Robosense Perception communication 3.1

Robosense Perception communication 3.1

- 0. 适用范围和符号约定
  - 0.1 符号约定
  - 0.2 适用范围
- 1. 通信模块功能
  - 1.1 通信功能
  - 1.2 通信功能整体流程
- 2. 通信工程的配置及使用指南
  - 2.1 通信工程源代码结构
  - 2.2 基础配置
  - 2.3 通信参数详细配置
    - 2.3.1 Native配置
    - 2.3.2 Ros配置
    - 2.3.3 Ros2配置
    - 2.3.4 NativeBytes配置
      - 2.3.4.1 针对SDK_2_X协议的版本
      - 2.3.4.2 针对SDK_3_0协议的版本
      - 2.3.4.3 针对SDK_3_1协议的版本
    - 2.3.5 Protobuf 配置
    - 2.3.6 V2R配置
    - 2.3.7 Websocket配置
  - 2.4 通信实现源代码说明
- 3. API的使用方法
  - 3.1 接口介绍
    - 3.1.1 发送端接口介绍
    - 3.1.2 接收端接口介绍
  - 3.2 接口调用方法
    - 3.2.1 发送端接口调用方法
    - 3.2.2 接收端接口调用方法
- 4. 通信协议
  - 4.1 ROS通信协议
  - 4.2 ROS2通信协议
  - 4.3 Native 通信协议
    - 4.3.1 消息头
    - 4.3.2 消息内容
  - 4.4 NativeBytes 通信协议
    - 4.4.1 SDK2.x通信协议
      - 4.4.1.1 消息头(Header)
      - 4.4.1.2 消息内容(Content)
        - 4.4.1.2.1 objects / attention_objects
        - 4.4.1.2.2 freespace
        - 4.4.1.2.3 lanes
        - 4.4.1.2.4 curbs
        - 4.4.1.2.5 non_ground_indices
        - 4.4.1.2.6 ground_indices
        - 4.4.1.2.7 background_indices
        - 4.4.1.2.8 point_cloud
        - 4.4.1.2.9 pose
    - 4.4.2 SDK3.0通信协议
      - 4.4.2.1 消息头(Header)

4.4.2.2 消息内容(Content)
4.4.2.2.1 objects / attention_objects
4.4.2.2.2 status
4.4.2.2.3 freespace
4.4.2.2.4 lanes
4.4.2.2.5 curbs
4.4.2.2.6 axis_lidar_pose
4.4.2.2.7 global_car_pose
4.4.2.2.8 point_cloud
4.4.3 SDK3.1通信协议
4.4.3.1 消息头(Header)
4.4.3.2 消息内容(Content)
4.4.3.2.1 timestamp
4.4.3.2.2 global pose
4.4.3.2.3 gps origin
4.4.3.2.4 status pose map
4.4.3.2.5 status
4.4.3.2.6 valid indices
4.4.3.2.7 objects / attention objects
4.4.3.2.8 point cloud
4.4.3.2.9 freespace
4.4.3.2.10 lanes
4.4.3.2.11 roadedge
4.4.3.2.12 sematic
4.5 V2R 通信协议
4.5.1 Robosense V2R 1.0~1.4 协议字段说明
4.5.2 Robosense V2R 1.5 协议字段说明
4.5.3 Robosense V2R 1.6 协议字段说明

0. 适用范围和符号约定

0.1 符号约定

- Robosense 通信模块为**全开源代码**提供，为表述源代码工程相关的文件目录/路径等方便，约定通信模块工程的根目录路径为**RS_COMM_ROOT_DIR**，SDK配置文件根目录为**RS_CONFIG_ROOT_DIR**

0.2 适用范围

- 通信模块功能: 介绍Robosense的通信模块的支持情况，**适用范围**: 入门了解
- 通信工程的配置及使用指南: 介绍如何配置和使用通信模块以及通信模块源文件介绍，**适用范围**: Robosense技术支持，二次开发工程师
- API的使用方法：介绍通信的模块的外部接口以及调用方式。
- 通信协议: 介绍每一种通信方式的实现协议细节, **适用范围**: 二次开发工程师

1. 通信模块功能

1.1 通信功能

Robosense 感知模块支持7种通信方式：**Native**, **Ros**, **Ros2**,**NativeBytes**, **Protobuf** ,**V2R**以及**基于WebSocket协议的网页显示**，其中各种通信方式的特点如**表1-1**所示，主要实现如下功能：

- 发送/接收感知障碍物(Object/Attention Object)
- 发送/接收(Freespace)
- 发送/接收路沿(Curbs)
- 发送/接收车道线(Lanes)
- 发送/接收点云(PointCloud)

表1-1 通信方式汇总表格

通信方式	实现方式	备注
Native	通过cereal进行数据序列化,通过socket udp进行数据发送或接收	(1) **依赖robosense perception common模块** (2) 支持发送和接收 (3) Native协议详细
Ros	基于ROS的消息发布和订阅机制实现通信	(1) **依赖ROS 和 robosense percepeption common 模块** (2) 支持发送和接收 (3) ros协议详细
Ros2	基于ROS2的消息发布和订阅机制实现的通信	(1) **依赖ROS2 和 robosense percepeption common 模块** (2) 支持发送和接收 (3) ros2协议详细
NativeBytes	按照robosense 小端序列进行数据序列化，通过socket udp进行数据发送或接收	(1) **依赖robosense perception common模块** (2) 支持发送和接收 (3) 实现SDK2.x和SDK3.0的Native通信兼容，为了作为区分，将该通信发送记为**NativeBytes** (4) 协议参考robosense 对应的sdk的通信协议文档
Protobuf	按照Google protobuf协议进行数据序列化，通过socket udp进行数据发送或者接收	(1) **依赖robosense perception common模块** (2) 支持发送和接收 (3) Protobuf协议详细
V2R	按照robosense V2R协议进行数据序列化，通过socket udp/tcp client/tcp server进行数据发送或者接收	**(1) 依赖robosense perception common模块** (2) 支持发送和接收 (3) 协议参考robosense对应的v2r协议文档
Websocket	按照Websocket协议	**(1) 依赖robosense perception common模块** (2) 依赖websocketpp/boost和pcl(部分配置场景需要) (3) 协议参考robosenes3.0对应的sdk3.0的通信协议文档

1.2 通信功能整体流程

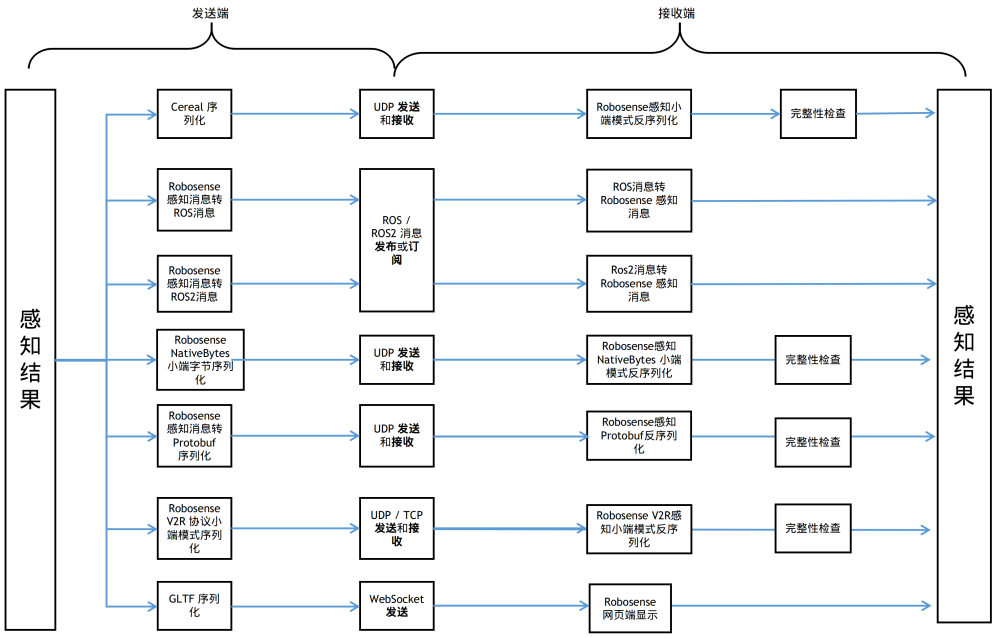

图1-1: 通信整体结构示意图

2. 通信工程的配置及使用指南

首先，通信模块的总开关在usr_config.yaml中（文件路径为：**RS_CONFIG_ROOT_DIR/usr_config/usr_config.yaml**）。若需要启用SDK的通信模块，需要将**run_communication**的开关**置为true**。如下所示：

```
#* General *#
general:
  application: "V2r" #V2r,Pseries,SmartSensor
  log_level: "info" #error,warning,info,debug,trace
  log: true #true,false for logging file save in /tmp/rs_sdk.log
  run_perception: true    #flag to control using perception algorithm or not
  run_localization: false
  run_communication: true
```

在**把run_communication**的开关**置为true**以后，通信模块将在运行demo时被启用。为了让通信模块能够正常运行并采用适当的方式进行通信，**还需要对通信模块的参数进行配置**。 通信模块的配置文件包含两层结构: (1) ***基础配置***， (2) ***详细配置***。 这两层结构的正确配置对通信模块的正常运行有重要作用。

基础配置 **的配置及使用示例**将在 **2.2节** 中进行介绍说明。***详细配置*** **的配置及使用示例**将在 **2.3节** 中进行介绍说明。在完成了2.2节和2.3节的配置之后，运行demo即可正常启用通信模块的功能。

若用户想通过调用API的方式来使用通信模块的功能，可以前往 *第3节 API使用方法* 进行阅读。不过采用调用API的方式来使用通信模块的功能仍需要对本节中提到的配置文件进行设置。因此，建议用户先阅读完本节，再阅读第三节。

2.1 通信工程源代码结构

目录名称	内容说明	备注
RS_COMM_ROOT_DIR/cmake	包含编译通信模块的CMake相关文件	
RS_COMM_ROOT_DIR/communication	包含通信模块实现	
RS_COMM_ROOT_DIR/custom	实现Robosense感知消息结构和通信数据消息结构转换	
RS_COMM_ROOT_DIR/message	包含ROS，ROS2的消息内容	
RS_COMM_ROOT_DIR/third_part	包含第三方开源代码	

2.2 基础配置

基础配置 相当于目录，主要用于配置各种通信方式的 ***详细配置*** 的YAML文件的路径，如下所示为***基础配置***的文件内容:（文件路径为:**RS_CONFIG_ROOT_DIR/system_config/communication_config/rs_communication.yaml**）：

```
#* support method: 1.Native 2.Ros 3.Ros2 4.NativeBytes(2_x, 3_0 ,3_1) 5.Proto
6.RobosenseV2R 7. Websocket 8.Custom*#

- method: Native
  config:
    include: /system_config/communication_config/native/config.yaml

# - method: Ros
#   config:
#     include: /system_config/communication_config/ros/config.yaml

# - method: Ros2
#   config:
#     include: /system_config/communication_config/ros2/config.yaml

# - method: NativeBytes
#   config:
#     include: /system_config/communication_config/native_bytes/config_2_x.yaml

# - method: NativeBytes
#   config:
#     include: /system_config/communication_config/native_bytes/config_3_0.yaml

# - method: NativeBytes
#   config:
#     include: /system_config/communication_config/native_bytes/config_3_1.yaml

# - method: Proto
#   config:
#     include: /system_config/communication_config/proto/config.yaml

# - method: RobosenseV2R
#   config:
#     include: /system_config/communication_config/v2r/config.yaml
```

```
# - method: Websocket
#   config:
#     include: /system_config/communication_config/websocket/config.yaml

# - method: Custom
#   config:
#     include: /system_config/communication_config/custom/config.yaml
```

其中, robosense节点下为Robosense提供的感知通信方法有以下八种：**native_udp, ros, ros2, native_bytes_udp, protobuf_udp, v2r_udp/tcp, websocket, custom**。

不同通信方法的配置文件为相对于**RS_CONFIG_ROOT_DIR路径的相对路径。 例如native_udp通信方式的通信配置路径为**"/system_config/communication_config/native/config.yaml"，**那么，对应的实际文件绝对路径为:** RS_CONFIG_ROOT_DIR/system_config/communication_config/native/config.yaml.

使用时，需要在上述 *基础配置* 中把需要使用的通信方法取消注释，即可在运行时启用对应的通信方式。例如，在上述的基础配置中，选用的就是Native（即native_udp）通信方式。**在启用该通信方式后，还需要在其对应的*详细配置* 中进行参数设置**。设置方法将在 **2.3节** 的对应通信方式下进行说明。

此外，通信模块支持将相同感知数据同时**发送多个远端**的功能，例如需要配置**native_udp_bytes**发送两个远端，那么**native_udp_bytes**节点的配置如下:

```
- method: NativeBytes
  config:
    include: /system_config/communication_config/native_bytes/config_2_x.yaml

- method: NativeBytes
  config:
    include: /system_config/communication_config/native_bytes/config_3_0.yaml
```

注意: 配置文件必须存在，并且只需要为相对**RS_COMM_ROOT_DIR/communication_config**路径下即可

2.3 通信参数详细配置

该部分为通信模块的***详细配置*** 参数，分别对应于***基础配置*** 中列出的不同的通信方式。下面针对具体的通信方式给出配置及快速使用说明。

2.3.1 Native配置

该配置对应于***基础配置*** 中的**Native方式（即native_udp方式）**。如下所示为该方式的配置文件的内容：（文件路径为:**RS_CONFIG_ROOT_DIR/system_config/communication_config/native/config.yaml**）

```
#* General *#
general:
  device_id: 0
socket:
  socket_address: 10.10.8.155    # 对于发送端为远端的IP, 对于接收端建议配置为0.0.0.0
  socket_port: 60082             # 对于发送端为远端的端口, 对于接收端为监听的端口
  socket_buffer_size: 4194304    # 默认即可
  max_msg_size: 32768            # 不允许大于63KByte,默认为32KByte
  timeout_ms: 150                # 接收超时, 默认150ms
  send_control:
```

```
    send_control_enable: true           # 如果发送点云时，该选项建议开启
    send_control_thres: 262144          # 默认为256KB
    send_control_ms: 3                  # 默认为2ms,建议 <= 4
    send_control_compress_enable: true # 如果发送点云时，该选项建议开启
  receive_control:
    receive_io_parallel_enable: true       # 如果发送点云，建议开启
    receive_io_parallel_degree: 4          # 根据机器设备配置，例如4核,建议配置不超过4
    receive_process_parallel_enable: true # 如果发送点云，建议开启
native:                                   # 发送额外内容控制项：配置为true时将发送，否
则不发送
  method: "Native"
  send_point_cloud: true
  send_attention_objects: true
  send_freespace: true
  send_lane: true
  send_roadedge: true
  send_sematic_indices: true
```

在使用的时候分为如下**两种**情况：

1. **本机发送且本机接收：**

   此时，需要将socket字段下的socket_address改为本机的ip。其他的配置的含义均在配置文件中以注释的方式给出。用户可以根据实际的需求进行更改。

2. **本机发送至远端或本机仅接收：**

   若本机为发送端，则将socket字段下的socket_address改为远端的的ip。若本机仅为接收端，则将socket字段下的socket_address改为0.0.0.0。

至此，通信方式为Native方式的配置已经结束。在编译完成整个SDK后，运行可执行文件rs_sdk_demo即可在完成感知后调用通信模块采用Native方式进行数据发送。此时在终端上会显示 send successed 的字样。

注意：Native通信方式支持发送、接收，其中接收端的可执行文件在编译后的/release/build/receiver_demo_beta目录下，终端执行可执行文件打印出接收结果进行验证。

2.3.2 Ros配置

该配置对应于***基础配置*** 中的**Ros方式**。如下所示为该方式的配置文件的内容：（文件路径为：**RS_CONFIG_ROOT_DIR/system_config/communication_config/ros/config.yaml**）

```
#* General *#
general:
  device_id: 0
control:
  topic_name: "percept_topic" # 发送的话题名称
  send_control:
    enable_parallel: true
    parallel_buffer_size: 2  # 当enable_parallel = true时，该参数有效，在发送点云
时，建议开启
ros:                          # 发送额外内容控制项：配置为true时将发送，否则不发送
  send_point_cloud: true
  send_attention_objects: false
  send_freespace: false
  send_lane: false
  send_roadedge: false
```

```
  send_sematic_indices: true
```

由于该通信方式采用的是ROS的消息订阅发送机制，因此在使用的过程中，只需要根据实际使用需求对该文件中的ros字段下的开关进行调整即可。

调整完毕后，通信方式为ros方式的配置已经结束。在编译完成整个SDK后，运行可执行文件rs_sdk_demo即可在完成感知后调用通信模块采用Ros方式进行数据发送。此时在终端上会显示 send successed 的字样。

若要在本机接收，则在一个新的终端中调用/release/build/receiver_demo_beta下的可执行文件即可。此时在终端上会打印receive successed的字样。同时，调用RVIZ可以可视化接收的结果。

注意：在一个终端中，ROS1和ROS2的环境不可以共存。因此在决定使用ROS1通信方式之前，务必确认当前的环境为ROS1的环境。否则该通信方式将不会生效。

2.3.3 Ros2配置

该配置对应于***基础配置*** 中的**Ros2方式**。如下所示为该方式的配置文件的内容：（文件路径为：**RS_CONFIG_ROOT_DIR/system_config/communication_config/ros2/config.yaml**）

```
#* General *#
general:
  device_id: 0
control:
  topic_name: "percept_topic" # 发送的话题名称
  send_control:
    enable_parallel: true
    parallel_buffer_size: 2  # 当enable_parallel = true时，该参数有效，在发送点云时，建议开启
ros:                         # 发送额外内容控制项：配置为true时将发送，否则不发送
  send_point_cloud: true
  send_attention_objects: false
  send_freespace: false
  send_lane: false
  send_roadedge: false
  send_sematic_indices: true
```

由于该通信方式采用的是ROS2的消息订阅发送机制，因此在使用的过程中，只需要根据实际使用需求对该文件中的ros字段下的开关进行调整即可。

调整完毕后，通信方式为ros2方式的配置已经结束。在编译完成整个SDK后，运行可执行文件rs_sdk_demo即可在完成感知后调用通信模块采用Ros2方式进行数据发送。此时在终端上会显示 send successed 的字样。

若要在本机接收，则在一个新的终端中调用/release/build/receiver_demo_beta下的可执行文件即可。此时在终端上会打印receive successed的字样。同时，调用RVIZ可以可视化接收的结果。

注意：在一个终端中，ROS1和ROS2的环境不可以共存。因此在决定使用ROS2通信方式之前，务必确认当前的环境为ROS2的环境。否则该通信方式将不会生效。

2.3.4 NativeBytes配置

该配置对应于***基础配置***中的**NativeBytes方式（即native_udp_bytes方式）**。 为了适配旧版本的SDK，配置又分为了 *针对SDK_2_X协议* 的版本和 *针对SDK_3_0协议* 的版本。有关上述旧版本与SDK3.1之间存在的字段差异以及差异是如何处理的说明，可以参考4.4节中的介绍。

2.3.4.1 针对SDK_2_X协议的版本

如下所示为该方式的配置文件的内容：（文件路径为：**RS_CONFIG_ROOT_DIR/system_config/communication_config/native_bytes/config_2_x.yaml**）

```
#* General *#
general:
  device_id: 0
socket:
  socket_address: 10.10.8.150 # 对于发送端为远端的IP，对于接收端建议配置为0.0.0.0
  socket_port: 60081          # 对于发送端为远端的端口，对于接收端为监听的端口
  socket_buffer_size: 4194304   # 默认即可
  max_msg_size: 15360           # 不允许大于63KByte,默认为32KByte
  timeout_ms: 150               # 接收超时，默认150ms
  send_control:
    send_control_enable: true           # 如果发送点云时，该选项建议开启
    send_control_thres: 262144          # 默认为256KB
    send_control_ms: 3                  # 默认为2ms,建议 <= 4
    send_control_compress_enable: true # 如果发送点云时，该选项建议开启
  receive_control:
    receive_io_parallel_enable: true       # 如果发送点云，建议开启
    receive_io_parallel_degree: 4          # 根据机器设备配置，例如4核,建议配置不超过4
    receive_process_parallel_enable: true # 如果发送点云，建议开启
native:
  custom_method: NativeBytesSDK2_X
  objects: true
  attention_objects: false
  object_supplement: false
  freespace: false
  lanes: false
  curbs: false
  non_ground_indices: false
  gound_indices: false
  background_indices: false
  point_cloud: false
  pose: false
```

由于该通信方式是利用socket_udp的方式进行通信。因此，在使用时应参照以下情况对该配置文件进行调整：

1. **若本机为发送端**，则需要将socket字段下的socket_address调整为远端的ip。
2. **若本机为接收端**，则需将socket字段下的socket_address调整为0.0.0.0。

   注意：NativeBytes通信方式支持发送、接收，其中接收端的可执行文件在编译后的release/build/receiver_demo_beta目录下，终端执行可执行文件打印出接收结果进行验证。

至此，通信方式为针对SDK_2_X协议的NativeBytes方式的配置已经结束。在编译完成整个SDK后，运行可执行文件rs_sdk_demo即可在完成感知后调用通信模块采用NativeBytes方式发送适配SDK_2_X协议的数据。此时在终端上会显示 send successed 的字样。

2.3.4.2 针对SDK_3_0协议的版本

如下所示为该方式的配置文件的内容：（文件路径为：**RS_CONFIG_ROOT_DIR/system_config/communication_config/native_bytes/config_3_0.yaml**）

```
#* General *#
general:
  device_id: 0
socket:
  socket_address: 10.10.8.239    # 对于发送端为远端的IP，对于接收端建议配置为0.0.0.0
  socket_port: 60082             # 对于发送端为远端的端口，对于接收端为监听的端口
  socket_buffer_size: 4194304    # 默认即可
  max_msg_size: 15360            # 不允许大于63KByte,默认为32KByte
  timeout_ms: 150                # 接收超时，默认150ms
  send_control:
    send_control_enable: true           # 如果发送点云时，该选项建议开启
    send_control_thres: 262144          # 默认为256KB
    send_control_ms: 3                  # 默认为2ms,建议 <= 4
    send_control_compress_enable: true # 如果发送点云时，该选项建议开启
  receive_control:
    receive_io_parallel_enable: true       # 如果发送点云，建议开启
    receive_io_parallel_degree: 4          # 根据机器设备配置，例如4核,建议配置不超过4
    receive_process_parallel_enable: true # 如果发送点云，建议开启
native:
  custom_method: NativeBytesSDK3_0
  objects: true
  attention_objects: true
  object_supplement: false
  freespace: false
  curbs: false
  lanes: false
  axis_lidar_pose: false
  point_cloud: false
  global_car_pose: false
```

由于该通信方式是利用socket_udp的方式进行通信。因此，在使用时应参照以下情况对该配置文件进行调整：

1. **若本机为发送端**，则需要将socket字段下的socket_address调整为远端的ip。
2. **若本机为接收端**，则需将socket字段下的socket_address调整为0.0.0.0。

   注意：NativeBytes通信方式支持发送、接收，其中接收端的可执行文件在编译后的/release/build/receiver_demo_beta目录下，终端执行可执行文件打印出接收结果进行验证。

至此，通信方式为针对SDK_3_0协议的NativeBytes方式的配置已经结束。在编译完成整个SDK后，运行可执行文件rs_sdk_demo即可在完成感知后调用通信模块采用NativeBytes方式发送适配SDK_3_0的数据。此时在终端上会显示 send successed 的字样。

2.3.4.3 针对SDK_3_1协议的版本

如下所示为该方式的配置文件的内容：（文件路径为：**RS_CONFIG_ROOT_DIR/system_config/communication_config/native_bytes/config_3_1.yaml**）

```
#* General *#
```

```
general:
  device_id: 0
socket:
  socket_address: 10.10.8.239    # 对于发送端为远端的IP，对于接收端建议配置为0.0.0.0
  socket_port: 60082             # 对于发送端为远端的端口，对于接收端为监听的端口
  socket_buffer_size: 4194304    # 默认即可
  max_msg_size: 32768            # 不允许大于63KByte,默认为32KByte
  timeout_ms: 150                # 接收超时，默认150ms
  send_control:
    send_control_enable: true           # 如果发送点云时，该选项建议开启
    send_control_thres: 262144          # 默认为256KB
    send_control_ms: 3                  # 默认为2ms,建议 <= 4
    send_control_compress_enable: true # 如果发送点云时，该选项建议开启
  receive_control:
    receive_io_parallel_enable: true       # 如果发送点云，建议开启
    receive_io_parallel_degree: 4          # 根据机器设备配置，例如4核,建议配置不超过4
    receive_process_parallel_enable: true # 如果发送点云，建议开启
native:
  custom_method: NativeBytesSDK3_1
  send_point_cloud: true
  send_attention_objects: true
  send_freespace: true
  send_lane: true
  send_roadedge: true
  send_sematic_indices: true
```

由于该通信方式是利用socket_udp的方式进行通信。因此，在使用时应参照以下情况对该配置文件进行调整：

1. **若本机为发送端**，则需要将socket字段下的socket_address调整为远端的ip。
2. **若本机为接收端**，则需将socket字段下的socket_address调整为0.0.0.0。

   注意：NativeBytes通信方式支持发送、接收，其中接收端的可执行文件在编译后的/release/build/receiver_demo_beta目录下，终端执行可执行文件打印出接收结果进行验证。

至此，通信方式为针对SDK_3_1协议的NativeBytes方式的配置已经结束。在编译完成整个SDK后，运行可执行文件rs_sdk_demo即可在完成感知后调用通信模块采用NativeBytes方式发送SDK_3_1的感知结果数据。此时在终端上会显示 send successed 的字样。

2.3.5 Protobuf 配置

如下所示为该方式的配置文件的内容：（文件路径为：**RS_CONFIG_ROOT_DIR/system_config/communication_config/proto/config.yaml**）

```
#* General *#
general:
  device_id: 0
socket:
  socket_address: 10.10.8.239    # 对于发送端为远端的IP，对于接收端建议配置为0.0.0.0
  socket_port: 60082             # 对于发送端为远端的端口，对于接收端为监听的端口
  socket_buffer_size: 4194304    # 默认即可
  max_msg_size: 32768            # 不允许大于63KByte,默认为32KByte
  timeout_ms: 150                # 接收超时，默认150ms
  send_control:
```

```
    send_control_enable: true           # 如果发送点云时，该选项建议开启
    send_control_thres: 262144          # 默认为256KB
    send_control_ms: 3                  # 默认为2ms,建议 <= 4
    send_control_compress_enable: true # 如果发送点云时，该选项建议开启
  receive_control:
    receive_io_parallel_enable: true       # 如果发送点云，建议开启
    receive_io_parallel_degree: 4          # 根据机器设备配置，例如4核,建议配置不超过4
    receive_process_parallel_enable: true # 如果发送点云，建议开启
native:
  custom_method: Proto
  send_objects: true
  send_attention_objects: true
  send_object_supplement: true
  send_freespace: false
  send_lane: false
  send_roadedge: false
  send_pose: false
  send_point_cloud: false
```

由于该通信方式是利用socket_udp的方式进行通信。因此，在使用时应参照以下情况对该配置文件进行调整：

1. **若本机为发送端**，则需要将socket字段下的socket_address调整为远端的ip。
2. **若本机为接收端**，则需将socket字段下的socket_address调整为0.0.0.0。

   注意：Proto通信方式支持发送、接收，其中接收端的可执行文件在编译后的/release/build/receiver_demo_beta目录下，终端执行可执行文件打印出接收结果进行验证。

至此，通信方式为针对SDK_3_1协议的Proto方式的配置已经结束。在编译完成整个SDK后，运行可执行文件rs_sdk_demo即可在完成感知后调用通信模块采用Proto方式发送SDK_3_1的感知结果数据。此时在终端上会显示 send successed 的字样。

2.3.6 V2R配置

如下所示为该方式的配置文件的内容：（文件路径为：**RS_CONFIG_ROOT_DIR/system_config/communication_config/v2r/config.yaml**）

```
general:
  device_id: 0
socket:
  socket_address: 10.10.8.239    # 对于发送端为远端的IP，对于接收端建议配置为0.0.0.0
  socket_port: 60082             # 对于发送端为远端的端口，对于接收端为监听的端口
  socket_buffer_size: 4194304    # 默认即可
  max_msg_size: 15360            # 不允许大于63KByte,默认为32KByte
  timeout_ms: 150                # 接收超时，默认150ms
  send_control:
    send_control_enable: true           # 如果发送点云时，该选项建议开启
    send_control_thres: 262144          # 默认为256KB
    send_control_ms: 3                  # 默认为2ms,建议 <= 4
    send_control_compress_enable: true # 如果发送点云时，该选项建议开启
  receive_control:
    receive_io_parallel_enable: true       # 如果发送点云，建议开启
    receive_io_parallel_degree: 4          # 根据机器设备配置，例如4核,建议配置不超过4
    receive_process_parallel_enable: true # 如果发送点云，建议开启
```

```
native:
  custom_method: RobosenseV2R
v2r:
   method: "TCP_SERVER"                    # "UDP_SENDER" / "TCP_CLIENT" /
"TCP_SERVER"
   version: "ROBOSENSE_V2R_V1.6"           #  Robosense V2R 版本:
"ROBOSENSE_V2R_V1.4"/"ROBOSENSE_V2R_V1.5"/"ROBOSENSE_V2R_V1.6"
   center: "CENTER"                        # "CENTER" / "ANCHOR"
   heading:
      velocity_heading_th: 0.5 # m/s
      acceleration_heading_th: 0.05 # m/s^2
      acceleration_valid: false #
```

当采用V2R通信方式时，可以选择利用socket_udp / socket_tcp_client / socket_tcp_server 三种方式中的一种进行通信。选用的方式是在配置表v2r字段下的method中将内容填写为所需要的通信方式。其中，

1. socket_udp方式：Robosense Lidar 作为UDP的发送端
2. socket_tcp_client 方式：Robosense Lidar作为客户端
3. socket_tcp_server 方式：Robosense Lidar作为服务器端

另外，v2r字段下的version表示的是不同的 Robosense V2R协议版本。目前支持的版本在配置表中以注释的形式列出。具体的协议差异可以参考4.4节中的介绍。

不论采用上述3种通信方式中的哪一种，在使用时都应参照以下情况对该配置文件进行调整：

不论采用上述3种通信方式中的哪一种，在使用时都应参照以下情况对该配置文件进行调整：

1. **若本机为发送端**，则需要将socket字段下的socket_address调整为远端的ip。
2. **若本机为接收端**，则需将socket字段下的socket_address调整为0.0.0.0。

   注意：V2R通信方式支持发送、接收，其中接收端的可执行文件在编译后的/release/build/receiver_demo_beta目录下，终端执行可执行文件打印出接收结果进行验证。

至此，通信方式为V2R的配置已经结束。在编译完成整个SDK后，运行可执行文件rs_sdk_demo即可在完成感知后调用通信模块发送适配对应V2R协议的数据。此时在终端上会显示 send successed 的字样。

2.3.7 Websocket配置

如下所示为该方式的配置文件的内容：（文件路径为：**RS_CONFIG_ROOT_DIR/system_config/communication_config/websocket/config.yaml**）

```
#* General *#
general:
  device_id: 0
common:
  socket_parallel_bufer_size: 2
  socket_listen_port: 8081 # websocket listen port
  web_update_frame_gap: 1 # websocket update frame gap if = 5 means 2Hz
enables:
  object:
    cube: true
    box: false
    polygon: false
    polyhedral: false
```

```
    tracking_point: false
    trajectory: false
    velocity_dir: true
    box_info: true
    label_info: true
    track_info: true
    gps_info: true
  attention_object:
    polygon: false
    polyhedral: true
  lane:
    lanes: false
  curb:
    curbs: false
  freespace:
    freespaces: false
  pointcloud:
    pointclouds: true
  map:
    maps: false
  roi:
    rois: true
objects: # Support Object Type and Color Attribution
    - type: unknown
      typeid: 0
      color: [153, 128, 153, 76]
    - type: pedestrian
      typeid: 1
      color: [255, 255, 0, 76]
    - type: bikelist
      typeid: 2
      color: [0, 255, 0, 76]
    - type: car
      typeid: 3
      color: [0, 255, 255, 76]
    - type: truck
      typeid: 4
      color: [0, 255, 255, 76]
    - type: ultra-track
      typeid: 5
      color: [0, 255, 255, 76]
attention_objects:
  color: [255, 0, 255, 255] # attention_object颜色
object_attachs:
  track_points:
    color: [0, 255, 0, 128]
    attach_object: false # 不关联Object颜色
  trajectory:
    color: [255, 0, 0, 128]
    attach_object: true # 关联物体颜色
  label_text:
    color: [255, 255, 255, 255]
    attach_object: false # 白色，不关联Object颜色
    text_size: 12
  track_text:
    color: [255, 255, 255, 255]
    attach_object: false # 白色，不关联Object颜色
    text_size: 12
```

```
  box_text:
    color: [255, 255, 255, 255]
    attach_object: false # 白色, 不关联Object颜色
    text_size: 12
  gps_text:
    color: [255, 255, 255, 255]
    attach_object: false # 白色, 不关联Object颜色
    text_size: 12
  velocity_dir:
    color: [0, 255, 0, 255]
    attach_object: false # 不关联Object颜色
  acc_dir:
    color: [0, 255, 0, 255]
    attach_object: false # 不关联Object颜色
lanes:
  color: [0, 255, 0, 255] # [56, 56, 56, 255]
curbs:
  color: [255, 0, 0, 255] # [56, 56, 56, 255]
freespaces:
  color: [77, 77, 77, 255]
transform:
  matrix: [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]
lidar_grids:
  grid_circle_ranges: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250]
# circle grid 的半径
  grid_circle_color: [160, 160, 164, 255]
pointcloud_color_map: # pointcloud color map
  color_map_enable: true
  color_map_type: "rviz" #
"autumn"/"bone"/"jet"/"winter"/"rainbow"/"ocean"/"summer"/"spring"/"cool"/"hsv"
/"pink"/"hot"/"parula"/"rviz"
  color_map_factors: ["intensity"] # "x" / "y" / "z" / "intensity"
  color_map_lows: [0] #
  color_map_highs: [128] #
  color_map_alphas: [1.0] #
  color_map_invert: false #
  color_map_label: false # always is false now
  default_color: [230, 230, 230, 255]
  point_size: 1.0
map_color_map: # map  pointcloud color map
  color_map_enable: true
  color_map_type: "rviz" #
"autumn"/"bone"/"jet"/"winter"/"rainbow"/"ocean"/"summer"/"spring"/"cool"/"hsv"
/"pink"/"hot"/"parula"/"rviz"
  color_map_factors: ["intensity"] # "x" / "y" / "z" / "intensity"
  color_map_lows: [0] #
  color_map_highs: [128] #
  color_map_alphas: [1.0] #
  color_map_invert: false #
  color_map_label: false # always is false now
  default_color: [230, 230, 230, 255]
  point_size: 1.0
roi_color_map:
  - - roi_type: 0
      filter_type: 0
      color: [0, 0, 255, 128] #
    - roi_type: 0
      filter_type: 1
```

```
      color: [0, 0, 255, 128] #
    - roi_type: 0
      filter_type: 2
      color: [0, 0, 255, 128] #
map_origin:
  lat: 0
  lon: 0
  alt: 0
fill_maps:
  -
"/home/sti/Desktop/20211008SDK_modify_sender_and_merge_communication_method/rs_
sdk_3.1_release/config/system_config/communication_config/websocket/fill_maps/f
inalmap.pcd" # map fill path
fill_rois:
  -
"/home/sti/Desktop/20211008SDK_modify_sender_and_merge_communication_method/rs_
sdk_3.1_release/config/system_config/communication_config/websocket/fill_rois/r
oi.yaml" # roi fill path
```

当采用该种通信方式时，Robosense Lidar将作为服务器主动向客户端推送经过websocket协议处理后的感知消息。其中 common 字段下的 socket_listen_port 为监听端口。一般情况下，该配置表的参数无需调整，直接使用即可。如确实有调整的需求，请联系 Robosense技术支持。

在编译完成整个SDK后，运行可执行文件rs_sdk_demo即可在完成感知后调用通信模块发送适配websocket协议的数据。此时在终端上会显示 send successed 的字样。

提醒，websocket通信方式仅支持发送，不支持接收。若有需要，用户需根据该通信方式的源代码自行实现接收功能。若强行在新的终端下调用工程根目录下test文件夹中的receiver.cpp，则接收端在初始化的时候将会直接报错退出。终端会打印出 "SDK 3.1 Not Support Robosense Websocket Data Receiver" 的报错字样。

2.4 通信实现源代码说明

通信方式	代码文件	备注
ROS	RS_COMM_ROOT_DIR/include/rs_perception/communication/internal/robosense_ros/*.h 或 RS_COMM_ROOT_DIR/src/internal/robosense_ros/*.cpp	其中*表示任意字符串
ROS2	RS_COMM_ROOT_DIR/include/rs_perception/communication/internal/robosense_ros2/*.h 或 RS_COMM_ROOT_DIR/src/internal/robosense_ros2/*.cpp	其中*表示任意字符串
Native / NativeBytes / V2R	RS_COMM_ROOT_DIR/include/rs_perception/communication/internal/robosense_native/*.h 或 RS_COMM_ROOT_DIR/src/internal/robosense_native/*.cpp	其中*表示任意字符串
Websocket	RS_COMM_ROOT_DIR/include/rs_perception/communication/internal/robosense_websocket/*.h 或 RS_COMM_ROOT_DIR/src/internal/robosense_websocket/*.cpp	其中*表示任意字符串

3. API的使用方法

本节将介绍通信模块的外部接口以及如何调用这些接口。

3.1 接口介绍

通信模块的**外部接口定义**位于**RS_COMM_ROOT_DIR/communication/include/rs_perception/communication/external** 中。其中 sender.h 对应于发送端，receiver.h 对应于接收端。下面将分别进行介绍。

3.1.1 发送端接口介绍

发送端的接口一共有3个，分别为 *init*，*send*，*registerErrorComm*。

注意：通信模块发送端的使用依赖于感知模块的输出结果。因此在调用发送端前需确认感知模块是否正常启用。

init：发送端初始化接口。该接口传入的参数已经加载好的yaml node（yaml node 的细节信息即为 第2节 中提到的配置信息）。接口示例如下：

```
void init(const RsYamlNode& config_node);  // config_node即为需要传入的yaml node数据。
```

send：发送端消息发送接口。**该接口传入的参数是感知模块输出的消息**。消息类型为：RsPerceptionMsg::Ptr &。在完成发送后，该接口会以 COMMUNICATION_ERROR_CODE 的形式返回 Success。接口的示例如下：

```
COMMUNICATION_ERROR_CODE send(const RsPerceptionMsg::Ptr& msg_ptr);
```

registerErrorComm：发送端错误代码注册接口。接口的示例如下：（该接口暂时闲置）

```
void registerErrorComm(const CommunicaterErrorCallback& cb);
```

3.1.2 接收端接口介绍

接收端的接口一共有3个，分别为*init*，*registerRcvComm*，*registerErrorComm*。

init：接收端初始化接口。该接口传入的参数已经加载好的yaml node（yaml node的细节信息即为 第2节 中提到的配置信息）。接口的示例如下：

```
void init(const RsYamlNode &config_node);
```

registerRcvComm：注册回调函数的接口。**该接口传入的是以函数封装的形式存在的函数**。其中被封装的函数的返回值应为void，形参类型应为 RsPerceptionMsg::Ptr &。接口的示例以及传入参数的示例如下：

```
// 接口的示例
void registerRcvComm(const PerceptReceiveCallback &cb);  //
PerceptReceiveCallback 为传入参数的类型

// 传入参数的示例
std::function<void(const RsPerceptionMsg::Ptr &)>;  // registerRcvComm实际传入的
数据格式。其中 RsPerceptionMsg 为Robosense感知消息的数据格式。
```

registerErrorComm：接收端错误代码注册接口。接口的示例如下：（该接口暂时闲置）

```
void registerErrorComm(const CommunicaterErrorCallback &cb);
```

3.2 接口调用方法

本小节将介绍如何**在ros环境下书写调用通信模块外部接口的文件**。因此，**ros的头文件必须包含**。

```
#include <ros/ros.h>
```

另外，在本小节中**约定**：调用发送端接口的文件记为 *sender_call*，调用接收端接口的文件记为 *receiver_call*。

在书写 *sender_call* 的时候，需要包含 **RS_COMM_ROOT_DIR/communication/include/rs_perception/communication/external/sender.h** ，即：

```
// 书写 sender_call 的时候需要include的内容
#include
"RS_COMM_ROOT_DIR/communication/include/rs_perception/communication/external/se
nder.h"

// 也可以参考如下形式
#include "rs_perception/communication/external/sender.h"
```

在书写 *receiver_call* 的时候，需要包含 **RS_COMM_ROOT_DIR/communication/include/rs_perception/communication/external/receiver.h**，即：

```
// 书写 reveiver_call 的时候需要include的内容
#include
"RS_COMM_ROOT_DIR/communication/include/rs_perception/communication/external/re
ceiver.h"

// 也可以参考如下形式
#include "rs_perception/communication/external/receiver.h"
```

3.2.1 发送端接口调用方法

注意：发送端的使用依赖于感知模块的输出结果。因此，在书写 *sender_call* 时，**还需要包含感知模块相关的头文件以保证感知模块可以正常被调用**。同时，为方便调试，也可以同时包含SDK中有关RVIZ显示的头文件。与RVIZ显示相关的功能的根目录与RS_COMM_ROOT_DIR同级，目录名为 rviz_display。用户可根据实际需求进行了解。

```
#include "rs_perception/perception/external/perception.h"  // Robosense 感知模块的相关头文件，具体路径可根据用户具体情况进行更改，此处为示例

#include "rviz_display/external/rviz_display.h" // Robosense RVIZ显示模块的相关头文件，具体路径可根据用户具体情况进行更改，此处为示例。
```

在调用时，用户首先需要在全局范围声明2个指针：① 感知模块的指针（记为 perception_ptr ），② 通信模块发送端的指针（记为 sender_ptr ）。（如有需要，可以再声明一个RVIZ显示的指针）

```
static Perception::Ptr perception_ptr; // 感知模块的指针
static Sender::Ptr sender_ptr; // 通信模块发送端的指针

// 以下为可选项:
static RvizDisplay::Ptr rviz_display_ptr; // RVIZ显示的指针
```

当相关指针初始化完毕以后，用户需要进行如下操作：

① 在 main 函数当中完成 yaml 配置文件的加载。

② 通过 “perception” 字段定位到感知模块的相关配置信息，利用这些信息对 perception_ptr 进行初始化。

③ 通过 “communication” 字段定位到通信模块的相关配置信息，把这些信息传入 sender_ptr 中的 init 接口，从而完成对 sender_ptr 的初始化。

④ 初始化一个 ros 的 subscriber 节点来订阅原始点云的信息。

```
// 加载 yaml配置文件。
std::string config_path = std::string(PROJECT_PATH) + "/config";  // 此处的 std::string(PROJECT_PATH) + "/config" 即为本文档中的 RS_CONFIG_ROOT_DIR。
RsYamlNode node = rsConfigParser(config_path);

// 初始化 perception_ptr
RsYamlNode perception_node;
rsYamlSubNode(node, "perception", perception_node);
perception_ptr.reset(new Perception);
perception_ptr->init(perception_node);

// 初始化 sender_ptr
RsYamlNode communication_node;
rsYamlSubNode(node, "communication" ,communication_node);
sender_ptr.reset(new Sender);
sender_ptr->init(communication_node);

// 初始化 ros subscriber节点
```

```
ros::Subscriber sub_fullscan = node_ptr->subscribe(topic, 1, fullscanCallback);
// fullscanCallback为ros的回调函数
```

yaml配置文件加载后形态展示（限于篇幅，仅展示与通信模块有关的配置，以 Native 通信方式为例）

```
communication:
  0:
    method: Native
    config:
      general:
        device_id: 0
      socket:
        socket_address: 10.10.8.239
        socket_port: 60082
        socket_buffer_size: 4194304
        max_msg_size: 32768
        timeout_ms: 150
        send_control:
          send_control_enable: true
          send_control_thres: 262144
          send_control_ms: 3
          send_control_compress_enable: true
        receive_control:
          receive_io_parallel_enable: true
          receive_io_parallel_degree: 4
          receive_process_parallel_enable: true
      native:
        send_point_cloud: true
        send_attention_objects: true
        send_freespace: true
        send_lane: true
        send_roadedge: true
        send_sematic_indices: true
```

在上述 subscriber 节点的回调函数中，**用户需先调用 perception_ptr 中的 perception 接口来获取感知结果。然后再将感知结果通过 sender_ptr 中的 send 接口进行发送**。（若用户需要通过RVIZ进行显示，可以将感知结果通过显示指针中的display接口进行显示。）

```
// 调用感知模块来获取感知结果
perception_ptr->perception(msg_ptr);  // msg_ptr 的类型为 RsPerceptionMsg

// 调用通信模块发送感知结果
sender_ptr->send(msg_ptr);

// 该项为可选项：调用RVIZ显示模块进行显示感知结果
rviz_display_ptr->display(msg_ptr);
```

3.2.2 接收端接口调用方法

不同于发送端，接收端不依赖于感知模块的输出结果。因此不需要在书写 *receiver_call* 中包含感知模块的相关头文件。**但是，为了方便接收端结果的展示，需要在文件中包含SDK中有关RVIZ显示的头文件**。

```
#include "rviz_display/external/rviz_display.h" // Robosense RVIZ显示模块的相关头文件，具体路径可根据用户具体情况进行更改，此处为示例。
```

在 main 函数中，用户需要声明2个指针：① 通信模块接收端的指针（记为 receiver_ptr ），② RVIZ显示的指针（记为 rviz_display_ptr ）。

```
// 声明并初始化通信模块接收端指针
Receiver::Ptr receiver_ptr(new Receiver);

// 声明并初始化RVIZ显示的指针
RvizDisplay::Ptr rviz_display_ptr(new RvizDisplay);
```

完成指针的声明和初始化以后，用户还需进行如下操作：

① 加载 yaml 配置文件。

② 通过 “rviz” 字段定位到RVIZ显示的相关配置信息。把这些信息送入 rviz_display_ptr 中的 init 接口进行初始化。

③ 通过 “communication” 字段定位到通信模块的相关配置信息，把这些信息传入 receiver_ptr 中的 init 接口，从而完成对 receiver_ptr 的初始化。

注意，在接收端文件中，不再需要初始化一个 ros 的 subscriber 节点。

```
// 加载 yaml配置文件。
std::string config_path = std::string(PROJECT_PATH) + "/config";  // 此处的 std::string(PROJECT_PATH) + "/config" 即为本文档中的 RS_CONFIG_ROOT_DIR。
RsYamlNode node = rsConfigParser(config_path);

// 初始化 rviz_display_ptr
RsYamlNode perception_node, rviz_node;
rsYamlSubNode(node, "perception", perception_node);
rsYamlSubNode(perception_node, "rviz", rviz_node);
rviz_display_ptr->init(rviz_node);

// 初始化 receiver_ptr
RsYamlNode communication_node;
rsYamlSubNode(node, "communication" ,communication_node);
receiver_ptr->init(communication_node);
```

初始化完成后，**用户需要通过lambda函数的形式完成对rviz_display_ptr中display接口的封装**。该接口（即display接口）的输入参数形式为：RsPerceptionMsg::Ptr &。封装完成后，将封装好的变量作为输入送入 receiver_ptr 中的 registerRcvComm 接口中。示例如下：

```
// 用lambda函数的形式完成封装
auto func = [rviz_display_ptr](const RsPerceptionMsg::Ptr& msg_ptr)
{rviz_display_ptr->display(msg_ptr);};
// 调用registerRcvComm激活回调函数
receiver_ptr->registerRcvComm(func);
```

至此，*sender_call* 和 *receiver_call* 的核心部分代码已经完成。用户在完善代码的其他细节之后，编译代码即可实现自主调用通信模块进行收/发信息。

再次提醒， websocket 通信方式仅支持发送，不支持接收。若有需要，用户需根据该通信方式的源代码自行实现接收功能。否则接收端在初始化的时候将会直接报错退出。同时会打印出 "SDK 3.1 Not Support Robosense Websocket Data Receiver"的字样。

4. 通信协议

4.1 ROS通信协议

基于ROS消息方式进行通信，通过ROS的消息订阅和投递机制进行消息发送和接收，发送消息内容详细参考ROS消息文件: **RS_COMM_ROOT_DIR/message/ros/ros_msg_ws/src/msg/*.msg**，其中，消息内容和ROS消息的对应关系如下表格所示，但是，ROS通信采用整体发送感知结果的发送进行通信，所以，实际进行通信时，发送的ROS消息为:**RsPerceptionMsg.msg**

消息类型	ROS/ROS2消息	备注
ROBO_MSG_OBJECT/ROBO_MSG_ATT_OBJECT	Object.msg	
ROBO_MSG_FREESPACE	FreeSpaceInfos.msg	
ROBO_MSG_LANE	Lanes.msg	
ROBO_MSG_CURB	Curbs.msg	
ROBO_MSG_POINT	Point5f.msg	
ROBO_MSG_AXISSTATUS	Pose.msg	
ROBO_MSG_GLOBALCAR_POSE	Pose.msg	
ROBO_MSG_AXISLIDAR_POSE	Pose.msg	

注意: 在Ros消息中定义的字段内容基本同Robosense Perception Common模块中定义的数据结构字段名称和含义基本一致，这个不再重复，如果存在疑问，联系Robosense技术支持或查看ros源代码:

4.2 ROS2通信协议

基于ROS2消息方式进行通信，通过ROS2的消息订阅和投递机制进行消息发送和接收，发送消息内容详细参考ROS2消息文件:
RS_COMM_ROOT_DIR/message/ros2/ros2_msg_ws/src/perception_ros2_msg/msg/*.msg，其中，消息内容和ROS2消息的对应关系如下表格所示，但是，ROS2通信采用整体发送感知结果的发送进行通信，所以，实际进行通信时，发送的ROS2消息为:**RsPerceptionMsg.msg**

消息类型	ROS/ROS2消息	备注
ROBO_MSG_OBJECT/ROBO_MSG_ATT_OBJECT	Object.msg	
ROBO_MSG_FREESPACE	FreeSpaceInfos.msg	
ROBO_MSG_LANE	Lanes.msg	
ROBO_MSG_CURB	Curbs.msg	
ROBO_MSG_POINT	Point5f.msg	
ROBO_MSG_AXISSTATUS	Pose.msg	
ROBO_MSG_GLOBALCAR_POSE	Pose.msg	
ROBO_MSG_AXISLIDAR_POSE	Pose.msg	

注意: 在Ros2消息中定义的字段内容基本同Robosense Perception Common模块中定义的数据结构字段名称和含义基本一致，这个不再重复，如果存在疑问，联系Robosense技术支持或查看ros2源代码:

4.3 Native 通信协议

Native通信协议基于UDP发送，由于存在一帧数据长度超过一个报文的载荷长度的问题，所以，一帧感知数据会**"分割"**为多个报文进行发送，所以，该类通信发送的数据发送报文的格式为: Header + Content，其中Header定义如**4.2.1**表格所述；Content为基于cereal序列化后的数据(LZ4压缩后的序列化数据)分片；对于接收端，基于 msgFrameId/msgTotalCnt/msgTotalLen/msgLocalCnt/msgLocalLen/msgIndex等进行数据排序和数据完整性检查.

4.3.1 消息头

字段名称	大小(Byte)	字段说明	备注
msgVersion	2	消息格式版本	当前默认版本为0x7E8E
msgType	2	消息类型	固定为:0xFFFE
deviceId	4	设备编号	默认为1
msgTimestampS	8	消息时间戳,单位秒(s)	
msgFrameId	4	消息帧编号	>= 0, 从设备感知开始，每处理一帧+1
msgTotalCnt	4	一帧感知结果中，对应消息类型的总消息(UDP 报文)个数	不同的消息类型，为了在复杂度和效率之间平衡，采用了不同的组织方式，所以，对于不同消息类型封装的UDP报文数不同
msgTotalLen	4	发送的数据长度	压缩时，为压缩后的数据长度；非压缩时，为原始数据的长度
msgTotalUncompressLen	4	非压缩的原始数据长度	压缩时，为原始数据的长度,非压缩时为0
msgLocalCnt	2	UDP报文中包含该类型消息的内容的数量	对于字节固定的消息，为了在延迟/吞吐效率/复杂度方面平衡，可能一个UDP报文包含多个内容
msgLocalLen	2	消息内容的长度	>= 0，特别地，长度为0对应空消息
msgIndex	2	同一帧相同类型消息数据报文索引	>=0, 从0开始编号，未填充有效值，默认填充0x0000
msgTotalFragment	2	消息分片数量	>= 0， 默认填充0x0000
msgFragmentIndex	2	消息分片编号	>=0, 从0开始编号，未填充有效值，默认填充0x0000
msgCheck16	2	包含消息头和消息内容的CRC16检查和	未填充有效值，默认填充0xFFFF
msgRes0	4	保留	未填充有效值，默认填充0x0000

4.3.2 消息内容

native协议下发送的内容如下表所示。其中表4.3.2-1为固定发送的消息内容。表4.3.2-2为根据配置文件选择发送的内容。

字段名称	注释
frame_id	类似/middle_lidar /left_lidar这种，为了区分每个雷达的每一帧
timestamp	每一帧数据的时间戳
globale pose	全局pose
gps origin	GPS原点信息
status pose map	坐标轴的pose
status	该帧的坐标系
valid indices	该帧有效点的索引
objects	该帧结果中的所有前景物体
device id	客户的设备id

表 4.2.2-1 固定发送的内容3

字段名称	注释
scan ptr	点云
attention objects	该帧结果中的一度目标
freespace	该帧结果中的可行驶区域
lanes	该帧结果中的车道线
roadedge	该帧结果中的路沿
sematic	该帧结果中的地面点/非地面点/背景点的索引

表 4.3.2-2 可选发送的内容

同时，在发送消息是可选LZ4无损压缩的方式对发送的内容进行压缩。(可在配置文件中配置该选项)

4.4 NativeBytes 通信协议

NativeBytes通信协议基于UDP发送，由于存在一帧数据长度超过一个报文的载荷长度的问题，所以，一帧感知数据会**"分割"**为多个报文进行发送。因此，该类通信发送的数据发送报文的格式为: Header + Content。根据SDK版本不同，Header和Content会有所不同。下面将分别介绍针对SDK2.x协议的Header和Content、针对SDK3.0协议的Header和Content以及针对SDK3.1的Header和Content。

4.4.1 SDK2.x通信协议

4.4.1.1 消息头(Header)

以下为针对SDK2.x的消息头。备注为空的三项表示消息内容会随着工程的运行而改变。

字段名称	大小(Byte)	字段说明	备注
msgTimestampMs	8	消息时间戳，单位(ms)	
msgVersion	2	消息格式版本	当前默认版本为0x0001
msgType	2	消息类型	
deviceId	4	设备编号	
msgFrameId	4	消息帧编号	>= 0, 从设备感知开始，每处理一帧+1
msgTotalCnt	4	一帧感知结果中，对应消息类型的总消息(UDP报文)个数	不同的消息类型，为了在复杂度和效率之间平衡，采用了不同的组织方式，所以，对于不同消息类型封装的UDP报文数不同
msgLocalCnt	4	UDP报文中包含该类型消息的内容的数量	对于字节固定的消息，为了在延迟/吞吐效率/复杂度方面平衡，可能一个UDP报文包含多个内容
msgLocalLen	2	消息内容的长度	>= 0，特别地，长度为0对应空消息
msgCheck16	2	包含消息头和消息内容的CRC16检查和	未填充有效值，默认填充0xFFFF
msgRes1[8]	8	保留	未填充有效值，默认填充0x00

4.4.1.2 消息内容(Content)

根据配置文件，用户可选择发送objects / attention_objects / object_supplement / freespace / lanes / curbs / global_car_pose / non_ground_indices / ground_indices / background_indices /point_cloud 这些信息中的一项或全部。下面将介绍上述各消息的序列化顺序。

注意：每个消息头后面跟的消息内容只会是下列消息内容中的1项！其中对objects / attention_objects / point_cloud / non_ground_indices / ground_indices / background_indices的特殊处理将在对应小节中进行说明。

4.4.1.2.1 objects / attention_objects

一帧感知结果当中会有**N个**object / attention object (N >= 0)。以下为1个object中含有的内容。**几点注意事项如下**:

1. **SDK2.X与SDK3.1在定义object的内容的时候有些许不同。这些不同之处在下表中以加粗的形式体现并在注释中进行了说明。**
2. **只有用户将配置表中的object_supplement置为true的时候，该object的Supplement Info才会随Core Info一起一并发送。否则只会发送object的Core Info。**
3. **在发送的时候，1个Header后面只搭配1个object。因此，与object有关的数据帧会有N个。**

序号	字段名称	数据类型	字节长度(byte)	注释
Core Info				
1	timestamp	double	8	每个object时间戳信息，单位：秒
2	priority_id	int32	4	优先级ID，根据障碍物的方位、距离、车道、类别、速度大小等信息来排序，ID越小优先级越高
3	exist_confidence	float	4	障碍物存在的概率
4	center	(float,3)	3*4	中心点，表示目标中心点坐标（x,y,z），单位：米
5	**center_cov**	**(float,3)**	**(3+6)*4**	**目标中心点的不确定度。 在SDK2.X中，该字段是一个3 * 3矩阵。在SDK3.1中，该字段是一个3 * 1的向量。因此SDK3.1的3x1的向量赋值给SDK2.x的矩阵的前3个元素，其他元素用0填充。**
6	size	(float,3)	3*4	框尺寸，表示目标框尺寸（长，宽，高），单位：米
7	**size_cov**	**(float,3)**	**(3+6)*4**	**目标框尺寸的不确定度。 在SDK2.X中，该字段是一个3 * 3矩阵。在SDK3.1中，该字段是一个3 * 1的向量。因此SDK3.1的3x1的向量赋值给SDK2.x的矩阵的前3个元素，其他元素用0填充。**
8	direction	(float,3)	3*4	框朝向，归一化三维向量，定义为框长轴的朝向

序号	字段名称	数据类型	字节长度(byte)	注释
9	**direction_cov**	**(float,3)**	**(3+6)*4**	**框朝向的不确定度。 在SDK2.X中，该字段是一个3 * 3矩阵。在SDK3.1中，该字段是一个3 * 1的向量。因此SDK3.1的3x1的向量赋值给SDK2.x的矩阵的前3个元素，其他元素用0填充。**
10	type	int32	4	目标类别，分为7类：CONE (圆柱体、锥桶等)，PED (行人)，BIC (骑行者)，CAR (小车)，TRUCK_BUS (大车)，ULTRA_VEHICLE (拖车)，UNKNOWN （未知类别）
11	type_confidence	float	4	目标类别置信度
12	attention_type	int32	4	一度目标类型，NONE表示为非一度目标，ATTENTION表示一度目标
13	motion_state	int32	4	运动状态，表示是否属于动态目标，默认为未知UNKNOWN，动态目标为MOVING，可能运动的目标为MOVABLE，不可能运动的静态目标为STATIONARY
14	tracker_id	int32	4	跟踪ID，表示连续时间序列中目标的唯一标识ID。范围是0~10000，未被成功跟踪的目标ID为-1
15	age	double	8	跟踪生命期，表示目标被成功跟踪的时间长度，单位：秒
16	velocity	(float,3)	3*4	速度，三维向量，表示为目标速度在(x,y,z)方向的分量，单位：米/秒

序号	字段名称	数据类型	字节长度(byte)	注释
17	**velocity_cov**	**(float,3)**	**(3+6)*4**	**速度的不确定度。 在SDK2.X中，该字段是一个3 * 3矩阵。在SDK3.1中，该字段是一个3 * 1的向量。因此SDK3.1的3x1的向量赋值给SDK2.x的矩阵的前3个元素，其他元素用0填充。**
18	acceleration	(float,3)	3*4	加速度，三维向量，表示为目标加速度在(x,y,z)方向的分量，单位：米/秒平方
19	**acceleration_cov**	**(float,3)**	**(3+6)*4**	**加速度的不确定度。 在SDK2.X中，该字段是一个3 * 3矩阵。在SDK3.1中，该字段是一个3 * 1的向量。因此SDK3.1的3x1的向量赋值给SDK2.x的矩阵的前3个元素，其他元素用0填充。**
20	**angle_velocity**	**float**	**3*4**	**角速度，单位：弧度/秒。 在SDK2.X中，该字段是一个3 * 1矩阵。在SDK3.1中，该字段是一个标量。因此，SDK3.1的标量赋给SDK2.x的向量的第一个元素，其他元素用0填充。**
21	anchor	(float,3)	3*4	重心点，三维向量，表示目标分割点云重心点坐标，单位：米
22	nearest_point	(float,3)	3*4	最近点，三维向量，表示目标点云中距离雷达最近的点坐标，单位：米
23	left_point	(float,3)	3*4	轮廓中的最左点
24	right_point	(float,3)	3*4	轮廓中的最右点
25	is_supplement	bool	1	是否序列化supplement信息

序号	字段名称	数据类型	字节长度(byte)	注释
Supplement info				
26	polygon_size	int32	4	polygon的总个数
27	polygon	(float,3) * N	3 * N * 4	轮廓点，由二维鸟瞰投影面上包围目标点云的最小多边形的顶点组成，N为多边形顶点个数
28	**reference**	**int32**	**4**	**该字段在SDK2.X中是一个向量。在SDK3.1中不提供。因此SDK3.1兼容发送数字0来代替reference点。**
29	latent_types_size	int32	4	latent_types的总个数
30	latent_types	(float) * N	N*4	N维向量，表示目标属于每个类别的概率，N为分类类别个数
31	size_type	int32	4	形状类型，默认为SMALL，中型大小目标为MEDIUM，大型目标为LARGE
32	in_roi	bool	1	表示障碍物是否在roi区域以内
33	**angle_velocity_cov**	**float**	**(3+6)*4**	**角速度的不确定度。 在SDK2.X中，该字段是一个3 * 3矩阵。在SDK3.1中，该字段是一个3 * 1向量。因此SDK3.1的3x1的向量赋值给SDK2.x的矩阵的前3个元素，其他元素用0填充。**
34	**angle_acceleration**	**float**	**3*4**	**角加速度，单位：弧度/秒平方。 该字段在SDK2.X中是一个3 * 1矩阵。在SDK3.1中是一个标量。因此，SDK3.1的标量赋给SDK2.x的向量的第一个元素，其他元素用0填充。**

序号	字段名称	数据类型	字节长度(byte)	注释
35	**angle_acceleration_cov**	**float**	**(3+6)*4**	**角加速度的不确定度。在SDK2.X中，该字段是一个3 * 3矩阵。在SDK3.1中，该字段是一个3 * 1向量。因此SDK3.1的3x1的向量赋值给SDK2.x的矩阵的前3个元素，其他元素用0填充。**
36	trajectory_size	int32	4	trajectory的总个数
37	trajectory	(float,3) * N	3 * N * 4	跟踪运动轨迹，表示历史运动轨迹，N为序列缓存长度，可设置
38	history_velocity_size	int32	4	history_velocity的总个数
39	history_velocity	(float,3) * N	3 * N * 4	跟踪历史速度，N为序列缓存长度，可设置
40	history_type_size	int32	4	history_type的总个数
41	history_type	(int32) * N	N*4	跟踪历史类别，表示历史序列中目标的类别，N为序列缓存长度，可设置
42	gps_longtitude	double	8	GPS经度
43	gps_latitude	double	8	GPS纬度
44	gps_altitude	double	8	GPS高程

4.4.1.2.2 freespace

一帧感知结果当中只有1个freespace消息。而一个freespace消息中包含N个由 distance, yaw_angle, free_prob 组成的数据。以下为freespace的消息内容。

序号	字段名称	数据类型	字节长度(byte)	注释
1	distance	float	4	
2	yaw_angle	float	4	
3	free_prob	float	4	
…				
重复序号1-3共N-1次				
…				

4.4.1.2.3 lanes

一帧感知结果当中只有1个lanes消息。而这个消息由N个lane消息构成。以下是一个lanes消息的内容。

注意：SDK2.X与SDK3.1在定义lanes的内容的时候有些许不同。这些不同之处在下表中以加粗斜体的形式体现并在注释中进行了说明。

序号	字段名称	数据类型	字节长度(byte)	注释
1	lane_id	int32	4	车道线位置ID，左侧LEFT_EGO、LEFT_ADJACENT、LEFT_THIRD、LEFT_FOURTH、LEFT_FIFTH、LEFT_SIXTH，右侧RIGHT_EGO、RIGHT_ADJACENT、RIGHT_THIRD、RIGHT_FOURTH、RIGHT_FIFTH、RIGHT_SIXTH，以及其他OTHER
2	curve	(float,6)	6*4	4个三次曲线拟合参数以及起止点。顺序为x_start, x_end, a, b, c, d
3	***end_point***	***(float,2+1)*2**	***(2+1) * 2 * 4***	***车道线起始点和终止点。 在SDK2.1中，start和end为31向量。在SDK3.1中，start和end为21向量。因此用0来填充z的值。顺序为start.x, start.y, 0，end.x, end.y，0。***
4	***track_id***	***int32***	***4***	***在SDK2.X中是一个标量。在SDK3.1中不提供。因此用0来填充。***
5	confidence	float	4	检测置信度
…				
重复序号1-5共N-1次				
…				

4.4.1.2.4 curbs

一帧感知结果当中只有1个curbs消息。而这个消息由N个curb消息构成。以下是一个curbs消息的内容。

注意：SDK2.X与SDK3.1在定义curbs的内容的时候有些许不同。这些不同之处在下表中以加粗斜体的形式体现并在注释中进行了说明。

序号	字段名称	数据类型	字节长度(byte)	注释
1	roadedge_id	int32	4	道路边界位置ID，左侧LEFT，右侧RIGHT，以及未知UNKNOW
2	curve	(float,6)	6*4	4个三次曲线拟合参数以及起止点。顺序为x_start, x_end, a, b, c, d
3	***end_point***	***(float,2+1)2***	***(2+1) * 2 * 4***	***道路边界起始点和终止点。 在SDK2.1中，start和end为31向量。在SDK3.1中，start和end为21向量。因此用0来填充z的值。顺序为start.x, start.y, 0, end.x, end.y， 0。***
4	***track_id***	***int32***	***4***	***在SDK2.X中是一个标量。在SDK3.1中不提供。因此用0来填充。***
5	confidence	float	4	检测置信度
…				
重复序号1-5共N-1次				
…				

4.4.1.2.5 non_ground_indices

一帧感知结果中的点云消息的数据量是很大的。从而导致非地面点的索引数量也会很多。因此在发送非地面点索引的时候需要进行 **“分包”** 的处理。**有关该消息的注意事项如下：**

1. **一片非地面点的索引会被分成M个数据包。每个数据包跟随1个消息头。因此在发送非地面点的索引的时候，一共会有M个与非地面点的索引有关的消息头。**
2. **每个非地面点的索引数据包中的索引的个数N会因为雷达的型号而有所不同。**
3. **每个非地面点的索引数据包中的点的序列化顺序如下所示。**

序号	字段名称	数据类型	字节长度(byte)	注释
1	index	int32	4	一个非地面点的点索引
…				
重复N-1次，N 与雷达型号有关				
…				

4.4.1.2.6 ground_indices

一帧感知结果中的点云消息的数据量是很大的。从而导致地面点的索引数量也会很多。因此在发送地面点索引的时候需要进行 **“分包”** 的处理。**有关该消息的注意事项如下：**

1. **一片地面点的索引会被分成M个数据包。每个数据包跟随1个消息头。因此在发送地面点的索引的时候，一共会有M个与地面点的索引有关的消息头。**
2. **每个地面点的索引数据包中的索引的个数N会因为雷达的型号而有所不同。**
3. **每个地面点的索引数据包中的点的序列化顺序如下所示。**

序号	字段名称	数据类型	字节长度(byte)	注释
1	index	int32	4	一个地面点的点索引
…				
重复N-1次，N 与雷达型号有关				
…				

4.4.1.2.7 background_indices

一帧感知结果中的点云消息的数据量是很大的。从而导致背景点的索引数量也会很多。因此在发送背景点索引的时候需要进行 **“分包”** 的处理。**有关该消息的注意事项如下：**

1. **一片背景点的索引会被分成M个数据包。每个数据包跟随1个消息头。因此在发送背景点的索引的时候，一共会有M个与背景点的索引有关的消息头。**
2. **每个背景点的索引数据包中的索引的个数N会因为雷达的型号而有所不同。**
3. **每个背景点的索引数据包中的点的序列化顺序如下所示。**

序号	字段名称	数据类型	字节长度(byte)	注释
1	index	int32	4	一个背景点的点索引
…				
重复N-1次，N 与雷达型号有关				
…				

4.4.1.2.8 point_cloud

一帧感知结果中的点云消息的数据量是很大的。因此在发送点云的时候需要进行 **“分包”** 的处理。**有关该消息的注意事项如下：**

1. **一片点云会被分成M个数据包。每个数据包跟随1个消息头。因此在发送点云的时候，一共会有M个与点云有关的消息头。**
2. **每个点云数据包中的点的个数N会因为雷达的型号而有所不同。**

3. **每个点云数据包中的点的序列化顺序如下所示。**

序号	字段名称	数据类型	字节长度(byte)	注释
1	point.x	float	4	点云中一个点的x值
2	point.y	float	4	点云中一个点的y值
3	point.z	float	4	点云中一个点的z值
4	point.intensity	float	4	点云中一个点的internsity值
…				
重复序号1-4共N-1次，N与雷达型号有关。				
…				

4.4.1.2.9 pose

一帧感知结果当中只有一个全局的pose。以下为该pose消息的序列化顺序。

序号	字段名称	数据类型	字节长度(byte)	注释
1	pose.x	float	4	该结果中的全局坐标轴pose的x值
2	pose.y	float	4	该结果中的全局坐标轴pose的y值
3	pose.z	float	4	该结果中的全局坐标轴pose的z值
4	pose.roll	float	4	该结果中的全局坐标轴pose的roll值
5	pose.pitch	float	4	该结果中的全局坐标轴pose的pitch值
6	pose.yaw	float	4	该结果中的全局坐标轴pose的yaw值

4.4.2 SDK3.0通信协议

4.4.2.1 消息头(Header)

以下为针对SDK3.0的消息头。备注为空的三项表示消息内容会随着工程的运行而改变。

字段名称	大小(Byte)	字段说明	备注
msgTimestampMs	8	消息时间戳，单位(ms)	
msgVersion	2	消息格式版本	当前默认版本为0x0003
msgType	2	消息类型	
deviceId	4	设备编号	
msgFrameId	4	消息帧编号	>= 0, 从设备感知开始，每处理一帧+1
msgTotalCnt	4	一帧感知结果中，对应消息类型的总消息(UDP报文)个数	不同的消息类型，为了在复杂度和效率之间平衡，采用了不同的组织方式，所以，对于不同消息类型封装的UDP报文数不同
msgLocalCnt	4	UDP报文中包含该类型消息的内容的数量	对于字节固定的消息，为了在延迟/吞吐效率/复杂度方面平衡，可能一个UDP报文包含多个内容
msgLocalLen	2	消息内容的长度	>= 0，特别地，长度为0对应空消息
msgIndex	2	同一帧相同类型消息数据报文索引	>=0, 从0开始编号，未填充有效值，默认填充0x0000
msgTotalFragment	2	消息分片数量	>= 0， 默认填充0x0000
msgFragmentIndex	2	消息分片编号	>=0, 从0开始编号，未填充有效值，默认填充0x0000
msgRes0;	2	保留	未填充有效值，默认填充0x0000
msgCheck16;	2	包含消息头和消息内容的CRC16检查和	未填充有效值，默认填充0xFFFF

4.4.2.2 消息内容(Content)

根据配置文件，用户可选择发送objects / attention_objects / object_supplement / freespace / lanes / curbs / axix_lidar_pose / global_car_pose / point_cloud 这些信息中的一项或全部。当用户选择发送objcets / attention_objects 时，工程会一并发送status信息。下面将介绍上述各消息的序列化顺序。

注意：每个消息头后面跟的消息内容只会是下列消息内容中的1项！其中对objects / attention_objects / point_cloud的特殊处理将在对应小节中进行说明。

4.4.2.2.1 objects / attention_objects

一帧感知结果当中会有**N个**object / attention object (N >= 0)。以下为1个object中含有的内容。**几点注意事项如下**:

1. **SDK3.0与SDK3.1在定义object的内容的时候有些许不同。这些不同之处在下表中以加粗斜体的形式体现并在注释中进行了说明。**

2. 只有用户将配置表中的**object_supplement**置为**true**的时候，该**object**的**Supplement Info**才会随**Core Info**一起一并发送。否则只会发送**object**的**Core Info**。
3. 在发送的时候，**1**个**Header**后面只搭配**1**个**object**。因此，与**object**有关的数据帧会有**N**个。

序号	字段名称	数据类型	字节长度(byte)	注释
Core Info				
1	timestamp	double	8	每个object时间戳信息，单位：秒
2	priority_id	int32	4	优先级ID，根据障碍物的方位、距离、车道、类别、速度大小等信息来排序，ID越小优先级越高
3	exist_confidence	float	4	障碍物存在的概率
4	center	(float,3)	3*4	中心点，表示目标中心点坐标（x,y,z），单位：米
5	center_cov	(float,3)	3*4	目标中心点的不确定度
6	size	(float,3)	3*4	框尺寸，表示目标框尺寸（长，宽，高），单位：米
7	size_cov	(float,3)	3*4	目标框尺寸的不确定度
8	direction	(float,3)	3*4	框朝向，归一化三维向量，定义为框长轴的朝向
9	direction_cov	(float,3)	3*4	框朝向的不确定度
10	type	int32	4	目标类别，分为7类：CONE (圆柱体、锥桶等)，PED (行人)，BIC (骑行者)，CAR (小车)，TRUCK_BUS (大车)，ULTRA_VEHICLE (拖车)，UNKNOWN （未知类别）
11	type_confidence	float	4	目标类别置信度
12	attention_type	int32	4	一度目标类型，NONE表示为非一度目标，ATTENTION表示一度目标
13	motion_state	int32	4	运动状态，表示是否属于动态目标，默认为未知UNKNOWN，动态目标为MOVING，可能运动的目标为MOVABLE，不可能运动的静态目标为STATIONARY
14	lane_pos	int32	4	目标所属车道标识
15	tracker_id	int32	4	跟踪ID，表示连续时间序列中目标的唯一标识ID。范围是0~10000，未被成功跟踪的目标ID为-1

序号	字段名称	数据类型	字节长度(byte)	注释
16	age	double	8	跟踪生命期，表示目标被成功跟踪的时间长度，单位：秒
17	velocity	(float,3)	3*4	速度，三维向量，表示为目标速度在(x,y,z)方向的分量，单位：米/秒
18	velocity_cov	(float,3)	3*4	速度的不确定度
19	acceleration	(float,3)	3*4	加速度，三维向量，表示为目标加速度在(x,y,z)方向的分量，单位：米/秒平方
20	acceleration_cov	(float,3)	3*4	加速度的不确定度
21	angle_velocity	float	4	角速度，单位：弧度/秒
22	angle_velocity_cov	float	4	角速度的不确定度
23	angle_acceleration	float	4	角加速度，单位：弧度/秒平方
24	angle_acceleration_cov	float	4	角加速度的不确定度
25	anchor	(float,3)	3*4	重心点，三维向量，表示目标分割点云重心点坐标，单位：米
26	nearest_point	(float,3)	3*4	最近点，三维向量，表示目标点云中距离雷达最近的点坐标，单位：米
27	is_supplement	bool	1	是否序列化supplement信息
Supplement info				
28	unique_id	uint	4	每个object的独有ID。同一个object前后在不同数据帧中的unique_id也是不同的。
29	***status***	***int32***	***4***	***3.1中无该字段，因此用0来填充以满足SDK3.0的object消息格式***
30	polygon_size	int32	4	polygon的总个数

序号	字段名称	数据类型	字节长度(byte)	注释
31	polygon	(float,3) * N	3 * N * 4	轮廓点，由二维鸟瞰投影面上包围目标点云的最小多边形的顶点组成，N为多边形顶点个数
32	left_point_index	int32	4	轮廓点中最左侧点索引
33	right_point_index	int32	4	轮廓点中最右侧点索引
34	latent_types_size	int32	4	latent_types的总个数
35	latent_types	(float) * N	N*4	N维向量，表示目标属于每个类别的概率，N为分类类别个数
36	size_type	int32	4	形状类型，默认为SMALL，中型大小目标为MEDIUM，大型目标为LARGE
37	mode	int32	4	检测类型，用于表示目标是精确检测的还是用于补盲得到的，精确检测的为FOCUS，用于补盲的为BSD，PREDICTED为根据时序信息预测的，默认为BSD，一般情况下，深度学习方法检测得到的障碍物类别为FOCUS，根据时序预测的为PREDICTED，其它为BSD，
38	in_roi	bool	1	表示障碍物是否在roi区域以内
39	tracking_state	Int32	4	跟踪状态，UNKNOW表示未知，INIT表示在初始化过程中，STABLE表示稳定跟踪，PREDICTION表示当前目标丢失，为预测状态
40	geo_center	(float,3)	3*4	三维向量，根据目标分割点云计算的形状中心点坐标，单位：米
41	geo_size	(float,3)	3*4	三维向量，根据目标分割点云计算的紧缩框大小，单位：米
42	trajectory_size	int32	4	trajectory的总个数

序号	字段名称	数据类型	字节长度(byte)	注释
43	trajectory	(float,3) * N	3 * N * 4	跟踪运动轨迹，表示历史运动轨迹，N为序列缓存长度，可设置
44	history_velocity_size	int32	4	history_velocity的总个数
45	history_velocity	(float,3) * N	3 * N * 4	跟踪历史速度，N为序列缓存长度，可设置
46	history_type_size	int32	4	history_type的总个数
47	history_type	(int32) * N	N*4	跟踪历史类别，表示历史序列中目标的类别，N为序列缓存长度，可设置
48	gps_mode	int32	4	GPS参考点类型，CENTER表示以中心点计算GPS坐标，CENTROID表示以重心点计算GPS坐标，NEAREST表示以最近点计算GPS坐标
49	gps_longtitude	double	8	GPS经度
50	gps_latitude	double	8	GPS纬度
51	gps_altitude	double	8	GPS高程

4.4.2.2.2 status

一帧感知结果当中只有1个status。以下为status的消息内容。

注意：只有选择发送objects / attention_objects的时候，status才会发送。

序号	字段名称	数据类型	字节长度(byte)	注释
1	status	int32	4	该结果中的坐标轴状态(是哪个坐标轴)

4.4.2.2.3 freespace

一帧感知结果当中只有1个freespace消息。而一个freespace消息中包含N个边界点。以下为freespace的消息内容。

注意：SDK3.0与SDK3.1在定义freespace的内容的时候有些许不同。这些不同之处在下表中以加粗斜体的形式体现并在注释中进行了说明。

序号	字段名称	数据类型	字节长度(byte)	注释
1	fs_pt.x	float	4	可行驶区域边界点三维坐标点
2	fs_pt.y	float	4	可行驶区域边界点三维坐标点
3	fs_pt.z	float	4	可行驶区域边界点三维坐标点
…				
重复序号1-3 共N-1遍				
…				
3N+1	***status***	***int32***	***4***	***3.1中无该字段，因此用0来填充以满足3.0的object消息格式***

4.4.2.2.4 lanes

一帧感知结果当中只有1个lanes消息。而这个消息由N个lane消息构成。以下是一个lanes消息的内容。

注意：SDK3.0与SDK3.1在定义lanes的内容的时候有些许不同。这些不同之处在下表中以加粗斜体的形式体现并在注释中进行了说明。

序号	字段名称	数据类型	字节长度(byte)	注释
1	***status***	***int32***	***4***	***3.1中无该字段，因此用0来填充以满足3.0的object消息格式***
2	lane_id	int32	4	车道线位置ID，左侧LEFT_EGO、LEFT_ADJACENT、LEFT_THIRD、LEFT_FOURTH、LEFT_FIFTH、LEFT_SIXTH，右侧RIGHT_EGO、RIGHT_ADJACENT、RIGHT_THIRD、RIGHT_FOURTH、RIGHT_FIFTH、RIGHT_SIXTH，以及其他OTHER
3	curve	(float,6)	6 * 4	4个三次曲线拟合参数以及起止点。顺序为x_start, x_end, a, b, c, d
4	end_point	(float,2)*2	2 * 2 * 4	车道线起始点和终止点。顺序为start.x, start.y, end.x, end.y
5	measure_status	int32	4	车道线获得方式：检测(DETECTION)和预测(PREDICTION)
6	confidence	float	4	检测置信度
…				
重复序号1-6共N-1次				
…				

4.4.2.2.5 curbs

一帧感知结果当中只有1个curbs消息。而这个消息由N个curb消息构成。以下是一个curbs消息的内容。

注意：SDK3.0与SDK3.1在定义curbs的内容的时候有些许不同。这些不同之处在下表中以加粗斜体的形式体现并在注释中进行了说明。

序号	字段名称	数据类型	字节长度(byte)	注释
1	***status***	***int32***	***4***	***3.1中无该字段，因此用0来填充以满足3.0的object消息格式***
2	roadedge_id	int32	4	道路边界位置ID，左侧LEFT，右侧RIGHT，以及未知UNKNOW
3	curve	(float,6)	6*4	4个三次曲线拟合参数以及起止点。顺序为x_start, x_end, a, b, c, d
4	end_point	(float,2)*2	2 * 2 * 4	道路边界起始点和终止点.顺序为start.x, start.y, end.x, end.y
5	measure_status	int32	4	道路边界获得方式：检测(DETECTION)和预测(PREDICTION)
6	confidence	float	4	检测置信度
…				
重复序号1-6共N-1次				
…				

4.4.2.2.6 axis_lidar_pose

一帧感知结果当中会存有 LIDAR_AXIS / VEHICLE_AXIS / ALIGN_AXIS / ROTATE_AXIS / GLOBAL_AXIS 这个5个坐标系对应的pose。这些pose将跟随一个消息头一并发出。一下为这些pose消息的序列化顺序。

序号	字段名称	数据类型	字节长度(byte)	注释
1	pose.x	float	4	该结果中的某坐标轴pose的x值
2	pose.y	float	4	该结果中的某坐标轴pose的y值
3	pose.z	float	4	该结果中的某坐标轴pose的z值
4	pose.roll	float	4	该结果中的某坐标轴pose的roll值
5	pose.pitch	float	4	该结果中的某坐标轴pose的pitch值
6	pose.yaw	float	4	该结果中的某坐标轴pose的yaw值
7	poseStatus	int32	4	该结果中的某坐标轴的名称。取值为LIDAR_AXIS = 0, VEHICLE_AXIS = 1, ALIGN_AXIS = 2, ROTATE_AXIS = 3, GLOBAL_AXIS = 4,
…				
重复序号1-7共4次				
…				

4.4.2.2.7 global_car_pose

一帧感知结果当中只有一个全局的pose。以下为该pose消息的序列化顺序。

序号	字段名称	数据类型	字节长度(byte)	注释
1	pose.x	float	4	该结果中的全局坐标轴pose的x值
2	pose.y	float	4	该结果中的全局坐标轴pose的y值
3	pose.z	float	4	该结果中的全局坐标轴pose的z值
4	pose.roll	float	4	该结果中的全局坐标轴pose的roll值
5	pose.pitch	float	4	该结果中的全局坐标轴pose的pitch值
6	pose.yaw	float	4	该结果中的全局坐标轴pose的yaw值
7	poseStatus	int32	4	该结果中的全局坐标轴的名称。取值为GLOBAL_AXIS = 4

4.4.2.2.8 point_cloud

一帧感知结果中的点云消息的数据量是很大的。因此在发送点云的时候需要进行 **“分包”** 的处理。**有关该消息的注意事项如下：**

1. **一片点云会被分成M个数据包。每个数据包跟随1个消息头。因此在发送点云的时候，一共会有M个与点云有关的消息头。**
2. **每个点云数据包中的点的个数N会因为雷达的型号而有所不同。**
3. **每个点云数据包中的点的序列化顺序如下所示。**

序号	字段名称	数据类型	字节长度(byte)	注释
1	point.x	float	4	点云中一个点的x值
2	point.y	float	4	点云中一个点的y值
3	point.z	float	4	点云中一个点的z值
4	point.intensity	float	4	点云中一个点的internsity值
5	label	int32	4	点云中一个点的label值,用于判定这个点是属于地面/非地面/背景/前景object/一度目标
...				
重复序号1-5共N-1次。N的大小根据雷达型号会有不同。				
...				

4.4.3 SDK3.1通信协议

4.4.3.1 消息头(Header)

以下为针对SDK3.1的消息头。备注为空的三项表示消息内容会随着工程的运行而改变。

字段名称	大小(Byte)	字段说明	备注
msgVersion	2	消息格式版本	当前默认版本为0x7E8E
msgType	2	消息类型	
deviceId	4	设备编号	默认为1
msgTimestampS	8	消息时间戳，单位秒(s)	
msgFrameId	4	消息帧编号	>= 0, 从设备感知开始，每处理一帧+1
msgTotalCnt	4	一帧感知结果中，对应消息类型的总消息(UDP 报文)个数	不同的消息类型，为了在复杂度和效率之间平衡，采用了不同的组织方式，所以，对于不同消息类型封装的UDP报文数不同
msgTotalLen	4	发送的数据长度	压缩时，为压缩后的数据长度；非压缩时，为原始数据的长度
msgTotalUncompressLen	4	非压缩的原始数据长度	压缩时，为原始数据的长度,非压缩时为0
msgLocalCnt	2	UDP报文中包含该类型消息的内容的数量	对于字节固定的消息，为了在延迟/吞吐效率/复杂度方面平衡，可能一个UDP报文包含多个内容
msgLocalLen	2	消息内容的长度	>= 0，特别地，长度为0对应空消息
msgIndex	2	同一帧相同类型消息数据报文索引	>=0, 从0开始编号，未填充有效值，默认填充0x0000
msgTotalFragment	2	消息分片数量	>= 0， 默认填充0x0000
msgFragmentIndex	2	消息分片编号	>=0, 从0开始编号，未填充有效值，默认填充0x0000
msgCheck16	2	包含消息头和消息内容的CRC16检查和	未填充有效值，默认填充0xFFFF
msgRes0	4	保留	未填充有效值，默认填充0x0000

4.4.3.2 消息内容(Content)

NativeBytes_3.1协议下发送的内容如下表所示。其中表4.4.3.2-1为固定发送的消息内容。表4.4.3.2-2为根据配置文件选择发送的内容。

字段名称	注释
timestamp	每一帧数据的时间戳
globale pose	全局pose
gps origin	GPS原点信息
status pose map	坐标轴的pose
status	该帧的坐标系
valid indices	该帧有效点的索引
objects	该帧结果中的所有前景物体

表 4.4.3.2-1 固定发送的内容

字段名称	注释
scan ptr	点云
attention objects	该帧结果中的一度目标
freespace	该帧结果中的可行驶区域
lanes	该帧结果中的车道线
roadedge	该帧结果中的路沿
sematic	该帧结果中的地面点/非地面点/背景点的索引

表 4.4.3.2-2 可选发送的内容

上述可选发送内容可以通过配置表来进行开关。下面将依次介绍上述内容的字段序列化顺序。

注意：

1. **当用户选择发送sematic的时候，消息将被分成 non_ground_indices / ground_indices / background_indices 三部分进行发送。**
2. **每个消息头后面跟的消息内容只会是下列消息内容中的1项！其中对objects / attention_objects / point_cloud / non_ground_indices / ground_indices / background_indices 的特殊处理将在对应小节中进行说明。**

4.4.3.2.1 timestamp

一帧感知结果只有1个时间戳，以下为时间戳的序列化顺序。

序号	字段名称	数据类型	字节长度(byte)	注释
1	timestamp	double	8	每一帧数据的时间戳

4.4.3.2.2 global pose

一帧感知结果当中只有一个全局的pose。以下为该pose消息的序列化顺序。

序号	字段名称	数据类型	字节长度(byte)	注释
1	pose.x	float	4	该结果中的全局坐标轴pose的x值
2	pose.y	float	4	该结果中的全局坐标轴pose的y值
3	pose.z	float	4	该结果中的全局坐标轴pose的z值
4	pose.roll	float	4	该结果中的全局坐标轴pose的roll值
5	pose.pitch	float	4	该结果中的全局坐标轴pose的pitch值
6	pose.yaw	float	4	该结果中的全局坐标轴pose的yaw值
7	poseStatus	int32	4	该结果中的全局坐标轴的名称。取值为GLOBAL_AXIS = 4

4.4.3.2.3 gps origin

一帧感知结果当中只有一个原始的GPS信息。以下为该GPS消息的序列化顺序。

序号	字段名称	数据类型	字节长度(byte)	注释
1	x	double	8	原始GPS经度
2	y	double	8	原始GPS纬度
3	z	double	8	原始GPS高程

4.4.3.2.4 status pose map

一帧感知结果当中会存有 LIDAR_AXIS / VEHICLE_AXIS / ALIGN_AXIS / ROTATE_AXIS / GLOBAL_AXIS 这个5个坐标系对应的pose。这些pose将跟随一个消息头一并发出。一下为这些pose消息的序列化顺序。

序号	字段名称	数据类型	字节长度(byte)	注释
1	pose.x	float	4	该结果中的某坐标轴pose的x值
2	pose.y	float	4	该结果中的某坐标轴pose的y值
3	pose.z	float	4	该结果中的某坐标轴pose的z值
4	pose.roll	float	4	该结果中的某坐标轴pose的roll值
5	pose.pitch	float	4	该结果中的某坐标轴pose的pitch值
6	pose.yaw	float	4	该结果中的某坐标轴pose的yaw值
7	poseStatus	int32	4	该结果中的某坐标轴的名称。取值为LIDAR_AXIS = 0, VEHICLE_AXIS = 1, ALIGN_AXIS = 2, ROTATE_AXIS = 3, GLOBAL_AXIS = 4,
…				
重复序号1-7共4次				
…				

4.4.3.2.5 status

一帧感知结果当中只有1个status。以下为status的消息内容。

序号	字段名称	数据类型	字节长度(byte)	注释
1	status	int32	4	该结果中的坐标轴状态(是哪个坐标轴)

4.4.3.2.6 valid indices

一帧感知结果当中，有效点的索引的数量是巨大的。因此在发送有效点的索引的时候需要进行 **“分包”** 的处理。**有关该消息的注意事项如下：**

1. **一片有效点的索引会被分成M个数据包。每个数据包跟随1个消息头。因此在发送有效点的索引的时候，一共会有M个与有效点的索引有关的消息头。**
2. **每个有效点的索引数据包中的索引的个数N会因为雷达的型号而有所不同。**
3. **每个有效点的索引数据包中的点的序列化顺序如下所示。**

序号	字段名称	数据类型	字节长度(byte)	注释
1	index	int32	4	一个有效点的点索引
…				
重复N-1次，N 与雷达型号有关				
…				

4.4.3.2.7 objects / attention objects

一帧感知结果当中会有**N个**object / attention object (N >= 0)。以下为1个object中含有的内容。**几点注意事项如下**:

1. **SDK3.0与SDK3.1在定义object的内容的时候有些许不同。这些不同之处在下表中以加粗斜体的形式体现并在注释中进行了说明。**
2. **只有用户将配置表中的object_supplement置为true的时候，该object的Supplement Info才会随Core Info一起一并发送。否则只会发送object的Core Info。**
3. **在发送的时候，1个Header后面只搭配1个object。因此，与object有关的数据帧会有N个。**

序号	字段名称	数据类型	字节长度(byte)	注释
Core Info				
1	timestamp	double	8	每个object时间戳信息，单位：秒
2	priority_id	int32	4	优先级ID，根据障碍物的方位、距离、车道、类别、速度大小等信息来排序，ID越小优先级越高
3	exist_confidence	float	4	障碍物存在的概率
4	center	(float,3)	3*4	中心点，表示目标中心点坐标（x,y,z），单位：米
5	center_cov	(float,3)	3*4	目标中心点的不确定度
6	size	(float,3)	3*4	框尺寸，表示目标框尺寸（长，宽，高），单位：米
7	size_cov	(float,3)	3*4	目标框尺寸的不确定度
8	direction	(float,3)	3*4	框朝向，归一化三维向量，定义为框长轴的朝向
9	direction_cov	(float,3)	3*4	框朝向的不确定度
10	type	int32	4	目标类别，分为7类：CONE (圆柱体、锥桶等)，PED (行人)，BIC (骑行者)，CAR (小车)，TRUCK_BUS (大车)，ULTRA_VEHICLE (拖车)，UNKNOWN （未知类别）
11	type_confidence	float	4	目标类别置信度
12	attention_type	int32	4	一度目标类型，NONE表示为非一度目标，ATTENTION表示一度目标
13	motion_state	int32	4	运动状态，表示是否属于动态目标，默认为未知UNKNOWN，动态目标为MOVING，可能运动的目标为MOVABLE，不可能运动的静态目标为STATIONARY
14	lane_pos	int32	4	目标所属车道标识
15	tracker_id	int32	4	跟踪ID，表示连续时间序列中目标的唯一标识ID。范围是0~10000，未被成功跟踪的目标ID为-1

序号	字段名称	数据类型	字节长度(byte)	注释
16	age	double	8	跟踪生命期，表示目标被成功跟踪的时间长度，单位：秒
17	velocity	(float,3)	3*4	速度，三维向量，表示为目标速度在(x,y,z)方向的分量，单位：米/秒
18	relative_velocity	(float,3)	3*4	相对速度，三维向量，表示为目标速度在(x,y,z)方向的分量，单位：米/秒
19	velocity_cov	(float,3)	3*4	速度的不确定度
20	related_velocity_cov	(float,3)	3*4	相对速度的不确定度
21	acceleration	(float,3)	3*4	加速度，三维向量，表示为目标加速度在(x,y,z)方向的分量，单位：米/秒平方
22	acceleration_cov	(float,3)	3*4	加速度的不确定度
23	angle_velocity	float	4	角速度，单位：弧度/秒
24	angle_velocity_cov	float	4	角速度的不确定度
25	angle_acceleration	float	4	角加速度，单位：弧度/秒平方
26	angle_acceleration_cov	float	4	角加速度的不确定度
27	anchor	(float,3)	3*4	重心点，三维向量，表示目标分割点云重心点坐标，单位：米
28	nearest_point	(float,3)	3*4	最近点，三维向量，表示目标点云中距离雷达最近的点坐标，单位：米
29	is_supplement	bool	1	是否序列化supplement信息
Supplement info				
30	unique_id	uint	4	每个object的独有ID。同一个object前后在不同数据帧中的unique_id也是不同的。
31	polygon_size	int32	4	polygon的总个数

序号	字段名称	数据类型	字节长度(byte)	注释
32	polygon	(float,3) * N	3 * N * 4	轮廓点，由二维鸟瞰投影面上包围目标点云的最小多边形的顶点组成，N为多边形顶点个数
33	left_point_index	int32	4	轮廓点中最左侧点索引
34	right_point_index	int32	4	轮廓点中最右侧点索引
35	latent_types_size	int32	4	latent_types的总个数
36	latent_types	(float) * N	N*4	N维向量，表示目标属于每个类别的概率，N为分类类别个数
37	size_type	int32	4	形状类型，默认为SMALL，中型大小目标为MEDIUM，大型目标为LARGE
38	mode	int32	4	检测类型，用于表示目标是精确检测的还是用于补盲得到的，精确检测的为FOCUS，用于补盲的为BSD，PREDICTED为根据时序信息预测的，默认为BSD，一般情况下，深度学习方法检测得到的障碍物类别为FOCUS，根据时序预测的为PREDICTED，其它为BSD，
39	in_roi	bool	1	表示障碍物是否在roi区域以内
40	tracking_state	Int32	4	跟踪状态，UNKNOW表示未知，INIT表示在初始化过程中，STABLE表示稳定跟踪，PREDICTION表示当前目标丢失，为预测状态
41	geo_center	(float,3)	3*4	三维向量，根据目标分割点云计算的形状中心点坐标，单位：米
42	geo_size	(float,3)	3*4	三维向量，根据目标分割点云计算的紧缩框大小，单位：米
43	trajectory_size	int32	4	trajectory的总个数

序号	字段名称	数据类型	字节长度(byte)	注释
44	trajectory	(float,3) * N	3 * N * 4	跟踪运动轨迹，表示历史运动轨迹，N为序列缓存长度，可设置
45	history_velocity_size	int32	4	history_velocity的总个数
46	history_velocity	(float,3) * N	3 * N * 4	跟踪历史速度，N为序列缓存长度，可设置
47	history_type_size	int32	4	history_type的总个数
48	history_type	(int32) * N	N*4	跟踪历史类别，表示历史序列中目标的类别，N为序列缓存长度，可设置
49	gps_mode	int32	4	GPS参考点类型，CENTER表示以中心点计算GPS坐标，CENTROID表示以重心点计算GPS坐标，NEAREST表示以最近点计算GPS坐标
50	gps_longtitude	double	8	GPS经度
51	gps_latitude	double	8	GPS纬度
52	gps_altitude	double	8	GPS高程

4.4.3.2.8 point cloud

一帧感知结果中的点云消息的数据量是很大的。因此在发送点云的时候需要进行 **“分包”** 的处理。**有关该消息的注意事项如下：**

1. **一片点云会被分成M个数据包。每个数据包跟随1个消息头。因此在发送点云的时候，一共会有M个与点云有关的消息头。**
2. **每个点云数据包中的点的个数N会因为雷达的型号而有所不同。**
3. **每个点云数据包中的点的序列化顺序如下所示。**

序号	字段名称	数据类型	字节长度(byte)	注释
1	point.x	float	4	点云中一个点的x值
2	point.y	float	4	点云中一个点的y值
3	point.z	float	4	点云中一个点的z值
4	point.intensity	float	4	点云中一个点的internsity值
5	label	int32	4	点云中一个点的label值,用于判定这个点是属于哪个object/attention object
…				
重复序号1-5共N次。N的大小根据雷达型号会有不同。				
…				

4.4.3.2.9 freespace

一帧感知结果当中只有1个freespace消息。而一个freespace消息中包含N个边界点。以下为freespace的消息内容。

序号	字段名称	数据类型	字节长度(byte)	注释
1	fs_pt.x	float	4	可行驶区域边界点三维坐标点
2	fs_pt.y	float	4	可行驶区域边界点三维坐标点
3	fs_pt.z	float	4	可行驶区域边界点三维坐标点
4	fs_confidence	float	4	该点的置信度
…				
重复序号1-3共N-1遍				
…				

4.4.3.2.10 lanes

一帧感知结果当中只有1个lanes消息。而这个消息由N个lane消息构成。以下是一个lanes消息的内容。

序号	字段名称	数据类型	字节长度(byte)	注释
1	lane_id	int32	4	车道线位置ID，左侧LEFT_EGO、LEFT_ADJACENT、LEFT_THIRD、LEFT_FOURTH、LEFT_FIFTH、LEFT_SIXTH，右侧RIGHT_EGO、RIGHT_ADJACENT、RIGHT_THIRD、RIGHT_FOURTH、RIGHT_FIFTH、RIGHT_SIXTH，以及其他OTHER
2	curve	(float,6)	6*4	4个三次曲线拟合参数以及起止点。顺序为x_start, x_end, a, b, c, d
3	end_point	(float,2) * 2	2 * 2 * 4	车道线起始点和终止点。顺序为start.x, start.y, end.x, end.y
4	measure_status	int32	4	车道线获得方式：检测(DETECTION)和预测(PREDICTION)
5	confidence	float	4	检测置信度
...				
重复序号1-5共N次				
...				

4.4.3.2.11 roadedge

一帧感知结果当中只有1个roadedges消息。而这个消息由N个roadedge消息构成。以下是一个roadedge消息的内容。

序号	字段名称	数据类型	字节长度(byte)	注释
1	roadedge_id	int32	4	道路边界位置ID，左侧LEFT，右侧RIGHT，以及未知UNKNOW
2	curve	(float,6)	6 * 4	4个三次曲线拟合参数以及起止点。顺序为x_start, x_end, a, b, c, d
3	end_point	(float,2)*2	2 * 2 * 4	道路边界起始点和终止点.顺序为start.x, start.y, end.x, end.y
4	measure_status	int32	4	道路边界获得方式：检测(DETECTION)和预测(PREDICTION)
5	confidence	float	4	检测置信度
…				
重复序号1-5共N次				
…				

4.4.3.2.12 sematic

一帧感知结果中，语义点云的索引的数据量是巨大的。因此在发送语义点云的索引的时候需要进行 **“分包”** 的处理。为了便于区分，点的语义类型在消息头当中进行了说明。**同一个数据包中只有相同语义的点的索引值**。

语义点索引共3类：地面点索引 / 非地面点索引 / 背景点索引。这些索引在序列化的时候的顺序及结构是完全一样的。因此本文档不作分开说明。**有关该消息的注意事项如下：**

1. **一片地面点索引 / 非地面点索引 / 背景点索引会被分成M个数据包。每个数据包跟随1个消息头。因此在发送有效点的索引的时候，一共会有M个与地面点索引 / 非地面点索引 / 背景点索引有关的消息头。**
2. **每个地面点索引 / 非地面点索引 / 背景点索引数据包中的索引的个数N会因为雷达的型号而有所不同。**
3. **每个地面点索引 / 非地面点索引 / 背景点索引引数据包中的点的序列化顺序如下所示。**
4. **同一个数据包中只有相同语义的点的索引值。**

序号	字段名称	数据类型	字节长度(byte)	注释
1	index	int32	4	一个非地面点 / 地面点 / 背景点的点索引
...				
重复N-1次，N 与雷达型号有关				
...				

4.5 V2R 通信协议

1. 协议约定开始字节为 0x7E7E,结束字节为 0x7E7D;
2. 协议约定采用二进制方式通信(即字节流)，协议数据为多字节时，采用小端模式传输（即低位字节先发送，高位字节后发送，例如int 类型数据0x11223344, 则发送数据顺序为 0x44 0x33 0x22 0x11);
3. 协议校验采用 CRC16-x25 方式,校验数据为除去报头和报尾以及校验位本身, 即设备类型至数据区(包括数据区);
4. 感知物体上报周期为雷达工作扫描周期(典型为100 毫秒), 特别地，感知区域内未检测到感知物体时，发送数据区长度为0的数据类型帧
5. 雷达状态消息上报周期为雷达工作扫描周期(典型为100毫秒)
6. Robosense雷达端暂不接收和处理任何对端发送的消息数据

Robosense V2R通信协议的定义详见表2-1

序号	字段名称	数据类型	字段长度(byte)	注释
0	数据帧头	Uchar[2]	2	包头(取值为0x7E7E)
1	设备类型	Uchar	1	设备类型: 每一个bit位代表是否支持的感知能力 bit0:1--激光雷达 0--无 bit1~7: 未使用
2	数据帧类型	Uchar	1	传输的数据类型 0x00---感知数据消息 0x01---设备状态消息 其他未使用
3	数据帧长度	Ushort	2	数据区长度
4	设备ID	Uint64	8	设备ID，0xFFFFFFFFFFFFFF表示无效值
5	时间戳	UInt64	8	ms为单位的时间戳, 0xFFFFFFFFFFFFFF表示无效值
6	数据帧数据区	data	n (根据具体数据而定)	要传输的数据 *(详见表 2-2/2-3-1/2-3-2/2-3-3)*
7	校验位	Ushort	2	采用CRC16-x25校验方式
8	数据帧尾	UChar[2]	2	包尾(取值为0x7E7D)

表 2-1 通信协议的定义

表 2-2 提供了消息类型为 0x01 的数据段中消息的详细列表

序号	字段名称	数据类型	字段长度(byte)	注释
1	设备ID	Uint64	8	设备ID
2	设备状态	Uint16	2	设备状态0x00 表示正常其他表示异常

表 2-2 消息类型：0x01 数据段消息详细列表

4.5.1 Robosense V2R 1.0~1.4 协议字段说明

序号	字段名称	数据类型	字段长度(byte)	注释
1	障碍物所处区域	UInt16	2	标识感知的区域: BIT15-8: 表示区域类型 BIT7-0: 表示区域的编号 对应字节为0xFF时表示无效值 对应字节取值为0xFF表示无效值
2	障碍物类型	Uint16	2	0-未定义 1-行人 2-非机动车 3-小车 4-大车 5-超大车 6-路锥
3	障碍物ID	Uint	4	同一感知物体在被追踪时间内ID保持不变,0xFFFFFFFF时表示无效值
4	障碍物长度	Float	4	物体本身长度,单位m
5	障碍物宽度	Float	4	物体本身宽度,单位m
6	障碍物高度	Float	4	物体本身高度,单位m
7	障碍物朝向	Float	4	物体框的长边方向(0~360),以正北为参考轴，顺时针旋转为正，单位:度
8	障碍物速度	Float	4	单位:m/s
9	障碍物航向	Float	4	WGS84/CGCS2000 下的方向角度(0-360)，以正北为参考轴，顺时针旋转为正，单位:度
10	障碍物加速度	Float	4	单位:m/s^2，-1表示无效值
11	障碍物加速度方向	Float	4	WGS84/CGCS2000 下的方向角度(0-360)以正北为参考轴，顺时针旋转为正，单位:度，**(加速度无效时，该值无效)**

序号	字段名称	数据类型	字段长度(byte)	注释
12	障碍物经度	Double	8	WGS84 /CGCS2000 坐标系(物体中心点经度) 原始坐标为雷达坐标系,物体中心点 XY
13	障碍物纬度	Double	8	WGS84 /CGCS2000 坐标系(物体中心点纬度) 原始坐标为雷达坐标系,物体中心点 XY
.....				
重复序号1-13 # 序号1-13为一个object的信息				
.....				

表 2-3-1 消息类型: 0x00数据段消息详细列表 (对应协议版本v1.0-v1.4)

4.5.2 Robosense V2R 1.5 协议字段说明

表 2-3-2 为 Robosense V2R 1.5版本协议。**该版本在1.4协议的基础上增加了 *相对距离* 这个数据项**。

序号	字段名称	数据类型	字段长度(byte)	注释
1	障碍物所处区域	UInt16	2	标识感知的区域: BIT15-8: 表示区域类型 BIT7-0: 表示区域的编号 对应字节为0xFF时表示无效值 对应字节取值为0xFF表示无效值
2	障碍物类型	Uint16	2	0-未定义 1-行人 2-非机动车 3-小车 4-大车 5-超大车 6-路锥
3	障碍物ID	Uint	4	同一感知物体在被追踪时间内ID保持不变,0xFFFFFFFF时表示无效值
4	障碍物长度	Float	4	物体本身长度,单位m
5	障碍物宽度	Float	4	物体本身宽度,单位m
6	障碍物高度	Float	4	物体本身高度,单位m
7	障碍物朝向	Float	4	物体框的长边方向(0~360),以正北为参考轴，顺时针旋转为正，单位:度
8	障碍物速度	Float	4	单位:m/s
9	障碍物航向	Float	4	WGS84/CGCS2000 下的方向角度(0-360)，以正北为参考轴，顺时针旋转为正，单位:度
10	障碍物加速度	Float	4	单位:m/s^2，-1表示无效值
11	障碍物加速度方向	Float	4	WGS84/CGCS2000 下的方向角度(0-360)以正北为参考轴，顺时针旋转为正，单位:度，**加速度无效时，该值无效**

序号	字段名称	数据类型	字段长度(byte)	注释
12	障碍物经度	Double	8	WGS84 /CGCS2000 坐标系(物体中心点经度) 原始坐标为雷达坐标系,物体中心点XY
13	障碍物纬度	Double	8	WGS84 /CGCS2000 坐标系(物体中心点纬度) 原始坐标为雷达坐标系,物体中心点XY
14	**相对距离**	**Float**	**4**	**单位: m, 相对距离** *(1.5版本新增项目)*
……				
重复序号1-14 #序号1-14为一个object的信息				
……				

表 2-3-2 消息类型: 0x00 数据段消息详细列表(对应协议版本 v1.5)

4.5.3 Robosense V2R 1.6 协议字段说明

表 2-3-3 位Robosense V2R 1.6版本协议。**该版本协议在1.4版本协议的基础上增加了 *障碍物中心点位置的XYZ值以及障碍物的海拔高度* 这4个数据项**。

序号	字段名称	数据类型	字段长度(byte)	注释
1	障碍物所处区域	UInt16	2	标识感知的区域: BIT15-8: 表示区域类型 BIT7-0: 表示区域的编号 对应字节为0xFF时表示无效值 对应字节取值为0xFF表示无效值
2	障碍物类型	Uint16	2	0-未定义 1-行人 2-非机动车 3-小车 4-大车 5-超大车 6-路锥
3	障碍物ID	Uint	4	同一感知物体在被追踪时间内ID保持不变, 0xFFFFFFFF时表示无效值
4	**障碍物中心点位置X**	**Float**	**4**	**物体的相对位置X分量，单位m** *(1.6版本协议新增数据项)*
5	**障碍物中心点位置Y**	**Float**	**4**	**物体的相对位置Y分量，单位m** *(1.6版本协议新增数据项)*
6	**障碍物中心点位置Z**	**Float**	**4**	**物体的相对位置Z分量，单位m** *(1.6版本协议新增数据项)*
7	障碍物长度	Float	4	物体本身长度,单位m
8	障碍物宽度	Float	4	物体本身宽度,单位m
9	障碍物高度	Float	4	物体本身高度,单位m
10	障碍物朝向	Float	4	物体框的长边方向(0~360),以正北为参考轴，顺时针旋转为正，单位:度

序号	字段名称	数据类型	字段长度(byte)	注释
11	障碍物速度	Float	4	单位:m/s
12	障碍物航向	Float	4	WGS84/CGCS2000 下的方向角度(0-360)，以正北为参考轴，顺时针旋转为正，单位:度
13	障碍物加速度	Float	4	单位:m/s^2，-1表示无效值
14	障碍物加速度方向	Float	4	WGS84/CGCS2000 下的方向角度(0-360)以正北为参考轴，顺时针旋转为正，单位:度，**加速度无效时，该值无效**
15	障碍物经度	Double	8	WGS84 /CGCS2000 坐标系(物体中心点经度) 原始坐标为雷达坐标系,物体中心点 XYZ 单位: m, 相对距离
16	障碍物纬度	Double	8	WGS84 /CGCS2000 坐标系(物体中心点纬度) 原始坐标为雷达坐标系,物体中心点 XYZ 单位: m, 相对距离
17	**障碍物海拔**	**Double**	**8**	**WGS84 /CGCS2000 坐标系(物体中心点海拔) 原始坐标为雷达坐标系,物体中心点 XYZ 单位: m, 相对距离** *(1.6版本协议新增数据项)*
……				
重复序号1-17 # 序号1-17为一个object的信息				
……				

表 2-3-3 消息类型: 0x00 数据段消息详细列表(对应协议版本 v1.6)