Robosense SDK localization 模块简介

修订历史

文档版本	修订内容	修订时间	拟制
0.0.1	初次发行	2021-11-24	Yufan

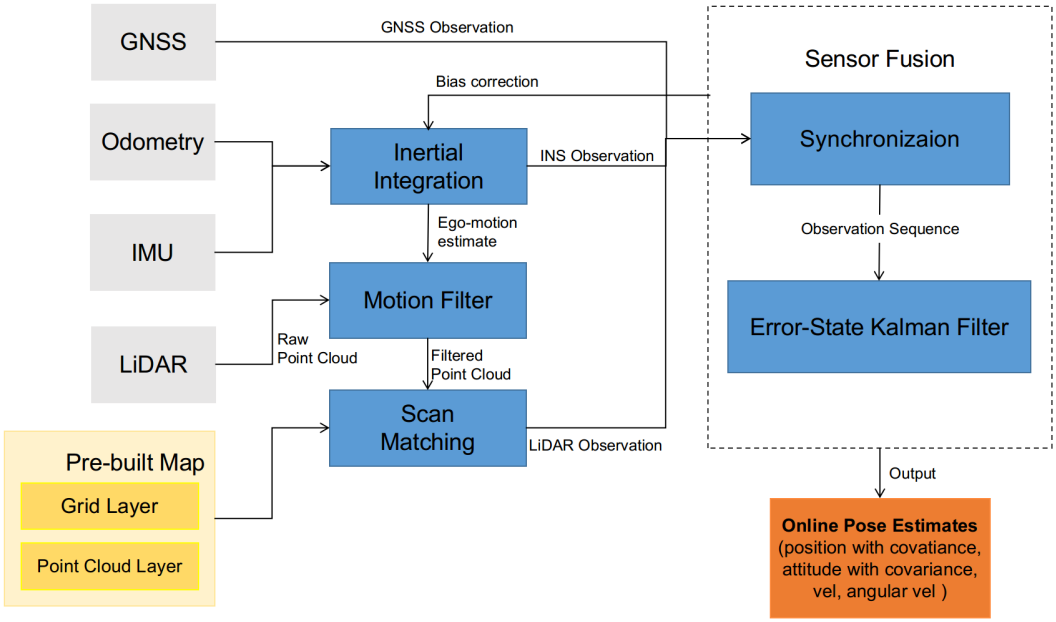
1、关于本文档

- 本文档是对**Robosense SDK** （version>= v3.1）中的**localization**模块的简介和使用说明。
- 下文中，**Robosense SDK**简称为**RS-SDK**
- 适用的**RS-SDK**版本 ≥ 3.1.0
- 下文中，用`root`来表示rs_sdk的根目录路径。其他文件的路径用**相对路径**来描述。比如，`config/system_config`或者`root/config/system_config`表示的是`~/rs_sdk_3.1_release/config/system_config`。

2、localization 模块简介

RS-SDK的**localization**模块是**Robosense**公司研发的以激光雷达为主要传感器的定位软件。该软件的定位算法基于高精度地图和多传感器融合，在激光雷达的基础上融合IMU、RTK和车速等传感器信息，为自动驾驶系统提供实时的定位信息。

localization模块采用了松耦合的融合框架，如下图所示。



前端每一个传感器单独地对一个或多个状态量进行观测。上图中，GNSS可对位置进行测量，IMU可对速度，角速度和位置进行测量，Odometry是对速度进行测量。而LiDAR则是基于与先验地图的匹配(scan matching)，给出关于位置和姿态的观测。前端传感器的观测值被送入后端的融合模块。融合模块是基于一个考虑了传感器之间消息同步误差的误差状态卡尔曼滤波器。滤波器最终输出关于位置，姿态，速度，角速度的估计值，也就是rs_localization输出的实时定位结果。

图中所依赖的两份地图文件(Pre-built Map)须使用**RS-MAP**建图软件生成。生成地图的具体方法请参考**RS-MAP**的说明文档。

3、快速启动

3.1 编译

编译前，确保`root/CMakeLists.txt`中的`COMPILE_LOCALIZATION`选项设置为`ON`。

```
set(COMPILE_LOCALIZATION ON)
```

在`root`中，执行

```
mkdir build
cd build && cmake .. && make -j
```

3.2 参数配置

首先，将两份地图文件`*.gridmap`和`*.rsmap`放入`config/system_config/localization_config`文件夹中。

然后，在`config/usr_config/usr_config.yaml`文件中，设置如下参数：

```
general:
```

```
  application: "Pseries"  # 只有Pseries支持运行localization
  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，根据需要进行设置
  run_localization: true # 开启localzation
  run_communication: false  # 是否运行通信模块

# 配置定位参数
localization:
  common:
    grid_map: localization_map.gridmap # 设置为相应的地图文件名
    rsmap: localization_map.rsmap  #设置为相应的地图文件名
    localization_mode: 0 # 定位模式，下文中单独介绍
```

3.2.1 传感器配置

传感器配置文件在 `config/system_config/sensor_config` 中。

使用IMU、GNSS、车速这三种传感器，分别设置 `config/system_config/sensor_config` 中的 `rs_imu.yaml`、`rs_gnss.yaml`、`rs_odom.yaml`。将其中的 `common:message_source` 设为0，可关闭该传感器使用，设为其他正数，表示不同的传感器数据来源。

使用LiDAR时，需要在 `config/system_config/sensor_config/lidar/rs_lidar.yaml` 中的 `lidar` 部分中以yaml列表的形式添加一个或多个lidar的配置。

传感器之间的相互空间关系需要通过外参文件输入到RS_SDK中。外参文件在 `config/usr_config/calibration.yaml` 中。

3.2.1 定位初始化方法配置

localization模块提供了3种获得**初始位置**的方式，通过 `localization:common:localization_mode` 来设置。

localization_mode	模式
0	使用实时收到的GPS位置进行初始位姿计算，这是最常用的初始化方式
2	使用给定的经纬度进行初始位姿计算
4	使用给定的xyz坐标进行初始位姿计算，xyz是表示在地图坐标系中的三维位置

3.2.2 运动学估计方法设置

localization 中实现了两种从传感器数据估计车身运动的方法。一种是单独使用IMU，另外一种是同时使用IMU和车速。

(1) 使用IMU+LiDAR进行定位，`config/system_config/localization_config/default_config/default.yaml` 中，选择 `ImuObserver`

```
- include:
/system_config/localization_config/default_config/Observer/ImuObserver.yaml
# - include:
/system_config/localization_config/default_config/Observer/ImuOdomObserver.yaml
```

(2) 使用车速+IMU+LiDAR进行定位，则选择`ImuOdomObserver`

```
# - include:
/system_config/localization_config/default_config/Observer/ImuOdomObserver.yaml
- include:
/system_config/localization_config/default_config/Observer/ImuOdomObserver.yaml
```

注意，这两个配置只能二选一，必须用`#`注释掉其中一个。

3.2.3 使用地图类型设置

localization模块支持使用两种地图进行定位。一种是使用栅格地图gridmap, 一种是使用点云地图rsmap。使用gridmap进行定位需要使用NVidia显卡和CUDA。

建议使用gridmap进行定位。

(1) 使用gridmap时，
`config/system_config/localization_config/default_config/default.yaml`中，选择grid_map_server 和 ProbabilityHistogramObserver。

```
  Observers:
    # - include:
/system_config/localization_config/default_config/Observer/LidarEsObserver.yaml
    - include:
/system_config/localization_config/default_config/Observer/ProbabilityHistogramOb
server.yaml
  MapServer:
    include:
/system_config/localization_config/default_config/grid_map_server.yaml
  # include: /system_config/localization_config/default_config/rs_map_server.yaml
```

(2) 使用rs_map时，
`config/system_config/localization_config/default_config/default.yaml`中，选择rs_map_server 和 LidarEsObserver。

```
  Observers:
    - include:
/system_config/localization_config/default_config/Observer/LidarEsObserver.yaml
    # - include:
/system_config/localization_config/default_config/Observer/ProbabilityHistogramOb
server.yaml
  MapServer:
    # include:
/system_config/localization_config/default_config/grid_map_server.yaml
    include: /system_config/localization_config/default_config/rs_map_server.yaml
```

3.3 运行

上述提到的所有参数配置完毕时候，运行sdk的demo程序即可

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

4 获取定位结果

目前提供两个发送定位的方式，一种是通过ROS消息发出，另一种是使用protobuf对定位结果序列化后通过UDP发出。

在`config/usr_config/usr_config.yaml`在中的`localization: result_sender`中可以配置发送方式。

4.1 通过ROS发送

定位结果通过`nav_msgs/Odometry`消息类型发送定位消息，里面包含了车辆当前的位置，朝向，速度，角速度等信息，同时还会将定位结果转换成经纬度，以`sensor_msgs/NavSatFix`的消息类型发出。

```
  result_sender:
    - method: Ros
      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
```

当定位结果通过ROS发送，可通过Rviz工具来可视化定位结果：

```
rviz -d config/rviz/localization.rviz
```

4.2 通过protobuf发送

```
  result_sender:
   - 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
```

定位结果的protobuf消息定义如下：

```
syntax="proto2";
package Proto_msg;
message VehicleState
{
 optional double timestamp = 1;
 optional uint32 seq=2;
 optional string parent_frame_id=3;
 optional string frame_id=4;
 optional uint32 status=5;              // 当前定位状态
 repeated double origin=6;              // 地图原点经纬度
 repeated double fix=7;                 // 定位位置结果：经纬度
 repeated double pos=8;                 // 定位位置结果：朝向
 repeated double pos_cov=9;             // 定位位置置信度（协方差）
 repeated double orien=10;              // 定位朝向结果
 repeated double orien_cov=11;          // 定位朝向置信度（协方差）
 repeated double angular_vel=12;        // 角速度
 repeated double angular_vel_cov=13;    // 角速度置信度（协方差）
 repeated double linear_vel=14;         // 线速度
 repeated double linear_vel_cov=15;     // 线速度置信度（协方差）
 repeated double acc=16;                // 加速度
 repeated double acc_cov=17;            // 加速度置信度（协方差）
}
```

4.3 定位状态定义

localization模块的状态消息数据类型为一个uint32_t (32位的无符号整形)，其中包括了几个与该模块相关的不同状态量, 具体定义如下:

位数	状态	详细定义
1	定位模块初始化	0: 未初始化 1: 已经初始化
2	密钥检验结果	0: 密钥检验未开始或未通过 1: 密钥检验通过
3 ~5	定位初始化状态	000: 无定义 001: 未获得初始位姿 010: 正在计算初始位姿 011: 重新计算初始位姿 100: 定位初始化失败 101: 初始化成功
6~8	定位结果状态	000: 无定义 001: 空闲 (未开始进行定位) 010: 定位正常 011: 定位精度差 100: 定位丢失
9~32	保留	无定义

使用时可以通过位运算获取相应的状态。这里推荐使用定义一个union的方式来获取状态。

定义一个联合体(union)如下:

```
union RSLocalizationStatus
{
    struct
    {
      unsigned int module_initialized : 1;
      unsigned int key_check : 1;
      unsigned int localization_init : 3;
      unsigned int localization_status : 3;
    };
    uint32_t status_int = 0;
 };
```

假设获得了一个`uint32_t`的定位状态`raw_status`,可通过如下代码解析:

```
uint32_t raw_status = getStatusFromSomewhere();

RSLocalizationStatus status;
status.status_int = raw_status;

int loc_state = status.localization_status;
if(loc_state == 1)
    std::cout << "定位未开始" <<  std::endl;
else if(loc_state == 2)
    std::cout << "定位正常" <<  std::endl;
else if(loc_state == 3)
    std::cout << "定位精度差" <<  std::endl;
else if(loc_state == 4)
```

```
    std::cout << "定位丢失" <<  std::endl;
```