An Old Dog with New Tricks: Lessons from Building an AV Testbed with Autoware

Steven Korzelius[1*], Daniel Vadranapu[2*], Oakley Thomas[3*],
Chaozhe R. He[4**], Adel W. Sadek[5***], Chunming Qiao[6*]
[*]*Department of Computer Science and Engineering*
[**]*Department of Mechanical and Aerospace Engineering*
[***]*Department of Civil, Structural and Environmental Engineering*
University at Buffalo (SUNY), Buffalo, New York, USA
smk32@buffalo.edu[1], dvadrana@buffalo.edu[2], oakleyth@buffalo.edu[3],
chaozheh@buffalo.edu[4], asadek@buffalo.edu[5], qiao@buffalo.edu[6]

***Abstract*—While building a testbed with a working Autonomous Vehicle (AV) running an open-source Autonomous Driving Stack (ADS) is essential for supporting AV-related research and experiments, it is also challenging, especially for academic researchers. In particular, academic researchers often have both limited funds to acquire the latest hardware and software components, and limited human resources and experience to integrate these components into a testbed. In this paper, we report the steps taken and lessons learned in upgrading the ADS on our 2017 Lincoln MKZ to the latest Autoware version (Universe), as well as associated efforts in localization, generating HD maps, integrating with simulators, and building digital twins. We describe the challenges encountered, including current deficiencies in Autoware Universe, and some of the solutions to overcome these challenges. We also present results obtained from our experiments and point out future research and experimental work. The source code of our work can be found at https://github.com/ub-cavas/ub-lincoln-docker. We believe that this paper could provide the community working on AV research useful insights into both building and using an AV experimental platform based on an open-source autonomous driving stack.**

***Index Terms*—autonomous, vehicle, autoware, hd-map, digital-twin**

I. Introduction

Autonomous Vehicle (AV) technology has emerged as a transformative innovation with the potential to revolutionize transportation systems. However, the complexity of AV systems and the stringent safety requirements necessitate rigorous testing and validation before deployment. A robust testbed that integrates both real-world and simulation-based evaluation environments is crucial for accelerating AV development while ensuring safety and reliability [1]. Autonomous vehicle companies such as Waymo, Motional, Aurora, and Plus, working with vehicle manufacturers and suppliers, spend hundreds of millions of dollars annually to maintain their test vehicle fleet so as to support technology development. However, matching such effort in academia is difficult, if not outright impossible, due to the lack of funding, resources, and technological know-how. While there are a few papers describing open-source, academic research-oriented platforms [1], [2], and many more papers that report research results, to our best knowledge, no work on maintaining a robust and up-to-date AV testbed, and in particular, upgrading an "old" vehicle platform with a "new" open-source autonomous driving stack (ADS) exists.

In this paper, we report our effort in developing and maintaining an AV testbed, allowing it to support AV research continuously. Our AV testbed development and maintenance efforts focus on meeting two goals: 1) to have a vehicle platform, built upon an "old" car (2017 Lincoln MKZ), with the latest open-source ADS Autoware Universe; and 2) to have a simulation platform of the physical test track. With the vehicle and simulation platforms, we aim to develop a cohesive testbed designed to address the challenges faced during AV technique development: scalability, cost-effectiveness, and safety of AV testing. By combining real-world operations with advanced simulation capabilities, this testbed will continue to serve as a powerful tool for advancing the development of autonomous vehicles and their associated technologies.

Specifically, the vehicle platform leverages the open-source ADS called Autoware, providing a flexible and scalable foundation for controlling and operating a real vehicle. By utilizing Autoware, the platform benefits from a modular architecture that supports various hardware configurations and advanced autonomous driving functionalities. Complementing this, the simulation platform employs a digital twin of the test track to replicate the physical environment virtually. This integration enables seamless transitions between physical and simulated testing, allowing developers to refine algorithms, validate safety protocols, and assess performance under diverse scenarios. The contributions of the work are as follows:

- We detailed the maintenance process of updating a testbed with an outdated ADS to the latest version
- We provide the recipe to configure and calibrate the updated stack and can deliver satisfactory baseline performance with the testbed
- We provide a guideline on building a simulation tool with instructions on building digital twins for a customized test track

We believe these contributions can provide insight to the research community on conducting AV research with a testbed using an open-source autonomous driving stack (ADS).

The remainder of this paper is organized as follows: In Section II, we provide background on open-source ADSs and a brief survey of related work. Section III discusses our AV platform setup and the stack update steps. Section IV describes the steps that enable driving the vehicle with ADS and demonstrates the performance. Section V describes the steps to establish a simulation environment for the testbed, including the steps to build a digital twin for the test track. In Section VI we discuss our learning, suggest needs for further system improvements and project future work. We conclude the paper in Section VII.

II. Background

In this section, we survey common architecture for autonomous driving stack (ADS) and survey three of the widely used open-source ADSs in academia.

A. Autonomous Driving Stack Overview

Autonomous driving stacks provide a comprehensive framework enabling vehicles to drive autonomously. These stacks integrate multiple layers of software that work together to perceive the environment, make decisions, and control the vehicle. The main components of the system and how they work together are described below and summarized in Fig. 1.

1) Perception: This component processes data from sensors such as LiDAR, RADAR, and cameras to detect and identify objects, lanes, pedestrians, and other vehicles in the environment. Sensor fusion algorithms can combine data from multiple sensors to create a cohesive and accurate understanding of the vehicle's surroundings.

2) Localization: Accurate localization is essential for autonomous driving. This component helps the vehicle determine its exact position in space relative to its environment. Localization measurement hardware such as Global Navigation Satellite Systems (GNSS), Inertial Measurement Units (IMU), and visual/wheel odometry can help refine the vehicle's position in relation to a high-definition map.

3) Mapping: High-definition maps store detailed information about the driving environment, including lane markings, traffic signals, and other key features. Autonomous stacks often use these maps as a reference to help the vehicle understand its drivable environment and route.

4) Planning: The planning component is responsible for path planning and decision-making. It includes route planning, which decides the best path to reach a destination and motion planning, which determines how to follow that path smoothly while avoiding obstacles.

5) Control: The control component converts planned routes and maneuvers into actuator actions by managing the vehicle's steering, acceleration, and braking. The control component also ensures the vehicle follows the planned path accurately and safely.

6) Simulation and Testing: Many autonomous stacks support simulation environments for testing algorithms and scenarios safely. This allows developers to refine the system in virtual environments before deploying it in real-world situations.

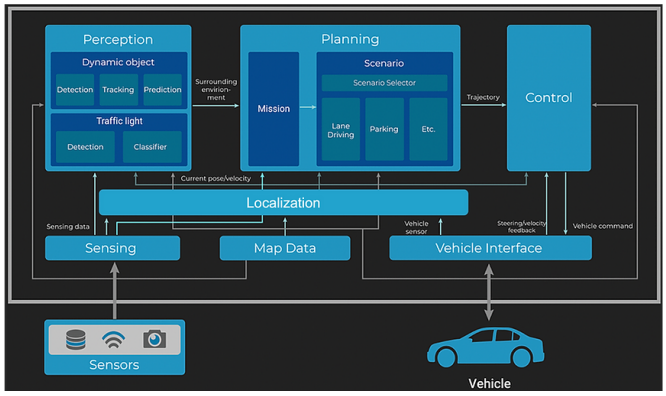

Fig. 1: Autonomous Driving Stack Data Flow [3]

B. Open-Source Autonomous Driving Stacks

An autonomous driving stack provides a foundational framework that accelerates the research of autonomous driving technologies. By offering prebuilt modules for critical systems such as perception, localization, planning, and control, a stack allows researchers to focus on innovation in specific modules of interest. Furthermore, open-source stacks foster collaboration and knowledge sharing, allowing researchers to leverage a community of contributors and access state-of-the-art algorithms. Additionally, compatible open-source simulation tools are available, which not only enable safe and cost-effective means for research complementary to real-world testing, but also open research opportunities to those who may not have access to a vehicle platform. Here, we briefly describe three popular open-source ADSs.

1) Autoware: Autoware, developed by Tier IV, is a prominent open-source software stack designed for autonomous vehicles [3]. Originally developed for academic research and prototyping, it has grown into a modular and robust platform suitable also for industrial and commercial applications. Autoware provides end-to-end solutions, including localization, perception, planning, and control. The stack has multiple versions, each with unique features. *Autoware.AI*, launched in 2015, was the original version of the platform developed by Nagoya University. Built on ROS, it provided an all-in-one solution for autonomous driving, including modules for perception, localization, planning, and control. While it gained traction in academia and research, its monolithic structure and lack of scalability presented challenges for deployment in production environments. *Autoware.Auto*, introduced in late 2018 under the guidance of the Autoware Foundation, aimed to address the limitations of its predecessor by adopting ROS2 for better modularity, real-time performance, and security. It was designed to comply with safety standards like ISO 26262, making it suitable for production environments in industrial and commercial applications such as autonomous shuttles and logistics. *Autoware Universe*, released in 2021, serves as a transitional platform, merging features from Autoware.AI into a ROS2-based framework. It offers backward compatibility for developers transitioning from Autoware.AI while incorporat-

ing advancements from Autoware.Auto. This hybrid approach makes it a flexible solution for developers exploring both prototyping and real-world deployments.

2) Apollo: Apollo, developed by Baidu, is a comprehensive and scalable open-source platform for autonomous driving [4]. It provides a full-stack solution, covering everything from hardware integration to high-level algorithms for perception, planning, and control. Apollo's capabilities include HD mapping, real-time localization, advanced sensor fusion, and robust simulation tools for testing and validation. Its modular architecture and extensive hardware support make it suitable for various applications, including autonomous taxis, delivery robots, and research initiatives. As a community-driven platform, Apollo benefits from extensive contributions and testing from both commercial and academic partners, which accelerates its development and helps it maintain a high level of performance and reliability in real-world applications. Like Autoware, Apollo has released several versions, each adding more capability to its feature set.

3) OpenPilot: OpenPilot, created by Comma.ai, is an open-source software stack that focuses on Advanced Driver Assistance Systems (ADAS) and provides a pathway to semi-autonomous driving [5]. It is designed for aftermarket installation on compatible vehicles, enabling features like lane-keeping, adaptive cruise control, and driver monitoring. OpenPilot is lightweight and utilizes end-to-end learning techniques to enhance its functionality continuously. Actively maintained and frequently updated, it has gained popularity among individual developers and car enthusiasts looking to add autonomous capabilities to their personal vehicles. With its ease of use and robust performance, OpenPilot represents an accessible entry point into self-driving technology.

III. Vehicle Platform

In this section, we describe the original vehicle platform that we start with, highlight the challenges, and report the updates we made on the vehicle software system and configurations to run the latest Autoware stack. The original vehicle platform is a 2017 Lincoln MKZ retrofitted by AutonomouStuff [6] with the hardware and software listed below.

A. Hardware

1) Computer: The ADS runs on an industrial PC (IPC) which is a Neousys Nuvo-6108GC with an Intel Xeon CPU (3.60GHz x 8), 32 GB of RAM, several 1TB SSDs and an NVIDIA GeForce GTX 1080 Ti.

2) LiDAR: The vehicle platform sensor stack includes a Velodyne LIDAR model VLP-32C with a 360-degree field of view and 32 scanning lines, each with a range of up to 200m.

3) GNSS with RTK & IMU/INS: The localization stack includes a NovAtel GNSS unit model PwrPak7D with dual antenna providing a high precision position and heading solution. The GNSS is capable of processing real-time kinematic (RTK) corrections for centimeter-level positioning. The localization stack also has an STIM300, a high-precision IMU that aids in positioning in the Inertial Navigation System (INS). The INS uses data from an IMU to calculate position, velocity, and attitude. When combined with GNSS, the two systems complement each other, with the INS providing stable relative positioning during periods when the GNSS signal is degraded or unavailable.

4) Dataspeed Drive-By-Wire Kit: The gateway to the vehicle is provided by a Dataspeed Drive-By-Wire (DBW) kit. The 2017 Lincoln MKZ is designed for DBW and with the Dataspeed DBW interface, we can send commands directly to the vehicle to control it, and read the data back from the vehicle for feedback [2].

B. Software

The original software on the vehicle platform ran on Ubuntu 16.04 using an early version of Autoware.AI with ROS. Each of the sensors also had an associated driver node to interface with the hardware and was open-source when we received the vehicle.

C. Challenges

1) Hardware Challenges: As this hardware was originally installed when the vehicle was retrofitted, the sensors (GNSS/LiDAR) and vehicle control hardware (DBW) have aged. While most of the hardware had firmware updates that improved usability and stability, updating firmware, requires a manual update and reconfiguration of the software drivers that interface between Autoware and the hardware. Such processes are non-trivial, tedious, and mostly missing clean unified instructions.

2) Software Challenges: The original Autoware.AI stack had only minimal base features, and the overall performance was unstable. Since then, official update releases have been applied selectively with non-trivial efforts each time. Despite the features added with the new updates, due to the backward compatibility issues with older hardware, the overall system became even less stable. Meanwhile, staying with the older version while avoiding the upgrade work would have limited the research capability of the AV platform.

D. Software & Configuration Updates

1) Software Systems: The onboard IPC OS was upgraded to Ubuntu 22.04 LTS using Autoware Universe with ROS2. While each of the sensors had a drive node available for ROS, the migration to ROS2 made finding, adapting or converting older ones necessary through the upgrade process. This was also complicated with hardware firmware version changes as newer drivers sometimes required newer firmware, which may not provide data in the same format Autoware required it.

2) Vehicle Configuration Setup: To enable Autoware Universe to run properly on a vehicle, two main configurations need to be setup so the stack has the required information. This part is critical as the configurations are specific to the vehicle platform and sensor layout.

a) Vehicle Specifications: The vehicle specification configuration contains all of the parameters required to define the vehicle that Autoware will use. This includes vehicle measurements such as *tire_wheel_radius* and other base vehicle dimensions. This configuration also describes how the vehicle behaves in simulation and contains the 3D model for simulation. While some of these configuration values were used in older versions of Autoware each major upgrade required them to be enhanced, updated and put into different locations of the ADS.

b) Sensor Kit: The sensor kit configuration describes the layout and types of sensors used in the ADS mounted on the vehicle. Similar to the Vehicle Specifications, the Sensor Kit contains the dimensional parameters of the sensors being used, their location with respect to the vehicle, and the 3D models used for them in simulation.

3) Interface Nodes: We created two interface nodes that translate and/or transform ROS messages between Autoware nodes and the Dataspeed DBW interface. This is needed because the messages that Autoware is expecting are different than what the DBW interface provides/accepts. Fig. 2 illustrates the workflow between Autoware and these nodes. This involves ROS topic renaming, data conversion, and message remapping similar to what was reported in [2].

a) Dataspeed Bridge Node: The Dataspeed bridge node ensures control messages (gear, steering, and speed control) are converted between Autoware-compatible ROS2 messages and DBW-compatible ones.

b) Data Transform Node: The data transform node is designed to listen to data streams from multiple sensors, including the NovAtel GNSS, LiDAR sensor, and vehicle velocity. The LiDAR data undergoes preprocessing to remove points in close proximity to the vehicle, which could otherwise be falsely identified as obstacles. Additionally, GPS coordinates are transformed into (X, Y, Z) coordinates. These transformed coordinates play a critical role in localizing the vehicle and ensuring accurate navigation within the environment.

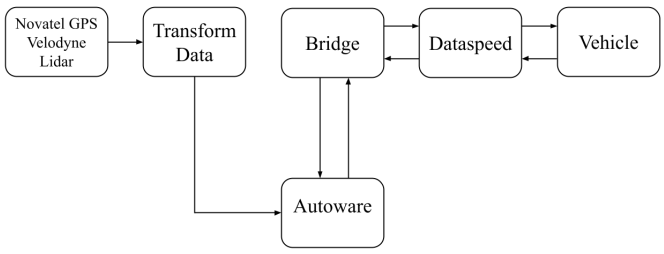

Fig. 2: Autoware/Dataspeed Bridge & Data Transform Nodes

4) Firmware Upgrades:

a) GNSS/RTK/INS: The firmware on the GNSS receiver needed to be updated to fix issues with configuration changes not being saved and RTK correction streaming. The new firmware required a new configuration in order to enable the RTK corrections from our local Continuously Operating Reference Station (CORS) network. This also required changes in the above Data Transform Node as the messages had changed between firmware versions.

b) Dataspeed DBW System: The Dataspeed DBW system was also updated with firmware. This enabled more ways to communicate data to the DBW, simplifying some of our control conversions. This firmware update was not an open-source but proprietary and a charged service by Dataspeed.

5) Validation: Initially, the control part of this setup was tested and validated in the vehicle platform in a parking lot. The objective of the test was to ensure that the vehicle could respond accurately to the control commands sent by Autoware. To achieve the full stack baseline driving performance with ADS, an HD map is essential for localization and planning module in the ADS. The next section will cover the process of map creation, along with the calibration of LiDAR and GNSS sensors.

IV. Drive with Autoware ADS

In this section, we describe the effort that enables driving the vehicle with Autoware ADS. Specifically, we report the steps to generate an HD map of our designated test track and report the autonomous driving performance of the Autoware ADS stack on the test track.

A. HD Map Generation

The latest Autoware ADS localizes the vehicle by matching the LiDAR scan with the HD map. Thus, an HD map is needed to execute autonomous driving with Autoware ADS in a certain area. HD maps are necessary for many autonomous driving functions such as lane-level navigation, behavior planning, and traffic sign recognition for rule enforcement, etc.

The steps to create an HD map are the following:

- calibrate the LiDAR and GNSS/INS
- generate a LiDAR point cloud map from raw LiDAR scan results when driving in the target area
- generate vector map using the point cloud map and the road map of the target area

The vector map generated at the last step along with the point cloud map will serve as our completed HD map that can be used by Autoware.

1) LiDAR and GNSS/INS Calibration: The LiDAR and GNSS/INS must be calibrated to generate a high-quality LiDAR point cloud map without duplication or ghosting of the real objects. A multi-stage extrinsic calibration method, *LiDAR2INS* [7], was used that effectively utilizes environmental planar features to enable fast and accurate calibration. This process requires data of repeated figure-8 maneuvers by the vehicle; see Fig. 3. The calculated calibrated parameters by *LiDAR2INS* based on the recorded data by our vehicle are summarized in Table I.

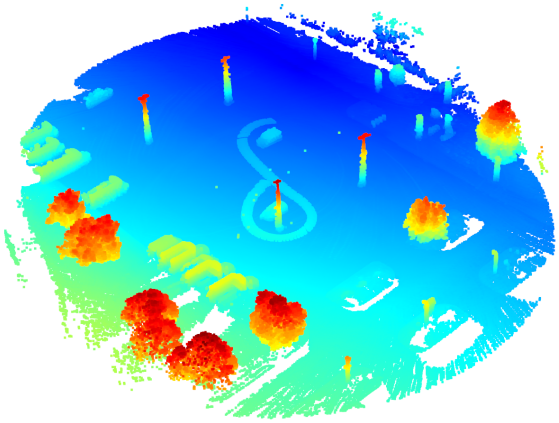

Fig. 3: LiDAR and GNSS/INS Calibration, Figure-8 Pattern

TABLE I: LiDAR and GNSS/INS Calibration Parameters

$Extrinsic_{Trans} =$	$\begin{bmatrix} 1.471 & 2.293 & 1.278 \end{bmatrix}$
$Extrinsic_{Rot} =$	$\begin{bmatrix} 0.99925572 & -0.00225209 & 0.02664469 \\ -0.00166006 & 0.99485878 & -0.01421335 \\ -0.03853884 & -0.10124692 & 0.99954392 \end{bmatrix}$
$Extrinsic_{RPY} =$	$\begin{bmatrix} 0.9974 & 0.0 & 0.0720 \\ 0.0 & 1.0 & 0.0 \\ -0.0720 & 0.0 & 0.9974 \end{bmatrix}$

2) Point Cloud Map Creation: With the LiDAR and GNSS/INS calibrated, we can properly collect route LiDAR data of the target area. Data collection requires driving the full route of the target area while continuously recording the LiDAR, GNSS, and INS data. Once the route LiDAR data has been collected, post-processing is complete, which optimizes the point cloud by removing duplicate points that are very close to each other and creating a map that starts and ends at the same point in 3D space. During the collection process, the point cloud scans can slowly drift from each other. If the full route during the data collection was a loop, the start and end points may not align. To address this misalignment and achieve loop closure, the LIO-SAM mapping framework was used [8]. Adapted from LeGO-LOAM [9], which uses an iterative closest point method, LIO-SAM formulates LiDAR-inertial odometry using a factor graph, enabling the integration of various relative and absolute measurements, including loop closures, from multiple sources as factors into the system. Fig. 4 illustrates the LiDAR point cloud map overlay on the Google Map view of our test track. As it can be seen, the point cloud matches with the landscape well after the post-processing.

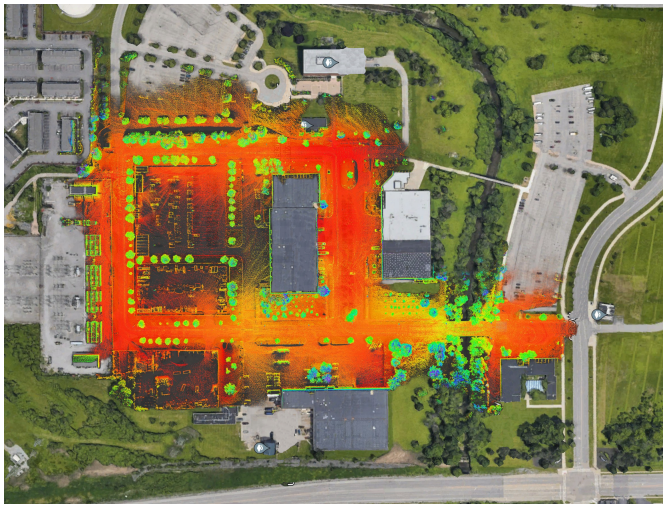

Fig. 4: Google Map of our test track overlaid with post-processed LiDAR point cloud

3) Vector Map Creation: The final step in the process involves using the post-processed point cloud map as a basis to create the vector map. The vector map will contain information about road geometry, lane boundaries, traffic control devices and rules (e.g. traffic stop signs, speed limits), crosswalks, and other critical features at user preference. We use the Vector Map Builder tool by Tier IV [10], which takes the post-processed LiDAR point cloud and allows for the manual creation of a vector map. Once imported, we set the Military Grid Reference System (MGRS) zone information for the point cloud and set the output format to be the Autoware compatible "Lanlet2". We use the target area road map as a guide to manually draw lane lines overlaid on the point cloud map, indicating the drivable lane area. Other road information may also be added to the map, such as speed limits for lane segments, direction of travel, traffic control devices, etc. Fig. 5 illustrates the drivable lane area created in Vector Map Builder.

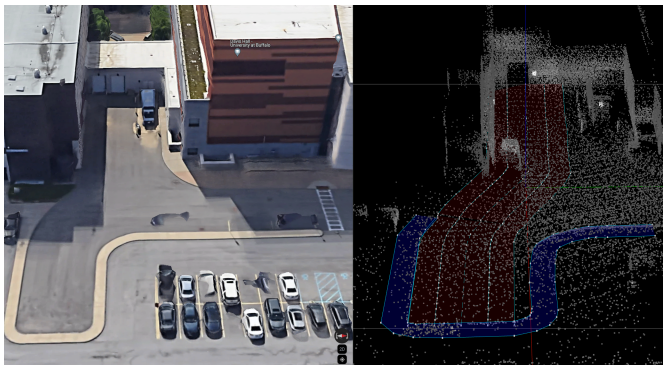

Fig. 5: Vector Map Lane Creation

With the point cloud and vector map generated, we now have the two pieces that form the HD map, Autoware's localization module can function properly. The module first

leverages GNSS-based coordinates to provide an estimate of the vehicle's position in the point cloud. It then uses the normal distribution transformation (NDT) matching technique to align point cloud data with the point cloud in the HD map, further refining the localization result. Fig. 6 shows the NDT matches and the vehicle being placed in the map based on those matches.

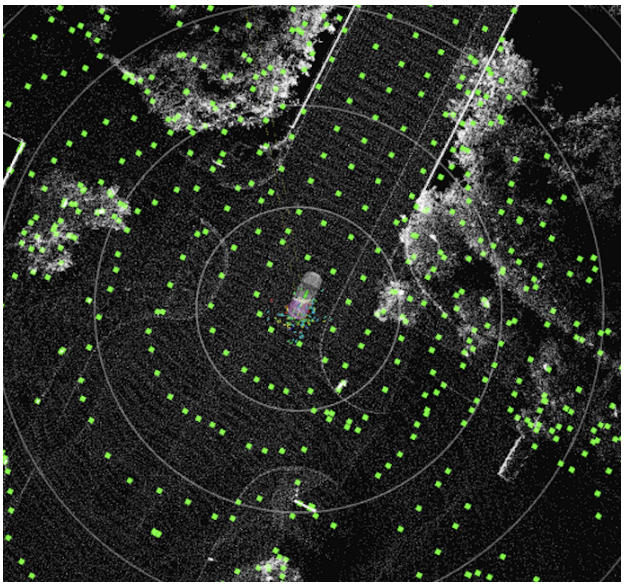

Fig. 6: NDT Localization on Test Track

B. Autonomous Driving Performance

The procedure for driving with Autoware ADS is as follows. First, all necessary drivers and nodes are properly launched. An RViz window will launch, visualizing the target area with the vehicle properly localized. Then, a goal pose can be set in the RViz window, which serves as the destination point and pose for the vehicle. Once the goal pose is configured, Autoware planning modules generate an optimal path from the current location to the target while staying within the allowable area. Once the path is generated, the system then transitions to autonomous mode, enabling the vehicle to follow the planned path using real-time localization, perception, and control inputs. The perception and localization modules operate in real time, accounting for moving obstacles within the environment. In Fig. 7, a screenshot of the RViz window is shown, which is when the Autoware ADS is driving the vehicle autonomously. The object detected (if any) will be shown in the view. The green arrow indicates the planned path while its color (green means accelerating, red means decelerating), and the steering speedometer and indicator light arrows show the control actions.

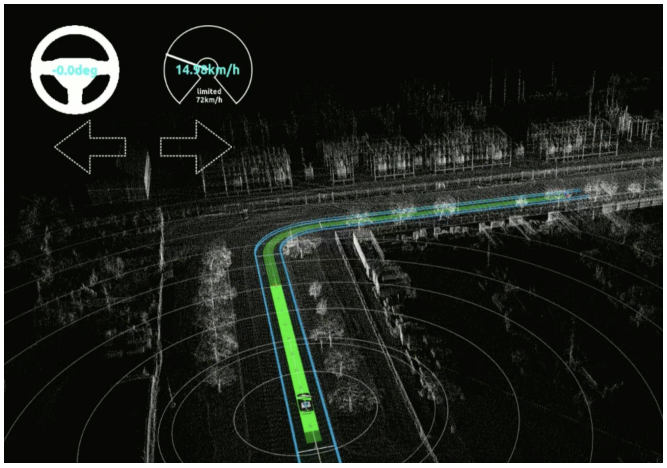

Fig. 7: Planned Drive To Goal

The successful operation of these procedures demonstrates that the software and configuration updates, along with the calibration and HD map generation we reported, are functioning properly. This means a working vehicle platform testbed that can serve as the baseline for further research is available.

To showcase the drive performance, we conduct test runs on our test track. A point cloud map of the test track loop is shown in Fig. 8. Our test run will start at the beginning of our test track loop, drive around the loop, make three right turns, then stop at a stop sign, and make a final left turn back to the starting position. In order to create a baseline comparison, and because of the manual process of creating the vector map lanes, we decided to drive the test track two times, once manually by a human and once in autonomous mode. The manual drive serves as our human baseline & ground truth trajectory, with us driving down the center of the lane, just as we set our vector map to have the vehicle drive down while in autonomous mode. We recorded the position of the vehicle, using the onboard GNSS system with RTK corrections giving us position down to 3 cm, along with the vehicle's steering angle and velocity, from the vehicle DBW, for both runs.

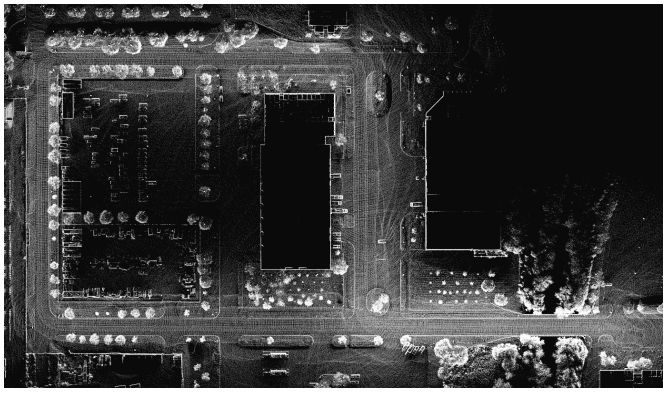

Fig. 8: Point Cloud Map of our Test Track

In Fig. 9 the comparison results of the test drives are plotted. In all panels, the autonomous run by Autoware is plotted as red

dashed curves, while the baseline run by human is plotted as green solid curves. As shown in panels (a) - (c), the drives were very similar with little deviation, indicating that the Autoware ADS can drive the vehicle following the center of the lane well. In panel (d), the speed profile is plot against the distance traveled. The Autoware run followed the speed limits (target speed, shown as the blue dashed line) on each road segment, as we set them in the vector map, while the human driver drove at a higher speed. We remark that at a distance of 880 [m], there is a stop sign, and Autoware ADS detected it, and executed a stop briefly according to the traffic rule, then resumed driving. There are three right turns followed by one left turn in the loop. In panel (e) the steering angle profile are plotted against distance traveled. As can be seen, the steering maneuvers by the Autoware ADS closely mimic those of humans, both in magnitude and timing. Overall, the performance of Autoware ADS is satisfactory and can serve as a baseline for further research and development.

V. Simulate with Autoware ADS

In this section, we report our effort to establish a simulation environment to develop and test with Autoware ADS. We describe our survey of simulation platforms before making a design selection, and showcase our progress in creating a realistic digital twin of our test track.

Autonomous driving simulation tools enable researchers to validate and compare ADS design choices on a variety of customizable scenarios in a safe and efficient way. Researchers, including those who do not have access to a real vehicle, can reuse real-world test cases or generate new scenarios to conduct research on autonomous driving techniques. As a result, simulation tools play a pivotal role in the research and development of autonomous driving and the development of these tools account for a major stream of autonomous driving research.

Our goal with simulation is to pick one open-source simulation tool, integrate it with Autoware ADS, and establish a realistic representation of our test track, therefore allowing productive research and development of autonomous driving techniques. To this goal, our major effort has been spent on simulation tool evaluations, digital twin building of our test track, and integration into the simulation tool.

A. Simulation Environments

Our ideal simulation tool needs to satisfy the following two requirements: For one, it should support an accurate scenario representation for the target research orientation. For a full stack ADS development, this means good graphical and scene representation (e.g., road geometry, weather and lighting conditions), simulated sensor kits, as well as realistic pedestrian and traffic behavior. This requirement mitigates the design gap and enables a seamless transition from validating in simulation to validating in the real world. For the other, it should seamlessly integrate with an ADS. For a full stack ADS development, this means that the input and output definition and specification for the ADS are identical between

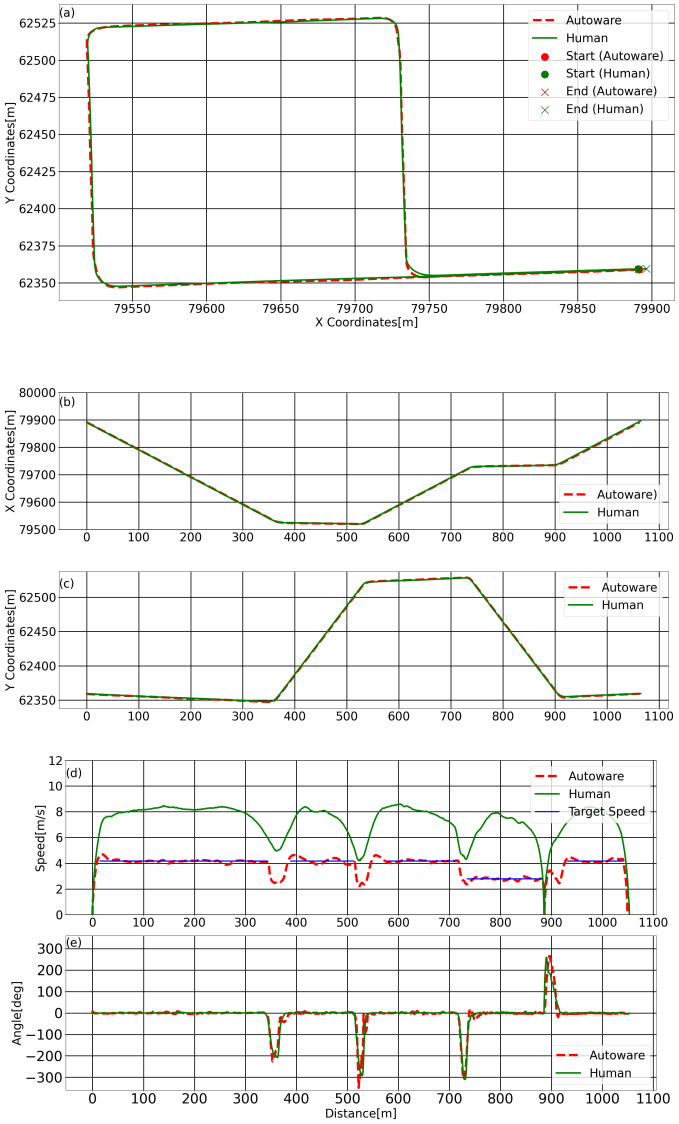

Fig. 9: Test Track Drive Results

the simulation and the real vehicle, i.e., zero implementation gap between the simulation and the real vehicle.

With these two requirements in mind, we surveyed popular open-source simulation tools to identify a foundation for our digital twin. Specifically, to meet the first requirement when evaluating the tools, we also look at their capability of integrating with other specialized simulators for certain scene representations, such as SUMO for traffic flow representation [11] or NS-3 for network communication representation [12].

1) RoadRunner: RoadRunner by MathWorks (formerly by VectorZero), is a 3D environment modeling and simulation tool designed for creating and testing scenarios for autonomous driving systems, advanced driver-assistance systems (ADAS), and other transportation applications [13]. It provides tools for designing detailed, realistic virtual environments, including roads, intersections, traffic signs, and complex terrains. These environments are used to simulate and validate vehicle

perception, planning, and control algorithms. RoadRunner can be a useful tool for developers working on safety-critical systems, enabling cost-effective testing and development in virtual environments before deploying on physical roads.

When we started working on building the digital twin of the test track, we used RoadRunner to create our model. The ability to import a road network from OpenStreetMap (OSM) was very helpful to get started. We eventually determined that OSM did not provide us with enough fidelity to create a useful digital twin. The simulation portion of RoadRunner also had some deficiencies as we were unable to integrate with our other simulators, SUMO & NS3, as data sources. As we started looking at the next simulator, we found that we were able to export most of our model via OpenDRIVE & OpenSCENARIO into CARLA, and chose to migrate to that platform.

2) CARLA: CARLA (CAR Learning to Act) is an open-source simulator designed for the development, training, and validation of autonomous driving systems [14]. It provides a platform for researchers, developers, and industry professionals to simulate urban and highway driving environments, test driving policies, and validate algorithms in a realistic yet controlled setting. CARLA is widely used by academia and industry for developing autonomous driving technologies, enabling a risk-free and cost-effective approach to validate algorithms before real-world deployment.

When we started working with CARLA, we were able to import most of our models from RoadRunner. CARLA provided better support for working with multiple data sources, including SUMO. Although this got us up and running quickly, we also ran into different issues. The process of keeping the RoadRunner model up to date, just to export and import into CARLA, became tedious and problematic. Anything that was created directly in CARLA would be lost on the next import of an updated RoadRunner model. We also found that our road network, originally imported from OSM, was incomplete or changed during the import process. This occasionally required several import attempts before success. One of our issues with CARLA was its lack of integration with the newest versions of Autoware. Since one of our requirements for the simulation tool is that it can assist in the Lincoln MKZ Autoware Universe update, we moved on from CARLA.

3) AWSIM & AWSIM Labs: Our Autoware difficulties were eventually relieved with the integration of AWSIM, which provided the same benefits as CARLA while remaining compatible with Autoware Universe. AWSIM [15], an open-source Unity-based simulator developed by Tier IV, was designed to integrate seamlessly with Autoware. It features realistic physics, sensor models (LiDAR, cameras, GNSS, IMU), and support for dynamic objects like pedestrians and traffic. AWSIM enables testing and evaluating perception, planning, and control algorithms in a safe virtual environment. It is optimized for ROS2 communication, making it ideal for working with Autoware.

AWSIM Labs [16] is a specialized fork of AWSIM, developed by the Autoware Foundation to foster autonomous driving innovation. While it builds on AWSIM's core capabilities, it emphasizes enhanced simulation features, such as detailed traffic environments and high-definition digital twins. AWSIM Labs uses a more efficient rendering pipeline compared to AWSIM. AWSIM Labs is a newer development and we are still evaluating its benefits over AWSIM.

We surveyed three simulation tools, and chose AWSIM as the tentative tool for our digital twin. After comparing RoadRunner, CARLA, and AWSIM we decided AWSIM offers comparable graphics quality and integrates with Autoware Universe more tightly than RoadRunner or CARLA. We are actively using the AWSIM simulation engine while attempting to maintain portability to other engines as new developments arise.

B. Digital Twin for Test Track

With AWSIM selected and configured as the simulation tool, we now shift focus to creating a realistic digital twin model of the test track. We want to leverage the perception and localization capability of the Autoware ADS running on the physical vehicle to achieve a realistic digital twin model for full-stack ADS research and development.

Specifically, the Poisson surface reconstruction algorithm [17] by the open-source tool CloudCompare [18] is applied on the point cloud map used for HD map generation, which outputs a mesh representation of the environment. Once imported into the digital twin, the mesh is used for localization purposes with the virtual LiDAR sensor also hidden from the camera's view. This allows us to separately develop the visuals of the simulator without loss of localization accuracy.

Without an accurate visual mapping from the physical to the digital world, certain test results, findings, and validations derived from the digital twin are less valuable. To reliably extrapolate discoveries made using a digital twin, reality must be mirrored as closely as possible. Now that we have a working digital twin of our test bed, our focus has moved to improving the graphical fidelity of the simulator. Next, we explore using Open Street Maps, Google Earth, and Gaussian Splatting for the 3D reconstruction and present results to demonstrate that evolution.

1) Open Street Maps: Open Street Map [19] (OSM) provides a strong visual foundation for the digital twin. OSM models have proven useful throughout development by outlining the relative location and scale for higher-fidelity visual models while remaining computationally inexpensive. Fig. 10 (a) illustrates the preliminary OSM models that we imported into our digital twin.

2) Google Earth: Due to OSM's lack of precise model geometry and accurate texturing, we upgraded the digital twin's graphics to a quality comparable to Google Earth [20]. Our initial solution was to run photogrammetry techniques on screenshots taken from Google Earth using MeshRoom [21]. Although this method provided higher quality results than OSM, the process was not scalable and only produced quality renderings at the object level (capturing one building as opposed to capturing the entire scene).

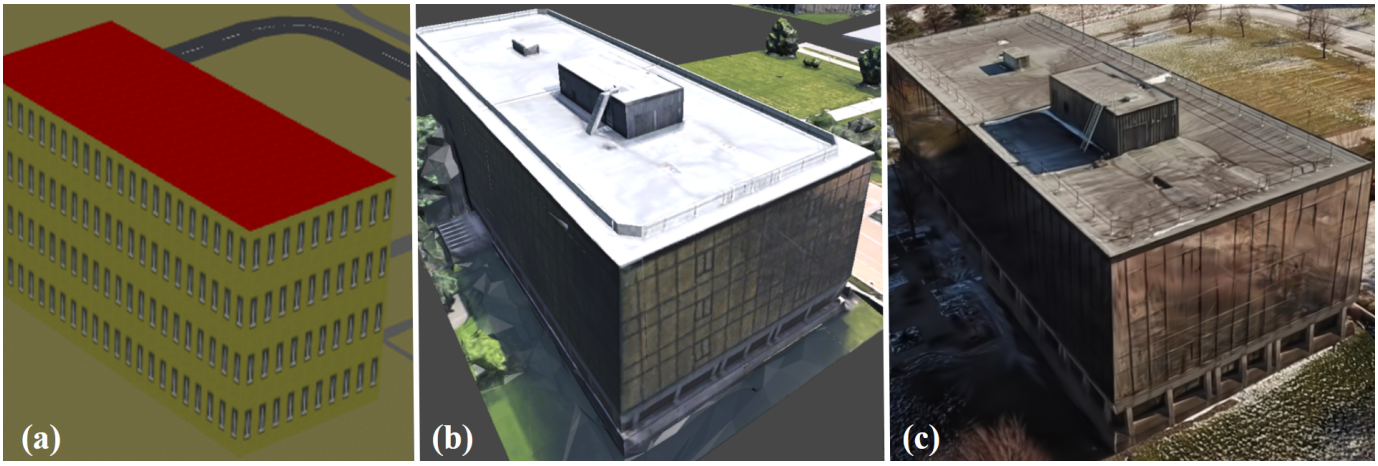

Fig. 10: Comparison of Digital Twin Reconstructions

To address the scalability concerns, we queried Google Earth textured models using a web browser and utilized open-source tools RenderDoc [22] and MapsModelImporter [23] to capture and export the Google Earth textured models. This pipeline was more streamlined than the photogrammetry method and provided a means for scene-level reconstruction, resulting in a faster asset turnaround time. Most of the visuals in the current version of our digital twin utilize this reconstruction technique, see Fig. 10 (b). Although a significant improvement from OSM, the issue of poor texture quality and imprecise model geometry remains. This deficiency becomes most apparent when objects are observed from the perspective of the vehicle.

3) Gaussian Splatting: Gaussian Splatting is an emerging 3D reconstruction technique with unprecedented capabilities that can "achieve state-of-the-art visual quality while maintaining competitive training times and importantly allow high-quality real-time[...] novel-view synthesis at 1080p resolution." [24]. To experiment, we flew a DJI Mavic Pro drone over the test track and generated a splat of a building on the route, using the published code. Fig. 10 (c) shows a Gaussian Splat rendering of this reconstruction. This method has by far provided the best visual quality and supports import directly into Unity via a third-party plugin [25].

4) Running the Simulation with the Digital Twin: After selecting a simulation tool and creating a preliminary digital twin model of our test track, we gained the ability to experiment and validate in our simulation environment. Fig. 11 shows our virtual vehicle localizing in our test track digital twin.

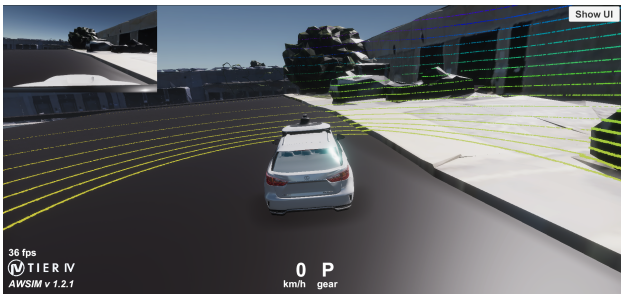

Fig. 11: Using AWSIM with our Digital Twin Model

VI. Discussion & Future Work

In this section, we summarize the lessons learned and propose future research directions.

A. Autoware

While working with Autoware throughout this paper, a few shortcomings were identified and in some instances provided us with larger than expected challenges. We detail those here in hopes of bringing them to light so future users are aware and as improvement notes for developers.

1) Missing or Incomplete Features: While building out this testbed, we would often run into a missing feature that we thought would be important. We found ourselves building out these features, which detracted from our time developing the rest of the testbed. Features were often released as we were progressing on a parallel path. Two of these examples are pure GNSS localization and GNSS / LiDAR fusion-based localization, both of which were not available when we started this project. With such features available now, we can focus on integrating rather than reinventing them.

2) Documentation: The other roadblock was incomplete or missing documentation of the needed module or process with respect to their usage and configuration. This is slowly being addressed and may differ currently from how we originally configured our testbed. While this allows us to try some of the methods in a different way in the future and conclude the pros and cons of each method, it hinders our progress toward developing AV technologies. This is one of the main motivations of this paper, and we urge the community to spend time on documentation regarding setup and configuration, in order to benefit other researchers.

3) Build Management: Another challenge that was a chore to manage was making sure the research team was building the same version and using the same builds across various

computer setups. This is complicated further when we have students joining and leaving the team, and getting new members up and running quickly and efficiently. Because of this, we are in the process of using Autoware's Docker container system, which should make this effort much more manageable.

B. Simulation and Digital Twin

Developing a digital twin that can be trusted to validate an ADS is not a trivial task. As long as there remains a disparity between the physical and simulated environments, there is potential to improve the simulation. This project resulted in a working digital twin that serves to benefit both autonomous vehicles and other digital twins that require graphical reconstructions. We will continue focusing on refining the digital twin and simulation pipeline.

1) Gaussian Splatting: With the successful import of our test track digital twin into our simulation environment, and with emerging research surrounding Gaussian Splatting [24], we intend to update our digital twin model to fully support Gaussian Splats as the primary graphical component of the simulation. In doing so, we hope to expand the digital twin to integrate with splats sourced from the cameras and LiDAR onboard the Lincoln MKZ. We believe the use of Gaussian Splatting technology in AV simulators is crucial to developing a visually robust and more accurate digital twin model. Our future efforts also intend to expand Gaussian Splatting to reconstruct graphical environments from autonomous vehicle datasets. We see great potential for further research on Gaussian Splat integration, with work to be done on dynamic Gaussian Splatting, optimization, and unbounded scenes.

VII. Conclusion

In this paper, we reported our effort on enabling our old 2017 Lincoln MKZ to be able to drive and simulate with the latest Autoware Universe autonomous driving stack. We detailed the steps in setting up localization, creating an HD map, setting up a simulation pipeline, and building digital twins of the test track. We demonstrated satisfactory baseline performance in the vehicle and in our digital twin simulation. We also discussed on the lessons learned, current limitations and future project research directions in maintaining the testbed.

Acknowledgment

This material is based upon work supported by the Federal Highway Administration (FHWA) under Grant No. 693JJ32540148. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the Author(s) and do not necessarily reflect the view of the Federal Highway Administration.

References

[1] G. Mehr, P. Ghorai, C. Zhang, A. Nayak, D. Patel, S. Sivashangaran, and A. Eskandarian, "X-car: An experimental vehicle platform for connected autonomy research," *IEEE Intelligent Transportation Systems Magazine*, vol. 15, no. 2, pp. 41–57, 2023.

[2] H. Guo, J. Li, N. K. Saravanan, J. Wishart, and J. Zhao, "Developing an automated vehicle research platform by integrating autoware with the dataspeed drive-by-wire system," SAE Technical Paper 2024-01-1981, Tech. Rep., 2024.

[3] Autoware Foundation, "Autoware," https://github.com/autowarefoundation/autoware, 2024, accessed: 2024-01.

[4] Baidu, "Apollo," https://github.com/ApolloAuto/apollo, 2024, accessed: 2024-01.

[5] Comma.ai, "openpilot," https://github.com/commaai/openpilot, 2024, accessed: 2024-01.

[6] AutonomouStuff, "autonomoustuff," https://autonomoustuff.com/solutions/autonomous-vehicle-control, 2024, accessed: 2024-01.

[7] G. Yan, J. Pi, C. Wang, X. Cai, and Y. Li, "An extrinsic calibration method between lidar and gnss/ins for autonomous driving," *arXiv preprint arXiv:2209.07694*, 2022.

[8] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and R. Daniela, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5135–5142.

[9] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4758–4765.

[10] E. Yurtsever, J. Lambert, Y. Ninomiya, and K. Takeda, "T1: A 3-hour primer on real world autonomous driving: Hands-on introduction to autoware," *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2018.

[11] 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*. ThinkMind, 2011.

[12] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[13] Mathworks, "Roadrunner," https://www.mathworks.com/products/roadrunner.html, 2024, accessed: 2024-01-12.

[14] 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.

[15] TIER IV, "Awsim: Autonomous vehicle simulator," https://github.com/tier4/AWSIM, 2024, accessed: 2024-01-05.

[16] Autoware Foundation, "Awsim-labs: Autonomous vehicle simulation labs," https://github.com/autowarefoundation/AWSIM-Labs, 2024, accessed: 2024-12-01.

[17] M. Kazhdan, "Poisson surface reconstruction," https://github.com/mkazhdan/PoissonRecon, 2024, accessed: 2024-02-12.

[18] D. Girardeau-Montaut, "Cloudcompare - 3d point cloud and mesh processing software," https://www.danielgm.net/cc/, 2024, accessed: 2024-02-12.

[19] OpenStreetMap contributors, "Openstreetmap," https://www.openstreetmap.org, 2024, accessed: 2024-01-10.

[20] Google, "Google maps," https://www.google.com/maps, 2024, accessed: 2024-12-20.

[21] AliceVision, "Meshroom: A 3d reconstruction software," https://github.com/alicevision/Meshroom, 2024, accessed: 2024-02-12.

[22] B. Karlsson, "Renderdoc, version 1.19," https://github.com/baldurk/renderdoc/releases/tag/v1.19, 2022, accessed: 2024-03-20.

[23] E. Michel, "Maps models importer," https://github.com/eliemichel/MapsModelsImporter/releases/tag/v0.6.0, 2023, accessed: 2024-03-20.

[24] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering." *ACM Trans. Graph.*, vol. 42, no. 4, pp. 139–1, 2023.

[25] A. Pranckevičius, "Unity gaussian splatting," https://github.com/aras-p/UnityGaussianSplatting, 2024, accessed: 2024-08-24.