Kuura: Leveraging Eclipse Kuksa in Vehicular Data Collection and Digital Twin Creation Environment

Olli Timonen, Toni Bomström, Nicklas Stafford, Samuli Määttä, Alireza Bakhshi Zadi Mahmoodi, Tero Päivärinta and Ella Peltonen

Empirical Software Engineering in Software, Systems, and Services, University of Oulu, Finland

Abstract

Increased sensing and computing capabilities in cars are crucial for advanced traffic and driving automation. However, novel data delivery, testing, and machine learning pipelines are still needed to harness the full capabilities of automotive sensing solutions. At the same time, vehicular digital twins are needed to enable versatile testing and simulation capabilities. This paper depicts the Vehicle-In-The-Loop (VIL) cloud interface and verifies data consistency regardless of the source. The study aims to determine how data collected from simulation corresponds to real test drive data. The data is collected from both simulation and actual test drives. Utilising the MQTT protocol, data is stored on a cloud server and further fed into Unreal Engine 5, where the test drive is replayed, and its correspondence to the real drive is ensured. This work offers a new perspective on verifying data consistency between simulated and real test drives and complements the vehicle abstraction opportunities provided by Eclipse KUKSA. Our results highlight digital twin creation as a part of automotive software development and set premises for testing and validating complex use cases, such as traffic accidents and extreme weather, that can rarely or only with severe expenses be tested in real-life situations.

Keywords

Vehicular Computing, Data Transfer, Digital Twins,

1. Introduction

Today's cars hold considerable computational and sensing capabilities that are crucial for advanced traffic and driving automation, applications spanning from safety features to fully automated vehicles. However, data delivery and management protocols and interfaces that are required for machine learning pipelines are still primarily closed in company-specific silos [1]. The first efforts for creating open-source data transfer protocols and interfaces from the car to the cloud environment include Eclipse Kuksa [2], of which this work also bases, but considerable work is still required for data validation and benchmark efficiency of the proposed frameworks. Data sharing through open interfaces can boost innovation by more efficient and accurate machine learning models that cover more expansive geographic areas and use cases [3].

At the same time, digital twins can enable versatile testing and simulation capabilities as seen with applications in industry, energy, and transportation verticals [4]. Indeed, digital twins have attracted much research interest in recent years [5, 6]. The concept of the digital twin is broad, and definitions may vary. Still, the main idea is to model physical systems with digital means and update these digital models dynamically based on measurement data. In essence, digital twin methods provide digital spaces where reality can be modelled virtually [7] as it is or would be in unseen but possible situations. Indeed, the creation of the digital twin as part of automotive software development sets premises for testing and validating complex use cases, such as traffic accidents and extreme weather, that can rarely or only be tested in real-life situations with severe expenses. The utilisation of digital twins for automotive software development opens avenues for testing different sensors and components in actual use cases, such as studying the longevity of such components and proposing novel learning strategies that combine multiple data sources.

This paper describes the Vehicle-In-The-Loop (VIL) cloud interface. It verifies data consistency regardless of the source: a real car on the road or a virtual object in the digital twin environment. The overview of the Kuura platform is provided in Figure 1. We use KUKSA.val [8] that provides a vehicle abstraction layer to enable the management and use of vehicle signals. As a digital twin modelling framework, we use Unreal Engine 5. Capabilities of utilising such a game engine in digital twin creation have been successfully demonstrated in wind power plants [9] and cultural tourism [10]. Using a game engine and a VIL cloud interface enables visual simulation that complements the capabilities provided by KUKSA.val; KUKSA.val is used to collect data from a test drive in a real car. For validation, we determine how

TKTP 2024: Annual Doctoral Symposium of Computer Science, 10.-11.6.2024 Vaasa, Finland

olli.timonen@oulu.fi (O. Timonen); toni.bomstrom@proton.me (T. Bomström); Nicklas.Stafford@oulu.fi (N. Stafford); Alireza.BakhshiZadiMahmoodi@oulu.fi (A. B. Z. Mahmoodi); tero.paivarinta@oulu.fi (T. Päivärinta); ella.peltonen@oulu.fi (E. Peltonen)

0009-0006-4132-9496 (O. Timonen); 0009-0004-3192-7711 (A. B. Z. Mahmoodi); 0000-0002-3374-671X (E. Peltonen)

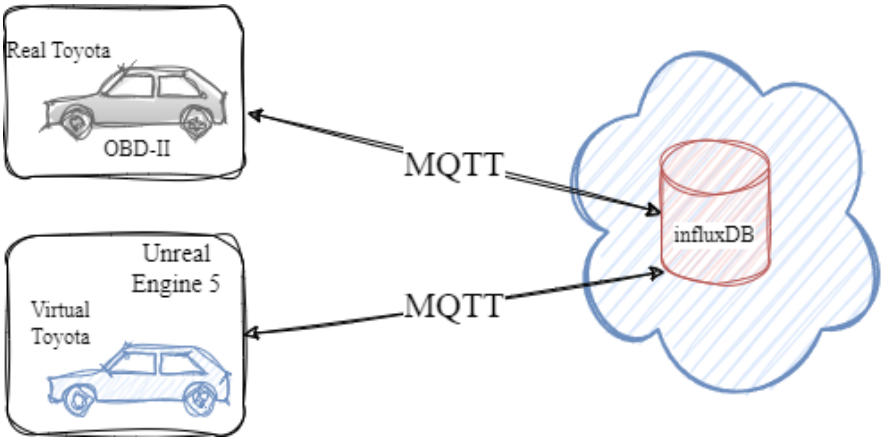

Figure 1: Overview of Kuuras overall architecture.

data collected from the simulation corresponds to real test drive data in a real-life driving scenario. Utilising the MQTT protocol, data is stored on a cloud server and further fed into Unreal Engine 5, where the test drive is replayed, and its correspondence to the real drive is ensured. This work offers a new perspective on verifying data consistency between simulated and real test drives and complements the vehicle abstraction opportunities provided by Eclipse KUKSA.

The main contributions of this work are the following: 1) We provide the Kuura platform for similarly collecting vehicular sensor data to a real car and similar test runs in a digital twin environment. With this, we extend the KUKSA.val environment to better fit digital testing and validation tasks. 2) We explore Unreal Engine 5 as a vehicular digital twin environment and provide a pipeline to deploy such digital twins with simulated and real test drives. 3) We experiment with the data consistency between simulated and real test drives and further demonstrate the power of game engine-based digital twins in vehicular computing and sensing scenarios.

2. Background

2.1. Vehicle as a Sensing Device

Modern cars implement technologies for automatic braking, Cooperative Adaptive Cruise Control (CACC), prevention of unwanted lane crossing, distance keeping, and so on, to supply drivers' own cognition and prevent accidents. For further technological advancement, vehicles will require artificial intelligence (AI) and machine learning (ML) capabilities depending on effective data transfer and management systems. With increased networking and computing capabilities, vehicles and their supporting edge and cloud back-ends can perform challenging inference and learning tasks to support drivers' cognition and automate the driving scenario [11, 12]. Such intelligent systems demand training data, which in-vehicle sensors and external databases can provide. How to make the data available, processed, and utilised in a challenging real-time and mobile environment is a timely research question.

Vehicular safety systems (and any other relevant applications) of any level of driving autonomy require data from in-vehicle sensors [13], such as cameras, LiDARs, radars, and speed meters [14, 15]. This information can be used to, for example, improve lane [16] and road pothole [17] recognition. Solutions for detecting drivers' behaviour while using smartphones during driving [18] and drunk driving [19] have been explored. However, the results underline that human drivers' perception and reasoning still maintain an advantage compared to fully automatic vehicles [20].

However, most of the in-vehicular and driver's personal sensors and interfaces are brand-specific or closed, limiting access to the data, computing, and networking capabilities and thus hindering vehicular application development. To enable connected vehicles to utilise all the available data sources, AI/ML computing resources, and networking capabilities, open-sourced general interfaces and software platforms need to be defined [1]. On-board diagnostics (OBD) protocol refers to a vehicle's self-diagnostic and reporting capability. The more advanced OBD-II is a protocol homogenised into the vehicle itself, allowing software-defined onboard operations and, most importantly, collecting a wide range of vehicular data to the software-defined vehicle's case. This includes but is not limited to engine load, coolant temperature, fuel pressure, engine revolutions per minute (RPM), vehicle speed, intake air temperature, airflow rate,

throttle position and many types of sensor data like oxygen sensors and fuel system status. OBD-II has relatively easy access to the mentioned sensors, which is enough to prove the concept. In the future, research conducted with vehicular sensors can utilise direct access to the vehicle's controller area network (i.e. CAN bus) and standardised architectures such as AUTOSAR for wider data access.

2.2. Automotive Simulations

Driving and traffic simulators are used in the automotive industry as an alternative to costly and potentially dangerous real-life testing [21]. The advantages of such practices highlight effectiveness in analysing human driving behaviour and essential traffic situations often too hazardous to test in real-life scenarios, such as extreme weather, congestion, and accidents [22]. This can be especially emphasised in the increasing reliance on simulation technologies for assessing human driving factors. The more real-life, photo-realistic simulations enable simultaneous testing of vehicle dynamics and stochastic pedestrian, driver, and vehicle interactions in various scenarios [23].

However, traditional simulators often have limitations in emulating real-life behaviour and perception. This has led to a growing interest in game engines as simulation platforms for developing and testing autonomous vehicle control systems [24]. Several vehicular simulators, such as CARLA, AirSim and CarSIM, provide simulation capabilities and environments to support vehicle research and development. These platforms have been used to study vehicle autonomy, safety and performance. CARLA is a free open-source simulator to support autonomous vehicle systems' development, training, and validation. AirSim is a simulator for drones and cars developed by Microsoft. It can also provide the possibility to experiment with deep learning, computer vision and reinforcement learning algorithms in autonomous vehicles and the creation of complex and changeable environments and additional sensor modalities [25]. CarSim is a vehicle dynamics simulation platform that allows the simulation of vehicle behaviour in different conditions and environments, including motor dynamics, through Simulink models. It can be used to create accurate models of vehicles and simulate their behaviour under different road surfaces, weather conditions, and traffic situations [26], but is not open-sourced.

In this study, we use Unreal Engine, renowned for its versatility, high-quality graphics and realistic physics simulation, which is useful for simulating vehicles [21]. Competing game engines include Unity and CryEngine, of which CryEngine is the smaller project. The main arguments that favour Unreal Engine are it is free of cost for research and commercial projects until making one million revenue, has open source code even it is precisely under source made available license, making it suitable for various applications in autonomous and driving-support test cases [22]. The key differences between Unity and Unreal Engine are summarised in Table 1. Unreal Engine has typically been considered a better choice for 3D games, while Unity has been considered a strong choice for 2D games.

The previous literature emphasises the importance of determinism in simulation environments to ensure repeatability, allowing for trustworthy and easily debuggable results. Game engines still may come with challenges of non-deterministic behaviours. For example, the investigation by Chance et al. [24] reveals significant simulation variance in CARLA, particularly due to actor collisions and system-level resource utilisation. As such, accuracy investigation is one of the key goals in our preliminary work presented in this paper.

2.3. Automotive Digital Twins

Digital twins (DT) in the context of vehicles is an emerging field that has attracted significant attention in both industry and academia [27]. Digital twins are virtual representations of physical entities, such as vehicles, that aim to mirror the lives and behaviours of their real-world counterparts [28]. These digital replicas use the best available physical models, sensor updates, and other data sources to simulate and predict the behaviour of the corresponding physical twin [29, 30]. One area where digital twins have shown great potential is in the automotive industry, particularly for electric vehicles [31]. Digital twins can greatly benefit electric vehicles, which have gained greater market share in recent years. By creating a digital twin of an electric vehicle, manufacturers and researchers can simulate and optimise its performance, energy consumption and other key parameters. Unlike traditional simulators, digital twins provide beyond capabilities for human-machine interaction and performing data-driven actions in real-world scenarios.

Digital twins also play a key role in the design and development of autonomous vehicles [32]. The concept of digital twins is closely related to the transition to data-driven vehicles, as it enables the analysis and validation of autonomous vehicle designs [33]. By exploiting digital twin technologies, researchers can assess the safety and security of autonomous vehicles and identify potential risks and vulnerabilities. Furthermore, combining digital twins with combined vehicle technology and cloud computing has led to the development of the Mobility Digital Twin (MDT) framework [34]. These frameworks consist of digital representations of people, vehicles, and transport, which enable the analysis and optimisation of mobility and large-scale traffic systems. By exploiting real-time data and simulations, MDT frameworks can support decision-making processes and improve the

Feature	Unreal Engine	Unity
Developer	Epic Games	Unity Technologies
Programming Languages	C++, Blueprint	C#
Source Code	Open source	Not open source
Pricing	Free for research and for commercial use up to 1 million revenue, 5% comission after that	Free version available
Learning Curve	Steep	Easy to learn with intuitive user interface
Graphics	Photorealistic graphics, used in AAA games	High-quality graphics, but not as refined as Unreal
Physics and Simulation	Ragdoll physics, physics-based destruction, fluid simulation	Easily integrated and well-rounded with other engine features
2D vs. 3D	Excellent 3D-development, especially for creating photorealistic environments and visual effects	Strong 2D development capability, excellent choice for 2D game projects

Table 1
Comparison of Unreal Engine and Unity

efficiency and safety of transportation systems.

Digital twins enable the simulation, optimisation, and analysis of vehicle performance, energy consumption, and safety and security. Combining digital twins with connected vehicle technology and cloud computing will extend their capabilities to optimise mobility systems. As technology advances, digital twins can be expected to play a key role in shaping the future of vehicles and transport systems. As such, current technologies aim to create models for distributed multi-agent cyber-physical systems using co-simulation [35]. Such large-scale digital twins should be able to make predictions about the future condition and behaviour of the vehicle [36]. However, AI-based digital twin capabilities require data cooperation and load-balancing, scheduling, and network security schemes over vehicle-to-cloud computing continuum [37].

2.4. Open-sourced Automotive Software

Open source software refers to software that has a publicly available and editable source code. This allows collaborative development and innovation. One of the most remarkable benefits of open-source software is its flexibility and customizability, as user communities can adapt the software to their specific needs. Open source is also cost-effective as it is free and reduces dependencies on specific software providers. The use of open source also offers opportunities for innovation in automotive software development and promotes the use of new technologies and solutions [38, 39].

For the automotive industry, open-source software presents some unique challenges as vehicular software, by default, has life-critical safety and reliability requirements [40]. Technically, anyone can modify the source code, which may create unwanted surprises and vulnerabilities. The maintenance and support of open-source software can be uncertain if their developers and community are not active or committed. While open-source software is dynamic and constantly changing, vehicles purchased today will remain in traffic for decades. In addition, there is a need for precise quality control and software certification in the automotive industry, which can be challenging to implement in an open-source environment because access to representative designs and industry-standard methodologies is limited. This limitation challenges researchers as automotive companies do not openly share their development life-cycles and verification methods, each maintaining proprietary techniques. Given this scenario, there is a growing demand for open-source solutions to support the development and research of automotive applications, emphasizing the need for open-source benchmarks to facilitate research across various aspects of automotive application development. [41].

3. Kuura Implementation

3.1. Design Principles

The cornerstone of our framework is grounded in the principle of open-source development, ensuring transparency and collaborative potential. Simplicity is at the core, paving the way for effortless future evolution. Our design philosophy revolves around creating a system that is not just functional today but remains adaptable and maintainable for tomorrow's innovations. The essence of this framework is to avoid complexity instead of embracing a minimalist approach that prioritises ease of understanding and operation. One must consider the life cycle of software components, as updates and dependencies are inevitable. The framework architecture

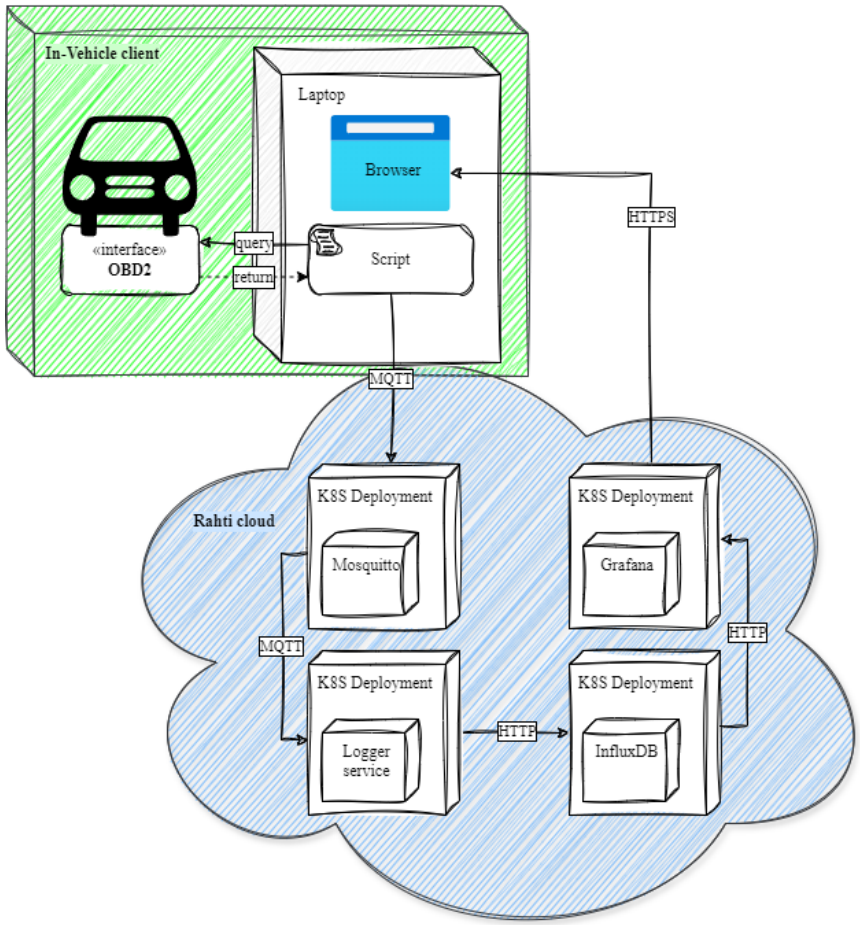

Figure 2: Deployment diagram of the Kuura vehicular data collection system.

is designed to handle these, avoiding obsolescence and incompatibility.

3.2. Kuura Architecture Design

The general architecture of Kuura is shown in Figure 2. We chose the Unreal Engine 5 game engine because of its versatility in creating realistic simulations. This is an essential part of the research objective to verify the consistency of the data between the simulation and the real test runs. The MQTT protocol was chosen to collect and transfer the data to the cloud server and the game engine, as it is reliable and efficient for real-time data transfer, which might be the next step in the research and, thus, critical requirements in our study. The MQTT protocol operates asynchronously and is considered an ideal choice for IoT applications that often operate on limited devices or low-bandwidth networks.

The Kuura framework presents a cohesive suite of components, each selected for robustness and simplicity. At its foundation lies the integration of Kuksa, SMAD, and Kuura, delineating a timeline of iterative progress as detailed in Table 2. Each iteration is a response to the evolving needs and challenges encountered. Kuksa, initially misaligned with its focus on automotive app stores and firmware updates, has since been archived [8, 2]. SMAD was unsustainable due to its complexity and poor documentation [42]. Our simplified stack emerges as a response, stripping away the superfluous to focus on functionality. It leverages OpenShift (run on CSC Rahti container cloud [1]) for its cost-efficiency compared to Microsoft Azure. This pragmatic approach is engineered to reduce complexity, cost, and maintenance overhead,

[1] https://rahti.csc.fi/

Purpose	Eclipse Kuksa Cloud	SMAD stack	Kuura (this paper)
Cloud Service Provider	Microsoft Azure	Microsoft Azure	OpenShift
Deployment Platform	Kubernetes	Kubernetes	OKD
Client-Server Messaging Infrastructure Broker	Eclipse Hono	Eclipse Hono	Eclipse Mosquitto
Serverside Messaging Infrastructure	-	Ambassador and Kafka with Zookeeper	Python script
Client Message Persistence	InfluxDB	MongoDB	InfluxDB
Client Message Data Modelling	Kuksa.VAL	Kuksa.VAL	Client implementation
Client Firmware Updates	Eclipse hawkBit	-	-
Client Appstore	Kuksa Appstore	-	-
Messaging Telemetry Storage	-	MongoDB	-
Data Visualization	-	Node-RED	Grafana
Deployment Monitoring	-	Prometheus Monitoring, InfluxDB, and Grafana	-
Message Tracing	-	Jaeger Trace	-

Table 2
Eclipse Kuksa Cloud, SMAD stack, and Kuura software components.

streamlining operations without compromising capability.

Each framework iteration — Kuksa, SMAD, and Kuura — brings new insights. Kuksa's archival signals a pivot away from its original automotive-centric focus. SMAD's downfall was its complexity and reliance on now-inaccessible Kubernetes Helm charts. Kuura emerges as the distilled essence of its predecessors, embodying simplicity and sustainability. By eliminating non-essential components, Kuura adapts existing functionalities with more straightforward tools, significantly reducing cost and complexity and enabling an environment conducive to continuous development and operation.

3.3. Vehicle Data Reader

The OBD-II is a port designed for diagnostic purposes. It has multiple buses available. These buses include the CAN bus, SAE-1850 and ISO-9141-2. The automotive manufacturers can also provide other networks at their discretion [43]. The bus we are most interested in is the CAN bus. On some vehicles, the CAN bus available at the OBD connector can be protected by a gateway device restricting access to some data from the OBD port. Unlike the CAN bus inside the car, you must poll the OBD port to receive any data. While we could get most of the data we wanted from the OBD port, some data, like the steering wheel position, was unavailable. This makes the OBD port unsuitable as a data source for our purposes, as it would make it quite difficult to drive the virtual car in Unreal accurately.

In the evaluation phase, we collected data from an OBD-II Bluetooth adapter connected to a Toyota RAV4 car. A laptop computer running Linux was connected to the adapter, and a script was run to record data from the vehicle in a log file. The successful log file collection was further important in developing the auto-client script for future larger tests and ensuring the whole system's functionality. Practical testing in the first phase was carried out by driving the car and ensuring the data was stored correctly and its format was manageable.

3.4. Data Transmission

MQTT makes it trivial to multi-cast the collected data if we want to enable multiple clients to listen to the generated data simultaneously. One example of such a scenario is live visualisation of the data while saving it to a database without additional latency. While we could also save the data and then fetch it from the database, this would add latency to the visualisation. MQTT also has built-in, easy-to-configure security mechanisms. Setting up MQTT with SSL is very easy, and configuring the MQTT broker to require client certificates for communication is also very easy. The connection can also be set up to require a username and password.

We could also use HTTP or raw TCP/UDP sockets as an alternative for MQTT. While HTTP offers security measures similar to MQTT, it does not have multi-cast by default. While it is not hard to implement, MQTT has it built in and is most likely already done correctly. One advantage HTTP has over MQTT is the ability to communicate directly between two applications, eliminating the need for a broker in cases where there is only one client.

Raw sockets are the most basic option, and they don't come with any of the advanced features included in

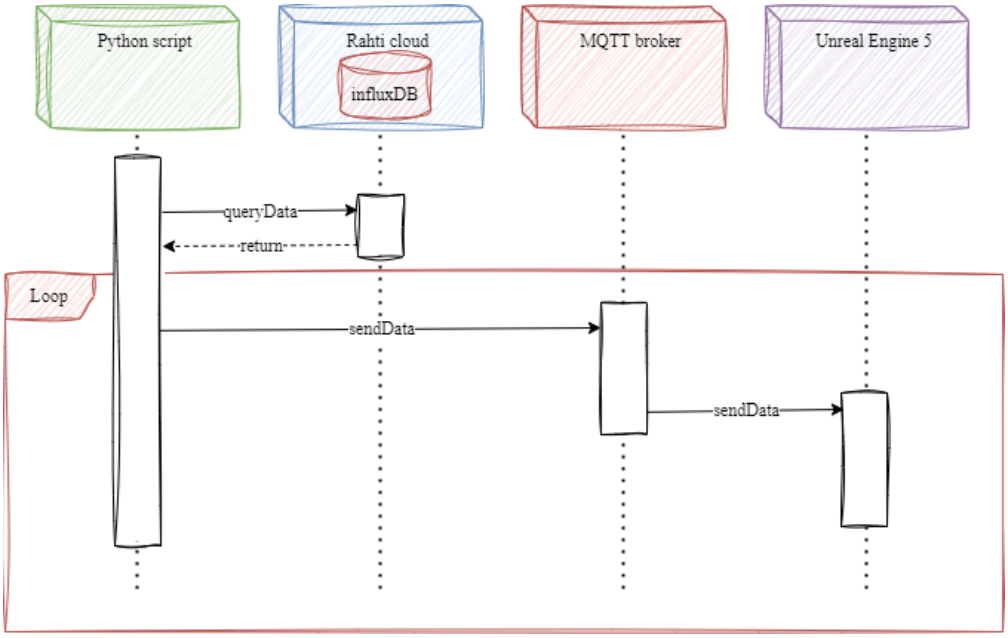

Figure 3: Sequence diagram of the vehicular data transferred to Unreal Engine 5.

MQTT out of the box. However, they are very versatile and can be used for various purposes. One advantage the sockets would offer is the ability to write raw can data as is to the socket. This would enable saving raw can dumps in a database with minimal overhead if we ever needed/wanted to support it. One problem with multi-cast solutions is that the provider has no idea if any clients are listening for the sent data unless the clients have been programmed to provide feedback when they are listening. This makes it harder to implement the provider in a way that it holds the messages in memory or saves them locally in case the data is sent to nowhere.

In the experiments, a laptop was used as the in-vehicle client running Ubuntu 22.04 LTS, and the script collecting the data was written in Python using an OBD library [44]. The script writes read values into a CSV file locally and publishes them using the MQTT protocol. The backend was deployed on CSC's Rahti as RedHat OpenShift deployments. On the server side, Mosquitto MQTT broker forwards the published messages to subscribers. The most important subscriber is a Python service that stores received messages in an InfluxDB instance. As an additional demonstration, Grafana was deployed to provide a real-time dashboard for the published and stored data. The sequence diagram is provided in Figure 3.

3.5. Cloud Environment

The cloud environment receives data from the MQTT broker. The environment also has a Python script that connects to the broker to receive the data from the vehicle. The script stores all of the messages received by InfluxDB. The point name and field are derived from the MQTT topic. The timestamp is also gotten from the MQTT message payload. Since the timestamp is included in the message, we could use any database solution to store the data. If the timestamp were missing, however, then a time series database would be our only option since the message times are crucial for playback at a later time. By getting the message time from the provider, we can ensure that network conditions do not affect the accuracy of the recorded timestamps.

InfluxDB, a time series database used to store large amounts of time-stamped data due to its high performance and scalability, was stored at the onset of the process. Storage is essential in handling large amounts of data that emanate from driving vehicles. A Python script was then used in the next stage of the data-processing workflow. This script had two main functionalities: First, it reads GPS point data pre-recorded into a JSON file, which is vital in mapping out routes of vehicles. Secondly, this script establishes a connection with InfluxDB to retrieve useful information within a particular range. This recovery is critical for evaluating the vehicle's performance and environmental conditions during various

experiment stages.

At this point, the processed data goes through an MQTT broker using a Python script. Once more, this protocol provides lightweight messaging, providing fast and reliable real-time information transmission that would be needed for the simulation environment.

3.6. Simulation Environment

Multiple reasons contributed to the choice of Unreal Engine 5 game engine, including the capacity to create realistic simulations of real car driving and the possibility of driving a car in a simulation, thereby generating corresponding data. The research aimed to ensure uniformity between the simulation and actual driving, thus requiring realistic simulations. Unreal Engine 5 is also open-source, which meets one of the implementation principles of the study, making future development as easy as possible.

The research utilised the MQTT protocol, one of the key IoT connections and data collection components. Unreal Engine does not have native MQTT support. For this reason, we used the NinevaStudios MQTT-utilities extension with some modifications. This extension allowed MQTT data communication, which is essential for collecting data from the simulation, with minor adjustments made to transfer it to cloud storage securely. Through this connection, it was possible to develop a dynamic and interactive simulation environment.

Lastly, we simulate a car running along received GPS points as shown in Figure 4. In the simulation, the vehicle's movement was driven by speed data acquired from the MQTT broker. As a result, real-time synchronisation between the GPS points and speed data gave an actual representation of the journey made by the vehicle, hence allowing for the immersion of details about its performance in different circumstances. Such a holistic approach to data storage, processing, transmission, and visualisations shows how diverse technologies can be integrated into high-level vehicular data analysis and simulation.

The initial version of the Kuura presented in this paper has a dynamic road generated as the car moves around, thus simplifying testing by making it independent of environmental conditions. This method enables better flexibility in the testing process because it does not require a predefined route or special environmental circumstances. Generating dynamic roads is essential to ensure the reliability of the data collection system. This phase, built on the multiple approaches used in the study, emphasises adaptability and precision. By generating the road during runs accuracy of collected data could be instantly evaluated. It is particularly advantageous to work within this dynamic environment for the purposes of identifying and solving prospective issues within a workflow for data processing that would make it strong and efficient. This technique also makes Unreal Engine simulation more elaborate. It allows different scenarios to be run on a platform without sticking to a single static map, giving the evaluation process more flexibility.

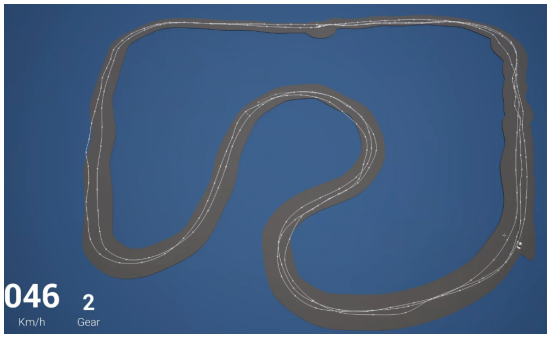

Figure 4: Simulated route in the virtual environment, based on the real data points collected during the experiments.

4. Experimentation

4.1. Real-life Experiment

The real-world tests were conducted in the OuluZone vehicle testing area using a Toyota RAV4 Hybrid 2019 vehicle. A closed area, such as OuluZone, was chosen because it allows for assessing the drives and their safety. The significance of this place is that it helped gather and analyse information in real-life scenarios, thus allowing comparison and verification with data collected from virtual and actual driving instances. Besides being a recreational driving and sports centre, OuluZone is also a notable site for research and learning, especially on autonomous cars and related technologies.

Several laps were driven during the tests, some with the cruise control set at different speeds (30km/h, 40km/h, and 50km/h) to facilitate the validation of results in the simulation with data collected at a constant speed. Laps were also driven without cruise control at varying speeds. Driving data was collected during the test via an OBD-II Bluetooth adapter connected to a laptop running Linux. This allowed for the vehicle data to be logged and its format managed. Towards the end of the tests, a USB adapter enhanced data collection.

4.2. Virtual Experiment

Our virtual experiment utilised Unreal Engine 5.3.2 to drive test drive scenarios comparable to our real-world data collection efforts. In this experiment, we used the same logger used during actual test drives with a real car, ensuring a uniform approach to data acquisition and proving that the logger could be used without changes in both

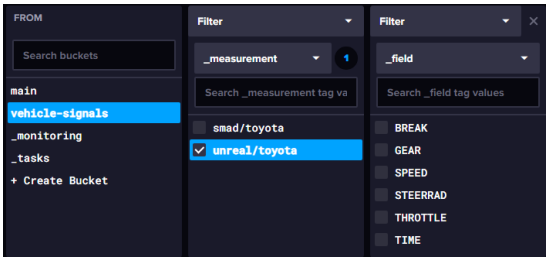

Figure 5: Screenshot of influxDB which contains both real-world data (smad/toyota) and virtually collected data (unreal/toyota).

Figure 6: A picture of the car driving in the virtual OuluZone 3D environment using the data collected in the real OuluZone.

environments. We gathered data on speed and time from the virtual test drive, which can be cross-verified with the real car's outputs. The current limitation of real-world data collection stems from the OBD-II interface's inability to provide comprehensive vehicle diagnostics. In the virtual setting, we collected additional data such as gear, throttle, brake application, and steering angle. These were predominantly included for illustrative purposes, aiming to demonstrate the extensive data collection possibilities within a simulated environment. It is important to note that verifying these additional parameters will become feasible with future access to the CAN bus, allowing for a more detailed and accurate comparison between virtual and real vehicle data.

4.3. Experimentation Results

In our validation process, we specifically focused on comparing the collected GPS data and speed data between the actual and virtual driving tests conducted in Unreal Engine 5. As shown in Figure 5, the same InfluxDB database successfully contains both real-world data (smad/toyota) and virtually collected data (unreal/toyota). This design will further allow simultaneous analysis of both virtual and real-world data sets, allowing us to expand the digital twin creation capabilities with virtual realities and actual real-life test runs, independently of the data source.

As illustrated in Figure 6, we successfully mapped the collected GPS data onto the 3D model of the racetrack in runtime from cloud and verified its accuracy. This demonstrates that our virtual environment can accurately replicate real driving conditions. The speed data collected in the database corresponded with the data obtained in the Unreal Engine 5 simulation, confirming the consistency of data in both real and virtual driving scenarios. While the data transmitted from the game engine to the server was also accurate, at this stage, our primary focus was on verifying the accuracy of speed and time information. Expanding this experimentation to cover a wider range of variables is possible in future research.

5. Discussion and Conclusions

In this study, we have aimed to bring new insights into vehicular data collection and the creation of digital twins by using the Eclipse Kuksa platform and Unreal Engine 5 to simulate driving scenarios. Our main focus was providing an overview of the simplified vehicular data collection architecture that can be easily developed for further projects and verifying the consistency between real and simulated vehicular data through practical real-world experimentation.

Using the MQTT protocol for sending data and Unreal Engine 5 for simulation has allowed us to compare real driving data with simulated ones. This method makes digital twins more reliable and allows later use for testing in many conditions that are hard or expensive to create in real life, like very bad weather or different kinds of traffic situations.

We encountered challenges in data collection via the OBD-II protocol because it is filtered and does not allow the collection of all possible data. This limitation highlighted the need for more comprehensive data acquisition methods like the CAN bus. The data collection limitations prompted us to consider future enhancements in our methodology to achieve a more accurate and encompassing digital representation of the vehicle.

Our findings open up possibilities for future research directions, including optimising data transmission methods for improved efficiency and exploring bi-directional data flow between the digital twin and the vehicle. Such advancements could potentially enable real-time vehicle control based on digital twin data.

By integrating additional simulation models and considering more sophisticated data collection interfaces, we anticipate that future iterations of this work will address the current limitations and unlock new capabilities for digital twins in automotive research and development. The potential for these technologies to improve vehicle safety, efficiency, and innovation is immense, paving the

way for a more interconnected and intelligent transportation ecosystem.

Future efforts should be made using the CAN bus instead of the OBD-II to improve accuracy completeness and to have access to all possible data the vehicle provides. Reconsidering data transmission methods, like MQTT, for more efficient data multicasting is also a possible future direction. In the future, we are also looking into sending data from the game engine to the car instead of just storing it in the cloud, having the car drive in real life and the game engine simultaneously with as little latency as possible and importing Eclipse Arrowhead to extend possibilities with simulation models, such as using the architecture with Matlab Simulink or corresponding open-sourced physics modelling software.

Acknowledgments

The work has been supported by the EU HORIZON project CHIPS-JU CIA FEDERATE (grant number 101139749), Business Finland project 6G Visible (grant number 10743/31/2022), and the Finnish Research Council project Northern Utility Vehicle Laboratory Consortium GO!-RI (grant number 352726).

References

[1] E. Peltonen, A. Sojan, T. Päivärinta, Towards real-time learning for edge-cloud continuum with vehicular computing, in: 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), IEEE, 2021, pp. 921–926.

[2] A. Banijamali, P. Kuvaja, M. Oivo, P. Jamshidi, Kuksa∗: Self-adaptive microservices in automotive systems, in: International Conference on Product-Focused Software Process Improvement, Springer, 2020, pp. 367–384.

[3] J. Nickerson, K. Lyttinen, J. L. King, Automated Vehicles: A Human/Machine Co-learning Perspective, Technical Report, SAE Technical Paper, 2022.

[4] F. Tao, B. Xiao, Q. Qi, J. Cheng, P. Ji, Digital twin modeling, Journal of Manufacturing Systems 64 (2022) 372–389.

[5] M. Liu, S. Fang, H. Dong, C. Xu, Review of digital twin about concepts, technologies, and industrial applications, Journal of Manufacturing Systems 58 (2021) 346–361. Digital Twin towards Smart Manufacturing and Industry 4.0.

[6] F. Tao, H. Zhang, A. Liu, A. Y. Nee, Digital twin in industry: State-of-the-art, IEEE Transactions on Industrial Informatics 15 (2018) 2405–2415.

[7] B. R. Barricelli, E. Casiraghi, D. Fogli, A survey on digital twin: Definitions, characteristics, applications, and design implications, IEEE Access 7 (2019) 167653–167671.

[8] A. Banijamali, P. Jamshidi, P. Kuvaja, M. Oivo, Kuksa: A cloud-native architecture for enabling continuous delivery in the automotive domain, in: International Conference on Product-Focused Software Process Improvement, Springer, 2019, pp. 455–472.

[9] J. V. Sørensen, Z. Ma, B. N. Jørgensen, Potentials of game engines for wind power digital twin development: an investigation of the unreal engine, Energy Informatics 5 (2022) 1–30.

[10] F. Sang, H. Wu, Z. Liu, S. Fang, Digital twin platform design for zhejiang rural cultural tourism based on unreal engine, in: 2022 International Conference on Culture-Oriented Science and Technology (CoST), IEEE, 2022, pp. 274–278.

[11] A. Alhilal, T. Braud, P. Hui, Distributed vehicular computing at the dawn of 5g: a survey, arXiv:2001.07077 (2020).

[12] Y. Khaled, M. Tsukada, J. Santa, T. Ernst, On the design of efficient vehicular applications, in: VTC Spring 2009-IEEE 69th Vehicular Technology Conference, IEEE, 2009, pp. 1–5.

[13] S. Baidya, Y. Ku, H. Zhao, J. Zhao, S. Dey, Vehicular and edge computing for emerging connected and autonomous vehicle applications, in: Proc. of the 57th Design Automation Conference (DAC), 2020.

[14] M. Munz, M. Mahlisch, K. Dietmayer, Generic centralized multi sensor data fusion based on probabilistic sensor and environment models for driver assistance systems, IEEE Intelligent Transportation Systems 2 (2010).

[15] F. Garcia, D. Martin, A. De La Escalera, J. M. Armingol, Sensor fusion methodology for vehicle detection, IEEE Int Transportation Systems 9 (2017).

[16] Q. Li, L. Chen, M. Li, S. L. Shaw, A. Nüchter, A sensor-fusion drivable-region and lane-detection system for autonomous vehicle navigation in challenging road scenarios, IEEE Transactions on Vehicular Technology 63 (2014) 540–555.

[17] A. Ghose, P. Biswas, C. Bhaumik, M. Sharma, A. Pal, A. Jha, Road condition monitoring and alert application, in: IEEE International Conference on Pervasive Computing and Communications Workshops, IEEE, Lugano, Switzerland, 2012, pp. 489–491.

[18] Y. Wang, J. Yang, H. Liu, Y. Chen, M. Gruteser, R. P. Martin, Sensing vehicle dynamics for determining driver phone use, in: Int. conf. on mobile systems, applications, and services, 2013, pp. 41–54.

[19] J. Ljungblad, B. Hök, A. Allalou, H. Pettersson, Passive in-vehicle driver breath alcohol detection using advanced sensor signal acquisition and fusion, Traffic injury prevention 18 (2017).

[20] B. Schoettle, Sensor fusion: A comparison of sensing capabilities of human drivers and highly automated vehicles, University of Michigan (2017).

[21] D. Michalík, M. Jirgl, J. Arm, P. Fiedler, Developing an unreal engine 4-based vehicle driving simulator applicable in driver behavior analysis—a technical perspective, Safety 7 (2021) 25.

[22] S. Malik, M. A. Khan, H. El-Sayed, Carla: Car learning to act — an inside out, Procedia Computer Science 198 (2022) 742–749. 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks / 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare.

[23] A. Dubs, V. C. Andrade, M. Ellis, S. Ganley, B. Karaman, O. Toker, A photo-realistic simulation and test platform for autonomous vehicles research (????).

[24] G. Chance, A. Ghobrial, K. McAreavey, S. Lemaignan, T. Pipe, K. Eder, On determinism of game engines used for simulation-based autonomous vehicle verification, IEEE Transactions on Intelligent Transportation Systems (2022).

[25] W. Jansen, E. Verreycken, A. Schenck, J.-E. Blanquart, C. Verhulst, N. Huebel, J. Steckel, Cosysairsim: A real-time simulation framework expanded for complex industrial applications, in: 2023 Annual Modeling and Simulation Conference (ANNSIM), IEEE, 2023, pp. 37–48.

[26] Q. Liu, D. Xie, S. Hu, J. Wu, Research on dynamic performance simulation of in-wheel motor electric vehicle based on carsim-simulink, in: Journal of Physics: Conference Series, volume 1820, IOP Publishing, 2021, p. 012109.

[27] A. Fuller, Z. Fan, C. Day, C. Barlow, Digital twin: Enabling technologies, challenges and open research, IEEE access 8 (2020) 108952–108971.

[28] J. A. Ross, K. Tam, D. J. Walker, K. D. Jones, Towards a digital twin of a complex maritime site for multi-objective optimization, in: 2022 14th International Conference on Cyber Conflict: Keep Moving! (CyCon), volume 700, IEEE, 2022, pp. 331–345.

[29] S. Maulik, D. Riordan, J. Walsh, Dynamic reduction-based virtual models for digital twins—a comparative study, Applied Sciences 12 (2022) 7154.

[30] A. M. Madni, C. C. Madni, S. D. Lucero, Leveraging digital twin technology in model-based systems engineering, Systems 7 (2019) 7.

[31] D. Piromalis, A. Kantaros, Digital twins in the automotive industry: The road toward physical-digital convergence, Applied System Innovation 5 (2022) 65.

[32] S. Almeaibed, S. Al-Rubaye, A. Tsourdos, N. P. Avdelidis, Digital twin analysis to promote safety and security in autonomous vehicles, IEEE Communications Standards Magazine 5 (2021) 40–46.

[33] T. Fuchs, M. Zinser, K. Renatus, B. Bäker, Data model of automotive digital twins, ATZelectronics worldwide 16 (2021) 52–57.

[34] Z. Wang, R. Gupta, K. Han, H. Wang, A. Ganlath, N. Ammar, P. Tiwari, Mobility digital twin: Concept, architecture, case study, and future challenges, IEEE Internet of Things Journal 9 (2022) 17452–17467.

[35] M. Palmieri, C. Quadri, A. Fagiolini, C. Bernardeschi, Co-simulated digital twin on the network edge: A vehicle platoon, Computer Communications 212 (2023) 35–47.

[36] G. Bhatti, H. Mohan, R. R. Singh, Towards the future of smart electric vehicles: Digital twin technology, Renewable and Sustainable Energy Reviews 141 (2021) 110801.

[37] D. Chen, Z. Lv, Artificial intelligence enabled digital twins for training autonomous cars, Internet of Things and Cyber-Physical Systems 2 (2022) 31–41.

[38] S. Kochanthara, Y. Dajsuren, L. Cleophas, M. van den Brand, Painting the landscape of automotive software in github, in: Proceedings of the 19th International Conference on Mining Software Repositories, 2022, pp. 215–226.

[39] S. Niæetin, R. Šandor, G. Stupar, N. Tesliæ, Maximizing the efficiency of automotive software development environment using open source technologies, in: 2018 IEEE 8th International Conference on Consumer Electronics-Berlin (ICCE-Berlin), IEEE, 2018, pp. 1–3.

[40] Y. Zhang, Y. Ning, C. Ma, L. Yu, Z. Guo, Empirical study for open source libraries in automotive software systems, IEEE Access (2023).

[41] F. A. da Silva, A. C. Bagbaba, A. Ruospo, R. Mariani, G. Kanawati, E. Sanchez, M. S. Reorda, M. Jenihhin, S. Hamdioui, C. Sauer, Special session: Autosoc-a suite of open-source automotive soc benchmarks, in: 2020 IEEE 38th VLSI Test Symposium (VTS), IEEE, 2020, pp. 1–9.

[42] H. Hirvonsalo, P. Seppänen, On deployment of eclipse kuksa as a framework for an intelligent moving test platform for research of autonomous vehicles, in: Proceedings of the 2nd Eclipse Research International Conference on Security, Artificial Intelligence, Architecture and Modelling for Next Generation Mobility, RWTH Aachen University, 2021.

[43] K. McCord, Automotive Diagnostic Systems: Understanding OBD I and OBD II, CarTech Inc, 2011.

[44] Obd library for python 3, https://github.com/brendan-w/python-OBD, 2023. Accessed: 2023-11-12.