# CARLA Simulator

Open-source simulator for autonomous driving research

CARLA Team

December 3, 2020

# Contents


## 0.1 ScenarioRunner

ScenarioRunner is a module that allows traffic scenario definition and execution for the CARLA simulator. The scenarios can be defined through a Python interface or using the OpenSCENARIO standard.

ScenarioRunner can also be used to prepare AD agents for their evaluation, by easily creating complex traffic scenarios and routes for the agents to navigate through. These results can be validated and shared in the CARLA Leaderboard, an open platform for the community to fairly compare their progress, evaluating agents in realistic traffic situatoins.

The CARLA forum has a specific section regarding ScenarioRunner, for users to post any doubts or suggestions that may arise during the reading of this documentation.

### 0.1.1 Quick start

**Get ScenarioRunner** — Tutorial on how to download and launch ScenarioRunner.

**First steps** — Brief tutorials on how to run different types of scenarios.

**Create a new scenario** — Tutorial on how to create a new scenario using ScenarioRunner.

**Metrics module** — Explanation of the metrics module.

**F.A.Q.** — Some of the most frequent installation issues.

**Release notes** — Features, fixes and other changes listed per release.

### 0.1.2 References

**List of scenarios** — Example scenarios available in ScenarioRunner.

**OpenScenario support** — Support status of OpenSCENARIO features.

### 0.1.3 Contributing

**Code of conduct** — Standard rights and duties for contributors.

**Coding standard** — Guidelines to write proper code.

**Contribution guidelines** — The different ways to contribute to ScenarioRunner.

# Chapter 1

# Quick start

## 1.1 Get ScenarioRunner

This tutorial explains how to download ScenarioRunner and run a simple example to test it. ScenarioRunner needs CARLA in order to run, and must match the CARLA version being used. If the CARLA being used is a build from source, download ScenarioRunner from source. If the CARLA being used is a package, download the corresponding version of ScenarioRunner.



### 1.1.1 Installation summary

Show command line summary for the quick start installation

```
# Decide whether to use a package or make the build from source


# Option A) Use a ScenarioRunner package
   # 1. Install a CARLA package:
      https://carla.readthedocs.io/en/latest/start_quickstart/
   # 2. Download the matching ScenarioRunner package:
      https://github.com/carla-simulator/scenario_runner/releases
   # 3. Extract the content wherever needed.

   # Update the release:
   # 1. Delete previous CARLA and ScenarioRunner versions.
   # 2. Download the latest CARLA release.
   # 3. Download the matching ScenarioRunner release.


# Option B) Download ScenarioRunner from source
   # 1. Build CARLA from source:
      https://carla.readthedocs.io/en/latest/build_linux/
   # 2. Clone the ScenarioRunner repository:
git clone https://github.com/carla-simulator/scenario_runner.git
   # 3. Install requirements according to the Python version to be used:
   # For Python 2.x:
sudo apt remove python-networkx #if installed, remove old version of networkx
pip2 install --user -r requirements.txt
   # For Python 3.x:
sudo apt remove python3-networkx #if installed, remove old version of networkx
pip3 install --user -r requirements.txt

   # To update ScenarioRunner from source:
```

```
   # 1. Update CARLA:
      https://carla.readthedocs.io/en/latest/build_update/
   # 2. Go to the ScenarioRunner repository, master branch
cd ~/scenario_runner
git branch master
   # 3. Pull the latest changes from the repository
git pull
```

### 1.1.2 A. Download a ScenarioRunner release

The releases of ScenarioRunner are packages containing: * A ScenarioRunner version tied to a specific CARLA release. * A few example scenarios written in Python.

The process to run a ScenarioRunner release is quite straightforward.

**1. Download a CARLA release.** Follow the process in the CARLA quick start. **2. Download the matching ScenarioRunner release.** All of the releases are listed here.

Both versions have to match. If the CARLA release is *0.9.9*, use also ScenarioRunner *0.9.9*. [Here](https://github.com/carla-simulator/scenario runner) is a brief list of compatibilities between CARLA and ScenarioRunner.

**3. Extract the content.** The directory does not matter.

#### Update the release

The packaged version requires no updates. The content is bundled and thus, tied to a specific release of CARLA. Everytime there is a new CARLA release, there will be a matching one for ScenarioRunner. All the releases are listed here:

- CARLA releases
- Scenario Runner releases

To run the latest or any other release, delete the previous and install the one desired.

### 1.1.3 B. Download ScenarioRunner from source

The ScenarioRunner source repository contains the most experimental features that run with the latest development version of CARLA. It requires no build, as it only contains Python code for the ScenarioRunner module.

**1. Build CARLA from source.** Follow the docs to build on Linux or Windows.

ScenarioRunner needs CARLA to run, so the minimum requirements for CARLA stated in the docs are also necessary to run ScenarioRunner.

**2. Clone the ScenarioRunner repository.**

```
git clone https://github.com/carla-simulator/scenario_runner.git
```

**3. Install the requirements according to the Python version to be used.** First go to the main ScenarioRunner directory

```
cd ~/scenario_runner/
```

- **For Python 2.x.**

```
sudo apt remove python-networkx #if installed, remove old version of networkx
pip2 install --user -r requirements.txt
```

- **For Python 3.x**

```
sudo apt remove python3-networkx #if installed, remove old version of networkx
pip3 install --user -r requirements.txt
```

py-trees newer than v0.8 are not supported.

**Update from source**

**1. Update the CARLA build** Follow the docs to update CARLA.

**2. Go to the main ScenarioRunner directory.** Make sure to be in the local master branch.

```
cd ~/scenario_runner
git branch master
```

**3. Pull the latest changes from the repository.**

```
git pull
```

**4. Add environment variables and Python paths** These are necessary for the system to find CARLA, and add the PythonAPI to the Python path.

- **For Linux**

```
# ${CARLA_ROOT} is the CARLA installation directory
# ${SCENARIO_RUNNER} is the ScenarioRunner installation directory
# <VERSION> is the correct string for the Python version being used
# In a build from source, the .egg files may be in: ${CARLA_ROOT}/PythonAPI/dist/ instead of
    ${CARLA_ROOT}/PythonAPI
export CARLA_ROOT=/path/to/your/carla/installation
export SCENARIO_RUNNER_ROOT=/path/to/your/scenario/runner/installation
export PYTHONPATH=$PYTHONPATH:${CARLA_ROOT}/PythonAPI/carla/dist/carla-<VERSION>.egg
export PYTHONPATH=$PYTHONPATH:${CARLA_ROOT}/PythonAPI/carla/agents
export PYTHONPATH=$PYTHONPATH:${CARLA_ROOT}/PythonAPI/carla
export PYTHONPATH=$PYTHONPATH:${CARLA_ROOT}/PythonAPI
```

Change the command lines with the proper paths, according to the comments.

- **For Windows**

```
# %CARLA_ROOT% is the CARLA installation directory
# %SCENARIO_RUNNER% is the ScenarioRunner installation directory
# <VERSION> is the correct string for the Python version being used
# In a build from source, the .egg files may be in: ${CARLA_ROOT}/PythonAPI/dist/ instead of
    ${CARLA_ROOT}/PythonAPI
set CARLA_ROOT=\path\to\your\carla\installation
set SCENARIO_RUNNER_ROOT=\path\to\your\scenario\runner\installation
set PYTHONPATH=%PYTHONPATH%;%CARLA_ROOT%\PythonAPI\carla\dist\carla-<VERSION>.egg
set PYTHONPATH=%PYTHONPATH%;%CARLA_ROOT%\PythonAPI\carla\agents
set PYTHONPATH=%PYTHONPATH%;%CARLA_ROOT%\PythonAPI\carla
set PYTHONPATH=%PYTHONPATH%:%CARLA_ROOT%\PythonAPI
```

Change the command lines with the proper paths, according to the comments.

To permanently set the environment variables, go to *edit the environment variables of this account*. This can be quickly accessed by writing *env* on the Windows' search panel.

### 1.1.4 Run a test

Running the follow vehicle example First of all, you need to get latest master branch from CARLA. Then you have to include CARLA Python API to the Python path:

If working with builds from source, make sure to upload these. Download the latest content in the master branches.

**1. Run the CARLA server.**

- **A) In a build from source** go to the CARLA directory and launch the server in the editor.

```
cd ~/carla # Change the path accordingly
make launch
# Press Play in the UE Editor
```

- **B) In a CARLA package** run the server directly.

```
./CarlaUE4.sh
```

**2. Start an example scenario.** Open another terminal and go to the directory where ScenarioRunner is downloaded. For the sake of this test, the follow leading vehicle scenario will be used.

```
# Inside the ScenarioRunner root directory
python scenario_runner.py --scenario FollowLeadingVehicle_1 --reloadWorld
```

If using a Python 3.x version, run the command with 'python3'.

**3. Test the scenario with manual control.** Open a new terminal and run the `manual_control.py`. A new window should pop up, with an ego vehicle in the middle of the street. Move forward and the leading vehicle will appear.

```
# Inside the ScenarioRunner root directory
python manual_control.py
```

The scenarios have a timeout of one minute approximately, for the agent to be launched. If the timeout appears, the follow leading vehicle example should be launched again.

Run the 'manual control.py' found in the ScenarioRunner package/repository, not CARLA .

**4. Explore other options.** Run the Scenario Runner with the flag `--help` to explore other command line parameters and some basic descriptions. For example, to avoid automatically (re-)load the CARLA world, skip the command line option `--reloadWorld`.

```
python scenario_runner.py --help
```

Thus concludes the installation process for ScenarioRunner. In case any unexpected error or issue occurs, the CARLA forum is open to everybody. There is an *ScenarioRunner* category to post problems and doubts regarding this module.

## 1.2 Getting Started Tutorial

This tutorial refers to the latest versions of CARLA (at least 0.9.5)

Welcome to the ScenarioRunner for CARLA! This tutorial provides the basic steps for getting started using the ScenarioRunner for CARLA.

Download the latest release from our GitHub page and extract all the contents of the package in a folder of your choice.

The release package contains the following

- The ScenarioRunner for CARLA
- A few example scenarios written in Python.

### 1.2.1 Installing prerequisites

The current version is designed to be used with Ubuntu 16.04, Python 2.7 or Python 3.5. Depending on your Python version, execute:

```
#Python 2.x
sudo apt remove python-networkx #if installed, remove old version of networkx
pip2 install --user -r requirements.txt
#Python 3.x
sudo apt remove python3-networkx #if installed, remove old version of networkx
pip3 install --user -r requirements.txt
```

Note: py-trees newer than v0.8 is *NOT* supported.

### 1.2.2 Running the follow vehicle example

First of all, you need to get latest master branch from CARLA. Then you have to include CARLA Python API to the Python path:

```
export CARLA_ROOT=/path/to/your/carla/installation
export
    PYTHONPATH=$PYTHONPATH:${CARLA_ROOT}/PythonAPI/carla/dist/carla-<VERSION>.egg:${CARLA_ROOT}/PythonAPI/carla
```

NOTE: ${CARLA_ROOT} needs to be replaced with your CARLA installation directory, and needs to be replaced with the correct string. If you build CARLA from source, the egg files maybe located in: ${CARLA_ROOT}/PythonAPI/dist/ instead of ${CARLA_ROOT}/PythonAPI.

Now, you can start the CARLA server from ${CARLA_ROOT}

```
./CarlaUE4.sh
```

Start the example scenario (follow a leading vehicle) in an extra terminal:

```
python scenario_runner.py --scenario FollowLeadingVehicle_1 --reloadWorld
```

If you require help or want to explore other command line parameters, start the scenario runner as follows:

```
python scenario_runner.py --help
```

To control the ego vehicle within the scenario, open another terminal and run:

```
python manual_control.py
```

Note: If you do not wish to automatically (re-)load the CARLA world, you can skip the command line option *–reloadWorld*

### 1.2.3 Running all scenarios of one scenario class

Similar to the previous example, it is also possible to execute a sequence of scenarios, that belong to the same class, e.g. the “FollowLeadingVehicle” class.

The only difference is, that you start the scenario_runner as follows:

```
python scenario_runner.py --scenario group:FollowLeadingVehicle
```

### 1.2.4 Running other scenarios

A list of supported scenarios is provided in List of Supported Scenarios. Please note that different scenarios may take place in different CARLA towns. This has to be respected when launching the CARLA server.

### 1.2.5 Running scenarios using the OpenSCENARIO format

To run a scenario, which is based on the OpenSCENARIO format, please run the ScenarioRunner as follows:

```
python scenario_runner.py --openscenario <path/to/xosc-file>
```

Please note that the OpenSCENARIO support and the OpenSCENARIO format itself are still work in progress. More information you can find inOpenSCENARIO support

### 1.2.6 Running route-based scenario (similar to the CARLA AD Challenge)

To run a route-based scenario, please run the ScenarioRunner as follows:

```
python scenario_runner.py --route <path/to/route-file> <path/to/scenario_sample_file> [route id]
    --agent <path/to/agent_file>
```

Example:

```
python scenario_runner.py /scenario_runner/srunner/routes_debug.xml
    /scenario_runner/srunner/data/all_towns_traffic_scenarios1_3_4.json 0 --agent
    srunner/autoagents/npc_agent.py
```

If no route id is provided, all routes within the given file will be executed.

By doing so, ScenarioRunner will match the scenarios to the route, and they'll activate when the ego vehicle is nearby. However, routes need an autonomous agent to control the ego vehicle. Several examples are provided in srunner/autoagents/. For more information about agents, please have a look into theagent documentation

## 1.3 Create a new scenario tutorial

This tutorial describes how you can create and run a new scenario using the ScenarioRunner and the ScenarioManager suite.

Let us call the new scenario *NewScenario*. To create it, there are only few steps required.

### 1.3.1 Creating an empty Python class

Go to the Scenarios folder and create a new Python class with the name *NewScenario* in a new Python file (*new_scenario.py*). The class should be derived from the *BasicScenario* class. As a result, the class should look as follows:

```
class NewScenario(BasicScenario):
    """
    Some documentation on NewScenario
    :param world is the CARLA world
    :param ego_vehicles is a list of ego vehicles for this scenario
    :param config is the scenario configuration (ScenarioConfiguration)
    :param randomize can be used to select parameters randomly (optional, default=False)
    :param debug_mode can be used to provide more comprehensive console output (optional,
        default=False)
    :param criteria_enable can be used to disable/enable scenario evaluation based on test criteria
        (optional, default=True)
    :param timeout is the overall scenario timeout (optional, default=60 seconds)
    """


    # some ego vehicle parameters
    # some parameters for the other vehicles


    def __init__(self, world, ego_vehicles, config, randomize=False, debug_mode=False,
        criteria_enable=True,
                 timeout=60):
        """
        Initialize all parameters required for NewScenario
        """


        # Call constructor of BasicScenario
        super(NewScenario, self).__init__(
          name="NewScenario",
          ego_vehicles,
          config,
          world,
          debug_mode,
```

```
        criteria_enable=criteria_enable))



    def create_behavior(self):
        """
        Setup the behavior for NewScenario
        """


    def create_test_criteria(self):
        """
        Setup the evaluation criteria for NewScenario
        """
```

### 1.3.2 Filling the Python class

In the NewScenario class, you have to define the following methods mentioned in the code example.

#### Initialize Method

The initialize method is intended to setup all parameters required for the scenario and all vehicles. This includes selecting the correct vehicles, spawning them at the correct location, etc. To simplify this, you may want to use the *setup_vehicle()* function defined in basic_scenario.py

#### CreateBehavior method

This method should setup the behavior tree that contains the behavior of all non-ego vehicles during the scenario. The behavior tree should use py_trees and the atomic behaviors defined in *atomic_scenario_behavior.py*

#### CreateTestCriteria method

This method should setup a list with all evaluation criteria for the scenario. The criteria should be based on the atomic criteria defined in *atomic_scenario_criteria.py*.

Note: From this list a parallel py_tree will be created automatically!

### 1.3.3 Adding the scenario configuration

Finally the scenario configuration should be added to the examples/ folder. If you extend an already existing scenario module, you can simply extend the corresponding XML, otherwise add a new XML file. In this case you can use any of the existing XML files as blueprint.

If you want to add multiple ego vehicles for a scenario, make sure that they use different role names, e.g.

```
    <scenario name="MultiEgoTown03" type="FreeRide" town="Town03">
        <ego_vehicle x="207" y="59" z="0" yaw="180" model="vehicle.lincoln.mkz2017" rolename="hero"/>
        <ego_vehicle x="237" y="-95.0754252474" z="0" yaw="90" model="vehicle.tesla.model3"
            rolename="hero2"/>
    </scenario>
```

## 1.4 Metrics module

The metrics module relies on the CARLA recorder to facilitate the easy calculus and monitoring of any kind of parameter. After the scenario is finished, the recorder stores the simulation information. Users can then define their own metrics, and check the corresponding results with no need to play the simulation again and again.

This section covers an explanation of the different elements that form the module, a step by step tutorial of how to create a new metric, and a reference of the functions that query the recording and define the metrics.

***Structure of the module*** ___How to use the metrics module___ *1. Record a scenario* 2. Define the metrics *3. Run the metrics manager* ___Recording queries reference___ *Generic actor data* Generic simulation data *Actor accelerations* Actor angular velocities *Actor controls* Actor transforms *Actor velocities* Scene lights *Traffic lights* Vehicle lights

### 1.4.1 Structure of the module

Similarly to the main ScenarioRunner module, the Metrics module is run using a main script, `metrics_manager.py`, and the rest of the information is contained inside a folder structure. Here is a brief introduction to the main script.

- `metrics_manager.py` — The main script of the module. Run this to show the results of the set of metrics. The script has the usual `host` and `port` arguments, and some more to set the metrics and recording to be used.
  - `host` *(string)* – IP address where a CARLA simulation is running. Default is `(127.0.0.1)`.
  - `port` *(int)* – TCP port where the CARLA simulation is running. Default are `2000` and `2001`.
  - `metrics` — Path to the metrics to be used.
  - `log` — Path to the `.log` file containing the recording (relative to the environment variable `SCENARIO_RUNNER_ROOT`).
  - `criteria` *(optional)* — Path to a JSON file with the criteria of the scenario.

The rest of the elements that shape the module can be found in the `srunner/metrics` folder. These folder has been divided in three subfolders.

- `srunner/metrics/data` — Stores information about the scenarios. By default, it has six files, which are part of the examples metric.
- `srunner/metrics/examples` — Contains some example metrics, and the metric containing the base class `BaseMetric` to create new metrics.
  - `basic_metric.py` – Contains the base class `BaseMetric`. All the metrics are inherited from it.
  - `criteria_filter.py` – Returns a JSON with the most important attributes of the criteria.
  - `distance_between_vehicles.py` – Returns the distance betwen two vehicles. Useful to show how to query the recording.
  - `distance_to_lane_center.py` – Calculates the distance between the vehicle location and the center of the lane. Useful to show how to access the map API information..
- `srunner/metrics/tools` — Contains two key scripts that allow to query the recording.
  - `metrics_parser.py` – Transforms the string provided by the recording to a dictionary.
  - `metrics_log.py` – Provides with several functions to query the dictionary created with `metrics_parser.py`. These functions are the easiest way to access information of a scenario. They listed in areference in the last segment of this page.

### 1.4.2 How to use the metrics module

#### 1. Record a scenario

The metrics module needs for a recording of a simulation in order to work. Otherwise, it has no data to make the calculations of the metrics.

Use the `record` argument. Add the path where the information should be saved should be saved. The path must be relative to the environment variable `SCENARIO_RUNNER_ROOT`.

```
python scenario_runner.py --scenario <scenario_name> --record <path/to/save/the/recorder/file>
```

By recording the scenario two files will be created in the desired path. These files will later be used as the `log` and `criteria` arguments in the `metrics_manager.py`. **1. A CARLA recording** *(`.log`)* — Contains simulation data per frame. To know more about this, read the recorder docs. **2. A criteria file** *(.json)* — The criteria of the scenario parsed as a dictionary, and stored in a JSON file. The keys of the dictionary are the names of the criteria. The values are the attributes of each criteria.

Only the JSON serializable attributes will be parsed, the rest will be ignored.

By default, both files are named after the scenario that is run. If the scenario is named `example.py`, the files will be `example.log` and `example.json`.

#### 2. Define the metrics

It is time to create the desired metrics that will be used for the scenario. The possibilities are endless, but for the sake of this tutorial, the metric described in `srunner/metrics/examples/distance_between_vehicles.py` will be used as an example. Let's dig a little bit into the code itself.

```
class DistanceBetweenVehicles(BasicMetric):
"""
```

```
Metric class DistanceBetweenVehicles
"""


def _create_metric(self, town_map, log, criteria):
    """
    Implementation of the metric. This is an example to show how to use the recorder,
    accessed via the log.
    """


    # Get the ID of the two vehicles
    ego_id = log.get_ego_vehicle_id()
    adv_id = log.get_actor_ids_with_role_name("scenario")[0]  # Could have also used its type_id


    dist_list = []
    frames_list = []


    # Get the frames both actors were alive
    start_ego, end_ego = log.get_actor_alive_frames(ego_id)
    start_adv, end_adv = log.get_actor_alive_frames(adv_id)
    start = max(start_ego, start_adv)
    end = min(end_ego, end_adv)


    # Get the distance between the two
    for i in range(start, end):


        # Get the transforms
        ego_location = log.get_actor_transform(ego_id, i).location
        adv_location = log.get_actor_transform(adv_id, i).location


        # Filter some points for a better graph
        if adv_location.z < -10:
            continue


        dist_v = ego_location - adv_location
        dist = math.sqrt(dist_v.x * dist_v.x + dist_v.y * dist_v.y + dist_v.z * dist_v.z)


        dist_list.append(dist)
        frames_list.append(i)


    # Use matplotlib to show the results
    plt.plot(frames_list, dist_list)
    plt.ylabel('Distance [m]')
    plt.xlabel('Frame number')
    plt.title('Distance between the ego vehicle and the adversary over time')
    plt.show()
```

**2.1. Name the metric**. First of all, as it was previously mentioned, all metrics are childs of the `BasicMetric` class.

```
    class DistanceBetweenVehicles(BasicMetric):
```

**2.2. Name the main method**. A metric only requires one method in order to run, `_create_metric()`, which has three arguments.

- **`town_map`** — Instance of the carla.Map() where the scenario took place in.

- `log` — Instance to the `MetricsLog` class, with all the functions needed to access the recorder dictionary.
- `criteria` — A JSOn with the criteria dictionary. The file provided when recording a scenario that is later provided to the `metrics_manager.py`.

```
    def _create_metric(self, town_map, log, criteria):
```

**2.3. Implement the metric**. The code will vary depending on the metric itself.

The example metric `DistanceBetweenVehicles` calculates the distance between the ego vehicle, and the car it follows (adversary). To do so, it needs the `id` of the two vehicles. These can be retrieved from the log using the `get_ego_vehicle_id()` and `get_actor_ids_with_role_name("scenario")[0]` functions.

```
    # Get the ID of the two vehicles
    ego_id = log.get_ego_vehicle_id()
    adv_id = log.get_actor_ids_with_role_name("scenario")[0]  # Could have also used its type_id
```

The `id` are used to retrieve the frames where the vehicles were alive using `get_actor_alive_frames(actor_id)`.

```
    # Get the frames both actors were alive
    start_ego, end_ego = log.get_actor_alive_frames(ego_id)
    start_adv, end_adv = log.get_actor_alive_frames(adv_id)
    start = max(start_ego, start_adv)
    end = min(end_ego, end_adv)
```

Now everything is ready to loop through those frames, get their transforms, and calculate the distance. To get the transform, `get_actor_transform(actor_id, frame)` is used.

```
    dist_list = []
    frames_list = []

...

    # Get the distance between the two
    for i in range(start, end):

        # Get the transforms
        ego_location = log.get_actor_transform(ego_id, i).location
        adv_location = log.get_actor_transform(adv_id, i).location

        # Filter some points for a better graph
        if adv_location.z < -10:
            continue

        dist_v = ego_location - adv_location
        dist = math.sqrt(dist_v.x * dist_v.x + dist_v.y * dist_v.y + dist_v.z * dist_v.z)

        dist_list.append(dist)
```

The vertical condition of the adversary is to only take into account the adversary when it is driving normally.

Lastly, use matplotlib to define which should be the output of the metric when running `metrics_manager.py`.

```
        # Use matplotlib to show the results
        plt.plot(frames_list, dist_list)
        plt.ylabel('Distance [m]')
        plt.xlabel('Frame number')
        plt.title('Distance between the ego vehicle and the adversary over time')
        plt.show()
```

**3. Run the metrics manager**

Finally it is time to used the information retrieve from the scenario, and the metrics that have been defined to return some metrics information. Let's run the module using the example metric that has been used so far, and an example log named after it.

```
python metrics_manager.py --metric srunner/metrics/examples/distance_between_vehicles.py --log
    srunner/metrics/data/DistanceBetweenVehicles.log
```

A simulation must be running. Otherwise, the module will not be able to acces the map API.

This will create a new window with the results plotted. The script will not finish until the ouput window is closed.

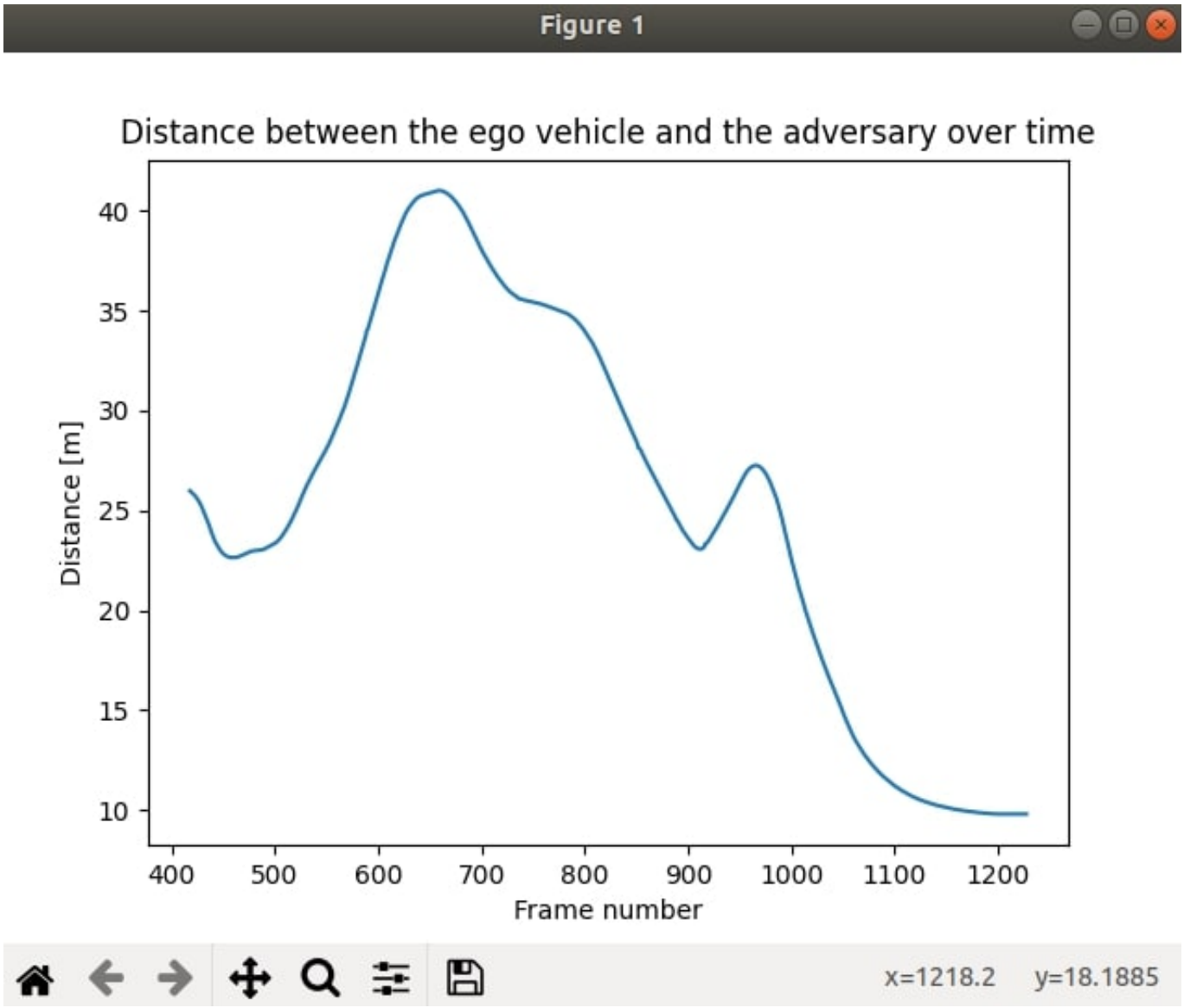


Figure 1.1: metrics_plot

### 1.4.3 Recording queries reference

When defining a metric, all the information about the scenario is accessed via the `MetricsLog` class (the `log` argument at the `_create_metric()` function). This class is located at `srunner/metrics/tools/metrics_log.py`, and this reference is a following of the functions contained in it.

#### Generic actor data

- **get_ego_vehicle_id**(**self**) Returns the `id` of the ego vehicle.
  - **Return** — int
- **get_actor_ids_with_role_name**(**self**, **role_name**) Returns a list of actor `id` that match the given `role_name`.
  - **Return** — list
  - **Parameters**

        * `role_name` (*str*) — `role_name` of the actor.
- **get_actor_ids_with_type_id**(**self**, **type_id**) Returns a list of actor `id` that match the given `type_id`, according to fnmatch standard.
  - **Return** — list
  - **Parameters**
    * `type_id` (*str*) — `type_id` of the actor.
- **get_actor_attributes**(**self**, **actor_id**) Returns a dictionary with all the attributes of an actor.
  - **Return** — dict
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
- **get_actor_bounding_box**(**self**, **actor_id**) Returns the bounding box of the specified actor.
  - **Return** — carla.BoundingBox
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
- **get_traffic_light_trigger_volume**(**self**, **traffic_light_id**) Returns the trigger volume of the specified traffic light.
  - **Return** — carla.BoundingBox
  - **Parameters**
    * `traffic_light_id` (*int*) — `id` of the traffic light.
- **get_actor_alive_frames**(**self**, **actor_id**) Returns a tuple with the first and last frame an actor was alive. Note that frames start at 1, not 0.
  - **Return** — tuple
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.

**Generic simulation data**

- **get_collisions**(**self**,**actor_id**) Returns a list of dictionaries with two keys. `frame` is the frame number of the collision, and `other_id`, a list of the ids the actor collided with at that frame.
  - **Return** — list
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
- **get_total_frame_count**(**self**) Returns an int with the total amount of frames the simulation lasted.
  - **Return** — int
- **get_elapsed_time**(**self**, **frame**) Returns a float with the elapsed time of a specific frame.
  - **Return** — float
  - **Parameters**
    * `frame` (*int*) — Frame number.
- **get_delta_time**(**self**, **frame**) Returns an float with the delta time of a specific frame.
  - **Return** — float
  - **Parameters**
    * `frame` (*int*) — Frame number.
- **get_platform_time**(**self**, **frame**) Returns a float with the platform time of a specific frame.
  - **Return** — float
  - **Parameters**
    * `frame` (*int*) — Frame number.

**Actor accelerations**

- **get_actor_acceleration**(**self**, **actor_id**, **frame**) Returns the acceleration of the actor at a given frame. Returns None if the actor `id` doesn't exist, the actor has no acceleration, or the actor wasn't alive at that frame.
  - **Return** — carla.Vector3D
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
    * `frame` (*int*) — Frame number.
- **get_all_actor_accelerations**(**self**, **actor_id**, **first_frame**=None, **last_frame**=None) Returns a list with all the accelerations of the actor at the frame interval. By default, the frame interval comprises all the recording.
  - **Return** — list(carla.Vector3D)
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
    * `first_frame` (*int*) — Initial frame of the interval. By default, the start of the simulation.
    * `last_frame` (_int_) — Last frame of the interval. By default, the end of the simulation.
- **get_actor_accelerations_at_frame**(**self**, **frame**, **actor_list**=None) Returns a dict where the keys are the frame number, and the values are the carla.Vector3D of the actor at the given frame. By default, all actors are

considered but if *actor_list* is passed, only actors in the list will be checked.
  - **Return** — list(carla.Vector3D)
  - **Parameters**
    * `frame` (*int*) — Frame number.
    * `actor_list` (_int_) — List of actor `id`.

**Actor angular velocities**

- **get_actor_angular_velocity**(**self**, **actor_id**, **frame**) Returns the angular velocity of the actor at a given frame. Returns None if the actor id doesn't exist, the actor has no angular velocity, or the actor wasn't alive at that frame.
  - **Return** — carla.Vector3D
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
    * `frame` (*int*) — Frame number.
- **get_all_actor_angular_velocities**(**self**, **actor_id**, **first_frame**=None, **last_frame**=None) Returns a list with all the angular velocities of the actor at the frame interval. By default, the frame interval comprises all the recording.
  - **Return** — list(carla.Vector3D)
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
    * `first_frame` (*int*) — Initial frame of the interval. By default, the start of the simulation.
    * `last_frame` (_int_) — Last frame of the interval. By default, the end of the simulation.
- **get_actor_angular_velocities_at_frame**(**self**, **frame**, **actor_list**=None) Returns a dictionary where the keys are the frame number, and the values are the carla.Vector3D of the actor at the given frame. By default, all actors are considered but if `actor_list` is passed, only actors in the list will be checked.
  - **Return** — list(carla.Vector3D)
  - **Parameters**
    * `frame` (*int*) — frame number.
    * `actor_list` (_int_) — List of actor ids.

**Actor controls**

- **get_vehicle_control**(**self**, **vehicle_id**, **frame**) Returns the control of a vehicle at a given frame. The `manual_gear_shift` attribute will always be False.
  - **Return** — carla.VehicleCotnrol
  - **Parameters**
    * `vehicle_id` (_int_) — `id` of the vehicle.
    * `frame` (*int*) — Frame number.
- **get_vehicle_physics_control**(**self**, **vehicle_id**, **frame**) Returns the physics control of a vehicle at a given frame.
  - **Return** — carla.VehiclePhysicsControl
  - **Parameters**
    * `vehicle_id` (_int_) — `id` of the vehicle.
    * `frame` (*int*) — Frame number.
- **get_walker_speed**(**self**, **walker_id**, **frame**) Returns the speed of a walker at a given frame.
  - **Return** — carla.Vector3D
  - **Parameters**
    * `walker_id` (*int*) — `id` of the walker.
    * `frame` (*int*) — Frame number.

**Actor transforms**

- **get_actor_transform**(**self**, **actor_id**, **frame**) Returns the transform of the actor at a given frame. Returns None if the actor id doesn't exist, the actor has no transform, or the actor wasn't alive at that frame.
  - **Return** — carla.Transform
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
    * `frame` (*int*) — Frame number.
- **get_all_actor_transforms**(**self**, **actor_id**, **first_frame**=None, **last_frame**=None) Returns a list with all the transforms of the actor at the frame interval. By default, the frame interval comprises all the recording.
  - **Return** — list(carla.Transform)
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.

    * `first_frame` (*int*) — Initial frame of the interval. By default, the start of the simulation.
    * `last_frame` (_int_) — Last frame of the interval. By default, the end of the simulation.
- **get_actor_transforms_at_frame**(**self**, **frame**, **actor_list**=None) Returns a dictionary where the keys are the frame number, and the values are the carla.Transform of the actor at the given frame. By default, all actors are considered but if `actor_list` is passed, only actors in the list will be checked.
  - **Return** — list(carla.Transform)
  - **Parameters**
    * `frame` (*int*) — Frame number.
    * `actor_list` (_int_) — List of actor `id`.

**Actor velocities**

- **get_actor_velocity**(**self**, **actor_id**, **frame**) Returns the velocity of the actor at a given frame. Returns None if the actor id doesn't exist, the actor has no velocity, or the actor wasn't alive at that frame.
  - **Return** — carla.Vector3D
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
    * `frame` (*int*) — Frame number.
- **get_all_actor_velocities**(**self**, **actor_id**, **first_frame**=None, **last_frame**=None) Returns a list with all the velocities of the actor at the frame interval. By default, the frame interval comprises all the recording.
  - **Return** — list(carla.Vector3D)
  - **Parameters**
    * `actor_id` (*int*) — `id` of the actor.
    * `first_frame` (*int*) — Initial frame of the interval. By default, the start of the simulation.
    * `last_frame` (_int_) — Last frame of the interval. By default, the end of the simulation.
- **get_actor_velocities_at_frame**(**self**, **frame**, **actor_list**=None) Returns a dict where the keys are the frame number, and the values are the carla.Vector3D of the actor at the given frame. By default, all actors are considered but if *actor_list* is passed, only actors in the list will be checked.
  - **Return** — list(carla.Vector3D
  - **Parameters**
    * `frame` (*int*) — Frame number.
    * `actor_list` (_int_) — List of actor `id`.

**Scene lights**

- **get_scene_light_state**(**self**, **light**, **vehicle_id**, **frame**) Returns the state of a scene light for a given frame. The light state group will always be carla.LightGroup.None.
  - **Return** — carla.LightState
  - **Parameters**
    * `light_id` (*int*) — `id` of the scene light.
    * `frame` (*int*) — Frame number.

**Traffic lights**

- **get_traffic_light_state**(**self**, **traffic_light_id**, **frame**) Returns the state of a traffic light at a given frame.
  - **Return** — carla.TrafficLightState.
  - **Parameters**
    * `traffic_light_id` (*int*) — `id` of the traffic light.
    * `frame` (*int*) — Frame number.
- **is_traffic_light_frozen**(**self**, **traffic_light_id**, **frame**) Returns whether or not a traffic light is frozen at a given frame.
  - **Return** — bool
  - **Parameters**
    * `traffic_light_id` (*int*) — `id` of the traffic light.
    * `frame` (*int*) — Frame number.
- **get_traffic_light_elapsed_time**(**self**, **traffic_light_id**, **frame**) Returns the elapsed time of a traffic light at a given frame.
  - **Return** — float
  - **Parameters**
    * `traffic_light_id` (*int*) — `id` of the traffic light.
    * `frame` (*int*) — Frame number.
- **get_traffic_light_state_time**(**self**, **traffic_light_id**, **state**, **frame**) Returns the maximum time of a specific state of a traffic light at a given frame.
  - **Return** — float

- **Parameters**
  - `traffic_light_id` (*int*) — `id` of the traffic light.
  - `state` (carla.TrafficLightState) — `id` of the actor.
  - `frame` (*int*) — Frame number.

**Vehicle lights**

- **get_vehicle_lights**(**self**, **vehicle_id**, **frame**) Returns a list with the active lights of a specific vehicle at a given frame.
  - **Return** — list(carla.VehicleLightState)
  - **Parameters**
    - `vehicle_id` (_int_) — `id` of the vehicle.
    - `frame` (*int*) — Frame number.
- **is_vehicle_light_active**(**self**, **light**, **vehicle_id**, **frame**) Checks whether or not a given vehicle light is active for a vehicle at a specific frame.
  - **Return** — bool
  - **Parameters**
    - `light` (carla.VehicleLightState) — The light state to be compared with the vehicle.
    - `vehicle_id` (_int_) — `id` of the vehicle.
    - `frame` (*int*) — Frame number.

## 1.5 Frequently Asked Questions

### 1.5.1 I receive the error "TypeError: 'instancemethod' object has no attribute 'getitem' " in the agent navigation

This issue is most likely caused by an outdated version of the Python Networkx package. Please remove the current installation (e.g. sudo apt-get remove python-networkx) and install it using "pip install –user networkx==2.2".

### 1.5.2 No scenario visible and I receive the message "No more scenarios .... Exiting"

In case you receive the following output

```
Preparing scenario: FollowLeadingVehicle_1
ScenarioManager: Running scenario FollowVehicle
Resetting ego-vehicle!
Failure!
Resetting ego-vehicle!
ERROR: failed to destroy actor 527 : unable to destroy actor: not found
No more scenarios .... Exiting
```

and you see nothing happening, it is most likely due to the fact, that you did not launch a program to control the ego vehicle. Run for example manual_control.py, and you should now see something happening.

### 1.5.3 Scenario Runner exits with error when using –debug commandline parameter

In case you receive the following output

```
UnicodeEncodeError: 'ascii' codec can't encode character '\u2713' in position 58: ordinal not in
    range(128)
```

Please set environment variable

```
PYTHONIOENCODING=utf-8
```

### 1.5.4 Table of Contents

### 1.5.5 Latest Changes

**:rocket: New Features**

- Added a sensor barrier for the agents to ensure that the simulation waits for them to render their data.
- Added an option to produce a machine-readable JSON version of the scenario report.
- Added a static obstacle evasion OpenSCENARIO scenario
- Added support for OSC Routing options #### :bug: Bug Fixes
- Fixed metrics-manager.py failing to run with port argument
- Fixed exception when using OSC scenarios without EnvironmentAction inside Storyboard-Init
- Fixed bug causing the TrafficManager to not be correctly updated at asynchronous simualtions
- Fixed shutdown issue in ScenarioRunner causing to not switch to asynchronous mode
- Fixed OSC TeleportAction within Story
- Fixed runtime exception on RouteScenario without an agent parameter
- Fixed bug causing the InTimeToArrivalToVehicle atomic to crash if one of the actors was a a static object
- Fixed writing result files when using OpenSCENARIO under Windows (CARLA: prefix is removed from the filename) #### :ghost: Maintenance
- Added check to ensure OSC names (for story/act/maneuver) are unique

### 1.5.6 CARLA ScenarioRunner 0.9.10

**:rocket: New Features**

- Renamed some agent labels inside Jenkins CI pipelines for new standard proposals.
- Added support for Jenkins CI pipelines doing automated testing and docker images creation.
- **Very important:** CarlaActorPool has been removed and all its functions moved to the CarlaDataProvider:
  - The spawning functions have been refactored. All the *setup* functions have been removed, and its functionalities moved to their *request* counterparts. For example, previously *request_new_actor* just called *setup_actor*, but now *setup_actor* no longer exists, and the spawning is done via *request_new_actor*. They have also been unified and are now more consistent.
  - Changed *ActorConfiguration* to *ActorConfigurationData.parse_from_node*
  - Renamed the *map* element at routes to *town*, matching the scenario configuration files
- Added new environment variables needed. They can be seen at (Docs/getting_scenariorunner.md).
- Improved the visual display of the information from the *output* and *file* arguments.
- Routes are now deterministic in regards to the spawning scenarios when more than one are at the same location
- The BackgroundActivity functionality has been unchanged but some tweaks have been made, fixing a previous patch. As a result, the *amount* parameter at *ActorConfigurationData* has been removed.
- Remade how ScenarioRunner reads the scenarios files. It now reads all scenarios inside the *srunner/scenarios* folder without needing to import them. Scenarios outside that folder will still need the *–additionalScenario* argument.
- The new weather parameters (related to fog) are now correctly read when running scenarios outside routes.
- Enable weather animation during scenario execution (requires ephem pip package)
- Changed manual control to be in par with the CARLA version. Among others, added vehicle lights, recording and some new sensors
- Removed unsupported scenarios (ChallengeBasic and BackgroundActivity, VehicleTurnLeftAtJunction)
- Added a new metrics module, which gives access to all the information about a scenario in order to allow the user to extract any desired information about the simulation. More informationhere
- Removed the default randomness at the ControlLoss scenario
- OpenSCENARIO support:
  - Added support for controllers and provided default implementations for vehicles and pedestrians. This required changing the handling of actors, which results in that now all actors are controlled by an OSC controller. Supported controllers:
    - Pedestrian controller
    - NPC vehicle controller (based on CARLA LocalPlanner)
    - Simple vehicle controller to set velocities not brake/throttle, and consider obstacles in the forward-facing region.
    - External controller (to forward control to external entities)
  - Added initial speed support for pedestrians for OpenSCENARIO
  - Support for EnvironmentActions within Story (before only within Init). This allows changing weather conditions during scenario execution
  - Added support for RelativeSpeedCondition
  - Added support for AccelerationCondition
  - Added support for TimeOfDayCondition
  - Added support for OffroadCondition
  - Added support for CollisionCondition

  - Added support for EndOfRoadCondition
  - Added support for TimeHeadwayCondition
  - Added support for TrafficSignalCondition
  - Added support for AcquirePositionAction
  - Extended FollowLeadingVehicle example to illustrate weather changes
  - Created example scenarios to illustrate usage of controllers and weather changes
  - Extended LaneChangeAction to allow lane changes of multiple lanes
  - Reworked the handling of Catalogs to make it compliant to the 1.0 version (relative paths have to be relative to the scenario file)
  - The RoadNetwork can be defined as global Parameter
  - Fixed handling of relative positions with negative offset
  - Added support for local ParamaterDeclarations
  - Added support for Parameters within Catalogs
  - Added support for ParameterAssignments for CatalogReferences
  - Fixed name handling of Parameters: Parameter declerations must not start with a leading ‘’, *butwhentheparameterisu* is required.
  - Fixed use of Parameters for multiple instances of the same Catalog element
  - Fixed use of relative initial positions for any actor
  - Added possibility to use synchronous execution mode with OpenSCENARIO
  - Fixed use of relative paths in CustomCommandAction
  - Fixed use of ControllerCatalogs
- Atomics:
  - Several new atomics to enable usage of OSC controllers
  - WeatherBehavior to simulate weather over time
  - UpdateWeather to update weather to a new setting, e.g. sun to rain
  - UpdateRoadFriction to update the road friction while running
  - new RelativeVelocityToOtherActor trigger condition, used to compare velocities of two actors
  - new TriggerAcceleration trigger condition which compares a reference acceleration with the actor's one.
  - new TimeOfDayComparison trigger condition, comparing the simulation time (set up by the new weather system) with a given *datetime*.
  - Added new *OffRoadTest* criteria.
  - Added new *EndofRoadTest* criteria, to detect when a vehicle changes between OpenDRIVE roads.
  - CollisionTest criterion can now filter the collisions for a specific actor, or actor type_id.
  - Added a *duration* argument to *OnSidewalkTest* criteria, which makes the criteria fail after a certain time has passed, instead of doing so immediately. The default behavior has been unchanged.
  - InTimeToArrivalToVehicle has had its two actor arguments swapped, to match all the other behaviors.
  - Added *along_route* flag to InTimeToArrivalToVehicle, to take into account the topology of the road
  - Changed the inputs to TrafficLightStateSetter to match the other atomics, but the functionality remains unchanged
  - Improved LaneChange atomic to allow lane changes of multiple lanes

**:bug: Bug Fixes**

- Fixed bug causing parsing RelativeTargetSpeed tag to fail.
- Fixed missing ‘six’ in requirements.txt
- Support OpenSCENARIO parameters also if they're only part of a string value
- Support Routes in Catalogs
- Fix parsing of properties within ControllerCatalogs
- Add cleanup of instantiated OpenSCENARIO controllers
- Do not register SIGHUP signal in windows
- Fixed initial speed of vehicles using OpenSCENARIO
- Fixed bug causing an exception when calling BasicScenario's *_initialize_actors* with no other_actors.
- Fixed bug causing the route to be downsampled (introduced by mistake at 0.9.9)
- Fixed bug causing *output* argument to not display the correct number with *InRouteTest* and *RouteCompletionTest* criterias (the succces and failure was correctly displayed)
- Fixed bug causing OpenSCENARIO's SpeedCondition to not work as intended
- Fixed bug causing CollisionConditions not to work properly in OpenSCENARIO
- Fixed bug causing the *group:* functionality to behave incorrectly when moving scenarios around.
- Fixed bug causing FollowLeadingVehicle and FollowLeadingVehicleWithObstacle scenarios to not properly end
- Fixed bug causing CollisionTest to ignore multiple collisions with scene objects
- Fixed bug causing NoSignalJunctionCrossing to not output the results of the scenario
- Fixed bug causing SyncArrival to fail when the actor was destroyed after the behavior ended
- Fixed bug with ending roads near stop signals to break the simulation

- Fixed exception bug in spawn function of CarlaDataProvider
- Fixed access to private member of CARLA LocalPlanner inside OSC NpcVehicleControl
- Fixed bug causing LaneChange to break the simulation if the asked lane change was impossible, instead of correctly stopping it
- Fixed bug causing ChangeLane scenarios to never end
- Fixed handling of OSC LanePosition (#625)
- Fixed bug causing the route repetitions to spawn different background activity
- Fixed bug causing the rotate_point function inside RunningRedLightTest to not function properly. #### :ghost: Maintenance
- Exposed traffic manager port flag to enable the execution of multiple scenarios on a single machine.

### 1.5.7 CARLA ScenarioRunner 0.9.9

**:rocket: New Features**

- OpenSCENARIO support:
  - Support for OpenSCENARIO 1.0 (a converter for old scenarios is available)
  - Added support for position with Lane information (roadId and laneId)
  - Added support to use a non-CARLA OpenDRIVE map (instead of CARLA towns)
  - Added support for TimeOfDay tag
  - Added support for scenarios with no actors
  - Added support for TimeToCollisionCondition with freespace.
  - Added support for TimeHeadwayCondition with freespace.
- Scenario updates:
  - Scenarios that are part of RouteScenario have had their triggering condition modified. This will only activate when a certain parameter is set, and if not, the old trigger condition will still be applied.
- Atomics:
  - ChangeAutopilot now calls a TM instance, and allows to change its parameters
  - Added WaitUntilInFront behavior and InTimeToArrivalToVehicleSideLane trigger condition, useful for cut ins
  - Added new trigger condition, AtRightestLane, which checks if the actor is at the rightmost driving lane
  - Added new criteria, ActorSpeedAboveThresholdTest, useful to check if the ego vehicle has been standing still for long periods of time.
- Setting up actors in batch now also randomizes their colors
- When running routes, the weather parameters of each route can now be changed at will. Check the first route at srunner/data/routes_training.xml to see the correct format to do so. By default the weather is now a sunny midday.
- **Important** All challenge related content has been removed. This functionality has been improved and is now part of the Leaderboard. As a consequence:
  - The path to the autoagents has changed from …/challenge/autoagents to …/autoagents
  - The path to the route and scenario descriptions has changed from …/challenge to …/data #### :bug: Bug Fixes
- Fixed spawning bugs for scenario DynamicObjectCrossing when it is part of a route
- Fixed spawning bugs for scenarios VehicleTurningRight, VehicleTurningLeft when they are part of a route
- Fixed bug causing the GPS coordinates given to the agents to be wrongly calculated
- Fixed bug when setting up actors in batch causing to ignore the spawn points given.
- Fixed bug where CollisionTest was counting as multiple hits collisions that displaced the actor for a long distance.
- Fixed bug causing the simulation to end after running in synchronous mode
- Fixed bug when using the WaypointFollower atomic to create new LocalPlanners for on-the-fly created actors (#502)
- Fixed bug causing the scenarios to run faster than real time. #### :ghost: Maintenance
- Removed perform_carla_tick() function at CarlaDataProvider, which was a workaround for world.tick()

### 1.5.8 CARLA ScenarioRunner 0.9.8

**:rocket: New Features**

- Added "–timeout" command line parameter to set a user-defined timeout value
- Scenario updates:
  - Changed traffic light behavior of scenarios 7, 8 and 9. The new sequence is meant to greatly improve the chances of the ego vehicle having to interact at junctions.
- OpenSCENARIO support:
  - Added initial support for Catalogs (Vehicle, Pedestrian, Environment, Maneuver, and and MiscObject types only) #### :bug: Bug Fixes

- Fixed #471: Handling of weather parameter (cloudyness -> cloudiness adaption)
- Fixed #472: Spawning issue of pedestrians in OpenSCENARIO
- Fixed #374: Usage of evaluation critieria with multiple ego vehicles in OpenSCENARIO
- Fixed #459: Add initial support for Catalogs (Vehicle, Pedestrian, Environment, Maneuver, and and MiscObject types only)
- Fixed wrong StandStill behavior which return SUCCESS immediatly on a standing actor
- Fixed scenario bug causing junction related scenarios (4, 7, 8 and 9) to not spawn due to lane changes. #### :ghost: Maintenance
- Added watchdog to ScenarioManager to handle timeouts and CARLA crashes
- Added timeout for CARLA tick() calls to avoid blocking CARLA server calls

### 1.5.9 CARLA ScenarioRunner 0.9.7

**This is the *first* release to work with CARLA 0.9.7 (not the patch versions 0.9.7.x)** #### :rocket: New Features * Challenge routes can be directly executed with the ScenarioRunner using the –route option * Agents can be used with the ScenarioRunner (currently only for route-based scenarios) * New scenarios: - Added example scenario for lane change - Added cut-in example scenario * Scenario updates: - Scenarios 7 to 10 are now visible when running routes (instead of being triggered in the background). Their methodology has remained unchanged * Scenario atomics: - Added new OutsideRouteLanesTest atomic criter that encompasses both SidewalkTest and WrongLaneTest, returning the percentage of route that has been traveled outside the lane. - InRouteTest is now more forgiving. The max distance has been increased, but staying above the previous one will eventually also cause failure - Changed SidewalkTest atomic criteria to also track other type of out of lane conditions - SidewalkTest and WrongLaneTest atomic criterias now track the amount of meters traversed - CollisionTest atomic criteria now correctly ignores multiple micro-collisions with the same object - Added LaneChange and TrafficLightSateSetter behavior atomics - Added AccelerateToCatchUp behavior atomic - Added get_transform() method for CarlaDataProvider - Added support for weather conditions - Added basic version check to ensure usage of correct CARLA version - WaypointFollower atomic can handle pedestrians - Extensions in WaypointFollower atomic for consecutive WaypointFollowers (one WF cancels the previous one) * Extended OpenScenario support: - Added support for UserDefinedActions (e.g. to run additional scripts) - Added init speed behavior for vehicles - Added support for relative velocities - Extended convert_position_to_transform with RelativeWorld, RelativeObject and RelativeLane osc_positions - Added new trigger atomics InTriggerDistanceToOSCPosition and InTimeToArrivalToOSCPosition to support relative osc_positions - Added new atomic behaviour ActorTransformSetterToOSCPosition - Workaround for relative osc_positions: World is started earlier to support relative osc_positions in story init - Added delay condition support in convert_condition_to_atomic - Added support for pedestrians - Full support for SimulationTime condition - Added weather support - Updated implementation to be closer to upcoming OpenSCENARIO standard - AfterTermination, AtStart conditions are supported - Added initial support for lateral action: LaneChange - Added initial support for OSCGlobalAction to set state of traffic signal - FollowRoute action is supported for vehicles and pedestrians, for global world positions. - Added support for RoadCondition: Friction - Redundant rolename object property is no longer required - Added support for global parameters - Fixed coordinate system to use right-hand as default. Left-hand CARLA system can be used by adding “CARLA:” at the start of the description in the FileHeader. - Added support to change actor color - Added support for a default actor model, in case the stated model is not available - Added support for MiscObjects (besides vehicles and pedestrians) - Reworked traffic signal handling: The name has to start now either with “id=” or “pos=” depending on whether the position or id is used as unique identifier - Actor physics can now be set via Object Properties () #### :bug: Bug Fixes * Fixed wrong handling of OpenSCENARIO ConditionGroups, which should be handled as parallel composites, not sequences * Fixed #443: Repetitions in OpenSCENARIO were not properly working * Fixed bug causing RunningStopTest atomic criteria to trigger when lane changing near a STOP signal * Fixed bug causing RunningRedLightTest atomic criteria to occasionally not trigger * Fixed bug causing occasional frame_errors * Fixed #426: Avoid underground vehicles fall forever by disabling physics when spawning underground. * Fixed #427: Removed unnecessary warnings when using get_next_traffic_light() with non-cached locations * Fixed missing ego_vehicle: compare actor IDs instead of object in CarlaDataProvider in get_velocity, get_transform and get_location * Avoided use of ‘controller.ai.walker’ as walker type in DynamicObjectCrossing scenario * Fixed WaypointFollower behavior to use m/s instead of km/h * Fixed starting position of VehicleTurnLeft/Right scenarios * Fixed spawn_point modification inside CarlaActorPool.setup_actor() * Fixed result of DrivenDistanceTest * Fixed exception in manual_control on fps visualization * Cleanup of pylint errors for all autonomous agents * Fixed randomness of route-based scenarios * Fixed usage of radians instead of degrees for OpenSCENARIO * Fixed ActorTransformSetter behavior to avoid vehicles not reaching the desired transform * Fixed spawning of debris for ControlLoss scenario (Scenario01) * Fixed CTRL+C termination of ScenarioRunner #### :ghost: Maintenance * Increased speed of actor initialization by using CARLA batch mode and buffering CARLA blueprint library * Split of behaviors into behaviors and conditions * Moved atomics into new submodule scenarioatomics * Updated documentation for all behaviors, conditions and test criteria * Refactoring of scenario configurations and parsers * Extended WaypointFollower atomic behavior to be able to use the current actor speed * Removed usage of ’import *’ to have cleaner Python imports* Removed broad-except and bare-except where possible * Python-Scenarios: Removed obsolete categories * ScenarioRunner: Removed scenario dictonary, use imports directly * CarlaDataProvider: Simplified update_light_states() to remove code duplication * Timer: class

TimeOut() is derived from SimulationTimeCondition() to avoid code duplication * Moved backported py_trees classes and methods to tools/py_trees_port.py to avoid code duplication * Removed setup_environment.sh * Adaptions to CARLA API Changes - Renamed GnssEvent to GnssMeasurement

### 1.5.10 CARLA ScenarioRunner 0.9.6

**This is the *first* release to work with CARLA 0.9.6** #### :ghost: Maintenance * Adapted to CARLA API changes - Frame rate is set now via Python - Renamed frame_count and frame_number to frame - Removed wait_for_tick() calls

### 1.5.11 CARLA ScenarioRunner 0.9.5.1

**This is the *last* release that works with CARLA 0.9.5** #### :rocket: New Features * Added initial support for OpenScenario v0.9.1 * Added support for multiple ego vehicles plus an example * Added commandline option for output directory * Added option to load external scenario implementations (in python) * Added option to scenario_runner to load external scenario XMLs * Atomic behaviors: - Extended KeepVelocity atomic behavior to support duration/distance based termination - Extended StandStill atomic behavior to support duration based termination - Added behavior to activate/deactivate autopilot #### :bug: Bug Fixes * Fixed WaypointFollower initialization

### 1.5.12 CARLA ScenarioRunner 0.9.5

**This is the *first* release to work with CARLA 0.9.5** #### :rocket: New Features * Added support for CARLA challenge - Added logging functionalities to challenge_evaluator_routes.py - Added wall clock timeout for the CARLA challenge - Added background scenario to generate dynamic traffic using autopilot - Updated compatibility with Python 2.7 for the challenge evaluator - Updated WaypointFollower behavior - Added detect_lane_obstacle() helper function which identifies if an obstacle is present in front of the reference actor - Added test to detect vehicles running a stop - Updated the reference position for a scenario is now called trigger_point - Added universal access to the map without re-calling get_map() - Added criteria_enable flag to enable/disable criteria tree - Added multiple helper methods for generic scenario execution. - Added pseudo-sensors for SceneLayoutMeasurements and ObjectMeasurements for Track4 of the CARLA AD challenge - Added track identification for autonomous_agent.py - Added HDMap pseudo-sensor - Added new traffic event logger - Added various helper methods to allow generic scenario execution - Added method to calculate distance along a route - In challenge mode spawn exception are caught and the corresponding scenario is removed * Added new atomic behaviors using py_trees behavior tree library - BasicAgentBehavior: drive to target location using CARLA's BasicAgent - StandStill: check if a vehicle stands still - InTriggerDistanceToNextIntersection: check if a vehicle is within certain distance with respect to the next intersection - WaypointFollower: follows auto-generated waypoints indefinitely or follows a given waypoint list - HandBrakeVehicle: sets the handbrake value for a given actor - ActorDestroy: destroys a given actor - ActorTransformSetter: sets transform of given actor - ActorSource: creates actors indefinitely around a location if no other vehicles are present within a threshold - ActorSink: indefinitely destroys vehicles that wander close to a location within a threshold - InTriggerDistanceToLocationAlongRoute: check if an actor is within a certain distance to a given location along a given route * Added new atomic evaluation criteria - Added running red light test - Added running stop test - Added wrong way test * Added NHTSA Traffic Scenarios - Updated all traffic scenarios to let the other actors appear upon scenario triggering and removal on scenario end - ManeuverOppositeDirection: hero vehicle must maneuver in the opposite lane to pass a leading vehicle. - OtherLeadingVehicle: hero vehicle must react to the deceleration of leading vehicle and change lane to avoid collision and follow the vehicle in changed lane - SignalizedJunctionRightTurn: hero vehicle must turn right into the same direction of another vehicle crossing straight initially from a lateral direction and avoid collision at a signalized intersection. - SignalizedJunctionLeftTurn : hero vehicle is turning left at signalized intersection, cuts across the path of another vehicle coming straight crossing from an opposite direction. #### :bug: Bug Fixes * Fixed SteerVehicle atomic behavior to keep vehicle velocity #### :ghost: Maintenance * Reworked scenario execution - Updated folder structure and naming convention in lowercase - Extended CarlaDataProvider with method to get next relevant traffic light - Every scenario has to have a configuration provided as XML file. Currently there is one XML file for each scenario class - The scenario runner is now responsible for spawning/destroying the ego vehicle - Added a CarlaActorPool to share scenario-related actors between scenarios and the scenario_runner - Renamed vehicle -> actor - If all scenarios in one configurations file should be executed, the scenario_runner can be started with –scenario group: - Generalized ControlLoss and FollowLeadingVehicle scenarios - Added randomization option to scenario_runner and scenarios - The scenario behavior always starts with a wait behavior until the ego vehicle reached the scenario starting position - Created method _initialize_actors in basic scenario that can be overridden for scenario specific actor initialization * Updated NHTSA Traffic Scenarios - OppositeVehicleRunningRedLight: Updated to allow execution at different locations

### 1.5.13 CARLA ScenarioRunner 0.9.2

**This release is designed to work with CARLA 0.9.2** #### :rocket: New Features * Added Traffic Scenarios engine to reproduce complex traffic situations for training and evaluating driving agents * Added NHTSA Traffic

Scenarios - FollowLeadingVehicle: hero vehicle must react to the deceleration of a leading vehicle - FollowLeadingVehicleWithObstacle: hero vehicle must react to a leading vehicle due to an obstacle blocking the road - StationaryObjectCrossing: hero vehicle must react to a cyclist or pedestrian blocking the road - DynamicObjectCrossing: hero vehicle must react to a cyclist or pedestrian suddenly crossing in front of it - OppositeVehicleRunningRedLight: hero vehicle must avoid a collision at an intersection regulated by traffic lights when the crossing traffic runs a red light - NoSignalJunctionCrossing: hero vehicle must cross a non-signalized intersection - VehicleTurningRight: hero vehicle must react to a cyclist or pedestrian crossing ahead after a right turn - VehicleTurningLeft: hero vehicle must react to a cyclist or pedestrian crossing ahead after a left turn - ControlLoss: Hero vehicle must react to a control loss and regain its control * Added atomic behaviors using py_trees behavior trees library - InTriggerRegion: new behavior to check if an object is within a trigger region - InTriggerDistanceToVehicle: check if a vehicle is within certain distance with respect to a reference vehicle - InTriggerDistanceToLocation: check if a vehicle is within certain distance with respect to a reference location - TriggerVelocity: triggers if a velocity is met - InTimeToArrivalToLocation: check if a vehicle arrives within a given time budget to a reference location - InTimeToArrivalToVehicle: check if a vehicle arrives within a given time budget to a reference vehicle - AccelerateToVelocity: accelerate until reaching requested velocity - KeepVelocity: keep constant velocity - DriveDistance: drive certain distance - UseAutoPilot: enable autopilot - StopVehicle: stop vehicle - WaitForTrafficLightState: wait for the traffic light to have a given state - SyncArrival: sync the arrival of two vehicles to a given target - AddNoiseToVehicle: Add noise to steer as well as throttle of the vehicle

# Chapter 2

# References

## 2.1 List of Supported Scenarios

Welcome to the ScenarioRunner for CARLA! This document provides a list of all currently supported scenarios, and a short description for each one.

#### FollowLeadingVehicle

The scenario realizes a common driving behavior, in which the user-controlled ego vehicle follows a leading car driving down a given road in Town01. At some point the leading car slows down and finally stops. The ego vehicle has to react accordingly to avoid a collision. The scenario ends either via a timeout, or if the ego vehicle stopped close enough to the leading vehicle

#### FollowLeadingVehicleWithObstacle

This scenario is very similar to 'FollowLeadingVehicle'. The only difference is, that in front of the leading vehicle is a (hidden) obstacle that blocks the way.

#### VehicleTurningRight

In this scenario the ego vehicle takes a right turn from an intersection where a cyclist suddenly drives into the way of the ego vehicle,which has to stop accordingly. After some time, the cyclist clears the road, such that ego vehicle can continue driving.

#### VehicleTurningLeft

This scenario is similar to 'VehicleTurningRight'. The difference is that the ego vehicle takes a left turn from an intersection.

#### OppositeVehicleRunningRedLight

In this scenario an illegal behavior at an intersection is tested. An other vehicle waits at an intersection, but illegally runs a red traffic light. The approaching ego vehicle has to handle this situation correctly, i.e. despite of a green traffic light, it has to stop and wait until the intersection is clear again. Afterwards, it should continue driving.

#### StationaryObjectCrossing

In this scenario a cyclist is stationary waiting in the middle of the road and blocking the way for the ego vehicle. Hence, the ego vehicle has to stop in front of the cyclist.

#### DynamicObjectCrossing

This is similar to 'StationaryObjectCrossing', but with the difference that the cyclist is dynamic. It suddenly drives into the way of the ego vehicle, which has to stop accordingly. After some time, the cyclist will clear the road, such that the ego vehicle can continue driving.

#### NoSignalJunctionCrossing

This scenario tests negotiation between two vehicles crossing cross each other through a junction without signal. The ego vehicle is passing through a junction without traffic lights And encounters another vehicle passing across the junction. The ego vehicle has to avoid collision and navigate across the junction to succeed.

#### ControlLoss

In this scenario control loss of a vehicle is tested due to bad road conditions, etc and it checks whether the vehicle is regained its control and corrected its course.

#### ManeuverOppositeDirection

In this scenario vehicle is passing another vehicle in a rural area, in daylight, under clear weather conditions, at a non-junction and encroaches into another vehicle traveling in the opposite direction.

#### OtherLeadingVehicle

The scenario realizes a common driving behavior, in which the user-controlled ego vehicle follows a leading car driving down a given road. At some point the leading car has to decelerate. The ego vehicle has to react accordingly by changing lane to avoid a collision and follow the leading car in other lane. The scenario ends via timeout, or if the ego vehicle drives certain distance.

#### SignalizedJunctionRightTurn

In this scenario right turn of hero actor without collision at signalized intersection is tested. Hero Vehicle is turning right in an urban area, at a signalized intersection and turns into the same direction of another vehicle crossing straight initially from a lateral direction.

#### SignalizedJunctionLeftTurn

In this scenario hero vehicle is turning left in an urban area, at a signalized intersection and cuts across the path of another vehicle coming straight crossing from an opposite direction.

### 2.1.1 OpenSCENARIO Support

The scenario_runner provides support for the OpenSCENARIO 1.0 standard. The current implementation covers initial support for maneuver Actions, Conditions, Stories and the Storyboard. If you would like to use evaluation criteria for a scenario to evaluate pass/fail results, these can be implemented as StopTriggers (see below). However, not all features for these elements are yet available. If in doubt, please see the module documentation in srunner/tools/openscenario_parser.py

An example for a supported scenario based on OpenSCENARIO is available here

In addition, it is recommended to take a look into the official documentation available here and here.

#### Migrating OpenSCENARIO 0.9.x to 1.0

The easiest way to convert old OpenSCENARIO samples to the official standard 1.0 is to use *xsltproc* and the migration scheme located in the openscenario folder. Example:

```
1 xsltproc -o newScenario.xosc migration0_9_1to1_0.xslt oldScenario.xosc
```

#### Level of support

In the following the OpenSCENARIO attributes are listed with their current support status.

**General OpenSCENARIO setup** This covers all part that are defined outside the OpenSCENARIO Storyboard

| Attribute | Support | Notes |
|---|---|---|
| `FileHeader` | Yes | Use “CARLA:” at the beginning of the description to use the CARLA coordinate system. |
| `CatalogLocations` | | |
| `ControllerCatalog` | Yes | While the catalog is supported, the reference/usage may not work. |

| Attribute | Support | Notes |
|---|---|---|
| CatalogLocations EnvironmentCatalog | Yes | |
| CatalogLocations ManeuverCatalog | Yes | |
| CatalogLocations MiscObjectCatalog | Yes | |
| CatalogLocations PedestrianCatalog | Yes | |
| CatalogLocations RouteCatalog | Yes | While the catalog is supported, the reference/usage may not work. |
| CatalogLocations TrajectoryCatalog | Yes | While the catalog is supported, the reference/usage may not work. |
| CatalogLocations VehicleCatalog | Yes | |
| ParameterDeclarations | Yes | |
| RoadNetwork LogicFile | Yes | The CARLA level can be used directly (e.g. LogicFile=Town01). Also any OpenDRIVE path can be provided. |
| RoadNetwork SceneGraphFile | No | The provided information is not used. |
| RoadNetwork TafficSignals | No | The provided information is not used. |
| Entities EntitySelection | No | The provided information is not used. |
| Entities ScenarioObject CatalogReference | Yes | The provided information is not used. |
| Entities ScenarioObject MiscObject | Yes | The name should match a CARLA vehicle model, otherwise a default vehicle based on the vehicleCategory is used. BoundingBox entries are ignored. |
| Entities ScenarioObject ObjectController | No | The provided information is not used. |
| Entities ScenarioObject Pedestrian | Yes | The name should match a CARLA vehicle model, otherwise a default vehicle based on the vehicleCategory is used. BoundingBox entries are ignored. |
| Entities ScenarioObject Vehicle | Yes | The name should match a CARLA vehicle model, otherwise a default vehicle based on the vehicleCategory is used. The color can be set via properties (‘Property name=“color” value=“0,0,255” ’). Axles, Performance, BoundingBox entries are ignored. |

**OpenSCENARIO Actions** The OpenSCENARIO Actions can be used for two different purposes. First, Actions can be used to define the initial behavior of something, e.g. a traffic participant. Therefore, Actions can be used within the OpenSCENARIO Init. In addition, Actions are also used within the OpenSCENARIO story. In the following, the support status for both application areas is listed. If an action contains of submodules, which are not listed, the support status applies to all submodules.

| GlobalAction | Init support | Story support | Notes |
|---|---|---|---|
| `EntityAction` | No | | |
| `EnvironmentAction` | Yes | No | |
| `ParameterAction` | No | | |
| `InfrastructureAction` | | | |
| `TrafficSignalAction` | | | |
| `TrafficAction` | No | | |
| `InfrastructureAction` | | | |
| `TrafficSignalAction` | | | |
| `TrafficSignalControllerAction` | No | | |
| `InfrastructureAction` | | | |
| `TrafficSignalAction` | | | |
| `TrafficSignalStateAction` | No | Yes | As traffic signals in CARLA towns have no unique ID, they have to be set by providing their position (Example: TrafficSignalStateAction name="pos=x,y" state="green"). The id can also be used for none CARLA town (Example: TrafficSignalStateAction name="id=n" state="green") |

**GlobalAction**

| UserDefinedAction | Init support | Story support | Notes |
|---|---|---|---|
| `CustomCommandAction` | No | Yes | This action is currently used to trigger the execution of an additional script. Example: type="python /path/to/script args". |

**UserDefinedAction**

| PrivateAction | Init support | Story support | Notes |
|---|---|---|---|
| `ActivateControllerAction` | No | Yes | Can be used to activate/deactive the CARLA autopilot. |
| `ControllerAction` | Yes | | AssignControllerAction is supported, but a Python module has to be provided for the controller implementation, and in OverrideControllerValue-Action all values need to be `False`. |

| PrivateAction | Init support | Story support | Notes |
|---|---|---|---|
| `LateralAction LaneChangeAction` | No | Yes | Currently all lane changes have a linear dynamicShape, the dynamicDimension is defined as the distance and are relative to the actor itself (RelativeTargetLane). |
| `LateralActionLaneOffsetAction` | No | | |
| `LateralAction LateralDistanceAction` | No | | |
| `LongitudinalAction LongitudinalDistanceAction` | No | | |
| `LongitudinalAction SpeedAction` | Yes | | |
| `SynchronizeAction` | No | | |
| `TeleportAction` | Yes | | |
| `VisibilityAction` | No | | |
| `RoutingAction AcquirePositionAction` | No | Yes | |
| `RoutingAction AssignRouteAction` | No | Yes | Route Options (shortest/fastest/etc) are supported. Shortests means direct path between A and B, all other will use the shortest path along the road network between A and B |
| `RoutingAction FollowTrajectoryAction` | No | | |

**PrivateAction**

**OpenSCENARIO Conditions** Conditions in OpenSCENARIO can be defined either as ByEntityCondition or as ByValueCondition. Both can be used for StartTrigger and StopTrigger conditions. The following two tables list the support status for each.

| EntityCondition | Support | Notes |
|---|---|---|
| **ByEntityCondition** | `AccelerationCondition` Yes | |
| `CollisionCondition` | Yes | |
| `DistanceCondition` | Yes | *freespace* attribute is still not supported |
| `EndOfRoadCondition` | Yes | |
| `OffroadCondition` | Yes | |
| `ReachPositionCondition` | Yes | |
| `RelativeDistanceCondition` | Yes | *freespace* attribute is still not supported |
| `RelativeSpeedCondition` | Yes | |
| `SpeedCondition` | Yes | |
| `StandStillCondition` | Yes | |
| `TimeHeadwayCondition` | Yes | |
| `TimeToCollisionCondition` | Yes | |
| `TraveledDistanceCondition` | Yes | |

| ValueCondition | Support | Notes |
|---|---|---|
| ParameterCondition | Yes | The level of support depends on the parameter. It is recommended to use other conditions if possible. Please also consider the note below. |
| SimulationTimeCondition | Yes | |
| StoryboardElementStateCondition | Yes | startTransition, stopTransition, endTransition and completeState are currently supported. |
| TimeOfDayCondition | Yes | |
| TrafficSignalCondition | Yes | As traffic signals in CARLA towns have no unique ID, they have to be set by providing their position (Example: TrafficSignalCondition name="pos=x,y" state="green"). The id can also be used for none CARLA town (Example: TrafficSignalCondition name="id=n" state="green") |
| TrafficSignalControllerCondition | No | |
| UserDefinedValueCondition | No | |

**ByValueCondition**

In the OpenSCENARIO 1.0 standard, a definition of test / evaluation criteria is not def
purpose, you can re-use StopTrigger conditions with CARLA. The following StopTrigg
for evaluation criteria are supported through ParameterConditions by providing the crit
the condition:

```
* criteria_RunningStopTest
* criteria_RunningRedLightTest
* criteria_WrongLaneTest
* criteria_OnSideWalkTest
* criteria_KeepLaneTest
* criteria_CollisionTest
* criteria_DrivenDistanceTest
```

**OpenSCENARIO Positions** There are several ways of defining positions in OpenSCENARIO. In the following we list the current support status for each definition format.

| Position | Support | Notes |
|---|---|---|
| LanePosition | Yes | |
| RelativeLanePosition | Yes | |
| RelativeObjectPosition | Yes | |
| RelativeRoadPosition | No | |
| RelativeWorldPosition | Yes | |
| RoadPosition | No | |
| RoutePosition | No | |
| WorldPosition | Yes | |

# Chapter 3

# Contributing

## 3.1 Contributor Covenant Code of Conduct

### 3.1.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

### 3.1.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

### 3.1.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

### 3.1.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

### 3.1.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at [INSERT EMAIL ADDRESS]. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

### 3.1.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

Coding standard

> *This document is a work in progress and might be incomplete.*

## 3.2 General

- Use spaces, not tabs.
- Avoid adding trailing whitespace as it creates noise in the diffs.
- Comments should not exceed 120 columns, code may exceed this limit a bit in rare occasions if it results in clearer code.

Python

- All code must be compatible with Python 2.7, 3.5, and 3.6.
- Pylint should not give any error or warning (few exceptions apply with external classes like `numpy`, see our `.pylintrc`).
- Python code follows PEP8 style guide (use `autopep8` whenever possible).

C++

- Compilation should not give any error or warning (`clang++ -Wall -Wextra -std=C++14 -Wno-missing-braces`).
- Unreal C++ code (CarlaUE4 and Carla plugin) follow the Unreal Engine's Coding Standard with the exception of using spaces instead of tabs.
- LibCarla uses a variation of Google's style guide.

# Chapter 4

# Contributing to CARLA

We are more than happy to accept contributions!

How can I contribute?

- Reporting bugs
- Feature requests
- Improving documentation
- Code contributions

## 4.1 Reporting bugs

Use our issue section on GitHub. Please check before that the issue is not already reported, and make sure you have read our CARLA Documentation and FAQ.

## 4.2 Feature requests

Please check first the list of feature requests. If it is not there and you think is a feature that might be interesting for users, please submit your request as a new issue.

## 4.3 Improving documentation

If you feel something is missing in the documentation, please don't hesitate to open an issue to let us know. Even better, if you think you can improve it yourself, it would be a great contribution to the community!

We build our documentation with MkDocs based on the Markdown files inside the "Docs" folder. You can either directly modify them on GitHub or locally in your machine.

Once you are done with your changes, please submit a pull-request.

**TIP:** You can build and serve it locally by running `mkdocs` in the project's main folder

```
$ sudo pip install mkdocs
$ mkdocs serve
```

Code contributions

So you are considering making a code contribution, great! we love to have contributions from the community.

Before starting hands-on on coding, please check out our issue board to see if we are already working on that, it would be a pity putting an effort into something just to discover that someone else was already working on that. In case of doubt or to discuss how to proceed, please contact one of us (or send an email to carla.simulator@gmail.com).

**What should I know before I get started?** Check out the "CARLA Documentation" to get an idea on CARLA. In addition you may want to check theGetting started document.

**Coding standard** Please follow the currentcoding standard when submitting new code.

**Pull-requests** Once you think your contribution is ready to be added to CARLA, please submit a pull-request.

Try to be as descriptive as possible when filling the pull-request description. Adding images and gifs may help people to understand your changes or new features.

Please note that there are some checks that the new code is required to pass before we can do the merge. The checks are automatically run by the continuous integration system, you will see a green tick mark if all the checks succeeded. If you see a red mark, please correct your code accordingly.

**Checklist**

- [ ] Your branch is up-to-date with the `master` branch and tested with latest changes
- [ ] Extended the README / documentation, if necessary
- [ ] Code compiles correctly

## Acknowledgements

This work is done by Infotiv under VALU3S project.
This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.