Robosense Perception SDK User Guide (Version 1.6.6)

Robosense Perception SDK is a integrated software develop kit that packages the common perception algorithms for LIDAR used in robotic environmental perception, especially in autonomous driving, including localization, free space (traversable zone) detection, object detection, tracking and classification. It is convenient to develop your own application, such as ADAS, autonomous driving or robotic self navigation etc. based on our SDK.
Our perception SDK is in progress now, **currently included function list**:

- **Ground points estimation**
- **Object points segmentation**
- **Dynamic object Tracking**
- **Foreground object Classification**
- **Localization**
- **HD map ROI filter for drivable free space**

more functions and features like freespace or drivable zone detection will be involved in the future …

precautions

- The localization module (a deb file for installation) included in the sdk package is ***just for demostation use**, if you need localizaiton function, please contact our technical support for advanced support (like HD 3D pointcloud map building service).
- **Do not support Virtual machine installation !** and **Do not support Windows operation system now !** You have to install it in a local linux based System, like Ubuntu.
- **C++ based**
- **This SDK is not for RS-Box**, RS-Box is softerware-hardware integrated product and can be updated by our web tools, please do not try to configure this SDK into RS-Box by yourself !! If need, please contact our technical support.

1. Precaution & preparing

1) system & software dependencies

The SDK supports **Ubuntu 14.04 64bit** with **ROS indigo (desktop full version)** and **Ubuntu 16.04 64bit** with **ROS kinectic (desktop full version)**. Besides, the following components should also be installed:

- **libpcap1.8.1**
  Download the source file from http://www.tcpdump.org/ first, and install some prerequisites:

```
sudo apt-get install flex bison byacc -y
```

then, unzip the source file in somewhere (for example in ~/Downloads/), and:

```
cd ~/Downloads/libpcap
./configure
make
sudo make install
```

- **wireshark**
  It is commonly used in pcap file capturing and parsing, using:

```
sudo apt-get install wireshark
```

- **doxygen (optional)**
  In order to get doc for our API introduction in detail, you need to install doxygen to generate it automaticly:

```
sudo apt-get install doxygen
```

- **build-essential (optional)**
  In order to avoid possible compiler error, use:

```
sudo apt-get install build-essential
```

to get support from kinds of build or develop tools.

2) About Authority

You need to get a key to activate the SDK. We build our authority mechanism that binding your testing computer device and our SDK software, so you need to give your **mac address of the ethernetcard (Not wlan) on your testing computer equipment (Not the lidar device !)** and we will return you a specific key file named “release.bin”, you should put it in the /key folder. **You should backup your specific key in case that it gets lost or overrided when update SDK**.

2. Package introduction

Before using the SDK, you should have some knowledge about Linux and ROS and some cmake experiences, and are familiar with catkin workspace mechanism. Please download the following packages from `/Driver` , `/Lidar Mounting Calibration Tool` , `/SDK` and `/Localization install file` (if use localization) path respectively.

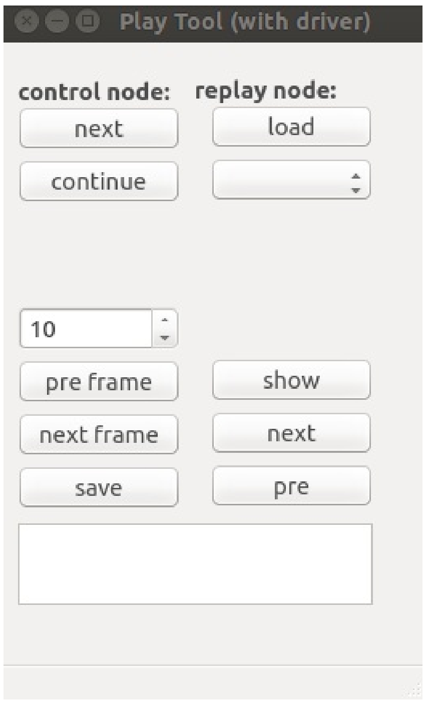
1) play_tool (lidar driver):

- **What is this tool for ?**

This is a control tool for playing offline pcap file. base on lidar drive package (i.e., rslidar driver pacage is included so that you need not another single driver package) by inserting the python format control script. It provides the control functions including continuous mode or frame by frame mode switching and some bug retrieving. Since driver package is included, it provides driver function (data receiving and parsing) as well.

- **How to use ?**

This is what it looks like:



In order to get the tool worked, user has to launch the related node (“**control_node**” and “**save_node**”) in launch file first. When it launched successful, user can change the play mode (“continue” or “frame by frame”) by click the `continue` or `next` button and press `save` to save the exception frame data in pcd format and replay it or deliver it to our engineer for debugging. This tool is designed as a auxiliary tool to help using our SDK. Note that it is developed upon **Qt module**, so you need install some prerequisites first with the follow instructions:

Install dependencies for Ubuntu 14.04 indigo

```
sudo apt-get install libnlopt-dev freeglut3-dev qtbase5-dev libqt5opengl5-dev libssh2-1-dev libarmadillo-dev libpcap-dev gksu libgl1-mesa-dev
 libglew-dev
```

Install dependencies for Ubuntu 16.04 kinetic

```
sudo apt-get install libnlopt-dev freeglut3-dev qtbase5-dev libqt5opengl5-dev libssh2-1-dev libarmadillo-dev libpcap-dev gksu libgl1-mesa-dev
 libglew-dev python-wxgtk3.0
```

Note that the `rslidar` driver package included in this tool is just for demostration and debug use, it may be not the latest version of our driver. When used in real application, please replace the `play_tool` with `rslidar` driver package in the U disk provided with the lidar device together.

2) auto_align_tool (lidar mounting calibration tool):

- **What is this tool for ?**

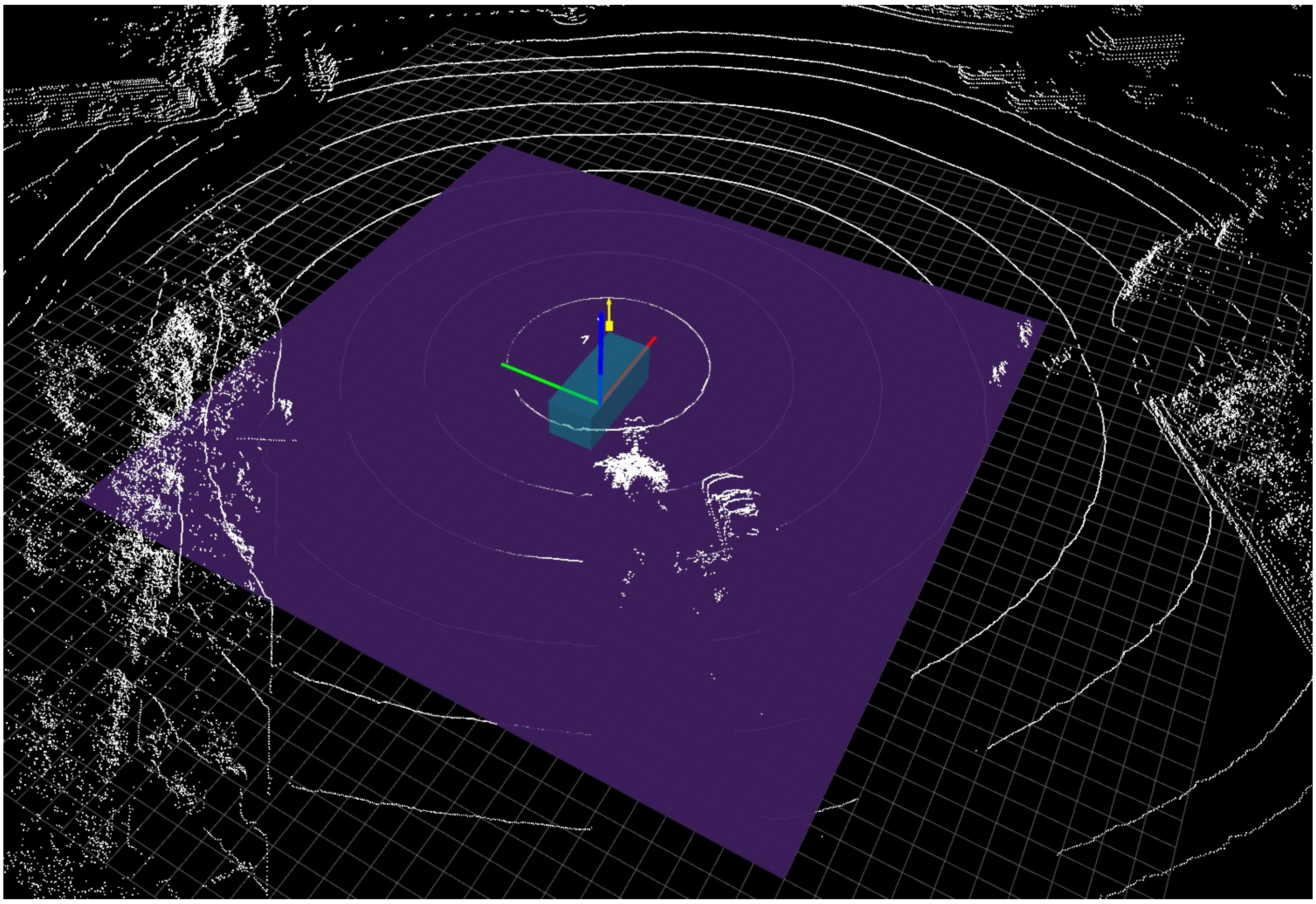
This is an auxiliary calibration tool for lidar mounting configuration parameters, i.e., x, y, z, roll, pith, yaw of lidar mounted relative to the mother vehicle. The main purpose of building this tool is to help users from the laborious work of measuring those parameters by hand. From pespective of algorithm, calibrating those parameters is a useful preprocesing which translate the various application situations to a relatively “normalized space” that can span the applicable range of the SDK. **In one word, this tool is designed to estimate the lidar mounting height and orientation (x-axis of the local lidar coordinates) and keep the ground points horizontal.**

- **How to use ?**
  Finding an open flat place (length or width > 15 m) , mounting the lidar to the mother vehicle and keep the lidar fixed tightly. Receive data and make the calibration in real time or collecting data in bag format and make calibration offline. The data source can be configured in launch file. **Notice that this tool should be used with company of lidar driver package (`play_tool` mentioned before)** to get original data parsed.

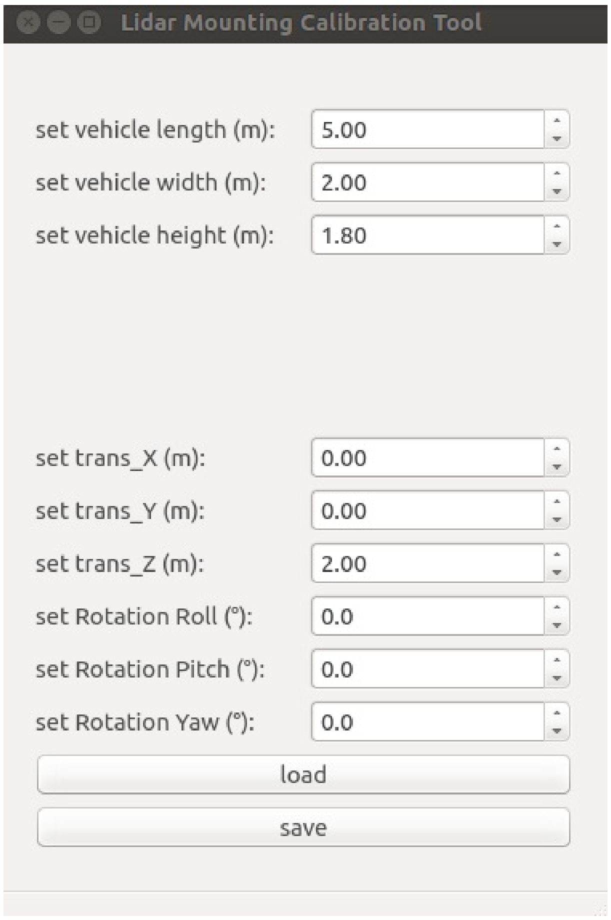
Launch the tool by:

```
cd Robosense_sdk
roslaunch auto_align_tool lidar_calibration.launch
```

There are orther launch files in /launch directory for users to chose. This is what it looks like in working:



The purple transparent plane is the virtul standard ground reference, the cyan box is the virtual vehicle, the axies (red is x, green is y, blue is z) is the vehicle reference coordinate system (see below for detail), and the yellow small box and arraw are the virtual lidar and its orientation. Users can calibrate the lidar mounting parameters through translation and rotation by the contral panel:



Lidar Mounting Calibration Tool

set vehicle length (m): 5.00
set vehicle width (m): 2.00
set vehicle height (m): 1.80

set trans_X (m): 0.00
set trans_Y (m): 0.00
set trans_Z (m): 2.00
set Rotation Roll (°): 0.0
set Rotation Pitch (°): 0.0
set Rotation Yaw (°): 0.0

Recommended way:
1) set the vehicle size first according to your own testing platform;
2) adjust the lidar height (`set trans_z`);
3) adjust the lidar position (`set trans_x` and `set trans_y`);
4) adjust the yaw angle (`set Rotation Yaw`);
5) adjust the pointcloud to keep ground points hided in the purple plane through adjusting the roll and pitch angle (`set Rotation Roll` and `set Rotation Pitch`).

When calibration done, click `save` button to save the result and then the following perception procedure can query the calibrated mounting parameters automatically through the path configuration in launch file. If user wants to recalibrate from the last result, click the `load` button to recover the last result and going on.

3) SDK:

There are two parts included: “/**Environment**” and “/**Core**”.

a) Core

The core of the SDK is composed of **header files** and **library files** which are the minimum required for third-part development, named with **robo_perception_sdk_xxx** (perception sdk with version and platform specification) in the “/SDK/Core” path. It is convenient to use the SDK just by including the header files and libraries to your project, just like using OpenGL. Note that the SDK libraries **only depend on OpenCV, PCL, Eigen and Boost**, **NOT depend on ROS**, which is just used for convinience to debug and show results. We provide an example “**Using SDK without ROS example**” in the root path of SDK repository to show how to use the SDK without ROS.

b) Environment

In order to test the sdk core file, we provide an testing environment named “**robosense_sdk**” in the “/SDK/Environment” path. **We store the core file and environment separately so that you can update just by replacing “/include” and “/lib” files minimum changes**. While, this is only suitble for the case with small changes between versions, when big changes happen, users have to follow the version updating specificaiton and redownload and use the latest environment file.

Archieve Tree:

- /car_rviz_model

  > contains the model file for drawing the virtual car in rviz;

- /cfg

  > contains the rviz configuration file for visualization;

- /data

  > contains: 1) the recorded offline lidar data for offline test; 2) map and ROI filter-mask data if used with localization, the path should be configured in `"perception_args.xml"` (see below for detail).

- /demo

  > **contains two examples showing how to use our SDK in detail**, “test_without_map.cpp” is a case without HD map version, and “test_with_map.cpp” is the total version with HD map and localization;

- /tools

  > contains the necessary src file for visualization and socket communication module for translating perception results in different coordinates;

- /doc

  > contains the API description in detail for our SDK in doxygen format (html files, you need to open in a browser such as chrome), you have to run doxygen to generate it automatically;

- /include (**should copy from “/SDK/Core/Ubuntu_xxxx/”**)

> contains all the “.h” files, you should include it to your own project;

- /key

> contains the key file to activate and authorize the SDK;

- /launch

> contains the ros launch files which is convenient for running all needed pacakages in user-defined way all together, there are several launch files, showing usage in different situations like online/offline and use map or not.

- /lib (**should copy from “/SDK/Core/Ubuntu_xxxx/”**)

> contains all the “.so” files, you should link it to your own project;

- /model

> contains the model file for machine-learning based algortithms in the SDK, like classification, please download it and place them in this folder or link to this folder, of course you can reconfig the path by modify the launch file below the following;

- /usr_args

> contains all the configurations needed for user with “xml” file, default name is `"perception_args.xml"` ;

- others

> we also provide a “CMakeLists.txt” file to help you simplify the project configuration, if you are familar with cmake tool.

Users can get detail infomation about APIs by doc file. In order to reduce the package size as much as possible, we provide a “Doxyfile” file in /test folder, users can generate doc file automatically by doxygen tool:

```
cd ~/Robosense_sdk/src/test
doxygen Doxyfile
```

4) Localization module package

This model is provided by a deb installation file, for example, for Ubuntu 16.04:

```
sudo dpkg -i ros-kinetic-robosense-localization_0.1.2-0xenial_amd64.deb
```

3. Quick-start

1) Compile the SDK

Download the following packages and unzip them:

- “**play_tool**” from “/Drivier”;
- “**auto_align_tool**” from “/Lidar Mounting Calibration Tool”;
- “**robo_perception_sdk_xxx**” from “SDK/Core”;
- “**robosense_sdk**” from “SDK/Environment”.

Build a workspace folder named like “/RobosenseSDK”in your home directory, for example:

```
cd ~
mkdir RobosenseSDK
cd RobosenseSDK
mkdir src
```

Then put the “**play_tool**”, “**auto_align_tool**” and “**robosense_sdk**” package into the /src directory, and extract the sub folders “/include” and “/lib” from “**robo_perception_sdk_xxx**” and put them into the “**robosense_sdk**” package. The result directory tree should be like this:

```
|___RobosenseSDK (ros workspace)
    |___src
        |___play_tool
        |___auto_align_tool
        |___robosense_sdk
            |___include
            |___lib
            |___...
```

Get rid of authority problem:

```
cd ~/RobosenseSDK/src
sudo chmod -R 777 *
```

Next try:

```
cd ~/RobosenseSDK
catkin_make
```

You may encounter problems like “**can not find libpcap.so.1**”, try:

```
cd /etc
sudo gedit ld.so.conf
```

add

```
/usr/local/lib
/usr/lib
```

to the last line of the file, and save. Then reload the configure file:

```
ldconfig
```

You may also encounter OpenCV problem (often occured under Ubuntu kinetic) which is usually caused by the library name compatibility, you can solve the problem by make solft link through a script (like opencv_softlink.sh). For example, with Ubuntu16.04 and ROS kinectic, if the debug info says “libopencv_xxx.so.3.2 not found! “, try:

```
sudo ln -s /opt/ros/kinetic/lib/libopencv_core3.so /opt/ros/kinetic/lib/libopencv_core3.so.3.2
sudo ln -s /opt/ros/kinetic/lib/libopencv_highgui3.so /opt/ros/kinetic/lib/libopencv_highgui3.so.3.2
sudo ln -s /opt/ros/kinetic/lib/libopencv_ml3.so /opt/ros/kinetic/lib/libopencv_ml3.so.3.2
sudo ln -s /opt/ros/kinetic/lib/libopencv_video3.so /opt/ros/kinetic/lib/libopencv_video3.so.3.2
sudo ln -s /opt/ros/kinetic/lib/libopencv_imgproc3.so /opt/ros/kinetic/lib/libopencv_imgproc3.so.3.2
sudo ln -s /opt/ros/kinetic/lib/libopencv_imgcodecs3.so /opt/ros/kinetic/lib/libopencv_imgcodecs3.so.3.2
```

Try “catkin_make” again.
After compiled 100% sucessfully, you shoud have to prepare some data needed to run the SDK.

2) Preparing data

- **Get model file**
  Since mechine learning and deep learning techniques are involved in our SDK, some pretrained model files are needed. Considering the big file size, we have uploaded them in Baidu Netdisk with SDK software packages together: **links: https://pan.baidu.com/s/1cKxcPw, extract code: pwnz**. You should download them to your local

disk. If you need Map and localization function, you can also get the localization installation files from here.
Note that our SDK and other related data will be updated using this way, you should checkout some times in this netdisk address.

3) Configurate the launch file

Launch file is an offline configuration file that used in ROS to ensemble the nodes together to accomplish some robotic applications. It depends on the topic communication mechanism, providing a covenient way to launch all the nodes needed at once. For the more, some global parameters can also be transported from launch file, so you need not to change the source code all the time which leaves out the re-compiling procedure. You do have to take some time to get into it.

We provide two ready-to-use launch, for example, “16_offline_without_map.launch” (“< !– ” and “–>” means annotation, like “//” in C++ code):

```
<?xml version="1.0"?>

<launch>

<!--parse and publish point cloud-->
  <node name="rslidar_node" pkg="rslidar_driver" type="rslidar_node" output="screen" >
    <param name="model" value="RS16"/> # lidar device type setting
    <param name="pcap" value="$(find robosense_sdk)/data/2017-09-28-10-41-26/lidar.pcap" /> # offline pcap path
    <!--param name="device_ip" value="192.168.1.200"/--> # device ip setting if multiple lidar is used, can not omit
    <param name="port" value="6699" /> # device receive port
  </node>

  <node  name="cloud_node" pkg="rslidar_pointcloud" type="cloud_node" output="screen" >
    <param name="model" value="RS16"/> # lidar device type setting
    <param name="hori_angel_begin" value="0"/> # lidar data scan range, clockwise
    <param name="hori_angel_end" value="0"/> # lidar data scan range, clockwise
    # calibration file path
    <param name="curves_path" value="$(find rslidar_pointcloud)/data/rs_lidar_16/curves.csv" />
    <param name="angle_path" value="$(find rslidar_pointcloud)/data/rs_lidar_16/angle.csv" />
    <param name="channel_path" value="$(find rslidar_pointcloud)/data/rs_lidar_16/ChannelNum.csv" />
  </node>

<!--main testing node--> # main node calling sdk module, source code is in the test/demo folder
  <node pkg="robosense_sdk" type="test_without_map"  name="test_without_map"  output="screen" >
    # pcap file path used for data verification when worked offline with pcap data.
    <param name="pcap" value="$(find robosense_sdk)/data/2017-09-28-10-41-26/lidar.pcap" />
    <param name="key_path" value="$(find robosense_sdk)/key" /> # key "release.bin" file path
    <param name="ethernet_name" value="eth0" /> # logic name of ethernetcard, key authority need
    <param name="port" value="6699" />  # device receive port, should be same as rslidar_node setting
    # "perception_args.xml" path
    <param name="args_path" value="$(find robosense_sdk)/usr_args/perception_args.xml" />
    # lidar configuration file path, should be generated automatically by the auto_align_tool
    <param name="lidar_config_path" value="$(find auto_align_tool)/save_config/align.txt" />
    # set receive ip and port for communication module if you want deliver the perception results out
    <param name="receiv_ip" value="192.168.1.99" />
    <param name="receiv_port" value="60000" />
  </node>

<!--car module for ui display-->
  <group>
    <arg name="gui" default="False" />
    <param name="use_gui" value="$(arg gui)"/>
    <param name="robot_description" textfile="$(find robosense_sdk)/car_rviz_model/no_map/default.urdf" />
    <node pkg="joint_state_publisher" type="joint_state_publisher" name="joint_state_publisher"  />
    <node pkg="robot_state_publisher" type="state_publisher" name="robot_state_publisher"  />
  </group>

<!--rviz show-->
  <node pkg="rviz" type="rviz" name="rviz"  args="-d $(find robosense_sdk)/cfg/robosense_no_map_test.rviz"/>

</launch>
```

Besides the above launch file, there are two other configuration files named “**perception_args.xml**” and “**align.txt**”:

i) The `"align.txt"` is the calibration file for lidar mounting generated by the `auto_align_tool` , default path is `/auto_align_tool/save_config` , looks like this:

```
vehicle_size: 4.5 2 2
lidar_config: 0 0 1.9 0.007 -0.009 0
```

“vehicle size” is length, width and height of the vehicle, and “lidar_config” is x, y, z, roll, pitch, yaw for the lidar mounting relative to vehicle coordinate.

ii) The `"perception_args.xml"` is a user-specific configuration file that is used to tune some settings of the SDK with specific application, which is more complicated and detailed by a new chapter, see below.

4) Configurate the user configuration xml file

we provide a xml configuration file named `"perception_args.xml"` , in order to help users to customize SDK settings to meet specific need. The following is a example with comments (right or above of the cofigure line), **if no comments provided, means users have not to care and tune it**, we list it out just for senior engineer and ourself to debug, it will be removed in the future when sdk gets mature enough.

```
<?xml version="1.0" encoding="utf-8"?>
<RoboUsrConfig>
    <lidar_type> # lidar type configuration, including name, horizonal and vertical angle resolution, with degree unit, and frequency
        <name>RS16</name>
        <hori_resolution>0.18</hori_resolution>
        <vert_resolution>2</vert_resolution>
        <frequency>10</frequency>
    </lidar_type>
    <use_tracking>true</use_tracking> # enable tracking or not, can save some time if set false
        <use_classification>true</use_classification> #enable classification or not,  can save some time if set false
    <obj_limit> # geometry restrictions to filter out static building-like objects
        <obj_max_height>5</obj_max_height>
        <obj_size_height>4.3</obj_size_height>
        <obj_size_length>20</obj_size_length>
        <obj_size_width>5</obj_size_width>
        <obj_length_width_ratio>10</obj_length_width_ratio>
        <obj_max_area>50</obj_max_area>
    </obj_limit>
    <is_pre_merge>true</is_pre_merge> # pre-merge operation
    <is_merge>true</is_merge> # merge sub-segments according semantic info
    <is_geo_filter>true</is_geo_filter> # geometry filter switch,  should be paired of obj_limit to use
    # automatic adjust pointcloud, to enhance robustness against bumpiness
    <is_auto_align>true</is_auto_align>
    # output the perception results in vehicle or lidar coordinate, if false, the output results will be in lidar coordinate
    <is_recover_to_vehicle>false</is_recover_to_vehicle>
    <use_data_enhance>true</use_data_enhance> # use data enhance or not in preprocessing
    <ground_range> # range setting for ground estimation
        <xmin>-50</xmin>
        <xmax>50</xmax>
        <ymin>-20</ymin>
        <ymax>20</ymax>
    </ground_range>
    <detect_range> # range setting for detection
        <xmin>-40</xmin>
        <xmax>80</xmax>
        <ymin>-20</ymin>
        <ymax>20</ymax>
    </detect_range>
    <ignore_range> # range setting for ignore zone, used for near zone removal
        <xmin>-2.3</xmin>
        <xmax>2.3</xmax>
        <ymin>-1.2</ymin>
        <ymax>1.2</ymax>
    </ignore_range>
    <min_pts_num>3</min_pts_num> # sparse filter, object with points number less than it will be discarded
    # if use sequential frames to evaluate the robustness for a tracker, will cost more time
    <track_seq_evaluate>false</track_seq_evaluate>
    <track_global_refine>true</track_global_refine> # if use global self velocity information to fine-tune the detection
    <track_range>3</track_range> # association searching range for a tracker and current objects
    <predict_frame_num>5</predict_frame_num> # tracker will predict for 5 frames when lost or get shaded (invisible)
    <class_mode>2</class_mode> # object classification method, we provide 0~2, default is 2
    <with_enhance>true</with_enhance> # object classification, with enhance version
    # model file path for object classification
    <classify0_model_path>/home/wangbin/data/model/classification/classification_model_0.xml</classify0_model_path>
    <classify1_model_path>/home/wangbin/data/model/classification/classification_model_1.txt</classify1_model_path>
    <use_roi>false</use_roi> # use roi filter-map or not, should be coupled with map and localization module
    <roi_filter_map>/home/wangbin/data/GYY/feature_map</roi_filter_map> # roi filter map path, if use_roi set true, must provide
</RoboUsrConfig>
```

Users should have to take some time to get familiar with the configuration, and tune it carefully to get the best result.

5) Configurate the distance measuring calibration file

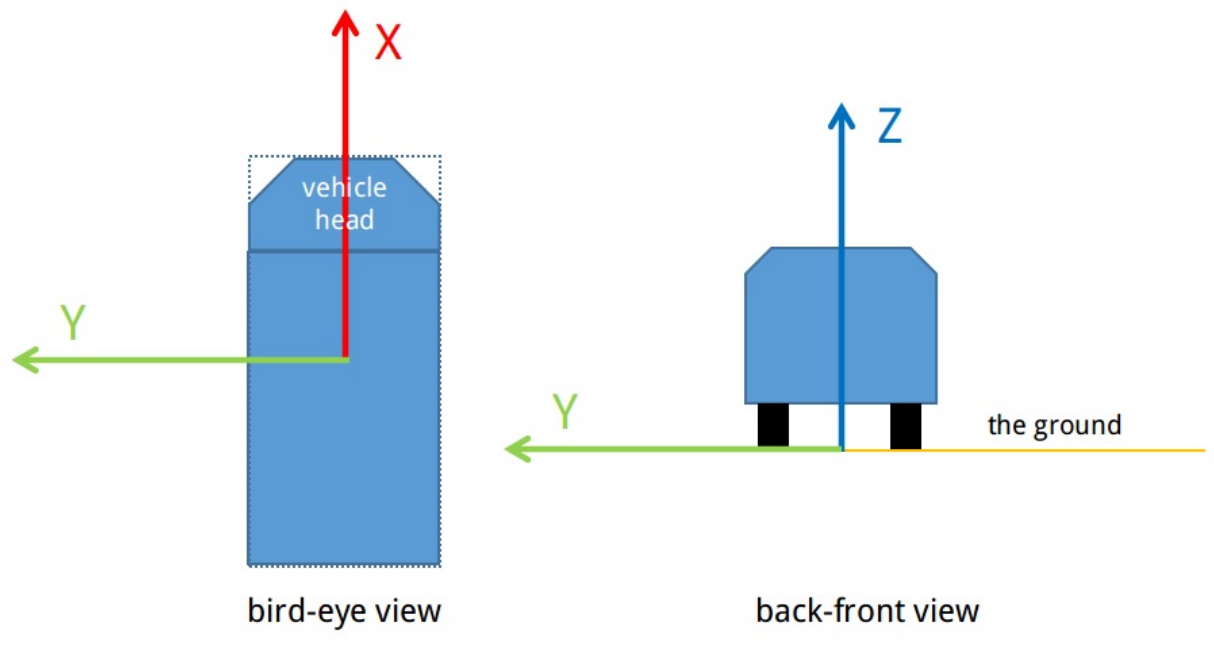
Be Cautious! Every lidar device has a specific calibration file for distance measuring. If not configured right, the output lidar pointcloud will seems distorted with abnormal curves. The configure file are stored in the U disk provided together with the lidar device, named with “**angle.csv**”, “**curves.csv**” and “**ChannelNum.csv**”. Please preserve the configuration file well with every lidar device correspondence. Before using the ldiar device, please replace the right calibration file with the default in “**/play_tool (if use play_tool)/rslidar/rslidar_pointcloud/data**”, of course, user can also configure the path different in launch file.

4. SDK coordinate setings

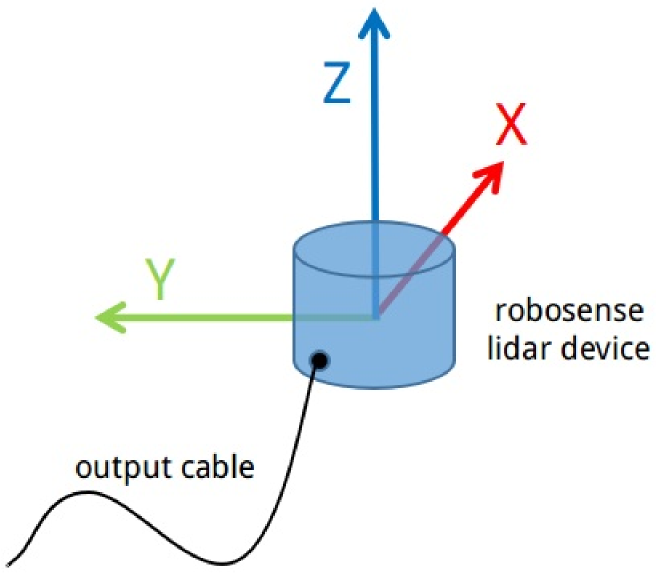
Before running the demo, knowing the coordinates setting is necessary. Our SDK is designed following some coordinate settings, including lidar coordinate, vehicle coordinate, global coordinate, **all the coordinate is right-handed**.

- **Standard Reference vehicle Coordinate**

X points the right ahead of the vehicle, Y points left and Z points sky, origin is set to the center of the orthographic projection of the vehicle to the ground, that is the z coordinate of the origin is on the ground. **If your vehicle coodinate is defined different, you need to translate yourself, that is you have to translate the perception results to your own vehicle coordinate by extra transformation.**



- **lidar Coordinate**
  X points the opposite direction of the output cable, or the output cable points the negtive direction of X-axis, if you mounting your lidar toward the front, then Y points the left, and z point to the sky, like the vehicle coordinate.



- **global Coordinate**
  This depends the settings of user, should be right-handed.

As mentioned before in chapter 2.2, we provide a calibration tool to calibrate the transformation between lidar coordinate and vehicle coordinate, Don't worry if you do not know the mathematical knowledge about the transformation matrix.

5. Run the demo

After tedious configuration step by step, we finally reach the excitting destination, Run!

```
cd ~/RobosenseSDK
source devel/setup.bash
```

if you use zsh:

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

and last:

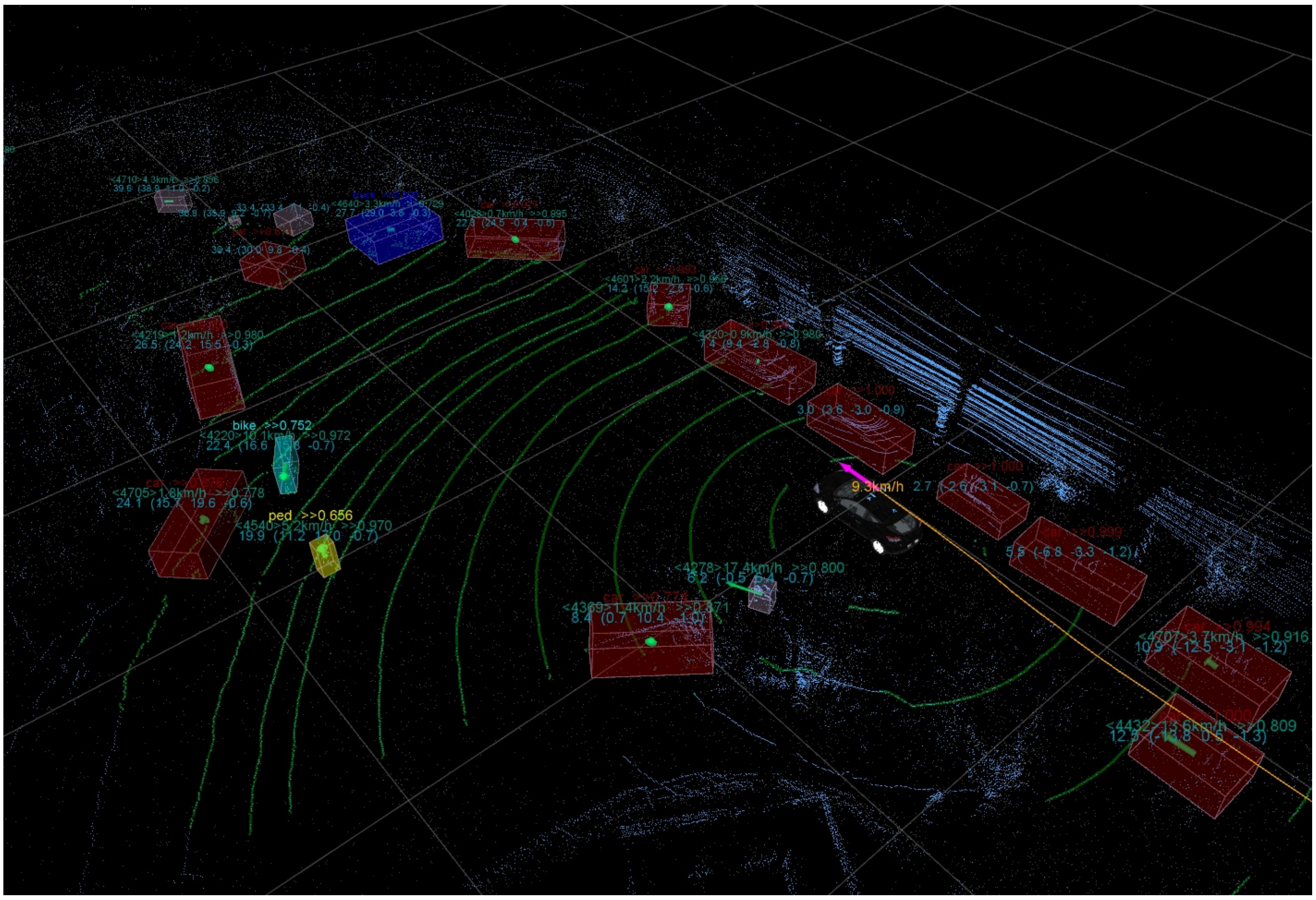
```
roslaunch robosense_sdk 16_offline_without_map.launch
```

If the key passes authority, you will see the following outprint in the prompt window:

```
"key verified."
"go into online/offline branch“
”data verified.“
```

If not, that means the key fails, please recheck the “**key path**” and “**ethernet_name**” or the **mac address** and make sure they are right. If you can not solve the problem after trying all the effert, please contact us.

If sucess, you will see the following perception results:



- Grid (gray cell)

> this a distance reference grid to aid you to estimate the distance measuring, by default, every cell’s edge is 10 meters long;

- Ground Points (green points in circle in ground)

> the percepted ground points, or likely, the drivable zone, note that they are range-filtered by “**ground_range**” in `"perception_args.xml"` ;

- Obstacle Points (sky blue points)

> the percepted non-ground points, or likely, the obstacle points, note that they are range-filtered by “**detect_range**” in `"perception_args.xml"` ;

- PerceptInfo (colorful subtransparent box, text, green arrow etc.)

> this is the perception results, including boundingbox, class type and confidence, tracking velocity and tracking probability etc. You can checkout in “Namespaces” under the topic. Note that they are not including all the perception information, such as the distance to self vehicle, classify confidence, tracking life etc. You can refer the display functions in `/robosense_sdk/tools/robo_draw_rviz.cpp for detail and learn how to modify the display code and show other perception information;

- Original Points (white small points)

> this is the original pointcloud received from the driver (`rslidar` package in `play_tool`), can be used to evaluate the original lidar measuring performance;

- selfCar Info

> self velocity in global and yaw angle in purple arrow;

- Car Model

> the virtual car display model for demostration

Of course, you can modify the display and write your own launch file and run it. Note that this is a example for offine usage, **if used with localization, you have to confiure more things, please refer the `"online_test_with_map"` case in `/test demo & data/` path in netdisk for detail**.

6. Run the demo online with lidar device in realtime

If you want to test online with lidar device realtime, please check the communication between lidar device and testing computer.

1. Configure PC IP

By default, the RSLIDAR is configured to **192.168.1.200** as its device IP and **192.168.1.102** as PC IP that it would communicate. The default port is 6677 as RSLIDAR tele port number, while 6699 as the data port on the PC.
So you need configure your PC IP as a static one **192.168.1.102**.

2. Run and view realtime data

We have provide an exmaple launch file under rslidar_pointclodu/launch, we can run the launch file to view the point cloud data.
(1). Open a new terminal and run:

```
cd ~/Robosense_sdk
source devel/setup.bash
roslaunch rslidar_pointcloud rs_lidar_16.launch
```

(2). Open a new terminal and run:

```
rviz
```

Set the Fixed Frame to “**rslidar**”.
Add a Pointcloud2 type and set the topic to “**rslidar_points**”.

If pointcloud data is showing correctly, then try to run the online test (“online_without_map.launch”).

7. How to integrate the SDK into your own project

1) prepare dependences

The SDK is provided by libraries way, so it is the same as the common usages of libraries: including the header files and linking the corresponding libraries into your own project. **Note that the SDK do not depend on ROS**, the demo just use ROS to simplify the multi-threads coding and visualization. Before it runs, you have to make sure that the underlaying libraries are installed correctly.

- **For Ubuntu 14.04, the depending libraries are:**
  PCL 1.7.1
  OpenCV 2.4.8 (2.4.x should be ok)
  Boost 1.5.4
  Eigen 3.2.0
  Libpcap 1.8.1

- **For Ubuntu 16.04, the depending libraries are:**
  PCL 1.7.2
  OpenCV 3.2.0 dev (3.x should be ok)
  Boost 1.5.8
  Eigen 3.2.92
  Libpcap 1.8.1

Other versions for the above metioned libraries are not tested, compatibility are not confirmed, you may have a try. **You may refer the usage in `/robosense_sdk/demo` , two examples are provided: with map or without map (localization)**. An example is also provided for **non-ROS** situation, checkout in our sdk repository (baidu netdisk for now), named “/Using SDK without ROS example”

2) Data IO structure

Input

- standard pcl pointcloud type:

```
pcl::PointCloud<pcl::PointXYZI> pointcloud;
```

Note that the pointcloud should be sorted as a 2D range image in polar coordinate, i.e., the height of the pointcloud should not be 1, for example, for RS-16, the height of the input pointcloud should be 16. This is for performance reason. Our driver outputs sorted pointcloud. If you want to get rid of ROS and get a sorted pointcloud, please refer the function “unpack” in `/play_tool/rslidar/rslidar_pointcloud/src/rawdata.cc` . And we will provide a non-ROS driver soon, please checkout now and then.

- vehicle velocity

```
float velocity;
```

The unit of input velocity is m/s.

- velocity yaw angle:

```
float yaw;
```

The unit of input yaw angle is radian.

- localization: GPS/IMU and OBD output, please see the detail in localization module, if used with localization module.

Output

we provide a demo that shows how to use the SDK, the perception result is output as a percpetion list, containing all the percepted objects in one frame, for every percepted object, the output information contains:

- location

(x, y, z), object location in lidar coordinate or vehicle coordinate (up to user configure), with unit meters;

- direction

(x, y, z, in vector format), object direction in lidar coordinate or vehicle coordinate (up to user configure), is mainly coincident with velocity direction for dynamic object, and keep parallel to the longest edge direction of bounding box for static object;

- yaw

yaw angle of object boundingbox direction, can be seem as another description for object `direction` , it is equivalent to object `direction` when we simplify the work settings in x-y plane, i.e., `direction.x = cos(yaw)` , `direction.y = sin(yaw)` , with unit rad;

- size

(x: length, y: width, z: height), notice that the `size` is measured under the local object coordinate system where the x-axis points to the longest edge direction, i.e., `size.x >= size.y` , with unit meters;

- nearest_point

the nearest point of object to self vehicle in lidar coordinate or vehicle coordinate (up to user configure), used to estimate collision, with unit meters;

- tracker_id

the id of the tracker which the object is belong to;

- track_prob

possibility for the object belong to a tracker, 0 ~ 1, the greater, the better;

- velocity

(x: Vx, y: Vy), object velocity in lidar relative coordinate, with unit m/s;

- acceleration

(x: Ax, y: Ay), object acceleration in lidar relative coordinate, with unit m/s^2;

- velocity_abs

(x: Vx, y: Vy), object velocity in global coordinate, with unit m/s;

- acceleration_abs

(x: Ax, y: Ay), object acceleration in global coordinate, with unit m/s^2;

- angle_velocity

angle velocity, as it is not a vector, we do not make difference between local and blobal coordinate, with unit rad/s;

- life

total tracking time for an object, including visible and invisible frames, with unit seconds;

- visible_life

total tracking time for an object, only including visible frames, with unit seconds;

- label

0 (unknow), 1 (pedestrain), 2 (bike), 3 (car), 4 (truck/bus);

- label_confidence

confidence of classification, 0 ~ 1, the greater, the better.

all the above infomation are packed with a C++ struct named `"PerceptOutput"` .

Transfer perception results between machines

Users may not need to re-develop on the SDK, on the contrary, just need a ready-to-use solution. We provide a communication demo based on socket in `/robosense_sdk/tools/communication` folder, users can make a reference to write their own code.

8. Contact us

If you have specific needs or you want feedback bugs or you just want feed us your specific data, please contact us:

```
email: bwang@robosense.cn
```

Note that the feedback data should be in ".pcap" or rosbag format, rosbag is highly recommended. Note that in order to shrink the size of bag file, please **do not record "/rslidar_points", record "rslidar_packets" instead**. And what's more, **do not forget to feedback the lidar calibration file (3 csv file) together** since we need it to parse the original packets data.