation of the map.

- Open a command line window and start the patrol task:

```
$ sudo su

$ source devel/setup.bash

$ roslaunch start start_patrol.launch map_name:=my_map_name
```

- Open a second command line window to start RVIZ visualization:.

```
$ rosrun rviz rviz -d src/lidar_slam_3d/start/rviz/start_patrol.rviz
```

Caution.

- When running `start_patrol.launch`, an error may be reported indicating that there is no tf from the `map` coordinate system to the `base_link` coordinate system. This is because the `prm_localization` node failed when relocating based on the point cloud, mainly because the machine dog was not placed in the same position as when the map was initially built before running the patrol before running the patrol program.

Remote visualization of robots

Here is an example of the `Go1` robot.

Remote visualization is achieved through multi-computer communication of ROS. Therefore, the NX computer of the robot needs to be connected to the same LAN as the host computer, and `ROS_MASTER_URI` and `ROS_IP` need to be configured.

The configuration is as follows.

- Plug a Wifi receiver into the USB port of the `NX` computer on the `Go1` robot, as shown in the figure above.
- On the `NX` computer, connect to the same LAN where the host computer is located through the WiFi receiver.
- Assume that the IP of the robot receiver is `192.168.1.76` and the IP of the upper computer is `192.168.1.115`. You can ping each other to make sure the network is open.
- Open the file `~/.bashrc` on `NX` to configure the robot `NX` as `ROS Master` as follows.

  ```
  export ROS_MASTER_URI=http://192.168.1.76:11311
  export ROS_IP=192.168.1.76
  ```

- Open the file `~/.bashrc` in the upper machine to configure it as a slave as follows

  ```
  export ROS_MASTER_URI=http://192.168.1.76:11311
  export ROS_IP=192.168.1.115
  ```

- Then, open a terminal in the OP and use `ssh` to log in remotely to the robot `NX` and go under the 3d slam program path.

  ```
  ssh unitree@192.168.1.76
  ```

  ```
  cd ~/UnitreeSLAM/catkin_lidar_slam_3d
  ```

- Then, start the build or patrol program as described in the previous start method in Build or Patrol.
- Then, open a new 2nd terminal on the host computer and start the build or patrol Rviz visualization remotely according to the start method in Build or Patrol earlier.