

robosense® LiDAR

Model M1 120° FOV 155m Range 1000pts/s



| 版本号 | 修订内容 | 修订时间 | 拟制 |
|-------|---------------------|------------|--------|
| 0.1.0 | 初次发行 | 2021-01-15 | Ethan |
| 0.2.0 | 增加多雷达融合模式、感知结果输出调整 | 2021-02-23 | Ethan |
| 0.3.0 | 增加refine模块 | 2021-04-01 | Ethan |
| 0.4.0 | 增加rs_sensor和同步机制 | 2021-08-25 | Brooks |
| 0.4.1 | 增加镜像目标和路牌检测模块（beta） | 2021-11-03 | Owen |

1、关于本文档

2、SDK介绍

2.1 算法简介

2.2 功能介绍

3、参数说明

3.1 usr_config.yaml 配置

3.2 calibration.yaml配置

4、快速启动

4.1 环境依赖

4.2 下载

4.3 编译

4.4 运行

4.5 查看感知结果

5、感知输出列表

5.1 Object 障碍物检测信息

5.2 RsFreeSpace可行驶区域信息

5.3 Lane 车道线信息

5.4 RoadEdge 道路边界信息

6、感知通信方式

附录

A: 软件授权方式说明

B: API使用说明

C: 相关定义说明

D: 坐标系说明

E: 感知参数说明

F: 环境与硬件配置依赖说明

1、关于本文档

Robosense 感知SDK是由Robosense开发人员设计开发的**Smart Sensor**方案的感知软件包。

2、SDK介绍

2.1 算法简介

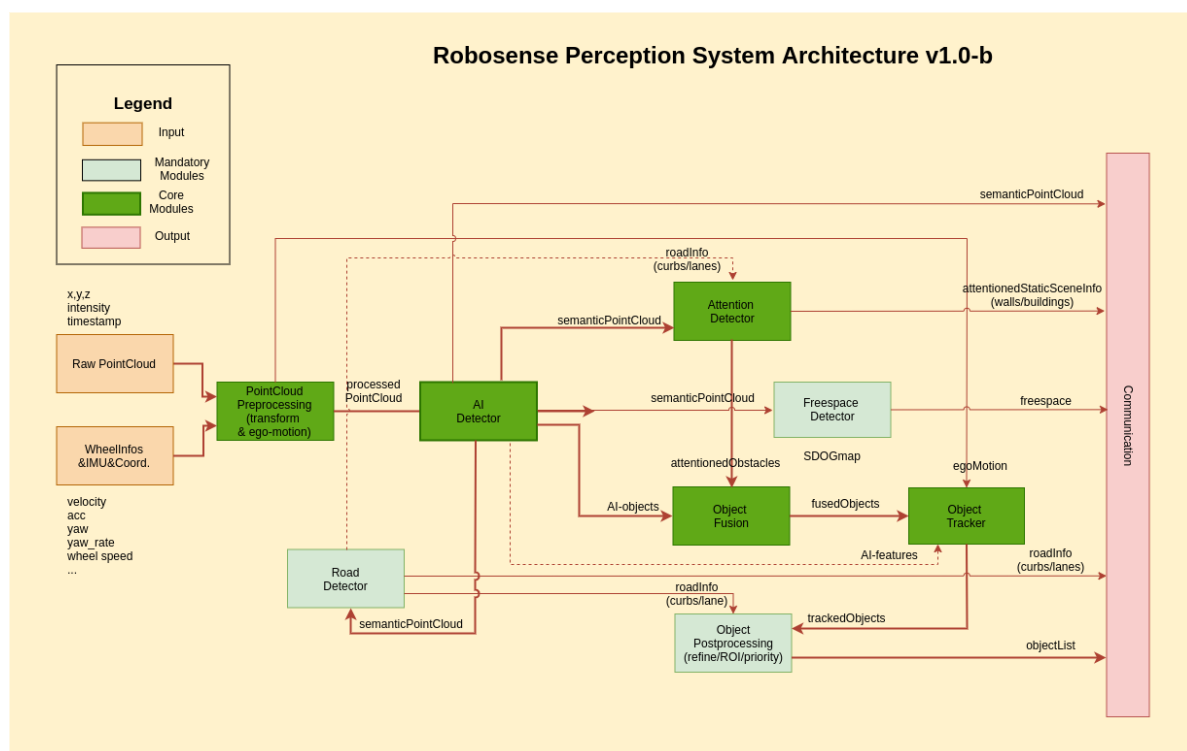
Robosense 感知SDK在架构设计上充分考虑安全可靠与高性能需求，创新地融合了基于几何规则的传统点云算法和基于数据驱动的深度学习算法，可以为自动驾驶决策规划模块提供像素级的全方位、结构化、高精度的感知信息。基于数据闭环，后续可以在不改变架构的情况下，不断提升性能，将更多、更强的感知能力赋予自动驾驶汽车。

2.2 功能介绍

Robosense 感知SDK同时支持单雷达和多雷达感知，提供如下核心功能：

- 前景目标（Object）检测和分类（圆柱体/行人/非机动车/小车/大车/拖车/未知物体）
- 一度目标（Attention Object）检测、栅栏（Barrier）检测
- 传统分割聚类
- 可行驶区域（Freespace）检测
- 路沿（Roadedge）和车道线（Lane）检测
- 前景目标跟踪（Tracking）
- 地面检测
- 高反路牌检测功能（beta测试版本）
- 镜像目标检测（beta测试版本）

Robosense 感知SDK架构如下图所示：



图中虚线框部分为可选模块。SDK既支持单雷达感知，也支持多雷达的前融合或后融合感知。图中模块介绍如下：

- PointCloud Preprocessing：预处理模块，包括点云变换、过滤及特征变换等
- AI Detector：基于深度学习的感知模块
- Attention Detector：一度目标检测模块
- Object Fusion：一度目标与深度学习感知目标的融合模块
- Freespace Detector：可行驶区域检测模块
- Road Detector：路沿与车道线检测模块
- Object Tracker：目标跟踪模块
- Object Postprocessing：后处理模块，包括对目标进行验证、筛选及输出变换等
- Communication：通信模块

3、参数说明

3.1 usr_config.yaml 配置

```
## General ##
general:
  application: "SmartSensor" # 感知策略 #V2r,Pseries,SmartSensor
  log_level: "info" # log等级, 可选error,warning,info,debug,trace, 默认为info
  log: true # 是否存储log信息, 若设置为true, log信息存储在 /tmp/rs_sdk.log
  run_perception: true # 是否运行感知
  run_localization: false # 是否运行定位 (仅在Pseries策略下提供)
  run_communication: false # 是否运行通信模块

## Preprocessing ##
preprocessing:
  common:
    lidar_fusion: false
  result_sender:
    - method: Ros # 发送定位所使用点云 (目前只支持ros方式)
      send_fusion_lidar: false
      fusion_lidar_topic: /fusion_lidar_points

## Perception ##
perception:
  general:
    model: system_config/perception_config/model
    authorization: # SDK的授权方式。若运行的设备无法使用USB_KEY时, 将authorization节点
    解注释则可以使用软件授权的方式进行授权。具体参见附录A
      root: "xxxxxx"
      key: "/home/sti/RobosensePerceptionKeyFile.key"
  lidar:
    common:
      data_fusion: true # 多雷达感知时, 是否进行原始数据级融合, true表示前融合算法,
      false表示后融合算法。默认为true。
    sub:
      - ai_detection:
          strategy: "PolarisAiDetection" # ai感知方法
          PolarisAiDetection:
            use_cuda_acc: true # 是否使用GPU加速, 默认为true
            bbox_mode: "reg_box" # 输出检测框类别, 紧缩框"tight", 角度回归
            框"reg_ang", 回归框"reg_box"
            detect_range: # 检测范围
              xmin: -100.
              xmax: 100.
              ymin: -80.
              ymax: 80.
              zmin: -3.
              zmax: 3.5
  save_result: # DEFAULT
    enable: false # 是否开启结果保存
    save_percept_result: true # 是否保存感知结果 (需保证enable开启)
    save_pcd: true # 是否保存点云为pcd (需保证enable开启)
    save_dir: "/media/sti/1TB-HDD/Tmp/save" # 存储位置
```

```

#* Localization *#
localization:
  common:
    localization_mode: 0 #0: use gps 2: use GPS coordinate 4: use xyz
coordinate
    use_fusion_cloud: true # true: use fusion cloud to do localization
    grid_map: localization_map.gridmap
    rsmmap: localization_map.rsmmap

  result_sender:
    - method: Ros #发送定位结果(目前只支持ros方式)
      localization_freq: 30
      send_pos_ros: true #@type: bool @detail: if true, the
localization algorithm result will be sent through ROS
      send_pos_ros_topic: /rs_pose
      send_fix_ros_topic: /rs_fix
      send_map_ros: true
      send_map_ros_topic: /rs_map
      send_path_ros: true
      send_path_ros_topic: /rs_path

    # - method: Proto
    #   localization_freq: 30
    #   send_pos_and_path: true
    #   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

#* LiDAR & Sensors *#
sensor:
  lidar:
    common:
      msg_source: 2 # lidar msg来源设置
# 0--不使用雷达
# 1--在线雷达
# 2--ROS发送的packet信息
# 3--ROS发送的points 信息
      send_packets_ros: false # 如果设置为true, 原始packets 会通过ROS发出
      send_points_ros: true # 如果设置为true, 原始的points 会通过ROS发出

    lidar:
      - driver:
          include: /system_config/sensor_config/lidar/mems_front/lidar.yaml #
驱动的配置文件
          ros:
            ros_recv_points_topic: /mems_front/rslidar_points # 如果msg_source设
置为3,则需要设置ROS接收的points topic

```

```

    ros_recv_packets_topic: /mems_front/rslidar_packets # 如果msg_source
设置2,则需要设置为ROS接收的packets topic
    ros_send_points_topic: /mems_front/rslidar_points # ROS发送的points
topic
    ros_send_packets_topic: /mems_front/rslidar_packets # ROS发送的packets
topic

can:      # can data(协议需定制)
    msg_source: Ros      #在线和接收Ros消息两种格式 Online or Ros
    send_ros_msg: false
    ros_recv_topic: /can_data
    ros_send_topic: /can_data
    tag: Lxone # Lxone
    kvaser_channel: 0

external_pose: # 外接pose信息(信息格式需要定制转换)
    msg_source: Sq
    ros_recv_topic: /odo_pos320_lane/fused_gps

```

3.2 calibration.yaml配置

```

lidar:
  - parent_frame_id: /base_link # 车体坐标系
    frame_id: /middle_lidar # 雷达的frame id
    device_type: RSM1 # 雷达类型
    x: 0 # 雷达坐标系到车体坐标系pose(x,y,z,roll,pitch,yaw)
    y: 0
    z: 1.73
    roll: 0
    pitch: 0.02
    yaw: 0
base:
  frame_id: /base_link

```

注：各模块参数说明请参考[附录E](#)

4、快速启动

4.1 环境依赖

SDK已经实现纯C++，除AI运行所需Runtime（不同平台的推理引擎，例如Nvidia的TensorRT，Xilinx的Vitis等）外，不依赖第三方库，但为了方便调试和展示，建议配合ROS使用。SDK建议运行软件和硬件环境依赖请参考[附录E](#)。

4.2 下载

```

#本部分仅robosense内部人员使用，如有需求请联系相关技术人员获取sdk。
git clone git@gitlab.robosense.cn:PerceptionGroupShared/rs_sdk_3.1_release.git
--recursive

```

4.3 编译

```
cd rs_sdk
./install_dependency.sh #安装依赖, 仅第一次使用时需要运行此脚本, 然后重启
mkdir build
cd build && cmake .. && make
```

4.4 运行

- 接收ROS点云数据

通过驱动解析ROS发送的packets数据, 发送ROS点云(请参考rslidar_sdk说明使用)。

- 参数说明usr_config.yaml中sensor的msg_source设置为2, ros_rcv_packets_topic设置为bag包发送的topic
- 运行自动标平程序, 获取雷达pose, 结果会自动填写到参数说明里面的calibration.yaml

```
cd build
./demo/auto_align
#注: 该功能仅第一次运行时使用, 如更改雷达安装位置或其他数据, 则需重新运行
#该功能依赖ros与pcl, 默认不进行编译, 如需使用请在/demo/CMakeLists.txt打开
auto_align编译选项
```

- 运行rs_sdk_demo启动感知程序:

```
cd build
./demo/rs_sdk_demo
```

- 雷达在线运行

运行在线雷达不需要另外跑驱动, 只需在线雷达的demo便可以接收点云输出感知结果:

- 配置在线雷达参数, 在usr_config.yaml 设置msg_source为1, 然后设置system_config/sensor_config/lidar/mems_front/lidar.yaml中设置正确的msop_port和difop_part等参数(如使用pcap文件, 需打开read_pcap开关, 并配置pcap文件绝对路径)。
- 运行自动标平程序, 获取雷达pose, 结果会自动填写到参数说明里面的calibration.yaml文件中:

```
cd build
./demo/auto_align
#注: 该功能仅第一次运行时使用, 如更改雷达安装位置或其他数据, 则需重新运行
#该功能依赖ros与pcl, 默认不进行编译, 如需使用请在/demo/CMakeLists.txt打开
auto_align编译选项
```

- 运行rs_sdk_demo启动感知程序:

```
cd build
./demo/rs_sdk_demo
```

4.5 查看感知结果

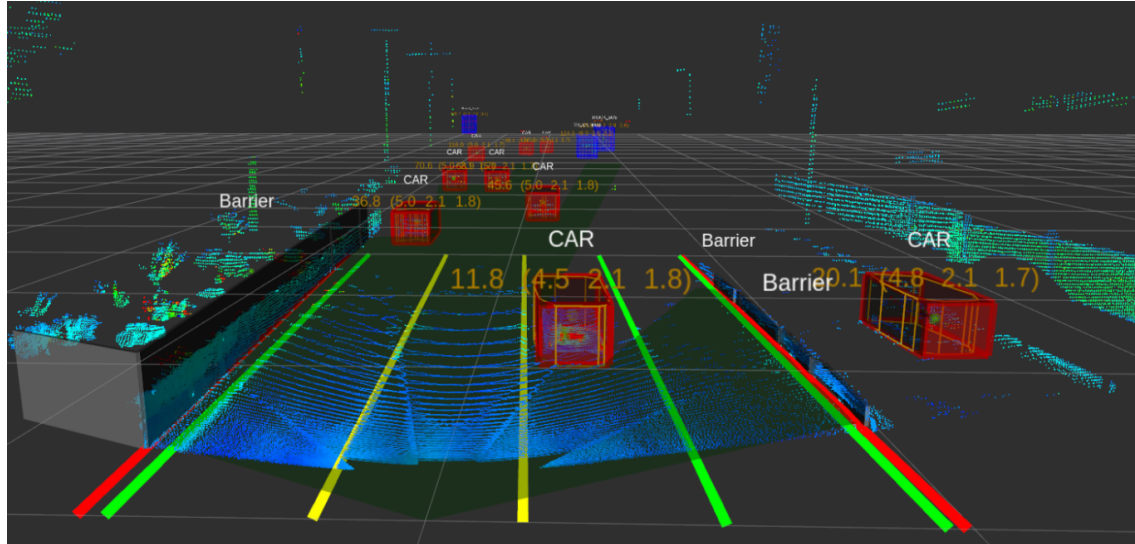
- 加载感知rviz文件

加载config/rviz目录下的rviz文件显示感知结果，命令如下：

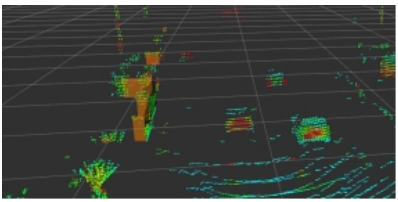
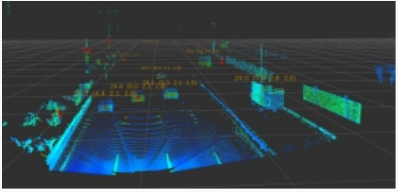
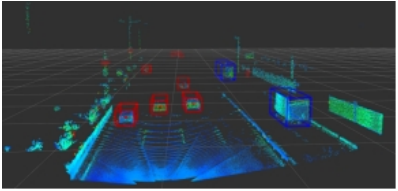
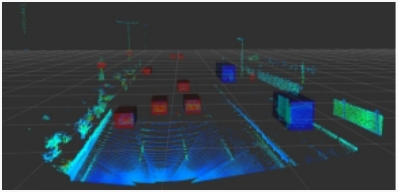
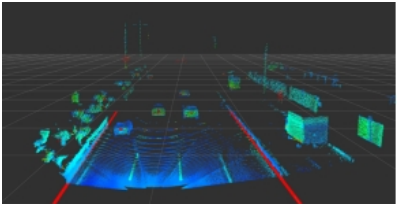
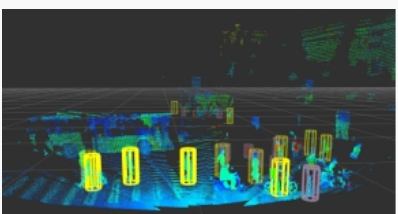
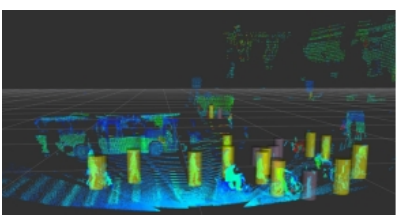
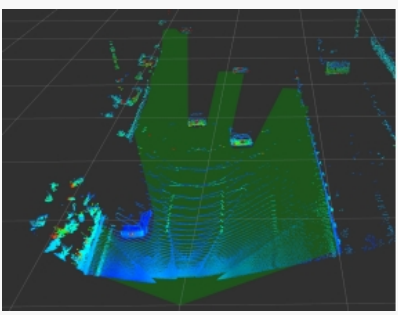
```
rviz -d config/rviz/perception.rviz
```

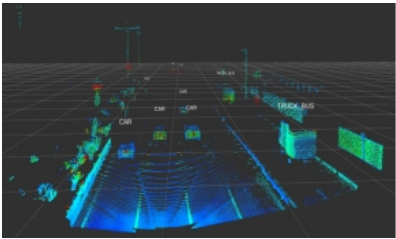
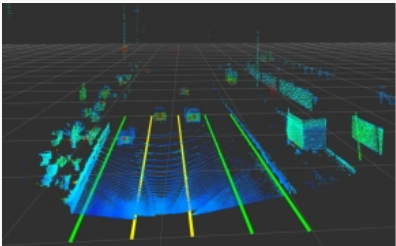
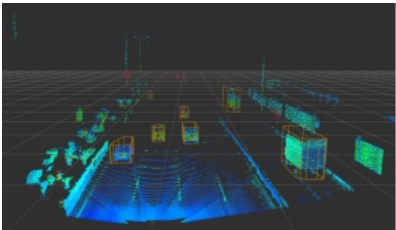
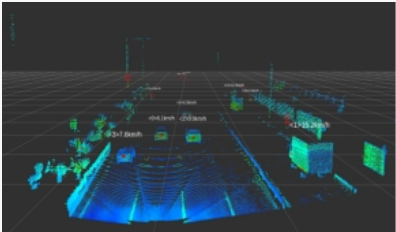
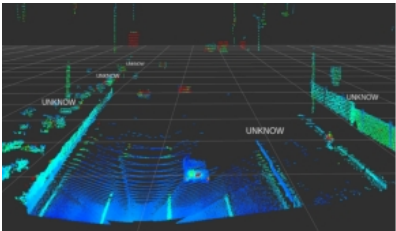
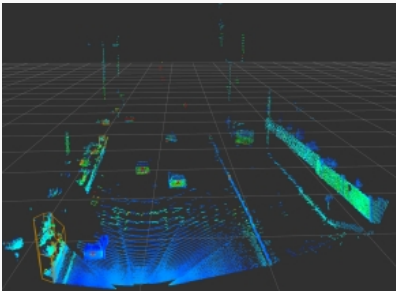
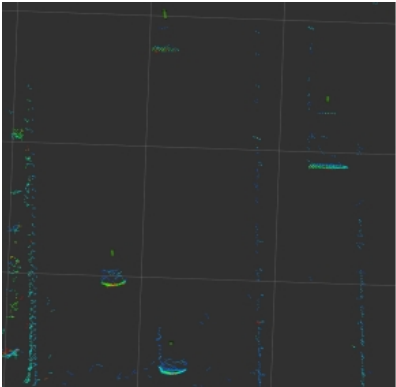
- 感知结果显示

感知结果rviz显示结果如下图所示：



- Rviz感知显示信息(MakerArray/namespaces)

| 显示列表 | 说明 | 图例 |
|--------------|----------------------------|--|
| attention | 一度目标 |  |
| box_info | 目标框中心点距离车体的距离 和 目标框的size信息 |  |
| box | 目标框 |  |
| cube | 目标立方体 |  |
| roadedge | 道路边界 |  |
| cylinder_box | 行人圆柱体框 |  |
| cylinder | 行人圆柱体 |  |
| freespace | 可行驶区域 |  |

| 显示列表 | 说明 | 图例 |
|--------------------|----------------|--|
| label_info | 目标类别信息 |  |
| lane | 车道线 |  |
| polygon | 目标紧缩多边形 |  |
| track_info | 目标跟踪信息 |  |
| unknown_label_info | 一度聚类目标的label信息 |  |
| unknown_polygon | 一度目标紧缩多边形 |  |
| vel_dir | 目标对应车体的速度方向 |  |

| 显示列表 | 说明 | 图例 |
|--------------------|--------|--|
| barrier | 栅栏 |  |
| barrier_label_info | 栅栏类别信息 |  |

5、感知输出列表

感知软件包输出各项感知结果，在应用中以结构体的形式输出。所有的输出默认为车体坐标系（参见附录）：

| 输出内容 | 数据格式 | 说明 |
|--------------------|-------------|--------------------------------|
| objects | Object*N | 前景检测目标输出列表，每个Object信息详见下表 |
| attention_objects | Object*N | 一度目标输出列表，每个Object信息详见下表 |
| lanes | Lane*N | 车道线输出列表，每个Lane信息详见下表 |
| roadedges | Roadedge*N | 道路边界输出列表，每个Roadedge信息详见下表 |
| freespace_ptr | RsFreeSpace | 可行驶区域信息，FreeSpaceInfo信息详见下表 |
| ground_indices | int*N | 地面点云在原始点云中索引 |
| non_ground_indices | int*N | 非地面点云在原始点云中的索引 |
| background_indices | int*N | 背景点云(非地面点云去除目标点云后的点云)在原始点云中的索引 |

5.1 Object 障碍物检测信息

Objects是指由若干个Object组成的列表，其中，每个Object包含如下信息：

| 数据项 | 类型 | 说明 |
|----------------------|-----------|--|
| timestamp | double | 每个object时间戳信息，单位：秒 |
| priority_id | int | 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 |
| exist_confidence | float | 障碍物存在的概率 |
| center | (float,3) | 中心点，表示目标中心点坐标 (x,y,z)，单位：米 |
| center_cov | (float,3) | 目标中心点的不确定度 |
| size | (float,3) | 框尺寸，表示目标框尺寸（长，宽，高），单位：米 |
| size_cov | (float,3) | 目标框尺寸的不确定度 |
| direction | (float,3) | 框朝向，归一化三维向量，定义为框长轴的朝向 |
| direction_cov | (float,3) | 框朝向的不确定度 |
| type | int32 | 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） |
| type_confidence | float | 目标类别置信度 |
| attention_type | int32 | 一度目标类型，NONE表示为非一度目标，ATTENTION表示一度目标 |
| motion_state | int32 | 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 |
| lane_pos | int32 | 目标所属车道标识 |
| tracker_id | int | 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 |
| age | double | 跟踪生命期，表示目标被成功跟踪的时间长度，单位：秒 |
| velocity | (float,3) | 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 |
| relative_velocity | (float,3) | 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 |
| velocity_cov | (float,3) | 速度的不确定度 |
| related_velocity_cov | (float,3) | 相对速度的不确定度 |
| acceleration | (float,3) | 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 |
| acceleration_cov | (float,3) | 加速度的不确定度 |
| angle_velocity | float | 角速度，单位：弧度/秒 |
| angle_velocity_cov | float | 角速度的不确定度 |

| 数据项 | 类型 | 说明 |
|------------------------|------------------|--|
| angle_acceleration | float | 角加速度，单位：弧度/秒平方 |
| angle_acceleration_cov | float | 角加速度的不确定度 |
| anchor | (float,3) | 重心点，三维向量，表示目标分割点云重心点坐标，单位：米 |
| nearest_point | (float,3) | 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 |
| polygon | (float,3) * N | 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 |
| left_point_index | int | 轮廓点中最左侧点索引 |
| right_point_index | int | 轮廓点中最右侧点索引 |
| cloud_indices | (int32) * N | 目标点云索引列表，由目标分割点在原始点云中的索引号组成，N为目标分割点云中点个数 |
| latent_types | (float) * N | 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 |
| size_type | int32 | 形状类型，默认为SMALL，中型大小目标为MEDIUM，大型目标为LARGE |
| mode | int32 | 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， |
| in_roi | bool | 表示障碍物是否在roi区域以内 |
| tracking_state | Int32 | 跟踪状态，UNKNOWN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 |
| geo_center | (float,3) | 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 |
| geo_size | (float,3) | 三维向量，根据目标分割点云计算的紧缩框大小，单位：米 |
| trajectory | (float,3) * N | 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 |
| history_velocity | (float,3) * N | 跟踪历史速度，*N*为序列缓存长度，可设置 |
| history_type | (int32) * N | 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 |
| gps_mode | int32 | 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 |

| 数据项 | 类型 | 说明 |
|---------------|--------|-------|
| gps_longitude | double | GPS经度 |
| gps_latitude | double | GPS纬度 |
| gps_altitude | double | GPS高程 |

5.2 RsFreeSpace可行驶区域信息

RsFreeSpace 以vector结构保存三维点坐标(points)和对应的置信度，所有点围成的封闭区域为可行驶区域，包含如下信息：

| 数据项 | 类型 | 说明 |
|---------------|-------------|-----------------------|
| fs_pts | (float,3)*N | 可行驶区域边界点三维坐标，N表示边界点数目 |
| fs_confidence | (float) * N | 可行驶区域边界点的置信度，N表示边界点数目 |

5.3 Lane 车道线信息

| 数据项 | 类型 | 说明 |
|----------------|-------------|---|
| lane_id | int32 | 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 |
| curve | (float,4) | 三次曲线拟合参数：4个 |
| end_point | (float,2)*2 | 车道线起始点和终止点 |
| measure_status | int32 | 车道线获得方式：检测(DETECTION)和预测(PREDICTION) |
| confidence | float | 检测置信度 |

5.4 RoadEdge 道路边界信息

| 数据项 | 类型 | 说明 |
|----------------|-------------|---------------------------------------|
| roadedge_id | int32 | 道路边界位置ID，左侧LEFT，右侧RIGHT，以及未知UNKNOWN |
| curve | (float,4) | 三次曲线拟合参数：4个 |
| end_point | (float,2)*2 | 道路边界起始点和终止点 |
| measure_status | int32 | 道路边界获得方式：检测(DETECTION)和预测(PREDICTION) |
| confidence | float | 检测置信度 |

6、感知通信方式

详见通信文档。

附录

A: 软件授权方式说明

- 概述

针对没有USB条件的设备，SDK3.1 可以使用验证 RobosensePerceptionKey.key 文件的方式来进行授权验证。用户只需要使用 Robosense 提供的 RobosenseRegister_SDK 软件生成一个 RobosensePerceptionKeyFile.bin 文件，然后将这个文件发送给 Robosense 工作人员即可。

下面将针对 RobosenseRegister_SDK 的使用方法以及如何将 SDK3.1 配置成验证 RobosensePerceptionKey.key 文件的模式进行详细说明。

- RobosenseRegister_SDK 的使用方法

1. 下载和运行

- x86平台Ubuntu系统下载地址：<http://120.237.87.134:15000/d/f/682987505749900329>
- aarch64平台Ubuntu系统下载地址：<http://120.237.87.134:15000/d/f/682987497138994208>

下载完成并解压缩后，直接运行目录中的可执行文件“RobosenseRegister_SDK”即可打开软件。

2. 生成 RobosensePerceptionKeyFile.bin 文件

根据软件界面最下方的说明进行操作。为保证密码的正确性，建议勾选“显示密码”选项以做核对。



当 RobosensePerceptionKeyFile.bin 文件生成完毕以后，可以发送给软件中的电子邮件地址，也可以发送给与您沟通的 Robosense 工作人员。

- SDK3.1配置授权方式说明

1. 将 Robosense 工作人员发送给您的 RobosensePerceptionKey.key 文件保存至本地目录。
2. 打开 SDK3.1 中 config 目录下的 usr_config.yaml 文件，参考如下代码块进行修改。

```
# 本代码块为usr_config.yaml的一部分

## Perception ##
perception:
  general:
    model: system_config/perception_config/model
    authorization: # 将authorization节点解注释则可以使用
RobosensePerceptionKey.key 文件进行授权
    root: "j" # 这里填写生成 RobosensePerceptionKeyFile.bin 文件时所填写
    的 Root 用户密码
    key: "/home/sti/RobosensePerceptionKeyFile.key" # 这里填写
RobosensePerceptionKey.key 文件的路径。
```

注意：请不要漏了双引号！

- 按照上述步骤修改完 usr_config.yaml 以后，SDK3.1 的授权验证方式便切换成功。此时，无需 USB_KEY 也可以正常运行 SDK3.1 提供的 demo。

B: API使用说明

- 以下通过p1_perception_test来简单说明如何使用sdk感知功能：

```
using namespace robosense;
using namespace robosense::perception;
static int g_frame_cc = 0;
static RvizDisplay::Ptr g_rviz_display_ptr;
static Perception::Ptr g_perception_ptr;
// 获得点云后的回调函数，这里会进行点云的感知，以及感知结果的发送
void fullscanCallback(const pcl::PointCloud<pcl::PointXYZI>::Ptr &pts_msg)
{
    unsigned long long int umTimestamp = pts_msg->header.stamp;
    double timestamp = static_cast<double>(g_frame_cc) * 0.1;
    if (umTimestamp > 0.1) {
        timestamp = pts_msg->header.stamp * 1e-6;
    }
    // 构建感知使用的消息，注意可以在sub_lidar_msgs_map中配置多路雷达
    RsPerceptionMsg::Ptr msg_ptr(new RsPerceptionMsg);
    RsLidarFrameMsg::Ptr lidar_msg_ptr(new RsLidarFrameMsg);
    // 往单个雷达的消息中填点云
    transPcCloudToRslCloud(pts_msg, lidar_msg_ptr->scan_ptr);
    lidar_msg_ptr->frame_id = "/middle_lidar";
    lidar_msg_ptr->timestamp = timestamp;
    msg_ptr->sub_lidar_msgs_map["/middle_lidar"] = lidar_msg_ptr;
    // 使用感知模块对点云进行感知
    g_perception_ptr->perception(msg_ptr);
    // 通过ros发送感知结果，可使用RVIZ来显示
    g_rviz_display_ptr->display(msg_ptr);
    g_frame_cc++;
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "p1_perception_demo");
    ros::NodeHandlePtr node_ptr;
    node_ptr.reset(new ros::NodeHandle);
    // 加载所有感知的配置
    std::string config_path = std::string(PROJECT_PATH) + "/config";
```



```

    RsYamlNode node = rsConfigParser(config_path);
// 初始化感知
    RsYamlNode perception_node;
    rsYamlSubNode(node, "perception", perception_node);
    g_perception_ptr.reset(new Perception);
    g_perception_ptr->init(perception_node);
// 初始化发送
    RsYamlNode rviz_display_node;
    rsYamlSubNode(perception_node, "rviz", rviz_display_node);
    g_rviz_display_ptr.reset(new RvizDisplay(node_ptr, rviz_display_node));
// 加载点云topic的配置
// local_yaml_file配置了不同雷达点云的topic，驱动中配置的发送topic要与此处一致
    RsYamlNode local_config_node;
    std::string local_yaml_file = std::string(PROJECT_PATH) +
"/test/config.yaml";
    if (!rsLoadYamlFile(local_yaml_file, local_config_node)) {
        RERROR<<"load local file failed!";
        RS_THROW("");
    }
    std::string topic = "/mems/rslidar_points";
    rsYamlRead(local_config_node, "main_input_cloud_topic", topic);
// 订阅点云的topic，一旦接收到点云就会调用回调函数fullscanCallback来进行感知
    ros::Subscriber sub_fullscan = node_ptr->subscribe(topic, 1,
fullscanCallback);
    RINFO<<"ready!";
    ros::spin();
    return 0;
}

```

如果希望单独调用感知sdk中的子模块，以下给出了核心模块的调用方式，具体可参考smartsensor_perception中的使用方式。

- **ai检测模块**

实现ai的检测，输入为点云，输出检测目标、地面点分割、背景点分割和非地面点分割。对应类为AiDetection。

ai检测模块的调用接口定义如下：

```

void perception(const RsLidarFrameMsg::Ptr& msg_ptr);
参数:
msg_ptr (RsLidarFrameMsg::Ptr) - 消息中需要包含:
    scan_ptr (RsPointCloudGPT::Ptr) - 点云

```

获取ai检测结果的接口如下：

```

void getResult(Any::Ptr& any);
参数:
any (Any::Ptr) - any指针指向ai检测结果，一个AiDetectionMsg的实例，其中包含:
    objects (VecObjectPtr) - 检测出来的目标
    ground_indices - 地面点索引
    non_ground_indices - 非地面点索引
    background_indices - 背景点索引

```

以下给出ai检测模块的具体调用方法：

```
// 定义AiDetection类的实例
lidar::AiDetection::Ptr lidar_ai_detection_ptr_(new lidar::aiDetection);
// 初始化AiDetection实例,
// ai_detection_node为AiDetection配置, 参考配置文件
lidar_ai_detection_ptr_>init(ai_detection_node);
// 调用ai检测
lidar_ai_detection_ptr_>perception(msg_ptr_>rs_lidar_result_ptr);
// 获取ai检测结果
lidar_ai_detection_ptr_>getResult(any_ptr);
```

• 传统聚类分割模块

实现通过传统聚类分割检测目标的功能, 输入为点云、背景点索引, 输出为通过传统聚类分割检测到的目标。对应类为Segmentor。传统聚类分割检测的调用接口定义如下:

```
void perception(const RsLidarFrameMsg::Ptr& msg_ptr);
参数:
msg_ptr (RsLidarFrameMsg::Ptr) - 消息中需要包含:
    scan_ptr (RsPointCloudGPT::Ptr) - 点云
    background_indices (VecInt) - 背景点索引
```

获取传统聚类分割检测结果的接口如下:

```
void getResult(Any::Ptr& any);
参数:
any (Any::Ptr) - any指针指向传统聚类分割检测的结果, 一个SegmentorMsg的实例, 其中包含:
    objects (VecObjectPtr) - 检测出来的目标
    processed (bool) - 是否调用过
```

以下给出传统聚类分割检测模块的具体调用方法:

```
// 定义Segmentor类的实例
lidar::Segmentor::Ptr lidar_segmentor_ptr_(new lidar::Segmentor);
// 初始化Segmentor实例,
// segmentor_node为Segmentor配置, 参考配置文件
lidar_segmentor_ptr_>init(segmentor_node);
// 调用传统聚类分割检测
lidar_segmentor_ptr_>perception(msg_ptr_>rs_lidar_result_ptr);
// 获取传统聚类分割检测结果
lidar_segmentor_ptr_>getResult(any_ptr);
```

• 一度目标和freespace检测模块

实现一度目标和freespace的检测, 输入为背景点, 输出一度目标和freespace。对应类为BasicDetection。

一度目标和freespace检测模块的调用接口定义如下:

```
void perception(const RsLidarFrameMsg::Ptr& msg_ptr);
参数:
msg_ptr (RsLidarFrameMsg::Ptr) - 消息中需要包含:
    scan_ptr (RsPointCloudGPT::Ptr) - 点云
    background_indices (VecInt) - 背景点索引
```

获取一度目标和freespace检测结果的接口如下：

```
void getResult (Any::Ptr& any);
```

参数：

any (Any::Ptr) - any指针指向一度目标和freespace检测结果，一个BasicDetectionMsg的实例，其中包含：

attention_objects (VecObjectPtr) - 检测出来的一度目标

freespace_ptr (RsFreeSpace::Ptr) - freespace边界上的点

以下给出一度目标和freespace检测模块的具体调用方法：

```
// 定义BasicDetection类的实例
lidar::BasicDetection::Ptr lidar_basic_detection_ptr_ (new
lidar::BasicDetection);
// 初始化BasicDetection实例,
// basic_detection_node为BasicDetection配置, 参考配置文件
lidar_basic_detection_ptr_ ->init (basic_detection_node);
// 调用一度目标和freespace沿检测
lidar_basic_detection_ptr_ ->perception (msg_ptr_ ->rs_lidar_result_ptr);
// 获取一度目标和freespace检测结果
lidar_basic_detection_ptr_ ->getResult (any_ptr);
```

● 车道线与路沿检测模块

实现车道线和路沿的检测，输入地面点和背景点，输出车道线和路沿的曲线。对应类为RoadDetection。

车道线和路沿检测模块的调用接口定义如下：

```
void perception (const RsLidarFrameMsg::Ptr& msg_ptr);
```

参数：

msg_ptr (RsLidarFrameMsg::Ptr) - 消息中需要包含：

scan_ptr (RsPointCloudGPT::Ptr) - 点云

ground_indices (VecInt) - 地面点的索引

background_indices (VecInt) - 背景点索引

获取车道线和路沿结果的接口如下：

```
void getResult (Any::Ptr& any);
```

参数：

any (Any::Ptr) - any指针指向车道线和路沿检测结果，一个RoadDetectionMsg的实例，其中包含：

lanes (VecLanePtr) - 检测出来车道线

roadedges (VecRoadedgePtr) - 检测出来的路沿

以下给出车道线与路沿检测模块的具体调用方法：

```
// 定义RoadDetection类的实例
lidar::RoadDetection::Ptr lidar_road_detection_ptr_(new
lidar::RoadDetection);
// 初始化RoadDetection实例,
// road_detection_node为RoadDetection配置, 参考配置文件
lidar_road_detection_ptr_>init(road_detection_node);
// 调用车道线与路沿检测
lidar_road_detection_ptr_>perception(msg_ptr_>rs_lidar_result_ptr);
// 获取车道线与路沿检测结果
lidar_road_detection_ptr_>getResult(any_ptr);
```

- 跟踪模块

主要实现对目标的跟踪, 为每个目标赋一个唯一的跟踪ID, 并估计目标的运动状态。对应类为Tracking。

跟踪模块的调用接口定义如下:

```
void perception(const RsLidarFrameMsg::Ptr& msg_ptr);
参数:
msg_ptr (RsLidarFrameMsg::Ptr) - 包含检测对象的消息
    scan_ptr (RsPointCloudGPT::Ptr) - 点云
    objects (VecObjectPtr) - 检测对象的集合
```

获取跟踪结果的接口如下:

```
void getResult(Any::Ptr& any);
参数:
any (Any::Ptr) - any指针指向跟踪结果, 一个TrackingMsg的实例, 其中包含:
    objects (std::vector<RsTrackObject::Ptr>) - 当前帧目标的跟踪结果
    trajectories (std::vector<RsTrajectory::Ptr>) - 目标的轨迹的集合
```

以下给出了跟踪的具体调用方法, 其中跟踪结果中对象的类型为RsTrackObject, 可根据实际情况, 将相应的值赋给检测对象:

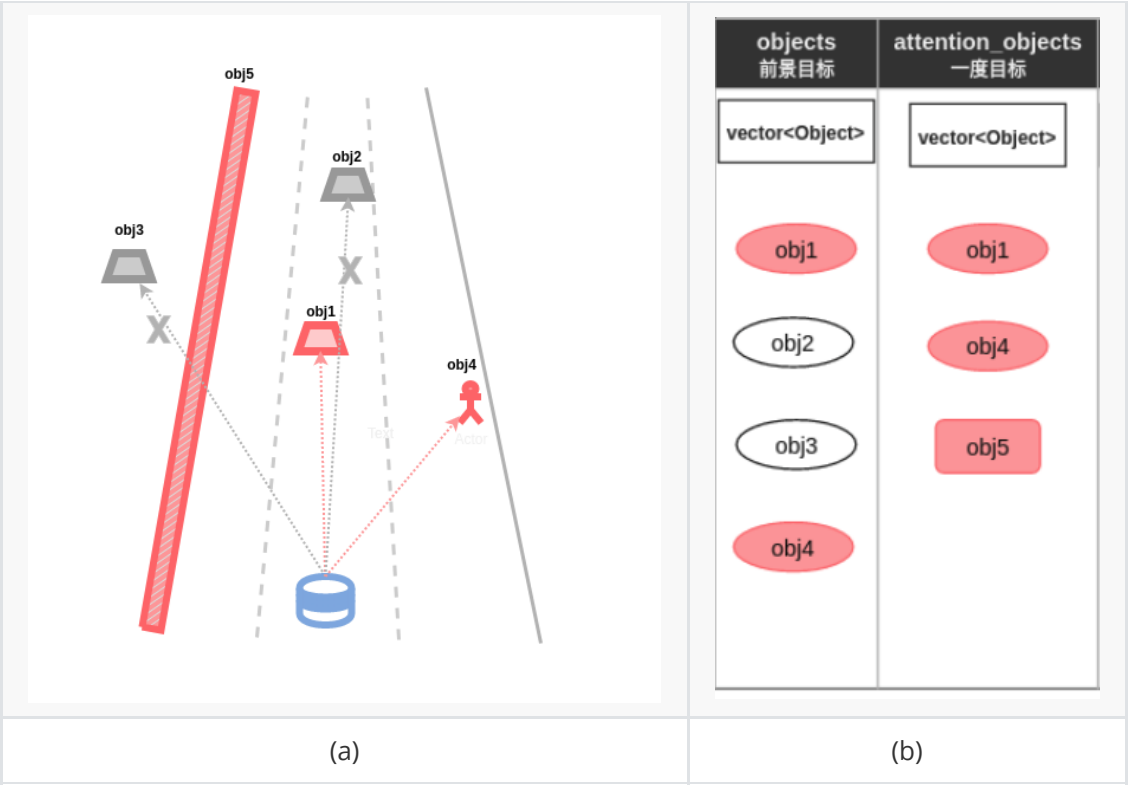
```
// 定义Tracking类的实例
lidar::Tracking::Ptr lidar_tracking_ptr_(new lidar::Tracking);
// 初始化Tracking实例, tracking_node为Tracking配置, 参考配置文件
lidar_tracking_ptr_>init(tracking_node);
// 调用跟踪
lidar_tracking_ptr_>perception(msg_ptr_>rs_lidar_result_ptr);
// 获取跟踪结果
lidar_tracking_ptr_>getResult(any_ptr);
```

C: 相关定义说明

- 一度目标

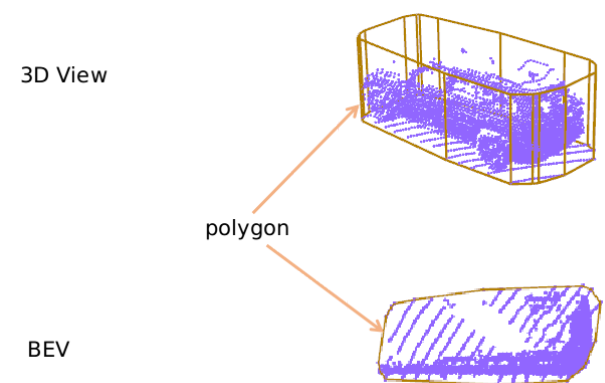
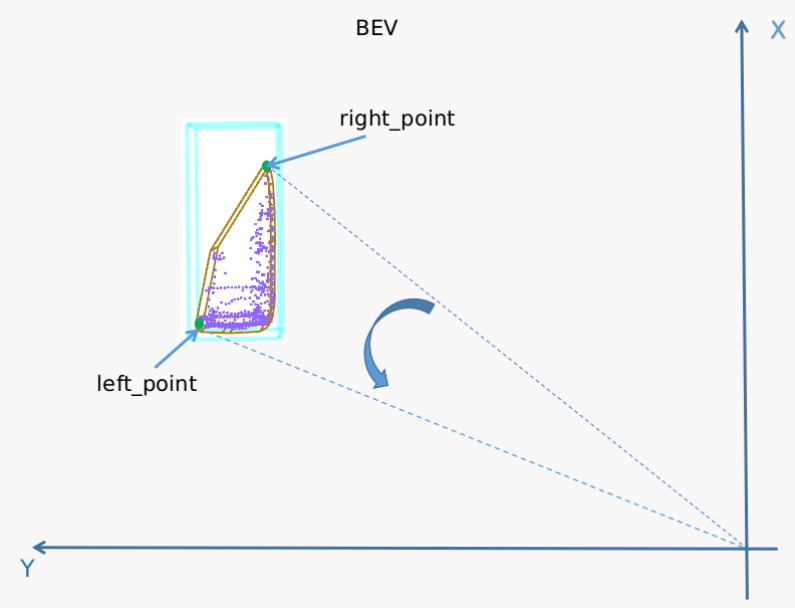
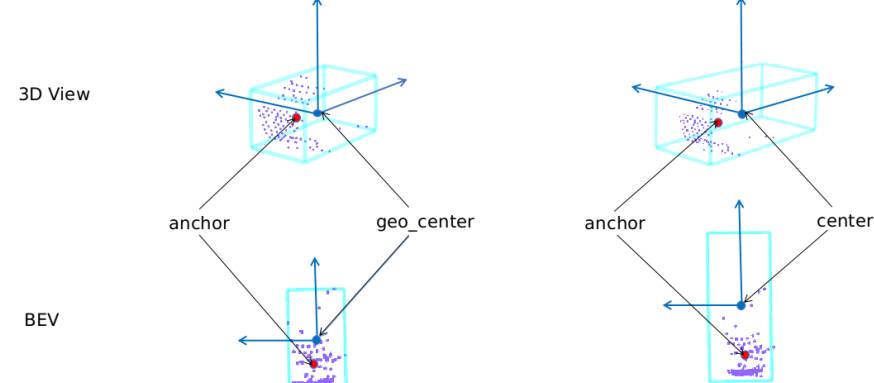
一度目标指的是雷达可直接观测到的目标, 目标与雷达的连线之间不存在其他的障碍物, 这类目标称为一度目标, 如下图(a)所示, 一度目标用红色标识。通俗的说, 一度目标代表ego-car可直接与其发生碰撞的目标。一度目标可以是前景目标, 如人或者车, 也可以是护栏/树木/路沿等非前景目标。

一度目标与前景目标共享相同的结构体信息，但一度目标不参与跟踪，因此没有跟踪信息输出。一度列表中包含已经存在于前景目标列表中的目标，还包括背景中新检测的目标，这类目标分类类型一律为UNKNOWN，一度目标与前景目标的关系如下图(b)所示



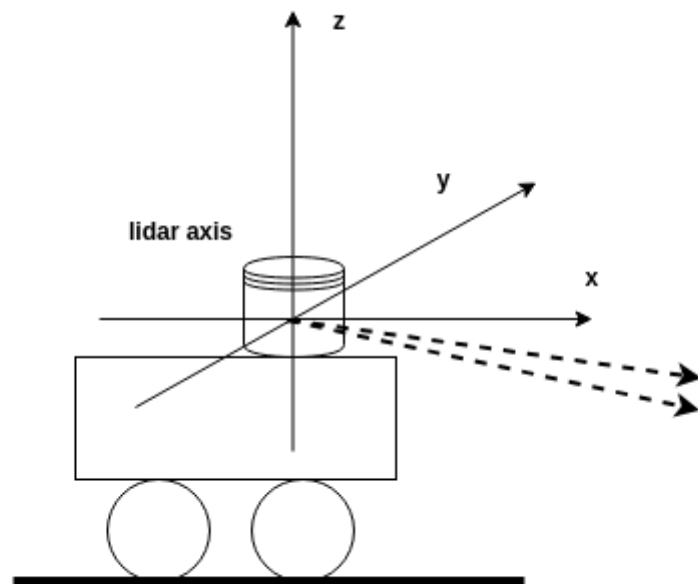
• Object相关说明

- polygon:** 轮廓点，二维投影平面上包围障碍物多边形的系列轮廓点。
- left_point:** 最左边点，是一个3维向量，表示从上往下鸟瞰绕着车体逆时针旋转，最晚碰到障碍物的那个点。
- right_point:** 最右边点，是一个3维向量，表示从上往下鸟瞰绕着车体逆时针旋转，最早碰到障碍物的那个点。
- anchor:** 重心点,是一个3维向量，分别表示在(x,y,z)三个维度下的坐标。
- geo_center:** 障碍物根据点云实际大小计算的中心点，即将目标框沿朝向方向缩小至贴紧点云后计算得到的目标框中心点。

| 属性 | 示意图 |
|--------------------------|--|
| polygon |  <p>3D View</p> <p>BEV</p> |
| left_point、right_point |  <p>BEV</p> <p>left_point</p> <p>right_point</p> |
| anchor、center、geo_center |  <p>3D View</p> <p>BEV</p> <p>anchor</p> <p>geo_center</p> <p>center</p> |

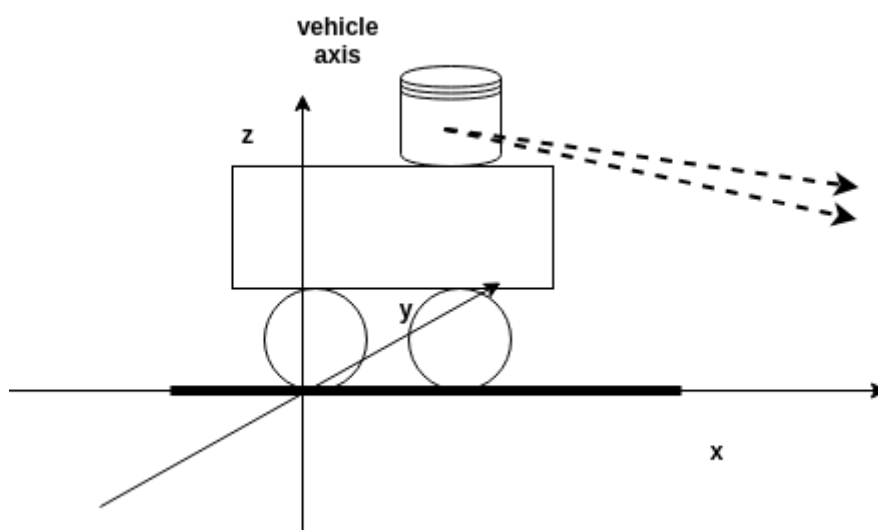
D: 坐标系说明

- 雷达坐标系 (lidar_axis)
以雷达中心为坐标中心的坐标系。每一个雷达保存的雷达坐标系的pose永远是(0,0,0,0,0,0)，因为是从雷达坐标系变换到自身。



- 车体坐标系 (**vehicle_axis**)

以车体后轴与地面相交处为中心，x-y平面与地面平行，x轴指向车辆前进方向的坐标系。



- 全局坐标系 (**global_axis**)

全局坐标系(global_axis)，指的是在配合定位或者其他有高精度地图的应用中，存在一个全局位置的坐标系，每个雷达(在这里不再有主雷达概念)相对与这个全局位置的pose就是全局坐标系pose。

E: 感知参数说明

用户可配置参数全位于config文件夹下，usr_config文件夹中的文件优先级最高，会默认替换system_config中对应位置的参数。关于usr_config文件夹下配置文件的配置方法请参考小节3（参数说明）。System_config文件夹中为工程更详细的配置项，communication_config为通信相关的配置项，perception_config为感知相关的配置项，preprocessing_config为传感器同步相关的配置项，sensor_config为sensor相关配置项，localization_config为定位相关配置项。

- rs_perception.yaml感知配置参数总表

```

## General ##
general:
  _application_: ""
  tictoc_average: true # 是否计算运行平均时间
  aixe_out: "GLOBAL_AXIS" # 感知结果输出坐标系, 可选LIDAR_AXIS, VEHICLE_AXIS和
GLOBAL_AXIS
  _base_dir_: ""
  model: /system_config/perception_config/model # 模型存放路径
calibration:
  include: /usr_config/calibration.yaml # 标定文件存放路径
lidar:
  include: /system_config/perception_config/lidar/rs_lidar.yaml # 雷达感知的
配置文件存放路径
camera:
  include: /system_config/perception_config/camera/rs_camera.yaml # 图像感
知的配置文件存放路径

## HdMap ##
hdmap: # 事件检测功能。当usr_config.yaml中的application选择为V2r策略或Pseries
时。该功能会根据下方配置进行调用。
  enable: false # 是否开启事件检测功能。
  filter_obj: false # true: 开启ROI物体过滤功能
  event_detect: false # true: 开启ROI内事件检测功能。
  rois:
    include: /system_config/perception_config/hdmap/roi.yaml # roi具体配置文
件存放路径。该文件由工具生成, 详情联系robosense技术支持。

## V2rPerception ##
V2rPerception: # 当usr_config.yaml中的application为V2r的时候, 需要配置全局
pose。
  global_pose:
    x: 0
    y: 0
    z: 0
    roll: 0
    pitch: 0
    yaw: 0
## gps ##
gps: #DEFAULT
  gps_longitude: 113.961121
  gps_latitude: 22.584291
  gps_altitude: 0

## Rviz ##
rviz:
  prefix: "" # DEFAULT
  strategy: "EFFICIENT" #DEFAULT "EFFICIENT", "ORIGIN" # 两种不同的结果显示配色
模式。
  display_axis: "VEHICLE_AXIS" # "VEHICLE_AXIS", "GLOBAL_AXIS" # 结果显示基于
的坐标系。
  map:
    - "" #DEFAULT address file for map
    - "" #DEFAULT address file for map
  road:
    - "" #DEFAULT
  frame_id: "/base_link" #DEFAULT
  map_frame_id: "/map" #DEFAULT

```



```

pub_cloud_keys: # 可供显示的点云, 不需要的直接注释即可。注意缩进。
- "origin" #publish origin cloud          原始点云
- "ground" #publish ground cloud          地面点点云
- "background" #publish background cloud  背景点点云
- "non_ground" #publish non_ground cloud  非地面点点云
- "clusters" #publish clusters cloud      通过聚类得到的结果的点云
- "semantic" #publish semantic cloud      语义点云

pub_marker_keys: # 可供显示的感知结果
#   - "acc_dir" #publish acc_dir marker    目标的加速度方向
#   - "atten" #publish attention marker    一度目标
#   - "barrier" #pubulish barrier marker   物理围栏
#   - "box_infos" #publish box_infos marker 除UNKNOWN类型以
外的目标的目标框中心点距离车体的距离和目标框的size信息
#   - "box_lines" #publish box_lines marker 目标类型为
CAR/TRACK_BUS/ULTRA_VEHICLE/BIC的目标框
#   - "cube" #publish cube marker          目标类型为
CAR/TRACK_BUS/ULTRA_VEHICLE/BIC的目标立方体
#   - "cylinder" #publish cylinder marker   目标类型为
PED/CONE的圆柱体
#   - "cylinder_lines" #publish cylinder_lines marker 目标类型为
PED/CONE的圆柱体框
#   - "freespace" #publish freespace marker 可行驶区域
#   - "gps" #publish gps marker            目标的GPS信息
#   - "label_infos" #publish label_infos marker 目标的类别信息
#   - "lane" #publish lane marker          车道线
#   - "polygon" #publish polygon marker    目标的紧缩多边形
#   - "atten_polygon" #publish attention polygon marker 一度目标的紧缩多
边形
#   - "atten_label" #publish attention label marker 一度目标的类别信
息
#   - "roadedge" #publish roadedge marker   路沿
#   - "track_infos" #publish track_infos marker 目标的跟踪信息
#   - "trajectory" #publish trajectory marker 目标的轨迹信息
#   - "vel_dir" #publish vel_dir marker      速度不为0的目标的
速度方向
#   - "scene_struct_cube" #publish scene_struct_cube 交通牌的目标立方
体
#   - "scene_struct_box_lines" #publish scene_struct_box 交通牌的目标框
#   - "scene_struct_info" #publish scene_struct_info text 交通牌的类别信息
#   - "mirror_cube" #publish mirror cube      镜像目标的立方体
#   - "mirror_box_lines" #publish mirror box    镜像目标的目标框

## SaveResult ##
save_result: #DEFAULT 是否保存SDK产生的结果
  enable: false #DEFAULT true表示保存
  save_percept_result: true #DEFAULT 保存感知结果
  save_pcd: true #DEFAULT 保存PCD文件
  save_dir: "/media/sti/1TB-HDD/Tmp/save" #DEFAULT 保存路径, 需确保存在。

## auto_align ##
auto_align: #雷达标平
  max_range_x: 20.
  min_range_x: 2.
  max_range_y: 10.
  min_range_y: -10.
  min_hight: -2.5
  max_hight: 2
  plane_fit_max_distance: 0.05

```

```
update_calibration_file: true # 是否自动更新标定文件
calibration_path: "/config/usr_config/calibration.yaml" # 标定文件保存路径
(需保证文件是存在的)
main_frame_id: "/middle_lidar" # 标定数据的主frame_id
```

- **rs_lidar.yaml**基本感知信息

```
## RsPerception ##
## Common ##
common:
  data_fusion: false # 是否启动数据级融合，若为true则表示前融合算法，false为后融合
  算法，单雷达默认true

## MultiLidar ## # 多雷达感知相关配置
sub:
  - config:
      include: /system_config/perception_config/lidar/middle_lidar/lidar.yaml
  - config:
      include: /system_config/perception_config/lidar/left_lidar/lidar.yaml
  - config:
      include: /system_config/perception_config/lidar/right_lidar/lidar.yaml

## PostFusion ##
post_fusion:
  strategy: "MonkeyPostFusion" # 后融合算法策略
  MonkeyPostFusion:
    iou_thre: 0.1
    reg_box: true
    box_size_check: true
#
##* RefineFilter *#
refine_filter:
  enable: false # 是否启动RefineFilter算法
  strategy: "MoffiRefineFilter" # RefineFilter算法策略
  MoffiRefineFilter:
    filter_range: # filter作用范围
      xmin: -100
      xmax: 100
      ymin: -100
      ymax: 100
      zmin: 0
      zmax: 3
    bev_grid_size: 1.0
    height_threshold_for_ground: 0.25
    height_threshold_for_object: 0.2
#
##* BasicDetection *#
basic_detection:
  enable: true
  strategy: "SimbaBasicDetection"
  SimbaBasicDetection:
    detect_range:
      xmin: -100.0
      xmax: 100.0
      ymin: -50.0
      ymax: 50.0
      zmin: 0.
```

```

        zmax: 3.
        min_hori_angle: -60 # freespace最小极角
        max_hori_angle: 60
#
##* RoadDetection *#
road_detection:
    enable: false # 是否开启该功能
    strategy: "SnakeRoadDetection"
    SnakeRoadDetection:
        detect_range:
            xmin: -10.0
            xmax: 60.0
            ymin: -20.0
            ymax: 20.0
        display_range:
            xmin: 0.0
            xmax: 80
        grid_size_x: 0.2 # 栅格的大小
        grid_size_y: 0.2
        lane_width: 3.2 # 标准车道的宽度
        ground_fusion_num: 1 #多帧叠加
        enable_predict: true # 当点云信息匮乏时, 是否预测车道线
        tracking: true # 是否对车道线进行追踪
#
##* Tracking *#
tracking:
    enable: true # 是否开启该功能
    strategy: "PigeonTracking"
    PigeonTracking:
        predict_time: 0.5 # 目标的生存时间。当一个目标消失的时间超过设定值时, 被认定为为一个新目标。
        history_num: 10 # 历史数据的缓存帧数
        basic_velocity_noise: 0.8 # 当目标的速度小于设定值时, 被认为速度为0。
        match_distance_max: 3.0 # 目标关联的最大距离
        enable_predict_missing: false # 当目标丢失的时候, 是否根据历史数据进行目标补全。
        enable_bsd_tracking: true # 是否追踪目标类型为UNKNOWN的目标
#
##* Postprocessing *#
postprocessing:
    strategy: "EulerPostprocessing"
    EulerPostprocessing:
        with_unknown_objects: true
        barrier_filter: true
        gps_translate: true
        gps_type: center #center/centroid/nearest
        sequence_fusion: # 序列融合
            min_hori_angle: -60 #Fov的大小
            max_hori_angle: 60
            enable_type_fusion: false # 是否对目标进行类别纠正
            enable_box_fusion: false # 是否优化目标的回归框
            enable_delay_output: false # 当目标不能够稳定检测时, 是否延迟目标的输出
            enable_refine_orientation: false # 优化目标的朝向 (主要判定目标的朝向是否需要180调转)
            enable_estimate_object_uncertainty: true # 计算目标的不确定度
            enable_dynamic_static_detection: true # 计算目标的动静态
            enable_estimate_exist_confidence: true # 计算目标的存在概率
#

```

```

##* Mirror_detection##
mirror_detection:
  enable: true # 是否开启镜像目标检测功能
  strategy: "PhantomMirrorDetection"
  PhantomMirrorDetection:
    range: # 检测范围
      xmin: 0
      xmax: 100
      ymin: -60
      ymax: 60
    shelter_thre_min: 0.8 # 遮挡阈值

```

- **lidar.yaml**单个雷达感知配置参数

```

##* Preprocessing *#
preprocessing:
  strategy: "DramaPreprocessing" # 前处理策略
  DramaPreprocessing:
    _frame_id_: ""
    vehicle_filter:
      enable: false # 是否开启vehicle filter
      xmin: -2. # vehicle filter过滤的范围
      xmax: 2.
      ymin: -2.
      ymax: 2.
      zmin: -2.
      zmax: 2.
    lidar_filter:
      enable: false # 是否开启lidar filter
      xmin: -2. # lidar filter过滤的范围
      xmax: 2.
      ymin: -2.
      ymax: 2.
      zmin: -2.
      zmax: 2.

##* CnnDetection *#
ai_detection:
  enable: true # 是否开启该功能
  strategy: "PolarisAiDetection"
  PolarisAiDetection:
    _frame_id_: "" # 默认即可
    _lidar_type_: "" # 默认即可
    device_id: 0 # 默认即可
    enable_fp_16: false
    max_workspace: 30 # 默认即可
    use_cuda_acc: true
    bbox_mode: "reg_box" # "tight", "reg_ang", "reg_box"
    detect_range:
      xmin: 0.0
      xmax: 200.
      ymin: -50
      ymax: 50
      zmin: -3.
      zmax: 3.5
    regress_height_thd: 0.5 # 默认即可。前景栅格里，点的高度比框高+0.5高的点为背景
    regression_box_confidence_thre: 0.0 # 预留参数，默认即可

```

```

car_nms_thres: 0.1 # 默认即可。两车之间的iou大于该阈值即被合并
small_obj_nms_thres: 0.3 # 默认即可。两个小物体之间的iou大于该阈值即被合并
min_pts_num: 3 # 默认即可。一个目标拥有的最少的点的个数
ground_height_thd: 1.2 # 默认即可。地面栅格中的地面点需要低于该阈值
ground_h_error_thd: 0.2 # 默认即可。地面栅格中，地面点与最低点的高度差需要低于
该阈值

encrypt: true # 预留，默认即可
rotate_angle: 0 # 检测范围旋转的角度

##* GroundFilter *#
ground_filter:
  enable: true # 是否开启该功能（当AI开启和该模块同时开启时，该模块将失效）
  strategy: "BearGroundFilter"
  BearGroundFilter:
    _frame_id_: "" # 默认即可
    detect_range:
      xmin: 0.
      xmax: 50.
      ymin: -50.
      ymax: 50.
      zmin: -1.
      zmax: 1.5
    unit_size: # bev栅格大小
    max_herr: 0.1 # 栅格高度差
    max_h: 0.5 # > 0.3

##* SegMentor *#
segmentor:
  enable: false # 是否开启该功能
  strategy: "BearSegmentor"
  BearSegmentor:
    _frame_id_: ""
    detect_range:
      xmin: 0.
      xmax: 50.
      ymin: -100.
      ymax: 100.
      zmin: -1.0
      zmax: 2.5
    unit_size: 0.5 # BEV栅格大小
    seg_scan_x: 1. # 聚类生长阈值（x方向）
    seg_scan_y: 1.
    seg_min_pts: 5 # 每个簇中拥有的最小点个数

##* scene_struct_detection *#
scene_struct_detection:
  enable: false # 是否开启该功能
  strategy: "PhantomScenceStructDetection"
  view_options:
    frontView_res: 0.4
    frontView_range_y: 80
    frontView_range_z: 25
    frontView_min_z: 0
    board_frontView_connectivity: 1

    verticalView_range_y: 80
    verticalView_range_x: 300
    verticalView_res: 0.4

```

```
board_verticalView_connectivity: 1

board_intensity_min: 240
tunnel_frontShadow_depth: 10

road_board_x_width_max: 0.4
road_board_density_onwidth_min: 50
road_oard_density_onpanel_min: 9
object_cloud_pointsnum_min: 9
```

F: 环境与硬件配置依赖说明

硬件平台与软件环境详细依赖参见下表：

| | | | | |
|------------|---------------------------------------|---------------------------------|---------------------|------------------|
| 硬件平台 | | | | |
| | 通用工控机平台 | | Jetson AGX Xavier | 基于Xavier-AD的域控平台 |
| 独立显卡 | >= GeForce GTX 1050 6.1 <= 计算能力 < 8.6 | >= GeForce RTX 3060 计算能力 == 8.6 | - | - |
| 内存 | >= 8GB | >= 8GB | == 16GB | >= 8GB |
| 软件环境 | | | | |
| Jetpack 版本 | - | - | == 4.5 | - |
| 操作系统 | Linux Ubuntu 16.04/18.04 | Linux Ubuntu 18.04 | QNX 7.0.0 | |
| 显卡驱动 | >= 430.34 | >= 460.56 | - | - |
| DriveOS | - | - | - | >= 5.1.15 |
| ROS | 16.04: melodic 18.04: bionic | bionic | - | |
| cuda | >= 10.0 | >= 11.1 | >= 10.2 (刷机时安装) | >= 10.2 |
| cudnn | >= 7.3.1 <= 7.6.5 | >= 8.0.4 <= 8.2.1 | >= 8.0.0 (刷机时安装) | >= 6.2.0 |
| 常用系统操作命令: | | | | |

| 硬件平台 | | | | |
|------|---|--|--|---|
| | 1. nvidia-smi -l查看gpu占用和利用率 2. 打开system monitor应用，监视sdk的进程 | | 1. 查看cpu占用率和gpu占用率 (1) jtop，如果没有此命令，使用 sudo -H pip install -U jetson-stats安装 (2) tegrastats 2. top查看进程状态 | 1. 查看cpu占用率和gpu占用率 (1) tegrastats (2) tegrastats > resources.txt 将记录保存下来 |
| 备注 | 1. 具体GPU计算能力参考 http://developer.nvidia.com/cuda-gpus 2. 目前只支持来自NVIDIA的硬件平台，其它非NVIDIA平台尚在开发中 3. 通用工控机平台必须安装显卡驱动，对于sdk3.1，cuda和cudnn非必须，sdk3.1以下必须安装 4. 在Ubuntu 18.04上运行sdk确保LD_LIBRARY_PATH的路径没有包含旧工程的库路径 | | | |

电脑推荐：

| | |
|------|---|
| 推荐配置 | 系统：Ubuntu1604/1804；处理器：推荐i5以上；显卡：推荐RTX2060\3060等；内存：16G+；硬盘：256G+（SSD） |
| 参考链接 | https://item.jd.com/72357212979.html?cu=true&utm_source=www.xiaoxiaodediyi.xyz&utm_medium=tuijiang&utm_campaign=t_1003545251_&utm_term=103a2e1044f94c8094df2d2c1aacc117#crumb-wrap https://item.jd.com/100016586396.html?cu=true&utm_source=www.xiaoxiaodediyi.xyz&utm_medium=tuijiang&utm_campaign=t_1003545251_&utm_term=50ea0290b90f4fed8c098ac2f9a8e339 https://item.jd.com/10032324160155.html?cu=true&utm_source=www.xiaoxiaodediyi.xyz&utm_medium=tuijiang&utm_campaign=t_1003545251_&utm_term=09cc4597d6d9442f80bb191e3c9dd561 |



robosense

www.robosense.ai

深圳市速腾聚创科技有限公司

service@robosense.cn / 0755-86325830 / 深圳市南山区留仙大道3370号南山智园崇文园区3栋10-11层



Q RoboSense LiDAR