

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
版本记录和版本号变动与修订记录	2
一. 上位机--CARLA 自动驾驶排行榜 Leaderboard 使用说明	5
1. Leaderboard 介绍	5
1.1 Leaderboard 简介	5
2. 运行环境及其部署	6
2.1 运行环境	6
2.2 CARLA 排行榜包安装.....	6
3. 使用 Leaderboard 创建代理.....	8
3.1 启动 CARLA.....	8
3.2 启动 Leaderboard	8
二. 下位机--自动驾驶 DORA 平台软件系统使用说明	10
1. DORA 介绍	10
1.1DORA 简介.....	10
2. 运行环境及其部署	11
2.1 运行环境.....	11
2.2Dora 安装.....	11
2.3 其他库安装.....	12
3. 节点创建以及启动	13
3.1Dora 的 node 示例（C++语言）	13
4. 各节点功能说明	16
4.6 接收节点.....	16
4.7 处理 GNSS 节点	17
4.8 Map 节点	18
4.9 Planning 节点	19
4.10 雷达处理节点.....	20
4.11 控制和发送节点.....	21
4.12 可视化节点.....	21
5. 程序运行	22
三. Leaderboard 与 Dora 的联合仿真系统说明	24
1. 概述	24

2. 核心依赖库	24
3. 传感器数据与车辆控制指令处理	24
3.1 GPS 模块	24
3.2 IMU 模块	25
3.3 LiDAR 模块	25
3.4 车辆控制指令处理	26
3.5 关键配置参数	26
3.6 数据流向图	27
4. 使用说明	27
5. 测试 Demo 要点与说明	27

一. 上位机--CARLA 自动驾驶排行榜 Leaderboard 使用说明

1. 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](#)多种流量场景。可以在[此页面](#)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](#)）

2. 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
```

（2）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-](https://github.com/carla-simulator/scenario_runner.git)

[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

(3) 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

3. 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：

```
#!/bin/bash

export TEAM_AGENT=${LEADERBOARD_ROOT}/leaderboard/autoagents/human_agent.py

export ROUTES=${LEADERBOARD_ROOT}/data/routes_devtest.xml
export ROUTES_SUBSET=0
export REPETITIONS=1
#!/bin/bash

export TEAM_AGENT=${LEADERBOARD_ROOT}/leaderboard/autoagents/human_agent.py
export ROUTES=${LEADERBOARD_ROOT}/data/routes_devtest.xml
export ROUTES_SUBSET=0
export REPETITIONS=1

export DEBUG_CHALLENGE=1
export CHALLENGE_TRACK_CODENAME=SENSORS
export CHECKPOINT_ENDPOINT="${LEADERBOARD_ROOT}/results.json"
export RECORD_PATH=
export RESUME=

#!/bin/bash

python3 ${LEADERBOARD_ROOT}/leaderboard/leaderboard_evaluator.py \
--routes=${ROUTES} \
--routes-subset=${ROUTES_SUBSET} \
--repetitions=${REPETITIONS} \
--track=${CHALLENGE_TRACK_CODENAME} \
--checkpoint=${CHECKPOINT_ENDPOINT} \
--debug-checkpoint=${DEBUG_CHECKPOINT_ENDPOINT} \
--agent=${TEAM_AGENT} \
--agent-config=${TEAM_CONFIG} \
--debug=${DEBUG_CHALLENGE} \
--record=${RECORD_PATH} \
--resume=${RESUME}
```


- (4) 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。
- (5) 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](#)

二. 下位机--自动驾驶 DORA 平台软件系统使用说明

1. DORA介绍

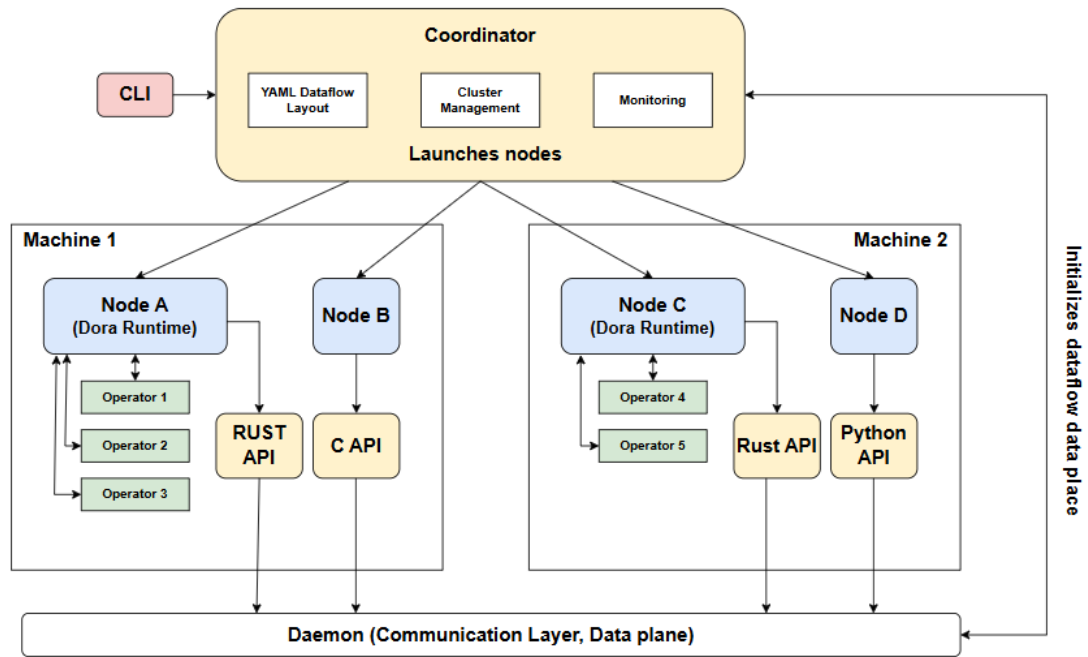
1.1DORA简介

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。

```
nodes:
- id: camera
  path: opencv-video-capture
  inputs:
    tick: dora/timer/millis/20
  outputs:
    - image
    -
- id: plot
  path: opencv-plot
  inputs:
    image: camera/image
```

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>)

2. 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

（2）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"

（3）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&ut

[m_medium=distribute.pc_search_result.none-ta](#)

3. 节点创建以及启动

3.1 Dora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"

2. 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

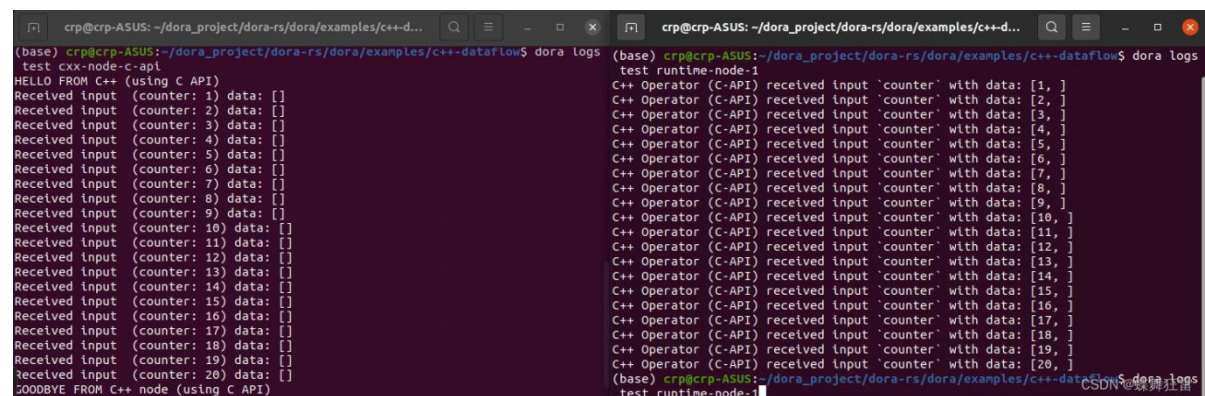
#打开一个命令窗

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



```
(base) crp@crp-ASUS: ~/dora_project/dora-rs/dora/examples/c++-dataflow$ dora logs test cxx-node-c-api
HELLO FROM C++ (using C API)
Received input (counter: 1) data: []
Received input (counter: 2) data: []
Received input (counter: 3) data: []
Received input (counter: 4) data: []
Received input (counter: 5) data: []
Received input (counter: 6) data: []
Received input (counter: 7) data: []
Received input (counter: 8) data: []
Received input (counter: 9) data: []
Received input (counter: 10) data: []
Received input (counter: 11) data: []
Received input (counter: 12) data: []
Received input (counter: 13) data: []
Received input (counter: 14) data: []
Received input (counter: 15) data: []
Received input (counter: 16) data: []
Received input (counter: 17) data: []
Received input (counter: 18) data: []
Received input (counter: 19) data: []
Received input (counter: 20) data: []
GOODBYE FROM C++ node (using C API)

(base) crp@crp-ASUS: ~/dora_project/dora-rs/dora/examples/c++-dataflow$ dora logs test runtime-node-1
C++ Operator (C-API) received input 'counter' with data: [1, ]
C++ Operator (C-API) received input 'counter' with data: [2, ]
C++ Operator (C-API) received input 'counter' with data: [3, ]
C++ Operator (C-API) received input 'counter' with data: [4, ]
C++ Operator (C-API) received input 'counter' with data: [5, ]
C++ Operator (C-API) received input 'counter' with data: [6, ]
C++ Operator (C-API) received input 'counter' with data: [7, ]
C++ Operator (C-API) received input 'counter' with data: [8, ]
C++ Operator (C-API) received input 'counter' with data: [9, ]
C++ Operator (C-API) received input 'counter' with data: [10, ]
C++ Operator (C-API) received input 'counter' with data: [11, ]
C++ Operator (C-API) received input 'counter' with data: [12, ]
C++ Operator (C-API) received input 'counter' with data: [13, ]
C++ Operator (C-API) received input 'counter' with data: [14, ]
C++ Operator (C-API) received input 'counter' with data: [15, ]
C++ Operator (C-API) received input 'counter' with data: [16, ]
C++ Operator (C-API) received input 'counter' with data: [17, ]
C++ Operator (C-API) received input 'counter' with data: [18, ]
C++ Operator (C-API) received input 'counter' with data: [19, ]
C++ Operator (C-API) received input 'counter' with data: [20, ]
```

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)

4. 各节点功能说明

4.6接收节点

```
- id: receive
  custom:
    source: dora-hardware/vendors/carla_bridge/receiver/build/one
    #source: ../../dora-hardware/vendors/gnss/CGI_611/drive_dora
    inputs:
      tick: dora/timer/millis/20
    # queue_size: 1000
    outputs:
      - DoraNavSatFix
      - DoraQuaternionStamped
      - pointcloud
  env:
    DEVICE_INDEX: 0
```

功能：接收上位机发送的GPS,IMU以及点云信息

4.7处理GNSS节点

```
- id: gnss_poser
  custom:
    source: sensing/gnss_poser/build/gnss_poser_core
    inputs:
      # tick: dora/timer/millis/50
      DoraNavSatFix: receive/DoraNavSatFix
    outputs:
      - DoraSentence
      - DoraNavSatFix
      - DoraQuaternionStamped
      - DoraTwistStamped

- id: gnss_ekf
  custom:
    source: localization/ekf_localizer/build/gnss_ekf
    inputs:
      DoraNavSatFix: gnss_poser/DoraNavSatFix
      DoraQuaternionStamped: receive/DoraQuaternionStamped
      # tick: dora/timer/millis/50
      # queue_size: 500
    outputs:
      - DoraGnssPose
```

功能：处理GNSS信息，坐标及位姿转换

4.8Map节点

```
- id: pub_road # 发布地图轨迹
  custom:
    source: sensing/gnss_poser/pubroad
    inputs:
      tick: dora/timer/millis/20
    outputs:
      - road_lane

- id: road_lane_publisher_node # 获取自身位姿
  custom:
    source: map/road_line_publisher/build/road_lane_publisher_node
    inputs:
      # tick: dora/timer/millis/50
      road_lane: pub_road/road_lane
      DoraGnssPose: gnss_ekf/DoraGnssPose
    outputs:
      - cur_pose_all # 输出频率 2ms /20ms
```

功能：发布地图轨迹及计算在地图下的自身位姿

4.9 Planning节点

```
- id: task_pub_node
  custom:
    source: planning/mission_planning/task_pub/build/task_pub_node
    inputs:
      tick: dora/timer/millis/20
    outputs:
      - task_exc_service
      - road_attri_msg

- id: planning #局部路径
  custom:
    source: planning/routing_planning/build/routing_planning_node
    inputs:
      # tick: dora/timer/millis/50
      road_lane: pub_road/road_lane #全局地图
      cur_pose_all: road_lane_publisher_node/cur_pose_all
      road_attri_msg: task_pub_node/road_attri_msg
      # SetSpeed_service: task_exc_node/SetSpeed_service
      # routing_service: task_exc_node/routing_service
      # SetStop_service: aeb_node/SetStop_service
      # VehicleStat: control/VehicleStat
    outputs:
      - raw_path #频率更新慢，不定时长 1ms - 5s
      - Request
```

功能：任务发布及路径规划

4.10 雷达处理节点

```
- id: ground_filter
  custom:
    source: peception/patchwork-plusplus-ros-master/build/demo
    inputs:
      pointcloud: receive/pointcloud
    outputs:
      - pointcloud_no_ground

- id: euclidean_cluster
  custom:
    source: peception/euclidean_cluster/build/euclidean_cluster
    queue_size: 3
    inputs:
      pointcloud: ground_filter/pointcloud_no_ground
    outputs:
      - LidarRawObject
      - rgb_pointcloud
```

功能：处理雷达点云信息

4.11 控制和发送节点

```
- id: lon_control
  custom:
    source: control/vehicle_control/lon_controller/build/lon_controller_node
    inputs:
      # tick: dora/timer/millis/50
      Request: planning/Request
    outputs:
      - TrqBreCmd

- id: latcontrol
  custom:
    source: control/vehicle_control/lat_controller/build/lat_controller_node
    inputs:
      # tick: dora/timer/millis/50
      # VehicleStat: control/VehicleStat
      cur_pose_all: road_lane_publisher_node/cur_pose_all
      raw_path: planning/raw_path
    outputs:
      - SteeringCmd #100hz

- id: ctrl_trans
  custom:
    source: control/vehicle_control/ctrl_trans/build/ctrl_trans
    inputs:
      # tick: dora/timer/millis/100
      SteeringCmd: latcontrol/SteeringCmd
      TrqBreCmd: lon_control/TrqBreCmd
```

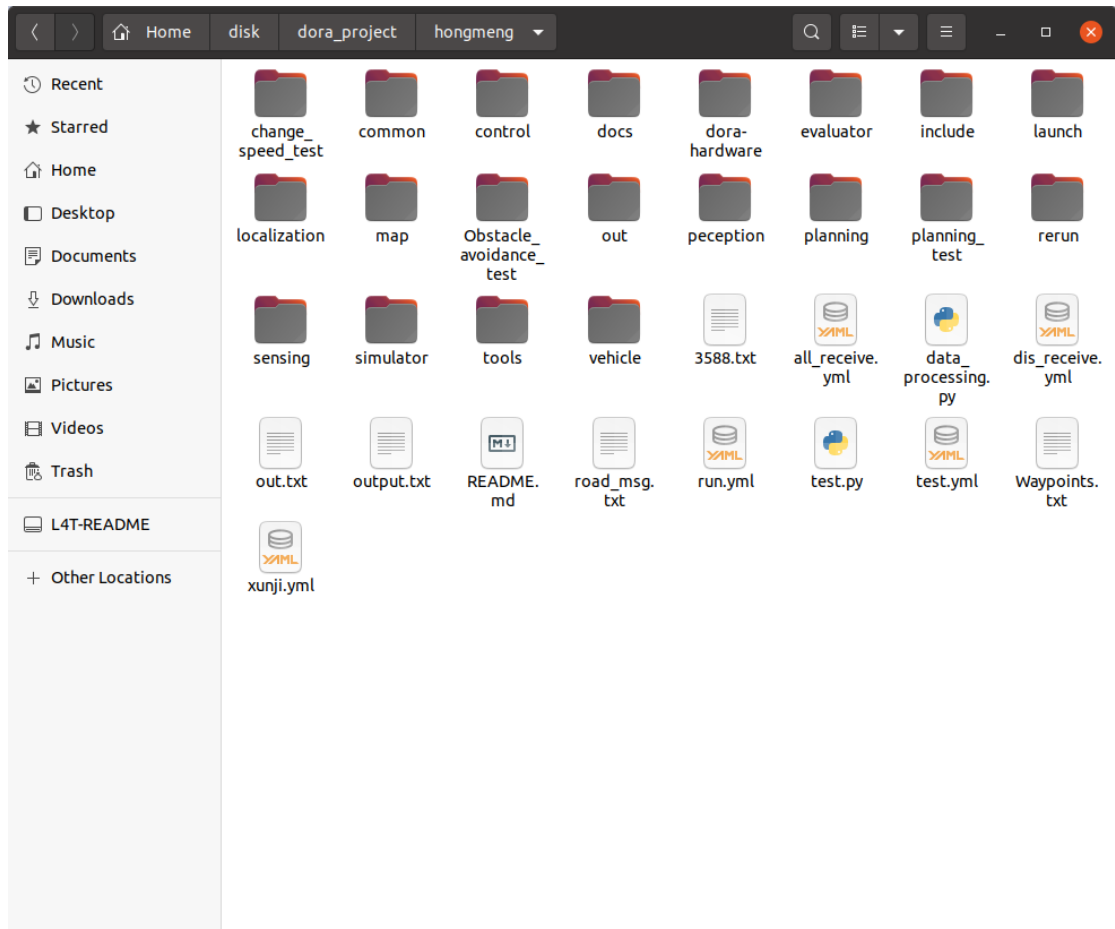
功能：横向控制和纵向控制以及向上位机发送控制信息

4.12 可视化节点

```
- id: rerun
  custom:
    source: rerun/build/to_rerun
    inputs:
      pointcloud: receive/pointcloud
      raw_path: planning/raw_path
      DoraGnssPose: gnss_ekf/DoraGnssPose
```

功能：可视化路径与点云信息

5. 程序运行



在yml所在目录打开终端运行，运行方法见3.1.3

```
nano@nano-desktop: ~/disk/dora_project/hongmeng
nano@nano-desktop: ~/disk/dora_project/hongmeng
Processing input event, point cloud size: 58102
Output sent successfully with id: pointcloud, length: 929648, point cloud size: 58102
(base) nano@nano-desktop:~/disk/dora_project/hongmeng$ dora logs test gnss
rsllidar driver for dora (TCP and UDP version)
Starting UDP receiver on port 12345
Dora context initialized successfully.
TCP receiver started, listening on port 5005
Received data length: 348612
Parsed point cloud with 58102 points.
Received a stuffed point cloud message with 58102 points and pushed it to the queue.
Processing input event, point cloud size: 58102
Output sent successfully with id: pointcloud, length: 929648, point cloud size: 58102
(base) nano@nano-desktop:~/disk/dora_project/hongmeng$
(base) nano@nano-desktop:~/disk/dora_project/hongmeng$ dora logs test ground_filter
patchwork-plusplus for dora
Operating patchwork++...
Inititalizing PatchWork++...Sensor Height: 1.723000Num of Iteration: 3Num of LPR: 20Num of min. points: 10
Seeds Threshold: 0.300000Distance Threshold: 0.125000Max. range:: 80.000000Min. range:: 2.700000Normal vec
tor threshold: 0.707000adaptive_seed_selection_margin: -1.200000Num. zones: 4Num. sectors: 16, 32, 54, 32
Num. rings: 2, 4, 4, 4
elevation_thr_: 0.0000, 0.0000, 0.0000, 0.0000
flatness_thr_: 0.0000, 0.0000, 0.0000, 0.0000
INITIALIZATION COMPLETE
[c node] dora context initialized
Received input point cloud - Seq: 0, Stamp: 0, Point count: 58102
Time taken : 0.022305965423583984375
Time taken to sort: 0
Time taken to pca : 0.004714488983154296875
Time taken to estimate: 0.0024530887603759765625
Time taken to Revert: 9.5367431640625e-07
Time taken to update : 4.0531158447265625e-06
Sending output point cloud - Seq: 0, Stamp: 0, Point count: 1516
(base) nano@nano-desktop:~/disk/dora_project/hongmeng$ dora logs test euclidean_cluster
euclidean_cluster for dora
euclidean_cluster: [c node] dora context initialized
Received input pointcloud data. Seq: 0, Stamp: 0, Point count: 1516
Point cloud clustering completed. Cluster count: 1
Sending LidarRawObject data. Object count: 1
Sending rgb_pointcloud data. Point count: 1516
Successfully sent pointcloud data. Point count: 1516
(base) nano@nano-desktop:~/disk/dora_project/hongmeng$ dora logs test aeb_node
AEB_NODE
AEB Activated! Threat detected! Object x_pos: 1.52329
(base) nano@nano-desktop:~/disk/dora_project/hongmeng$
```

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

1. 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。

2. 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

3. 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

3.1.2 GPS 数据处理逻辑：

（1）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`。

(2) 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`)、角速度 (`gx`, `gy`, `gz`)、朝向 heading (磁力计获取)。

(2) 数据发送

通过 UDP 发送至`UDP_IP:UDP_PORT_GNSS_IMU`。

3.3 LiDAR 模块

3.3.1 LiDAR 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

3.3.2 LiDAR 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

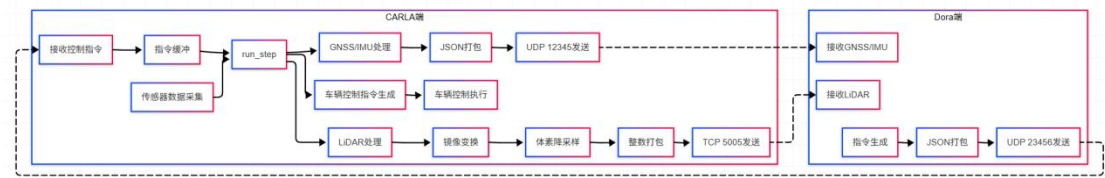
在`run_step`函数中，将指令应用于`VehicleControl`对象，控制车辆行驶。

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

参数名	描述	默认值	备注
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



4. 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：`./dora.sh`，等待传感器初始化及网络连接。
- (3) 启动 Dora 端

5. 测试 Demo 要点与说明

- (1) 地图配置调整：在本项目中，存在三个重要的场景文件，分别是 `AEB_Scenario.xml`、`Obstacle_Avoidance_Scenario.xml` 以及 `Pure_Tracking_Scenario.xml`。其中，`AEB_Scenario.xml` 代表自动紧急制动（AEB）场景；`Obstacle_Avoidance_Scenario.xml` 对应避障场景；而 `Pure_Tracking_Scenario.xml` 则表示纯跟踪场景。具体操作是在本 Demo 中修改 `~/leaderboard/data/` 目录下的相关 `.xml` 文件。
- (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) 参考路径说明：用于测试的三个场景（自动紧急制动（AEB）场景、避障场景、纯跟踪场景）所对应的路径段文件，均存储在 `~/leaderboard/path` 路径下。每个场景的路径文件包含了全球导航卫星系统（GNSS）行驶路径，以及基于东

北天坐标系（ENU）的坐标信息。