Driving On: Taking Our Autoware AV Testbed Further

Harsha Meka*, Bharadwaj Gutha*, Steven Korzelius*, Chaozhe R. He, Adel W. Sadek, Chunming Qiao

***Abstract*—Open-source autonomous driving stacks now support cutting-edge features such as using end-to-end LLMs for full autonomy in test environments, yet real-world deployments of these systems remain small. As the stacks evolve, primary functionality improves, interfaces shift, and maintaining a bridge between the software stack evolution and the physical world of the vehicle can be challenging. This paper presents our efforts to keep our testbed up to date with the latest version of Autoware (v0.45.1), removing legacy custom packages to improve performance and maintainability. We detail the updated vehicle interfaces, sensor integration, refined LiDAR-based localization, Docker based image deployment and HD map creation pipeline. The system is evaluated through real-world autonomous driving experiments by validating localization and map creation, verifying perception & obstacle detection, and optimizing planning and control performance. We discuss lessons learned in building and operating an experimental platform using open-source Autoware, highlighting how autonomy requires iterative, ongoing development. Our work is fully open-sourced and can be found at https://github.com/ub-cavas/ub-lincoln-docker**

***Index Terms*—autonomous vehicle, Autoware, HD-map, SLAM, Docker**

I. Introduction

Development and maintenance of Autonomous Driving System (ADS), given their complexity, has occupied the major attention of many research groups that own Autonomous Vehicle (AV) platforms, as opposed to pure algorithm development. Real-world deployment of AV needs even more safety considerations and regulatory requirements compliance of ADS. In contrast, industrial AV development (Waymo, Zoox, Tesla, etc.) can rely on proprietary software platforms supported by substantial investment in research, development, and large-scale testing infrastructure, at a scale that cannot be met by academic research teams. There are a few open-source ADS, such as Autoware [1] and Baidu Apollo [2] along with recent Vision Language Action models from NVIDIA [3]. These stacks address different aspects of autonomy such as simulation, AI reasoning, and algorithm development. Among them, Autoware stands out as it represents one of the most complete open-source ADS at present, providing a modular, Robot Operating System (ROS) 2 based pipeline that supports real-world sensing, localization, perception, planning, and control.

Despite the fact that Autoware provides a complete open-source ADS, bridging the gap between simulation and a real-world deployment remains a large challenge, especially in the academia. Real-world deployment comes with many challenges and constraints related to hardware interfacing, sensor data flow, timing, safety oversight and real-world driving conditions, which are not fully captured in simulations.

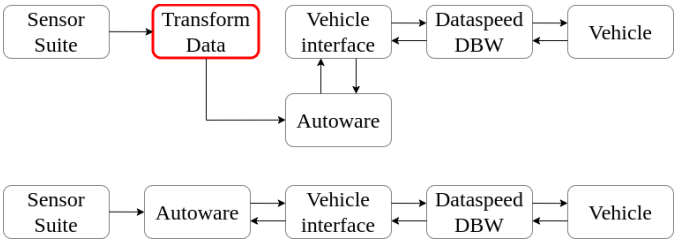


Fig. 1: Old (Top) & updated architecture (Bottom)

In this paper, we report on the evolution of the system architecture from our previous deployment [4]. Our old architecture, shown in Fig. 1 top row, relies on a vehicle interface package that translates the Drive-By-Wire (DBW) messages into Autoware compatible messages and vice versa . The custom transform package serves as a sensor pipeline, feeding data to the corresponding ROS2 topics. While these components make the ADS compatible with early versions of Autoware, they hinder a smooth upgrade to future versions. Bypassing portions of the legacy Autoware pipeline with these custom nodes created unnecessary computational overhead, causing delays in sensor data processing. This made the system increasingly difficult to maintain and upgrade as Autoware evolved.

Further, we focus on updating our prior architecture to achieve full alignment with the latest upstream Autoware design, by updating the vehicle interface and moving away from the custom transform packages as shown in Fig. 1 bottom row. These custom packages, which worked as a sensor pipeline, feeding modified sensor data into the respective ROS2 topics, some of which bypassed downstream tasks, led to computational overhead and delays in processing sensor data. To overcome these issues, this work adopts the canonical flow of the Autoware pipeline, which collectively improved the performance and code maintainability. This restructuring, along with the dockerization of our workspace, produced improved compatibility with upstream pipelines, enhanced system maintainability, and increased deployment reproducibility by eliminating dependency and setup variability issues.

Thus, the key contributions of this paper are:

- A real-world autonomous driving deployment using Autoware Universe v0.45.1.
- A dockerized container image with a full Autoware software stack enabling reproducible deployment and simplified dependency management.

* These authors contributed equally in this work. Harsha Meka, Bharadwaj Gutha, Steven Korzelius and Chunming Qiao are with the Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY, USA {hmeka, bgutha, smk32, qiao}@buffalo.edu. 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.

- Validation of LiDAR-based localization in real-world driving scenarios using an improved High Definition (HD) mapping pipeline.

The remainder of this paper is organized as follows: Section II provides an overview of the Autoware Universe and outlines the system design of the work. Section III highlights the current upgrades to the prior architecture. Section IV presents HD map creation pipeline and localization performance. Section V describes the experimental testing setups, lessons learned, and system limitations. Section VI concludes the paper with some discussion and details of possible future work.

II. Background and System Design Principles

Modern Autonomous Driving Systems, such as Autoware, provide a modular setup that integrates the sensing, hardware, and vehicle actuation into a complete decision-making system, which can speed up academic research and real-world deployment. Being open-source, it also allows rapid development within the stack via regular foundational updates along with community contributions. Here, we briefly introduce the ADS, Autoware.

Autoware was first developed by Shinpei Kato at Nagoya University, which was built using ROS1 as the underlying middleware, initially focused on research and development. Autoware.AI was first presented at ROSCon 2017 as the first open source all in one autonomous driving stack, later evolving towards solving commercial and industrial solutions such as Autonomous Valet Parking and Autonomous Cargo Delivery at airstrips. The previous versions had some limitations so to overcome them they migrated into Autoware.Auto in late 2018 as the next generation successor of Autoware.AI. This new version was based on ROS2. Autoware Core/Universe, released in 2021, is also ROS2 based, adopts a new architecture which provides more modularity, configuration driven customization, and upstream compatibility helping both real-world and prototyping deployments.

Autonomous driving systems represent a well-organized pipeline which fuses sensor data, plans the next actions, converts them to control commands which are sent to the vehicle to navigate in real time. This pipeline generally represent a sequential flow of sensing, localization, perception, planning, and control. Each of these components represent the state of the environment or the vehicle thus enabling the stack to make precise decisions under real-world constraints.

The system components and their data flow are described below:

- Sensing: This is the first stage of the pipeline where the raw data from the sensors such as LiDAR, Camera, GNSS, IMU, Radar are acquired and then processed into time-synchronized data streams and generalized format, which acts as a baseline for the downstream tasks.
- Localization: This component of the pipeline is responsible for locating the position and orientation of the vehicle with respective to the external world.
- Perception: This module processes the data received from the sensing module which allows the system detect world objects such as vehicles, lanes, and traffic signals.
- Planning: The planning component is responsible for path planning and decision making aspects of the system which includes picking the most efficient paths while avoiding obstacles in the environment.
- Control: This is the final module which is responsible for controlling the vehicles maneuvers including acceleration, braking and steering, based on the environment.

Autoware comes with a great deal of modularity which makes customization attractive and allows for custom packages or interface layers to bridge gaps between hardware and the given base autonomy stack.

III. System Overview and Deployment Setup

In this section, we lay out the developed system hardware and software architecture. Our deployment setup on the hardware side is as follows:

- Vehicle Platform: Lincoln MKZ 2017 Hybrid.
- DBW System: Dataspeed.
- Sensors: Velodyne VLP32 LiDAR, Novatel OEM7 GNSS+INS with STIM600 a 6-axis IMU.
- High Performance Computing (HPC) unit: Neousys Nuvo 10108GC with Intel i9-14th Gen Processor, 64GB RAM, 12GB NVIDIA RTX 4080 Super.

In order to fully support the latest version of Autoware, the HPC needed an upgrade since the cuDNN, tensorRT and spconv versions used by Autoware were not supported by our original NVIDIA GeForce 1080Ti graphics card. The cuDNN, tensorRT and spconv libraries are essential for running the inference models that ships with Autoware by default. These deep learning models are necessary for the perception module, which performs obstacle detection and tracking. The planning and control modules also depend on this processed perception data.

While we used the following packages on the software side:

- Operating System: Ubuntu 24.04, Docker, NVIDIA Container Toolkit.
- ROS2 Packages: Vehicle Interface, Sensor Kit, Vehicle Kit, nebula_ros driver for LiDAR data.

Notable changes were made in the Sensor Kit and the Vehicle Interface packages. The Sensor Kit was updated so that the sensor drivers are launched from a single script, with all topic remapping applied in-place rather than changing the drivers' source code, to provide greater flexibility and modularity. The sensor kit was updated to match and work with the latest version of Autoware. Similarly, our Vehicle Interface also required additional topics to be included on Autoware's side compared to our previous version, due to changes in how Autoware handles control messages in later versions. The Vehicle Interface was updated with turn signal indication, actuation status/command, control mode status/command, and a key topic subscription from Autoware, which is the `emergency_command`. These topics were crucial for the

proper use of Autoware with the Vehicle Interface. The `emergency_command` was necessary to be included in the Vehicle Interface because when this topic is not subscribed, Autoware does not send the actual control messages needed to move or brake the vehicle. This is also a standard practice in AV space, since this enables the system to send the Minimum Risk Maneuver / Minimum Risk Condition status by setting the `emergency_command` to 1.

IV. LiDAR-Based Localization and Mapping

A. Localization Strategy

Autonomous driving depends on many things, localization being a primary one. LiDAR based localization is very reliable for real-world applications when compared to other approaches like vision-based methods, as it is very robust to light variations and a structured outdoor environment. LiDAR based localization is the native approach supported by the Autoware pipeline.

Autoware's localization strategy starts with incoming sensor feed from the LiDAR scans, which are matched against the prerecorded HD pointcloud map using the NDT scan matcher to get the vehicle pose with respect to the world. However, the initial pose is obtained using Global Navigation Satellite System (GNSS) readings to get the initial pose estimation, which is then taken over by the NDT algorithm to accurately update the position of the vehicle with cm-level accuracy based on the map.

B. Experiments with SLAM

Simultaneous Localization and Mapping (SLAM) is a key component required for efficient functioning of localization within Autoware. As previously mentioned, Autoware requires an HD map of the area it will operate in as a frame of reference. The HD maps are going to be a pointcloud map which can be built using only LiDAR, LiDAR-IMU or LiDAR-IMU-GNSS. Although many SLAM algorithms are available for building these maps, each has its own purpose and methodology. A very popular one being used for this type of work is LIO-SAM [5], which is an improvement from LOAM [6]. We investigated several SLAM algorithms to create an accurate HD map.

Our criteria for selecting the ideal SLAM Algorithm to integrate into our workflow are:

- A working ROS2 fork or a method to use the ROS1/ROS2 bridge.
- Support for a 6 axis IMU and GNSS location messages.
- Support for loop closure and stitching multiple point cloud maps together with the goal of creating a larger map.

However, we found no existing solution that meets all our criteria and our sensor setup, so some tradeoffs had to be made. LIO-SAM [5] checked most of the boxes for us, but the algorithm highly depended on the IMU being a 9-axis one. LIO-SAM inspired SLAM implementations like LIORF [7], and `LIO_SAM_6AXIS` [8] that support 6-axis IMU exist, but we did not see promising results using them. They all started to drift/fail on larger maps for our setup. For context, our current map comprises an area of 108214 square meters, while the map we want to build is around 409806 square meters, which is around 4 times the original map.

We then explored GLIM [9], which is aimed at being an all-in-one solution for all SLAM needs. It worked well for larger maps as well, but it had manual loop closure and lacked proper GNSS implementations. A tested plugin for the algorithm to use GNSS data is a work in progress for multi session mapping using GNSS.

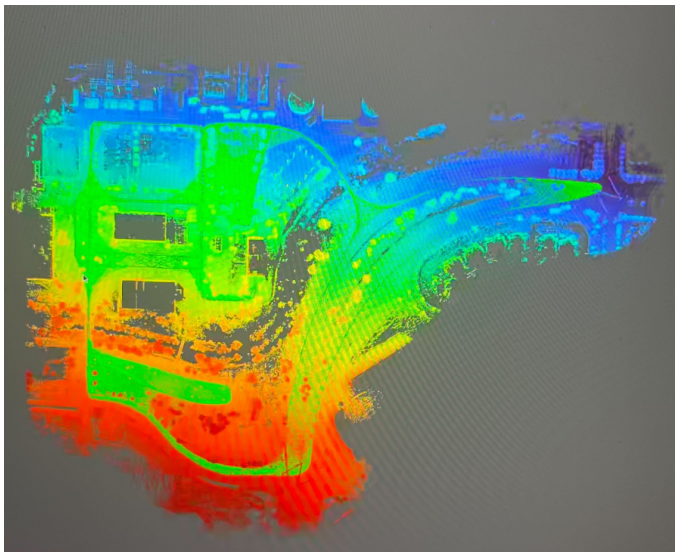


Fig. 2: Map generated by GLIM

Our mapping pipeline is explained in detail in the following section.

C. HD Map Generation and Improvements

Building an HD pointcloud map plays a crucial part in the use of Autoware. In this work, pointcloud map generation is performed using a SLAM algorithm, and use Direct LiDAR Inertial Odometry (DLIO) [10], a LiDAR-inertial odometry based system, to build the HD map. This process includes recording data as a ROS2 bag file while driving the vehicle at 5-10 [km/h] to obtain stable LiDAR scans and achieve sufficient point cloud density of the environment, while simultaneously using the IMU feed to build a closed-loop global map. The resulting SLAM generated point cloud map is the "ground truth" HD map for the system.

The raw point cloud map from the SLAM algorithm is refined by cleaning outlier points, removing noise, and applying various filters to improve the map's overall consistency using CloudCompare [11]. The processed SLAM map is then rotated about 180 degrees so that it aligns with the real-world. The rotated map is then geo-referenced along the Military Grid Reference System (MGRS) map using the `pc_utm_to_mgrs_converter` [12] ROS2 package. Autoware currently supports 3 types of coordinate systems namely Local, UTM, and MGRS. Geo-referencing the maps

allowed us to use the GNSS system to automatically obtain the initial pose of the vehicle without any manual intervention. In our case, the map was referenced to our MGRS grid zone. 17TPH is the corresponding MGRS grid code to our test area.

Once the final point cloud map was generated, the file was imported to Vector Map Builder [13] provided by Tier IV where we construct the lane boundaries, road geometry and semantic attributes such as speed limit, crosswalks and other road signs necessary which makes the map readable by Autoware. The lanelets are the crucial part that help Autoware understand the rules of the road in which it is operating in, with the resulting map shown in Fig. 3.

The biggest pain point in developing the HD map was the constant back-and-forth between the processed and converted maps. The map was aligned as closely as possible to the real world by picking a point in the real world and validating its alignment with the real world using GNSS coordinates from Google Maps. Vector Map Builder provides an option to navigate to a specific coordinate, which, given GNSS latitude and longitude, moves the map to the corresponding point in the point cloud.

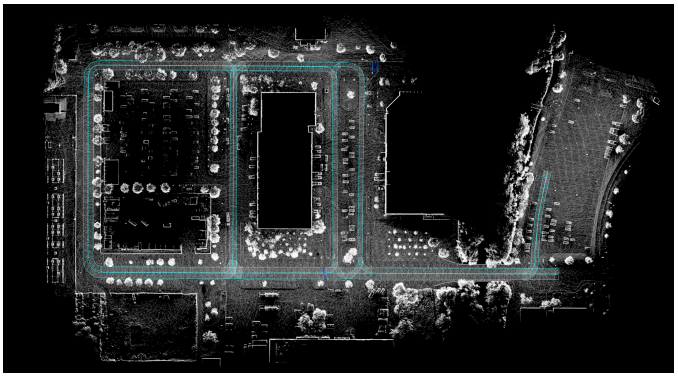


Fig. 3: Updated HD map

D. Localization Performance

Localization performance was benchmarked by running multiple runs in a real-world environment across the test area. The system showed a strong and stable pose estimate and consistent trajectory alignment with respect to vehicle position. The system's performance drops in dynamic environments or when the density of the environment decreases, increasing the likelihood of localization failure. The system automatically recovers when there are enough map features in the sensor's field of view.

V. Real-World Autonomous Driving Experiments

This section highlights real-world experiments conducted to validate the deployment of Autoware on the vehicle platform. All the test cases were performed in a real-world environment to evaluate the system's robustness and to check that the system behavior aligns with real-world observations.

A. Experimental Setup and Docker Deployment

A Docker-based workflow enabled rapid, iterative development, with updates to the stack rebuilt and redeployed directly to the container on the vehicle. Eliminating manual dependency management or host-level reconfiguration. Our Docker image is built on top of the ROS2 humble Docker image created and maintained by Open-Source Robotics. Our Docker setup uses Docker Compose, which makes it easier to use Docker images. It enables granular control of the shared file spaces, networks, USB ports to be used by the image. This allowed us seamless usage of existing networking interfaces on the host computer to be utilized by the Docker container as well. The file named docker_compose.yaml in our repository serves this exact purpose.

We also use various helper scripts that directly download various files such as the `hd_map`, `lanelet2_map`, `map_config.yaml` along with our calibrated acceleration and brake maps. The `ub_lincoln.repos` file contains the various repositories that are a part of the successful functioning of Autoware. We also have a script that sets up the DDS according to Autoware's recommendation [14] when launching the container. The Docker configuration and deployment scripts used in this work are publicly available.

B. Localization & Validation Vector Map Adjustments

Localization accuracy was evaluated by comparing the vehicle pose in real-world environments against the geometry of the test track and the generated HD map. LiDAR-based localization output was observed to align consistently with map geometry, visual representations, and Localization trajectory overlay showed strong agreement with the physical road layout.

In addition, vector map adjustments were made iteratively based on real-world testing and observation. Small deviations between lane boundaries and actual road geometry were corrected to improve the map's lane alignment with the physical world, resulting in stable path planning and navigation.

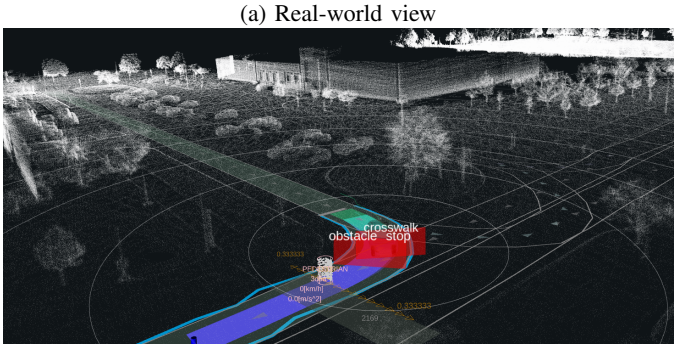
C. Perception and Obstacle Detection Experiments

Perception experiments were done on both static and dynamic objects during real-world testing. The crosswalk environment was one of the primary test cases to evaluate obstacle detection and avoidance as depicted in Fig. 4a and Fig. 4b. The perception stack successfully detected pedestrians within the crosswalk region and classified them with their semantic labels. And similar experiments were conducted with other objects, such as cars and trucks, resulting in successful detection of their semantic labels as well.

Path planning outputs were evaluated by comparing the trajectories against the real-world road layout. Generated paths followed the lane boundaries precisely which were defined in the HD map.

The accel–brake map needed to be calibrated, which was done using the accel–brake map calibrator package provided by Autoware [15]. The default map was used as the basis for calibration. We launched Autoware along with

(a) Real-world view



(b) Autoware perception

Fig. 4: Pedestrian detection during crosswalk experiments: (a) shows the physical environment and (b) shows the corresponding Autoware stack visualization.

the accel–brake map calibrator package. Upon letting the car drive autonomously, the calibrator package collected data across all acceleration and deceleration values and calibrated the map based on the actuation commands sent and the feedback received from the DBW. The autonomous driving mode only accounted for a certain range of velocities. In order to achieve broader ranges, manual driving was also performed on top of the already calibrated values to achieve smoother braking and acceleration. The calibration window is shown in Fig. 5. The figure shows a graph from the calibration process. The X-axis depicts the speed in m/sec while the Y-Axis depicts the position of the acceleration/brake pedal, positive being acceleration values while negative being the brake values. In the image, gray cells mean no data has been acquired for that specific pedal value and speed. The cell color changes from blue to red as the number of valid data in the cell accumulates. The goal is to achieve as close to red as possible and Autoware recommends to focus the ranges of 0-6 [m/s] in speed and +0.2 to -0.4 in the accel/brake pedal values for smoother acceleration and braking. The package uses two types of calibration methods to choose from, namely, `update_offset_four_cell_around` and `update_offset_each_cell`. During our calibration, we chose the default method, which was `update_offset_four_cell_around`. Autoware also provides a steering command calibrator, which was not needed in our scenario since the steering was working

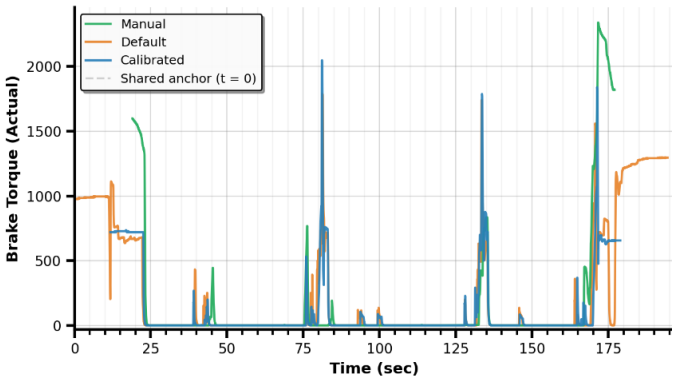


Fig. 5: Acceleration & Brake Map Calibration

smoothly and accurately.

From Tables I & II, it can be inferred that the calibrated brake map performs 52.1 [%] better than Default, with a Mean Absolute Error of 152.1 [Nm] versus 317.5 [Nm]. This demonstrates that the calibrated brake map more accurately imitates manual driving behavior compared to the default configuration.

TABLE I: Brake Torque Distribution Across Three Maps

Metric	Manual	Default	Calibrated
Mean Brake Torque [Nm]	199.2	186.5	96.4
Std Dev [Nm]	532.6	352.3	247.0
Max Torque [Nm]	2340	1788	2048

TABLE II: Pairwise Error Analysis

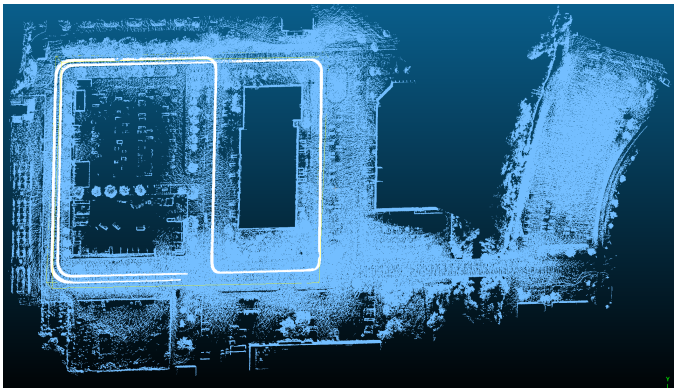
Comparison	MAE [Nm]	RMSE [Nm]	Improvement [%]
Calibrated vs Default	152.1	316.7	52.1

D. Comparison of Pointcloud Map with World Map

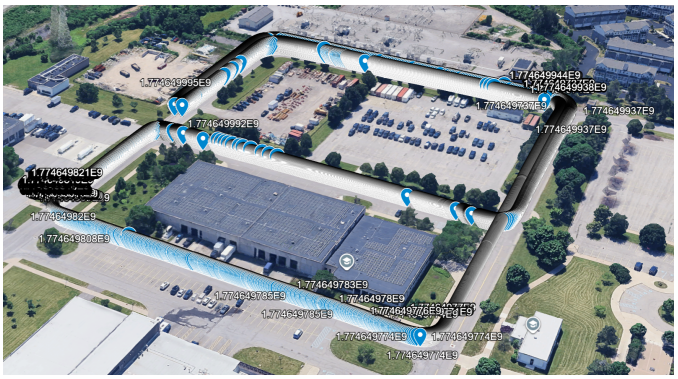
The point cloud map was benchmarked in the real-world by capturing the trajectory from Autoware and comparing it with the GNSS points collected alongside the trajectory as the ground truth. In Fig. 6a, the white line depicts the trajectory that Autoware planned from the start point and end goal. On the other hand, Fig. 6b shows the GNSS points visualized in Google Earth. This shows that the mapping method used is accurate and is a viable way to create 3D point cloud maps. However, improvements can be made to SLAM to incorporate GNSS data so that the map is aligned with the world frame.

VI. Conclusion & Future Work

In this work, we showcased a stable upstream compliant real-world deployment of our autonomous testbed running Autoware Universe. The system successfully validates localization, perception and planning in real-world conditions. Our Docker based development enables other project teams to adapt and utilize our version of Autoware for their experiments with ease and minimal environment setup issues.



(a) Autonomous Driving Trajectory



(b) Real-World Comparison of the Trajectory

Fig. 6: Comparison of the planned autonomous trajectory (a) with the corresponding GNSS ground truth visualized in Google Earth (b).

All of our work is open-sourced and can be found at this https://github.com/ub-cavas, which includes our Vehicle Interface, Sensor Kit, Vehicle Kit, and Docker repositories.

Building upon these upgrades, further work will explore high-level autonomy enhancements beyond modular pipelines, including end-to-end and agent–based decision making. With emerging research in vision–language–action (VLA) models, large language models (LLM)–driven reasoning to enable the system to be more flexible and perform better in complex environments.

Our future upgrades/improvements planned are:

- Upgrade to a 128-line LiDAR to better test and tweak the obstacle stop, obstacle cruise and obstacle avoidance maneuvers.
- Acquire & Install New Cameras for Perception tasks like Traffic Light Detection, Camera-LiDAR Localization etc.
- Try the Diffusion Model based planner and to take strides focusing on end-to-end Autonomous Driving Scenarios.
- Benchmark the Agnocast DDS with the CycloneDDS in real-world deployment scenarios.

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] Autoware Foundation, “Autoware universe,” https://github.com/autowarefoundation/autoware.universe, accessed: Jan. 2024.

[2] H. Huang, Y. Chen, H. Zhang *et al.*, “Apollo: An open autonomous driving platform,” in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1–6.

[3] NVIDIA, “Nvidia alpamayo: Open vision–language–action models for autonomous vehicles,” https://developer.nvidia.com/drive/alpamayo, accessed: Jan. 2026.

[4] 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.

[5] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, “Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping,” pp. 5135–5142, 2020.

[6] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time,” Berkeley, CA, USA, 2014.

[7] LIORF Developers, “LIORF,” 2024, accessed: Jan. 2024. [Online]. Available: https://github.com/YJZLuckyBoy/liorf

[8] LIO_SAM_6AXIS Developers, “LIO_SAM_6AXIS,” 2024, accessed: Jan. 2024. [Online]. Available: https://github.com/JokerJohn/LIO_SAM_6AXIS

[9] S. Koide, M. Yokozuka, S. Oishi, and A. Banno, “Globally consistent lidar-inertial mapping,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 365–372, 2021.

[10] K. Chen, R. Nemiroff, and B. T. Lopez, “Direct lidar-inertial odometry: Lightweight lio with continuous-time motion correction,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[11] CloudCompare Developers, “Cloudcompare,” https://github.com/CloudCompare/CloudCompare, accessed: Jan. 2026.

[12] Autoware Foundation, “Converting utm to mgrs map,” https://autowarefoundation.github.io/autoware-documentation/main/tutorials/integrating-autoware/creating-maps/converting-utm-to-mgrs-map/, accessed: Jan. 2026.

[13] Tier IV, “Vector map builder (ll2) feature overview,” https://tools.tier4.jp/, accessed: Jan. 2026.

[14] Autoware Foundation, “Autoware dds settings,” https://autowarefoundation.github.io/autoware-documentation/main/installation/additional-settings-for-developers/network-configuration/dds-settings/, accessed: Jan. 2024.

[15] ——, “Accel–brake map calibrator; autoware universe documentation,” https://autowarefoundation.github.io/autoware_universe/main/vehicle/autoware_accel_brake_map_calibrator/, accessed: Jan. 2026.