Enabling Time Synchronization with Hardware-in-the-Loop Integration on a Data-Driven Middleware for Autonomous Vehicles Simulations

Ilton Pfleger Junior, Leonardo Passig Horstmann, Antônio A. Fröhlich
Software/Hardware Integration Lab, Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil
{pfleger*, horstmann, guto}@lisha.ufsc.br

***Abstract*—Building reliable simulation scenarios and digital twins is a keystone for the development process of critical systems such as autonomous vehicles. In this sense, the ability to integrate simulators with software-, hardware-, and even vehicles-in-the-loop is fundamental for increasing the reliability of the simulations. Nevertheless, guaranteeing time synchronization amongst the different components of the system is a challenging task that must consider the different capabilities regarding timing in each device. In this work, we build upon a data-driven middleware to provide transparent hardware-in-the-loop integration to an autonomous vehicle simulation scenario and extend the original implementation to account for time synchronization by taking advantage of a high-precision time source provided by the hardware components. We operate the middleware in Linux over a NVIDIA Jetson Orin AGX, measuring an average deviation of 6ms when compared to the high-precision hardware-in-the-loop timestamps, with the deviation exceeding the established safe margin over 69% of the observations. The attained results corroborate the need for time synchronization on digital twins, especially when considering the strict timing requirements of critical systems such as autonomous vehicles.**

***Index Terms*—digital-twins, hardware-in-the-loop, time synchronization**

I. Introduction

Simulation is a critical step in the design, implementation, and deployment of critical systems, such as Autonomous Vehicles (AV). Integrating simulators with software-, hardware-, and even vehicles-in-the-loop is fundamental for increasing the robustness in the verification of AV solutions [6]. Moreover, many rely on simulation tools such as CARLA [3] and Gazebo [8] to build digital twins solutions for validating security and safety aspects for AI-based decision-making on autonomous systems.

In this context, Hoffmann et al. [2], [5] introduced a data-driven middleware to integrate Autonomous Systems simulators and external tools. The middleware creates an intermediary layer between the simulator and external tools by modeling the inputs and outputs as SmartData [4], which allows for accounting for timeliness, security, and communication. Their solution aims at easing the integration of the autonomous system simulation with other tools as it abstracts data sources as SmartData. Thus, replacing or introducing new data sources is transparent to the application. Moreover, the SmartData abstraction allows for the middleware to promote time synchronization as timing is intrinsic to SmartData.

Time synchronization between Hardware-in-the-Loop (HIL) components must consider devices have different capabilities regarding timing. Some devices have no time source and thus are not able to produce valid timestamps for each data sample. Other devices can account for timing but must be synchronized based on an external time source. At last, we have devices that produce their own, reliable, time source as the Global Navigation Satellite System (GNSS). Synchronizing timing among simulation tools, HIL, and Software-in-the-Loop (SIL) solutions is challenging when considering different notions of timing on synchronous and asynchronous executions, different representations of time, and different levels of precision.

In this work, we build upon the middleware presented by Hoffmann et al. to provide transparent HIL integration to AV simulation tools while addressing the problem of time synchronization amongst simulation, SIL, and HIL components. We take advantage of the high-precision time source of a HIL GNSS device and the abstractions of the SmartData middleware to synchronize time for the data being produced. Once data introduced into the SmartData network is synchronized in time, external tools in data transformation steps can operate time according to user-level configuration, which determines what processes, HIL, and SIL tools must account for time and what not. By introducing time synchronization, we extend the proposed middleware and provide the user with a solution for building more reliable digital twins for the development, test, and validation of Autonomous Vehicles.

The remainder of this paper is organized as follows: Section II presents basic concepts on SmartData. Section III describes the SmartData middleware proposed by Hoffmann et al. [2], [5]. Section IV describes the HIL components integrated and the time synchronization mechanism. Section V presents a case study with results corroborating the need for time synchronization mechanisms on a digital twin system. Section VI ends this work with some final remarks.

II. SmartData

SmartData was conceived to be the primary (if not the only) abstraction used by application programmers to interact with

This work was partially supported by FUNDEP grants Rota 2030/Linha VI 29271.02.01/2022.01-00 and 29271.03.01/2023.04-00.

the physical world on a network of sensors and actuators. A SmartData is a piece of data enriched with enough metadata to make it self-contained regarding semantics, spatial location, and timing [4]. The SmartData Interface is depicted by Figure 1.

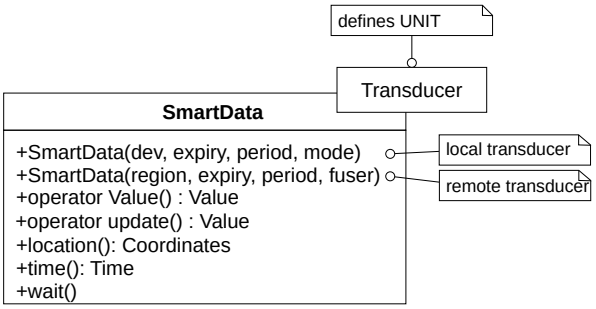

Fig. 1. The SmartData Interface [4].

A Transducer is a single interface to interact with both Sensor and Actuator in SmartData. As depicted in Figure 1, a SmartData is associated with a Transducer in the general case. A Transducer comprises the definition of the `UNIT` that the SmartData will consider. The data semantics is encapsulated as a `UNIT`, a 32-bit type identifier designating either an *SI Physical Quantity* or plain digital data, inspired by the Transducer Electronic Data Sheets in the IEEE 1451 standard [7]. From this attribute, we can extract the semantics of the data, like the size and a range of valid values.

From the sensor perspective, Transducers can include information regarding sensing error, and sensing time and value from the most up-to-date sample (accessed through `operator Value()` and `time()`).

Finally, the relationships amongst SmartData are modeled in the form of Interests. An Interest encompasses definitions like timing, security, criticality, and sampling modes. In this sense, a SmartData A can be interested in a SmartData B considering a period, expiry, and sampling mode. Thus, B must adapt itself to attend to A requirements.

III. The Middleware

The SmartData-based data-driven Middleware for integrating Autonomous Vehicles Simulation tools was introduced by Hoffmann et al. [2], [5]. Their solution assumes that Simulators provide at least four interfaces, one for time synchronization (read and write of internal simulation time), one for describing the simulated scenario, one for reading sensor data, and one for writing actuators data. The middleware models the data used on a simulator and creates an intermediary layer between the simulator and the external tools by defining their respective inputs and outputs as SmartData. A message bus is used for communication between SmartData following their Interest relations. Messages are exchanged following a specific protocol. Nevertheless, the architecture presented is agnostic of protocol.

Figure 2 presents a component diagram of the proposed architecture. For a simulator to be connected to the SmartData middleware, the domain of the application must be decomposed into SmartData. The Simulator then connects to the internal SmartData Network through a SmartData_Handler implementing a Simulator_Wrapper able to access the control interfaces and feed the SmartData Sensors. Any SmartData connected to the bus can read this data. Transformational SmartData can both read and write into the message bus. Finally, Actuator SmartData can read from the message bus and interact with the `SmartData_Handler` accordingly, closing the control loop. Note that only the Sensor and Actuators are connected to a Simulator. Thus, the solution introduced by Hoffmann et al. allows one to create and customize Transformer SmartData to promote additional functionalities, interacting with the simulation by reading the Sensor SmartData of Interest and sending the resulting data to the Actuator SmartData, which will be sent back to the Simulator.

IV. Hardware-in-the-Loop Integration and Time Synchronization

In this section, we introduce each of the HIL components integrated into the SmartData Framework, namely the OAK-D Long Range (LR) Camera from Luxonis, a Velodyne PUCK VLP-16 LiDAR (Light Detection and Ranging), and a LPMS IG1P-CAN GNSS and IMU (Inertial Measurement Unit) sensor. After presenting the HIL components, we finish this section by describing the process for time synchronization.

A. Camera

Cameras play a vital role in the functionality of autonomous vehicles, serving as the primary means for environmental perception, decision-making, and recognition models. In this case, the Luxonis OAK-D LR, a precise stereo depth sensor equipped with a 4-TOPS processor capable of running any AI model independent of host, has been integrated. The captured data are converted into an Image SmartData with a Digital UNIT. This camera model features an internal oscillator with μs precision.

B. LiDAR

LiDARs are sensors that provide a perception of the environment based on the reflection of hundreds of laser beans. They are often paired with cameras to enhance reliability and safety in conditions where a regular camera may be compromised, such as fog or rain. The chosen model for this work is the Velodyne PUCK VLP-16. This hardware is equipped with 16 channels (16 vertical lasers) that cover a ±15° vertical field of view (FOV) and rotate 360° horizontally at a maximum speed of 1300 RPM, providing up to 300,000 PPS. For this work, the grabbed data by the LiDAR is represented by a Point Cloud (PCD) SmartData, which represents an array of points with (x, y, z, i) information where x, y, and z are coordinates of the reflection and i its intensity. This LiDAR is equipped with an internal oscillator that assigns a timestamp to each PCD packet. This clock can be synchronized with an external GNSS receiver or with the host by a serial port.

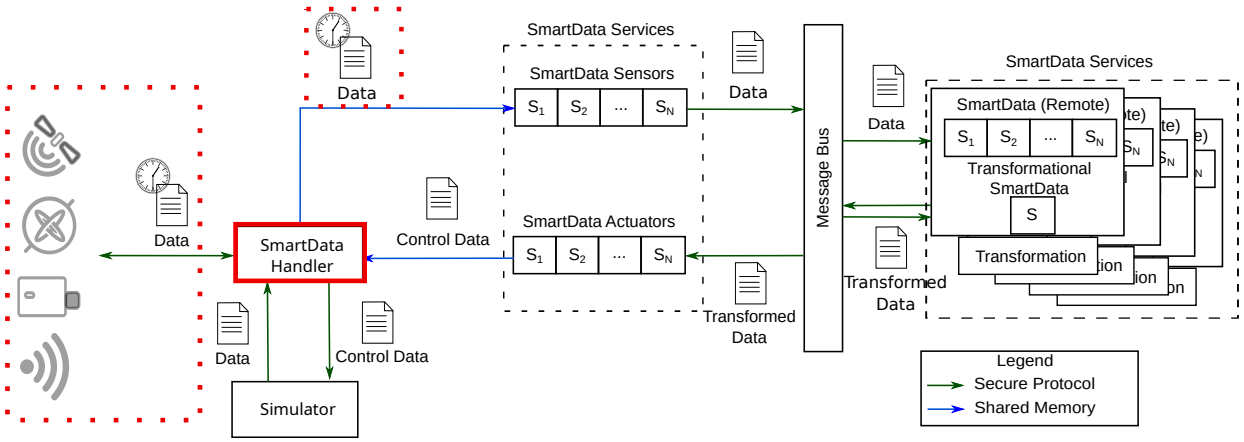

Fig. 2. Overview of the integration of the SmartData and Simulators.

C. IMU and GNSS

IMUs provide a wide range of basic physical quantities for a particle. For this project, we only used a 3-axis acceleration, gyroscope, and orientation heading (magnetometer) from all the collected data. Furthermore, GNSSs are also crucial to this integration because they provide geo-referencing information about the vehicle, which includes longitude, latitude, altitude, velocity and precise timestamps. These timestamps came with nanosecond precision, which we will use to synchronize all other sensors. To collect all this data we use an LPMS IG1P-CAN sensor and encapsulate it into 11 (eleven) SmartData one for each reading unit.

D. Integration to the SmartData Middleware and Time Synchronization

As mentioned in Section III, the SmartData middleware expects a handler to be defined implementing four interfaces, one for time synchronization, one for reading and replace readings of sensors, one for the description of simulation scenarios, and one for writing actuation data. To integrate the HIL components into the SmartData middleware, one must change the SmartData handler so that, instead of collecting all data from the simulation tool, we use the HIL components to collect the data they produce.

In this way, we first implemented wrappers that provide the interface for reading data on each HIL component following the implementation of a SmartData transducer as depicted in Figure 1. In other words, each wrapper provides an implementation of the operator Value() so that the SmartData object can read the data produced by each HIL component.

For time synchronization to be possible, we changed the wrapper for the IMU and GNSS so that we would also capture the timestamp linked to every sampling. In specific, a GNSS-provided nanosecond-precision timestamp.

The HIL timestamp is formatted to match system-level timer resolution. The middleware then performs time synchronization whenever an application-level threshold is exceeded. The data produced from other HIL, SIL, and the simulator are then tagged with the synchronized timestamp. Therefore, the SmartData entering the middleware is temporally aligned and correctly timestamped.

The changes to the original SmartData middleware are highlighted in red, with dotted lines representing new concepts, like the HIL and time-synchronized data, and continuous lines representing changes to a component included in the original version, i.e., the SmartData Handler. Finally, considering the fact the middleware proposed by Hoffmann et al. [2], [5] is able to account for timing due to the inherent SmartData properties, the synchronized time can be propagated and operated by data transformations (Transformational SmartData) such as other HIL, SIL, AI models, and so on, up to the actuation stage.

V. Case-Study

To corroborate the need for a time synchronization mechanism on a digital twin system, we replicated the experiments conducted by Hoffmann et al. in [5]. They extended an Immitation Learning application originally proposed by Codevilla et al. [1] over the CARLA simulator [3].

In their experiments, Hoffmann et al. implemented the SmartData_Handler through the CARLA Client API, which enables simulation customizations during execution. For instance, one could add sensors, add vehicles, collect information, and alter parameters. This integration allowed for connecting the Simulator and Simulator_Wrapper functionalities and promoting the marshaling between CARLA complaint Sensor/Control Data into SmartData, and vice-versa.

In this way, the integration of the simulation with the SmartData middleware consists of intercepting data read by the Python Client and feeding them to SmartData Services for the respective SmartData Sensors via shared memory. Each processing algorithm, SIL, and HIL that operates over the observed data is then modeled as a Transformational SmartData Service. Control data is then modeled as SmartData Actuators Services that collect necessary data for actuation. The communication between different SmartData Services is conducted through a shared message bus. Finally, control data is returned to the simulator by the SmartData_Handler

via the Client API. Check Figure 1 for a depiction of this infrastructure.

As described in Section IV, to simulate a digital twin scenario, we adapted the SmartData handler implementation to account for the HIL data sources instead of capturing it from the simulator, in this case, CARLA. Furthermore, the SmartData framework was configured to operate over Linux in a NVIDIA Jetson AGX Orin with 2048 NVIDIA® CUDA® cores and 64 tensor cores, 12-core Arm Cortex-A78AE v8.2, 64-bit CPU, 3MB L2, and 6MB L3. To enable the SmartData framework to control time synchronization, we disabled the Linux clock synchronization feature.

The experiment conducted in this work focused on measuring the deviation of the Linux timer for each sample collected from the GNSS HIL component. To do so, we collected two thousand (2000) samples on a loop that collects the Linux timestamp every time the GNSS gets a fix (i.e., the HIL returns data for the IMU and GNSS, alongside the respective timestamp). This strategy yielded a sampling with period equivalent to 1 second with 0.02 seconds standard deviation. To properly compare GNSS timestamp to Linux timestamp, we formatted GNSS timestamp from ns to μs. The threshold for synchronization was set to $500\mu s$, representing half of the next order of magnitude for precision (i.e., ms).

A. Results

Considering the experiment described, Figure 3 presents a histogram for the deviation amongst Linux timestamp and GNSS timestamp. The line in red represents the configured threshold (i.e., $500\mu s$). The presented histogram suggests high variance amongst the values observed for timestamp deviation.

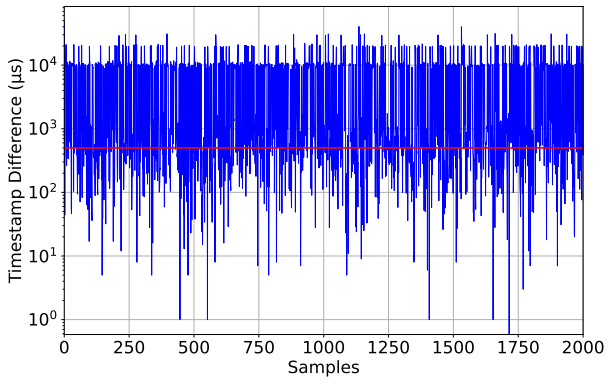

Fig. 3. Average Deviation from the SmartData Middleware over Linux to GNSS timestamp.

This notion of high variance is corroborated by the results presented in Table I. The high variability is demonstrated by the fact the average jitter, which represents the average difference between a singular deviation and the average of deviations, represents 96.98% of the average deviation itself. This, alongside an average deviation of approximately $6ms$, raises the need for time synchronization, especially when considering timing requirements for critical applications.

TABLE I
TIME SYNCHRONIZATION SUMMARY

Metric	Result	In Perspective
Average Deviation	6025.58 μs	0.6% of the 1s interval
Average Jitter	5843.89 μs	96.98% of avg.
Number of Actuations	1383	69.15% frequency

Finally, the number of actuations, representing the number of times the Linux timestamp was synchronized to the HIL timestamp, around 69.15% of the total number of verifications, also demonstrates the necessity for time synchronization for building relevant digital twin scenarios that support critical operations.

VI. FINAL REMARKS

This work built upon a SmartData-based data-driven middleware to provide transparent HIL integration to an autonomous vehicle simulation environment. We extended the original implementation to address the problem of time synchronization on digital twins by taking advantage of a high-precision time source provided by a GNSS hardware sensor attached to an IMU.

Experiments were conducted operating the middleware in a Linux-based Operating System, measuring an average deviation of 6ms when compared to the high-precision hardware-in-the-loop timestamps, with the deviation exceeding the established safe margin over 69% of the observations.

Future works include conducting more experiments with different hardware and software configurations to provide a more comprehensive analysis of the results obtained. Moreover, an in-depth investigation of the state-of-the-art in time-synchronization approaches for digital twins is necessary to provide relevant comparisons in terms of methods and overall performance.

REFERENCES

[1] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4693–4700, 2018.

[2] José Luis Conradi Hoffmann, Leonardo Passig Horstmann, and Antônio Augusto Fröhlich. Transparent integration of autonomous vehicles simulation tools with a data-centric middleware. *Design Automation for Embedded Systems*, 28(1):45–66, January 2024.

[3] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[4] Antônio Augusto Fröhlich. SmartData: an IoT-ready API for sensor networks. *International Journal of Sensor Networks*, 28(3):202, 2018.

[5] Jose Luis Conradi Hoffmann, Leonardo Passig Horstmann, and Antonio Augusto Frohlich. Integrating autonomous vehicle simulation tools using smartdata. In *2022 XII Brazilian Symposium on Computing Systems Engineering (SBESC)*, page 1–8. IEEE, November 2022.

[6] WuLing Huang, Kunfeng Wang, Yisheng Lv, and FengHua Zhu. Autonomous vehicles testing methods review. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 163–168, 2016.

[7] IEEE Instrumentation and Measurement Society. Ieee standard for a smart transducer interface for sensors and actuators - common functions, communication protocols, and transducer electronic data sheet (teds) formats. *IEEE Std 1451.0-2007*, pages 1–335, 2007.

[8] Open Robotics. Gazebo.