Enhancing the Realism of Autonomous Driving Simulation with Real-time Co-simulation

Qiwei Chen
chenqiwei@nuaa.edu.cn
The College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
Nanjing, China

Tiexin Wang
tiexin.wang@nuaa.edu.cn
The College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
Nanjing, China

Chengjie Lu
chengjielu@simula.no
Simula Research Laboratory
Oslo, Norway

Tao Yue
tao@simula.no
Simula Research Laboratory
Oslo, Norway

Shaukat Ali
shaukat@simula.no
Simula Research Laboratory
Oslo, Norway

ABSTRACT

Autonomous driving simulators are commonly used to develop autonomous driving systems (ADS) since they provide the flexibility to experiment with scenarios that could even be dangerous in a real setting. This flexibility, however, comes with the possibility of experimenting with unrealistic scenarios. To this end, we present an initial co-simulation framework integrating OpenModelica and CARLA to enable real-time communication between them. As a proof of concept, we experimented with two Modelica models (air resistance and energy consumption). We connected the two models with CARLA to enable real-time communication between them to ensure the realism of scenarios in addition to connecting CARLA with OpenWeather through its API to access real weather conditions. We conducted experiments with a specific virtual electric vehicle (Tesla Model 3) running on the *Town*06 map in CARLA. Results provide preliminary evidence that co-simulation with Modelica models improved the realism of the virtual vehicle in CARLA.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; • **Computing methodologies → Simulation support systems**; *Artificial intelligence.*

KEYWORDS

autonomous vehicles, real-time co-simulation, multi-paradigm modeling

ACM Reference Format:
Qiwei Chen, Tiexin Wang, Chengjie Lu, Tao Yue, and Shaukat Ali. 2022. Enhancing the Realism of Autonomous Driving Simulation with Real-time Co-simulation . In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3550356.3561558

1 INTRODUCTION

In the context of developing and testing autonomous driving systems (ADS), autonomous driving simulators are often employed. This is simply because it is costly and even impossible to test a real ADS in a real-world setting with all possible driving scenarios. Moreover, some scenarios are highly unsafe to be tried in the real-world. To this end, several autonomous driving simulators (e.g., CARLA [5]) are publicly available having different strengths and limitations, as discussed in [14]. For instance, CARLA can be used for end-to-end testing of ADS, but lacks the support of realistic simulation of vehicle dynamics.

Particularly, in the real world, the performance of vehicles is restricted by many factors, such as gas consumption or battery capacity. However, in the virtual world, virtual vehicles are very flexible and might be very unrealistic. For instance, with CARLA and LGSVL [21], the maximum speed of a virtual vehicle is only constrained by a pre-set speed limit in a simulation, and the vehicle can keep driving until reaching the destination or terminating the simulation. Therefore, such a virtual vehicle cannot fully capture the behavior of physical vehicles without considering energy consumption and air resistance, and is therefore unrealistic.

To enhance the realism of virtual vehicles in autonomous driving simulators, we propose to benefit from real-time co-simulation [23] of Modelica models and virtual vehicles, where a real-time connection between open source OpenModelica (an open-source Modelica-based modeling and simulation environment) [8], and CARLA is established. As shown in Figure 1, the overall application context of our approach is about bridging the gap between the physical and virtual worlds. Specifically, Modelica models capture/-model vehicle dynamics from the perspective of mechanical and electrical engineers. A virtual vehicle (which is situated in a virtual environment, i.e., autonomous driving simulators) digitally represents a vehicle in the real world. Moreover, weather conditions (e.g., temperature, humidity) in the autonomous driving environment can be simulated with real weather data of a specific geographical location. The virtual world is therefore represented by the Modelica models (simulating vehicle dynamics), the virtual vehicle (simulating vehicle behaviors such as route planning, steering, braking), and the virtual driving environment (i.e., CARLA).

In this paper, as the initial intent of realizing the aim described above, we propose a co-simulation framework that combines OpenModelica and CARLA. The framework connects two Modelica models with CARLA in real-time. The first model is an energy consumption model adapted from an existing Modelica model (built in MBEV [1]), whereas the second model captures air resistance behavior. In addition, we enabled CARLA to receive real-time weather data from OpenWeather [1] through its API. We conducted two sets of experiments. Within each set, real-time co-simulation between one of the two Modelica models (i.e., energy consumption and air resistance) and CARLA has been evaluated. Results show that the performance, in terms of speed, throttle and so on, of the virtual vehicle in CARLA exhibits significant differences with and without introducing the co-simulation with the Modelica models.

Structure. In Section 2, we provide the background for this work. Section 3 presents our approach, followed by its implementation in Section 4. In Section 5, we present the empirical evaluation and analyze the results. Section 6 gives discussions about this work. Section 7 presents the related work and we conclude the paper in Section 8.

2 BACKGROUND

In this section, we present the necessary background required to understand this paper, i.e., an introduction to Modelica, CARLA, and co-simulation.

Modelica [6, 9, 20] is an object-oriented, multi-domain, graphical, component modeling language for developing complex cyber-physical systems (CPS). It has been widely applied in various domains such as robotics, automotive and satellites. In addition, Modelica models have been applied for supporting simulations and optimizations (e.g., for heating and cooling systems [24] and for picking manipulator [27]). There exist tools that support modeling with Modelica, such as Dymola from Dassault Systems France [3], OpenModelica from the Open Source Modelica Consortium (OSMC) [8] and MapleSim from MapleSoft [13]. For our work, we opted for OpenModelica, which has the graphic and textual model editor, named OpenModelica Connection Editor (*OMEdit*), and *OMPython* for Python scripting. *OMEdit* is the user interface for modeling, simulation, and results plotting. *OMPython* is OpenModelica's Python scripting interface with the capability of dealing with commands and Modelica expressions for the purpose of simulation, plotting, etc.

OpenModelica also supports the Functional Mock-up Interface (FMI) [2] standard, which was designed for supporting model exchange, co-simulation, and most-importantly tool interoperability. More specifically, OpenModelica allows exporting simulation models as Functional Mock-up Unit (FMU) files, which can, subsequently, be imported to other modeling and simulation environments such as Unity. Of course, FMU files can also be imported to OpenModelica. In addition, there is a suite of Modelica libraries for vehicle systems modeling and analysis.

[1]https://openweathermap.org/
[2]https://www.fmi-standard.org/

CARLA (CAR Learning to Act) [5] is an open-source autonomous driving simulator providing various road characteristics such as urban layouts, blocks, and traffic signs. Weather conditions (e.g., time of the day) in CARLA are configurable. In addition, CARLA provides a multitude of vehicle models (i.e., virtual vehicles) and supports a flexible setup of sensor suites. With the help of these sensor suites, the state (e.g., location, speed, and acceleration vectors) of a particular vehicle model can be reflected and analyzed.

Compared with other autonomous driving simulators, e.g., LGSVL and CarMaker, CARLA continuously enhances its capabilities by integrating with ROS, and co-simulating with SUMO [3] and PTV-Vissim [4], etc. Furthermore, CARLA provides open-source code and open digital assets (e.g., urban maps and vehicle models).

Co-simulation. To run a co-simulation involving more than one simulators, a co-simulation scenario and an orchestrator algorithm are needed. Each simulator is capable of consuming inputs, exhibiting behaviors and producing outputs[11]. Co-simulation is not new; many state-of-art methods, such as discrete event-based co-simulation, continuous time co-simulation, and a mix of both exist [10]. Many different projects have benefited from it, and most reports co-simulations with more than two simulators, each of which is oriented toward models from different domains[10]. As identified in [23], the most acknowledged challenge of co-simulation is about standard communication interfaces and protocols among different simulators and the orchestrator algorithm. FMI has been considered as the most promising one for enabling co-simulation.

3 APPROACH

Our overall goal is to enhance the fidelity of autonomous vehicles via real-time co-simulations of OpenModelica and CARLA. As the concept demonstration, in this paper, we particularly focus on two application contexts. First, we introduce energy consumption (modeled and computed/simulated in OpenModelica) to virtual vehicles situated in CARLA, such that their behaviors are more realistic after accounting for the effect of the real-time battery discharging on the vehicle's speed. Second, we introduce air resistance to CARLA via the air density Modelica model, also to improve the fidelity of the virtual vehicle in CARLA. We would like to mention that CARLA's virtual vehicles have "endless" energy (i.e., battery) and behave in an ideal situation of not considering air resistance, which is unrealistic. We aim to make CARLA's virtual vehicles more real.

The overview of our approach is provided in Figure 1. As shown in the figure, through the co-simulation, the Modelica model in OpenModelica amends, in real-time, the virtual vehicle running in CARLA with additional information, as we discussed above. In the rest of the section, we present the two application contexts.

3.1 Enhancing Virtual Vehicle's Behaviors with Energy Consumption Information

In this subsection, we discuss building an energy consumption model, collecting engine data in real-time, and enabling co-simulation.
Building energy consumption model. We first start by developing an energy consumption Modelica model in OpenModelica, which is presented in Figure 2. As shown in the figure, the

[3]https://carla.readthedocs.io/en/0.9.13/adv_sumo/
[4]https://carla.readthedocs.io/en/0.9.13/adv_ptv/

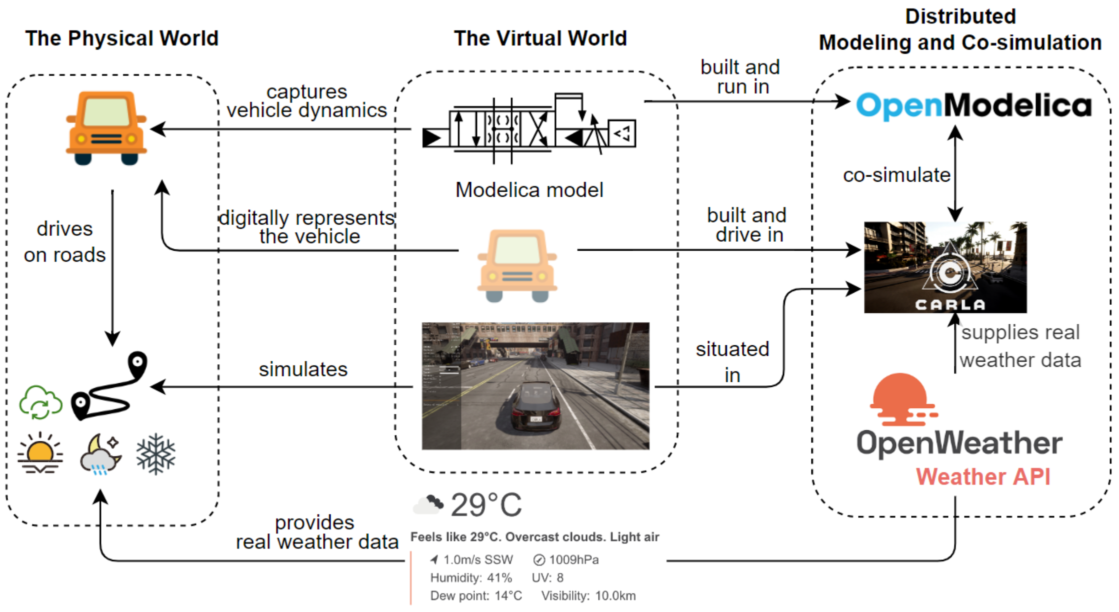


Figure 1: Context and Overview

engineInput block is an instantiation of the *CombiTimeTable* class in the standard library of OpenModelica. This block generates an output signal by linear interpolation in a table like the one shown in Listing 1. The first column contains time points (in *seconds*) and the second and third columns are data being interpolated, i.e., the engine rotating speed values (in rad/s) and the torque values (in $\mathrm{N \cdot m}$). This table is in the format of a txt file called *ReadData.txt*, which is imported by the *engineInput* block. The *toElePow* block then considers the given efficiency map to calculate the power from the engine's mechanical input, and passes the power value to the battery module. The *batt*1 block is the battery block, which discharges the amount of power based on the input value.

```
Time point   Engine rotating speed   Torque
5.00         469.33                  743.0
5.01         469.93                  743.0
5.02         470.52                  743.0
5.03         471.12                  743.0
5.04         471.71                  743.0
5.05         472.3                   743.0
5.06         472.89                  743.0
5.07         473.48                  743.0
5.08         474.08                  743.0
5.09         474.67                  743.0
5.10         475.25                  743.0
```

Listing 1: Example of the Engine Input Table (from CARLA)

Collecting engine data at real-time. CARLA provides Python API, through which we build a virtual world simulating the driving environment. This world is constantly refreshed after a time step. CARLA 0.9.13 provides the *showdebugtelemetry* function, which can display the engine data at each time step, while the vehicle is driving. Inspired by this function, we newly developed a function, named as *get_rpm* (partially shown in Listing 2), and added it to the CARLA source code, which is divided into two parts. The first part opens up memory with a size of 4 bytes when the simulator is turned on. The second part accesses the memory created in the previous step and writes the RPM (Revolutions Per Minute) value of the vehicle's engine to the memory every time the simulator is refreshed. Accessing the memory is fast enough to collect RPM values at each time step. At the same time, each vehicle in CARLA has a customized torque curve. According to the curve and a given RPM value, we can get the corresponding torque value. Then the RPM value can be converted into the rotating speed value when being divided by a constant. At each time step, the latest rotating speed value and torque value are added to the last line of a txt file called *WriteData.txt* via Python API.

Enabling co-simulation. During a simulation, we have another process running in a loop function (the orchestrator of this co-simulation). In each loop, we first read the *WriteData.txt* added by Python API and writes data to the *ReadData.txt* in the *engineInput* block. Then, we run the energy consumption model through OMPython to get the current energy consumption status, including the battery's current and power loss. Finally, these data are fed back to CARLA in real-time, and when the current is close to the limit value, the *emergency_stop* method provided by CARLA's Python API is called. Hence, the Modelica model in OpenModelica can constrain the vehicle's driving behaviors in CARLA.

```
#include <sys/mman.h>
#include <sys/stat.h>

```

```
int fd = shm_open("shared_memory", O_CREAT|O_RDWR, 0666 )
    ;
void *p = mmap(NULL, 4, PROT_READ|PROT_WRITE, MAP_SHARED,
     fd, 0);//open up memory

float rpm = GetVehicleMovementComponent()->
    GetEngineRotationSpeed();//get the RPM value
memcpy(p, &rpm, 4);//write the RPM value to the memory
```

Listing 2: Newly developed *get_rpm* function in CARLA

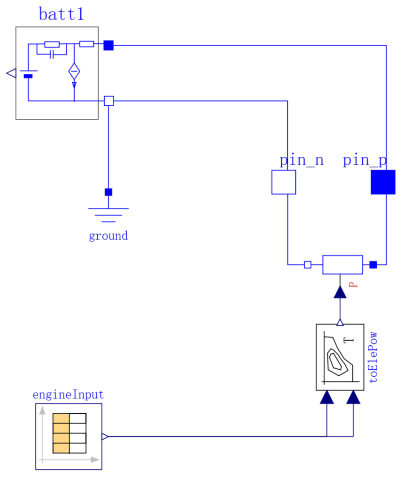


Figure 2: The energy consumption model

3.2 Enhancing Virtual Vehicle's Behaviors with Air Resistance Handling Capability

This subsection discusses building the air density model, collecting real-world data, and enabling co-simulation.

Building the air density model. Air resistance is calculated as $f = 1/2\rho SCxV^2$. In the formula, Cx is the air resistance coefficient, which is related to the material of the force bearing surface of the vehicle and is generally set to 0.3 for most vehicles. S is the windward area of the vehicle, which is generally set to 2.2 for normal vehicles. V is the speed of the vehicle. Finally, ρ is the air density (in kg/m^3). The air density is a dynamic value affected by weather conditions related to temperature and air pressure. The calculation formula is $\rho = P/(R * T)$, P and T are pressure (in Pa) and temperature (in K), R is specific gas constant, equal to 287.05 for dry air. The implementation of the model is given in Listing 3.

```
model AirDensity
  parameter Modelica.SIunits.Temperature T;
  parameter Modelica.SIunits.AbsolutePressure P;
  constant Real R=287.05;
  Modelica.SIunits.Density D;

  function returnDensity
    input Modelica.SIunits.Temperature T;
    input Modelica.SIunits.AbsolutePressure P;
    input Real R;
    output Modelica.SIunits.Density density;
  algorithm
    density := P/(R*T);
  end returnDensity;

equation
    D = returnDensity(T,P,R);

end AirDensity;
```

Listing 3: The air density model

Collecting real-world weather data. We obtain real-world weather data from *OpenWeather*, including information of environmental parameters such as temperature and air pressure, then we store these data in the json file format. During a simulation, we get access to the json file and import weather data to CARLA.

Enabling co-simulation. During a simulation, the weather data is passed to the air density model through OMPython, which outputs the density value. Next, the air resistance value at the current speed is calculated according to the air density, and then the air resistance is added through the *add_force* function in CARLA's Python API.

4 IMPLEMENTATION

In this section, we summarize the technical solution of our work as a whole and present the details with Figure 1. As illustrated in Figure 1, for this work, we employed OpenModelica to build two Modelica models (energy consumption model and air density model) for the purpose of enhancing the realism of dynamics of the virtual vehicles in CARLA. Our energy consumption model integrates some components from the MBEV model in the Simplified Modelling of Electric and Hybrid Vehicles in OMWebBook[1]. Since we are going to implement real-time co-simulation, we need to execute the Modelica model while the vehicle in CARLA is driving. OpenModelica provides different ways to execute Modelica models. In our implementation, we used two ways. The first way is to use OMPython. To be able to link Modelica models with CARLA's Python API, we have to use OMPython. The second way is through OMEdit, which displays simulation results more intuitively as plots compared with other ways.

In addition, we developed a new function in the source code (C++) of CARLA to get real-time RPM values. Moreover, we defined a function in CARLA's Python API to calculate torque values corresponding to the acquired RPM and torque curves. To increase the realism of simulated scenarios, we implemented another function in CARLA's Python API to import weather data from *OpenWeather* via its *Weather API*.

As illustrated in Figure 3, P and Q represent the workflows for enhancing the CARLA simulator with the energy consumption and the air density Modelica models, respectively. In the first workflow, we constructed two txt files. CARLA changes *WriteData.txt* in each time step, and the Modelica model reads *ReadData.txt* in each loop mentioned in 3.1. This ensures that when the Modelica model reads the file, another process will not change the file. Furthermore, because we deployed CARLA and OpenModelica on the same server, the communication between CARLA and OpenModelica is through Python API and files.

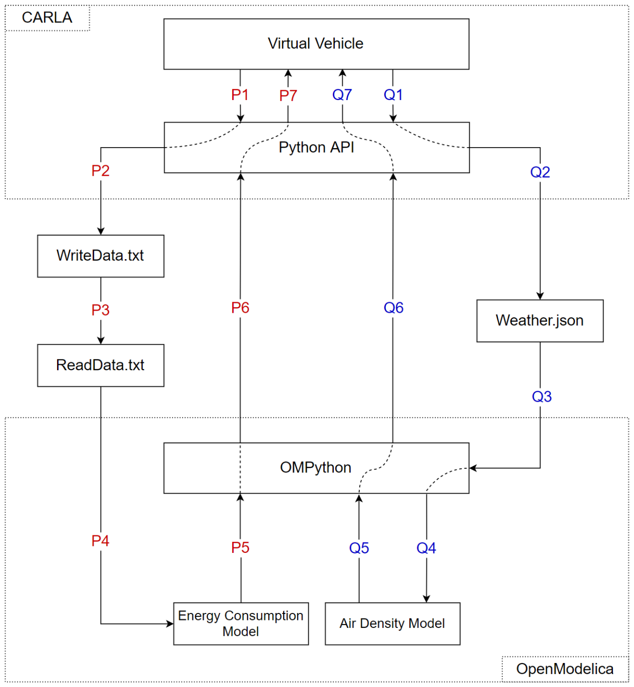


Figure 3: Enhanced simulator workflow with two models

5 EMPIRICAL EVALUATION

In this section, we first present the research questions aiming to answer (Subsection 5.1), followed by the experiment setting (Subsection 5.2) and design (Subsection 5.3 and Subsection 5.4), and results and analyses (Subsection 5.5).

5.1 Research Questions

To evaluate our approach, we design experiments to answer the following two RQs:

- RQ1: Does our approach effectively constrain virtual vehicles' behaviors in CARLA with energy consumption information from the Modelica model?
- RQ2: Does the real-time co-simulation support our approach effectively enhance the capability of virtual vehicles in CARLA to deal with air resistance?

5.2 Experiment Settings

This section provides settings of CARLA (and its ADS), OpenModelica, and driving scenarios.

Settings for CARLA simulator. For our experiments, we selected the *Town*06 map in CARLA, which is a long highway with many entries and exits. The virtual vehicle we used is Tesla Model 3 (an electric vehicle). As discussed in Section 3, the inputs of the energy consumption module at each time step are rotating speed and torque values. However, CARLA does not directly provide torque values. Instead, it provides a torque curve for each vehicle with the x-axis being RPM and the y-axis being torque. For our experiments, we got the torque curve of Tesla Model 3 in CARLA. Through this curve, we can get the torque value according to the RPM value at each time step. In addition, we choose CARLA's simulation mode as the synchronous one (otherwise than asynchronous) to collect data at each time step with the time duration being 0.01 seconds.

Our vehicle is controlled by the CARLA's behavior agent (ADS), a Python class in Python API, which can be modified by users. The agent can be configured with different maximum speed limits.

OpenModelica settings. To build and execute our Modelica models, we employed the latest stable version of OMEdit and OMPython on the Ubuntu Server. We can execute our Modelica models through the *simulate* method in OMPython.

Driving Scenario. To answer RQ1, we chose a urban driving scenario with a five-lane straight road between the start and the destination on *Town*06 map (Figure 4). The vehicle accelerates from the starting point and drives to the destination under the control of the CARLA's ADS. Note that the vehicle is expected to autonomously follow traffic rules and take actions such as lane-switch to avoid collisions. For RQ2, we chose the same scenario as RQ1 except that the distance between the destination and the starting point is farther.

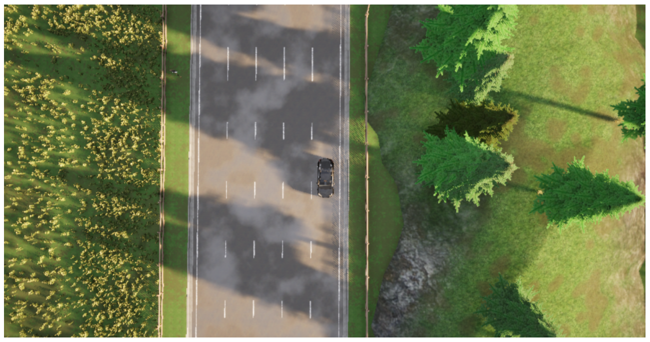


Figure 4: Employed driving scenario in our experiments

5.3 Experiment Design for Answering RQ1

To answer RQ1, we conducted two experiments. The first experiment simply lets the virtual vehicle run in CARLA and we collect the RPM value of the engine at run-time. In the second experiment, we collect RPM values and also apply our approach to enable real-time co-simulation between the energy consumption Modelica model and CARLA to constrain the speed of the virtual vehicle. More specifically, each collected RPM value at each simulation time step is used as the input to the energy consumption Modelica model, which consequently generates current values of the battery. Current values are then transferred, at real-time, to CARLA. CARLA subsequently adjusts the speed of the virtual vehicle to ensure that it does not go over the speed limit.

In both of the experiments, the speed limit of the vehicle was set to 60km/h, which is the maximum speed limit on many urban roads. The charge of the battery module of the energy consumption model (Section 3.1) was set to 100*3600, which is the default value for the MBEV model.

During the execution of the two experiments, we also collected driving speed (measured in km/h) of the virtual vehicle from the Python API in CARLA. After the execution of the two experiments, we run the energy consumption model in OMEdit with the rotating speed values and torque values collected from the two experiments as input of *engineInput* block and derive a plot of the battery current (measured in A) in the simulation results.

5.4 Experiment Design for Answering RQ2

To answer RQ2, we also conducted two experiments: with and without our approach applied to enable real-time co-simulation between air density model and CARLA. For these two experiments, to set up the environment of the driving scenario in CARLA, we employed real-world weather data of the Nanjing city on January 8, 2022. This is because, as discussed in Section 3.2, the air density model calculates air density based on weather conditions. We also collected the velocity (V) of the vehicle at each simulation time step, which is represented as a three-dimensional vector along x, y and z axies (v_x, v_y, v_z). We used the formula $f = 1/2\rho SCxV^2$ (Section 3.2) to calculate the air resistance.

In addition, because the weather in the real world generally changes slowly, it is not wise to simulate the time in the simulated world exactly the same as in the real world, we replace every hour in the real world with 100 time steps of the simulated world, which is 10 seconds. We set four different maximum speed limits for the vehicle, which are 80km/h, 90km/h, 100km/h, and 110km/h. Due to the default settings of CARLA's ADS, the maximum speed of the vehicle will be 2km/h less than the set maximum speed.

Each experiment was performed 30 times to account for the randomness. During the two experiments, we collected values of the speed and throttle of the vehicle via Python API. Note that the value range of the vehicle's throttle in CARLA is between 0 and 1.

5.5 Results and Analyses

This section presents results for RQ1 and RQ2 in Section 5.5.1 and Section 5.5.2 respectively.

5.5.1 RQ1. Figure 5 illustrates the real-time changes in the battery's current of the two experiments conducted to answer RQ1. We see that the simulation on the Modelica side terminates after 4 seconds. This is simply because the battery's current exceeded the allowed upper limit at the fourth second of the simulation. In other words, the vehicle runs faster in CARLA than the battery could be able to support, revealing a very unrealistic simulation scenario. On the other hand, after applying our approach, the Modelica model was able to provide real-time feedback to CARLA, which consequently was able to reduce the vehicle's throttle and keep the vehicle's speed within the battery's current tolerance range (as shown in Figure 6), which is below 1000A (as shown in Figure 5).

We would also like to acknowledge that, in our current implementation of the co-simulation, the current value outputted by the Modelica model is fed back to CARLA, then the vehicle's speed is reduced when the current is about to reach the limit value (as shown in Figure 6), which, by no means, an optimal solution. However, we argue that this work only aims to demonstrate the importance and necessaries of enabling the real-time co-simulation of Modelica and CARLA, but not optimize the vehicle planning and control.

5.5.2 RQ2. Figure 7 presents the speed changes of the vehicle over the time with the four different speed limits, blue dots are without

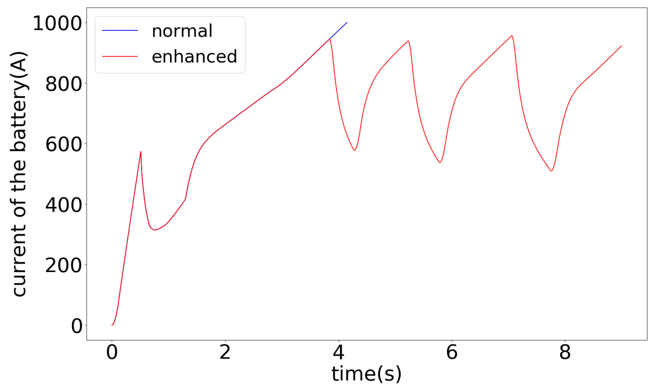


Figure 5: Runtime battery discharging of the vehicle

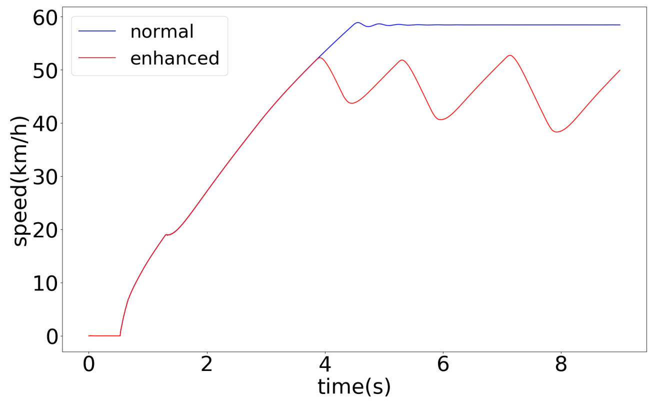


Figure 6: Runtime speed of the vehicle

air resistance and red dots are with. Each of the sub-figures is the scatter plot of 30 runs of the normal driving and air resistance enhanced driving experiments at each time step under each corresponding speed limit. The figures show that CARLA performed very stably across the 30 runs and at each time point differences in the speed of the 30 runs are very minor, as one can hardly notice variations.

One can notice from the figure that along with the speed limit increases, the impact of the air resistance on the vehicle speed increases, which is reflected as the time needed for the vehicle to speed up from 0 to the pre-set speed limit. More specifically, when the speed limit is 80km/h, not much difference can be observed between the two experiments (with and without introducing air resistance) when the vehicle accelerates from 0 to the maximum speed. But, the difference becomes increasingly noticeable when the speed limit goes up to 110km/h, implying that the vehicle took longer time to reach the speed limit with air resistance. We counted the average speed at each time point of the 30 runs with and without introducing air resistance under the four speed limits, Then we counted the average of speed difference at each time point from the 5th to 15th second since the speed of the vehicle with or without introducing air resistance is almost the same before the 5th second and after the 15th second. From Table 1, we can also see that along with the increase of the speed limit, the effect of air resistance increases, as the average speed difference increases from 0.296 all the way to 2.732 when the speed limit goes from 80km/h to 110km/h. The above results are consistent with cognitive common sense, that is, the faster the moving object, the stronger the feeling of air resistance, and the greater the impact of air resistance on the object, e.g., reducing the speed of the movement.

We also performed the Mann and Whitney statistical test [18] to assess statistical significance of the average speed without and with air resistance, and calculated the Vargha and Delaney metric $\hat{A}_{12}$ for effect size. We chose the significance level of 0.05 for all the tests. Results show that for all the speed limits, the p-values are less than 0.05 and $\hat{A}_{12} > 0.5$, suggesting that the average speed without air resistance is significantly larger than with air resistance.

Speed limit	80km/h	90km/h	100km/h	110km/h
Difference (km/h)	0.296	0.730	1.551	2.732

Table 1: Average speed differences of the vehicle (without air resistance - with air resistance)

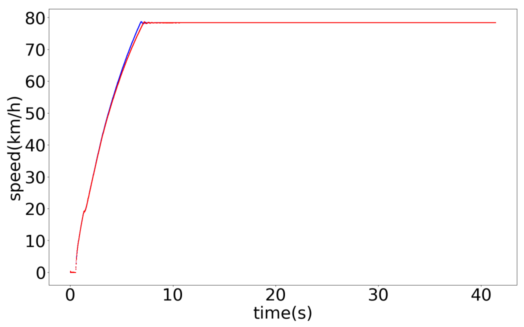
Figure 7 demonstrates the influence of air resistance on the vehicle accelerating to the maximum speed. However, it does not help to distinguish the normal driving and air resistance enhanced driving when the vehicle reaches the maximum speed. Nonetheless, it is clear that, in both experiments, when the vehicle reaches the maximum speed, its throttle values should be different. More specifically, the vehicle's throttle should be a little higher with air resistance because part of the throttle is for offsetting the air resistance. Therefore, in Figure 8, we plot throttle values to visualize the difference. It is clear from the figure that the normal driving exhibits lower throttle values than the air assistance enhanced driving in all the selected speed limits. In addition, one can observe that at a higher driving speed, more throttle is required to offset the air resistance. Therefore, even though we saw a drop in the vehicle's throttle stability after adding air resistance, we think this drop is acceptable.

6 DISCUSSION

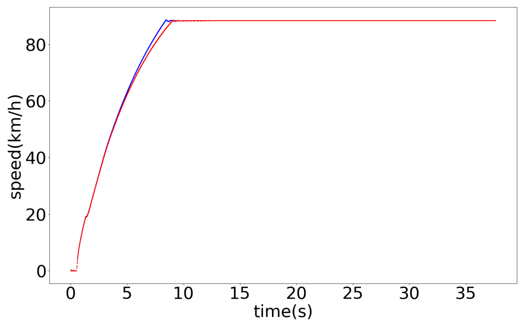
This section provides our key findings and discussions based on the experiment results.

Adapting to different simulators. We selected CARLA as the subject simulator and enhanced its fidelity via co-simulation with Modelica models. Moreover, our approach can be extended to other simulators, e.g., LGSVL [21]. Since LGSVL provides a Python API similar to CARLA, we can adapt a very similar technical solution to build a real-time co-simulation between OpenModelica and LGSVL as discussed in Section 3.

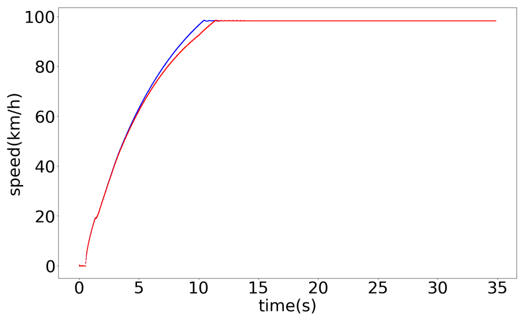
Testing autonomous vehicles with enhanced simulations. In practice, testing autonomous vehicles usually relies on simulations, and the fidelity of virtual vehicles in the simulations has non-negligible influence on a simulation-based testing approach. Based on the evaluation results, we found that an autonomous vehicle's speed and throttle significantly differ between enhanced and un-enhanced simulations. Hence, it is important to have an approach like ours to help construct testing scenarios to test autonomous vehicles in more realistic simulations.



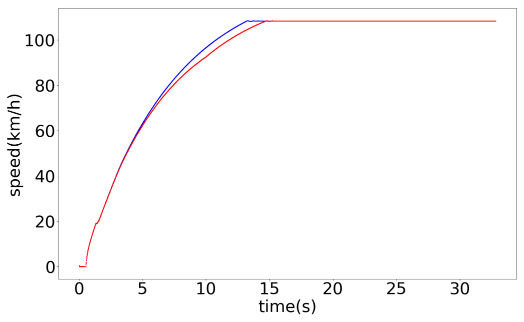
(a) The speed limit is 80km/h



(b) The speed limit is 90km/h



(c) The speed limit is 100km/h



(d) The speed limit is 110km/h

Figure 7: Speed changes of the vehicle under the simulations of different speed limits

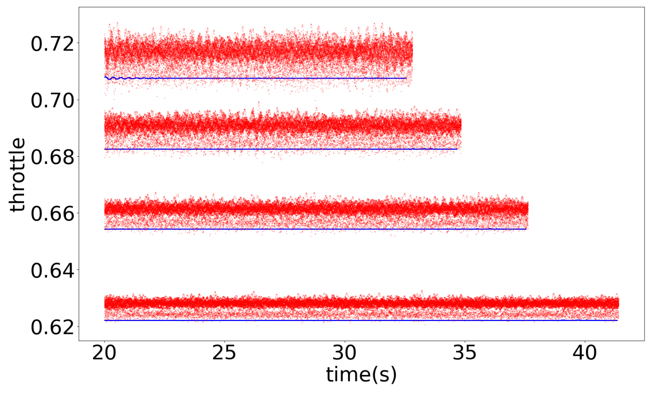


Figure 8: Throttle changes of the vehicle under the simulations of different speed limits

Real-time co-simulation vs. static co-simulation Before conducting the experiments reported in this paper, we also performed similar experiments on LGSVL, an autonomous driving simulator based on Unity. Unity provides an interface to support *FMI* and can directly import *FMU* files. At the same time, OpenModelica provides the function of outputting models as *FMU* files, so we can output Modelica models as *FMU* files and directly import them into Unity, which is very convenient. However, the limitation is that when we want to use different weather data, we need to import the *FMU* files again, which is contrary to our idea of enabling real-time simulation.

Impact of network communication protocols on simulations. As said earlier, we tried out similar experiments with LGSVL, on which Apollo was deployed. With these experiments, we also enhanced the virtual vehicle in terms of dealing with the air resistance in LGSVL. However, Apollo communicates with LGSVL with the UDP protocol through the bridge module, but this communication can be easily affected by network conditions and network conditions consequently affect simulation results. Therefore, we did not choose LGSVL as the simulator for the experiments reported in this paper.

Cost, Effectiveness, and Scalability One of the primary efforts to apply our approach is to build the Modelica models. However, such cost is typical when developing and testing autonomous vehicles. Also, note that there already exist model libraries in OpenModelica that can already be used to enhance simulators. Even if such modeling is required to be done from scratch, it pays off in terms of ensuring the quality of the ADS being developed. Moreover, such models can also be reused for different ADSs, thereby reducing this effort. However, we acknowledge that more dedicated experiments are needed to assess the cost and effectiveness of our approach in the future. Similarly, in terms of scalability, we need more detailed experiments in the future since we only demonstrated that our approach works with only two models and connection with CARLA.

7 RELATED WORK

As a modeling language, Modelica provides the capability of modeling mechanical, electrical and electronic components of complex CPS. In the automotive industry, Modelica has been commonly applied. For example, towards the development of small electric vehicles, as reported in [12], both vehicle dynamics and energy consumption have been taken into consideration when developing a full vehicle model based on the Vehicle Dynamics Library (VDL) of Dymola for Toyota. Focusing on e-tron (the Electric drive Audi sports car), the Modelica Standard Library was adopted to model the electric power steering system based on Dymola [4]. Towards the design and implementation of an automatic commercial vehicle transmission platform (TraXon) of TF, the authors of [15] proposed a Modelica library and a modular transmission model based on Dymola. In [7], a Modelica library was proposed for modeling electrified powertrain components. In summary, Modelica and its simulation environments have been adopted in industry for years, while more and more in-house Modelica libraries are increasingly being built.

Autonomous driving, as an important research and development direction of the automotive industry, attracts attentions from both academia and industry. To verify the key characteristics (e.g., safety, energy consumption and comfortableness) of Advanced Driver Assistance Systems (ADAS) or ADS of autonomous vehicles, co-simulation [23] has been adopted to simulate reality (with diverse kinds of simulators focusing on driving environments, virtual vehicles, etc) while reducing the costs, such as in generating driving scenarios and potential damages to real vehicles. Soma [25] designed and implemented the parallel operation of the Drag Force Modelica model and CARLA, and proved that the Drag Force model can return actions to CARLA through speed comparison experiments. The authors of [19] proposed a co-simulation framework combining IPG CarMaker (model vehicle dynamics) and VISSIM (a traffic flow simulation software) through a Matlab GUI, which specifically focuses on creating realistic traffic environment for ADS testing. Replacing VISSIM with SUMO, the authors of [17] proposed a co-simulation framework for cooperative driving functions. Towards the design and evaluation of ADAS, a co-simulation framework that combines Modelica models (vehicle dynamics), Simulink models (vehicle control) and Unity (environmental conditions) was proposed in [26]. To better integrate these models, a set of modeling tools Dymola, Simulink and Unity 3D and the integration tool OpenMETA have been used in [26]. Based on a real (physical) industrial logistic self-driving vehicle, the authors of [22] built a wheel model (with Modelica) to reflect and compute bore friction. With *FMI* and *FMU*, this model was deployed as ROS node and validated on the ActiveShuttle Dev Kit prototype. Focusing on the control of electric vehicles, in [2], a co-simulation platform that combines CarSim (simulator) and Simulink. Also by combining CarSim and Simulink, the authors of [16] carried out a co-simulation towards a self-built in-wheel motor drive vehicle. With this real vehicle, the authors performed tests on a specific campus road.

As a key enabling technique, co-simulation is widely used for testing and verification of ADSs. Diverse kinds of simulators (e.g., CARLA), vehicle dynamic models (built with Modelica) and vehicle control functions (modeled with Simulink) were (or being) developed by both academia and industry. While the mechanism of combined use of these models and tools are being proposed by researchers towards different practical objectives, such as finding safety-critical driving scenarios.

Towards a similar research target, Soma [25] introduced an existing Drag Force Modelica model into CARLA, whose feedback to CARLA was to change the speed of the vehicle. Compared with our approach, Soma's approach is not real-time, which could be easily affected by CARLA's autopilot mode. Also, the real weather conditions was not introduced.

Considering the real-time communication issue (among different simulators), some existing co-simulation works (e.g., [19]) achieve this by synchronously deploying and executing the simulators (by sharing corresponding information such as road network topology), while other works try to achieve real-time communication by sending real-time data (messages), for example via UDP [22] among simulators. In our approach, we achieve real-time co-simulation by instant messaging through Python API, which takes the simulation cycle into account.

8 CONCLUSION

Simulators play a key role in the development and testing of autonomous driving systems (ADS). Therefore, co-simulation that combines various simulation units (e.g., vehicle dynamics models, vehicle control models, and physical environment) is necessary to improve the realism of simulations. To enhance the fidelity of autonomous vehicles in virtual environment, we proposed a co-simulation framework integrating OpenModelica and CARLA. By employing OpenModelica, two Modelica models are developed for computing energy consumption and air resistance. By solving several technical challenges (e.g., modifying source code of CARLA, integrating third-party plug-ins), we integrated these two models with CARLA in real-time. Two sets of experiments were conducted to verify our co-simulation framework. Experiment results show that the co-simulation framework we proposed improved the realism of the virtual vehicle in CARLA. In our future work, we will consider more about the physical reality (e.g., build consistent connections among physical vehicles, virtual vehicles and Modelica models), extend the co-simulation framework to optimize the planning and control of virtual vehicles and combine our approach with simulation-based autonomous vehicles testing.

ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China under Grant No. 61872182. It is also partially supported by the Co-evolver project (No. 286898/F20) and the Co-tester project (No. 314544) funded by the Research Council of Norway.

REFERENCES

[1] Massimo Ceraolo. 2017. *Simplified Modelling of Electric and Hybrid Vehicles.* http://omwebbook.openmodelica.org/SMEHV
[2] H. Cui, C. Quan, Q. Xing, and W. Hu. 2016. Electric vehicle differential system based on co-simulation of Carsim/Simulink. In *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA).*
[3] M. Dempsey. 2007. Dymola for Multi-Engineering Modelling and Simulation. In *IEEE Vehicle Power & Propulsion Conference.*
[4] Andreas Deuring, Johannes Gerl, and Harald Wilhelm. 2011. Multi-Domain Vehicle Dynamics Simulation in Dymola. In *International Modelica Conference.*
[5] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. In *Conference on robot learning.* PMLR, 1–16.
[6] H. Elmqvist, S. E. Mattsson, and M. Otter. 1999. Modelica-a language for physical system modeling, visualization and interaction. In *IEEE International Symposium on Computer Aided Control System Design.*
[7] Nikolaos Fotias, Ran Bao, Hui Niu, Michael Tiller, Paul McGahan, and Adam Ingleby. 2021. A Modelica Library for Modelling of Electrified Powertrain Digital Twins. In *Modelica Conferences.* 249–261.
[8] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, and A. Sandholm. 2006. OpenModelica - A free open-source environment for system modeling, simulation, and teaching. In *IEEE International Symposium on IEEE Conference on Computer Aided Control System Design, IEEE International Conference on Control Applications.*
[9] P. Fritzson and P. Bunus. 2002. Modelica - a general object-oriented language for continuous and discrete-event system modeling and simulation. In *ANNUAL SIMULATION SYMPOSIUM.*
[10] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2017. Co-simulation: State of the art. *arXiv preprint arXiv:1702.00686* (2017).
[11] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2018. Co-simulation: a survey. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–33.
[12] Yutaka Hirano, Shintaro Inoue, and Junya Ota. 2014. Model-based Development of Future Small EVs using Modelica. In *International Modelica Conference.*
[13] J Hřebíček and Martin Řezáč. 2008. Modelling With Maple And MapleSim. (2008).
[14] P. Kaur, S. Taghavi, Z. Tian, and W. Shi. 2021. A Survey on Simulators for Testing Self-Driving Cars.
[15] Jochen Köhler, Michael Kübler, and Julian King. 2014. Transmission Modeling in Modelica: A consistent approach for several software development platforms. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden.* Linköping University Electronic Press, 259–264.
[16] Y. Li, H. Deng, X. Xu, and W. Wang. 2019. Modelling and testing of in-wheel motor drive intelligent electric vehicles based on co-simulation with Carsim/Simulink. *IET Intelligent Transport Systems* 13, 1 (2019), 115–123.
[17] Viktor Lizenberg, Mhd Redwan Alkurdi, Ulrich Eberle, and Frank Köster. 2021. Intelligent Co-Simulation Framework for Cooperative Driving Functions. In *2021 IEEE 17th International Conference on Intelligent Computer Communication and Processing (ICCP).* IEEE, 109–115.
[18] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.
[19] Demin Nalic, Arno Eichberger, Georg Hanzl, Martin Fellendorf, and Branko Rogic. 2019. Development of a Co-Simulation Framework for Systematic Generation of Scenarios for Testing and Validation of Automated Driving Systems. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC).* 1895–1901. https://doi.org/10.1109/ITSC.2019.8916839
[20] U. Objectoriented, P. Fritzson, and V. Engelson. 1998. Modelica — A unified object-oriented language for system modeling and simulation. In *European Conference on Object-oriented Programming.*
[21] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. Lgsvl simulator: A high fidelity simulator for autonomous driving. In *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC).* IEEE, 1–6.
[22] Nikolas Schrder, Oliver Lenord, and Ralph Lange. 2019. Enhanced Motion Control of a Self-Driving Vehicle Using Modelica, FMI and ROS. In *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019.*
[23] G. Schweiger, C. Gomes, G. Engel, I Hafner, J. Schoeggl, A. Posch, and T. Nouidui. 2019. An empirical survey on co-simulation: Promising standards, challenges and research needs. *Simulation modelling practice and theory: International journal of the Federation of European Simulation Societies* 95 (1 2019), 148–163.
[24] G. Schweiger, Per Ola Larsson, F. Magnusson, P. Lauenburg, and S. Velut. 2017. District heating and cooling systems - Framework for Modelica-based simulation and dynamic optimization. *Energy* 137, oct.15 (2017), 566–578.
[25] Øyvind Soma. 2021. *Prototyping Connection Between Digital Twin and Physical Twin for Autonomous Driving to Support Experimentation.* Master's thesis.
[26] Masahiro Yamaura, Nikos Arechiga, Shinichi Shiraishi, Scott Eisele, and Theodore Bapty. 2016. ADAS Virtual Prototyping using Modelica and Unity Co-simulation via OpenMETA. In *Deployment of High-fidelity Vehicle Models for Accurate Real-time Simulation.*
[27] C. Yan, J. Shuang, X. J. Zou, D. F. Xu, and W. L. Cai. 2009. Study of Modeling and Simulating for Picking Manipulator Based on Modelica. *Springer, Berlin, Heidelberg* (2009).