6、ORB_ SLAM2 PCL mapping

PCL official website address：http://pointclouds.org

pcl_ros wiki：https://wiki.ros.org/pcl_ros

pcl_ros github：https://github.com/ros-perception/perception_pcl

The operating environment and reference configurations for software and hardware are as follows:

- Reference model: ROSMASTER X3
- Robot hardware configuration: Arm series main control, Silan A1 LiDAR, AstraPro Plus depth camera
- Robot system: Ubuntu (version not required)+Docker (version 20.10.21 and above)
- PC Virtual Machine：Ubuntu（20.04）+ ROS2（Foxy）
- Usage scenario: Use on a relatively clean 2D plane

6.1、brief introduction

PCL (The Point Cloud Library) is a large open source project for 2D/3D image and point cloud processing. The PCL framework consists of many advanced algorithms, including filtering, feature estimation, Surface reconstruction, registration, model merging and segmentation, etc. These algorithms have many applications, such as filtering Outlier in noise data, splicing multiple groups of 3D point clouds, segmenting relevant parts of the scene, extracting key points and Computational geometry descriptors of geometric shapes to identify objects, using point clouds to create and visualize object surfaces, and so on. PCL has been successfully compiled and configured on platforms such as Linux, MacOS, Windows, and Android/iOS. To simplify development, PCL is divided into a series of small Codebase that can be compiled separately.

Basic concepts of PCL interfaces for point cloud libraries and ROS：

Point cloud library: provides a set of data structures and algorithms for processing 3D data；

ROS's PCL interface: Provides a set of messages and conversion functions between messages and PCL data structures.

From the perspective of C++, PCL contains a very important data structure, which is point clouds. This data structure is designed as a template class, which takes the type of point as a parameter of the template class. The point cloud class is actually a container that contains all the common information required by the point cloud, regardless of the type of point. The following are the most important public fields in the point cloud:

header： This field is of type pcl: PCLHeader. The acquisition time of the point cloud was specified.

points： This field is of type std:: vector<PointT...>, which is a container for storing all points. The PointT in the vector definition corresponds to the template parameter of the class, which is the type of the point.

width： This field specifies the width of the point cloud when organized into an image, otherwise it contains the number of points in the cloud.

height： This field specifies the height at which the point cloud is organized into an image, otherwise it will always be 1.

is_dense： This field specifies whether there are invalid values (infinite or NaN values) in the point cloud.

sensor origin： This field is of type Eigen:: Vector4f and defines the pose obtained by the sensor based on its translation relative to the origin.

sensororientation： This field is of type Eigen:: Ouarternionf and defines the pose obtained by sensor rotation.

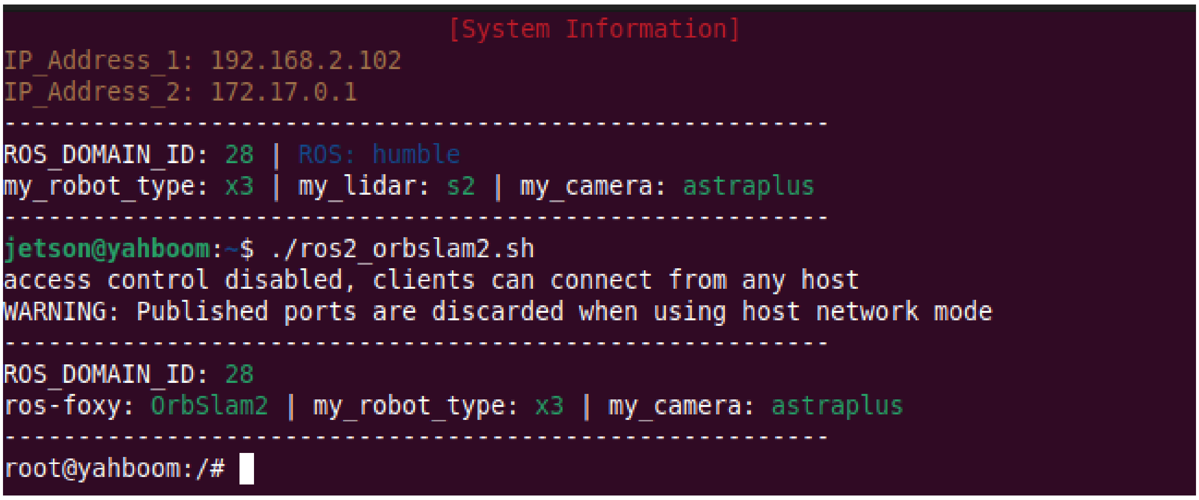
6.2、Using Tutorials

Note: The commands or locations mentioned below are all those in the Docker container unless otherwise specified.

Enter the Docker container (please refer to 【the Docker course section -5. Enter the Docker container of the robot】), and execute the following launch file on different terminals：

```
#Raspberry Pi 5 master user
~/run_docker.sh
#Orin master user
~/ros2_orbslam2.sh
```

Here, take orin master as an example and modify the corresponding car model

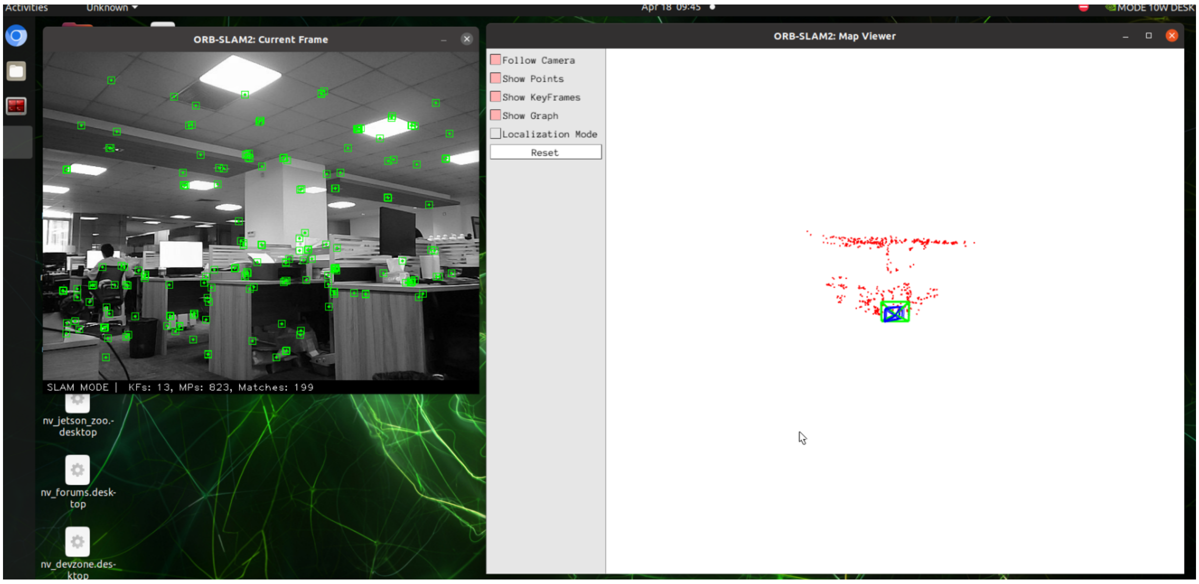


1、 Launch Camera

```
ros2 launch astra_camera astro_pro_plus.launch.xml
```

2、Launch orbslam to release camera pose, color and depth maps. Depending on the performance of different controllers, the waiting time here is approximately within 10 seconds

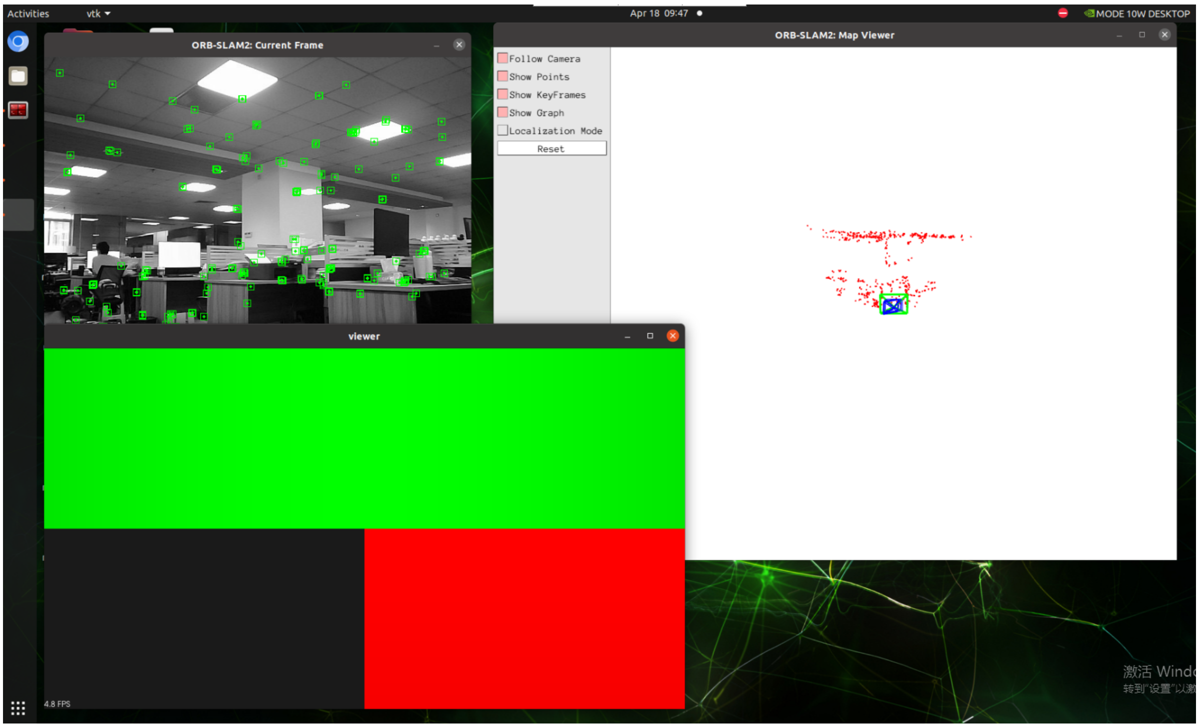
```
ros2 launch yahboomcar_slam orbslam_base_launch.py
```



3、 Start point cloud mapping

```
ros2 launch yahboomcar_slam orbslam_pcl_map_launch.py
```

After opening, a 【 viewer 】 window will pop up, and slowly move the camera. When an image appears, as shown in the following figure:



Need to scale and rotate the coordinate system, as shown in the following figure

Sliding roller: retraction and retraction

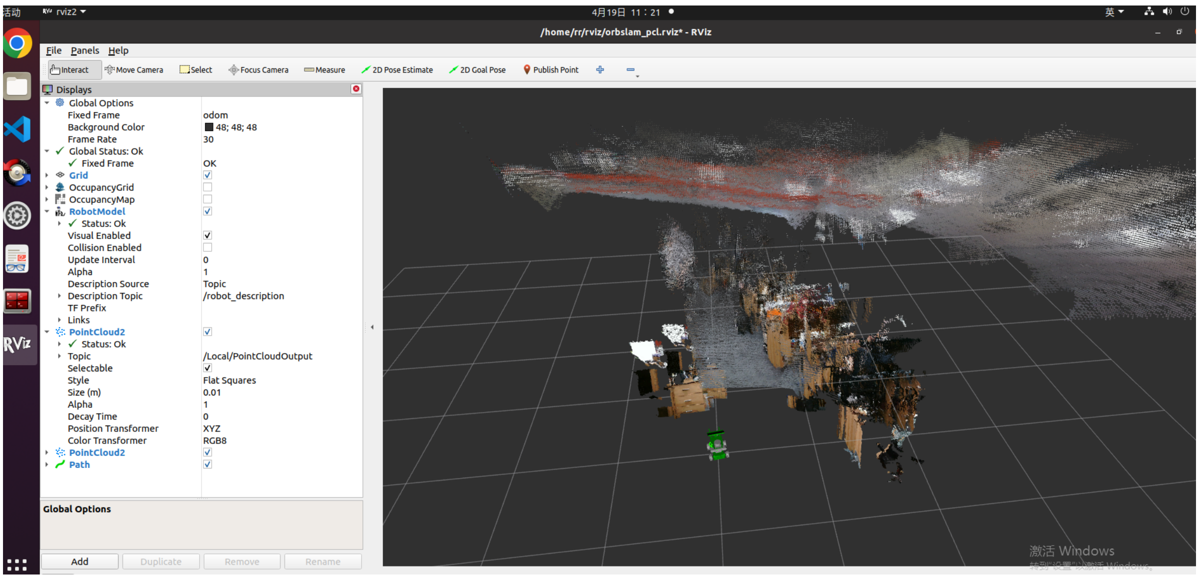
Hold down the scroll wheel: pan

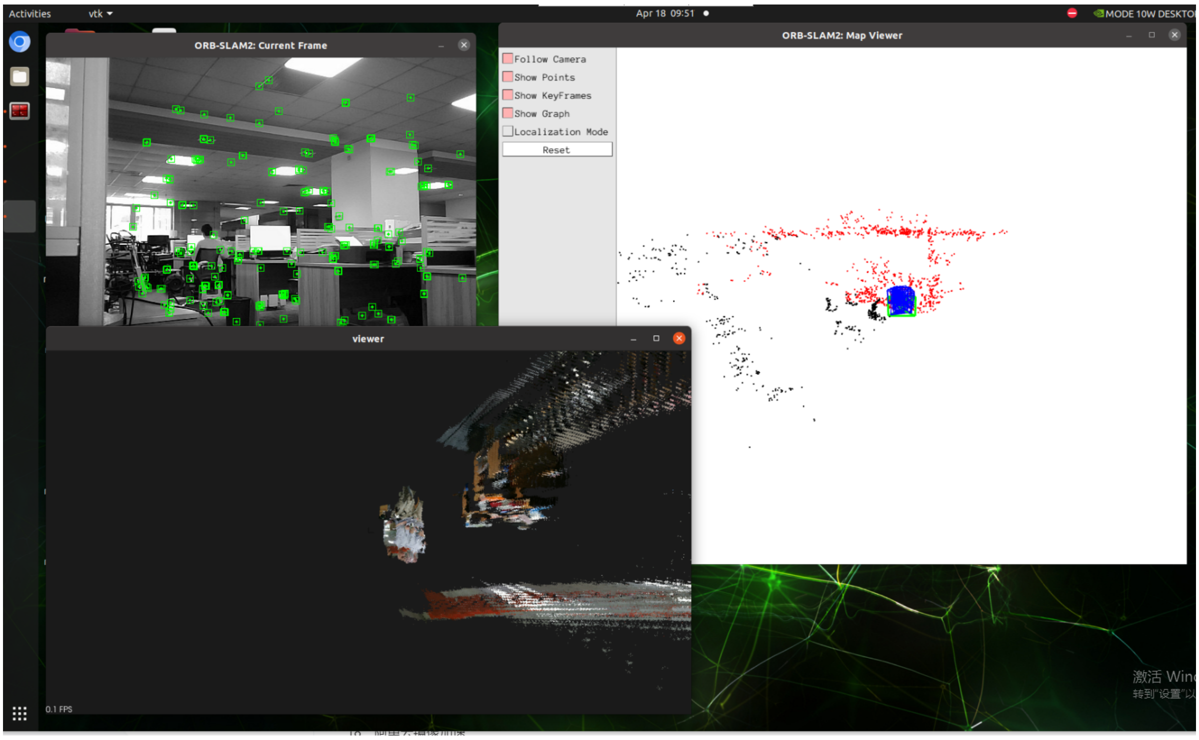
Left mouse button: Rotate

Right mouse button: Zoom in and out

You can simultaneously open rviz to view the real-time point cloud mapping effect. Considering the high CPU and memory costs during the point cloud mapping process, it is recommended to use the virtual machine to enable rviz to view:

```
ros2 launch yahboomcar_slam display_pcl_launch.py
```





4、Slowly move the camera to create the image as shown below. After creating, press **[Ctrl+c]**to close and save the PCD point cloud file resultPointCloudFile.pcd. The path is as follows:

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_slam/pcl/resultPointCloudFile.pcd
```

5、View the resultPointCloudFile.pcd file

（1）Method 1：Using PCL_ Viewer tool

Install PCL_ Viewer command：

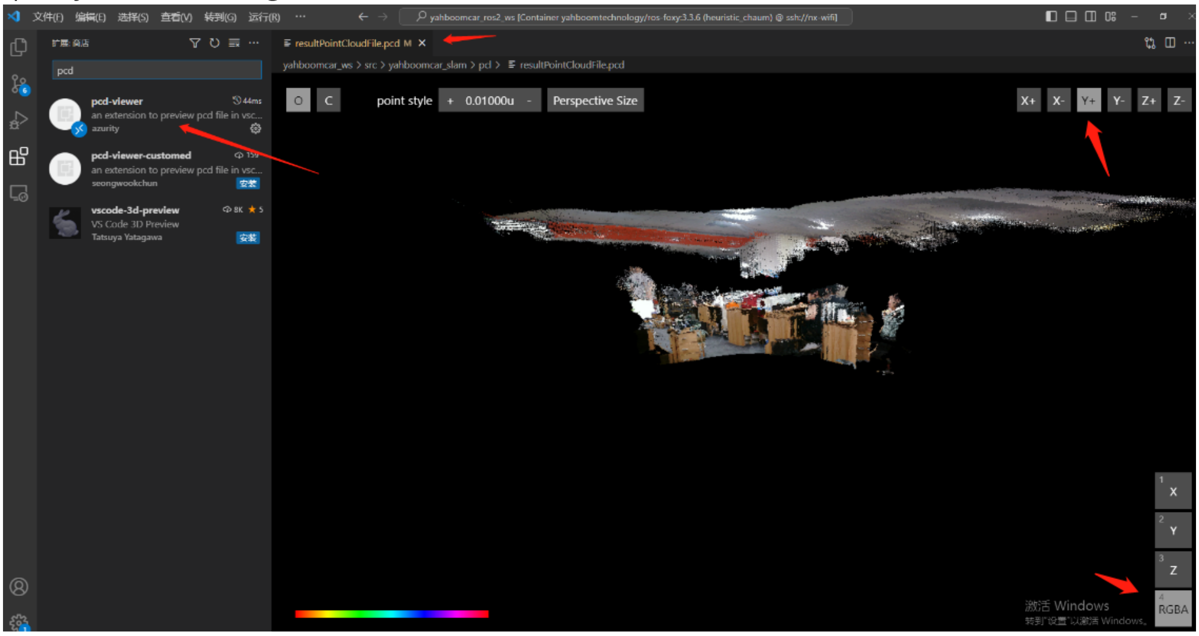
```
sudo apt-get install pcl-tools
```

Enter the directory where the PCD component is located and enter the following command to view the PCD point cloud map.

```
pcl_viewer resultPointCloudFile.pcd
```

（2）Method 2：Using the PCD viewer plugin for vscode **[recommended]**

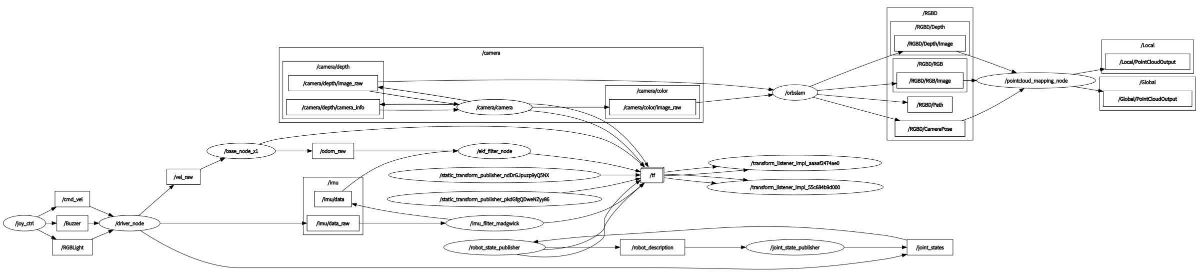
First install the pcd viewer plugin in vscode, and then open resultPointCloudFile.pcd to view the pcd point cloud image. Directly selecting the Y+direction and RGBA mode at the bottom here can quickly rotate the image to the main view .



6.3、Node resolution

6.3.1、Display Calculation Chart

```
rqt_graph
```



6.3.2、Pointcloud_ Mapping node details

```
rr@rr-pc:~/rviz$ ros2 node info /pointcloud_mapping_node
/pointcloud_mapping_node
  Subscribers:
    /RGBD/CameraPose: geometry_msgs/msg/PoseStamped
    /RGBD/Depth/Image: sensor_msgs/msg/Image
    /RGBD/RGB/Image: sensor_msgs/msg/Image
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /Global/PointCloudOutput: sensor_msgs/msg/PointCloud2
    /Local/PointCloudOutput: sensor_msgs/msg/PointCloud2
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /pointcloud_mapping_node/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /pointcloud_mapping_node/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /pointcloud_mapping_node/get_parameters: rcl_interfaces/srv/GetParameters
    /pointcloud_mapping_node/list_parameters: rcl_interfaces/srv/ListParameters
    /pointcloud_mapping_node/set_parameters: rcl_interfaces/srv/SetParameters
    /pointcloud_mapping_node/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

6.3.3、TF transformation

```
ros2 run tf2_tools view_frames.py
```