# Leaderboard与Dora联合仿真系统

**使用说明文档**

版本记录和版本号变动与修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 更改描述 | 创建/更改日期 | 创建/更改人 |
| 1.0 | 初始版本 | 2025-4-23 | 袁梓恒，杨毅培，赵华 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[Leaderboard与Dora联合仿真系统 1](#_Toc196483333)

[版本记录和版本号变动与修订记录 2](#_Toc196483334)

[一． 上位机--CARLA自动驾驶排行榜Leaderboard使用说明 5](#_Toc196483335)

[1. Leaderboard介绍 5](#_Toc196483336)

[1.1 Leaderboard简介 5](#_Toc196483337)

[2. 运行环境及其部署 6](#_Toc196483338)

[2.1 运行环境 6](#_Toc196483339)

[2.2 CARLA排行榜包安装 6](#_Toc196483340)

[3. 使用Leaderboard创建代理 8](#_Toc196483341)

[3.1启动CARLA 8](#_Toc196483342)

[3.2启动Leaderboard 8](#_Toc196483343)

[二． 下位机--自动驾驶DORA平台软件系统使用说明 10](#_Toc196483344)

[1. DORA介绍 10](#_Toc196483345)

[1.1DORA简介 10](#_Toc196483346)

[2. 运行环境及其部署 11](#_Toc196483347)

[2.1运行环境 11](#_Toc196483348)

[2.2Dora安装 11](#_Toc196483349)

[2.3其他库安装 12](#_Toc196483350)

[3. 节点创建以及启动 13](#_Toc196483351)

[3.1Dora的node示例（C++语言） 13](#_Toc196483352)

[4. 各节点功能说明 16](#_Toc196483353)

[4.6 接收节点 16](#_Toc196483354)

[4.7 处理GNSS节点 17](#_Toc196483355)

[4.8 Map节点 18](#_Toc196483356)

[4.9 Planning节点 19](#_Toc196483357)

[4.10 雷达处理节点 20](#_Toc196483358)

[4.11 控制和发送节点 21](#_Toc196483359)

[4.12 可视化节点 21](#_Toc196483360)

[5. 程序运行 22](#_Toc196483361)

[三．Leaderboard与Dora的联合仿真系统说明 24](#_Toc196483362)

[1. 概述 24](#_Toc196483363)

[2. 核心依赖库 24](#_Toc196483364)

[3. 传感器数据与车辆控制指令处理 24](#_Toc196483365)

[3.1 GPS模块 24](#_Toc196483366)

[3.2 IMU模块 25](#_Toc196483367)

[3.3 LiDAR模块 25](#_Toc196483368)

[3.4 车辆控制指令处理 26](#_Toc196483369)

[3.5 关键配置参数 27](#_Toc196483370)

[3.6 数据流向图 27](#_Toc196483371)

[4. 使用说明 27](#_Toc196483372)

[5. 测试Demo要点与说明 27](#_Toc196483373)

# 上位机--CARLA自动驾驶排行榜Leaderboard使用说明

## Leaderboard介绍

### 1.1 Leaderboard简介

CARLA AD 排行榜要求 AD 代理通过一组预定义的路线。对于每条路线，代理将在起点初始化并指示驾驶至目的地，通过 GPS 样式坐标、地图坐标或路线说明提供路线描述。路线在各种情况下定义，包括高速公路、城市地区、住宅区和农村环境。排行榜在各种天气条件下评估 AD 代理，包括日光场景、日落、雨、雾和夜晚等。



代理将面临基于 [NHTSA 类型的](https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/pre-crash_scenario_typology-final_pdf_version_5-2-07.pdf)多种流量场景。可以在[此页面](https://leaderboard.carla.org/scenarios/)中查看流量场景的完整列表，以下是一些示例。

* 车道合并。
* 变道。
* 在交通路口进行谈判。
* 环形交叉路口的谈判。
* 处理交通信号灯和交通标志。
* 让位于紧急车辆。
* 应对行人、骑自行车的人和其他因素。

（详情可见：[CARLA 自动驾驶排行榜](https://leaderboard.carla.org/)）

## 运行环境及其部署

### 2.1 运行环境

硬件环境：主机（性能要求：推荐显存大于8G，内存大于16G，硬盘大于100G）

软件环境：Linux系统/Ubuntu20.04

### 2.2 CARLA排行榜包安装

1. 下载CARLA排行榜

下载打包的CARLA排行榜版本

<https://leaderboard-public-contents.s3.us-west-2.amazonaws.com/CARLA_Leaderboard_2.0.tar.xz>

为了使用 CARLA Python API，需要安装对应的依赖项。以下是使用conda环境的示例：

# 创建虚拟环境并安装对应依赖

**conda create -n py37 python=3.7**

**conda activate py37**

**cd ${CARLA\_ROOT} # Change ${CARLA\_ROOT} for your CARLA root folder**

**pip3 install -r PythonAPI/carla/requirements.txt**

1. 获取排行榜和场景运行程序

# 下载 Leaderboard 存储库 leaderboard-2.0 分支

**git clone -b leaderboard-2.0 --single-branch** [**https://github.com/carla-simulator/leaderboard.git**](https://github.com/carla-simulator/leaderboard.git)

# 安装leaderboard-2.0分支所需的 Python 依赖项。

**cd ${LEADERBOARD\_ROOT} # Change ${LEADERBOARD\_ROOT}you’re your Leaderboard root folder**

**pip3 install -r requirements.txt**

# 获取Scenario Runner 存储库 leaderboard-2.0 分支

**git clone -b leaderboard-2.0 --single-branch** [**https://github.com/carla-simulator/scenario\_runner.git**](https://github.com/carla-simulator/scenario_runner.git)

#安装Scenario Runner 存储库所需的 Python 依赖项

**cd ${SCENARIO\_RUNNER\_ROOT} # Change ${SCENARIO\_RUNNER\_ROOT} for your Scenario\_Runner root folder**

**pip3 install -r requirements.txt**

1. 定义环境变量

#打开配置文件

**gedit ~/.bashrc**

#在配置文件中添加以下定义

**export CARLA\_ROOT=PATH\_TO\_CARLA\_ROOT**

**export SCENARIO\_RUNNER\_ROOT=PATH\_TO\_SCENARIO\_RUNNER**

**export LEADERBOARD\_ROOT=PATH\_TO\_LEADERBOARD**

**export PYTHONPATH="${CARLA\_ROOT}/PythonAPI/carla/":"${SCENARIO\_RUNNER\_ROOT}":"${LEADERBOARD\_ROOT}":"${CARLA\_ROOT}/PythonAPI/carla/dist/carla-0.9.14-py3.7-linux-x86\_64.egg":${PYTHONPATH}**

#使更改生效

**source ~/.bashrc**

## 使用Leaderboard创建代理

### 3.1启动CARLA

（1）在一个终端中运行CARLA服务器

**cd ${CARLA\_ROOT}**

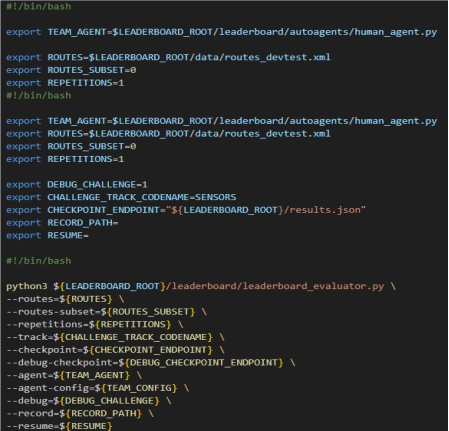
**./CarlaUE4.sh -quality-level=Epic -world-port=2000 -resx=800 -resy=600**

### 3.2启动Leaderboard

（1）在另一个终端中，导航到${LEADERBOARD\_ROOT}.虽然 Leaderboard 是使用 leaderboard\_evaluator.py脚本运行的，但使用的参数数量较多，直接使用终端执行操作可能会比较麻烦。因此，建议使用 bash 脚本。本排行榜提供了 run\_leaderboard.sh 脚本，运行脚本：

**./run\_leaderboard.sh**

（2）run\_leaderboard.sh脚本如下：



1. 这将启动一个pygame窗口，为您提供手动控制代理的选项。沿着彩色航点指示的路线到达您的目的地。该脚本在Town 12中加载两条路由。这有助于您了解Leaderboard。
2. 如何创建您自己的自动驾驶代理，详情可参阅[开始使用排行榜 2.0 - CARLA 自动驾驶排行榜](https://leaderboard.carla.org/get_started_v2_0/)

# 下位机--自动驾驶DORA平台软件系统使用说明

1. **DORA介绍**

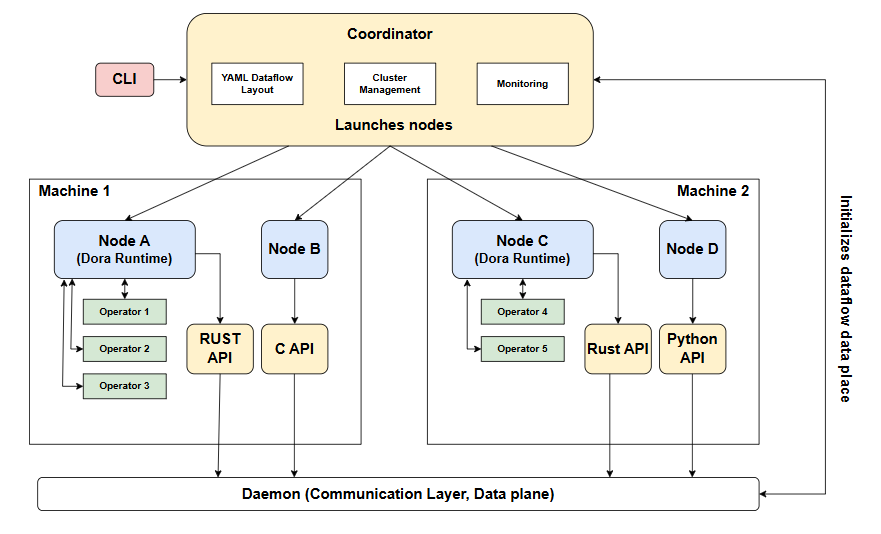
**1.1DORA简介**

面向数据流的机器人架构（Dataflow - Oriented Robotic Architecture，DORA）是一个能让机器人应用程序开发变得快速且简单的框架。dora - rs是它的一种实现。

dora - rs实现了一种声明式数据流范式，其中任务被拆分到作为独立进程隔离的各个节点之间。每个节点定义其输入和输出，以便与其他节点相连。



数据流范例的优势在于创建一个抽象层，使机器人应用程序模块化且易于配置。



（详情可见：https://github.com/dora-rs/dora）

1. **运行环境及其部署**

**2.1运行环境**

硬件环境：控制器（可使用英伟达的嵌入式平台，NVIDIA Orin，台式机，笔记本等）

软件环境：Ubuntu20.04

DORA版本:0.3.8

**2.2Dora安装**

1. 下载Dora官方git库

#更新rustc版本，安装cargo：

**sudo apt autoremove rustc**

**curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh**

**export PATH=~/.cargo/bin:$PATH**

**sudo apt install cargo**

#下载Dora官方git库

**mkdir dora\_project**

**cd dora\_project**

**git clone https://github.com/dora-rs/dora.git**（注意事项：#下载Dora官方仓库，默认下载最新版本，安装之前的版本需要到[Releases · dora-rs/dora (github.com)](https://github.com/dora-rs/dora/releases)下自主选择版本安装，我们默认选择0.3.8进行安装）

**cd dora**

**cargo build -p dora-cli --release**

1. 添加工程目录到环境变量

# add dora PATH for .bashrc

**gedit ~/.bashrc**

# add the code into .bashrc in last line **"PATH="$PATH:/home/xxxx/dora/target/release"**

1. 编译

# install dora for python

**pip install dora-rs==0.3.8 --force**

# build dora c++ API

**cd dora/examples/c++-dataflow**

#编译C++的节点库

**cargo run --example cxx-dataflow**

#编译dora-node-api-c库

**cargo build -p dora-node-api-c --release**

**2.3其他库安装**

如果需要可视化路径与雷达点云，需安装rerun，安装使用详情见博客<https://blog.csdn.net/candygua?ops_request_misc=%7B%22request%5Fid%22%3A%2292496963808d6b4e610cd09d76df32e5%22%2C%22scm%22%3A%2220140713.130064515..%22%7D&request_id=92496963808d6b4e610cd09d76df32e5&biz_id=206&utm_medium=distribute.pc_search_result.none-ta>

1. **节点创建以及启动**

**3.1Dora的node示例（C++语言）**

**3.1.1编译node节点（生成可执行文件）**

1.clang++编译

1.1运用clang++编译node-c-api/main.cc，并且链接静态库。

#在../dora/examples/c++-dataflow目录下执行如下命令

**mkdir build** #执行cargo run --example cxx-dataflow 后会有build文件夹，这步可以跳过，主要运用于自己创建文件夹的时候，需要创建build文件夹保存输出的文件。

**clang++ node-c-api/main.cc -lm -lrt -ldl -lpcap -pthread -std=c++14 -ldora\_node\_api\_c -L ../../target/release --output build/node\_c\_api**

注意：如果找不到头文件node\_api.h。在node-c-api/main.cc将#include"../../../apis/c/node/node\_api.h"改为自己电脑的路径，例如#include"/home/disk/dora\_project/dora/apis/c/node/node\_api.h"

1.2编译operator节点

运用clang++编译operator-c-api/operator.cc生成静态库和动态库。

#生成静态库

**clang++ -c operator-c-api/operator.cc -std=c++14 -o build/operator\_c\_api.o -fPIC**

#生成动态库

**clang++ -shared build/operator\_c\_api.o -o build/liboperator\_c\_api.so -ldora\_operator\_api\_c -L /home/nvidia/dora\_project/dora-rs/dora/target/debug**

#注意：/home/nvidia/dora\_project/dora-rs/dora/target/debug需要更换为你自己的路径，其目的是为了链接dora\_operator\_api\_c库。

注意：如果找不到头文件operator\_api.h。在operator-c-api/operator.cc将#include "../../../apis/c/operator/operator\_api.h"改为自己电脑的路径，例如#include "/home/disk/dora\_project/dora/c/apis/c/operator/operator\_api.h"

1. cmake编译

#进入工作空间

mkdir build

cd build

cmake ..

make

**3.1.2编写test.yml文件（测试）**

#创建test.yml并运行

touch test.yml

#并将以下代码写入其中：

nodes:

- id: cxx-node-c-api

custom:

source: build/node\_c\_api

inputs:

tick: dora/timer/millis/300

outputs:

- counter

- id: runtime-node-1

operators:

- id: operator-c-api

shared-library: build/operator\_c\_api

inputs:

counter: cxx-node-c-api/counter

outputs:

- half-status

**3.1.3运行示例**

#运行test.yml文件

dora up

dora start test.yml --name test

#查看日志

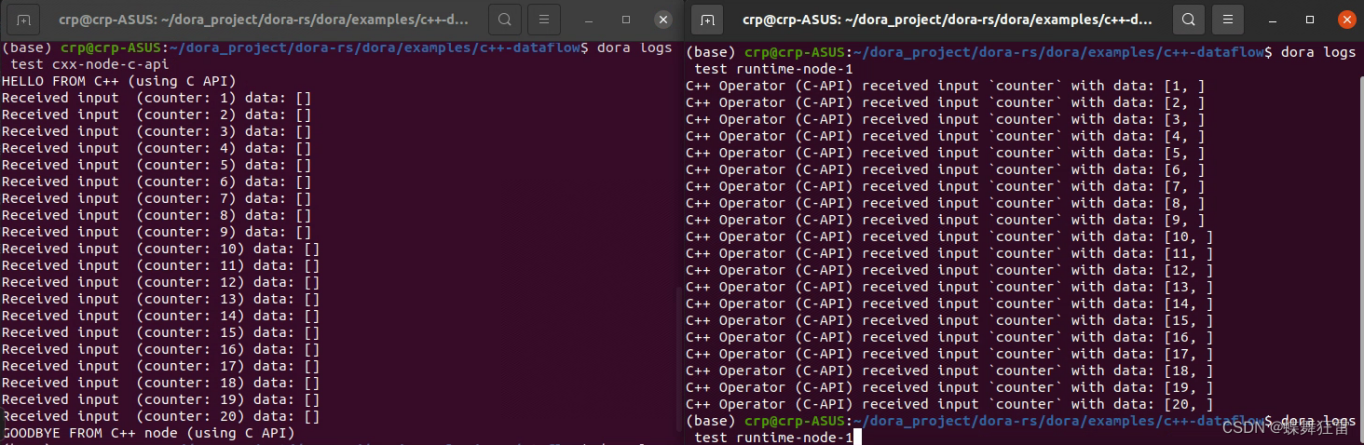
#打开一个命令窗

dora logs test cxx-node-c-api

#打开另一个命令窗

dora logs test runtime-node-1

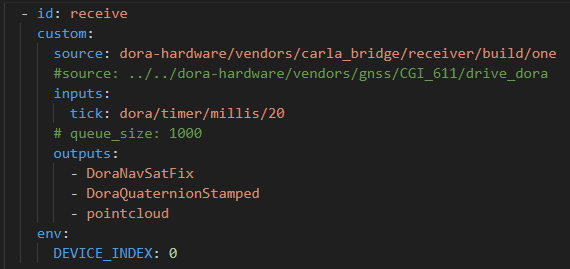
#运行结果



左边是cxx-node-c-api的日志，因为输入的是Dora的定时器没有数据。  
右边是runtime-node-1的日志，可以看见输入了从1到20的数据。

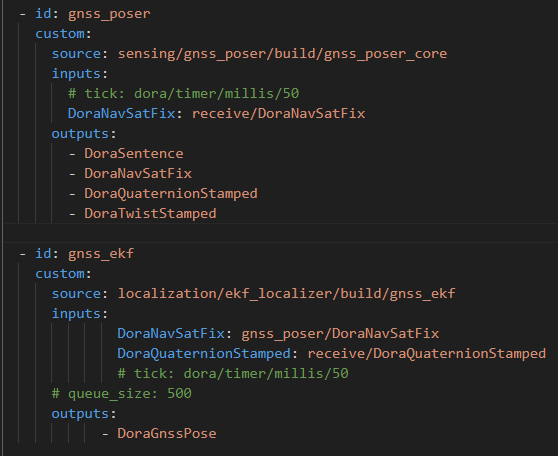
（详情见：https://blog.csdn.net/weixin\_44112228/article/details/135607575?ops\_request\_misc=&request\_id=&biz\_id=102&utm\_term=dorars&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~sobaiduweb~default-4-135607575.142^v101^pc\_search\_result\_base2&spm=1018.2226.3001.4187）

1. **各节点功能说明**
   1. **接收节点**



功能：接收上位机发送的GPS,IMU以及点云信息

* 1. **处理GNSS节点**



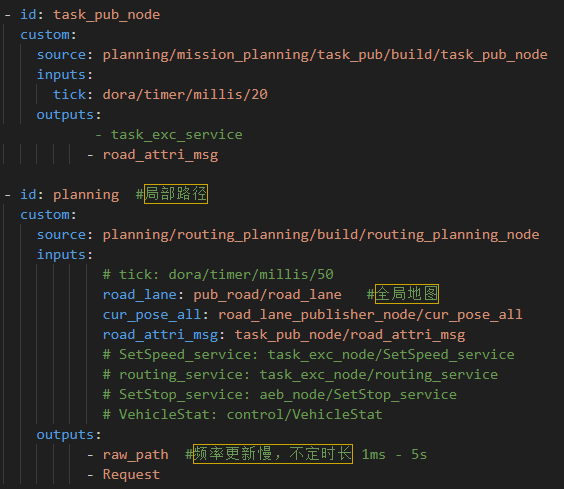
功能：处理GNSS信息，坐标及位姿转换

* 1. **Map节点**



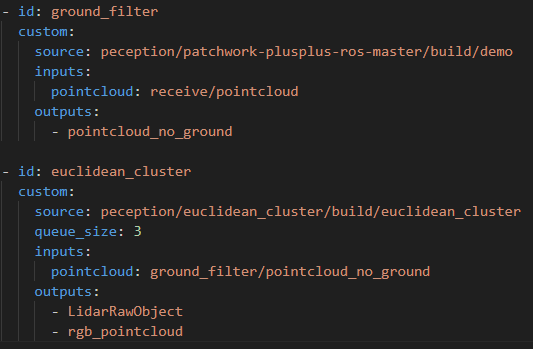
功能：发布地图轨迹及计算在地图下的自身位姿

* 1. **Planning节点**



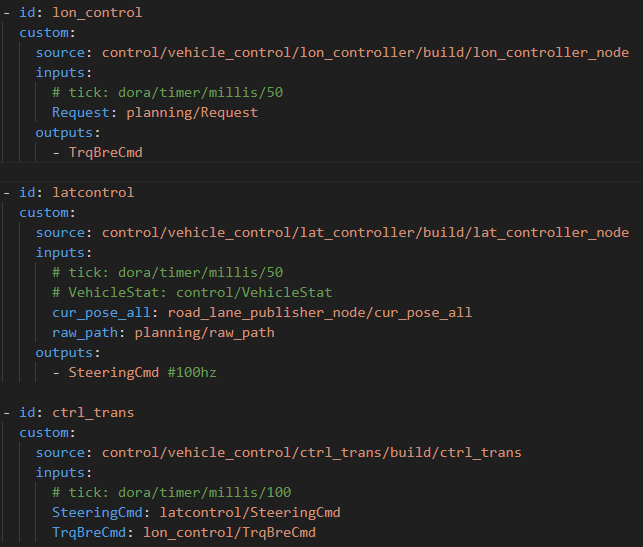
功能：任务发布及路径规划

* 1. **雷达处理节点**



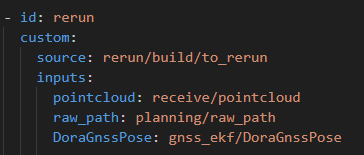
功能：处理雷达点云信息

* 1. **控制和发送节点**



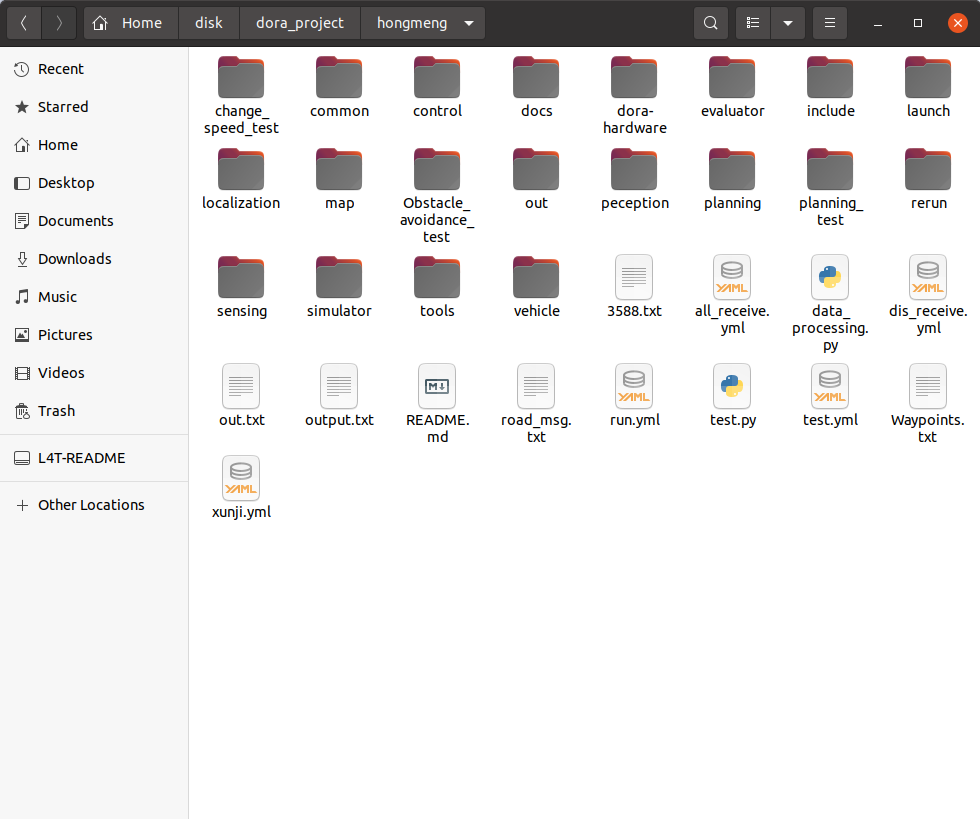
功能：横向控制和纵向控制以及向上位机发送控制信息

* 1. **可视化节点**

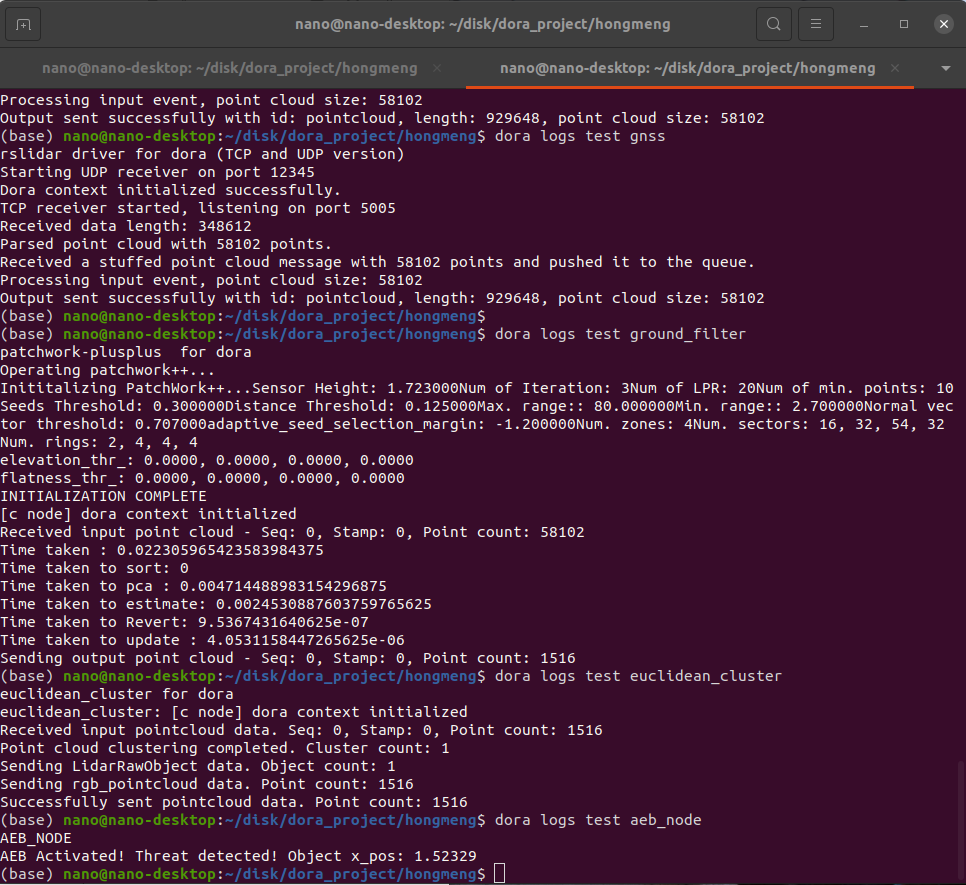


功能：可视化路径与点云信息

1. **程序运行**



在yml所在目录打开终端运行，运行方法见3.1.3



以上为各节点打印信息示例

# 三．Leaderboard与Dora的联合仿真系统说明

## 概述

本系统基于CARLA模拟器开发，与Dora通信，实现自动驾驶代理功能。核心功能涵盖传感器数据采集（GNSS、IMU、LiDAR）、数据处理（点云降采样、镜像变换）、网络通信（UDP/TCP）及车辆控制指令执行。通过多线程技术实现异步数据处理，保障系统实时性与稳定性。

## 核心依赖库

| 库名 | 功能描述 | 版本要求 |
| --- | --- | --- |
| socket | 网络通信 | 内置库 |
| json | 数据序列化与反序列化 | 内置库 |
| math | 数学计算（角度转换、三角函数等） | 内置库 |
| numpy | 数值计算与数组处理 | ≥1.16.0 |
| open3d | 点云处理（降采样） | ≥0.10.0 |
| threading | 多线程处理 | 内置库 |
| struct | 二进制数据打包与解包 | 内置库 |
| carla | CARLA模拟器接口 | 与CARLA版本匹配 |

## 传感器数据与车辆控制指令处理

### 3.1 GPS模块

#### 3.1.1 GPS数据采集配置

| 传感器ID | 位置（x, y, z） | 姿态（roll, pitch, yaw） | 功能描述 |
| --- | --- | --- | --- |
| GPS | (0, 0, 1.60) | (0.0, 0.0, 0.0) | 基础定位数据采集 |
| GPS1 | (1, 0, 1.60) | (0.0, 0.0, 0.0) | 辅助计算车辆航向角 |

#### 3.1.2 GPS数据处理逻辑：

（1） 航向角计算

利用双GPS坐标差（`x2-x1`, `y2-y1`），通过`atan2`函数计算相对于正东方向的航向角，并进行象限修正。

（2）数据发送

将原始GPS坐标（`x`, `y`, `z`）及计算后的航向角打包为JSON格式，通过UDP发送至`UDP\_IP:UDP\_PORT\_GNSS\_IMU`。

（3） 轨迹记录

将GPS坐标写入`gps\_path.txt`文件，用于后续轨迹分析。

### 3.2 IMU模块

#### 3.2.1 IMU数据采集配置

| 传感器ID | 位置（x, y, z） | 姿态（roll, pitch, yaw） | 功能描述 |
| --- | --- | --- | --- |
| IMU | (0, 0, 1.60) | (0.0, 0.0, 0.0) | 采集加速度与角速度数据 |

#### 3.2.2 IMU数据处理逻辑：

（1）数据提取

从IMU数据中解析加速度（`ax`, `ay`, `az`）和角速度（`gx`, `gy`, `gz`）。

（2） 数据融合

结合GNSS计算的航向角，生成包含传感器数据及航向信息的JSON数据包。

（3） 数据发送

通过UDP发送至`UDP\_IP:UDP\_PORT\_GNSS\_IMU`。

### 3.3 LiDAR模块

#### 3.3.1LIDAR数据采集配置

| 传感器ID | 位置（x, y, z） | 姿态（roll, pitch, yaw） | 核心参数 |
| --- | --- | --- | --- |
| LIDAR | (0, 0, 1.8) | (0, 0, -90) | 16通道，10Hz帧率，50000点/秒 |

#### 3.3.2LIDAR数据处理流程

（1） 数据接收

从CARLA模拟器获取LiDAR点云数据（仅保留`x`, `y`, `z`坐标）。

（2）镜像变换

调用`mirror\_point\_cloud`函数沿`x=0`轴进行镜像翻转。

（3） 数据缓冲

将处理后的点云数据存入`lidar\_buffer`，达到`lidar\_batch\_size`（50000点）后触发发送。

（4） 降采样与发送

对缓冲数据进行体素降采样（`voxel\_size=0.3`），转换为16位整数并打包，通过TCP发送至`TCP\_IP:TCP\_PORT\_LIDAR`。

### 3.4 车辆控制指令处理

#### 3.4.1控制指令接收配置

通过UDP监听`UDP\_CONTROL\_IP:UDP\_PORT\_CONTROL`，接收外部控制指令（JSON格式）。

#### 3.4.2控制指令解析与执行

（1）指令解析

从JSON数据中提取`steer`（转向）、`throttle`（油门）、`brake`（刹车）参数。

（2） 指令缓冲

将解析后的指令存入`control\_command`，通过线程锁确保数据一致性。

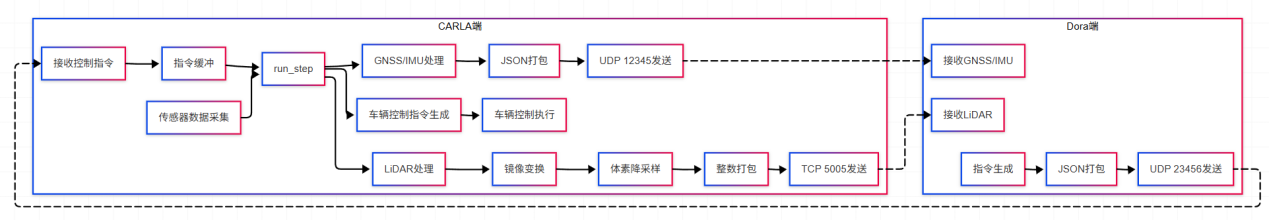
（3） 指令应用

在`run\_step`函数中，将指令应用于`VehicleControl`对象，控制车辆行驶。

### 3.5 关键配置参数

| 参数名 | 描述 | 默认值 | 备注 |
| --- | --- | --- | --- |
| TCP\_IP | LiDAR数据发送目标IP | 192.168.1.101 | 需与接收端匹配 |
| UDP\_IP | GNSS/IMU数据发送目标IP | 192.168.1.101 | 需与接收端匹配 |
| UDP\_CONTROL\_IP | 控制指令接收监听IP | 192.168.1.1 | 本地或远程控制端IP |
| UDP\_PORT\_GNSS\_IMU | GNSS/IMU数据发送端口 | 12345 | 自定义端口号 |
| UDP\_PORT\_CONTROL | 控制指令接收端口 | 23456 | 自定义端口号 |
| TCP\_PORT\_LIDAR | LiDAR数据发送端口 | 5005 | 自定义端口号 |
| Lidar\_batch\_size | LiDAR点云批量发送大小 | 50000 | 单次发送的最大点数 |

### 3.6 数据流向图



## 使用说明

（1）启动 CARLA 服务器

①切换目录：cd /your\_path/CARLA\_Leaderboard\_20

②启动服务器：./CarlaUE4.sh

（2）运行代理脚本

①进入脚本目录：cd /your\_path/CARLA\_Leaderboard\_20/leaderboard

②执行脚本：./test\_run.sh，等待传感器初始化及网络连接。

（3）启动 Dora 端

## ****测试Demo要点与说明****

（1）地图配置调整：本Demo修改`routes\_devtest.xml`属性，将车辆初始位置从默认停车位调整至直线道路，为简化测试场景。

（2）控制参数说明：本Demo未进行油门与转角标定，直接接收来自Dora端的定值控制指令。在Carla车辆中，油门控制值范围为0 ~1，转角控制值范围为-1~1。

（3）传感器设置变更：对`agent\_wrapper.py`属性进行调整，默认单GNSS限制，本Demo采用双GNSS传感器，以此简化车辆航向角的计算流程。