MultiTrans UI

alaa daoud

March 2023

Abstract

Autonomous driving technology has made significant advances in recent years, and it has become a major focus of research in the field of transportation. One critical aspect of the development of autonomous vehicles is the testing and evaluation of their capabilities in various scenarios. In this report, we present an overview of a Virtualization Framework for autonomous vehicles and robotic domains, which can be used to create synthetic environments for testing and evaluation of autonomous driving algorithms.

The tool is designed to provide a flexible and customizable framework for generating scenarios that can simulate a wide range of real-world driving situations, such as urban and highway driving, pedestrian and bicycle interactions, and adverse weather conditions. The scenarios can be created and modified using a simple user interface, and can be exported to various formats for use in simulation and testing environments.

In this report, we describe the architecture and functionality of the Virtualization Framework, and provide examples of its use in testing and evaluation of autonomous driving algorithms. We also discuss the potential applications of the tool in research and development of autonomous vehicles, and highlight some of the challenges and limitations of scenario generation for autonomous driving.

Overall, this report provides a technical introduction to a Virtualization Framework for autonomous vehicles, which can be a valuable resource for researchers and practitioners in the field of autonomous driving.

Keywords: Autonomous vehicles, scenario generation, simulation.

1 Introduction

Autonomous driving technology has made significant progress in recent years, and the development of autonomous vehicles has become a major focus of research in the field of transportation. However, one of the main challenges in the development of autonomous vehicles is ensuring their safety and reliability in a wide range of driving scenarios. To address this challenge, researchers and engineers have developed simulation tools for generating a variety of driving scenarios that can be used to test and evaluate the capabilities of autonomous driving algorithms.

In this report, we introduce a Virtualization Framework for autonomous vehicles, which provides a flexible and customizable framework for creating synthetic environments that simulate a wide range of real-world driving situations. The tool is designed to be user-friendly, with a simple interface that allows users to easily create and modify scenarios. It can also export scenarios to various formats for use in simulation and testing environments.

One critical aspect of autonomous driving is the ability of the vehicle's perception system to handle corner cases, which refer to difficult or unexpected situations that can arise during driving. These can include low-light conditions, adverse weather, unusual road layouts, and unexpected behaviors by pedestrians and animals. Dealing with corner cases is crucial for ensuring the safety and reliability of autonomous vehicles, and simulation tools can be particularly useful for generating and testing scenarios that are challenging to replicate in the physical world.

In the following sections, we will describe the architecture and functionality of the Virtualization Framework, and provide examples of its use in testing and evaluation of autonomous driving algorithms. We will also discuss the importance of corner cases and the role of simulation in generating and testing scenarios for visual perception systems. Overall, this report serves as a technical introduction to the Virtualization Framework, which we believe can be a valuable resource for researchers and practitioners in the field of autonomous driving.

2 AV Driving Datasets Formats and the Need for Virtualization

AV driving datasets are critical for developing and testing autonomous driving systems. These datasets typically contain sensor data from a variety of sources, including cameras, lidar, and radar, along with information about the vehicle's motion and the surrounding environment. However, different datasets may use different formats to represent this data, which can create challenges for researchers and developers who want to use these datasets in their work.

Two common formats for AV driving datasets are the NuScenes format and the Foxglove format. The NuScenes format, used in the NuScenes dataset, stores sensor data in a complex JSON-based format, with separate files for different sensor modalities and metadata. The Foxglove format, used in the Waymo Open Dataset and others, stores sensor data in a more compact binary format, along with metadata.

Converting between these formats can be important for several reasons. For example, researchers may want to compare the performance of different autonomous driving systems on different datasets, or they may want to use a specific dataset for training or testing their own system. In these cases, it may be necessary to convert the dataset into a format that can be easily used by the system.

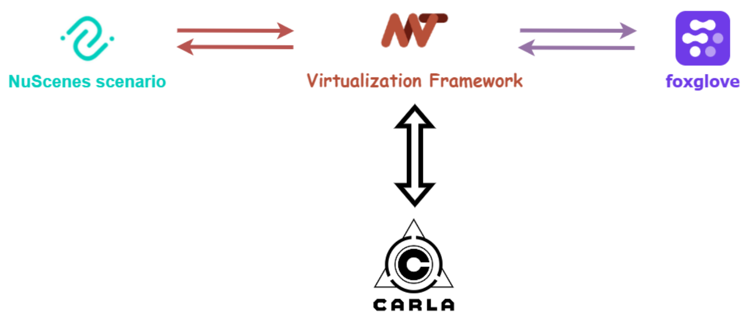

Figure 1: Abstract wotkflow of the Virtualization Framework

However, simply converting between formats is not always enough. In some cases, researchers may also need to virtualize the dataset, creating a simulated environment in which the autonomous driving system can operate. This can be important for several reasons, including:

- **Scalability:** Creating a virtualized dataset allows researchers to easily generate large amounts of data for testing and training, without the need for physical vehicles or sensors.
- **Flexibility:** Virtualizing the dataset can allow researchers to customize the environment in which the autonomous driving system operates, testing it under a wide range of conditions and scenarios.

To address these challenges, we propose a Virtualization Framework built on top of CARLA (see Figure 1). By providing a modular framework for generating and testing scenarios, this tool can help researchers and developers convert datasets between different formats, virtualize them, and test their autonomous driving systems under a wide range of conditions.

3 Proposed Software Architecture

Our Virtualization framework is built on top of the CARLA simulator, which provides a realistic 3D environment for testing autonomous driving algorithms. The framework communicates with the CARLA server using its Python API, which allows us to import and modify driving scenarios.

The main functionality of the framework is to import scenarios in various formats, such as NuScenes, OpenScenario, and ROS. These scenarios can then be modified and converted to other formats, as needed. The framework also allows the user to generate variants of a scenario, which can be useful for testing different algorithms or evaluating the performance of a single algorithm under different conditions.

During scenario execution, the framework can record data from various sensors, such as RGB cameras, semantic segmentation sensors, and object detection

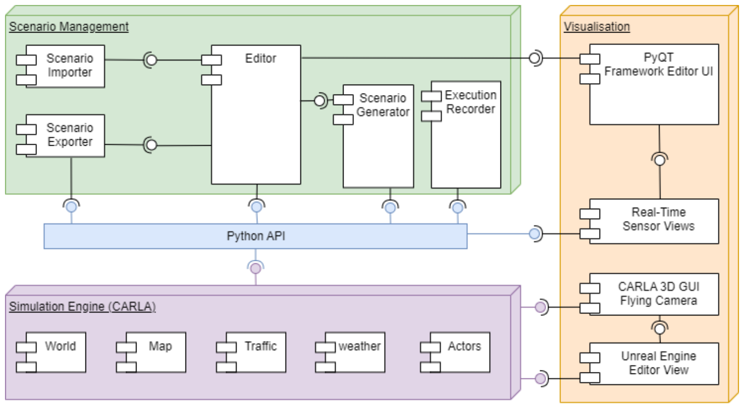

Figure 2: Virtualization Framework architecture

sensors. This data can be saved in various formats, such as image sequences, and can be used to generate labeled datasets with ground-truth labels. This allows us to train and evaluate the performance of the visual perception system of an autonomous vehicle.

The framework also includes an editor, which allows the user to modify or create new scenarios. This editor is where we can define corner cases, or challenging scenarios that can be difficult to replicate in the physical world. The editor provides a simple and intuitive interface for creating and modifying scenarios, with support for adding and removing objects, modifying object properties, and defining complex events on a scenario timeline.

Overall, our Virtualization framework provides a flexible and customizable platform for generating synthetic environments that simulate a wide range of real-world driving situations. It allows us to test and evaluate the performance of autonomous driving algorithms under various conditions, and provides a valuable tool for ensuring the safety and reliability of autonomous vehicles.

3.1 Framework Building Blocks

The Virtualization Framework follows a multi-layer architecture consisting of two main layers: the scenario management layer and the simulation engine layer. These layers are connected by the Python API, with each component of the scenario management layer serving as a client for the CARLA Simulation server.

3.1.1 Scenario Management Layer

The scenario management layer consists of the following building blocks:

Scenario importer: This module will be responsible for importing scenarios from different formats such as NuScenes, OpenScenario, or ROS. The importer should support different levels of complexity, from simple scenarios such as straight-line driving to more complex scenarios.

Scenario editor: The scenario editor module provides a user-friendly interface for creating and editing scenarios. It should allow the user to add or remove actors, modify their behavior, and adjust their positions and velocities.

Scenario generator: The scenario generator is responsible for generating variations of scenarios based on user-defined parameters. It should be able to create scenarios with varying weather conditions, traffic densities, and actor behaviors.

Result recorder: Records the execution results of the scenario, including sensor data such as RGB cameras, semantic segmentation, and bounding boxes, as well as other relevant data such as vehicle speed, location, etc. The recorded data can be used for offline analysis and evaluation of the scenarios, or a s labeled datasets with ground-trouth labels for visual perception training.

Scenario exporter: The scenario exporter module exports scenarios to different formats for use in other simulators or testing environments. It should support exporting to common formats such as NuScenes, OpenScenario and ROS.

3.1.2 Simulation Engine Layer

The simulation engine layer consists of the following building blocks:

Perception Module: The perception module provides real-time sensor views for debugging and testing visual perception algorithms. It should simulate the output of sensors such as cameras, lidars, and radars. The module can use the sensors provided by the CARLA simulator as a base.

Physics Engine: The physics engine simulates the physical behavior of the environment and objects in the scenario. It should simulate the dynamics and interactions of the vehicles, pedestrians, and other objects in the scene. This module is already provided by CARLA.

Traffic Manager: The traffic manager module manages traffic flow and behavior in the scenario. It should simulate the behavior of other vehicles, including lane-changing, speed adjustments, and reactions to traffic signals.

Weather Manager: The weather manager simulates various weather conditions and their effects on the scenario. It should simulate different weather conditions such as rain, fog, and snow, and their effects on visibility, friction, and vehicle dynamics.

Map Manager: The map manager provides access to pre-built maps and assets for creating scenarios. It should allow the user to select a map from a pre-built library or create a new map by specifying the road network, landmarks, and other details.

Obviously some of these building blocks are already provided by CARLA as modules, such as the perception module, physics engine, traffic manager, weather manager, and map manager. The Virtualization Framework builds on top of these modules, using them as foundational components for generating and testing scenarios. All these modules will communicate with each other through the Python API provided by the CARLA simulator. The API provides a way for the scenario management layer to send commands to the simulation engine layer and receive data from it. The scenario management layer can be seen as a client of the simulation engine layer, while the simulation engine layer is the server.

4 Scenario Editor: User Interface Design and Components

The UI provides a real-time view of sensor outputs, such as RGB camera, semantic segmentation, and object bounding boxes. Users can interact with the UI through various components to customize the scenario generation process.

4.1 Scenario Menu

The Scenario menu contains options to import and export scenarios. Users can import scenarios in various formats such as Nuscenes, Carla/OpenScenario, and Foxglove/ROS. The tool provides an option to export the current scenario, which opens a dialog box to choose the location and format of the resulting dataset.

4.2 Editing Tools

The editing tools in the UI allow users to add or remove objects, change object parameters, and control the sequence of events during the scenario. Users can dynamically adjust the background settings, such as weather conditions, map, and lighting. The editing tools provide a simple and intuitive way to create and modify scenarios.

4.3 Execution Controls

The UI includes start and stop buttons to control the execution of the scenario. Users can also start and stop image sequence recording to capture the simulation results.

In short, the UI of the Virtualization Framework provides a flexible and customizable framework for generating scenarios. Users can interact with real-time sensor outputs and use various editing tools to adjust the scenario settings.

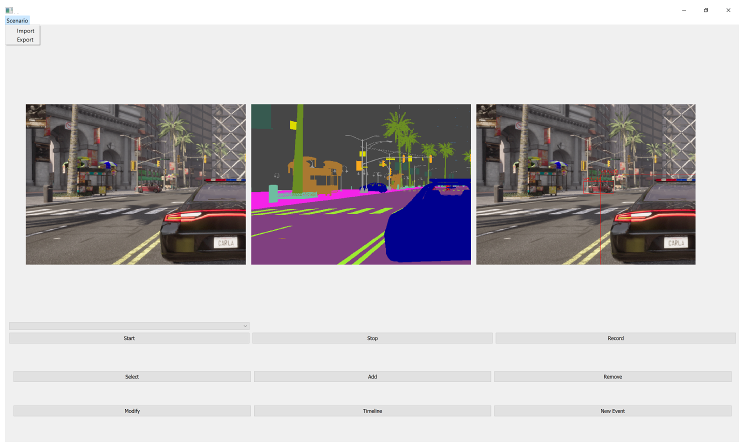

Figure 3: Simulation framework UI prototype

The execution controls allow users to run and record simulations, and the Scenario menu provides options for importing and exporting scenarios as shown in Figure 3

5 implementation

The Virtualization Framework, as mentioned earlier in this document, can be implemented using a multi-layer architecture, consisting of a scenario management layer and a simulation engine layer. These layers are connected by the Python API, which allows components of the scenario management layer to communicate with the simulation engine layer. In what follows we list some implementation datails of the different building blocks.

5.1 Simulation Engine Layer

CARLA provides several modules as part of its simulation engine layer:

- **World**: 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, and more. There is only one world per simulation, and it will be destroyed and substituted for a new one when the map is changed.
- **Actors**: In CARLA, actors are anything that plays a role in the simulation or can be moved around. This includes pedestrians, vehicles, sensors, and traffic signs. Actors are spawned in the simulation by carla.World and they require a carla.ActorBlueprint to be created.

- **Python API**: The Python API acts as a high-level querying system to navigate the world and manipulate the different actors. It provides a wide set of tools including Perception modules, Physics Engine, Traffic Manager, Weather Manager, and Map Manager.

- **Sensors**: Sensors are actors that retrieve data from their surroundings. They are crucial to create learning environment for driving agents.

- **Traffic Manager**: Traffic Manager is a module within CARLA that controls certain vehicles in a simulation from the client side. Vehicles are registered to Traffic Manager via the `carla.Vehicle.set_autopilot` method or command.SetAutopilot class. Control of each vehicle is managed through a cycle of distinct stages which each run on a different thread.

- **Weather Manager**: The weather is not a class on its own, but a set of parameters accessible from the world. The parametrization includes sun orientation, cloudiness, wind, fog, and more. To define a custom weather helper class carla.WeatherParameters is used.

- **Map Manager**: A map includes both the 3D model of a town and its road definition. A map's road definition is based on an OpenDRIVE file, a standardized, annotated road definition format. The way the OpenDRIVE standard 1.4 defines roads, lanes, junctions, etc. determines the functionality of the Python API and the reasoning behind decisions made.

These modules form the simulation engine layer of the multi-layer architecture. The Python API acts as a bridge between the simulation engine layer and the scenario management layer, allowing the Virtualization Framework to query the simulation engine and manipulate the actors and environment to create and test scenarios.

5.2 Scenario Management Layer

Modules belonging to this layer are built as clients for CARLA simulator server.

Scenario Importer: This module can be implemented using libraries such as rosbag for ROS-based scenarios or pandas for CSV-based scenarios.

Scenario Editor: Providing a graphical user interface for editing and creating scenarios, this module can be implemented using a front-end framework, which communicates with the Python back-end through an API.

Scenario Generator: Responsible for generating variations of scenarios based on user-defined parameters, this module can be implemented using techniques such as data augmentation or randomization, and can be customized based on the specific requirements of the scenario.

Result Recorder: This module can be implemented using libraries such as pandas for data storage and opencv for image processing.

Scenario Exporter: This module can be implemented using libraries such as pandas for data manipulation and numpy for array conversion.

These modules can be developed in a modular and extensible way, with each module communicating with the others through the Python API provided by CARLA. The scenario management layer can be seen as a client of the simulation engine layer, with the Python API acting as the interface between the two layers

5.3 User Interface

To implement our interface, we used the Python programming language and the QT framework. We also utilized the CARLA Python API to communicate with the CARLA server and retrieve data from the simulation.

To display real-time views of the sensors output, we set up listeners for each sensor, such as the RGB camera, semantic segmentation, and object bounding boxes. For example, to display the semantic segmentation view, we converted the image using CityScapePalette. To display the object bounding box view, we accessed the list of world objects and filtered them to keep only those in the camera view and closer than a specific threshold. For each object in this list, we retrieved the information about their bounding boxes and drew them on the RGB image from the camera. This allowed us to show the ground truth data, which we can use as part of our dataset.

We also implemented buttons for starting and stopping the scenario execution, as well as starting and stopping image sequence recording. To stop a scenario execution, we made the simulator wait for a signal to continue ticking, while starting it involved sending this signal to the server.

The Scenario menu in our interface includes an option to import a scenario from a variety of formats, such as nuscene, Carla/OpenScenario, or Foxglove, by opening a dialog box. We also added an option to export the current scenario, which opens a dialog box for the user to choose the location and format of the resulting dataset.

To enable editing of a scenario, we included a palette of tools that allows the user to add or remove objects, change the parameters of an object, control the sequence of events during the scenario, and dynamically set the scenario background, such as the weather condition, map, lighting, and more.

Overall, our interface provides a user-friendly way to interact with the CARLA simulation and create scenarios for autonomous vehicle testing and development

6 Conclusion

As developing scenario virtualization tools for Autonomous Vehicle (AV) testing is crucial to ensure the safety and efficiency of these systems. In this report, we introduced the motivation behind the development of such tools and presented a high-level overview of the architecture of a Virtualization Framework. We also discussed the importance of different dataset formats used in AV testing and the need for converting between them. Additionally, we highlighted the significance of virtualization and visualization in AV testing and presented CARLA as a widely used simulation platform that provides these features. Overall, the scenario manipulation and generation tool presented in this report aims to provide a flexible, scalable, and efficient solution for generating diverse and challenging scenarios for AV testing.