IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 25 April 2024, accepted 2 June 2024, date of publication 7 June 2024, date of current version 18 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3411070

RESEARCH ARTICLE

A Simulation Framework for Prototyping Intelligent Vehicle-to-Infrastructure Applications: A Case Study on RSU-Based Intersection Movement Assist for Connected Autonomous Vehicles

CHUN-TING WU, SHAO-HUA WANG, AND CHIA-HENG TU, (Member, IEEE)
Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 70101, Taiwan
Corresponding author: Chia-Heng Tu (chiaheng@ncku.edu.tw)

ABSTRACT A cooperative intelligent transport system (C-ITS) enables information sharing among ITS subsystems, such as vehicle and roadside infrastructure, with vehicle-to-everything (V2X) communications. Novel C-ITS applications aim to reduce traffic congestion and improve road safety. For example, a roadside unit (RSU) can sense the traffic status of an intersection and share the information to nearby vehicles to prevent potential collision, i.e., the intersection movement assist (IMA) scenario. Typically, C-ITS applications are developed in a simulated world to mitigate the high costs and safety hazards associated with the real-world counterpart. Unfortunately, existing simulation tools focus primarily on modeling vehicle-to-vehicle (V2V) communications, rather than the vehicle-to-infrastructure (V2I) communications involved in the above example. This poses a great challenge for developing V2I-based applications, especially for the application-level performance assessment on such systems. This work proposes a software framework that enables the simulation of full software stacks for the ITS subsystems, i.e., vehicle and RSU. This full-stack simulation capability enables hardware-in-the-loop simulations, evaluating the application-level performance and relative cost in the early stage of system development. This can shorten the time to market for C-ITS applications. We believe that this framework paves the way toward the development of novel V2I applications.

INDEX TERMS C-ITS, V2I communications, 3D simulation, image-based vehicle detection, design space exploration.

I. INTRODUCTION

With the advances of vehicle-to-everything communication technologies [1] and information technology, cooperative intelligent transport systems which involve the collaboration of ITS subsystems (e.g., pedestrian, vehicle, and roadway infrastructure) to enable ITS applications become increasingly important to improve road safety and traffic management toward smart cities. Examples of C-ITS applications [2], [3] include traffic jam warning, intersection collision warning, and traffic signal preemption. These applications rely on the exchange of environmental status data using cooperative V2X communications, such as the speed, driving direction, and position, to supply the data required by a specific C-ITS application. Furthermore, standards and/or protocols have been defined by international organizations, such as European Telecommunications Standards Institute (ETSI) [4], and Society of Automotive Engineers (SAE) International, to facilitate the development of C-ITS applications.

Simulation tools are often adopted during the early stage for the system development of C-ITS applications to test and validate the correctness and efficiency of novel ideas without considering the cost and safety issues that are arisen when the development is done in a physical world. Popular choices for evaluating the efficiency of traffic flow and management include PTV Vissim [5], SUMO [6], and TRANSIMS [7].

The associate editor coordinating the review of this manuscript and approving it for publication was Jiayi Zhang.

These traffic flow simulators can be further integrated with network simulators for building the applications with V2X communications, where VEINS [8] is such an example. VEINS uses SUMO to generate road traffic and OMNet++ (a network simulator) to handle the vehicle-to-vehicle communications, where the movement of car is simulated by SUMO and the corresponding communications among the cars are reflected in the simulation of vehicular communications by OMNeT++. It is interesting to note that VEINS can be further extended to model the vehicle-to-infrastructure communications between vehicles and a roadside unit.

While the above simulation tools are good for the development V2V-based communications (e.g., they put an emphasis on the simulation of traffic flows, which enable the modelling of the V2X communications for simulated vehicles), they are not suitable for the system prototyping for V2I-based C-ITS applications since they do not support the simulation of the full software stacks of an RSU and a vehicle (for an autonomous vehicle), respectively. In such a case, the application-level performance of a V2I C-ITS application is hard to be estimated during the system development stage with the simulations. Taking a V2I-based intersection movement assist (IMA) application for autonomous vehicles illustrated in Figure 1 as an example, a *connected autonomous vehicle*[1] obtains traffic information (i.e., the ambulance from the side) that is detected and broadcasted by an RSU (the roadway infrastructure) to avoid potential side collision, and unfortunately, the existing simulation tools cannot be used to estimate the end to end latency from the sensing done by the RSU to the reaction taken by the autonomous vehicle after receiving the traffic information. Therefore, critical design decisions cannot be made without the application-level performance information.

This work aims to provide a software platform to enable the development of a V2I-based application using a full-stack system simulation, where the software stack of each participant (e.g., vehicle, pedestrian, and roadside unit) can be run in the simulation system. The full-stack simulation further enables a hardware-in-the-loop simulation (i.e., the realistic computing hardware for each participant is adopted in the simulation), and hence, the application-level performance is able to be gauged during the simulations. This software platform facilitates the system prototyping and the performance measurement before the system is actually deployed in the filed, shortening the time to market. The contributions made by this work are summarized as follows.

- A simulation-based software framework is proposed and developed to facilitate the development of a V2I-based C-ITS application. This framework can be further used for the hardware-in-the-loop simulation to assess the delivered performance at the application level. To the best of our knowledge, we are not aware of any other work that enables the full-stack system simulation for C-ITS V2I-based application development.
- A case study of a real-world IMA application (defined in SAE J2945/1) is conducted to evaluate the effectiveness and the capability of the proposed framework. The experimental results demonstrate the correctness of the IMA application that is built with the CARLA [9] simulator to provide a virtual world and the traffic (the ambulance in Figure 1), the RSU for sensing the simulated traffic and broadcasting the V2I messages, and the connected autonomous vehicle that reacts to avoid the traffic accident based on the perceived data.
- A hardware design exploration for the RSU (in Figure 1) is performed as an example to further showcase the advantage of the full software stack enabled simulation. Three different computing hardware platforms are used to evaluate different performance and cost alternatives for the intelligent vehicle detection module of the RSU for sensing the traffic information.

[1]An autonomous driving vehicle that is equipped with a wireless communication device for V2X communications, as will be introduced in Section II-B.

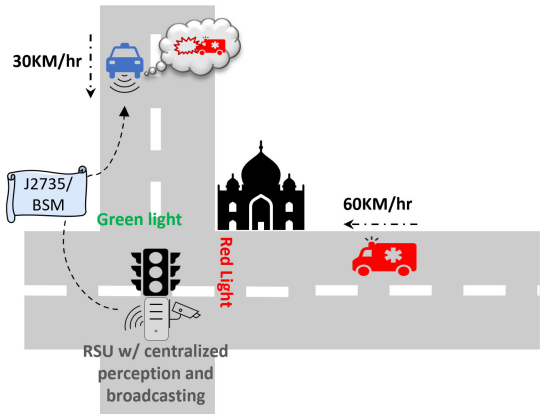


FIGURE 1. An example of utilizing RSU-centric traffic condition perception and broadcast for the IMA application scenario, where the RSU broadcasts the detected vehicle information (red ambulance) to the connected autonomous vehicle (blue) for collision avoidance at the intersection.

In the remainder of this paper, Section II provides the background information and the related work for V2X communications (and related international standards), applications, and simulations. The overview of the proposed software framework is introduced in Section III. The case study conducted in this work based on Figure 1 is described in Section IV. The experimental results of the proposed framework are given in Section V. Section VI concludes this work.

II. BACKGROUND AND RELATED WORK

This work provides a simulation platform for the development of vehicle-to-infrastructure communications for the intersection movement assist application. To offer the background information of this work, Section II-A gives an introduction to vehicle-to-everything communications and standards. The ITS applications built with the vehicle-to-everything communication support are introduced in Section II-B. The

existing simulation tools that are used for the development of ITS applications are described in Section II-C.

A. V2X COMMUNICATIONS AND STANDARDS

The wireless communications that are used to exchange information between vehicles (and their drivers) and other roadway entities (such as roadway infrastructure, other vehicles, and pedestrians) are called vehicle-to-everything communications. V2X communications can be further categorized into different types, such as vehicle-to-vehicle (V2V), vehicle-to-pedestrian (V2P), and vehicle-to-infrastructure (V2I) communications. For example, this work focuses on the V2I communications that transfer the data between a vehicle and a roadside unit. The V2X communications are enabled by either the IEEE 802.11 standards or the LTE based standards to physically transfer the data. The former standard uses the communication technology called Dedicated Short-Range Communications (DSRC) [10], whereas the latter uses the Cellular vehicle-to-everything (C-V2X) technology [11].

International standards, such as ETSI ITS-G5 and SAE J2735 [12], have been developed for vehicle communications in cooperative intelligent transport systems. These standards are constructed on top of the DSRC/C-V2X communication technologies. The SAE J2735 standard defines message formats, such as Basic Safety Message (BSM), Signal Phase and Timing (SPaT), Map Data (MAP), Signal Status Message (SSM), and Signal Request Message (SRM). These message formats can be used in the ITS application scenarios specified in the SAE J2945 standard [13]. For example, the specification of SAE J2945/1 focuses on vehicle-to-vehicle safety communications, where vehicles share their location and other status information via the J2735 BSM format. Particularly, the J2945/1 specification further defines a V2V based application scenario, **Intersection Movement Assist (IMA)** [14], preventing collisions between vehicles at the intersections. This is done by exchanging BSMs frequently via V2V communications to proactively avoid collisions. It is important to note that this work follows the IMA scenario with V2I communications (illustrated in Figure 1), instead of the V2V communications, as will be introduced in the following subsection.

B. V2X-BASED C-ITS APPLICATIONS

A considerable amount of efforts have been made for building V2X applications to improve road safety. These applications are developed from the perspective of vehicles (through V2X communications) or from the view point of roadside units (through V2I communications). The following paragraphs introduce the existing works from the two different perspectives.

V2X applications are built upon the communications among different entities on roads, such as vehicles, pedestrians, and roadside units, where each entity is equipped with a wireless communication device to transfer data. For example, a vehicle using a DSRC or C-V2X device to transfer data is referred to as a **connected vehicle**. Considering the coverage of wireless communications, a concept of collaborative environmental perception system for connected vehicles [15] is proposed to share perceived status (especially for those non-connected vehicles) to other nearby connected vehicles. Another similar work [16] is done by integrating open-source software, Autoware [17] (self-driving software) and OpenC2X (communication software), to share traffic status among *connected autonomous vehicles* in the vicinity. In addition to the V2V communications, the perceived traffic status (collected by a connected vehicle) can be shared with a roadside unit, which can further broadcast the related information to other connected vehicles. The data format used to record the traffic status follows the ETSI standard (i.e., Cooperative Awareness Messages, CAM).

V2I applications put an emphasis on the communications between vehicles and roadside units. An RSU acts as a proxy system [18] (for a connected vehicle) to sense surrounding status (for the information of non-connected vehicles) and to broadcast such information through V2V messages (i.e., CAMs). The detection of surrounding vehicles is done by a stereo vision technology attached to the RSU.

A similar RSU-centric approach has been done in [19] that implements a centralized perception and broadcast (CPS) application in an RSU, where a vision-based vehicle detection system is attached to the RSU that converts the information of the detected vehicle into the J2735 BSM format and broadcasts the information to the connected vehicles. This work builds the CPS application by extending the work [20] that focuses on the design of the control software for a roadside unit to enable multiple applications running on top of the roadside unit and different types of SAE J2735 messages are used to communicate with connected vehicles in the vicinity. Traffic signal preemption and priority applications, such as emergency vehicle signal preemption (EVSP) and transit signal priority (TSP), have been developed in the RSU software in [20]. The scalability issue of the control software of the RSU is further discussed in [19].

C. ITS APPLICATION SIMULATIONS

Simulation tools that are used for the development of intelligent transportation systems are introduced in the following subsections. These tools can be divided into three categories, each of which focuses on different perspectives. The first category is for traffic simulations (Section II-C1) that are often used to plan the transportation strategies in a city. The second category is for autonomous vehicle simulations (Section II-C2) that are used to develop autonomous vehicles, especially for the self-driving software. The last category is for vehicle-to-vehicle application simulations (Section II-C3), where the applications can be built on top of the simulations of multiple autonomous vehicles.

1) TRAFFIC SIMULATIONS

Simulation of Urban MObility (SUMO) [6] is an open-source software that is widely-adopted for traffic simulations, e.g.,

to make traffic management plans for smart city. It is used by transportation and urban planning research communities for the simulation of traffic flows and the evaluation of traffic control strategies. Users can create a realistic road network by importing the road network from OpenStreetMap [21]. Novel traffic signal timing plans can be added into SUMO and evaluate the performance of the plans under random generated traffic.

One advantage of SUMO is the capability of simulating a large road network and provide the intermodal simulation with pedestrians and random vehicles. This helps the performance analysis of complex traffic scenarios involving different vehicles, such as cars, buses, bicycles, and pedestrians. Additionally, SUMO provides a way to visualize the simulations, which is useful for users to observe and understand the simulation process and result. MATSim [22] is another open-source tool that is developed for large-scale scenarios, such as a city-wide or regional transportation plan. It is specialized in simulating millions of agents on huge networks rapidly, and is equipped with methods to analyze the simulation outputs. For instance, MATSim can be built for analyzing the impact of tolls on traffic, showing an increase in traffic in the peak hours and a decrease in the off-peak hours. TRANSIMS [7] is a free simulation tool for a regional analysis of transportation systems. Similar to SUMO, it supports traffic synthesis, activity generation, and the microscopic traffic simulation. It is interesting to note that TRANSIMS uses a different simulation model than SUMO; that is, the former uses the space-discrete time-discrete cellular automata simulation, and the latter adopts the space-continuous time-discrete approach. In addition to the open-source software for traffic simulations, there are commercial solutions for such a purpose, such as PTV Vissim [5]. They incline to provide the infrastructure for virtual testing and traffic simulation to facilitate the product development, e.g., shortening the time to market.

2) AUTONOMOUS VEHICLE SIMULATIONS

One of the popular, open-source software for autonomous vehicle simulations is the CARLA simulation that is built upon the famous game engine, Unreal Engine 4, for handling the 3D simulation of real-world scenes, as well as the physic effects of simulated objects. The major advantage of the autonomous vehicle simulators is that it enables the full stack simulation of autonomous driving software, as will be introduced in the following paragraphs. Among other similar projects, such as the LG SVL simulator [23] that is a no longer updated project and AirSim by Microsoft Research [24], CARLA has a broadened support for third-party software to extend its capability. OpenCDA [25], as will be introduced in the following subsection, is a good example that it includes a traffic simulator and a networking simulator to facilitate the vehicle to vehicle applications development.

A client-server architecture is adopted by CARLA for the simulations, as depicted in Figure 2. The CARLA

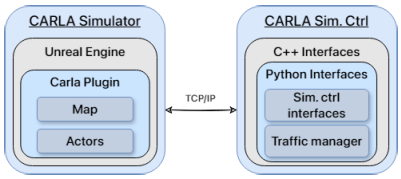


FIGURE 2. The software organization of the simulations with CARLA.

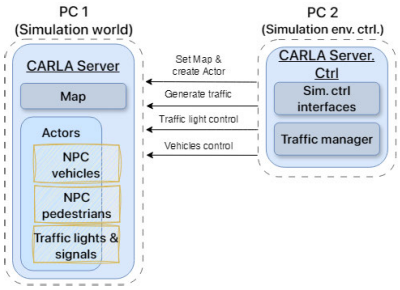


FIGURE 3. A closer view of the client and the server in a CARLA simulation.

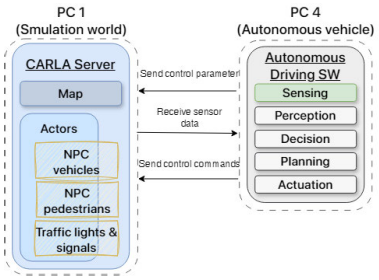


FIGURE 4. The communications between CARLA and Autoware.

server is responsible for handling the simulation itself and taking the requests (to control the simulation) from the CARLA client, where the client-server communications are established upon the TCP/IP connections. At the client side, a set of programming interfaces (C++/Python) is exposed to users to write the scripts to control the simulation parameters, such as map, vehicles, pedestrians, and weather conditions. Customized traffic scenarios can be created by the CARLA APIs to control the objects in the simulated world, such as vehicles and pedestrians that are referred to as *actors* in CARLA. A more detailed view of the client-server interactions is illustrated in Figure 3.

One of the key features of CARLA for the full (software) stack simulation of an autonomous vehicle is that the autonomous driving software is able to acquire the sensor data in the virtual world as its inputs as if the entire software system runs on a realistic autonomous vehicle. These sensor data, such as Lidar, Radar, and GPS, are transferred from the CARLA simulator to the autonomous driving software through the Robotic Operating System (ROS) bridge interface [26]. Particularly, a ROS message topic is used to encapsulate the data of a senor. Therefore, novel algorithms for autonomous driving can be developed and trained in the simulated world.

Autoware [17] is a commercial-grade, open-source software platform that is widely-adopted for building

autonomous vehicles. It often runs on top of Linux systems and is built with the ROS software framework for the inter-software-module communications. Autoware has the essential software implementation for the key functions for autonomous driving, including sensing, perception, decision, planning, and actuation. Users can develope novel algorithms in Autoware and validate their designs by the co-simulation of the CARLA simulation and Autoware, where there is a customized ROS bridge dedicated for exchanging data between CARLA (server) and Autoware (client). As shown in Figure 4, the CARLA server feeds the sensor data to Autoware that sends the control commands to the CARLA server after making the control decision based on the received sensor data, where the communications are done through the ROS bridge. It is noteworthy that the simulated vehicle controlled by the Autoware's commands is called the *ego* vehicle, which is different from those simulated vehicles controlled by CARLA simulator.

It is important to note that in Figure 3 and Figure 4, a machine is dedicated for the execution of the software because we would like to put an emphasis on the full stack simulation of an autonomous vehicle. For example, Autoware could be run on PC 4, and the driving knowledge is able to be trained in the setup illustrated in Figure 4. The advantage of such an arrangement for system development is that PC 4 (the computing hardware for an autonomous driving) can be plugged into a physical vehicle for autonomous driving. In this case, the effort of migrating the developed software system to a physical vehicle can be minimized. Furthermore, such an arrangement helps the exploration of the hardware specification of an autonomous driving vehicle at the system development stage since the delivered performance of such a computing hardware is able to be measured with the simulations.

3) V2V APPLICATION SIMULATIONS

OpenCDA [25] is an open-source framework that incorporates different software, such as CARLA, SUMO, and NS-3, for cooperative driving automation (SAE J3216), and can be extended for different vehicle-to-vehicle applications. Thanks to the support of multiple ego vehicles by CARLA, it is easy to simulate multiple autonomous vehicles with OpenCDA. In this case, multiple CARLA clients are present, each serves as an ego vehicle and can receive sensor data from the CARLA server for autonomous driving. The development of novel algorithms for V2V application can be facilitated via OpenCDA.

When integrated with SUMO, which is used to generate the map and the traffic flows required by the simulation, OpenCDA can run the simulation on the road network defined by the map, with the generated traffic scenarios. The vehicle-to-vehicle wireless communications may be adopted by the network simulator, NS-3. For instance, the network simulator can be added in OpenCDA to analyze if the simulated conditions can affect the network performance, e.g., the potential of massive packet drops.

VEINS (Vehicles in Network Simulation) [8] is an open-source software for V2V applications, where the V2V communications are done on top of vehicle networks. VEINS incorporates with existing software tools, SUMO and OMNeT++, for modeling road networks and vehicle networks, respectively. The road traffic simulation is handled by SUMO, and the vehicle network simulation is performed by OMNeT++. With the support from SUMO, VEINS is able to simulate a larger-scale scenario, such as a metropolitan area. Note that he physical layer modeling of vehicle networks is possible by further leveraging MiXim toolkit to take into account the effect of radio interference and obstacles on the simulated road.

4) DISCUSSION

To distinguish our work from the existing efforts, we present a summary of simulation tools that can be employed for the development of ITS applications. These tools can be broadly classified into two categories: traffic flow simulations and autonomous driving simulations. The former focuses on the aggregated behavior of traffic flow and evaluate the efficiency of a transporation infrastructure on a given road network. In contrast, the latter uses a detailed level modeling of an autonomous driving vehicle, mimicking the behavior of the self-driving logic with 3D simulations, which is useful for evaluating the algorithms and functionality of an autonomous driving vehicle. In between the two groups, there exists research works that combines the efforts to provide various V2X-oriented ITS applications. These research efforts are described as follows.

- Traffic flow simulations are important for the design, analysis, and evaluation of a transportation system. For example, traffic simulations can incorporate various traffic signal control strategies, such as fixed-time and adaptive methods, to determine the efficiency of these strategies on the road network of a urban area. These simulator frameworks listed in the table adopt the microscopic perspective for traffic simulations. The microscopic-level simulations take into account the interaction of individual vehicles (pedestrians) while evaluating the delivered performance of the transportation system. This is achieved by mimicking the flow of individual vehicles through a target road network, where different modeling techniques can be used in the simulations, such as car-following and lane changing mechanisms. Examples of traffic flow simulators include MATSim [22], PTV Vissim [5], SUMO [6], and TRANSIMS [7].
- Autonomous driving simulations aim to provide the digital version of a real driving environment for the design, development, and test of autonomous driving control systems by using difference driving scenarios. For example, different weather conditions, roads/intersections, and behaviors of moving objects, such as pedestrians and vehicles, can be simulated to observe and evaluate

the reactions of the autonomous driving logics. The level of detail for such simulations is at individual autonomous vehicles, where each autonomous vehicle can run in the simulated environment as if it is run on real, physical environment. This is achieved by the full-stack simulation of the autonomous driving software. Examples of autonomous driving simulators include CARLA [9] and LG SVL [23].

- V2V application simulations focus on exploring the possibilities achieved by vehicular ad hoc networks for ITS applications. A wireless ad hoc network can be established by the connected vehicles on the roads, and the traffic flow and safety could be improved as these connected vehicles can exchange information about their states, e.g., position, speed, and direction. An interesting example is that by leveraging V2V communications, connected autonomous vehicles can share their perceived information about the environment, and this cooperative perception is very useful for challenging driving situations, such as sever occlusions. Examples of V2V application simulation frameworks include OpenCDA [25] and VEINS [8].
- V2I application simulations put an emphasis on information exchanging between connected vehicles and road infrastructure. Possible applications include 1) emgergency response, allowing connected vehicles to clear the way for an emgergency vehicle to respond to a situation, 2) optimized traffic signals, adjusting traffic light timings based on real-time traffic flow to reduce waiting times, 3) eco-friendly driving, providing connected vehicles with information on fuel-efficient routes, and 4) collision avoidance, warning connected vehicles of hazards ahead or abnormal situations (e.g., an ambulance running a red light in our IMA application) to avoid crashes. Our proposed framework falls in this category.

Our proposed framework is very different from the simulation tools of the first three categories, in terms of functionality (V2I applications) and capability (hardware-in-the-loop simulations). The framework proposes the methodology for designing a V2I application and provides the actual hardware setup to realize the IMA application; this kind of V2X applications is not seen in the literature. Moreover, as CARLA supports the simulation of multiple ego vehicles, the application scenario described in Figure 1 could be changed to the EVSP application scenario. In this EVSP application, the red ambulance is also a connected vehicle making a signal preemption request to the RSU through the SSM/SRM messages, and the autonomous connected vehicle can respond to the situation after it receives the relevant messages (SPaT for the traffic light information) from the RSU.

Furthermore, our framework can be considered as a complement to the existing efforts. That is, when system designers want to augment the scope of a target application system, the above tools can be added into our framework to provide additional functionalities. For instance, a traffic flow simulator, such as SUMO, can be integrated into our framework (as done by OpenCDA) to run the IMA application simulation covering a larger area of road network; SUMO can simplify the procedures of setting traffic light plans and arranging a road network. If the capability of V2X communication is considered, a network simulation tool, such as NS-3 or OMNeT++, can be added to further analyze the impact the performance of the communication network on the overall application performance.

III. THE PROPOSED SOFTWARE FRAMEWORK

The proposed software that is able to be used for developing V2I Cooperative ITS applications is introduced in this section. In particular, the overall architecture of the proposed framework is presented in Section III-A. The key concept for building a V2I application with the framework, and the example V2I applications that can be added into the simulated system are elaborated in Section III-B. Section III-C discusses the discrepancies between the simulated system and the physical system, regarding the code porting and performance issues. It is important to note that it is possible to extend the CARLA-based software framework for V2V and V2P (vehicle to pedestrian) applications via connecting with the CARLA actors (i.e., non-player character vehicles and pedestrians objects) to exchanging messages. It is worth mentioned that the framework facilitates the development of the system for the V2I applications, the performance evaluation of the established system, and the exploration of different design alternatives for computing HW of such a system (Section V).

A. SOFTWARE ARCHITECTURE

The proposed framework is centered around the CARLA simulator introduced in Section II-C2 for creating a virtual environment modelling a real-world traffic condition with realistic road layout, surrounding buildings, traffic signals, moving vehicles, and pedestrians. The important components can be attached to the CARLA simulator to extend the capability and to interact with the virtual world. For establishing a V2I application, the framework consists of four software components, CARLA simulator, CARLA simulation controller, RSU, and connected autonomous vehicle, as illustrated in Figure 5. Each software component is assumed to run on a standalone computer (i.e., PC 1 to PC 4), enabling the hardware-in-the-loop simulation that can better evaluate the HW performance requirement for each component (for evaluating HW design alternatives). On the other hand, some of the components can be co-located at the same PC if the underlying hardware provides necessary computing power. The four components are described as follows.

- **CARLA simulator (*CARLA server*)** provides 3D simulations for real-world scenes and traffic that are

controlled by a *CARLA client*. The status of the simulated objects (e.g., moving vehicles and traffic light signals) can be retrieved by external software module (e.g., our developed CARLA adaptor in RSU SW, such as adjusting the traffic light signal to green light by the traffic light control in RSU SW for a priority signal application), and the status of the objects can be adjusted by the adaptor on top of the CARLA client/server communication protocol.

- **CARLA simulation control *(CARLA client)*** is responsible to set up the simulation environment for the simulation required by a target V2I ITS application. It is assumed that the physical environment for the application, such as road layout, buildings, trees, and traffic light poles, is pre-built in the map, so that this CARLA client can instruct the server to load the map to run the simulation. The internal interactions between the CARLA server and client are provided in Section II-C2.
- **RoadSide Unit** plays an important role in a V2I ITS application to communicate with connected vehicles and to report/update the traffic conditions. Furthermore, from the software perspective, the communications between an RSU and a traffic light (simulated by CARLA) are done by the CARLA adaptors (one for retrieving status and the other for issuing commands to adjust the traffic light signals of the intersection), whereas the communications between an RSU and a connected vehicle are done through the J2735 messages and protocol for the target V2X application. The V2I communications of the proposed framework are further introduced in Section III-B. With the communication facilities, the cooperative ITS applications are facilitated by RSU, for example, EVSP for traffic signal preemption for emergency vehicles, TSP for traffic signal priority for public transit, and CPS for environmental traffic sensing and broadcasting that are described in [20]. Especially, CPS is essential for the V2I application adopted in this work to collect runtime traffic status and broadcast to connected vehicles. CPS will be introduced in Section IV that gives a concrete example application scenario as the case study in this work.
- **Connected autonomous vehicle** is able to do the V2I communications with an RSU, and in the case study, it is specifically used to perform the IMA application that receives the runtime traffic status from an RSU and reacts if necessary (e.g., to control vehicle speed to avoid potential collision at the intersection, as will be further introduced in Section IV). By default, this autonomous vehicle is instructed to perform a designated self-driving task in the simulated world (described in Section II-C2), and it can be participated in the V2I application by further enhancements done on the autonomous driving software, by adding the communication facilities (both hardware and software part for V2X communications) and by introducing the collected traffic status into the autonomous driving algorithm for collision avoidance, which is further introduced in Section IV.

There are (logical) connections between the components being omitted in Figure 5 to focus more on how the proposed framework can be adopted for prototyping V2I applications. The figure puts an emphasis on the connections among the software components, and the physical communications of the machines to run the software are done with Gigabit Ethernet providing a baseline network performance. Such an arrangement is good for the system prototyping for validating application functionality correctness (e.g., which can be evaluated by the visual output generated by the CARLA simulator), and for providing a baseline performance delivered by the physical machines. That is, designers are able to evaluate the performance delivered by different hardware platforms for design space exploration. It is important to note that when V2X communication performance estimation is desired, it is possible to model the network performance with a mathematical model to enable a fast simulation speed or with a network simulator (e.g., NS) to provide a detailed network performance characteristics. Alternatively, the physical computer networking can be done by mounting physical V2X communication devices (for the hardware-in-the-loop simulation) to better reflect an end-to-end latency of certain operations; this is adopted in this work and is elaborated in Section IV.

Another advantage of the proposed framework is that it is able to develop a V2V application (on top of the V2I scheme mentioned above) with multiple connected autonomous vehicles. Thanks to the native support by CARLA simulator, it can be achieved by introducing extra instances of autonomous driving software (e.g., Autoware), each of which has its own OBU software for V2X communications (i.e., V2I and V2V communications in this context). In such a V2V application, the physical networking between the CARLA server and the *connected autonomous vehicles* should be a major concern since it demands for a larger networking bandwidth to transfer the sensor data required by these connected autonomous vehicles to make plans for self-driving in the simulated world. Furthermore, the V2V application logic can be developed on each instance of autonomous driving software to fulfill the application requirement.

B. SIMULATION FOR V2I APPLICATIONS

The RSU (PC 3) is the central to simulated V2I applications as it can collect the environmental status in the simulated world (PC 1), such as traffic conditions or traffic light signals, and can also enforce the status updates dictated by a V2I application, e.g., adjusting traffic light signals. Both of the functionalities are achieved by the developed CARLA adaptors in the RSU SW to interact with the 3D simulator (CARLA), as illustrated in Figure 6.

The other role of the RSU is to interact with connected (autonomous) vehicles (PC 4). It is worth mentioning that generally speaking, RSU can talk to ''connected vehicles''

that are equipped with V2X communication devices for V2I applications, where OBU SW is responsible for handling at the software layer and the V2X communication devices are used to physically transfer the data. In this work, as the IMA application is considered to showcase the capability of the proposed framework, we focus on connected autonomous vehicles; that is, connected vehicles have the capability for autonomous driving.

1) V2I APPLICATIONS IN RSU

To provide a basic support for V2I applications, RSU needs to provide the environmental status to those connected autonomous vehicles (i.e., J2735 MAP and SPaT messages for geo information and real-time traffic light signal information). With the basic traffic information, more advanced V2I applications can be established, such as EVSP, TSP, or CPS. Moreover, when a different V2I application is considered, a different type of J2735 message could be adopted. For example, the traffic signal preemption and priority applications (EVSP and TSP) would use the SSM/SRM to request for traffic light signal alternation and to receive the responses for the requests, respectively. On the other hand, when the RSU-centric vehicle status report is desired (for the CPS application), the BSM is used to record the status of vehicles nearby the intersection (that is governed by the RSU). It is worthy to note that although the CPS module in Figure 6 is responsible for preparing/emitting geo, traffic light, and surrounding vehicle information, it is possible that the different types of messages are handled by different software modules, depending on the software system design requirements.

C. DESIGN CONSIDERATIONS IN MIGRATING FROM THE SIMULATED SYSTEM TO THE PHYSICAL SYSTEM.

V2I applications rely on the external software interfaces of RSU to fulfill the application needs, and these external interfaces should be considered carefully while migrating the prototyping system from the simulated world to the physical counterpart; this is also true for the connected autonomous vehicles. These external interfaces can be divided into three categories: 1) acquiring surrounding traffic condition (e.g., geo information, moving vehicles), 2) accessing to the traffic light signals,[2] and 3) communicating with the surrounded connected (autonomous) vehicles. To fulfill the need of V2I applications, the first and second categories of the interfaces are majorly used to obtain the environmental information that is passed to the nearby connected (autonomous) vehicles via the third category of the interfaces.

To acquire the surrounding traffic condition (1st category) and broadcast to nearby connected (autonomous) vehicles (3rd category), the RSU can retrieve the status of simulated objects (e.g., the position and moving trajectory of a simulated vehicle or pedestrian) through the CARLA server/client interface (the left-half of Figure 6; described in Section II-C2), and the status of the objects are encapsulated into the corresponding types of J2735 messages sent to the connected vehicles (the right-half of Section 6). In such a case, the differences of the simulated and the physical worlds are 1) the way to access the traffic condition, and 2) the way to transmit the J2735 messages.

1) *The way to access the traffic condition* in the simulated environment is done by the "knowing-all" CARLA simulator. It is obviously not applicable to the physical world. Instead, to simplify the migrating effort and to provide an intelligent solution, an object-tracking-based solution can be adopted in the simulated world during the system prototyping to infer the moving objects in the simulated intersection handled by the RSU, and the detected object information can be passed to the nearby connected vehicles. This matches the application features provided by CPS [19], which is illustrated in Figure 8 of Section IV.
2) *The way to transmit the J2735 messages* can be identical in both simulated and physical world at the software layer if the adopted V2X communication library is able to expose the same software interfaces without depending on the underlying V2X communication devices. For example, our adopted library provided by Unex Technology Corp. has the flexibility to switch between the Ethernet network and the V2X communication network, where the former can be adopted in the simulated world through a Gigabit Ethernet network interface and the latter is used in the physical world through the V2X communication device. The major difference between the two communication mechanisms is the data transmission latency and bandwidth. In our experiments, in order to measure the performance delivered by an RSU and

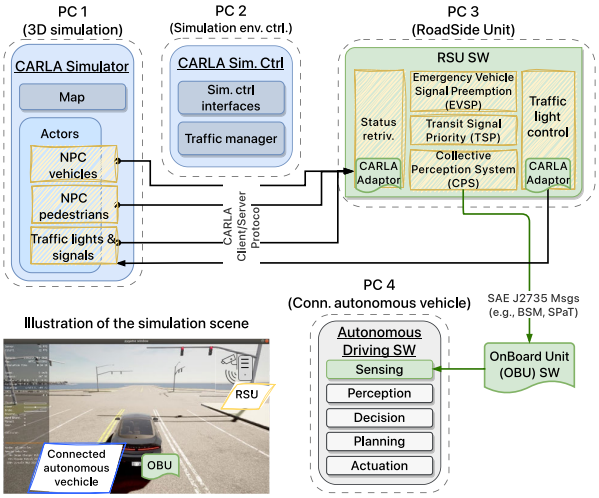


FIGURE 5. The organization of the four components of our proposed software framework for prototyping V2I applications, where the *I*nfrastructure is represented by the RSU (PC 3) and the *V*ehicle is the connected autonomous vehicle (PC 4). The concept of a V2I application enabled by our framework is depicted at the left-bottom corner.

[2] An RSU is assigned to handle the traffic lights of an intersection of the simulated world, which is also true in the real world system.

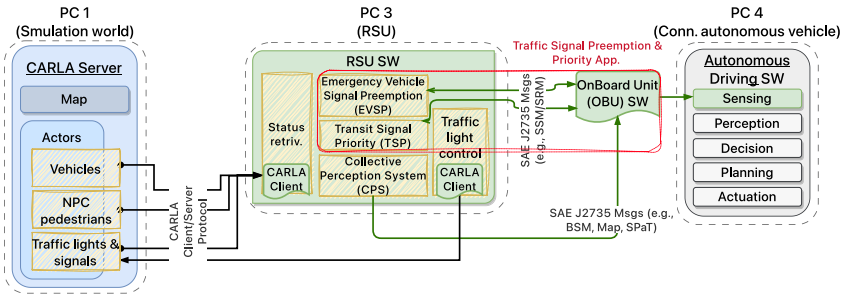


FIGURE 6. Highlights of the message flows for V2I applications using the proposed framework.

a connected autonomous vehicle via the hardware-in-the-loop simulation, the latter case is adopted and is further detailed in the following section.

To access traffic light signals (2nd category) and broadcast to nearby connected (autonomous) vehicles (3rd category), the RSU can leverage the CARLA server/client interfaces to interact with the object that is responsible for handling the traffic light signals for the very intersection (that is handled by the RSU), and then pass the retrieved information to the connected vehicles. When it is required by the V2I application for adjusting the traffic light signals (such as EVSP or TSP), the same interfaces can be used to change the signals in the simulated world for the traffic signal preemption and priority application requirements. In this case, the major differences between the simulated and physical worlds in accessing to the traffic light signals are twofold, which are listed as follows.

1) *The software (and hardware) interfaces* to accessing the traffic light signals in both worlds are different. For traffic signal accessing, the CARLA server/client interfaces are done by C++/Python interfaces over the TCP/IP protocol in the simulated world, which is very different from the physical world, where a realistic traffic light controller could be accessed by passing encoded commands to hardware registers through a specific hardware interface, such as RS-232. For example, in a physical world, an RSU has the access to the traffic light controller for an intersection through the RS-232 interface. A software abstraction layer is needed to accommodate the discrepancy between the simulated and physical worlds for accessing the traffic light signals of the virtual/realistic traffic signal controllers, so as to minimize the code porting overhead from the simulated system to the physical one. On the other hand, the hardware discrepancy results in potential performance differences. For example, the latency of Gigabit Ethernet is about 1 millisecond, whereas that of the RS-232 hardware is about 1064.3 milliseconds for a data transfer of 1024 bytes. In addition, the computing power between the simulated and physical "traffic light controllers" is different, and thus, when the performance delivered by the controller is desired, this factor should be taken into account.
2) *The traffic signal timing plans* used by a traffic light controller of an intersection should be taken into account when a target V2I involves the alternations of the traffic light signals, such as EVSP and TSP, to better reflect the performance delivered by the target V2I applications. In particular, if real-world signal timing plans (for traffic lights) are desired to be adopted in the simulation, the relevant control plans (e.g., fixed-time and full dynamic signal control plans [27]) could be implemented in the *traffic light control* module of the RSU to evaluate the impact of different control plans on the application performance. Alternatively, the signal timing plans can be implemented at the CARLA server side (by revising the *traffic light* actor or by adding an addition module attaching to the actor). When a V2I application does not require the modification of the traffic signals, the default traffic signal timing plan (offered by the traffic light actor in CARLA simulator) should be sufficient. It is interesting to note that *traffic-adaptive signal control* [27], although it is not a V2I application, can be implemented as an RSU application along with EVSP, TSP, and CPS to observe the resulting performance.

IV. CASE STUDY ON THE V2I-BASED IMA APPLICATION WITH THE HARDWARE-IN-THE-LOOP SIMULATION

This work conducts a case study on the IMA application illustrated in Figure 1 to showcase the effectiveness of the proposed framework. A more concrete scenario of the target IMA application conducted in this work is depicted in Figure 7. Specifically, a red car is a connected autonomous vehicle driving toward the intersection (it follows the path pointed by the blue arrow and is about to make a right turn with a green light), and a white emergency vehicle is a non-connected vehicle driving toward the intersection (it is a non-player character vehicle object controlled by CARLA simulator, following a fixed routing path along the red arrow to cross the intersection). As the active sensors (e.g., lidar) on the connected autonomous vehicle cannot detect the coming vehicle (the non-connected vehicle) because the building sits in between the two vehicles, it would cause a car accident (i.e., the red car collides with the white car). To avoid potential traffic collision, the CPS application [19] is deployed in the RSU to detect the vehicles driving toward the

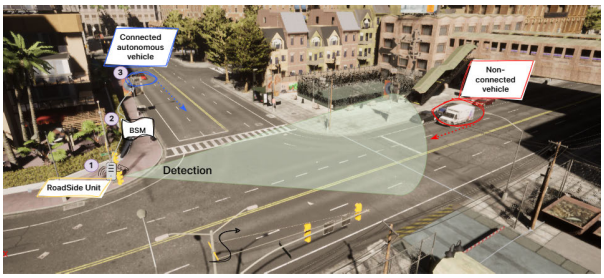


FIGURE 7. The CARLA simulated world for the IMA application scenario illustrated in Figure 1.

intersection and to broadcast the information of the detected vehicles to surrounding connected (autonomous) vehicles to avoid collision through the J2735 BSMs. The corresponding system established by the proposed framework to prototype the IMA application is illustrated in Figure 8.

A. CPS IN RSU

Instead of retrieving the moving vehicles from the CARLA simulator, an intelligent, object-tracking-based CPS design is adopted in this work (as described in Section III-C). The prototype system emulates the design that a camera is attached to the RSU to monitor and detect the vehicles moving toward the intersection. As illustrated in the left-half of Figure 8, in the prototype system, the RSU accepts the image stream (the black, dotted arrow) constantly from the CARLA server, as if there is a camera attached to the RSU to feed the image stream for vehicle detection (1). The information of the detected vehicle is then passed to the broadcast module (2) that encapsulates the information of the detected vehicles into the J2735 BSM format and broadcasts the BSMs periodically to surrounding connected (autonomous) vehicles (e.g., at 10 Hz per the J2945 standard). Based on our empirical experiences, the adopted solution for the image-based vehicle detection (1) has a great impact on the delivered performance of the RSU, and various solution alternatives are tested and evaluated in the following section. This also showcase that the proposed framework is also good for design space exploration during the system prototyping stage.

B. COLLISION AVOIDANCE IN CONNECTED AUTONOMOUS VEHICLE

The connected autonomous vehicle is planned to drive down the road and makes a right turn at the intersection by its autonomous driving logic. To cope with the IMA application scenario, the vehicle needs to receive the BSMs and to be aware of the surrounding vehicles that may potentially cause the collision and should avoid the collision when necessary. To this end, in the right half of Figure 8, an OBU module is used on the connected autonomous vehicle (PC 2) to receive the broadcasted BSMs and autonomous driving software on the vehicle should be modified accordingly (e.g., by revising the module in perception stage (3)) to interpret the decoded BSMs, determining if a countermeasure should be made based on the perceived surrounding traffic condition. The data flows of the BSMs sent from the RSU to the connected autonomous vehicle are illustrated by the green, dotted arrows.

C. THE HARDWARE-IN-THE-LOOP SIMULATION

The V2I-based IMA application is emulated by the CARLA simulator that is used to generate the workload driving the execution of the RSU and the connected autonomous vehicle for the collision avoidance. To better gauge the system performance, the physical hardware platforms are adopted for running the RSU software (Industrial PC), the autonomous driving software (PC 2), and the data transmission over the J2735 protocol (C-V2X comm. devices), where the CARLA simulator (PC 1) connects with the RSU over Gigabit Ethernet, and the RSU links to the connected autonomous vehicle (PC 2) through C-V2X communication hardware. Through generating the workloads by the CARLA simulator, the software correctness and the delivered performance of the RSU and the connected autonomous vehicle can be validated and evaluated with a lower cost without deployed in the field. The hardware and software specifications adopted for this case study, as well as the related performance results, are provided in the following section.

D. IMPLEMENTATION REMARKS

The workflow of the V2I application simulation is illustrated at the bottom of Figure 8. The workflow follows the flow of the vehicle data that are generated by the 3D simulator, detected by the RSU-CPS, and perceived by the autonomous driving software. The handling of the vehicle data requires the enhancements in the 3D simulator, the RSU, and the autonomous driving software. For the RSU, our developed CPS system [19] is adopted in the simulation, and a CARLA client software is added for redirecting the streaming video frames to the image-based vehicle detection module. The enhancements done in the 3D simulator and the autonomous driving software are introduced as follows.

- The IMA application scenario is virtually created by the CARLA simulator. A CARLA client software is used to establish the virtual world, as mentioned in Section Section II-C2. This is done by creating the required *actors* for NPC vehicles, pedestrians, traffic lights with the help from the `Blueprint` class, where the `Actor` and `Blueprint` classes are part of the CARLA Plugin library. These created objects are highlighted in the light yellow rectangles within the CARLA server of PC 1 in Figure 8.
- In order to emulate the image-based CPS of the RSU at the intersection illustrated in Figure 7, the location and position of the camera attached to the RSU should be configured to capture the traffic status for the intersection. As camera objects created by the Unreal/CARLA environment should be tied with certain *actors*, this default rule cannot meet

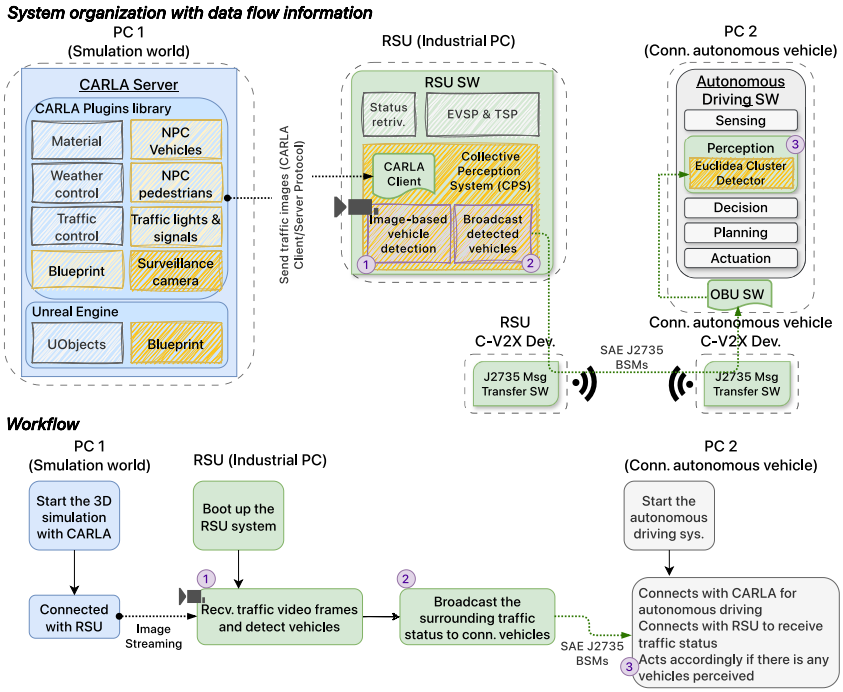


FIGURE 8. The system organization (top) and workflow (bottom) of the hardware-in-the-loop simulation for the IMA application scenario illustrated in Figure 7. The data flow information and the important software modules involved with the IMA application are highlighted in yellow rectangles. The workflow reflects the procedures of the V2I system simulation based on the data flow information.

the requirement of the V2I application development. To tackle the problem, we have added a new object class `SurveillanceCamera` within the `Blueprint` of Unreal Engine, which is the class name of the object factory (template) to create new object instances, so that an RSU-camera can be created and placed in arbitrary location in the simulated environment for runtime traffic monitoring. This object class is also exposed to users as ordinary *actors* in the CARLA plugin library, and the CARLA client at the RSU can retrieve the image streaming during the V2I application simulation for the image-based vehicle detection, as illustrated in the CPS of the RSU in Figure 8. The relevant enhancements made are represented as the dark yellow rectangles within the CARLA server of PC 1 in Figure 8.

- As a system prototype for the IMA application, in order to "perceive" the surrounding vehicles (recorded in the received BSMs) and to avoid collision, the coordinates of the vehicles (detected by the RSU) are converted into the coordinates recognized by the autonomous driving software, and the coordinates of these vehicles are "projected" into the coordinate system of the connected autonomous vehicle. This is achieved by injecting the decoded BSM data (the orientation and the coordinates of a detected vehicle) into the data flow of the autonomous driving software, so that the collision avoidance can be achieved by the obstacle-avoidance algorithm built in the autonomous driving software. In this work, Autoware is chosen as the autonomous driving software and the detected vehicle information is added to the Euclidean Cluster Detector (ECD) module that is represented as the dark yellow rectangle within the autonomous driving software in PC 2 and is originally designed to identify obstacles in the *point cloud* data provided by the Lidar sensor. In other words, the vehicles detected by the RSU are *added* as obstacles,[3] perceived by the ECD module. These detected vehicles are then fed into the Object Collision Estimator module to generate the routing path to avoid any potential collisions. Therefore, for example, the collision avoidance for the white car in Figure 7 can be accomplished as if Autoware can actively detect the white car by its Lidar sensor. The relevant results are further provided in Section V.

V. PERFORMANCE EVALUATION

The V2I IMA application introduced in Section IV is served as the case study to demonstrate the effectiveness of our proposed framework, assisting the system development of a V2I application system. To this end, a prototyping system with the hardware-in-the-loop simulation is established based on the description introduced in Section IV, where the connected autonomous vehicle drives at the speed of 30 KM/h and it is 60 KM/h for the non-connected vehicle. Three sets of experiments have been conducted and their results are

[3]In our experiment, the detected vehicles are added as objects to `/perception/object_recognition/objects` in Autoware.

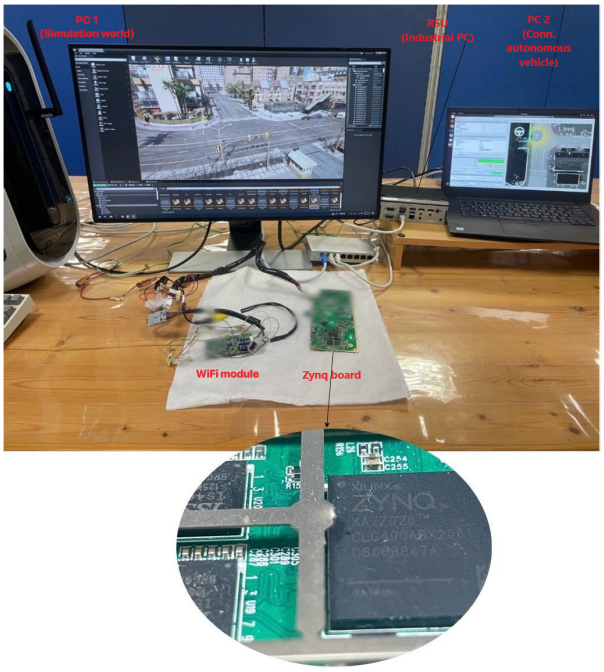
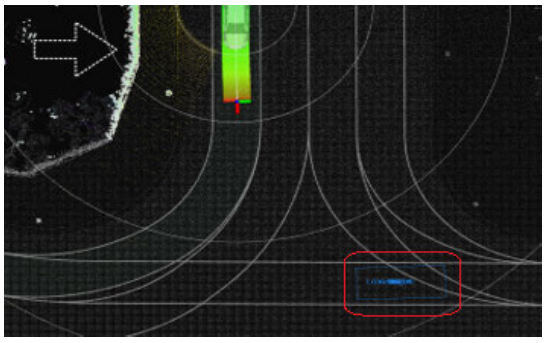


FIGURE 9. The actual hardware of the prototyping system for the IMA application illustrated in Figure 8. PC 1 (left), RSU (middle), and PC 2 (right) are connected by the Gigabit Ethernet network. The FPGA board (bottom) connects with PC 1 through the WiFi module on its left to transfer its inputs and outputs.

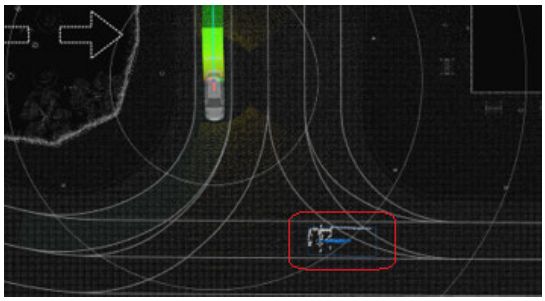
presented in the following three subsections. The functional correctness is validated and presented in Section V-A. The delivered performance of the prototyping system, in terms of the end-to-end latency, is Section V-B. In addition, based on the delivered performance in Section V-B, Section V-C further explores the performance of the various hardware designs for the CPS of the RSU. This design space exploration example demonstrates the advantage of our proposed framework. This is achieved by evaluating the performance of the image-based vehicle detection function (as illustrated in Figure 8) handled by different computation engines, i.e., CPU, GPU, and FPGA.

A prototyping system is built based on the system organization presented in Figure 8. The picture of the actual hardware is displayed in Figure 9, and the hardware and software specifications of the system are listed in Table 1. It is worth noting that the YOLO software is used to handle the image-based vehicle detection function within the CPS on the RSU (running on the CPU, GPU, and FPGA). When the CPU and GPU are used as the computation engine, the pre-trained model `YOLOv7` is adopted[4] for the vehicle detection. When the FPGA board is used to perform the detection task, the pre-trained model `YOLOv3-tiny` is served as the reference for developing our FPGA implementation, where the input image frame is set to the default resolution accepted by YOLOv3, 416×416.

[4]The official CPU and GPU implementations of `YOLOv7` are used in our experiments with the default configurations (i.e., `-config 0.25, -img-size 640`).



(a) The perception result for a non-connected vehicle reported by BSMs.



(b) The perception result for a non-connected vehicle scanned by Lidar sensor.

FIGURE 10. The screenshots of the perception results reported by Autoware to validate the detected vehicle (sent by BSMs from RSU) can be perceived by Autoware.

A. FUNCTIONAL CORRECTNESS VALIDATION

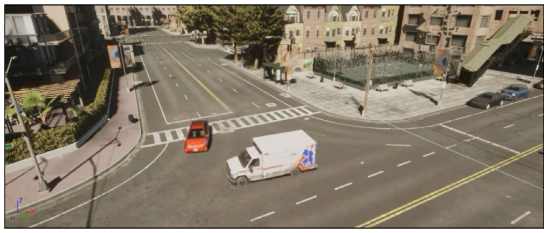
Two sets of experiments have been done to demonstrate the functional correctness of the prototyping system. First, the visual perception result provided by Autoware[5] is adopted to report the status of the non-connected vehicle (the ambulance in the IMA application illustrated in Figure 7). Figure 10a shows that the information of the detected vehicle can be fed to Autoware successfully, where the non-connected vehicle is able to appear as the scanned obstacle at the right-bottom corner (highlighted by the red rectangle). In order to confirm our result, the obstacle detected by the Lidar sensor equipped in the connected autonomous vehicle is also presented in Figure 10b, which suggests that the information of the detected vehicle (for the non-connected vehicle) can be presented in the Autoware system for obstacle avoidance.

Second, the visual output of the CARLA simulator further demonstrates the functional correctness of the IMA application. Figure 11a indicates that the connected autonomous vehicle is able to detect the non-connected vehicle via the broadcasted BSMs, and can react (slow down its speed) to avoid the side collision. On the other hand, without the CPS support, the connected autonomous vehicle cannot react in time, leading to the side collision (Figure 11b).

[5]In Autoware, a GUI interface is provided for users to observe the results during the software execution. In this case, we turn on the visualization window to observe the route planning and the perceived data.

TABLE 1. Hardware and software specifications of the system in Figure 8.

	Simulated world	RSU	Connected autonomous vehicle
CPU	Intel i7-12700KF	Intel i7-12700KF	Intel i7-9850H
GPU	NVIDIA RTX-3070		NVIDIA RTX-1650
DRAM	DDR5 32 GB	DDR4 4 GB	DDR4 16 GB
SSD	1TB	1TB	1TB
V2X library		Unex J2735 impl.	Unex J2735 impl.
V2X dev.		Unex C-V2X	Unex C-V2X
Comp. networking	Gigabit Ethernet	Gigabit Ethernet	
OS	Ubuntu 18.04	Ubuntu 20.04	Ubuntu 20.04
Ctrl. SW	CARLA ver. 0.9.13	RSU SW [19] & YOLO [19]	Autoware.Universe Galactic



(a) With CPS, the connected autonomous vehicle can slow down to avoid collision.

(b) Without CPS, the connected autonomous vehicle collides the other vehicle.

FIGURE 11. The screenshots (of the simulated world) for the IMA application illustrated in Figure 7 with and without the CPS support.

B. END TO END LATENCY ESTIMATION

Theoretically, one of the end to end latencies for the IMA application starts from the CARLA simulator feeding the image frames to the RSU and ends with Autoware *perceiving* the environmental status and making the control decisions. Considering the facts that 1) the latency of feeding the image frames to the RSU is actually not taken place in the physical system, and 2) the data of the IMA application flow through various software components across different hardware devices, the end to end latency is estimated by decomposing the overall latency into three parts (① , ② , and ③ as introduced in Figure 8) and calculating the summation of the three parts. The first part performs the image-based vehicle detection, the second part is encapsulating the information of the detected into BSMs that are further transferred to the connected autonomous vehicle, and the final part is receiving/injecting the BSMs into the perception stage of Autoware that runs the routing planning avoiding the detected obstacles, if any. The average time of each part for handling the vehicle detection and the relevant operations is 3.6 s for ① , 8.1 ms for ② , and 19 ms for ③ , leading to 3627 ms for the end to end latency. It is obvious that the first part, which uses YOLO v7 for detecting vehicles with the CPU, is the performance bottleneck, and we decide to explore various designs/implementations for the image-based vehicle detection in the following subsection. As for the second part, the average data transmission latency satisfies the performance requirement of the J2945 standard that demands the highest message broadcast frequency of 10 Hz. The third part seems to be a reasonable time for controlling the autonomous vehicle at the speed of 30 KM/h. Based on our further analysis, the injection of the information encapsulated in BSMs into the Autoware dataflow has a little influence on the overall processing time because the rate of incoming BSMs is low (10 Hz).

C. RSU-CPS HW DESIGN EXPLORATION

As the target IMA application is based on the J2735 BSM for sharing traffic information, the performance index for the prototyping system can be set at the processing rate to fit the BSM frequency requirement (10 Hz). Apparently, the reported processing speed of the image-based vehicle detection module (based on YOLOv7 software ① running on the eight-core CPU) is of 0.27 Hz, which is far less than the target rate. Thus, two additional platforms (GPU and FPGA [28]) are adopted for the performance testing for the vehicle detection, as listed in Table 2, where the frame per second (FPS) is used to measure the processing speed. The delivered performance is tested under the same workload that is a video clip pre-recorded by the CARLA simulator to emulate the output of the RSU-camera, which is the input for the image-based vehicle detection module of the RSU-CPS shown in Figure 8. Specifically, the three hardware platforms take the same inputs for the vehicle detection task and outputs the detected vehicles, which are further converted into the BSM format to represent the traffic information, as illustrated in Figure 12.

The performance results listed in Table 2 indicate that the GPU and FPGA platforms can meet the performance requirement. Nevertheless, while the FPGA platform can perform the vehicle detection at 10 FPS (which is handled by the programmable logic solely, operating at the frequency of 33.3 MHz) at the lowest hardware cost, it requires an extensive amount of engineering work (i.e., human resources) to migrate the software implementation of the object

TABLE 2. Performance and cost.

HW platform	Item	Attribute	Utilization	FPS(Yolo)	Cost ($)
CPU platform	CPU	Intel i7-12700KF (8 cores) 3 GHz	100%	0.27	$277.99
	RAM	4GB			
GPU platform	CPU	Intel i7-9850H 2.60GHz	17.2%	30	$3,079
	RAM	16GB			
	GPU	GTX 1650	90%		
	GPU Memory	3GB			
FPGA platform (ZYNQ 7020)	Processor system				$157.3
	CPU	ARM Cortex A9, dual-core 667MHz			
	On-chip meoroy	256KB			
	Programmable logic	16GB			
	Look-up Tables (LUTs)	53,200		10	
	Total block RAM	1.8 Mb			

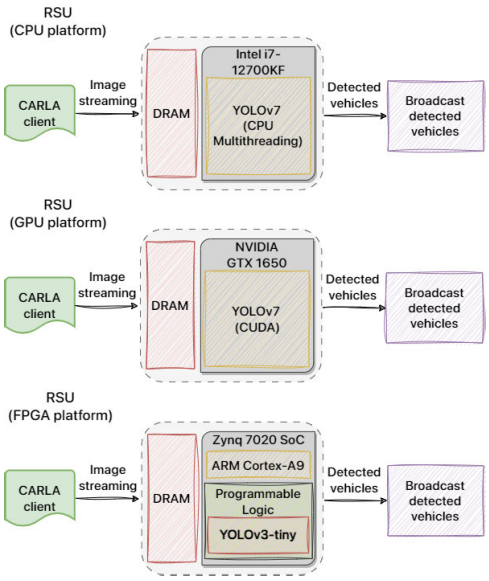


FIGURE 12. The high-level view of the three different hardware implementations for the image-based vehicle detection module within the RSU-CPS. This is a zoom-in view of the RSU-CPS illustrated in Figure 8.

detection to the corresponding hardware counterpart on the programmable logic, and if new application features are demanded, the enhancement would further require additional time for the feature development.

In particular, this FPGA implementation is realized by converting the software implementation (the `YOLOv3-tiny` from Darknet) into the corresponding hardware design on the programmable logic of Zynq 7020. This conversion is done manually on top of the software platform of Xilinx Vivado 2019.1 and DNNDK v3.1. The input and output of the vehicle detection hardware are orchestrated by the data movement hardware module implemented on the programmable logic of the Zynq SoC. To accelerate the communication performance, the data movement from the DRAM of the Zynq board to the local memory of the FPGA hardware is done over the AXI interface. The Cortex-A9 processor running on top of the PetaLinux 2019.1 is used to manage the FPGA hardware, e.g., initializing hardware and connecting with external devices to acquire input data and to transfer the output data. In current implementation, a straightforward mechanism is used to convert the software implementation into the FPGA counterpart, which utilizes approximately 96% of the available LUTs; this means that 51,320 out of 53,200 LUTs are used for the vehicle detection task and relevant operations. Further improvement is possible to optimize the implementation of the vehicle detection task, which can significantly reduce the LUTs utilization and increase the operating frequency.

Conversely, the GPU platform requires a higher hardware cost, but it presents a better performance result (30 FPS) with the software solution for the vehicle detection. It is important to note that a lower-end hardware platform (e.g., an Intel Atom processor with a built-in GPU support or an ARM Cortex processor with a 256-core Maxwell GPU from Tegra X1) could be further adopted to lower the hardware cost while satisfying the performance goal. It is also worth noting that the processing load for encapsulating and broadcasting the BSMs is low, and the main processors of the above hardware platforms can generate and transfer the BSMs at the fixed frequency easily.

VI. CONCLUSION

This work presents a software framework to develop V2I applications based on the open-source CARLA simulator. The advantage of the proposed approach is that it enables the full stack simulations for each involved RSUs and connected autonomous vehicles. The case study targeting the IMA application is used to showcase the capability of the proposed framework, and a system prototype has been built for the case study. Our experimental results show the framework is able to model the IMA application scenario with correct results. Furthermore, the end to end latency of the IMA application can be obtained via the prototyping system, where the performance bottleneck at the RSU-CPS is identified. Different hardware designs to implement the RSU-CPS have been explored, and the performance of the CPS has been tested on the hardware platforms to evaluate various hardware cost-performance alternatives.

REFERENCES

[1] K. Abboud, H. A. Omar, and W. Zhuang, "Interworking of DSRC and cellular network technologies for V2X communications: A survey," *IEEE Trans. Veh. Technol.*, vol. 65, no. 12, pp. 9457–9470, Dec. 2016.

[2] Z. Hameed Mir and F. Filali, "C-ITS applications, use cases and requirements for V2X communication systems—Threading through IEEE 802.11p to 5G," in *Towards a Wireless Connected World: Achievements and New Technologies*. Cham, Switzerland: Springer, 2022, pp. 261–285.
[3] B. Roberto, B. Bert, D. Matthias, F. Andreas, F. Walter, K. C. Christopher, K. Timo, K. Andras, L. Massimiliano, M. Cornelius, P. Timo, R. Matthias, S. Dieter, S. Markus, S. Hannes, V. Hans-Jörg, W. Benjamin, and Z. Wenhui. (2023). *Car-2-Car Communication Consortium-Manifesto*. Accessed: Oct. 6, 2023. [Online]. Available: https://www.car-2-car.org/
[4] (2023). *ETSI TR 102 638: Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions*. Accessed: Oct. 6, 2023. [Online]. Available: https://www.en-standard.eu/etsi-tr-102-638-v1-1-1-intelligent-transport-systems-its-vehicular-communications-basic-set-of-applications-definitions/
[5] (2023). *PTV Vissim*. Accessed: Oct. 6, 2023. [Online]. Available: https://www.ptvgroup.com/en/products/ptv-vissim
[6] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic traffic simulation using SUMO," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2018, pp. 2575–2582.
[7] L. Smith, R. Beckman, and K. Baggerly. (2023). *Transims: Transportation Analysis and Simulation System*. [Online]. Available: https://www.osti.gov/biblio/88648
[8] C. Sommer, D. Eckhoff, A. Brummer, D. S. Buse, F. Hagenauer, S. Joerer, and M. Segata, "Veins: The open source vehicular network simulation framework," in *Recent Advances in Network Simulation*. Cham, Switzerland: Springer, 2019, pp. 215–252.
[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," 2017, *arXiv:1711.03938*.
[10] J. B. Kenney, "Dedicated short-range communications (DSRC) standards in the United States," *Proc. IEEE*, vol. 99, no. 7, pp. 1162–1182, Jul. 2011.
[11] S. Chen, J. Hu, Y. Shi, L. Zhao, and W. Li, "A vision of C-V2X: Technologies, field testing, and challenges with Chinese development," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 3872–3881, May 2020.
[12] SAE Int. (2023). *SAE J2735: V2X Communications Message Set Dictionary*. [Online]. Available: https://www.sae.org/standards/content/j2735_202007
[13] (2023). *SAE J2945/1: On-Board System Requirements for V2V Safety Communications*. Accessed: SAE International. [Online]. Available: https://www.sae.org/standards/content/j2945/1_202004/
[14] H.-S. Seo, D.-G. Noh, C.-J. Lee, and S.-S. Lee, "Design and implementation of intersection movement assistant applications using V2V communications," in *Proc. 5th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2013, pp. 49–50.
[15] R. Miucic, A. Sheikh, Z. Medenica, and R. Kunde, "V2X applications using collaborative perception," in *Proc. IEEE 88th Veh. Technol. Conf. (VTC-Fall)*, Aug. 2018, pp. 1–6.
[16] M. Tsukada, T. Oi, A. Ito, M. Hirata, and H. Esaki, "AutoC2X: Open-source software to realize V2X cooperative perception among autonomous vehicles," in *Proc. IEEE 92nd Veh. Technol. Conf. (VTC-Fall)*, Nov. 2020, pp. 1–6.
[17] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, Nov. 2015.
[18] T. Kitazato, M. Tsukada, H. Ochiai, and H. Esaki, "Proxy cooperative awareness message: An infrastructure-assisted V2V messaging," in *Proc. 9th Int. Conf. Mobile Comput. Ubiquitous Netw. (ICMU)*, Oct. 2016, pp. 1–6.
[19] S.-H. Wang, W.-L. Yiu, C.-H. Tu, D.-W. Chang, and W.-H. Lee, "A software framework of roadside units for traffic condition perception and broadcast," in *Proc. Conf. Res. Adapt. Convergent Syst.*, Oct. 2022, pp. 1–8.
[20] C.-H. Su. (2021). *Design and Implementation of Roadside Unit Middleware for Intelligent Transportation System Applications*. [Online]. Available: https://hdl.handle.net/11296/jt4bx7
[21] S. Coast. (2023). *OpenStreetMap*. [Online]. Available: https://www.openstreetmap.org/
[22] A. Horni, K. Nagel, and K. Axhausen, *Multi-Agent Transport Simulation MATSim*. London, U.K.: Ubiquity Press, 2016.
[23] G. Rong, B. Hyun Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, "LGSVL simulator: A high fidelity simulator for autonomous driving," 2020, *arXiv:2005.03778*.
[24] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," 2017, *arXiv:1705.05065*.
[25] R. Xu, Y. Guo, X. Han, X. Xia, H. Xiang, and J. Ma, "OpenCDA: An open cooperative driving automation framework integrated with co-simulation," in *Proc. IEEE Int. Intell. Transp. Syst. Conf. (ITSC)*, Sep. 2021, pp. 1155–1162.
[26] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, May 2009, vol. 3, no. 3, p. 5.
[27] H. K. Lo and H. F. Chow, "Adaptive traffic control system: Control strategy, prediction, resolution, and accuracy," *J. Adv. Transp.*, vol. 36, no. 3, pp. 323–347, Sep. 2002.
[28] Z. Li and J. Wang, "An improved algorithm for deep learning YOLO network based on Xilinx Zynq FPGA," in *Proc. Int. Conf. Culture-Oriented Sci. Technol. (ICCST)*, Oct. 2020, pp. 447–451.

CHUN-TING WU is currently pursuing the Ph.D. degree in computer science and information engineering with the National Cheng Kung University, Taiwan. His major research interests include autonomous driving and vehicle-to-everything technologies.

SHAO-HUA WANG is currently pursuing the Ph.D. degree in computer science and information engineering with the National Cheng Kung University, Taiwan. His research interests include system performance analysis, optimization, robot operating systems, autonomous driving, and vehicle-to-everything.

CHIA-HENG TU (Member, IEEE) has been an Associate Professor with the Department of Computer Science and Information Engineering (CSIE), National Cheng Kung University (NCKU), since 2021. Before joining NCKU CSIE in August 2016, he worked as Postdoctoral Researcher with the MEDIATEK-NTU Advanced Research Center, National Taiwan University (NTU), from 2015 to 2016, and as the Research and Development Manager with the Institute for Information Industry, from 2012 to 2015, where he did his Research and Development Substitute Services after he completed his Ph.D. training from NTU, in 2012. His research interest includes developing tools (e.g., computer architecture simulators and performance analyzers/optimizers) for designing/optimizing specialized computer systems.