Constructing a Digital Twin of a Physical Testbed: A Practical Guide for Connected Autonomous Vehicle Simulation

Oakley Thomas*, Rahul Gedela*, Anurag Hruday Pirangi*, Jee Heum Rah, Yunchang Kum, Steven Korzelius, Chaozhe R. He, Adel W. Sadek, Chunming Qiao

***Abstract*— While connected and autonomous vehicles (CAVs) have huge potential to enable safe and efficient mobility, realizing it requires rigorous design and verification. Design and verification in a real-world setting are costly in both time and expense, and an unsafe/unverified design could harm other human road users. As a result, building a realistic digital twin for CAVs has attracted major attention. In this work, we present a practical guide for developing a robust digital twin framework to assist in the development and evaluation of CAVs. We leverage four pillars: Autoware as the autonomy stack, MathWorks' RoadRunner for map building, SUMO for traffic simulation, and CARLA for rendering and sensor simulation. We provide the detailed steps for creating a high-fidelity digital twin of a real-world proving ground. In doing so, we highlight key functionalities of the digital twin, explain implementation details, present results, and showcase a design and verification framework based on the digital twin. Finally, we discuss future research applications of the framework. Our code can be found here: `https://github.com/ub-cavas/UB-DigitalTwin`**

I. Introduction

Autonomous driving technology has the potential to dramatically improve the transportation system by reducing human error, which accounts for the vast majority of traffic accidents. This technology also has the potential to increase road safety and decrease traffic congestion. Beyond safety, autonomous vehicles (AVs) promise greater accessibility for elderly or mobility-impaired individuals and can optimize fuel efficiency at scale, contributing to lower emissions across fleets. However, before widespread deployment is feasible, the safety and reliability of these systems must be rigorously validated across an enormous range of edge cases, environmental conditions, and failure modes — challenges that are impractical or impossible to address solely through real-world testing. This is where digital twin technology becomes essential: by constructing high-fidelity virtual replicas of real-world environments, researchers and engineers can simulate thousands of scenarios at scale, accelerating development cycles while exposing system vulnerabilities in a controlled, cost-effective setting. Digital twins thus serve as a critical bridge between promising AV technology and its safe deployment needed for public adoption.

*These authors contributed equally to this work.

Oakley Thomas, Rahul Gedela, Anurag Hruday Pirangi, Jee Heum Rah, Chunming Qiao and Yunchang Kum are with the Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY, USA {oakleyth, ramasair, anuraghr, jeeheumr, smk32, qiao}@buffalo.edu and sgsaver12@gmail.com. Chaozhe R. He is with the Department of Mechanical and Aerospace Engineering chaozheh@buffalo.edu. Adel W. Sadek is with the Department of Civil, Structural and Environmental Engineering asadek@buffalo.edu.

In the context of autonomous driving, a digital twin is a virtual, dynamic representation of a connected and autonomous vehicle (CAV) system, including the ego vehicle and its surrounding environment. An effective digital twin assists CAV development from design to deployment. This includes generating training datasets [1], enabling iterative experimentation [2], and providing accurate end-to-end validation before deploying on real hardware. State-of-the-art methods also include integrating a digital twin with a physical test track in real time [2].

In this work, we present a practical guideline for constructing a co-simulation-friendly digital twin that can interact with a real test bed. Our contributions are:

- A proposed workflow for digitizing physical road networks with integrated Geographic Information System data.
- A Digital Twin framework and open source code base that integrates SUMO, CARLA, and Autoware, along with an example map and multiple safety-critical scenarios
- A proposed architecture for multi-agent simulation and Mixed Reality applications

The remainder of the paper is organized as follows. Section II reviews related research endeavors. Section III describes the steps for constructing a cross-simulator-compatible virtual environment of a real-world road network. Section IV discusses the implementation of co-simulation with the microscopic traffic simulator SUMO [3]. Section V concludes our work and discusses future applications.

II. Related Work

The existing literature on creating a digital twin varies depending on the intended use case. In this section, we review related work on two use cases relevant to our work: map reconstruction and co-simulation.

A. Environment Reconstruction

Wang et al. [4] demonstrate Unity-based CAV digital twins with strong visualization, but lacks native SUMO and Autoware integration. Mcity [2] is the gold standard for physical-digital integration but is focused on a single-site; our framework is site-agnostic, requiring only OSM data, USGS elevation data, and a LiDAR-equipped platform. 3D Gaussian Splatting [1] and its hierarchical extension [5] offer compelling photorealistic reconstruction; however, CARLA 0.9.16 (Unreal Engine 4.26) does not render splats natively, motivating our photogrammetry fallback. In summary, real-world environment

reconstruction is an active research problem that remains to be completely solved.

B. Co-Simulation Architectures

The baseline for CARLA–SUMO integration is the official co-simulation tooling [6], which defines a lock-step loop: advance SUMO via TraCI, mirror NPC positions and signal states into CARLA, then trigger a single world tick. This two-party design implicitly assumes CARLA is the sole time consumer. Introducing Autoware [7] breaks that assumption, since the Autoware-Carla interface [8] also issues world tick internally, causing CARLA to advance twice per logical step. Liu et al. [9] construct a joint SUMO–CARLA–Autoware platform for intersection testing, confirming demand for three-way co-simulation, but the exact synchronization mechanism between all three simulators is not mentioned. TeraSim [10] achieves full triad integration via an asynchronous Redis-mediated bus, prioritizing cloud scalability for statistical safety testing at the cost of strict timestamp causality. OpenCAMS [11] couples SUMO, CARLA, and OMNeT++ with a step-barrier model and explicit multi-client TraCI ordering, directly paralleling our design, but without a production Autoware integration. On the pairwise side, the CARLA-Autoware-Bridge [8] and SumoWare [12] establish ROS2-based CARLA-Autoware and SUMO-Autoware links, respectively. To the best of our knowledge, our external tick mechanism is the first open-source solution that explicitly resolves the double-tick conflict, yielding deterministic lock-step co-simulation across all three components and eliminating the time drift documented in Section IV.

III. Map Reconstruction with RoadRunner

In this section, we provide details about map generation for the digital twin. To assist validation transfer from simulation to reality, we prioritized reconstructing our real-world proving ground, as opposed to using prebuilt virtual environments. This process involved integrating environment representation data from public sources with manually collected data. The following workflow illustrates how we digitally reconstructed an environment and made it suitable for CAV development and testing.

A. Road Network Construction from OpenStreetMap

To obtain the road network of our proving grounds, we use the OpenStreetMap [13] web interface to manually define a region of interest (approximately 1 km^2) and export a ".osm" file containing road geometry that falls within the specified bounding box. This data then serves as the ground truth reference for the rest of the map construction process.

The ".osm" file is imported into RoadRunner [14] using its SD Map Viewer tool, which creates an intermediate graph-based representation consisting of nodes and links. Additionally, a world origin is automatically established based on the geographic extents of the imported data. In the absence of tags or values in the OSM data, the import process assigns defaults that ensure a complete network representation, although this process often requires manual refinement in later editing steps to reflect the physical location more accurately.

Importantly, the OSM-derived road geometry lacks elevation information and must be provided explicitly using external data sources. This limitation necessitates the next step of georeferencing for elevation integration, which ensures an accurate spatial representation.

B. Georeferencing and Elevation Data

To integrate 3D structure, we augmented the imported OSM road network with high-resolution elevation data.

Elevation data is obtained from The National Map [15], a U.S. Geological Survey (USGS) platform that provides geospatial datasets. Following RoadRunner's GIS integration guidelines, we imported a 3D Digital Elevation Model (DEM) that closely matched the geographic extent of the OpenStreetMap export, thereby reconstructing the elevation surface while ensuring spatial consistency between the datasets.

However, the newly imported GIS data overrides the existing world coordinate frame that was established during the OSM import, as RoadRunner prioritizes the most recently imported GIS projection. This later proved to be a problem when exporting the map to other 3D applications, as the new origin introduced a significant offset from the applications' origins, causing problems for the physics engines and scene navigation. To resolve this, we explicitly reconfigure the world origin to a known, real-world latitude and longitude, followed by a projection update to adjust the local coordinates of scene objects, preserving their geographic locations.

C. Environment Development Using Meshed Point Clouds

After establishing the base map, we focused on developing a geometrically and visually accurate representation of the real-world environment. Rather than relying solely on procedural asset placement, we based our reconstruction on real-world 3D sensor data, using a dense point cloud as our primary reference.

Raw point cloud data is captured from the proving grounds using our autonomous vehicle platform [16], a 2017 Lincoln MKZ equipped with LiDAR and cameras, which collected and aggregated data to produce a three-dimensional point cloud (PCD) using DLIO [17], reflecting the real-world geometry and spatial context. The raw PCD is cleaned and downsampled using CloudCompare [18]. Finally, a Poisson surface mesh [19] is generated from the points and exported as an FBX file. Upon importing the mesh into RoadRunner, spatial alignment with the previously established components is required. This alignment is performed both visually and geometrically by comparing road edges with the existing road geometry, and terrain contours and elevation changes with the DEM-derived elevation layer.

After alignment, the scene is populated, with the meshed PCD serving as the visual reference. RoadRunner's extensive library of built-in scene assets (e.g., vegetation, signage, barriers) is used, along with Gaussian Splatting and photogrammetry techniques for reconstructing the site's buildings. Asset

Fig. 1. Side-by-side comparison of the real and reconstructed environment.

positioning was guided by multiple complementary sources, including structural cues present in the meshed PCD, ground-level photographs captured on site, and aerial photography and satellite imagery from Google Maps [20]. The resulting environment resembles the real-world proving grounds in both layout and appearance.

D. Building Reconstruction via Gaussian Splatting and Photogrammetry

While RoadRunner provides a comprehensive library of generic scene assets, buildings, and other infrastructure unique to the environment cannot be represented using them. To address this limitation, we employ high-fidelity 3D reconstruction pipelines from PolyCam [21] to generate non-generic assets from real-world observations. Video data of the buildings and surrounding structures is collected using a DJI Mavic Pro equipped with a 12-megapixel camera, providing multiple viewpoints and wide coverage. The captured videos are then input to the 3D-GS and photogrammetry pipelines, resulting in the generation of Gaussian splats and textured mesh representations, respectively. The generated splats are manually refined using Supersplat [22], a browser-based Gaussian splat editing tool to remove artifacts arising from partial scene reconstruction or dynamic elements present during capture.

While Gaussian splats are well-suited for rendering and visualization, RoadRunner and CARLA 0.9.16 do not support Gaussian splat rendering yet. Hence, we generate textured 3D mesh versions of all assets via photogrammetry. The reconstructed assets are then exported in FBX format and subsequently aligned with the features visible in the point cloud, ensuring that all of them occupied their correct real-world positions and orientations within the scene; see Fig. 1.

E. RoadRunner as the Central Map Creation Tool

The primary motivation for adopting RoadRunner as the central map-creation tool is its native compatibility with the CARLA simulator [23] and AWSIM-Labs [24], which we had previously experimented with. RoadRunner is the recommended software for generating the map assets required by CARLA, ensuring structural and semantic consistency between the map design and simulation environments.

Moreover, RoadRunner is specifically designed for 3D scene creation for AV simulation and testing. Although the pipeline described in this work requires significant manual intervention, particularly during alignment and asset placement, RoadRunner significantly reduces the effort in producing a simulation-ready environment. RoadRunner exhibits a low learning curve,

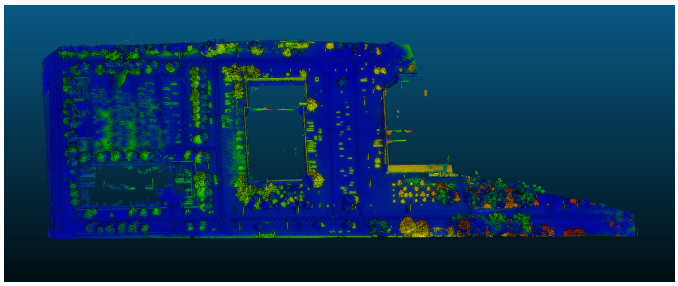

Fig. 2. CARLA map registered against the real-world point cloud. Quality of registration in the order: Blue (good match) > Green > Yellow > Red (poor match).

and its user-friendly interface and built-in tools enable rapid onboarding, allowing users to focus on map creation and integration.

F. Results

We evaluate the geometric fidelity of our map using two metrics: Iterative Closest Point (ICP) Root Mean Square Error (RMSE) and Mean Cloud-to-Cloud Distance ($\overline{d}$), defined as:

$$\text{ICP RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\|T(p_i) - q_i\|^2}, \tag{1}$$

$$\overline{d} = \frac{1}{N}\sum_{i=1}^{N} d(p_i, \mathcal{Q}), \tag{2}$$

where p_i are the source points, q_i are the corresponding target points, and $d(p_i, \mathcal{Q})$ is the distance from p_i to the nearest point in the target cloud $\mathcal{Q}$.

To evaluate geometric fidelity, we compare a mesh-based point cloud and a CARLA-generated LiDAR/point cloud map with a ground-truth point cloud captured at the real proving grounds, using point-pair matching and ICP registration in CloudCompare. The digital twin achieves an ICP RMSE of 0.635[m] and a mean cloud-to-cloud distance of 0.183 [m], while the CARLA map achieves 1.831 [m] and 1.077 [m], respectively, as summarized in Table I. The values indicate strong geometric correspondence between the digital twin and the real environment, and moderate correspondence between the CARLA map and the real environment. The higher error in the CARLA map could be attributed to minor inconsistencies in object placement, surface detail, and vegetation representation, as illustrated in Figure 2.

TABLE I
MAP ACCURACY COMPARISON

	ICP RMSE (1), [m]	**Mean Cloud-to-Cloud Distance (2), [m]**
Meshed Point Cloud	0.635	0.183
CARLA Map	1.831	1.077

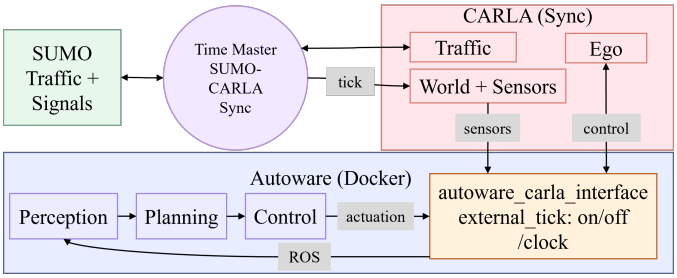

Fig. 3. Co-simulation architecture showing the Autoware–CARLA bridge for sensing and control, and the CARLA–SUMO synchronizer for traffic.

G. Limitations

While enabling the creation of a high-fidelity, simulation-ready environment, the proposed map-creation pipeline has the following limitations. Most notably, the geospatial accuracy of the reconstructed environment depends on the quality and resolution of external data sources, particularly OpenStreetMap road geometry, and DEM-derived georeferencing data. Additionally, multiple stages of the workflow rely heavily on manual intervention, particularly when handling lane semantics and during asset creation and placement. This limits scalability and introduces potential for subtle spatial inconsistencies.

Another major challenge was importing the map into the CARLA simulator. While RoadRunner and CARLA offer multiple methods for map import, the complex nature of the CARLA simulator means none of them achieve a completely seamless import. RoadRunner's guide for exporting with the CARLA plugin is informative, but additional effort is required to reimport and replace the non-generic scene assets in CARLA separately.

IV. Co-Simulation Architecture

Evaluating CAVs in complex urban environments requires a deterministic, closed-loop digital twin — one in which identical initial conditions produce identical trajectories, timestamps, and events across repeated runs — with the ego vehicle, traffic, and signals running on the same clock. This section presents a co-simulation framework that binds Autoware [7], CARLA, and SUMO [3], featuring a synchronization mechanism that enables the Autoware–CARLA bridge to operate under an external time master, eliminating timing conflicts in multi-simulator environments. CARLA's built-in traffic manager lacks signal control and scalability for dense intersection scenarios, motivating the addition of SUMO for microscopic traffic and signal timing (Fig. 3). However, existing SUMO–CARLA tools [6] assume CARLA is the sole time consumer; adding Autoware introduces timing conflicts we address in Sections IV-C, IV-D.

A. System Architecture

Autoware Universe (v0.43.1) runs in a Docker container with ROS2 Humble [25] and CycloneDDS, using the NVIDIA Container Toolkit for GPU acceleration. The container uses host networking and shared IPC for low latency, and mounts shared volumes for maps, calibration files, and pretrained models. CARLA runs as a standalone server in synchronous mode with a fixed step of 50 [ms], and SUMO runs as a TraCI-controlled process.

Two bridge layers handle communication (Fig. 3). The `autoware_carla_interface` serves as the ROS2 bridge, operating in closed loop: CARLA publishes sensor streams (LiDAR, camera, IMU, GNSS) to ROS2 at typical automotive rates (LiDAR/camera 10–15 [Hz], IMU 50–100 [Hz], GNSS 1–10 [Hz]), and Autoware returns actuation commands applied to the ego. After each update, the bridge publishes simulation time on `/clock`, and Autoware runs with `use_sim_time=true`. The SUMO–CARLA synchronizer advances SUMO via TraCI [26] each timestep, mirrors NPC positions and signal states into CARLA, triggers a single `world.tick()`, and synchronizes the ego state back so SUMO traffic can react to it. This synchronizer assumes exclusive authority over CARLA's time advance, an assumption the Autoware bridge violates, as described later.

B. SUMO Traffic Simulation

In our framework, SUMO serves as the backbone for scenario management, feeding structured interaction data to CARLA for visualization and subsequent perception by Autoware via simulated sensors. We leverage SUMO's TraCI Python interface to construct both random and scripted traffic scenarios, enabling precise control over safety-critical situations. Implemented scenarios span nominal and edge-case conditions, including unprotected left turns, intersection blockages, illegal pedestrian crossings, and failed U-turns. Because all TraCI-controlled movements are deterministic and repeatable, our framework supports rigorous, reproducible evaluation of autonomous driving systems, with plans to expand the scenario library in future iterations.

C. Timing Conflict and Resolution Strategy

Keeping CARLA physics, sensor timestamps, and traffic actor states in sync requires a single source of time advancement. In our framework, the SUMO–CARLA synchronizer serves as the global time master. The Autoware–CARLA bridge, however, also calls `world.tick()` within its sensor-processing loop as part of its default standalone behavior. With both components issuing tick calls, CARLA advances twice per logical timestep, so sensor timestamps no longer correspond to the synchronized traffic state.

Two approaches can resolve this conflict: modify the SUMO synchronizer to defer to the Autoware bridge's cadence, or disable the Autoware bridge's internal ticks. We adopt the latter because it isolates the fix and directly eliminates the competing-tick problem. The resulting modifications are formalized in Algorithm 1.

D. Implementation of External Time Control

To implement the Autoware-side timing fix, we extended `autoware_carla_interface` with a time-follower mode that delegates all tick authority to the SUMO–CARLA synchronizer, with 3 changes detailed below. The primary change

Algorithm 1: Synchronized Three-Way Co-Simulation

Input : Simulation time t, step size Δt
Output: Control command $\mathbf{u}$ for ego vehicle

Init: AC interface binds to existing world (Fix 1: no reload);

1. SUMO advances traffic to $t+\Delta t$ via TraCI;
2. foreach *vehicle v in SUMO* **do**
    Update NPC in CARLA with position of v;
3. Synchronizer triggers `world.tick()` in CARLA;
(Fix 2: AC interface suppress internal tick);
4. CARLA renders sensor data at $t+\Delta t$;
5. Autoware processes sensors $\rightarrow$ control $\mathbf{u}$;
(Fix 3: real-time pacing bypassed);
6. Queue $\mathbf{u}$ for ego *(applied next cycle)*;

return $\mathbf{u}$

is to suppress internal tick calls (issue 2 below). The remaining changes address related integration issues that arise once the bridge is no longer the exclusive time master.

(1) Bypassing world re-initialization: The original interface invokes `client.load_world()` immediately after connecting to CARLA. The SUMO synchronizer first connects and then maintains actor mappings between SUMO and CARLA. If the bridge then calls `load_world()`, CARLA resets, invalidating these mappings. In time-follower mode, we bypass world loading: the bridge attaches to the pre-loaded CARLA world via `get_world()`, preserving SUMO's synchronization context. The ego vehicle spawns without resetting CARLA, and the synchronization manager detects and registers the new actor.

(2) Suppressing internal tick calls: The bridge's default sensor-processing loop calls `world.tick()` internally. With both the bridge and the SUMO synchronizer issuing ticks, CARLA advances twice per logical timestep. In time-follower mode, the bridge's tick is suppressed, so CARLA advances exclusively through the SUMO synchronization loop, enforcing the lock-step order: advance SUMO via TraCI, synchronize actors bidirectionally, then tick CARLA exactly once.

(3) Disabling real-time pacing: The original bridge enforces real-time pacing by inserting sleep delays based on `max_real_delta_seconds`, preventing the simulation from running faster than wall-clock time. In co-simulation, this stalls the bridge, adds latency, and blocks faster-than-real-time runs. In time-follower mode, we bypass real-time throttling entirely. The bridge processes sensor data and publishes control commands as quickly as possible, then waits for the next externally triggered simulation update. Pacing is dictated solely by the orchestrator, which may run at real time, faster for batch testing, or slower for debugging.

With these modifications in place, Algorithm 1 summarizes the resulting lock-step execution (see also Fig. 3).

E. Evaluation Results

We validated the synchronization fix by logging SUMO and CARLA timestamps at each simulation step over 5000 steps (250 simulated seconds at simulation step $\Delta t = 50$ ms). Table II shows timestamp samples before and after the fix; Figure 4 shows the full run.

TABLE II
TIMESTAMP COMPARISON BEFORE AND AFTER SYNCHRONIZATION FIX

Step	SUMO (s)	CARLA (s)	Norm. (s)	Δ (s)	Ratio
Before Fix					
0	0.0	1157.3	0.0	0.0	–
100	5.0	1166.9	9.6	4.6	1.92×
500	25.0	1205.3	48.0	23.0	1.92×
1000	50.0	1253.3	96.0	46.0	1.92×
After Fix					
0	0.0	324.4	0.0	0.0	–
100	5.0	329.4	5.0	0.0	1.00×
500	25.0	349.4	25.0	0.0	1.00×
1000	50.0	374.4	50.0	0.0	1.00×

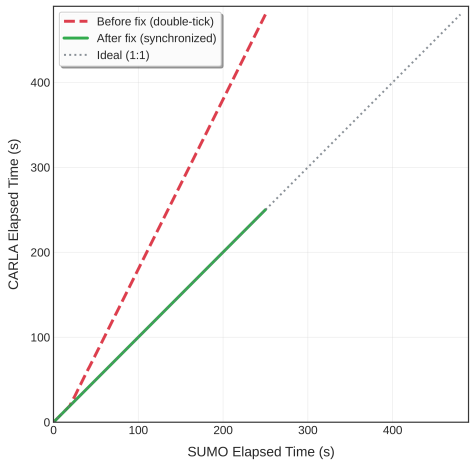

Fig. 4. CARLA vs. SUMO elapsed time over 250 simulated seconds. The dashed line shows pre-fix drift ($\sim$1.92×); the solid line shows synchronized behavior after the fix.

In Table II, Norm. denotes CARLA elapsed time relative to the initial timestamp, Δ is the drift between normalized CARLA and SUMO elapsed times, and Ratio is CARLA elapsed time divided by SUMO elapsed time (ideally 1.0). Before the fix, CARLA time advances at approximately 1.92× the rate of SUMO time. This near-doubling is consistent with our hypothesis: both the bridge and the synchronizer independently invoke `world.tick()`, causing CARLA to step twice per logical cycle. Over 250 simulated seconds, this accumulates to 230 s of drift (Fig. 4, dashed line). After suppressing the bridge's internal tick (Fix 2 in Algorithm 1), CARLA and SUMO elapsed times track within measurement precision along the ideal 1:1 diagonal. We confirmed deterministic reproducibility by executing the same scenario multiple times; all runs yielded identical timestamp logs.

F. Practical Considerations

We conclude this section with several integration details worth noting. CARLA and Autoware use different map formats (OpenDRIVE vs. Lanelet2), requiring y-axis inversion

or coordinate offsets, and CARLA maps lack lane-level signal annotations, so traffic light recognition needs manual HD map edits. Sensor frames do not match Autoware's sensor kit conventions out of the box, requiring custom TF transforms, and control gains are vehicle-specific. Finally, localization may require a manual GNSS pose reset at startup, and changing `fixed_delta_seconds` requires updating sensor rates to match.

V. Conclusions

This paper presented a modular, high-fidelity digital twin framework for CAV development and evaluation. By integrating Autoware for autonomy, RoadRunner for map reconstruction, CARLA for sensor-realistic simulation, and SUMO for deterministic traffic and scenario control, we demonstrated an end-to-end workflow that supports dataset generation, rapid prototyping, and safety-critical validation. A key contribution of this work is a deterministic three-way co-simulation architecture enabled by external time control, which resolves timing conflicts between autonomy, traffic, and rendering components and ensures repeatable experimental outcomes. In addition, we detailed a practical pipeline for reconstructing real-world environments and generating repeatable, safety-critical scenarios. Together, these contributions established a flexible and extensible foundation for future research in multi-agent simulation, mixed reality testing, and sim-to-real validation of autonomous driving systems.

Integrating multi-agent simulation opens the door to several compelling extensions in progress. A centralized authoritative CARLA server supporting variable client connections — representing Autoware instances, pedestrians, mixed reality agents, or smart infrastructure — would enable concurrent multi-vehicle simulation and richer scenario complexity. Mixed reality, in particular, presents a unique opportunity to pair physical vehicle dynamics with virtual environment and actors, offering a safer and more repeatable alternative to physical testing. Rather than injecting synthetic bounding boxes into the perception pipeline, we propose modifying raw sensor data directly to reflect the virtual world, enabling compatibility with recent advances in end-to-end autonomous driving stacks. Additionally, integrating with generative AI tools capable of photorealistic scenario rendering could address limitations in perception datasets while leveraging the traffic behaviors our framework already produces.

References

[1] T. Kerbl, A. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, 2023.

[2] "Mcity: Connected and automated vehicle test facility," https://mcity.umich.edu/, accessed: Jan. 2026.

[3] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo – simulation of urban mobility: An overview," in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011, pp. 63–68, accessed: 2026-01-30. [Online]. Available: https://sumo.dlr.de/pdf/simul_2011_3_40_50150.pdf

[4] Z. Wang, K. Han, and P. Tiwari, "Digital twin simulation of connected and automated vehicles with the Unity game engine," in *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 2021, pp. 1–4.

[5] B. Kerbl, A. Meuleman, G. Kopanas, M. Wimmer, A. Lanvin, and G. Drettakis, "A hierarchical 3d gaussian representation for real-time rendering of very large datasets," *arXiv preprint arXiv:2406.12080*, 2024.

[6] CARLA Team, "Sumo co-simulation (carla documentation)," https://carla.readthedocs.io/en/latest/adv_sumo/, accessed: 2026-01-30.

[7] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda *et al.*, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018.

[8] Autoware Foundation, "autoware_carla_interface (Autoware Universe Documentation)," https://autowarefoundation.github.io/autoware_universe/main/simulator/autoware_carla_interface/, accessed: 2026-01-30.

[9] B. Liu, H. Duan, J. He, R. Li, H. Li, and Y. Sun, "Construction of a joint simulation platform using SUMO, CARLA, and Autoware and its application in multi-vehicle interactions at unsignalized intersections," in *2025 7th International Conference on Information Science, Electrical and Automation Engineering (ISEAE)*, 2025, pp. 503–506.

[10] Michigan Traffic Lab, "TeraSim: Uncovering unknown unsafe events for autonomous vehicles through generative simulation," arXiv preprint arXiv:2503.03629, 2025, [Online]. Available: https://arxiv.org/abs/2503.03629.

[11] OpenCAMS Contributors, "OpenCAMS: An open-source connected and automated mobility co-simulation platform for advanced transportation research," arXiv preprint arXiv:2507.09186, 2025, [Online]. Available: https://arxiv.org/abs/2507.09186.

[12] TUM-VT, "SumoWare: Bridging SUMO and Autoware to assess AV-induced traffic impact," in *IEEE Intelligent Vehicles Symposium (IV)*, 2025, repository: https://github.com/TUM-VT/SumoWare.

[13] OpenStreetMap contributors, "OpenStreetMap," Data set, 2025, available under the Open Database License (ODbL) at https://www.openstreetmap.org. [Online]. Available: https://www.openstreetmap.org

[14] MathWorks, "RoadRunner," 2025, version 2025a. [Online]. Available: https://www.mathworks.com/products/roadrunner.html

[15] U.S. Geological Survey, "3D Elevation Program 1-Meter Resolution Digital Elevation Model," 2024, published Jan. 4, 2024. [Online]. Available: https://www.usgs.gov/the-national-map-data-delivery

[16] S. Korzelius, D. Vadranapu, O. Thomas, C. R. He, A. W. Sadek, and C. Qiao, "An old dog with new tricks: Lessons from building an av testbed with autoware," in *2025 IEEE 3rd International Conference on Mobility, Operations, Services and Technologies (MOST)*, 2025, pp. 176–185.

[17] K. Chen, R. Nemiroff, and B. T. Lopez, "Direct lidar-inertial odometry: Lightweight lio with continuous-time motion correction," *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3983–3989, 2023.

[18] CloudCompare, "CloudCompare (version 2.13) [GPL software]," http://www.cloudcompare.org/, 2024.

[19] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the Eurographics Symposium on Geometry Processing*, A. Sheffer and K. Polthier, Eds., 2006, available: https://hhoppe.com/poissonrecon.pdf.

[20] Google, "Google Maps," Web mapping service, 2025. [Online]. Available: https://maps.google.com

[21] Polycam, "Polycam: 3d capture and photogrammetry platform," https://poly.cam, 2023, [Online; accessed 30-Jan-2026].

[22] PlayCanvas contributors, "SuperSplat Editor," 2025, open-source browser-based tool. [Online]. Available: https://superspl.at/editor

[23] 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 (CoRL)*, 2017, pp. 1–16.

[24] Autoware Foundation, "AWSIM-Labs: Open-source simulator for autonomous driving," https://github.com/autowarefoundation/AWSIM-Labs, 2026, accessed: 2026-01-31. [Online]. Available: https://github.com/autowarefoundation/AWSIM-Labs

[25] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "The robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[26] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux, "Traci: An interface for coupling road traffic and network simulators," in *11th Communications and Networking Simulation Symposium (CNS '08)*. ACM, 2008, pp. 155–163.