Introduction to CARLA

Junwon Sung
Junghwan Yim

University at Buffalo
School of Engineering and Applied Sciences

CARLA

- **CARLA** is an open-source autonomous driving simulator. It was built from scratch to serve as a modular and flexible API to address a range of tasks involved in the problem of autonomous driving.

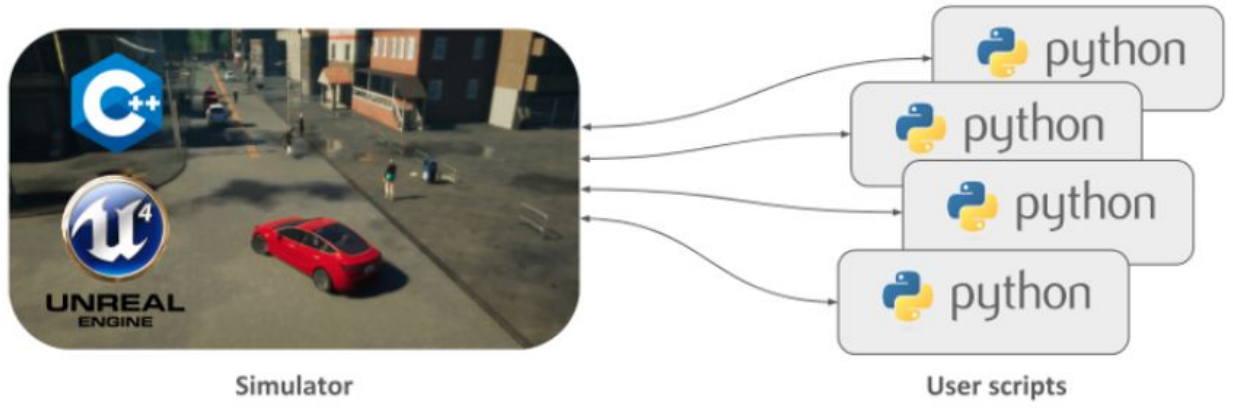
- **CARLA** is grounded on Unreal Engine to run the simulation and uses the ASAM OpenDRIVE standard (1.4 as today) to define roads and urban settings

- The simulator has to meet the requirements of different use cases within the general problem of driving (e.g. learning driving policies, training perception algorithms, etc.).

CARLA

CARLA simulator

- The **CARLA simulator** consists of a scalable client-server architecture.
- The server is responsible for everything related with the simulation it self (sensor rendering, computation of physics, updates on the world-state and its actors and much more)
- The CARLA simulator recommended system
  - Intel i7 gen 9th - 11th / Intel i9 gen 9th - 11th / ADM ryzen 7 / ADM ryzen 9
  - +16 GB RAM memory
  - NVIDIA RTX 2070 / 2080 / 3070 / 3080
  - Ubuntu 18.04



CARLA simulator

- Traffic manager: A built-in system that takes control of the vehicles besides the one used for learning. It acts as a conductor provided by CARLA to recreate urban-like environments with realistic behaviours.

- Sensors: Vehicles rely on them to dispense information of their surroundings. In CARLA they are a specific kind of actor attached the vehicle and the data they receive can be retrieved and stored to ease the process. Currently the project supports different types of these, from cameras to radars, lidar and many more.

- Recorder: This feature is used to reenact a simulation step by step for every actor in the world. It grants access to any moment in the timeline anywhere in the world, making for a great tracing tool.

CARLA simulator

- ROS bridge and Autoware implementation: As a matter of universalization, the CARLA project ties knots and works for the integration of the simulator within other learning environments.

- Open assets: CARLA facilitates different maps for urban settings with control over weather conditions and a blueprint library with a wide set of actors to be used. However, these elements can be customized and new can be generated following simple guidelines.

- Scenario runner: In order to ease the learning process for vehicles, CARLA provides a series of routes describing different situations to iterate on. These also set the basis for the CARLA challenge, open for everybody to test their solutions and make it to the leaderboard.

CARLA Core concepts

- First steps
  - 1st - World and Client
  - 2nd - Actors and Blueprints
  - 3rd - Maps and Navigation
  - 4th - Sensors and Data

- Advanced steps
  - CARLA offers a wide range of features that go beyond the scope of this introduction to the simulator

First Steps [1st-World and client]

- **The Client** is the module the user runs to ask for information or changes in the simulation. A client runs with an IP and a specific port. It communicates with the server via terminal. There can be many clients running at the same time. Advanced multi client managing requires thorough understanding of CARLA and synchrony.

- **The World** is an object representing the simulation. It acts as an abstract layer containing the main methods to spawn actors, change the weather, get the current state of the world, etc. There is only one world per simulation. It will be destroyed and substituted for a new one when the map is changed.

First Steps [2nd-Actors and Blueprints]

- An actor is anything that plays a role in the simulation.
  - Vehicles
  - Walkers
  - Sensors
  - The spectator
  - Traffic signs and traffic lights.

- **Blueprints** are already-made actor layouts necessary to spawn an actor. Basically, models with animations and a set of attributes. Some of these attributes can be customized by the user, others don't. There is a **Blueprint library** containing all the blueprints available as well as information on them.

First Steps [3rd-Maps and Navigation]

- **The map** is the object representing the simulated world, the town mostly. There are eight maps available. All of them use OpenDRIVE 1.4 standard to describe the roads.

- **Roads, lanes and junctions** are managed by the Python API to be accessed from the client. These are used along with the **waypoint** class to provide vehicles with a navigation path.

- **Traffic signs** and **traffic lights** are accessible as **carla.Landmark** objects that contain information about their OpenDRIVE definition. Additionally, the simulator automatically generates stops, yields and traffic light objects when running using the information on the OpenDRIVE file. These have bounding boxes placed on the road. Vehicles become aware of them once inside their bounding box.

First Steps [4th-Sensors and Data]

- **Sensors** wait for some event to happen, and then gather data from the simulation. They call for a function defining how to manage the data. Depending on which, sensors retrieve different types of **sensor data**.

- A sensor is an actor attached to a parent vehicle. It follows the vehicle around, gathering information of the surroundings. The sensors available are defined by their blueprints in the Blueprint library.

  - Cameras (RGB, depth and semantic segmentation)
  - Collision detector
  - Gnss sensor
  - IMU sensor
  - Lidar raycast
  - Lane invasion detector
  - Obstacle detector
  - Radar
  - RSS

Advanced steps

- **Traffic Simulation**

  supporting the co-simulation with below simulators :
  - **SUMO**
  - **PTV-Vissim co-simulation**

  support traffic scenario generated from :
  - **OpenScenario**
  - **Scenario Runner**

  Support file composed with program language for generating traffic scenario :
  - **Senics**

Handled by **Traffic manager in Carla**

Advanced steps

- **Customizing Map**

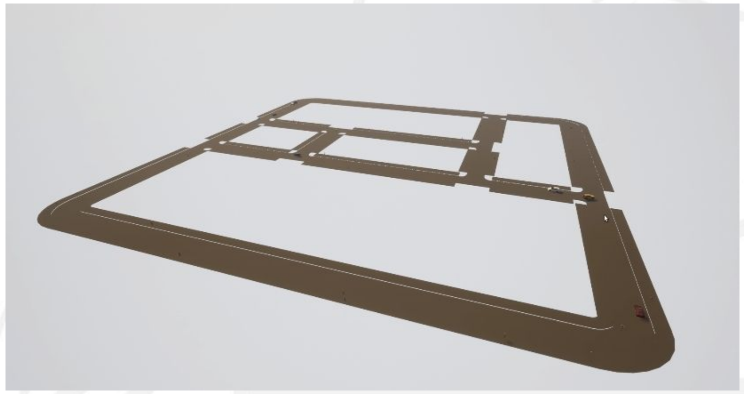
1. Get OpenDRIVE file containing the load shape
    - It could be directly ingest to Carla

```
python3 config.py -x opendrive/TownBig.xodr
```

The OpenDrive file could be get from webpage : https://www.openstreetmap.org

2. Import **OpenDrive** file to **RoadRunner**

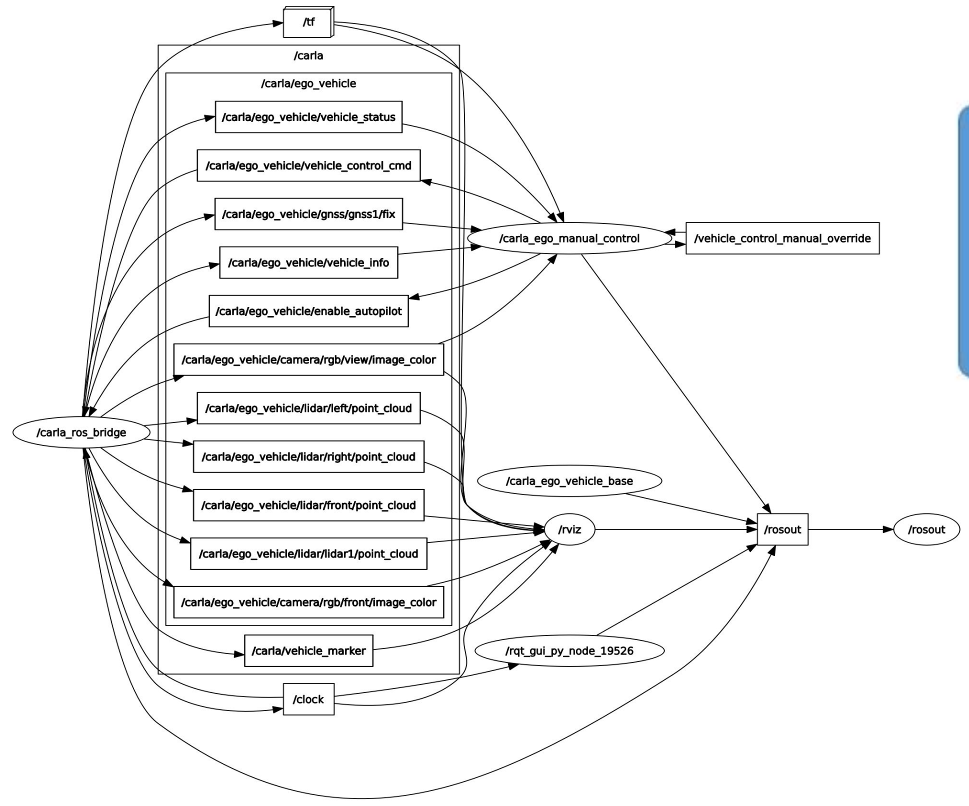
3. import **RoadRunner** file to **Unreal Engine**
    - Adding traffic sign and light, building, crosswalk etc are available.

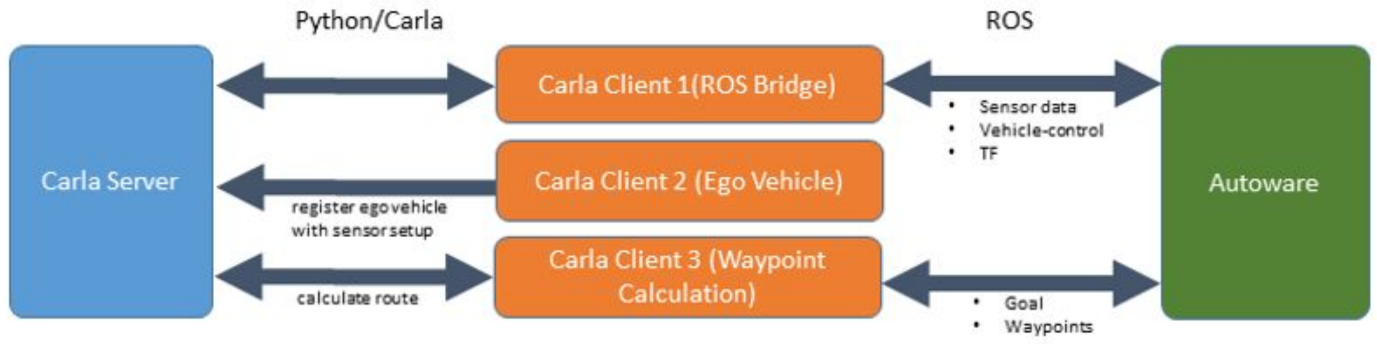


Advanced steps

https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_sensors/

- ROS Bridge


/tf
/carla
/carla/ego_vehicle
/carla/ego_vehicle/vehicle_status
/carla/ego_vehicle/vehicle_control_cmd
/carla/ego_vehicle/gnss/gnss1/fix
/carla/ego_vehicle/vehicle_info
/carla/ego_vehicle/enable_autopilot
/carla/ego_vehicle/camera/rgb/view/image_color
/carla/ego_vehicle/lidar/left/point_cloud
/carla/ego_vehicle/lidar/right/point_cloud
/carla/ego_vehicle/lidar/front/point_cloud
/carla/ego_vehicle/lidar/lidar1/point_cloud
/carla/ego_vehicle/camera/rgb/front/image_color
/carla/vehicle_marker
/clock
/carla_ros_bridge
/carla_ego_manual_control
/vehicle_control_manual_override
/carla_ego_vehicle_base
/rviz
/rosout
/rosout
/rqt_gui_py_node_19526


Python/Carla
ROS
Carla Server
Carla Client 1(ROS Bridge)
Carla Client 2 (Ego Vehicle)
Carla Client 3 (Waypoint Calculation)
Autoware
register egovehicle with sensorsetup
calculate route
Sensor data
Vehicle-control
TF
Goal
Waypoints

Thank you