robosense
速腾聚创

RS-Box

—— 用户手册

www.robosense.ai

目录

非常感谢您购买了 RS-Box 产品，请您认真阅读并祝您使用愉快！

1 安全提示

为避免损坏设备及违反保修条款，请勿打开 RS-Box。

- 阅读说明 - 请在使用本产品之前，认真阅读所有安全和操作说明。
- 遵循说明 - 请遵循所有操作和使用说明。
- 保留说明 - 请保留所有安全和操作说明，以备将来参考。
- 注意警告 - 请遵守产品和操作说明中的所有警告，以免发生意外。
- 产品维修 - 在操作中描述的内容之外，请不要尝试维修产品。如需维修，请及时联系本公司。

2 产品简介

RS-Box 是速腾聚创推出即插即用的激光雷达算法硬件平台。RS-Box 内置 RS-LiDAR-Algorithms 感知算法，用户无需进行繁琐的配置，配合 RS-LiDAR，即可快速、无缝地将激光雷达感知模块嵌入到自己的无人驾驶方案中，真正实现“一键获得自动驾驶激光雷达环境感知能力”。

RS-Box 由嵌入式硬件平台、独立操作系统和 RS-LiDAR-Algorithms 感知算法三大部分组成。RS-Box 独立运行 RS-LiDAR-Algorithms 点云感知算法，为自动驾驶提供高精度实时定位、障碍物识别与分类、动态物体跟踪等算法功能。

考虑到自动驾驶对实时性、低功耗、稳定性的要求，我们抛弃了目前以工控机为主的研发平台，选择低功耗的嵌入式硬件为基础平台，然后在算法软件层面上做针对性的优化，兼顾高性能和低功耗的同时，保证算法感知功能可以实时并稳定地工作。

为了方便调试和适配更多的硬件环境，RS-Box 提供了丰富的接口：LAN 网线接口，USB 3.0，micro USB 2.0，HDMI 等，以及标准 SD 卡槽，M.2 E，SATA 等存储扩展接口。这些接口可以满足用户对数据传输、多设备的连接、视频输出、存储扩展等的需求。

另外，RS-Box 提供了多种传感器的输入数据协议，用户可以根据数据协议连接 GPS、IMU、里程计等模块，在线工作时，RS-Box 以 10Hz 的频率向用户接收端实时发送定位感知列表数据流。

RS-Box 封装了 RS-LiDAR 数据的接收、解析、处理等环节，通过和 RS-LiDAR 设备连接，用户无需关心数据处理流程。无论是 Linux、Windows 或者其他系统设备，只需要连接 RS-Box，就可以实时调取感知结果，非常容易快速嵌入已有的无人驾驶系统中。

RS-Box 是基于速腾聚创 RS-LiDAR-16 和 RS-LiDAR-32 开发的，支持 RS-LiDAR 设备即插即用。RS-Box 也可以通过定制，实现与其他厂家激光雷达的组合使用。

3 电气接口

3.1 设备电源

设备供电要求规格为 19V-4.74A， 90WMAX，请使用出厂包装中的电源适配器。

3.2 电气接口说明

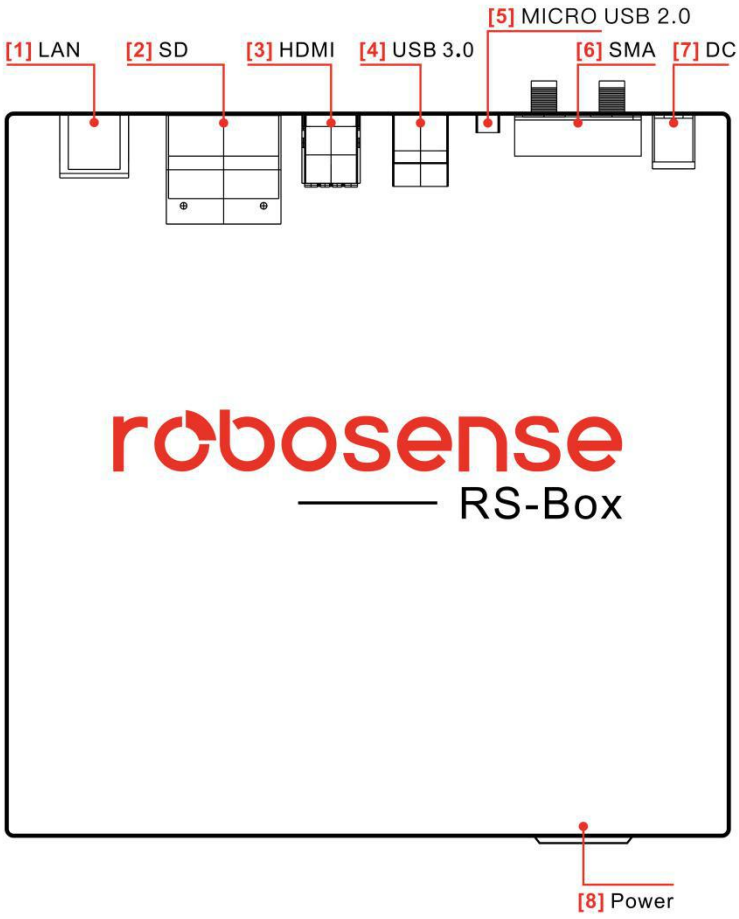


图 1 RS-Box 电气接口

其中：

[1] LAN：通过网线与交换机相连，而交换机同时也连接了 RS-LiDAR 和用户接收端，通过 LAN 口实现 RS-LiDAR 点云数据的接收，以及 RS-LiDAR-Algorithm 定位信息和感知结果发送输出；

[2] USB 3.0：连接 GPS、IMU、里程计等模块使用；

[3] DC：直流电源输入，请使用出厂电源适配器；

[4] Power: RS-Box 操作系统启动按钮；

其余电气接口目前不做使用。

4 用户使用步骤

1. 通过交换机连接激光雷达 RS-LiDAR，RS-Box 和用户接收端，并上电，按 RS-Box 中的“Power On”按键，启动 RS-Box。

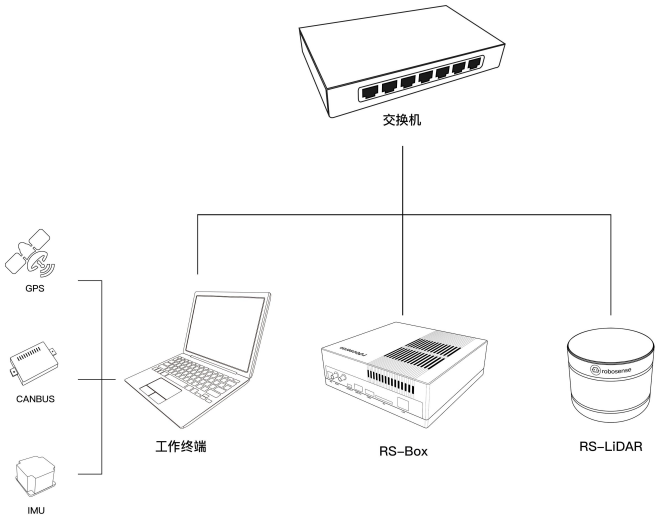


图 2 RS-Box 连接图

2. 通过 USB 扩展口连接 GPS、IMU、里程计等其他传感器。同时确保用户接收端和 RS-Box 的网络属于同一网段，RS-Box 默认的 IP 为 192.168.1.102。
3. 用户通过速腾聚创提供解析接口，轻松地把 RS-Box 输出的定位信息和感知结果应用到用户的系统中(参见下节**功能输出指南**)。

5 传感器数据输入

5.1 GPS

（结构体和发送示例代码，下同）

```
struct gps_msg
{
    double timestamp; //秒为单位
    float longitude;
    float latitude;
    bool empty; //false
};
```

//示例代码

```
#include "communication.h"
#include "ros/ros.h"
#include <sensor_msgs/NavSatFix.h>

Robosense::Communication *communicator;

void callback(const sensor_msgs::NavSatFixConstPtr gps)
{
    Robosense::gps_msg tmp_msg;
    tmp_msg.timestamp = gps->header.stamp.toSec();
    tmp_msg.longitude = gps->longitude;
    tmp_msg.latitude = gps->latitude;
    tmp_msg.empty = false;
    communicator->sendGpsMsg(tmp_msg);//  发送 gps 消息
}

int main(int argc,char **argv)
{
    ros::init(argc,argv,"gps_msg");
    ros::NodeHandle nh;
    communicator = new Robosense::Communication(); //初始化 socket 发送对象
```

```
    communicator->setServerIP("192.168.1.102");// 设置 ip，RS-Box 端 ip 为
192.168.1.102
    communicator->setPort(60004);//设置端口号，gps 为 60004
    communicator->initSender();//初始化发送对象
    ros::Subscriber sub_gps = nh.subscribe("/fix",1,callback); //订阅 gps
    ros::spin();
    return 0;
}
```

5.2 IMU

```
struct imu_msg
{
    double timestamp;
    float acc_x;      //acceleration_x
    float acc_y;     //acceleration_y
    float acc_z;     //acceleration_z
    float orientation_w;
    float orientation_x;
    float orientation_y;
    float orientation_z;
    bool empty;
};
```

//发送 IMU 示例代码

```
#include "communication.h"
#include "ros/ros.h"
#include <sensor_msgs/Imu.h>
#include <cstdlib>
Robosense::Communication *communicator;
void callback(const sensor_msgs::ImuConstPtr imu)
{
    Robosense::imu_msg tmp_msg;
    tmp_msg.timestamp = imu->header.stamp.toSec();
    tmp_msg.acc_x = imu->linear_acceleration.x;
    tmp_msg.acc_y = imu->linear_acceleration.y;
```

```
    tmp_msg.acc_z = imu->linear_acceleration.z;
    srand(time(0));
    tmp_msg.orientation_x = imu->orientation.x;
    tmp_msg.orientation_y = imu->orientation.y;
    tmp_msg.orientation_z = imu->orientation.z;
    tmp_msg.orientation_w = imu->orientation.w;
    tmp_msg.empty = false;

    communicator->sendImuMsg(tmp_msg);// 发送 imu 消息
}

int main(int argc,char **argv)
{
    ros::init(argc,argv,"imu_msg");

    ros::NodeHandle nh; //初始化 socket 发送对象

    communicator = new Robosense::Communication();

    communicator->setServerIP("192.168.1.102"); //设置 ip，RS-Box 端 ip 为
192.168.1.102

    communicator->setPort(60003); //设置端口号，imu 为 60003，gps 为 60004，canbus
车速为 60002

    communicator->initSender(); //初始化发送对象

    ros::Subscriber sub_imu = nh.subscribe("/nav440/nav440",1,callback); //订阅
imu 话题

    ros::spin();
    return 0;
}
```

5.3 里程计

```
struct obd_msg
{
    std::string head;
    double timestamp;

    float speed; //米每秒
```

```
    float steering;
    bool empty;
};
```

//发送示例代码

```
#include "communication.h"
#include "VehicleInfo.h"
#include "ros/ros.h"
#include <nav_msgs/Odometry.h>

Robosense::Communication *communicator;
ros::Subscriber sub_vel;

void callback(const nav_msgs::OdometryConstPtr &vehicle_info)
{
    Robosense::obd_msg tmp_msg;
    tmp_msg.timestamp = vehicle_info->header.stamp.toSec();
    tmp_msg.speed = vehicle_info->twist.twist.linear.x;
    tmp_msg.empty = false;

    communicator->sendObdMsg(tmp_msg);// 发送 obd 消息

}

int main(int argc,char **argv)
{
    ros::init(argc,argv,"velmsg");
    ros::NodeHandle nh;

    communicator = new Robosense::Communication(); //初始化 socket 发送对象

    communicator->setServerIP("192.168.1.102");// 设置 ip，RS-Box 端 ip 为

192.168.1.102

    communicator->setPort(60002);// 设置端口号，obd 为 60002

    communicator->initSender();//初始化发送对象

    sub_vel = nh.subscribe("/canbus/canbus",1,callback); //订阅 obd 消息

    ros::spin();
    return 0;
}
```

6 功能输出指南

在 RS-Box 正确连接，正常启动工作后，用户就可以在之相连的接收端接收和解析激光雷达的位置（在有地图的情况下）和周围环境的感知结果。RS-Box 是以 UDP socket 的形式发送出位置信息和感知结果。用户可以通过 Robosense::Communication 类轻松的解析成 Robosense::Pose 位姿信息和 Robosense::PerceptResultMsg 感知结果。

解析的程序包含在 communication 文件夹（https://pan.baidu.com/s/1cKxcPw, extract code: pwnz 中 SDK 软件包中的/tools/communication）中。

6.1 输出信息解析

用户把 communication 文件夹拷贝用户对应的工程中去，并配置到工程的构建系统里去（以 CMakeLists.txt 为例）。

解析的样本程序(src/rs_box_parse.cpp)如下：

```
#include "communication.h"
int main()
{
  Robosense::Communication *communicator;
  communicator = new Robosense::Communication();
  communicator->initReceiver();

  communicator->setPort(60000);  // 设置端口号，与发数据端口一致，默认 60000


  Robosense::Header header;       // 单帧传输数据 head，包括字符长度，传输分割目标个数，帧号 id，时间戳等信息

  Robosense::PoseMsg pose;            // 由定位返回的当前帧 lidar 的位置，姿态信息

  std::vector<Robosense::PerceptResultMsg> result_msgs; // 单帧所有的分割目标感知信息
```

```
  while(true)
  {
    std::vector<Robosense::PerceptResultMsg>().swap(result_msgs);  //  清空
result_msgs 所占内存
    communicator->receMsg(result_msgs, header, pose);

    // 用户可以从获得的 result_msgs, header, pose，获取自己需要的对应信息
    // ......
  }
 }
```

在工程中的 CMakeLists.txt 添加如下内容:

```
include_directories(
    communication

    )

 add_executable(rs_bos_parse
    src/rs_box_parse.cpp
    communication/communication.cpp
  )
```

6.2 输出数据结构

Robosense::Header 数据结构包括传输分割目标个数，字符总长度，帧号 id, 时间戳等信息。

Robosense::PoseMsg 信息包括激光雷达所在地图上坐标系的 x, y, z 位置和 roll, pitch, yaw 姿态信息。

Robosense::PerceptResultMsg 感知结果主要包括障碍物的 bounding box 信息，障碍物跟踪的 ID 信息，障碍物相对激光雷达的线速度，角速度，线加速度，角加速度信信息，以及障碍物的分类信息。

其中定位信息和感知输出信息的主要数据结构如下（详细请参见附录 communication.h 和 communication_type.h）:

```
/**
 * @brief header information of frame translation data
 * object_num: num of objects of perception results
 * length: the total bytes of the translated data
 * frame_id: frame_id of frame
 * timestamp: timestamp of the frame
 */
struct Header
{
  unsigned short int head;
  unsigned short int object_num;
  unsigned int length;
  unsigned long int frame_id;
  unsigned long int timestamp;
};

/**
 * @brief pose information, x, y, z represent the location, and roll, pitch, yaw
represent the attitude
 */
struct PoseMsg
{
  float x;
  float y;
  float z;
  float roll;
  float pitch;
  float yaw;
};

/**
 * @brief bounding box info for object
 * length: length of object
 * width: width of object
 * height: height of object
 * location: position of object center
 * direction: direction vetor of object front
 */
struct boxMsg
{
  float length;
  float width;
  float height;
  Point3fMsg location;
```

```
  Direction3fMsg direction;
};


/**
 * @brief perception result message of one object
 * box: the bounding box information of the object
 * track_id: the tracking id of object
 * track_probability: the possibility for an object belonging to a tracker
 * velocity: the velocity of object
 * accelerate: the acceleration of object
 * label: the classify label of object, 0-unknown, 1-pedestrian, 2-bicycle, 3-car,
4-truck
 */
struct PerceptResultMsg
{
  BoxMsg box;
  uint32_t track_id;
  float track_probability;
  Point2fMsg velocity;
  Point2fMsg accelerate;
  signed char label;
};
```

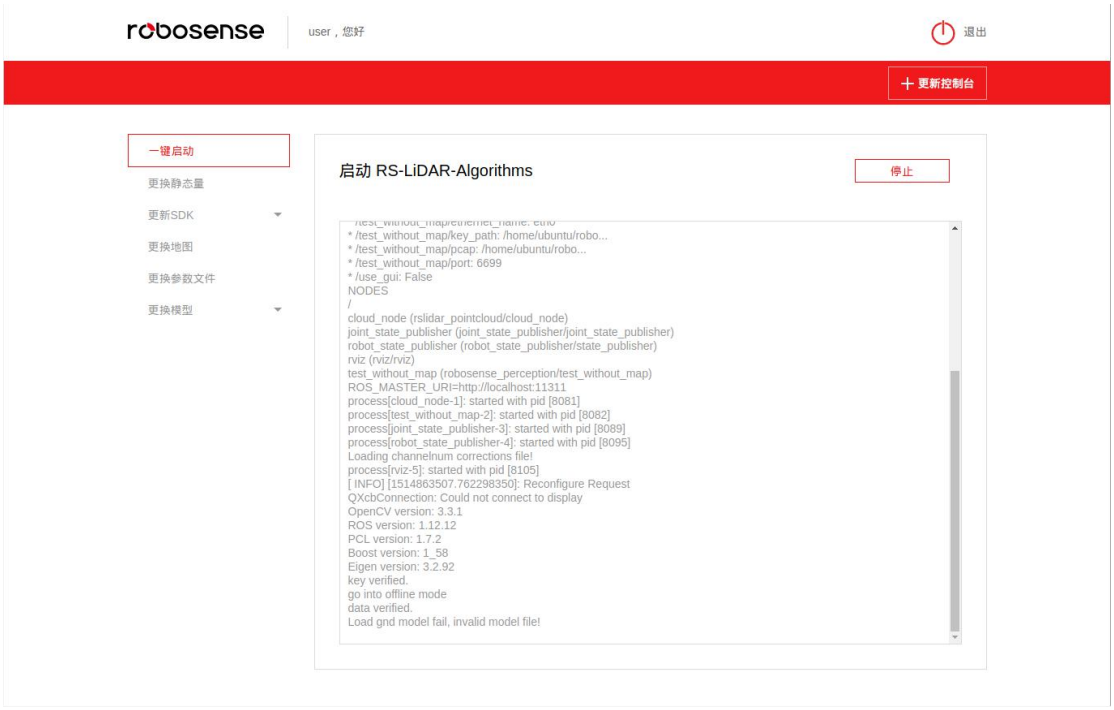
7 Web 设置

Web 设置功能为满足对 RS-Box 的基本操作而专门设计的,它具有启动,简单调试 RS-LiDAR-Algorithm,更新 RS-LiDAR-Algorithm SDK,修改 RS-LiDAR-Algorithm 参数,替换 RS-LiDAR 的静态参数,以及更新定位地图等功能。用户可以根据自身的需求,轻松定制 RS-Box。

在用户接收端打开 web 浏览器,输入 Rs-Box 的 IP 地址(自带网卡 IP 默认为 192.168.1.102,附赠 USB 网卡 IP 默认为 192.168.2.102)即可访问。

7.1 启动调试 RS-LiDAR-Algorithm

连接雷达时需使用自带网卡连接交换机,以使得 Rs-Box 和雷达处在同一网段。在工作终端的浏览器打开 http://192.168.1.102,进入启动页面,点击启动感知程序按钮。下方会简单显示启动的控制台输出,便于用户调试。



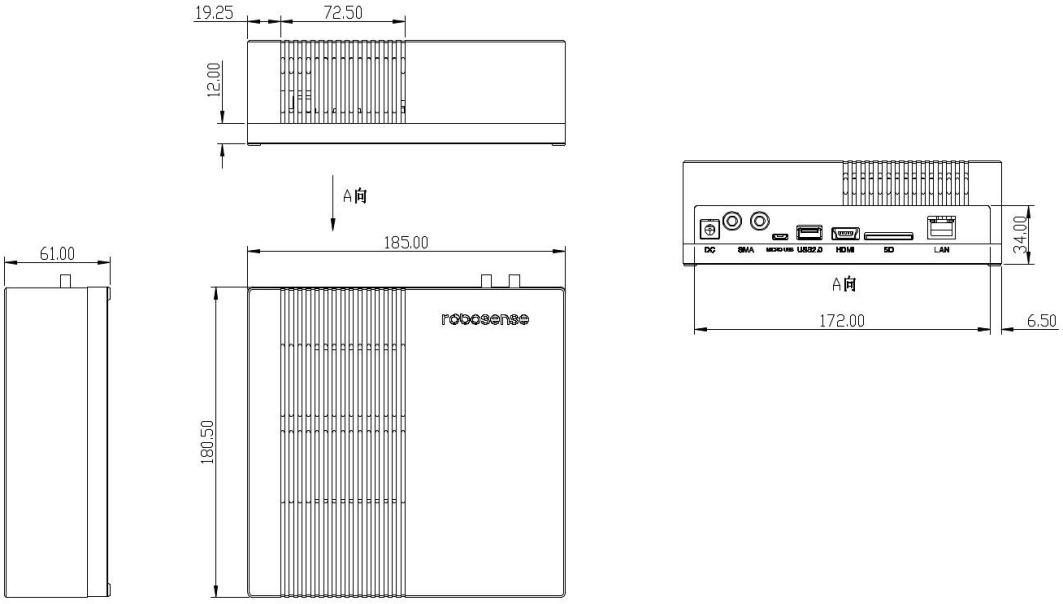
7.2 在线更新

使用其他在线更新功能时需要使用工作终端通过网线连接 Rs-Box 的 USB 网卡,此时工作终端和 Rs-Box 处在 19.168.2.0 网段。再将 Rs-Box 通过自带网卡或者 WiFi 连接到互联网即可使用在线更新功能,需确保自带网卡或 WiFi 所分配的 IP 不在 192.168.2.0 网段。

SDK 更新和控制台更新支持在线更新(一键更新)。在线更新需要 Rs-Box 可以连接互联网。更新时后台会自动检测当前版本是否是最新版本,如果不是将访问服务器自动下载最新的 SDK 并安装。

SDK 更新和配置文件都支持手动更新。用户可在 Rs-Box 官方后台(http://www.robosense.cn/rs-box)下载最新的最新的更新包或者配置文件,再通过上传按钮上传更新。

附录 A 机械尺寸


19.25
72.50
12.00
A向
61.00
185.00
180.50
robosense
DC
SMA
USB2.0
HDMI
SD
LAN
34.00
A向
172.00
6.50

400 6325 830

激光雷达 看见大世界

More than what you see

深圳市速腾聚创科技有限公司
Shenzhen Suteng Innovation Technology Co., LTD.

Address: 深圳市南山区桃源街道留仙大道 1213 号众冠红花岭工业南区 1 区速腾科技楼
Robosense Building, Block 1, South of Zhongguan Honghualing Industrial District, No. 1213 Liuxian Avenue, Taoyuan Street, Nanshan District, Shenzhen, China.

Web：www.robosense.ai Tel: 0755-8632-5830 Email: Service@sz-sti.com