# An hybrid simulation tool for autonomous cars in very high traffic scenarios

Mario Garzón, Anne Spalanzani

# An hybrid simulation tool for autonomous cars in very high traffic scenarios

Mario Garzón[1] and Anne Spalanzani[1]

***Abstract*— This article introduces an open source tool for simulating autonomous vehicles in complex, high traffic, scenarios. The proposed approach consists on creating an hybrid simulation, which fully integrates and synchronizes two well known simulators: a microscopic, multi-modal traffic simulator and a complex 3D simulator. The presented software tool allows to simulate an autonomous vehicle, including all its dynamics, sensors and control layers, in a scenario with a very high volume of traffic. The hybrid simulation creates a bi-directional integration, meaning that, in the 3D simulator, the ego-vehicle sees and interacts with the rest of the vehicles, and at the same time, in the traffic simulator, all additional vehicles detect and react to the actions of the ego-vehicle. Two interfaces, one for each simulator, where created to achieve the integration, they ensure the synchronization of the scenario, the state of all vehicles including the ego-vehicle, and the time. The capabilities of the hybrid simulation was tested with different models for the ego-vehicle and almost 300 additional vehicles in a complex merge scenario.**

## I. INTRODUCTION

The development of fully autonomous vehicles has been a very important area of research and development in the last few years and it continues to grow and develop nowadays. However, introducing and testing new, high-level, algorithms remains a challenging task. The reason for that is the implicit risk, complexity and variability of both the vehicles and the scenarios involved in such tests.

This work aims to facilitate the development and testing of these algorithms by introducing an hybrid simulation, which allows to have a scenario where a realistic model of an autonomous vehicle can operate in a road network where, in addition to crossings, traffic lights and other road elements, a large number of additional vehicles is present.

The definition of hybrid simulation can be very broad, however, in general terms it can be understood as the use of two or more models or simulation types that, when combined, can allow a more complex simulation or provide a better insight of the system being simulated [1]. In this work, the hybrid simulation is created via the full integration and synchronization of two well known simulation software: Gazebo and SUMO. The first one is powerful 3D simulation environment for autonomous robots [2], whereas the second is a microscopic and continuous road traffic simulation package designed to handle large road networks [3].

Although both SUMO and Gazebo can be used by themselves, none of them can compile with the required complexity for testing high level algorithms in autonomous vehicles. Moreover, the two simulators have complementary capabilities: on one hand, SUMO can provide a very high volume of traffic with a realistic behaviour, however, it can not simulate dynamics, different sensors or more complex vehicle models. On the other hand, Gazebo allows for a complex 3D simulation, any vehicle, sensor or component can be modelled with very high detail, its drawback is that in order to simulate a large number of vehicles it will be required to model and control each one individually, thus largely increasing computational costs and complexity of the simulation, furthermore, driver models and other traffic-specific tools are not easily available. By integrating the two simulators it is possible to obtain the best of both worlds and overcome their weak points.

The main purpose the proposed hybrid simulation is to contribute on the improvement of the capabilities of autonomous cars, by allowing to test the models, controllers and high level algorithms in very complex situations, such as the merging or lane changing during traffic jams, driving in mixed roads, or crossing an intersection with high traffic among many other situations. The common characteristic of these scenarios is that, in all of them, it is necessary to have a large number of other vehicles, pedestrians or bikes, in close interaction with the vehicle being tested or ego-vehicle. All vehicles in the simulation should response to the movements of the ego-vehicle, and vice-versa, which is the main difficulty and novelty of the proposed approach. Another important characteristic of this work, is that since the ego-vehicle is simulated in Gazebo, it is fully compatible with many open libraries, as well as the widely used ROS framework, therefore making it fully compatible with a large number of developments over the world, and reducing the simulation-to-reality gap by facilitating the portability of the developed algorithm to real world systems.

There has been a plethora of simulators for autonomous cars developed over the past years, most of them focused on car-racing or other driver-oriented experiences [4], [5]. However, those simulators may not provide the information or control over the agents required for testing applications of autonomous vehicles. More recently, new simulators have been developed, with a strong focus on autonomous driving. However, they are either strongly keep a secret[1], or they are focused on single applications [6]. A good approach, has been developed by Dosovitskiy et al. [7], however it is still limited on the number vehicles in the scene and it can only simulate some pre-defined vehicles, making it not suitable

[1] The authors are with Univ. Grenoble Alpes, Inria, Grenoble INP, 38000 Grenoble, France - mario.garzon-oviedo@inria.fr, anne.spalanzani@inria.fr

[1] https://www.theatlantic.com/amp/article/537648/

for testing any other model, control schema or detection algorithm developed outside of the scope of the simulator. Moreover, in contrast with some of the previous approaches, both the architecture and the source code proposed here, are open and available[2]. Therefore, they can be used and adapted to any autonomous vehicle simulated in Gazebo.

The remainder of this paper is structured as follows Section II presents a brief overview of the proposed methodology. Section III briefly describes the simulators and other tools used. Section IV details the implementation and the tools used for the hybrid simulation. Section V describes the scenarios and experiments used to test the capabilities of the system and, finally, Section VI presents the concluding remarks.

## II. METHODOLOGY OVERVIEW

This section presents a brief overview of the methodology proposed for the hybrid simulation. As aforementioned, two different simulators are used at the same time, each one of them provides different capabilities, and their combination results in one single hybrid simulation, which can enhance the capabilities of the two simulators.

In order to achieve a complete integration, it is necessary to unify or synchronize different aspects of the two simulations, namely: The scenario; The number and type of vehicles, pedestrians and bikes present on the simulation at each time; The state (*i.e*. behaviour, position and speed) of those vehicles, including the ego-vehicle; and finally, the time. An overview of how the methodology for this integration is presented in Figure 1, and will be explained next.

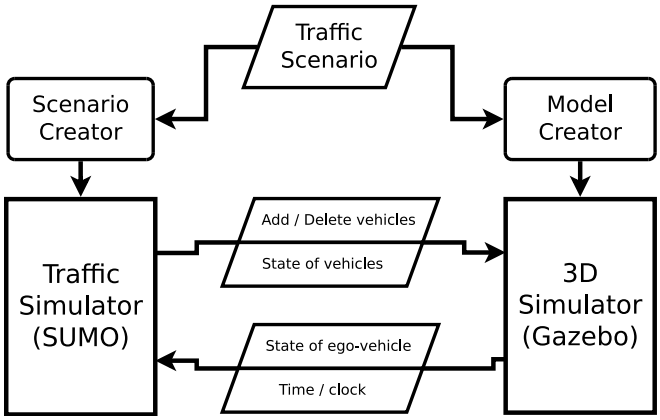



Fig. 1: Overview of the proposed methodology for hybrid simulation.

The first item that needs to be put in common is the scenario. It consists on a set of roads, defined by their types, shapes and lengths, it also includes the intersections, traffic lights, circulation restrictions and any other non-changing element or rule, that could affect the behaviour of the vehicles. The scenario needs to be defined beforehand, and it should be the same for both simulators. Once defined, the scenario needs to be interpreted by the simulators, this means that two different files, a *network file* and a *world file* should be created for SUMO and Gazebo respectively. Both of them are XML-type files, which can be created using different tools, the most common for SUMO is its *NETEDIT* graphical interface, whereas for Gazebo it is usually created directly in its graphical interface.

Once the scenario is established, the next step is to start adding the vehicles to the simulation. In this step there is an important separation: the ego-vehicle is modelled and spawned directly as a Gazebo model and it remains in the scene trough all the simulation; all other vehicles, on the other hand, are spawned or deleted in Gazebo when they are created or removed in SUMO, moreover, the position where those vehicles are introduced, is also controlled by SUMO.

The next step is the synchronization of the status of the vehicles in both simulators. As aforementioned, only the ego-vehicle is fully simulated in Gazebo, therefore its state is sent to SUMO. For all the other vehicles, the process goes the other way around, their actions, including reactions to movements of the ego-vehicle, are simulated in SUMO and their status is sent Gazebo.

The time is the final item that requires synchronization, and of course, it's crucial for obtaining a correct simulation. In the proposed approach, the time is generated by Gazebo, and is controlled in SUMO by using a step-by-step simulation, thus allowing a correct synchronization.

## III. SIMULATORS AND SOFTWARE TOOLS

This sections gives a very broad description of the two simulators used, as well as the main reasons to select them. It also describes the *Traci* interface, which is a key tool for this integration.

### A. SUMO Traffic simulator

The Simulation of Urban MObility (SUMO) simulator provides a microscopic, multi-modal traffic simulation. Its an open source software, that has a strong support, as it was developed by the DLR and it has been used for many projects worldwide. It's compatible with openstreetmaps and other map engines, making it easy to create realistic scenarios. SUMO performs a purely microscopic simulation, which means that each vehicle has his own explicit model and route, moreover, each vehicle individually moves through the network and reacts their surroundings.

There are many reasons for selecting this traffic simulator. Firstly because it provides a very realistic simulation, which is achieved by including very complex driver models, which can be highly parametrized, as well as allowing different types of vehicles, pedestrian and bikes. Moreover, it provides different tools for creating or importing networks and for configuring the simulation.

### B. Gazebo 3D simulator

The Gazebo 3D simulator has the ability of simulating robots in complex indoor and outdoor environments It offers

[2]Code available at `https://bitbucket.org/marioney/hybrid_simulation`

physics simulation with a high degree of fidelity, as well as a plethora of sensors and additional elements. It also provides different interfaces for both users and programs. It's widely used for designing robots and testing algorithms in realistic scenarios. It can work with different physics engines (e.g. ODE, Bullet, Simbody, DART), and provides a large library of robot models and environments, moreover, any robot can be modelled for its use in the simulator.

The selection of this simulator, is mainly due to its widely use within the robotics community, and also because it provides all the flexibility and robustness required for the hybrid simulation. Moreover, it offers the different options for adding additional vehicles, buildings, traffic lights and many other elements. Moreover, the ego-vehicle modelled for gazebo allows to include sensors and components similar to those found in the real world vehicle, thus reducing the effort needed for translating the developments to real world robots.

### C. TraCI

Finally, the Traffic Control Interface (TraCI), which is used for controlling the SUMO simulation, will be explained. TraCi allows real time control of the simulation, it can be used for retrieving values of simulated objects and for manipulating their behaviour "on-line".

This interface uses a TCP based client/server architecture, where SUMO acts as server and the control from gazebo acts as client. It provides commands for controlling the simulation, the traffic lights and other elements of the scenario and of course the vehicles. Moreover, using TraCI it is possible to start, pause or control the simulation step-by-step, therefore it's the ideal tool for achieving the required level of integration. Finally, a python library is also provided, thus facilitating its use in combination with the ROS framework and therefore with Gazebo.

## IV. IMPLEMENTATION DETAILS

This section explains the details of the implementation of the proposed approach for hybrid simulation.

As mentioned in Section II there are different items that need to be fully synchronized. to achieve this, two different algorithms where developed, a TraCI interface for SUMO and a Plugin for Gazebo.

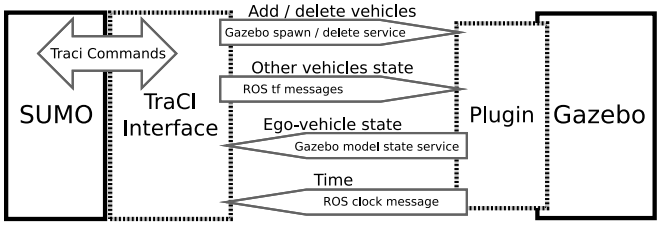



Fig. 2: Implementation details for the hybrid simulation.

A graphical representation of the data provided by each program, as well as the type of message used in each case in presented in Figure 2. It can be seen that the TraCI interface has two sides of communication. On the first side, it performs a bi-directional interaction with SUMO, and it does so by using different TraCI commands. On the second side, it communicates with the Gazebo plugin, and it does so by using ROS messages and services. For the Gazebo plugin, since it runs embedded on the simulator, there is no need for message passing with the simulator, whereas the communication with the TraCI interface is also based on ROS messages and services. Both algorithms will be explained in detail in the remainder of this section.

### A. SUMO Interface

The main objective of the SUMO interface is to control the execution flow of the traffic simulation, and to allow the message passing towards gazebo. As aforementioned, it transforms TraCI commands to ROS messages or services and vice-versa. Furthermore, it also controls the step-by-step execution of the simulation by using a ROS timer. Furthermore, since this timer uses the Gazebo-simulated clock, the temporal synchronization of both simulators is ensured. The interface, which was written in python, is summarized in Algorithm 1 and explained next.

**Input:** $F = \{f_1, f_2, \ldots, f_n\}$ : Vehicle flow
**Input:** $t$ : Time-step
$r$ : route file $\leftarrow createRouteFile(F)$
$startSimulation(r, t)$
$startTimer(t)$ : ROS-controlled timer
**TimerLoop**
  $evs$ : ego-vehicle state
  $evs \leftarrow getGazeboModelState()$
  **if** $evs \neq NULL$ **then**
    $setSumoEgoVehiceState(evs)$
  **end**
  $DV$ : vehicles departed in current time step
  $DV \leftarrow getSumoDepartedVehices()$
  **foreach** *departed veh.* $dv \in DV$ **do**
    $spawnVehicleInGazebo(dv)$
  **end**
  $V \leftarrow getSumoVehiceList()$
  **foreach** *vehicle* $v \in V$ **do**
    $vs \leftarrow getSumoVehiceState(v)$
    ROS tf mesage $tf$ $tf \leftarrow convert2ROStf(vs)$
    $publishROStfMessage(tf)$
  **end**
  $AV$ : vehicles arrived in current time step
  $AV \leftarrow getSumoArrivedVehices()$
  **foreach** *arrived veh.* $av \in AV$ **do**
    $deleteVehicleInGazebo(av)$
  **end**
  $SumoSimulationStep()$ : perform simulation step
**EndTimerLoop**

**Algorithm 1:** TraCI Interface for SUMO simulator

There are two main inputs for this algorithm, the desired vehicle flow, which defines the number of vehicles per unit of time passing each entry point of the road network and the time-step for this simulation. Having this, the initialization

process of the algorithm can be done, first, the route file is created, for this step, the definition of the scenario (e.g. route file mentioned in Sec. 1), and the provided vehicle flow are combined so as to obtain a file with all the elements of required to start the simulation.

The next step is to start of the simulation, which implies the connection of the interface with the SUMO simulator. Immediately after this, a timer is started. It should be clarified that this timer is based on the ROS-controlled clock, and it executes an cycle of the loop every time the timer reaches the previously-defined time-step, thus allowing the synchronization of both simulators.

The main loop, named *TimerLoop* in Algorithm 1, starts by calling a the Gazebo *get model state* service to obtain the status of the ego-vehicle (*i.e.* position, orientation, linear and angular speed). If the data received is valid, the interface will update the SUMO status of the ego-vehicle. Then, it's time to send information regarding the rest of the vehicles to Gazebo, and this is done in three steps: first, the *spawn model* service is used in order to add the vehicles departing, in the current time-step, to the simulation in Gazebo; second, the position and orientation of all vehicles, except the ego-vehicle, are read from SUMO, converted to a valid ROS transform message and published; the third step, is the deletion of those vehicles that have arrived to their destination, this is done using the *delete model* ROS service. Finally, the loop is ended by performing a new SUMO simulation step, this is achieved by sending the corresponding TraCI command.

### B. Gazebo Plugin

The second algorithm is a Gazebo plugin, this means that it is a complement that runs embedded in the Gazebo simulator. The main objective of the plugin is to control the position of the vehicles in gazebo, moreover, in contrast with the previous algorithm, the plugin does not control the execution of the simulation nor it handles the behaviour of the ego-vehicle, the reason for this is that those aspects are directly controlled by the Gazebo simulator, which also publishes the clock message required for time synchronization.

This plugin was written in C++ and it's summarized on Algorithm 2. Its behaviour is very straight-forward, it starts by connecting the plugin to the Gazebo simulation, and then, at each step of the simulation, it executes a routine that gets the list of models currently on Gazebo, and then reads the position and orientation of each one of them from the information available on the tf message (published by the SUMO interface). This information is converted to a Gazebo valid format, and finally, each model's pose is updated with this data.

## V. EXPERIMENTS AND RESULTS

With the objective of testing the capabilities of the hybrid simulation, a complex real world situation has been modelled. The scenario requires a merge action in presence of very high traffic, A real world image of the proposed scenario is presented in Figure 3, where the ego-vehicle is the car in the red circle and needs to merge with the traffic.

**Input:** $evn$ : Ego-vehicle name
$pluginConnection()$ : connect with Gazebo simulation
**UpateLoop** *: on each simulation step*
  $M$ : List of vehicles in simulation
  $M \leftarrow getGazeboModelList()$
  **foreach** *model.* $m \in M$ **do**
    $mid$ : id of vehicle (gazebo model name)
    $mid \leftarrow getModelName(m)$
    **if** $mid \neq evn$ *: not the ego-vehicle* **then**
      $vp$ : vehicle pose (from tf message)
      $vp \leftarrow lookupPoseFromTf(mid)$ $gp$ : Gazebo valid pose
      $gp \leftarrow transformToGazeboPose(vp)$
      $setGazeboPose(gp)$;
    **end**
  **end**
**EndUpateLoop**

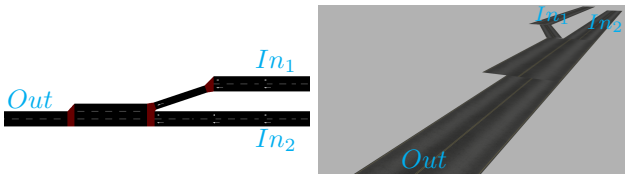
**Algorithm 2:** Gazebo plugin for controlling vehicles

Fig. 3: Proposed scenario, The ego-vehicle, in the red circle, needs to merge with the traffic.

It should be remarked that controlling the ego-vehicle, in order to solve the merge problem, is out of the scope of this work. Rather, the focus is on modelling the scenario itself, so as to provide a useful tool for developing and testing algorithms to solve these kind of problems. Therefore, in this experiment, the ego-vehicle movements are controlled by a human operator.

As mentioned in Section II, the first st[1] Univ. Grenoble Alpes, Inria, Grenoble INP, 38000 Grenoble, Franceep of the process is to define the scenario and translate it to both simulators. The result of this is depicted in Figure 4, where both the SUMO and the Gazebo empty scenarios are shown. The size and shape in both cases are identical, however, there are some differences, mainly because for SUMO the number of lanes, the connections and the direction of traffic flow is pre-defined in the scenario, whereas in Gazebo only the width of the road needs to be defined, furthermore, vehicles in gazebo may go off-road if the control does not prevent it.

The merge scenario is ensured because it has two entries and one exit, labelled $In_1$, $In_2$ and $Out$ in Figure 4. The rate at which additional vehicles appear in both entry points is defined by the desired vehicle flow (see Algorithm 1. Having this, as soon as the vehicles appear on the road, they

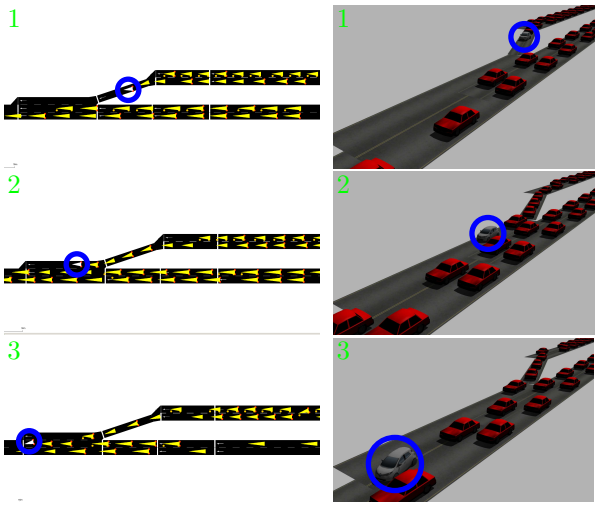



(a) SUMO (b) Gazebo

Fig. 4: Network design in SUMO and Gazebo. The scenario has two entry points ($In_1$, $In_2$) and one exit point ($Out$).

start to move towards the exit point. The ego-vehicle's initial position can be defined in the gazebo world, in this case it is the $In_1$ entry point (top-right in Figure 4a), and, as do every other vehicle, it will move towards the merge point in the centre of the network and then continue towards the exit.

The procedure described in previous sections is executed as soon as the hybrid simulation is launched, furthermore, in order to create a larger traffic jam, some of the vehicles can be commanded to start to move slowly after a certain point, thus generating a slow but dense traffic.




(a) SUMO (b) Gazebo

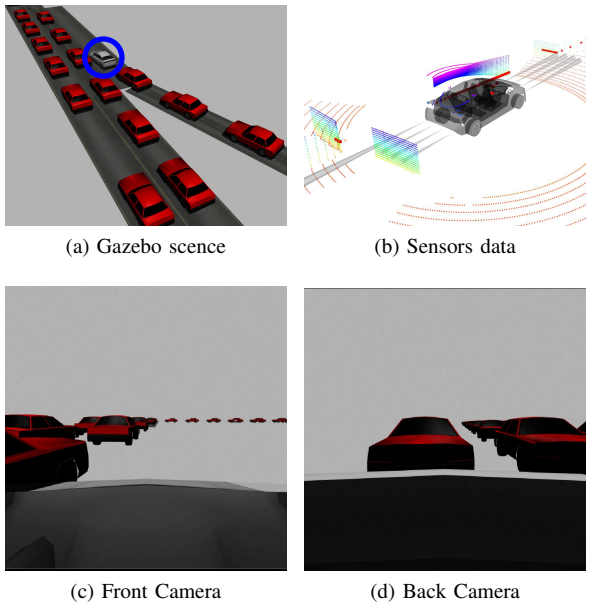
Fig. 5: Hybrid-simulation example: The ego-vehicle is marked with a blue circle. The other vehicles are shown in yellow (5a) and red (5b) respectively.

Figure 5 presents a sequence of movements in both simulators. In both cases, the ego-vehicle is shown in white, whereas, the other vehicles are shown in yellow in the SUMO simulation (Fig. 5a) and red in Gazebo (Fig. 5b). The number of additional vehicles participating in the simulation can be as high as needed, for this example, almost 300 additional vehicles where included. Furthermore, during this simulation, not only the integration of both simulators was tested, but also the sensors on-board the ego-vehicle. This data can be used for any detection or control algorithm.

Regarding the computational cost of the hybrid simulation, it is highly dominated by the Gazebo simulator. As with any 3D simulation engine, the level of detail of the ego-vehicle, the sensors modelled and the quality of the graphics, are the most critical elements in terms of computing requirements. The SUMO simulation, and its synchronization do not present any considerable addition on those requirements.

Some lateral movements may be performed by the vehicles controlled by SUMO, the reason for this is that lane-change is a high-level command in this simulator, and it's done in a fixed time. However this is problematic only in the case of a very slow speed of the given vehicle. Moreover, some options or fine-tuning on the lane-change models can help to prevent this issue. Also, it is possible to individually control one or a group of the other vehicles, by adding control modules to the TraCI interface.

Finally, it should be remarked that the sensors, cameras and every other component, as well as control algorithms, in this case for teleoperation, were fully operative on the ego-vehicle during the experiments. An example, with the scene on gazebo and some sensors data is presented on Figure 6.



(a) Gazebo scence (b) Sensors data

(c) Front Camera (d) Back Camera

Fig. 6: Gazebo scene with the ego-vehicle marked with a blue circle (6a). Laser and range sensor data (6b). Cameras on front and back of the ego-vehicle (6c, 6d).

## VI. CONCLUSIONS

The proposed hybrid simulation successfully integrates and synchronizes two different simulators. By combining a complex 3D simulation, where the ego-vehicle can be modelled in detail, with a high traffic simulator, which controls the behaviour of other vehicles, regardless of the number, it was possible to place the ego-vehicle in an scenario were different algorithms, can be tested in realistic, high traffic situations,

The proposed schema allows the inclusion of any type of ego-vehicle, this is due to the fact that the only requirement is that it should be modelled in the Gazebo 3D simulator.

This represents a large advantage over many other vehicles, or low level control algorithms can not be easily included, furthermore, it also allows to include many sensors and additional components that are already available for Gazebo, and it also helps to reduce the simulation-to-reality gap, because sensor messages, control algorithms and other components, can be much more similar to those available for real world vehicles.

A complex situation has been modelled, one that requires the interaction of a large number of vehicles, that correctly respond to the actions of the ego-vehicle. The bi-directional communication schema proposed, also allows to feed data to algorithms running on the ego-vehicle, and this is achieved without a significant increase on the computational cost of the simulation, even when a large number of additional vehicles is included.

Finally, the behaviour of the additional vehicles is mainly decided by car-following and lane-changing models, which can make them somehow predictable. Nevertheless, it is possible to modify those models or even add control modules for some or all vehicles within the TraCI interface, so as to obtain random or more realistic behaviours.

## SUPPLEMENTARY MATERIAL

The code used for the experiments presented on Section V is open and available at `https://bitbucket.org/marioney/hybrid_simulation`. In the same link full videos of the experiments can also be found.

## ACKNOWLEDGMENT

This work was funded under project CAMPUS (Connected Automated Mobilty Platform for Urban Sustainability) sponsored by Programme d'Investissements d'Avenir (PIA) of french Agence de l'Environnement et de la Maîtrise de l'Énergie (ADEME).

## REFERENCES

[1] T. Eldabi, M. Balaban, S. Brailsford, N. Mustafee, R. E. Nance, B. S. Onggo, and R. G. Sargent, "Hybrid simulation: Historical lessons, present challenges and futures," in *2016 Winter Simulation Conference (WSC)*, Dec 2016, pp. 1388–1403.

[2] C. Aguero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. Rivero, J. Manzo, E. Krotkov, and G. Pratt, "Inside the virtual robotics challenge: Simulating real-time robotic disaster response," *Automation Science and Engineering, IEEE Transactions on*, vol. 12, no. 2, pp. 494–506, April 2015.

[3] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, December 2012.

[4] B. Wymann, C. Dimitrakakisy, A. Sumnery, and C. Guionneauz, "Torcs: The open racing car simulator," 2015.

[5] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 102–118.

[6] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 1856–1860.

[7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.