

pcap to pcd

- [1.pcap解析](#)
 - [1.1 pcap通过ros播放](#)
 - [1.2 解析步骤](#)
 - [1.3 结果展示](#)
- 禾赛AT128（半固态）激光雷达**pcap convert to pcd**

1.pcap解析

数据输出采用千兆以太网 UDP/IP 通信协议，输出点云数据包。

多字节的字段均默认为无符号整型，按小端字节序；另行备注除外。

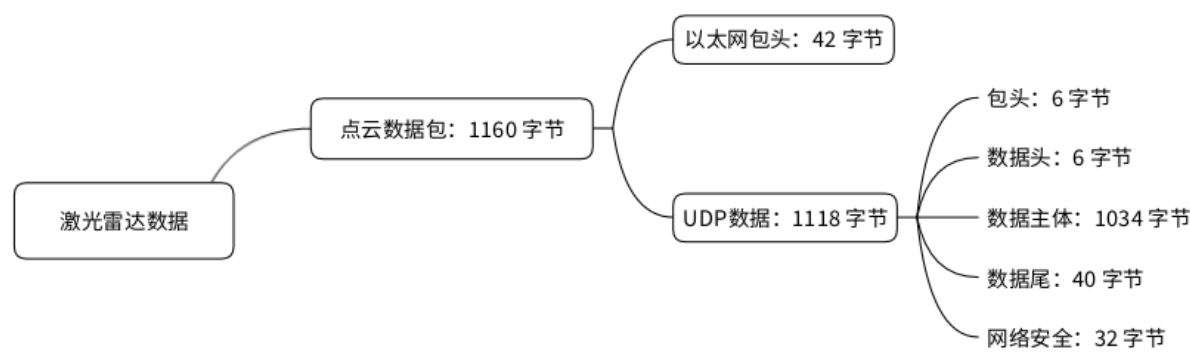


图 3.1 激光雷达数据结构

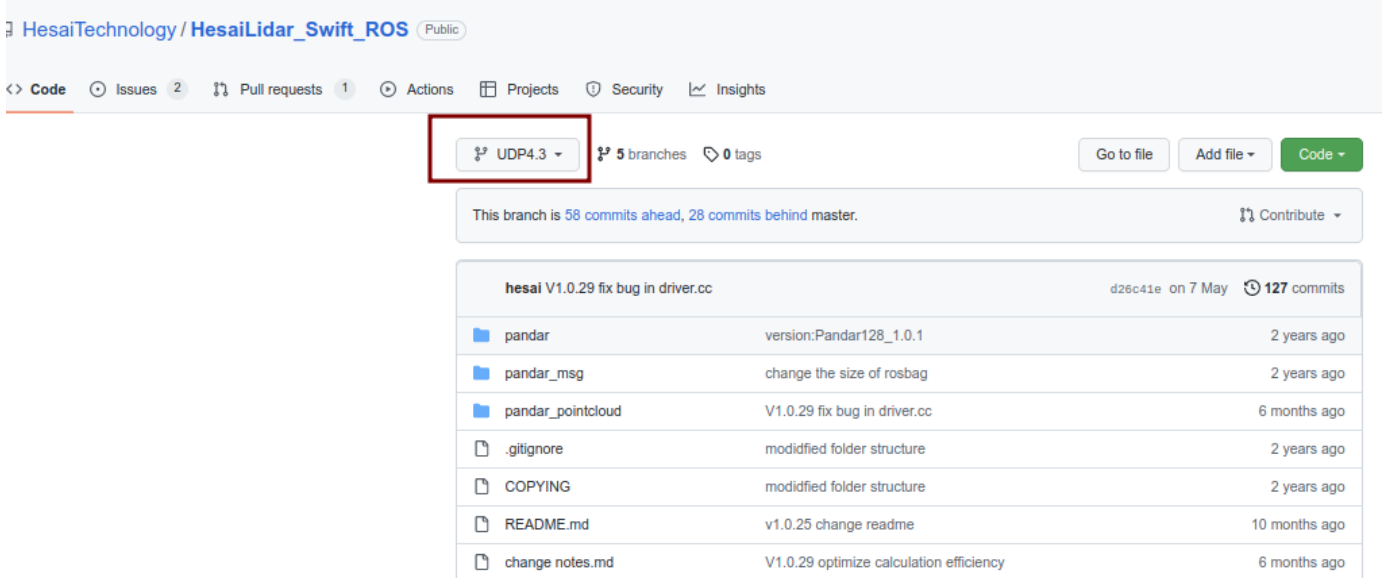
1.1 pcap通过ros播放

github中对pcap包的驱动程序:

[驱动程序连接](#)

https://github.com/HesaiTechnology/HesaiLidar_Swift_ROS

由于UDP4.7用来解析AT128，所以一定要在`git clone`后切换分支



- 这个仓库包括由 Hesai 技术公司生产的用于 Pandar 激光雷达传感器的 ROS 驱动程序。针对不同的系统和 UDP 协议版本包括分支。
- 该项目基于HesaiLidar_Swift_SDK开发，启动后将监测来自激光雷达的UDP数据包，解析数据并将点云框架发布到ROS的主题/pandar_points下。它也可以作为一个官方演示，展示如何使用HesaiLidar_Swift_SDK。

1.2 解析步骤

1. 下载库依赖 (Library Dependencies: libpcl-dev + libpcap-dev + libyaml-cpp-dev + libboost-dev)

```
$ sudo apt-get update
```

```
$ sudo apt install libpcl-dev libpcap-dev libyaml-cpp-dev libboost-dev
```

2. 下载和构建

- 下载Install catkin_tools:

```
$ sudo apt-get update
```

```
$ sudo apt-get install python-catkin-tools
```

python-catkin-tools这个工具的下载可能会出问题，但是已经安装好ros的情况下，直接跳过这步。

- 下载项目代码

```
$ mkdir -p rosworkspace/src
```

```
$ cd rosworkspace/src
```

```
$ git clone https://github.com/HesaiTechnology/HesaiLidar_Swift_ROS.git --recursive
```

- 切换分支(一定要切到UDP4.3)

```
$ git checkout <branch>
```

若要在新建的issue1分支进行提交，需要切换到issue1分支。

ps例如:

```
$ git checkout issue1
```

- 下载rosdep所需的依赖

```
$ cd ..
```

当前在rosworkspace工作目录下

```
$ rosdep install -y --from-paths src --ignore-src --rosdistro $ROS_DISTRO
```

- 编译

当前在rosworkspace工作目录下

```
catkin_make -DCMAKE_BUILD_TYPE=Release
```

4. 配置文件

- 寻找launch文件

在 `$ cd src/HesaiLidar_Swift_ROS/pandar_pointcloud/launch` 下存在三个launch文件，修改**PandarSwift_points.launch**配置参数；

参数说明如下图：

Configuration

```
$ cd src/HesaiLidar_Swift_ROS/pandar_pointcloud/launch
```

open PandarSwift_points.launch to set configuration parameters

Receiving data from connected LiDAR: config LiDAR IP address & UDP port, and leave the pcap_file empty

Parameter	Default Value
device_ip	192.168.1.201
port	2368
pcap_file	

Data source will be read from connected LiDAR when "pcap_file" is set to empty

Receiving data from pcap file: config pcap_file and correction file path

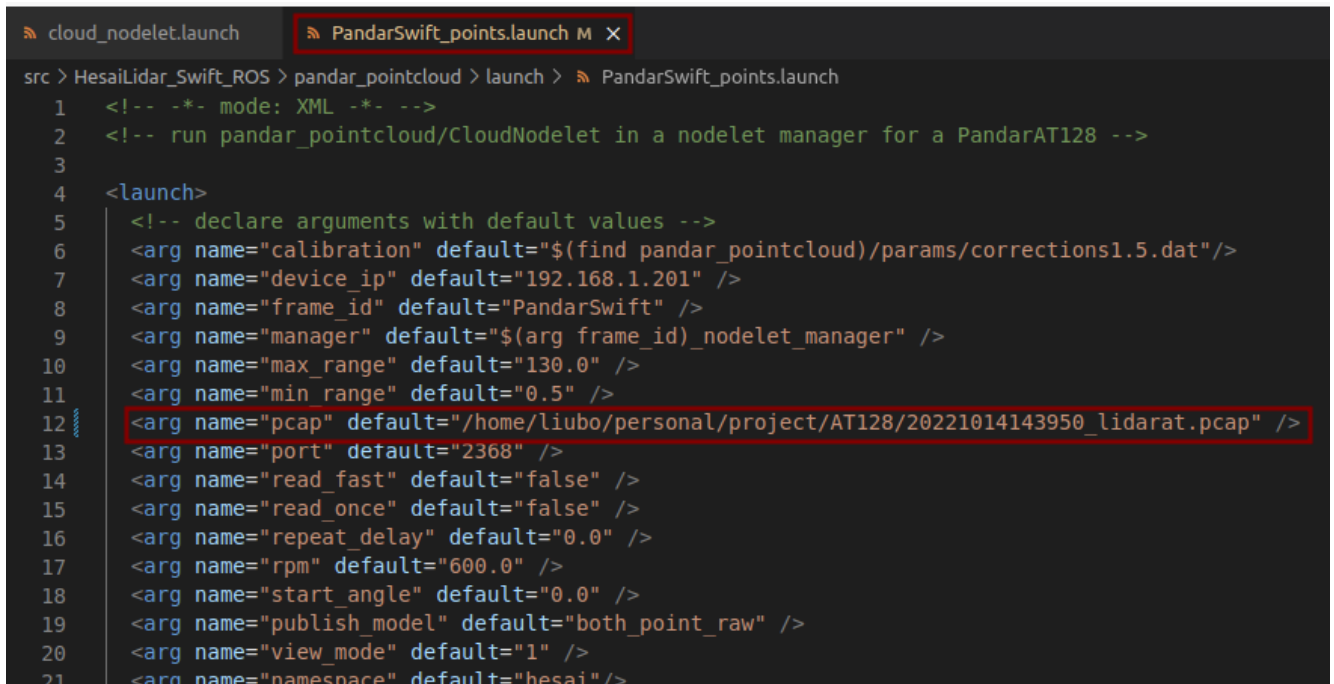
Parameter	Value
pcap	pcap file path

Data source will be read from pcap file instead of LiDAR once "pcap_file" not empty

两种情况：

- (1) 从连接的lidar接收数据：配置lidar IP、UDP端口、pcap文件路径留空

(2) 从pcap文件接收数据：只需配置pcap文件路径



```
src > HesaiLidar_Swift_ROS > pandar_pointcloud > launch > PandarSwift_points.launch
1  <!-- -*- mode: XML -*- -->
2  <!-- run pandar_pointcloud/CloudNodelet in a nodelet manager for a PandarAT128 -->
3
4  <launch>
5      <!-- declare arguments with default values -->
6      <arg name="calibration" default="$(find pandar_pointcloud)/params/corrections1.5.dat"/>
7      <arg name="device_ip" default="192.168.1.201" />
8      <arg name="frame_id" default="PandarSwift" />
9      <arg name="manager" default="$(arg frame_id)_nodelet_manager" />
10     <arg name="max_range" default="130.0" />
11     <arg name="min_range" default="0.5" />
12     <arg name="pcap" default="/home/liubo/personal/project/AT128/20221014143950_lidarat.pcap" />
13     <arg name="port" default="2368" />
14     <arg name="read_fast" default="false" />
15     <arg name="read_once" default="false" />
16     <arg name="repeat_delay" default="0.0" />
17     <arg name="rpm" default="600.0" />
18     <arg name="start_angle" default="0.0" />
19     <arg name="publish_model" default="both_point_raw" />
20     <arg name="view_mode" default="1" />
21     <arg name="namespace" default="hesai" />
```

5. 尝试运行Roslaunch

确保当前路径在rosworkspace目录下：

```
$ source devel/setup.bash
```

```
$ roslaunch pandar_pointcloud PandarSwift_points.launch
```

环境变量更新说明：

安装ros的官方教程，我们会发现在建立环境变量时会有给我们三种选择：

第一种：可以在你每次在启动新的shell时自动添加ROS的环境变量

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

```
source ~/.bashrc
```

第二种：只是在你当前的shell中添加ROS的环境变量

```
source /opt/ros/kinetic/setup.bash
```

第三种：使用的是zsh，而不是bash

```
echo "source /opt/ros/kinetic/setup.zsh" >> ~/.zshrc
```

```
source ~/.zshrc
```

这里我们使用前两种进行说明。

我们上面说过自己定义的全局变量和局部变量在注销bash时就会失效，想让自己定义的变量不失效的办法：写入配置文件

因此，就有了ros中第一种环境变量的建立：

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

```
source ~/.bashrc
```

第一行表示把" "中的字符串写入到~/.bashrc中(需要注销再登陆才会生效，>> 表示数据流输出重定向"追加"，>表示"替换")

第二行表示把配置文件读入当前的shell中。(立即生效)

所以有ros第二种环境变量建立方法

```
source /opt/ros/kinetic/setup.bash
```

把配置文件读取到当前的shell中。

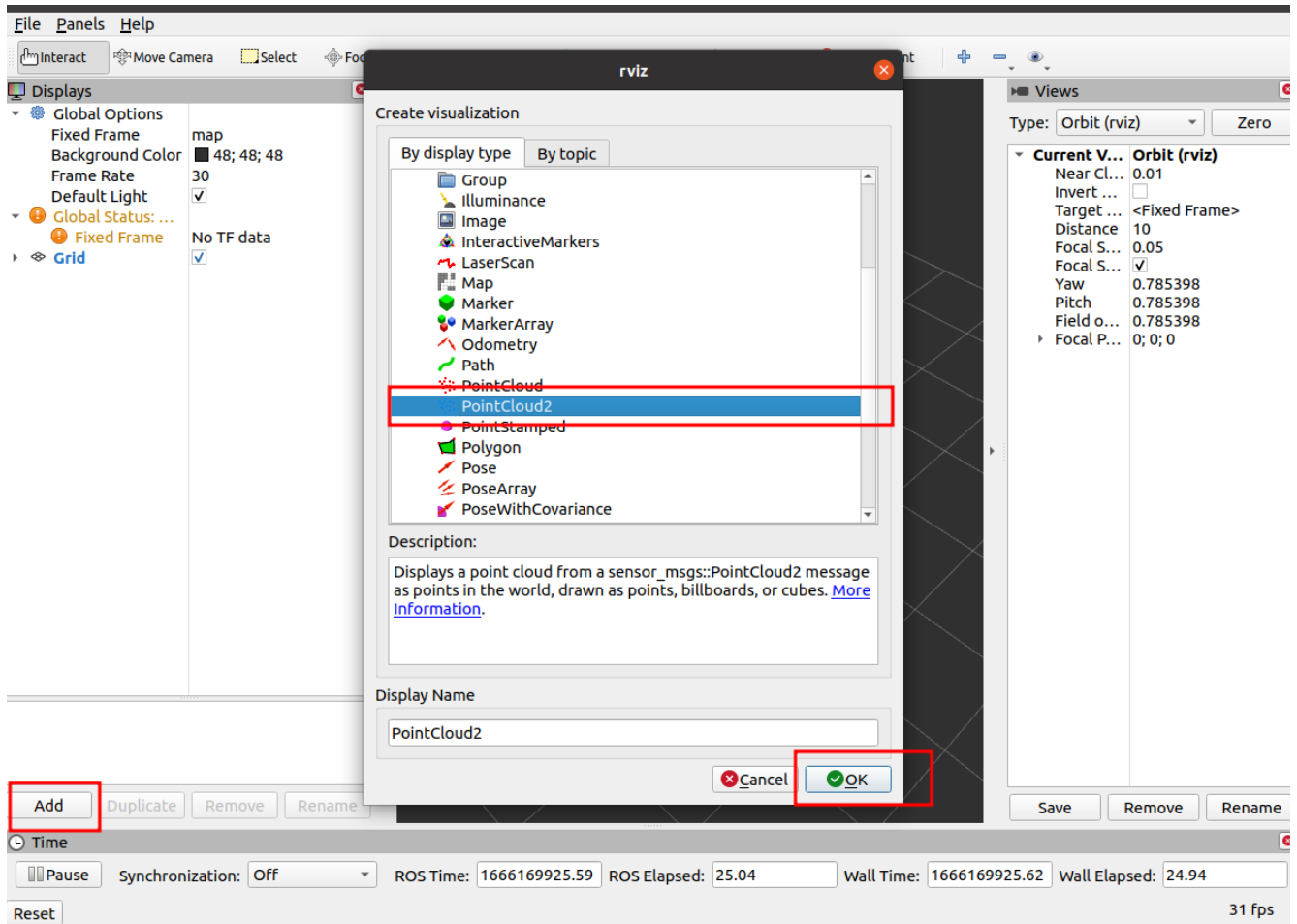
6. rviz查看点云

```
$ roscore 启动ros
```

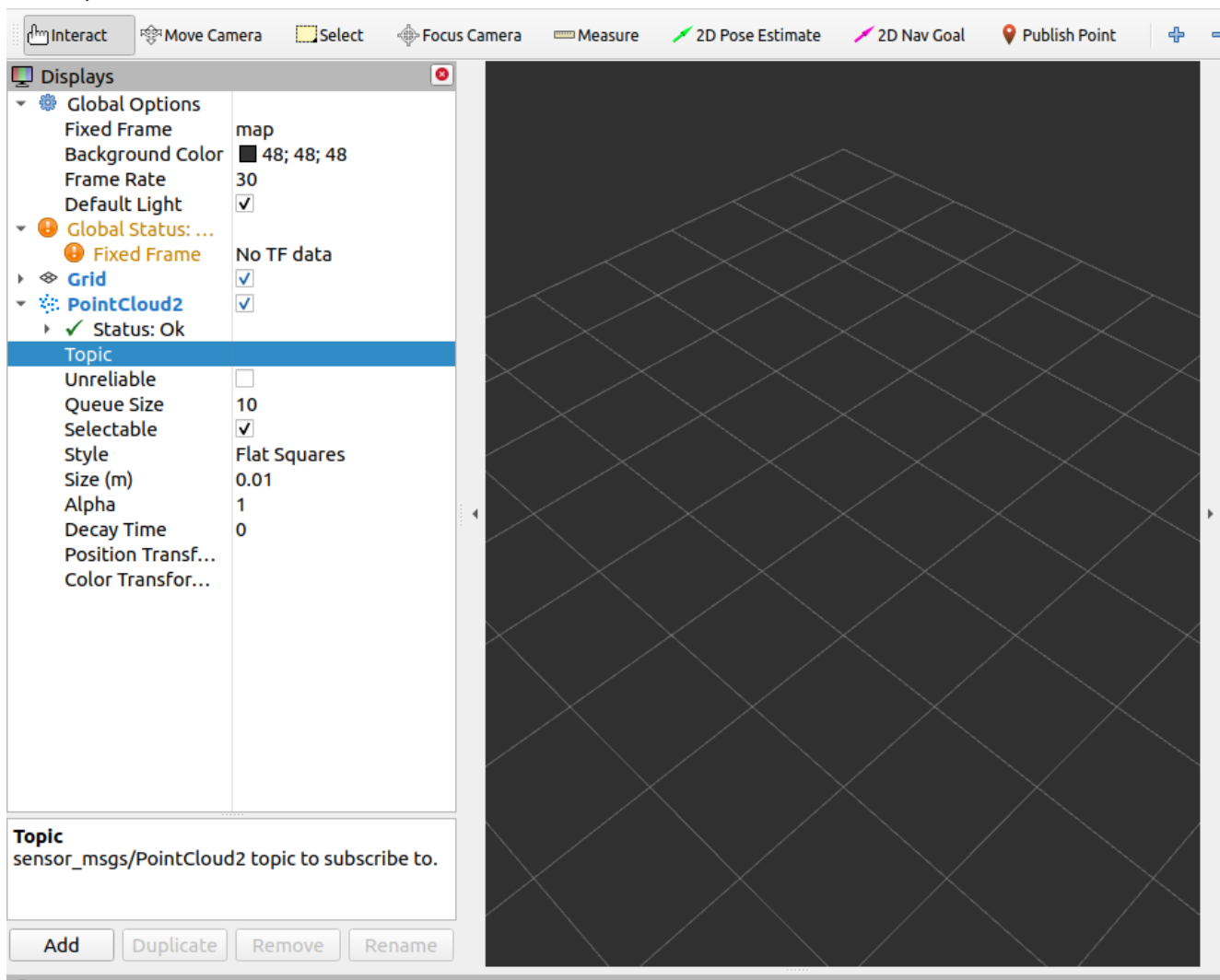
```
$ rviz 启动rviz
```

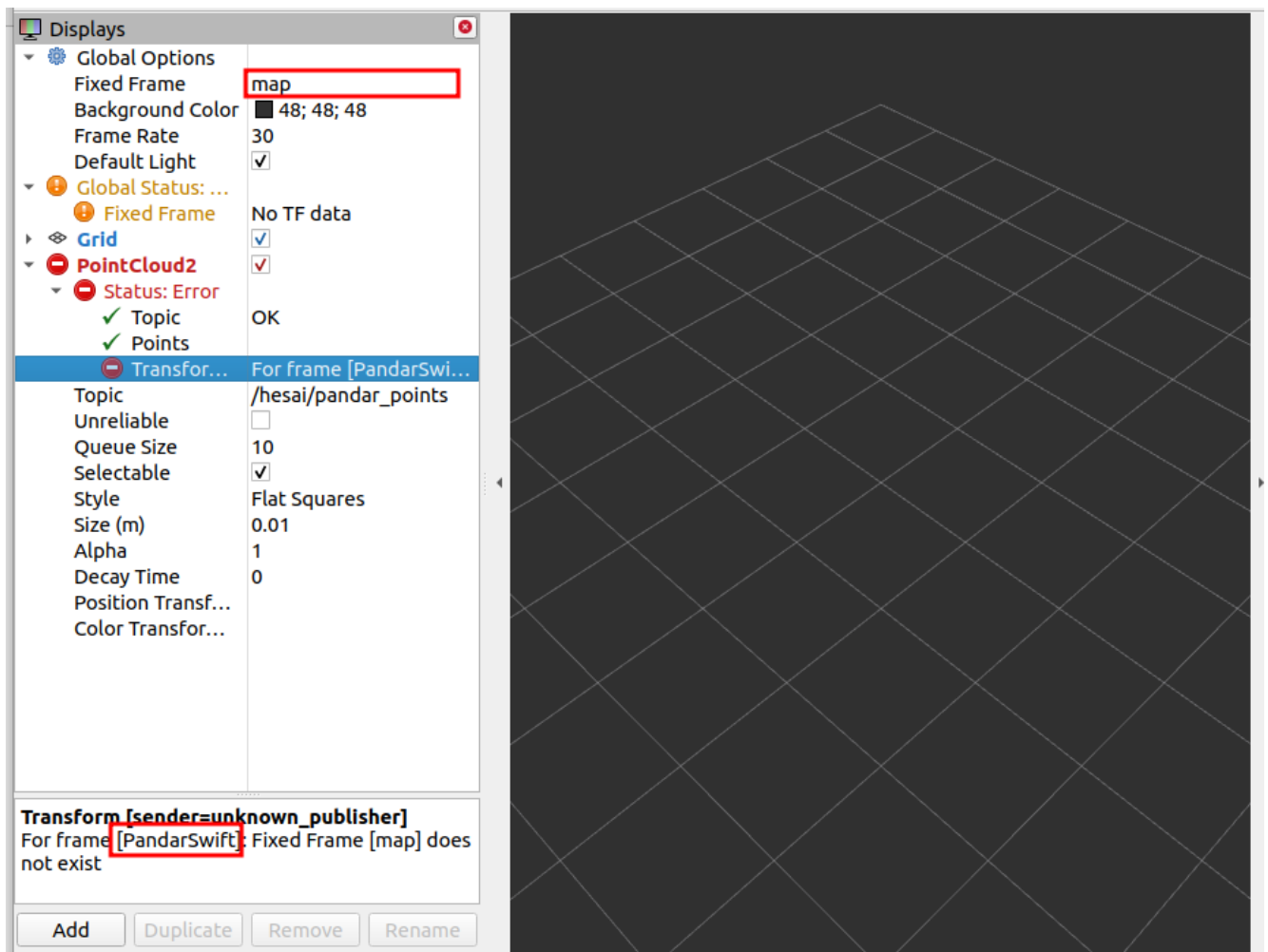
启动roslaunch之后，启动rviz：

- 接收加载pointcloud2消息

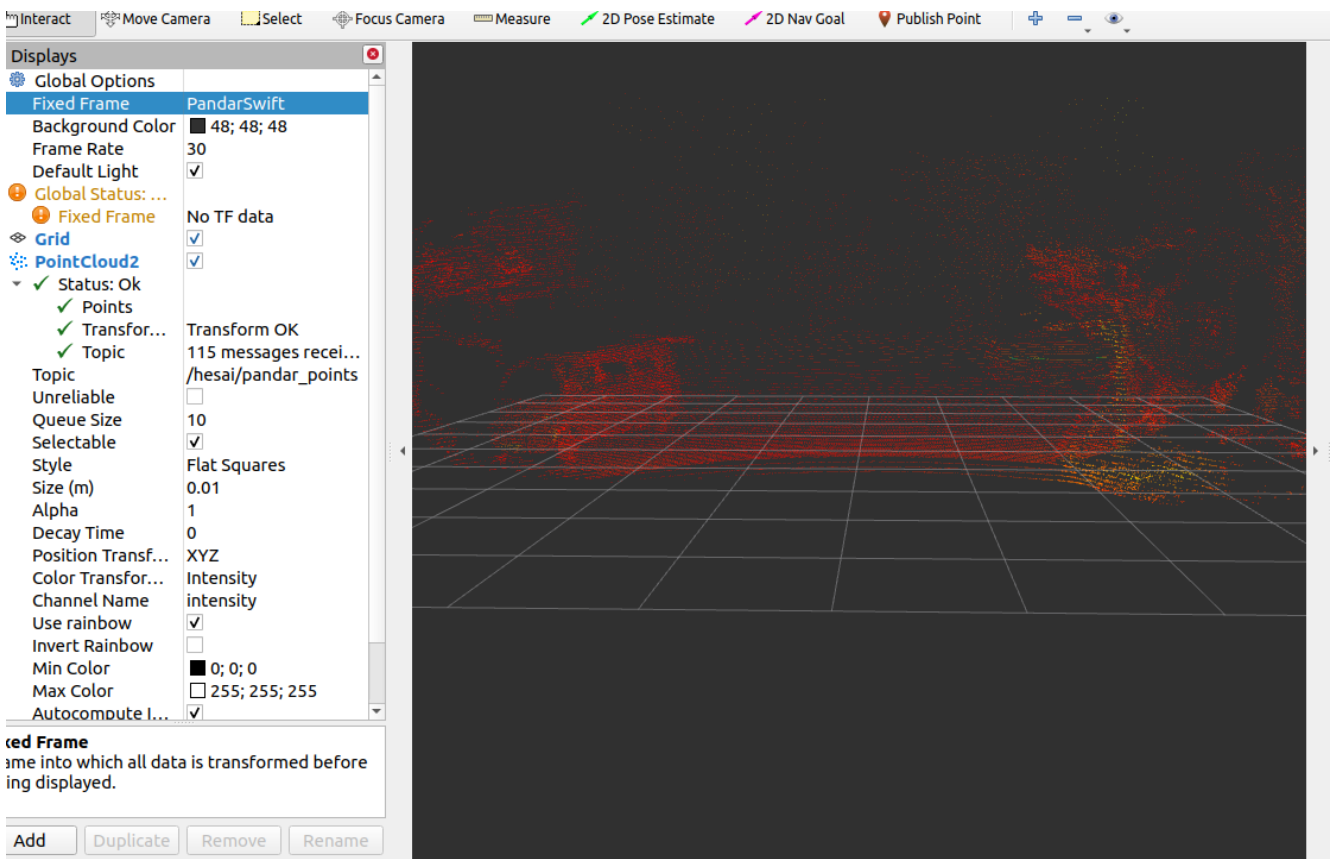


- 选择topic，将status中的transfrom信息添加到FixFrame





接收成功：



- 附加rostopic指令相关用法

rostopic is a command-line tool for printing information about ROS Topics.

Commands:

rostopic bw	display bandwidth used by topic
rostopic delay	display delay of topic from timestamp in header
rostopic echo	print messages to screen
rostopic find	find topics by type
rostopic hz	display publishing rate of topic
rostopic info	print information about active topic
rostopic list	list active topics
rostopic pub	publish data to topic
rostopic type	print topic or field type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'

rostopic list 查看当前话题:

```
[oh-my-zsh] Random theme 'skaro' loaded
969 ~/Desktop » rostopic list

/clicked_point
/diagnostics
/hesai/PandarSwift_nodelet_manager/bond
/hesai/PandarSwift_nodelet_manager_cloud/parameter_descriptions
/hesai/PandarSwift_nodelet_manager_cloud/parameter_updates
/hesai/pandar_gps
/hesai/pandar_packets
/hesai/pandar_points
/initialpose
/move_base_simple/goal
/rosout
/rosout_agg
/tf
/tf_static
```

rostopic echo rostopic_name 在屏幕上打印rostopic_name的消息

8. 同步使用Ros订阅者处理消息

```
pcd_writer.cpp M x CMakeLists.txt 1593173282.520140.pcd
c > pcd_reader > src > pcd_writer.cpp > main(int, char **)
48     }
49     cloud->height = cloud_msg->height;
50     cloud->width = cloud_msg->width;
51 }
52
53 pcl::PCDWriter writer;
54 void callback(sensor_msgs::PointCloud2::Ptr msg)
55 {
56     // pcl::PCLPointCloud2 cloud;
57     pcl::PointCloud<pcl::PointXYZ>::Ptr xyzi_cloud(new pcl::PointCloud<pcl::PointXYZ>);
58     // pcl_conversions::toPCL(*msg, cloud);
59     for (auto &f : msg->fields)
60     {
61         // std::cout << f.name << std::endl;
62         ROS_INFO("field %s", f.name.c_str());
63     }
64
65     fromROSMsg_DIY(msg, xyzi_cloud);
66     // exit(1);
67
68     writer.writeASCII("/home/liubo/caic/pointcloud/" + std::to_string(msg->header.stamp.toSec()) + ".pcd", *xyzi_cloud);
69 }
70
71 int main(int argc, char **argv)
72 {
73     ros::init(argc, argv, "pointfilter");
74     ros::NodeHandle nh;
75     auto ros_bag_callback =
76     {
77         nh.subscribe("/hesai/pandar_points", 100, callback);
78     };
79     ros::spin();
80     // system("pause");
81     return (0);
82 }
```

- ros订阅者读取发布者消息，xyz，intensity
- ros订阅者修改topic、存贮路径

执行：

```
$ source devel/setup.zsh
```

```
$ rosrn pcd_reader pcd_reader_writer
```

- 附加：cmakelsits文件解释rosrun命令：

```
cmake_minimum_required(VERSION 3.0.2)
project(pcd_reader)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  pcl_ros
  roscpp
)

# add_executable(${PROJECT_NAME}_node src/pcd_reader_node.cpp)
add_executable(${PROJECT_NAME} src/pcd_reader.cpp)
add_executable(${PROJECT_NAME}_writer src/pcd_writer.cpp)
```

- 附加：rosrn和roslaunch使用cmakelist配置：

1. 找到add_executable，去掉注释，将要编译的源文件替换成自己的

```
1 | add_executable(talker src/talker.cpp)
2 | add_executable(listener src/listener.cpp)
```

2. 找到target_link_libraries，去掉注释，修改成自己的

```
1 | target_link_libraries(talker
2 |     ${catkin_LIBRARIES}
3 | )
4 |
5 | target_link_libraries(listener
6 |     ${catkin_LIBRARIES}
7 | )
```

roslaunch + 包名 + 可执行文件

打开终端

```
1 | roscore
```

新建终端，发布节点发布话题

```
1 | source devel/setup.bash
2 | roslaunch pkg_test talker
```

新建终端，订阅节点接收消息

```
1 | source devel/setup.bash
2 | roslaunch pkg_test listener
```

roslaunch使用对比：

roslaunch需要一个一个打开节点，在节点较多的时候比较麻烦，这时候roslaunch就派上用场了。

我们把需要启动的节点写在launch文件中，launch脚本会自动帮我们运行需要的节点，而不需要手动逐个开启。但是launch脚本开启节点的顺序是不确定的。

我们在软件包下新建launch文件夹，并新建launch文件

```
1 | cd projects/catkin_test/src/pkg_test/  
2 | mkdir launch  
3 | cd launch  
4 | touch launch_test.launch
```

编写launch文件

```
1 | <launch>  
2 |   <node pkg="pkg_test" type="talker" name="talker2" output="screen"/>  
3 |   <node pkg="pkg_test" type="listener" name="listener2" output="screen"/>  
4 | </launch>
```

注意，pkg需要与软件包的名字相同，name与节点的名字倒是可以不同。

此时不需要重新编译。

启动launch文件，不需要通过roscore再运行master节点(roslaunch会启动的)

```
1 | roslaunch pkg_test launch_test.launch
```

效果如下：

```
process[master]: started with pid [12147]  
ROS_MASTER_URI=http://localhost:11311  
  
setting /run_id to 0e0c95b8-7347-11ea-af9e-74c63b05305b  
process[rosout-1]: started with pid [12162]  
started core service [/rosout]  
process[talker2-2]: started with pid [12166]  
process[listener2-3]: started with pid [12172]  
[ INFO] [1585655961.060439304]: hello world 0  
[ INFO] [1585655961.160509378]: hello world 1  
[ INFO] [1585655961.260546980]: hello world 2  
[ INFO] [1585655961.360574974]: hello world 3  
[ INFO] [1585655961.361227430]: I heard: [hello world 3]  
[ INFO] [1585655961.460514112]: hello world 4  
[ INFO] [1585655961.461045887]: I heard: [hello world 4]  
[ INFO] [1585655961.560579401]: hello world 5
```

roslaunch和roslaunch 都有自动补全命令，可以通过tab寻找包名和节点名

1.3 结果展示

- pcd文件以时间戳命名

