Digital Twin Simulation of Connected and Automated Vehicles with the Unity Game Engine

Ziran Wang*, Kyungtae Han, and Prashant Tiwari
Toyota Motor North America R&D, InfoTech Labs, Mountain View, CA, USA
{ziran.wang, kyungtae.han, prashant.tiwari}@toyota.com

2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI) | 978-1-6654-3337-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/DTPI52967.2021.9540074

***Abstract*—A Digital Twin is defined as a digital replica of a real entity in the physical world. In this study, the Digital Twin simulation is developed for connected and automated vehicles (CAVs) by leveraging the Unity game engine. A Digital Twin simulation architecture is proposed, which contains the physical world and the digital world. Particularly, the digital world consists of three layers, where the Unity game objects are built to simulate the "hardware", the Unity scripting API are used to simulate the "software", and external tools (e.g., SUMO, MATLAB, python, and/or AWS) are leveraged to enhance the simulation functionalities. A case study of personazlied adaptive cruise control (P-ACC) is conducted to showcase the effectiveness of the proposed Digital Twin simulation, where the ACC system can be designed to satisfy each driver's preference with the help of cloud computing.**

I. Introduction

With rapid developments of sensing, computing, and communication technologies, the term "connected and automated vehicle" (CAV) has become a significant popular topic in the research community during the past decade [1]. The level of automation and connectivity of vehicles has greatly improved, which enables these CAVs to not only drive under partial or full automation with the help from their on-board sensors, but also behave collaboratively through vehicle-to-everything (V2X) communications.

As the availability of testing CAV applications in the real world is always constrained by various factors (e.g., space, cost, and regulation), how to build high-fidelity simulation models of CAVs in digital environments becomes crucial. Along with the developments of CAV technologies, several tools came into play while conducting computer simulations of CAVs. Microscopic traffic simulators, such as SUMO [2], VISSIM [3] and Aimsun [4], allow users to model a relatively large amount of vehicles in a traffic environment. However, users of such simulators cannot implement high-fidelity CAV modules (e.g., perception and localization), neither can they manually control any of the simulated vehicles through external inputs.

Game engines are originally adopted by software developers to design video games, and they typically consist of a rendering engine for 2-D or 3-D graphics, a physics engine for collision detection and response, and a scene graph for the management of multiple elements (e.g., models, sound, scripting, threading, etc.). During the rapid development of game engines over the years, their functionalities have been brought to a wider scope: data visualization, training, medical, and military use. Additionally, game engines gradually become a popular option to simulate CAVs [5], which have been used by various researchers from the human perspective [6], the vehicle perspective [7]–[9], and the system perspective [10], [11].

Therefore, in this study, a Digital Twin simulation architecture is built based on the Unity game engine [12]. A Digital Twin is defined as a digital replica of a real entity in the physical world [13]. Given the characteristics of the Digital Twin and the powerful features of Unity, users are able to build comprehensive CAV models with all necessary modules, as well as realistic road environments with high-fidelity assets. The Digital Twin simulation also allows users (i.e., game players) to get fully immersed in the simulation environment through advanced human-machine interfaces (HMI). Besides the built-in functionalities of Unity, external tools can also be integrated in the simulation to maximize the strengths of the Digital Twin simulation.

The remainder of this paper is organized as follows: Section II introduces the general architecture of the Digital Twin simulation, with specific information on the Unity game engine and its interfaces with other simulation software. Section III conducts the modeling and evaluation works of the Digital Twin simulation with a CAV application case study. Finally, the paper is concluded with some future directions in section IV.

II. Digital Twin Simulation Architecture

As shown in Fig. 1, the Digital Twin simulation architecture consists of two layers. The lower layer is the physical world, which contains the real-world objects that are being simulated: road networks, ego CAVs, and other vehicles. The upper layer is the digital world, which has three sub-layers: Unity game objects, Unity scripting application programmable interface (API), and external tools.

A. Physical World

One of the key advantages of the Digital Twin simulation is that, each real object in the physical world will be simulated with its digital replica in the digital world. Therefore, in this study where we use Digital Twin simulation for CAVs, we consider real-world objects in the physical world that are most related: road networks, ego CAVs, and other vehicles.

For road networks, we consider all basic elements in the physical world, including road geometry, road grade, road type, road barrier, road surface marking, road sign (e.g., stop, speed), and road infrastructure (e.g., traffic signal, lamp). For certain cases, we even consider the surrounding environment, such as trees and buildings, to increase the fidelity of our Digital Twin simulation.

With respect to the vehicles in the physical world, we generally consider four different vehicle types: legacy vehicle (without connectivity or automation), connected vehicle

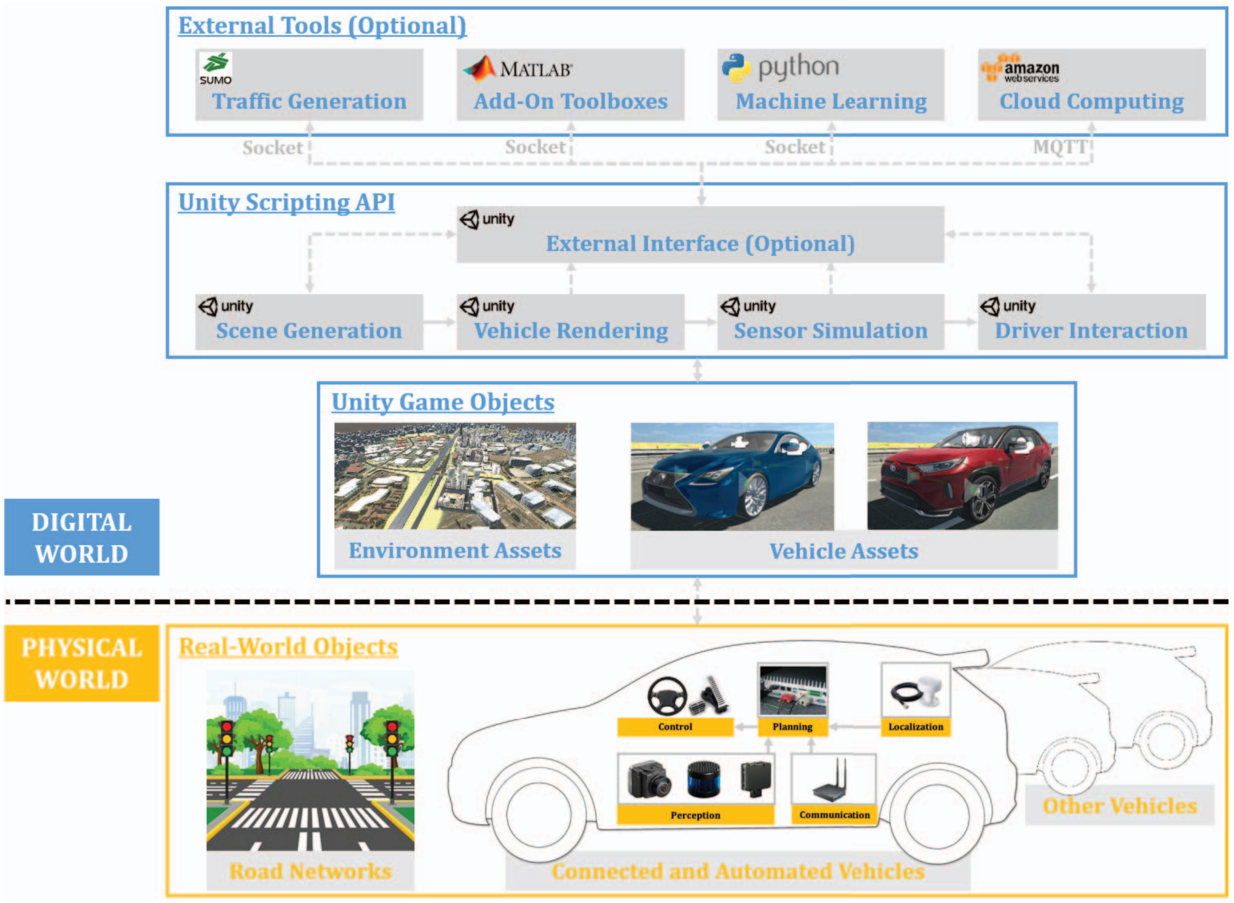

Fig. 1. Digital Twin simulation architecture with the physical world and the digital world

(without automation), automated vehicle (without connectivity), and CAVs. Although connectivity and automation will play increasing roles in our daily-driven vehicles, we will not have 100% market penetration rate of CAVs any time soon, and our road networks will be filled with all these four vehicle types in the foreseeable future.

B. Digital World

1) Unity Game Objects: In our Digital Twin simulation, the Unity game engine is used as the major platform for simulation given its following strengths:

- Graphics and visualization: Unity is originally for developing 3D video games, so it has a strong capability of graphics rendering and visualization. Unity also streamlines the prototyping and demonstration of CAV technologies with its easy-to-use user interface.
- Integration with external platforms: Unity enables easy integration with external hardware such as driving wheel & pedals, which makes human factor-related studies available. The well-designed scripting API of Unity also allows integration with external software, which enriches the functionalities of the simulation platform.
- Extensive assets: The official asset store of Unity allows developers and users to share their Unity assets, so any project can be built on top of others' projects instead being built from scratch.

The game objects in our Digital Twin simulation are built with high-fidelity Unity assets. For the simulation environment, an extensive range of assets can be found from the Unity asset store, including terrains, road segments, buildings, plants, weathers, and etc. Realistic simulation environments can be built with these assets in different game scenes, enabling high-fidelity testings of CAV applications in various settings (e.g., urban vs rural, arterial vs highway, sunny vs snowy).

Vehicles are the most important elements in this Digital Twin simulation, where existing vehicle assets can be leveraged to enable high-fidelity simulation to the largest extent. Instead of building by the users themselves, photorealistic 3D vehicle models are made available by designers through the Unity asset store and third-party websites (such as TurboSquid [14]). On top of the models themselves, which are more like hardware of the vehicles, software assets can also be built. There are extensive selections of vehicle powertrain/dynamics assets, vehicle sound assets, and even automated driving assets that make our Digital Twin simulation of CAVs easier.

2) Unity Scripting API: Unity provides users with a scripting API that is written in C#. Hierarchically, the Unity scripting API has four different elements: 1) Namespaces that provide access to classes written by the Unity developers; 2) Classes in each namespace that include various members; 3) Members of each class that provide specific functionalities; 4) Global classes and types that do not belong to namespaces.

In our Digital Twin simulation, we realize all of our desired functionalities through this API. As shown in Fig. 1, there are five major functionalities among others: scene generation, vehicle rendering, sensor simulation, driver interaction, and external interface (optional).

- Scene generation: Based on associated game objects, simulation scenes need to be generated with specific setups, such as initialization (i.e., which game objects should be enabled in the scene) and termination (i.e., when a certain condition is triggered).
- Vehicle rendering: Vehicles in the simulation are rendered in desired dynamics, which can be realized by associated scripts (e.g., motion planning and control).
- Sensor simulation: User-defined scripts can be implemented to vehicle game objects to simulation sensing technologies (e.g., perception, localization, and communication), thus transforming vehicles into CAVs.
- Driver interaction: For simulated vehicles with no automation, namely legacy vehicles and connected vehicles, players of the Unity simulation can act as the human drivers of such vehicles through external inputs (i.e., keyboards, mouses, or joysticks). For automated vehicles or CAVs, players can also interact with such vehicles through customized HMI.
- External interface (optional): User-defined scripts can enable communication among Unity and external tools, where data can be transmitted over TCP/IP or UDP sockets on local machines, or over MQTT/HTTP with cloud.

3) External Tools (Optional): This is an extension of the Unity scripting API, where users can take advantages of external tools to enhance the functionalities of the Digital Twin simulation. Specifically, with our customized external interface designed in the Unity scripting API, we can connect Unity with SUMO for traffic generation [15], with MATLAB to use add-on toolboxes [6], with python for machine learning resources [8], and with Amazon Web Services (AWS) for cloud computing. The former three implementations can be referred to associated references, while the AWS case is presented in the next section.

III. Case Study with Connected and Automated Vehicle Applications

In this section, we conduct a case study of CAV applications using the Digital Twin simulation. Specifically, we design an Advanced Driver-Assistance System (ADAS) called "Personalized Adaptive Cruise Control" (P-ACC), which models the ACC system in a personalized manner to satisfy each individual driver in terms of safety and comfort.

The general workflow of simulating this application is as follows:

1) At trip n, a driver manually controls the ego vehicle to follow the preceding vehicle. All related data is sampled in the simulation environment, where some preliminary data filtering processes are conducted in Unity.
2) During this trip n, the processed car-following data (such as time gap with respect to the preceding vehicle) is sent to AWS via the external interface.
3) Personalization algorithms are implemented on AWS to train the (newly received and historically stored) data by leveraging the powerful cloud computing capability.
4) At the beginning of trip $n+1$, the personalized guidance is sent from AWS to Unity and applied towards the P-ACC of the ego vehicle, such as setting the desired time gap of the ACC algorithm.

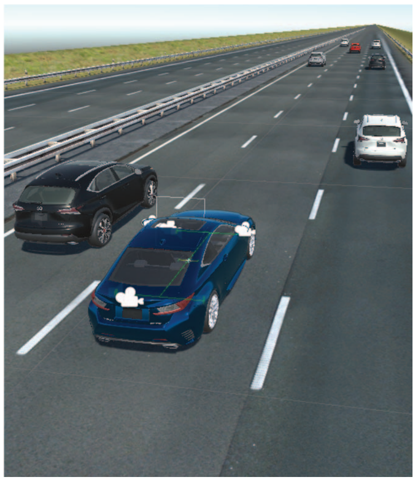
Fig. 2. A three-lane highway simulation environment in Unity, where the blue vehicle is the ego vehicle.

As can be seen from Fig. 2, a three-lane highway simulation environment is built in Unity, where the dark blue coupe in the middle is the ego vehicle that seats the driver. Non-player character (NPC) vehicles are also generated on all three lanes of the simulated highway segment, making the traffic environment realistic. Specifically, the leading NPC vehicles on each lane loads predefined speed trajectories that were generated by previous real-vehicle testings, while the following NPC vehicles are implemented with radar sensors and the PID-based ACC algorithm that enable them to follow preceding vehicles with desired time gaps.

It is without a doubt that multiple factors can affect drivers' car-following preferences, which vary from person to person. Those factors may include vehicle types, as a sporty coupe definitely has different vehicle dynamics than a SUV. Weather conditions can also play a huge difference, since visibility and road slipperiness affect drivers' car-following behaviors to a large extent. Therefore, we implement different vehicle types (as shown in Fig. 1) with unique settings such as drivetrain, motor force, brake force, center of mass, aerodynamics, and etc. Additionally, we also implement various weather conditions in the Digital Twin simulation, where some of them are illustrated in Fig. 3.

The aforementioned simulation step 1) and 2) can be seen from Fig. 4, which shows the Unity environment, driver-vehicle interaction (through an external Logitech G29 Driving Force racing wheel), the uplink message sent from Unity to AWS, and the downlink message sent from AWS to Unity. In this naturalist

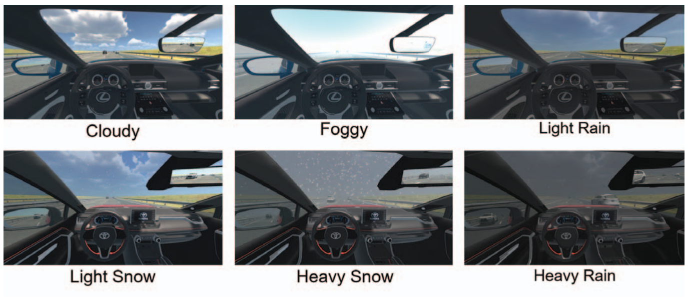

Fig. 3. Various weather conditions in the simulation environment that affect drivers' preferences of car following.

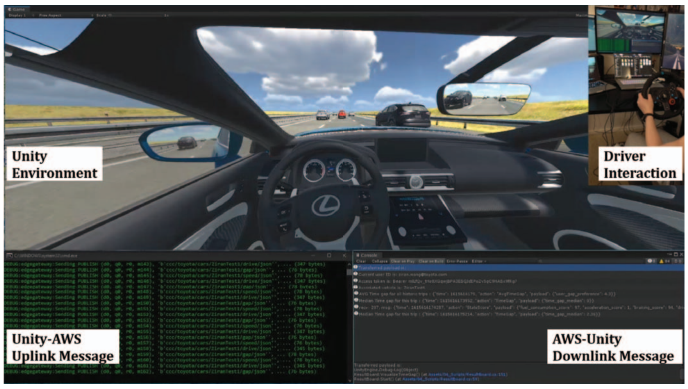

Fig. 4. A snap shot of the human-in-the-loop simulation with a driver controlling the ego vehicle through Logitech G29 Driving Force racing wheel, and both uplink and downlink data transmissions are happening between Unity and AWS.

driving trip, the driver manually controls the ego vehicle to follow the preceding vehicle, where the speeds of both the ego vehicle and the preceding vehicle, together with the distance gap measured by the radar sensor, are sent from Unity to AWS in real time.

On the other hand, once the uplink data is received on AWS, it firstly goes through a driver-vehicle association function which allows the data to be stored in the particular driver database and vehicle database. Then, we implement a personalized driving algorithm to calculate this driver's preferred time gap value considering the data from the current trip and all historical trips in the driver database. After the result is calculated (e.g., preferred time gap value is 1.7 second), it is sent to the ego vehicle in Unity, which can be applied to the P-ACC algorithm (through the Unity scripting API) when the driver turns on the automated control during a later trip.

IV. Conclusion and Future Work

In this study, the Digital Twin simulation has been developed for CAVs by leveraging the Unity game engine. A Digital Twin simulation architecture has been proposed, which contains the physical world and the digital world. Particularly, the digital world consists of three layers, where the Unity game objects are built to simulate the "hardware", the Unity scripting API are used to simulate the "software", and external tools (e.g., SUMO, MATLAB, python, and/or AWS) are leveraged to enhance the simulation functionalities. A case study of P-ACC has been carried out to showcase the effectiveness of the proposed Digital Twin simulation, where the ACC system can be designed to satisfy each driver's preference with the help of cloud computing.

One of the crucial next steps of this Digital Twin simulation is to integrate Robot Operating System (ROS) through the external interface in Unity, as ROS (or ROS 2) serves as the platform for many CAV systems on the market, which can further improve the fidelity of our Digital Twin simulation. Additionally, realistic wireless communication models, such as dedicated short-range communication (DSRC) or cellular-V2X (C-V2X), can be implemented to enable more research on the communication aspect of CAV applications.

Acknowledgment

The contents of this paper only reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views of Toyota Motor North America.

References

[1] Z. Wang, Y. Bian, S. E. Shladover, G. Wu, S. E. Li, and M. J. Barth, "A survey on cooperative longitudinal motion control of multiple connected and automated vehicles," *IEEE Intelligent Transportation Systems Magazine*, vol. 12, no. 1, pp. 4–24, 2020.
[2] Eclipse SUMO, "Simulation of urban mobility," Accessed: 2021-05-03. [Online]. Available: https://www.eclipse.org/sumo/
[3] PTV GROUP, "PTV Vissim is the world's most advanced and flexible traffic simulation software," Accessed: 2020-08-10. [Online]. Available: https://www.ptvgroup.com/en/solutions/products/ptv-vissim/
[4] Aimsun, "Aimsun: Traffic planning, simulation and prediction," Accessed: 2021-05-03. [Online]. Available: https://www.aimsun.com/
[5] J. Ma, C. Schwarz, Z. Wang, M. Elli, G. Ros, and Y. Feng, "New simulation tools for training and testing automated vehicles," in *Road Vehicle Automation 7*, G. Meyer and S. Beiker, Eds. Cham: Springer International Publishing, 2020, pp. 111–119.
[6] Z. Wang, X. Liao, C. Wang, D. Oswald, G. Wu, K. Boriboonsomsin, M. Barth, K. Han, B. Kim, and P. Tiwari, "Driver behavior modeling using game engine and real vehicle: A learning-based approach," *IEEE Transactions on Intelligent Vehicles*, pp. 1–1, 2020.
[7] Z. Wang, G. Wu, K. Boriboonsomsin, M. Barth *et al.*, "Cooperative ramp merging system: Agent-based modeling and simulation using game engine," *SAE International Journal of Connected and Automated Vehicles*, vol. 2, no. 2, 2019.
[8] Y. Liu, Z. Wang, K. Han, Z. Shou, P. Tiwari, and J. H. L. Hansen, "Sensor fusion of camera and cloud digital twin information for intelligent vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2020.
[9] Z. Wang, K. Han, and P. Tiwari, "Augmented reality-based advanced driver-assistance system for connected vehicles," in *2020 IEEE International Conference on Systems, Man, and Cybernetics*, Oct 2020.
[10] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.
[11] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–6.
[12] Unity, "Unity for all." [Online]. Available: https://unity.com/
[13] Z. Wang, X. Liao, X. Zhao, K. Han, P. Tiwari, M. J. Barth, and G. Wu, "A Digital Twin paradigm: Vehicle-to-Cloud based advanced driver assistance systems," in *2020 IEEE 91st Vehicular Technology Conference*, May 2020, pp. 1–6.
[14] TurboSquid, "3D models for professionals." [Online]. Available: https://www.turbosquid.com/
[15] X. Liao, X. Zhao, G. Wu, M. Barth, Z. Wang, K. Han, and P. Tiwari, "A game theory based ramp merging strategy for connected and automated vehicles in the mixed traffic: A unity-sumo integrated platform," *arXiv preprint arXiv:2101.11237*, 2021.