electronics

MDPI

Article

Virtual Scenario Simulation and Modeling Framework in Autonomous Driving Simulators

Mingyun Wen [1], Jisun Park [1], Yunsick Sung [1], Yong Woon Park [2] and Kyungeun Cho [1,*]

[1] Department of Multimedia Engineering, Dongguk University-Seoul, 30, Pildong-ro 1-gil, Jung-gu, Seoul 04620, Korea; wmy_dongguk@dongguk.edu (M.W.); jisun@dongguk.edu (J.P.); sung@mme.dongguk.edu (Y.S.)

[2] Department of Autonomous Things Intelligence, Graduate School, Dongguk University-Seoul, 30, Pildong-ro 1-gil, Jung-gu, Seoul 04620, Korea; ywpark@dongguk.edu

* Correspondence: cke@dongguk.edu; Tel.: +82-02-2260-3834

Abstract: Recently, virtual environment-based techniques to train sensor-based autonomous driving models have been widely employed due to their efficiency. However, a simulated virtual environment is required to be highly similar to its real-world counterpart to ensure the applicability of such models to actual autonomous vehicles. Though advances in hardware and three-dimensional graphics engine technology have enabled the creation of realistic virtual driving environments, the myriad of scenarios occurring in the real world can only be simulated up to a limited extent. In this study, a scenario simulation and modeling framework that simulates the behavior of objects that may be encountered while driving is proposed to address this problem. This framework maximizes the number of scenarios, their types, and the driving experience in a virtual environment. Furthermore, a simulator was implemented and employed to evaluate the performance of the proposed framework.

Keywords: autonomous driving simulator; scenario simulation and modeling framework; simulated virtual environment

Citation: Wen, M.; Park, J.; Sung, Y.; Park, Y.W.; Cho, K. Virtual Scenario Simulation and Modeling Framework in Autonomous Driving Simulators. *Electronics* **2021**, *10*, 694. https://doi.org/10.3390/electronics10060694

Academic Editors: Dong Seog Han, Kalyana C. Veluvolu and Takeo Fujii

Received: 16 February 2021
Accepted: 12 March 2021
Published: 16 March 2021

1. Introduction

Due to the requirements of securing appropriate sites and equipment, as well as the influence of weather and other factors, simulation and modelling in real world environments is considerably expensive. Moreover, data collection is not safe in such environments either. For example, the training of an unmanned ground vehicle using learning-based algorithms requires the collection of a significant amount of data outdoors [1], which is constrained by weather, time, and considerations for the safety of other vehicles and pedestrians. To resolve such problems, simulation technology based on virtual environments has been widely employed in skill training [2–7]. Existing simulation platforms often provide scenarios created by users using a graphical user interface (GUI) following simple rules with the aim of gradually imparting the target knowledge to users. In recent years, the utilization of simulation techniques has become quite common in the field of autonomous driving. Besides saving time and labor expenses, it avoids the necessity of repeated deployment of experimental environments in the real world. However, to ensure the robust training and accurate evaluation of autonomous driving algorithms, deployed virtual environments should include realistic agents like vehicles, humans, and animals, in addition to fundamental driving environment conditions. Furthermore, they should be capable of generating a wide variety of natural scenarios—in addition to normal scenarios wherein all agents obey traffic rules, hazardous scenarios that may induce accidents should also be simulated. This is essential, as real-world observations confirm that pedestrians often cross roads in the absence of traffic lights or by disobeying them and that vehicles often drift to the wrong side of the road due to wheel slip or high speed.

In recent years, several simulators have been proposed for training and evaluating driving algorithms [6–11]. They operate by either creating scenarios following a fully

automatic mode or creating scenarios based on manually inserted scripts or GUIs. None of the existing simulators are capable of creating agents that can perform various actions and animations. Furthermore, hazardous scenarios are not automatically created in most simulators [6,8,9,11,12]; they require manual creation by users via GUIs or scripting. However, the creation of various custom scenarios using GUIs is time-consuming, and appropriate coding prowess is required by the user.

In this paper, we focus on virtual scenario simulation for autonomous driving simulators rather than developing technologies for autonomous vehicles or their physical dynamics. This paper proposes a virtual scenario simulation and modeling framework for autonomous driving simulators that naturally and automatically creates both normal and hazardous scenarios, as well as providing a facile GUI to non-developer users to edit scenarios before and during simulation and modify season, weather, and lighting conditions.

The contributions of this study can be summarized as follows:

(1) The proposed framework is capable of simulating normal scenarios involving various agents, such as vehicles, humans, and animals and various 3D models for each type of agent. Further, the simulated agents can perform various actions—for instance, vehicle and human agents can cross roads by following traffic lights in a city map, and animal agents can eat the grass on the side of the road in a mountain map. Such scenarios correspond to common real-world scenarios.

(2) It is also capable of simulating hazardous scenarios, wherein certain agents deliberately violate traffic rules that may induce accidents. Hazardous scenarios are executed after considering the location and direction of the autonomous vehicle in order to manifest them in front of it.

(3) The proposed framework exhibits three operational modes—normal mode, automatic event mode, and custom event mode. Additionally, the framework provides an interrupt event function that allows user to edit scenarios during the simulation of three modes. Users can be afforded varying degrees of control to interfere with the three modes. With these three modes, users can realize a variety of scenarios.

(4) The proposed framework is equipped with a facile GUI that can be used even by non-developer users to enable the design of specific scenarios.

The remainder of this paper is structured as follows. Related works on autonomous driving simulators are outlined in Section 2. The proposed scenario simulation framework of the virtual environment is introduced in Section 3. The experimental results and their analysis are presented in Section 4. Finally, the proposed framework is discussed in Section 5.

2. Related Works

Technology related to autonomous vehicles has become a popular research topic in recent years. It is common to evaluate such systems using simulators because of the time-consuming, expensive, and risky nature of real-world experimentation. Several commercial automotive simulators have been proposed over the years. Among them, PreScan [13], dSPACE [14], ANSYS [15], rFpro [16], Cognata [17], and Metamoto [18] do not allow for user customization to achieve specific targets because they are not open-source. To keep up with emerging trends, some commercial automotive simulators—such as CarSim [19] and CarMaker [20], which both allow users to create scenarios by GUI—have added support to autonomous vehicles. Additionally, CarSim also allows users to create scenarios by scripts. However, neither support automatically generating scenarios.

On the other hand, in the field of reinforcement learning, several open-source simulators are available. Gazbo [21] and IssacSim [22] are open-source simulators of reinforcement learning for autonomous robots. They enable virtual robots to learn multiple scenarios under various virtual environment conditions related to physics and dynamic models and can improve their motion planning.

Torcs [23] and Deepdriving [24] are two previously used simulators for the reinforcement learning for autonomous vehicles. Even though they enabled the learning of handling, accelerating, and braking to aid automated driving, they did not include any virtual sensors

during the learning process. Moreover, they only included vehicles as surrounding agents and excluded pedestrians. Additionally, they did not allow for surrounding vehicles to be edited to create customized scenarios according to the desires of users.

AirSim [8], CARLA [10], and LGSVL [11] are some other examples of simulators used for the reinforcement learning of autonomous vehicles. Each of the aforementioned simulators implements a red, green, blue and depth (RGB-D) camera, Inertial Measurement Unit (IMU)/GPS, and lidar sensors, and each allows its users to edit environmental conditions, such as the map, time, and weather. RGB-D indicates the red, green, blue and depth, and RGB-D camera indicates the sensor which captures RGB images and depth information. IMU is a sensor that measures force, angular rate, etc. of the body which mounts the sensor. Additionally, users can edit specific scenarios using C++ or Python scripts to enable the model to learn and test an environment including pedestrians and surrounding vehicles in CARLA and LGSVL. However, an intimate understanding of coding languages is required by the users, making scenario creation difficult for non-developers. Apollo [9] is a simulator used to train autonomous vehicles that includes virtual sensors, such as RGB-D cameras, IMU, GPS, lidar, and radar. It is equipped with facile GUI-based tools to allow users to create and edit scenarios—map, time, and weather—according to their desires. Table 1 summarizes existing simulators and their characteristics. In the last column in Table 1, 'AN,' 'AH,' 'C,' and 'I' stand for automatic normal scenario, automatic hazardous scenario, custom scenario, and interrupt scenario, respectively. AirSim, LGSVL, and Apollo only simulate common scenarios where no accidents occur. CARLA considers scenarios that could lead to accidents, but the type is limited to only one type of crosswalk. In addition, existing simulators do not provide a function to diversify the scenario by interrupting the ongoing situation.

Table 1. Existing simulators. AN: automatic normal scenario; AH: automatic hazardous scenario; C: custom scenario; I: interrupt scenario; GUI: graphical user interface.

Simulators	Learning Target	Virtual Sensors	Scenario Functions	Features of Scenarios
gym-gazebo2	Robot	RGB-D camera, GPS, IMU, lidar, radar, and torque sensor	-	- Users can modify physics and lighting.
IssacSim	Robot	RGB-D camera, lidar, and IMU	-	
Torcs	Vehicle	RGB camera	-	- Scenarios include vehicle agents.
Deepdriving	Vehicle	RGB camera	-	
AirSim	Vehicle, drone	RGB-D camera, IMU, GPS, and lidar	AN	- Users can modify time of day, weather, and wind conditions. - Scenarios include vehicle, human, and animal agents. - The system creates scenarios in an automatic way.
CARLA	Vehicle, drone	RGB-D camera, IMU, GPS, and lidar	AN, AH, and C	- Users can modify map, weather, and lighting conditions. - Scenarios include vehicle and human agents. - Users can create specific scenarios using Python script.
LGSVL	Vehicle	RGB-D camera, IMU, GPS, lidar, radar, and ultrasonic	AN and C	
Apollo	Vehicle	RGB-D camera, IMU, GPS, lidar, and radar	AN and C	- Users can modify map, weather and lighting conditions. - Scenarios include vehicle and human agents.
CarSim	Vehicle	RGB camera, lidar, radar, and ultrasonic	C	- Users can modify map, weather, and lighting conditions. - Scenarios include vehicle, human and animal agents. - Users can create specific scenarios by GUI.
CarMarker	Vehicle	RGB camera, lidar, radar, and ultrasonic	C	
Ours	Vehicle	RGB camera, lidar, and radar	AN, AH, C, and I	- Users can create specific scenarios using GUI. - Users can modify season, weather, and lighting conditions. - Scenarios include vehicle, human, and animal agents.

The behaviors of all agents in normal scenarios simulated by the aforementioned simulators are hard-coded. In contrast, the agents in the framework proposed in this study are driven by motivation and mimic the behavior of humans in real life (i.e., as deduced based on big social data). Additionally, this framework enables the learning and testing of automatically generated hazardous scenarios manifested in front of a vehicle. Further, users can customize scenarios using a facile GUI both before and during simulation.

3. The Proposed Scenario Simulation Framework

3.1. Overview of the Framework

Our goal was to provide a framework for scenario creation in autonomous driving simulators. It should allow for the automatic creation of various normal scenarios for autonomous vehicles to learn how to drive in daily scenarios, as well as hazardous scenarios for autonomous vehicles to learn how to deal with unexpected dangerous scenarios. It should also provide a GUI to allow users to create custom scenarios to enable autonomous vehicles to learn to handle certain specific scenarios, as well as to check learning performance by inserting temporary test scenarios during autonomous vehicle training. To this end, we designed the following framework, which consists of three modules, as shown in Figure 1: (1) a scenario editor module that provides users with a GUI, updates the status of agents, and visualizes the simulation results; (2) a virtual autonomous vehicle module that simulates the dynamic physical model of an autonomous vehicle and the various sensors mounted on it; and (3) an artificial intelligence (AI) module that controls the logic of agents in the virtual environment.

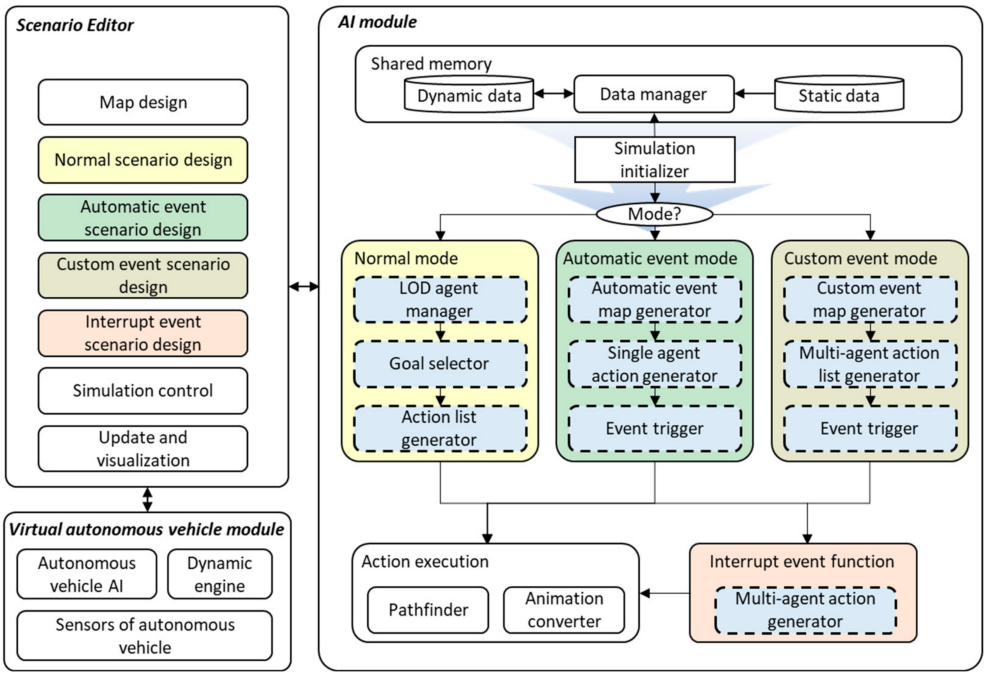

Figure 1. Architecture of the proposed scenario simulation framework. AI: artificial intelligence.

The scenario editor module is connected to the virtual autonomous vehicle module, and the AI module is connected via the network. The GUI in the scenario editor module is the primary interface used by users to configure and design simulations. The AI module receives the details of the configuration parameters from the scenario editor module and initializes the framework using the simulation initializer.

Dynamic and static data are stored in the shared memory, and they are transmitted to other components of the AI module via a data manager. Dynamic data comprise configuration messages and the status of agents received from the editing and visualization module. Static data comprise map data, such as road architecture, locations, and sizes of buildings. Three operational modes are implemented to control agents during scenario simulation—normal mode, automatic event mode, and custom event mode. In normal mode, normal scenarios in which human and animal agents are driven by motivation and all agents follow traffic rules are simulated. In automatic event mode, hazardous scenarios are simulated by motivating agents to deliberately break traffic rules. The custom event mode allows users to design events using the GUI to compose customized scenarios. Additionally, an interrupt event function is provided. The custom event mode is different

from the interrupt event function. The former is used to customize scenarios prior to the initialization of the simulation, whereas the latter is used to alter or introduce events during the simulation. All modes generate actions among agents and transmit them to the action execution module, where the pathfinder schedules the path of the agents and the animation converter converts actions to animations. Three types of agents can be simulated by this framework—vehicles, humans, and animals.

3.2. Normal Mode

Many simulators provide scenarios that involve agents performing normal actions [8,9,11], in which the human or animal agents mostly walk or idle by the side of road. This looks unnatural because in the real world, human and animal can perform rich actions driven by motivations. The involved human and animal agents of scenarios in our normal mode can perform more various actions driven by a novel motivation system [25,26]. In normal mode, the framework simulates a limited variety of popular scenarios involving vehicles, humans, and animals that are usually observed in real-world streets. The procedure followed during operation in the normal mode is shown in Figure 2. It can be divided into two stages—the configuration stage in the scenario editor and the execution stage in the AI module.

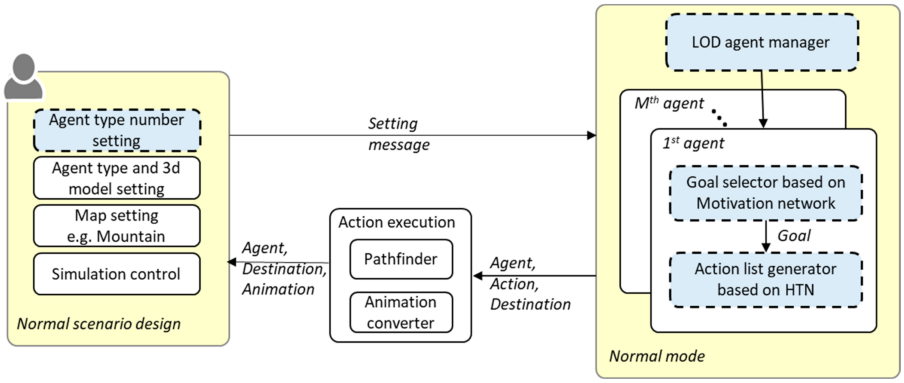

Figure 2. Overview of the normal mode. LOD: level of detail; HTN: hierarchical task planning.

During the configuration stage, users can choose the map type, agent types, 3D model types of each agent, and the number of agents of each type in scenario editor. Subsequently, the AI module generates agents that satisfy the chosen conditions once the simulation is initiated. During the AI module processing stage, the level of detail (LOD) agent manager receives the configuration information to generate the agents within a specified spawn area and destroys agents that stray outside it. Then, a goal is selected by the motivation network, and action lists are generated via hierarchical task planning (HTN) for all human and animal agents [25,26]. Finally, the action is transmitted to be executed by the action execution module, during which stage the associated animation is ascertained and the relevant path is identified based on destination by the pathfinder module.

3.2.1. Agent Management Based on LOD

The LOD is a technique that reduces the rendering workload during the graphical pipeline stage by assigning less detail to models that are farther from the virtual camera. The generation of agents surrounding the autonomous vehicle is essential to the simulation of driving scenarios in virtual environments. To ensure the smooth operation of the framework even on large maps, agents are only simulated within a certain radius of the autonomous vehicle. This area is termed the LOD area, and it coincides with the effect of the LOD. To avoid the sudden appearance of agents within the view of a wired autonomous vehicle in the real world, we subdivide the LOD area into two regions, as depicted in Figure 3. In the figure, O denotes the location of the autonomous vehicle, l_1 denotes the radius defining the LOD area, and l_2 denotes the radius of the region visible to the autonomous vehicle,

highlighted in gray. No new agents are generated inside the latter subregion. The region highlighted in green denotes the spawn area where new agents are generated.

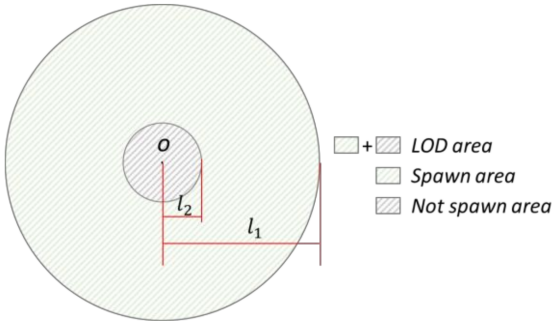

Figure 3. LOD area. *O*: the location of the autonomous vehicle; l_1: the radius defining the LOD area; l_2: the radius of the region visible to the autonomous vehicle, highlighted in gray.

Let a denote the set of identification numbers (IDs) of existing human agents in the framework and I be the ID repository of each agent type. The IDs of agents within a limited range need to be controlled because agents are frequently destroyed and spawned during simulation. Further, assume that the location of the autonomous vehicle is denoted by p_{ego}. Algorithm 1 describes the procedure to destroy human agents outside the LOD area. a' denotes the human agents to be destroyed. Once the current frame is processed in the AI module, it is transmitted to the editing and visualization module to destroy the constituent avatars. First, the distance of each agent from the autonomous vehicle is calculated and those that lie outside the LOD area are identified (lines 4–5). The IDs of agents leaving the LOD area are added to the agent ID repository and destroyed agent list (line 6). Following that, the existing agent list, a, is updated by removing the agents appearing in a' (line 8). The procedures to destroy animal and vehicle agents are identical to that of human agents.

Algorithm 1 Destroying agents that are out of the LOD area

Input: I, a, l_1, p_{ego}
Output: a, a', I
1: **procedure** DestroyAgents
2: $a' \leftarrow \emptyset$
3: **for** i: = 0 $\rightarrow$ $len(a)$ **do**
4: $p \leftarrow getLocation(a_i)$
5: **if** $dist(p, p_{ego}) > l_1$ **then**
6: $\boldsymbol{I}.append\ (a_i)$
7: $\boldsymbol{a'}.insert(a_i)$
8: $\boldsymbol{a}.remove(\boldsymbol{a'})$

Let $\boldsymbol{m}$ denote the set of available 3D models of agent type t, l_2 denote the radius of the gray area within the LOD area (as depicted in Figure 3), n denote the number of agents assigned to agent type t by the user, and a_{new} denote the generation data of newly generated agents that are transmitted to the editing and visualization module once each frame is processed. The newly generated agents are available to the AI module in the following frame. Algorithm 2 describes the procedure to spawn an agent of type t. The AI module attempts to keep the number of each agent type identical to the number configured by the user before initiating the simulation. Therefore, the number of current agents is always verified before spawning a new agent (line 3). When the number of agents is less than the configured number, the spawning algorithm generates a list of possible spawning locations within the spawn area (line 4). As long as the number of existing and newly generated agents remains below n, the spawn algorithm continuously generates new agents (lines 5–12). To achieve this, first, a location is randomly sampled from *spawnLocationList* and removed from *spawnLocationList* (lines 6–7). The first ID of the ID repository, I, is assigned to each new agent. Then, the remaining data required to generate the new agent are generated by the *generateAgent* function (line 10), which accepts the new agent ID, location, available 3D models for agent type t, and agent type t and then

outputs *newAgentData*. *newAgentData* is a data structure that includes agent ID, 3D agent model ID, agent generation location, and agent generation direction. The existing agent ID, a, is updated with the newly generated agent ID (line 9).

Algorithm 2 Spawning agents

Input: $I, a, m, t, l_1, l_2, n, p_{ego}$
Output: a, a_{new}, I
1: **procedure** SpawnAgents
2: $a_{new} \leftarrow \varnothing$
3: **if** $len(a) < n$ **then**
4: $spawnLocationList \leftarrow getSpawnLocationlist(p_{ego}, l_1, l_2, t)$
5: while $len(a) < n$ do
6: $p \leftarrow random(spawnLocationList)$
7: $spawnLocationList.remove(p)$
8: $a \leftarrow I.pop()$
9: $a.append(a)$
10: $newAgentData \leftarrow generateAgent(a, p, m, t)$
11: $a_{new}.append(newAgentData)$
12: $n \leftarrow n + 1$

In the normal mode, agents can only be generated within the spawn area highlighted in green in Figure 3. Human and animal agents are generated on the sidewalk in a city map or on the side of the road on a mountain map. To this end, we leverage a grid map of the terrain to identify the side of the road or the sidewalk. The size of the grid map denotes the size of the terrain of the virtual environment. First, the grid map is evenly divided into grid cells. Each cell is labeled using an ID representing the terrain type of the corresponding area in the virtual environment. The terrain type is detected by ray casting from a point above the corresponding location of the map downwards onto the map.

On the other hand, vehicles are generated on roads within the spawn area. A road grid map of identical resolution and size to the grid map of the terrain is generated in advance. By greedily calculating the distance between pairs of centers of grid cells and roads, each grid cell is assigned the ID of the closest road. The radius of the LOD and the location of the autonomous vehicle together uniquely determine the grid cells within the LOD area. The grid cells within the gray area depicted in Figure 3 are also determined by the corresponding radius. Finally, the IDs of roads that lie within the spawn area are retrieved by identifying the unique IDs of roads that lie within the LOD area but outside the gray area depicted in Figure 3.

3.2.2. Scenario Simulation in Normal Mode

The actions of human and animal agents in a scenario simulated in the normal mode are driven by their respective goals. These goals are assigned by the goal selector, as indicated in Figure 2, based on the motivation network. To generate the goal of an agent, its motivation is first selected and then mapped to a goal. The motivations and goals of humans are extracted from big social data [25], while those of animals are manually defined [26]. To satisfy each goal, an action list is generated by an action list generator using an HTN [26]. Each action on the action list is subsequently executed using the action execution function. However, the corresponding case for vehicle agent simulation is different due to the absence of motivations. To simulate various types of driving styles in the virtual system, we generated driving style data based on real driving data collected from real drivers. Driving speed, acceleration, deceleration, and steering data, which can reflect the driving style of any random driver, were collected. Following the collection of real driver data, they were clustered into several groups via affinity propagation. Each group represents a distinct driving style. Finally, different attributes of the driving data in one cluster were combined, and the driving data were augmented [27]. Vehicles in a virtual environment may exhibit two actions—driving and stopping. The destination of each vehicle on the road is decided beforehand. Whenever a vehicle arrives at its destination, a new destination is selected. The breadth-first search method (BFS) is utilized to identify the paths of vehicles on the road network. Vehicles in a virtual environment are assumed to be

compliant with traffic rules. The framework simulates traffic lights, whose state transitions are based on predefined state sequences and the duration of each state.

3.3. Automatic Event Mode

Automatic event mode is implemented to create hazardous scenarios based on real-life observations of pedestrians crossing the road in violation of traffic rules and vehicles driving on the wrong side of the road. This mode helps to improve the robustness of the driving model for autonomous vehicles by generating hazardous events in front of autonomous vehicles on roads. The procedure of the automatic event mode is depicted in Figure 4. As in the case of the normal mode, the user selects the map, agent type, and 3D agent models to initiate the simulation. However, the number of agents cannot be pre-configured because it is dependent on the event at each event location. An automatic event map generator adopts BFS to traverse the entire road network and sample event generation locations at intervals of fixed distance defined by the user [28]. The initial point of the traversal is randomly determined, thus ensuring that different automatic event maps can be implemented in different simulations. One event can be generated at each event generation location when it is triggered. Then, the action is transmitted to the action execution function.

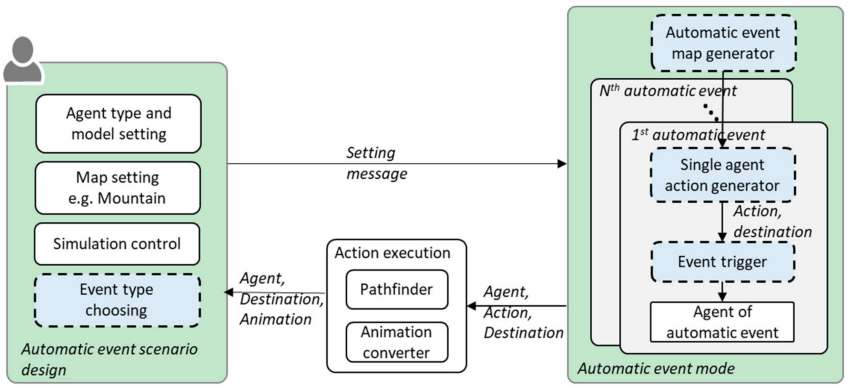

Figure 4. Overview of automatic event mode.

Each event in the automatic event mode involves only one agent [28]. Each agent is assigned a key action, which violates traffic rules or disturbs an autonomous vehicle, by a single agent action generator. Table 2 presents the complete list of hazardous events. The first type of events corresponds to an agent standing or stopping on the road in front of an autonomous vehicle. The 'crossing road' event represents a situation in which a human or animal agent crosses a road in front of an autonomous vehicle. The 'driving in front of an autonomous vehicle' event represents a vehicle driving slowly in front of an autonomous vehicle. The simulation of this scenario teaches the autonomous vehicle how to maintain an adequate distance from the vehicle in front of it. The 'driving on the wrong side of the road in front of an autonomous vehicle' event is simulated to teach the autonomous vehicle how to avoid collision with a vehicle in front of it. On a city map, events involving humans and animals may not be generated in some cases—for example, humans and animals are not likely to appear on highways. Therefore, the road type is considered by the single agent action generator. Further, the agent type can be pre-configured by the user using the GUI. For example, when the user sets the agent type to be solely human, only events involving humans are generated. In this case, no event is generated on the highway.

All automatic event maps and agent actions are generated during the initialization of the simulation. The action trigger is used during the simulation to execute the action of the relevant agent immediately before the arrival of the autonomous vehicle. Depending on the road network, thousands of event locations can be generated. Next, we describe an efficient method to identify appropriate event locations, which are defined to be locations that will be passed by the autonomous vehicle with very high probability. Multiple appropriate

event locations may be selected because when the autonomous vehicle approaches a junction, the framework cannot determine whether it will turn in either direction or continue straight ahead.

Table 2. Hazardous events.

Hazardous Event Type	Involved Agent Type
Blocking road	Vehicle, human, and animal
Crossing road (left to right)	Human and animal
Crossing road (right to left)	Human and animal
Driving in front of autonomous vehicle	Vehicle
Driving on the wrong side of the road in front of autonomous vehicle	Vehicle

The event map is saved in a dictionary, in which road IDs serve as keys and the lists of the IDs of events that can be generated on the corresponding roads serve as values. Let m denote the event map r_{ego} denote the ID of the road on which the autonomous vehicle is situated; p_{ego} and o_{ego} denote the autonomous vehicle's location and direction, respectively; and δ_1 and δ_2 denote the distance thresholds corresponding to which actions are prepared and executed, respectively. The road structure is taken to be identical to that described in [28]. Thus, we obtain an event ID list, m', following Algorithm 3. Then, the road grid map is leveraged to identify the road ID, r_{ego}, corresponding to the location of the autonomous vehicle and calculate its driving direction, *driverForward* (lines 3–6). Following that, the road list comprising the route of the autonomous vehicle is identified by searching the road network (lines 7–10). Subsequently, a list of event locations corresponding to any road list is identified (lines 11–13). The number of event locations is much smaller than the total number of event locations. Following the identification of an event location, the framework first generates an agent (line 16) and then verifies if the autonomous vehicle is farther than δ_2 from the event location. If it is, the idling action is assigned to the generated agent (line 20), which prevents the event from being finished before the autonomous vehicle is sufficiently close. Otherwise, the scheduled action is assigned to execute the event (lines 17–18).

Algorithm 3 Action trigger

Input: $m, r_{ego}, p_{ego}, o_{ego}, \delta_1, \delta_2$

1: **procedure** ActionTrigger
2: $\boldsymbol{m'} \leftarrow \varnothing$
3: **if** $angleBetween(getRoadDirection(r_{ego}), o_{ego}) \leq 90$ **then**
4: $driveForward \leftarrow True$
5: **Else**
6: $driveForward \leftarrow False$
7: **if** *driveForward* then
8: $\boldsymbol{r'} \leftarrow getRoadIDListAfterRoadInRange(r_{ego}, \delta_1)$
9: Else
10: $\boldsymbol{r'} \leftarrow getRoadIDListBeforeRoadInRange(r_{ego}, \delta_1)$
11: **while** $\boldsymbol{r'} \neq \varnothing$ **do**
12: $r \leftarrow Pop(r')$
13: $\boldsymbol{m'} \leftarrow \boldsymbol{m'} \cup getEventIDListByRoadID(r)$
14: $i \leftarrow 1$
15: **while** $i \leq length(\boldsymbol{m'})$ **do**
16: $a_i \leftarrow spawnAgentForEvent(m'_i)$
17: **if** $distance(getEventLocation(m'_i), p_{ego}) \leq \delta_2$ **then**
18: $executeEventAction(m'_i), a_i)$
19: **Else**
20: $executeEventIdlingAction(m'_i, a_i)$
21: $i \leftarrow i + 1$

3.4. Custom Event Mode

The custom event mode is implemented to enable users to create events to serve their requirements. It includes four components—a custom event map generator, a multi-agent action list generator, an action trigger, and an action execution function. Their relationship is depicted in Figure 5. First, the user configures the simulation parameters, including map configuration and custom event design. Then, the configuration message is transmitted to the AI module, and the custom event map is generated by the custom event map generator. This map is different from the one generated during the automatic event mode as the locations of events are specified by the user using the GUI in this case. However, the structure of the custom event map is similar to that of an automatic event map, and road IDs are continued to be used as keys to enable efficient the identification of appropriate event IDs during simulation. Users are allowed to create vehicle, human, and animal agents using the GUI and assign their 3D model types and action lists. Multiple agents can be included in one event, and multiple events can be included in one simulation, as depicted in Figure 5. The action trigger resembles that implemented in the automatic event mode, but it is equipped with a user-defined distance threshold that is defined using the GUI.

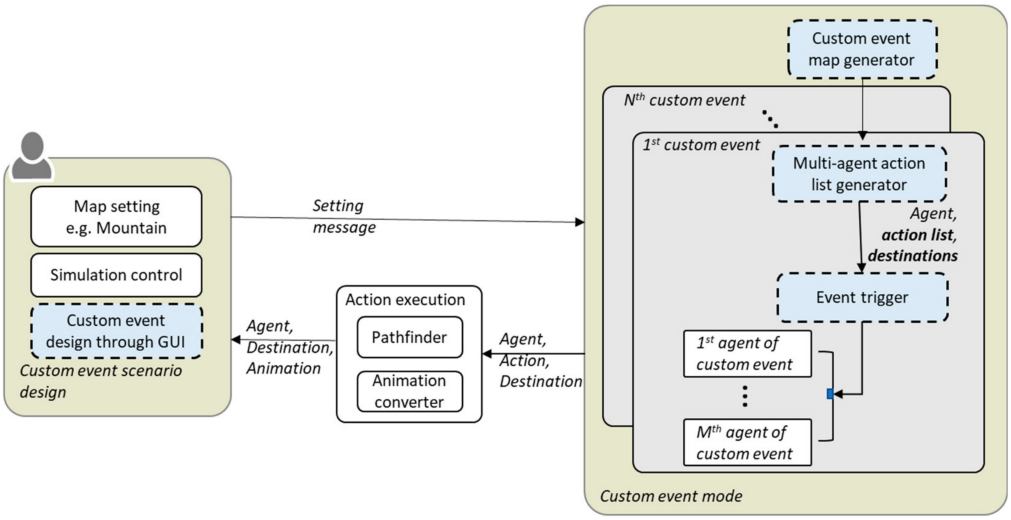

Figure 5. Overview of the custom event mode.

The configuration process of a custom event using a GUI is depicted in Figure 6. First, the user selects the map. After the map is loaded in the GUI, the user begins to create the events. Users can create multiple events within a single simulation. To create an event, the user first specifies the location of the event and then assigns an activation distance like δ_2 in the automatic event action trigger. When the distance between the autonomous vehicle and the location of the event is less than the defined threshold, the event is executed. To create an agent, the user adds it and then assigns its type, 3D model, and location. One or more actions are then added to the agent. To create an action, the user selects an animation type from a dropdown list. There are two types of available animations that are distinguished by whether or not they change the position of the agent—movable animation and non-movable animation. If the selected animation is movable, the destination is also required to be specified; otherwise, only the duration of the animation is configured by the user.

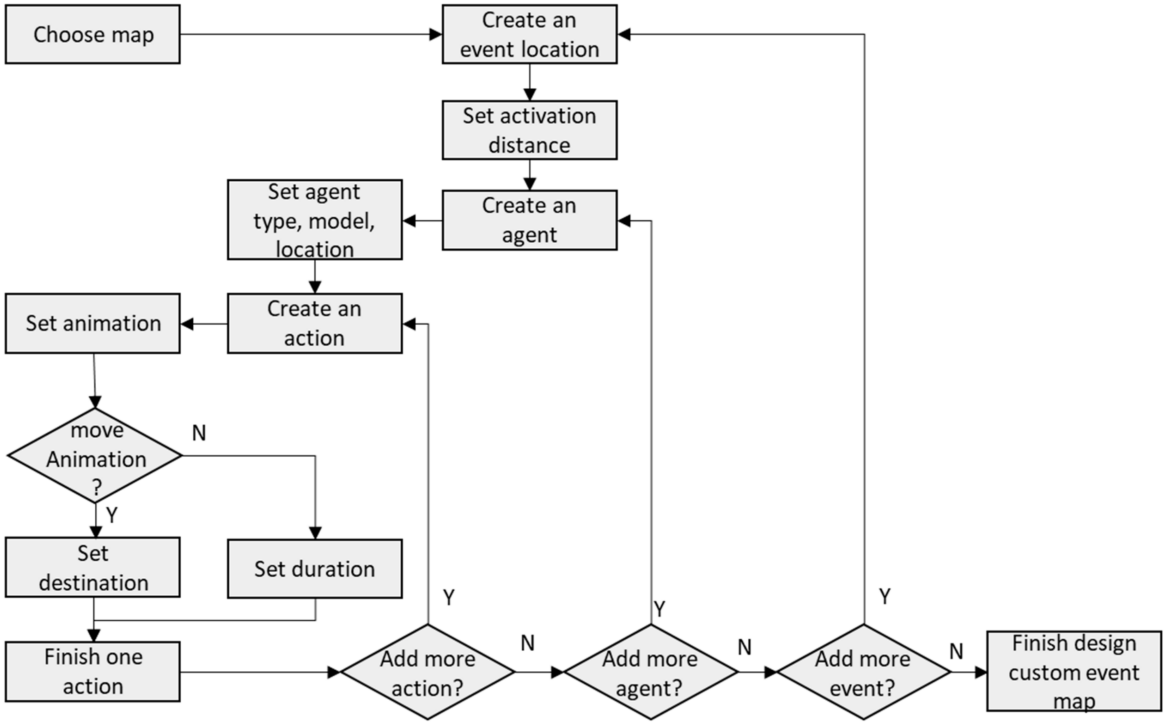

Figure 6. Configuration process of the custom event mode.

3.5. Interrupt Event Function

The interrupt event function differs from other modes, primarily in terms of its design stage. Its purpose is to support users' improvisation while the simulation is in progress by enabling the user to pause the simulation using the simulation control component in the GUI. The interrupt event function can simultaneously operate with other modes. Additionally, it can assign actions to existing agents in the framework, but this is only encouraged in conjunction with the normal mode because it adversely affects events in other modes. Furthermore, new agents may be created in this function. Figure 7 depicts the relationship between the other modes and the interrupt event function, as well as the overall procedure of the interrupt event function. The GUI in the interrupt event function is similar to that in the custom event mode, but multiple actions are not allowed to be assigned to an agent. It is designed to minimize the time required by the user to edit events during the pause. Events in the interrupt function are not equipped with action triggers and are directly executed after the simulation is resumed.

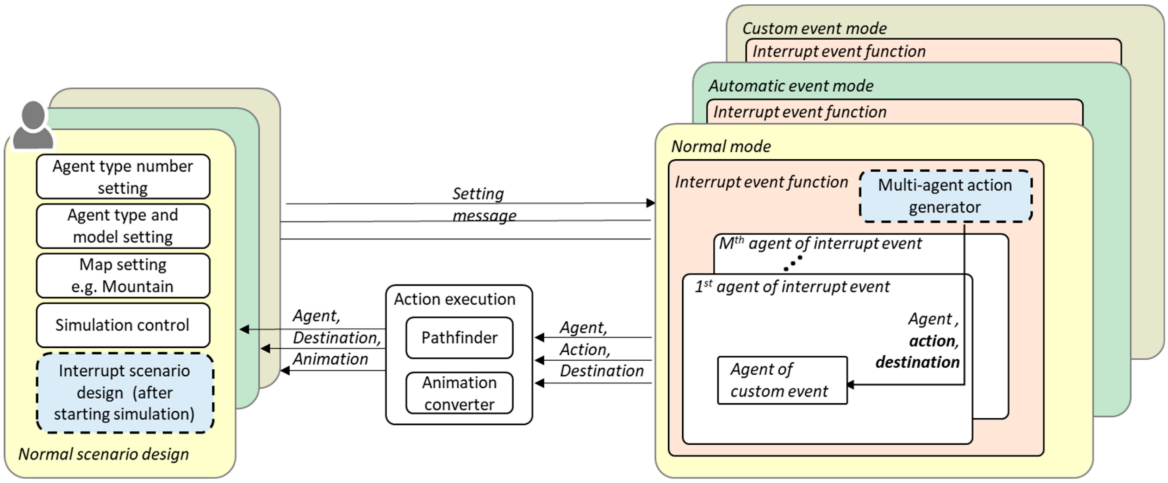

Figure 7. Overview of the interrupt event function.

4. Experiments and Analysis

In this section, we introduce the implementation platform and the scenario simulation results achieved in the normal mode, the automatic event mode, the custom event mode, and the interrupt event function.

4.1. The Implementation Environment

The AI module is implemented in the Python language. The editing and visualization module and the virtual autonomous vehicle module are implemented using the C++ application programming interface (API) in Unreal Engine. The experiments were carried on a desktop computer with an i9-9900 3.10 GHz CPU and NVIDIA GeForce RTX 2080Ti graphic card. Different modules are connected to each other by the Transmission Control Protocol/Internet Protocol (TCP/IP), which is a set of communication protocols used to interconnect network devices on the internet. We constructed two maps for the simulation—one city map and one mountain map. The 3D model list and animation list corresponding to each agent type are presented in Table 3. In the custom event and interrupt event modes, all animations are provided via the GUI at the same time as that when the user creates the actions of agents. All animations are used in the normal mode, but partial animations are applied in the automatic event mode.

Table 3. 3D model list and animation list of each agent type.

Agent Type	3D Model Type	Animation Type
Vehicle	Sedan, hatchback, SUV, convertible, sports car, limousine, scooter, bike, tow truck, snow blower, pick-up truck, fire truck, detachable truck, van truck, garbage truck, water sweeper, ambulance car, septic truck, and police car	No animations for vehicle.
Human	Boy, girl, young man, young woman, adult man, adult woman, policeman, traffic police, and fireman	Idle, walk, run, hit, carry, push, call (movable), call, shout, petting (squat), petting (stand), use phone (movable), use phone, talk (movable), talk, wave (single hand), and wave (both hand)
Animal	Dog, cat, wild boar, raccoon, water deer, roe deer, and deer	idle, walk, run, attack, hit, death, eat, sleep, sit, jump look around, and combo attack

In the remainder of this section, we demonstrate the differences between the GUI operation and simulation results of each mode. This is followed by analysis.

4.2. Normal Mode Results

The agent distribution in two frames by the LOD agent manager is depicted in Figure 8, where the red arrow represents the location and the direction of movement of the autonomous vehicle, the green points represent vehicles, the blue points denote humans, and the yellow points denote animals. It is evident from the figure that agents are only generated within the LOD area. This property reduces the workload of the entire framework. Further, humans and animals are observed to be primarily traveling on the sides of roads, and vehicles are observed to be driving only on the roads. This satisfies our design principle for the normal mode, by which agents' movements should follow traffic rules without intentionally disturbing the autonomous vehicle.

Figure 9 depicts the individual agents exhibiting motivation-driven movement in a simulation in the normal mode. The first panel depicts a boy speaking on a phone. The second panel depicts a boy crossing the road at a pedestrian crossing. The third panel depicts a wild boar eating on the side of the road. The fourth panel depicts a cat sitting on the sidewalk.

Figure 8. Distribution of agents in the normal mode. The red arrow represents the autonomous vehicle's location and direction of movement, and the green, blue, yellow points represent vehicles, humans, and animals, respectively.

Figure 9. Simulated agents in normal mode. From left to right: boy speaking on phone, boy crossing the road at pedestrian crossing, boar eating, and cat sitting on the sidewalk.

Simulation results corresponding to the mountain map and the city map are depicted in Figures 10 and 11, respectively. The simulations are different in the two cases. Animal agents only include 3D cat models in cities, while various other 3D animal models are available on mountain maps. The associated vehicle types in the mountain map are also different from those in cities. Buses or ambulances are unavailable on the mountain map. In the city map depicted in Figure 11, all agents are observed to follow traffic rules—humans cross roads at pedestrian crossings when the traffic light is green and vehicle stop when the traffic light is red.

Figure 10. Mountain scenario result of normal mode.

Figure 11. City scenario result of normal mode.

To make the simulation more realistic, we let more humans run when it is raining in the simulator in normal mode. Figure 12 shows two simulation results in snowy and rainy weather on the city map. The right figure shows that more human is running.

(**a**) Scenario in snowy weather (**b**) Scenario in rainy weather

Figure 12. Scenarios in different weather.

4.3. Automatic Event Mode Results

The automatic event mode is designed to simulate hazardous scenarios. We present the simulated results of the events described in Table 2 in the city and mountain maps. Figure 13 depicts the simulation results. Figure 13a depicts the event of an agent blocking the autonomous vehicle on the road, Figure 13b depicts the event of an agent crossing the road in front of the autonomous vehicle, Figure 13c depicts the event of a vehicle agent driving slowly in front of the autonomous vehicle, and Figure 13d depicts the event of a vehicle driving on the wrong side of the road in front of the autonomous vehicle.

(**a**) Blocking road

(**b**) Crossing road

(**c**) Drive in front of autonomous vehicle

(**d**) Drive on the wrong side of the road

Figure 13. Scenario simulation results in the automatic event mode.

4.4. Custom Event Mode Results

The custom event mode allows users to design events involving multiple agents. The design GUI is illustrated in Figure 14. The 'Add' button can be clicked to add an event, as depicted in Figure 15. The axis indicated in Figure 14 helps the user specify the event location in the three-dimensional virtual world. The activation distance specifies the distance between the autonomous vehicle and the event location at which the event will be executed, and this area is highlighted in pink in Figure 14. At each event location, the user can create a maximum of five agents—there are five empty slots in the GUI, each of which can be edited by the user. The user can configure the agent type, 3D agent model

type, and location to edit each slot, thus creating a new agent, as illustrated in Figure 15. The location of an agent is indicated by an axis that can be dragged to different points. A maximum of 20 actions can be assigned to each agent. After assigning the animation to one action, the user can input the destination of the action by changing the axis or defining a duration in the duration box, as illustrated in Figure 15.

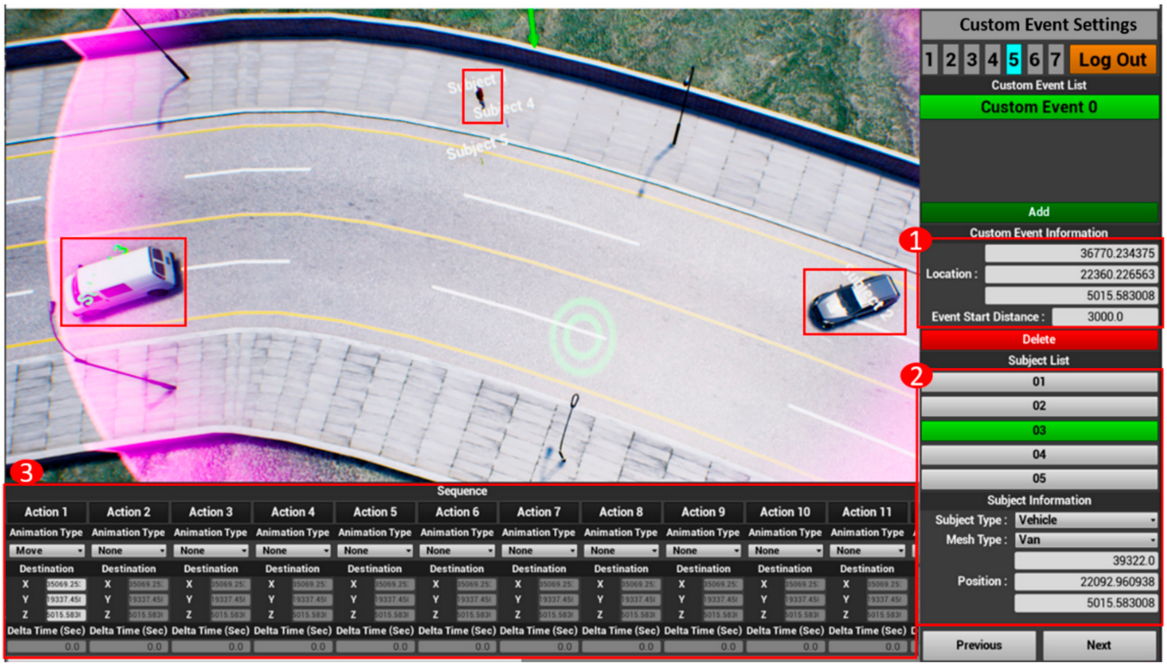

Figure 14. Editing UI in the custom event mode.

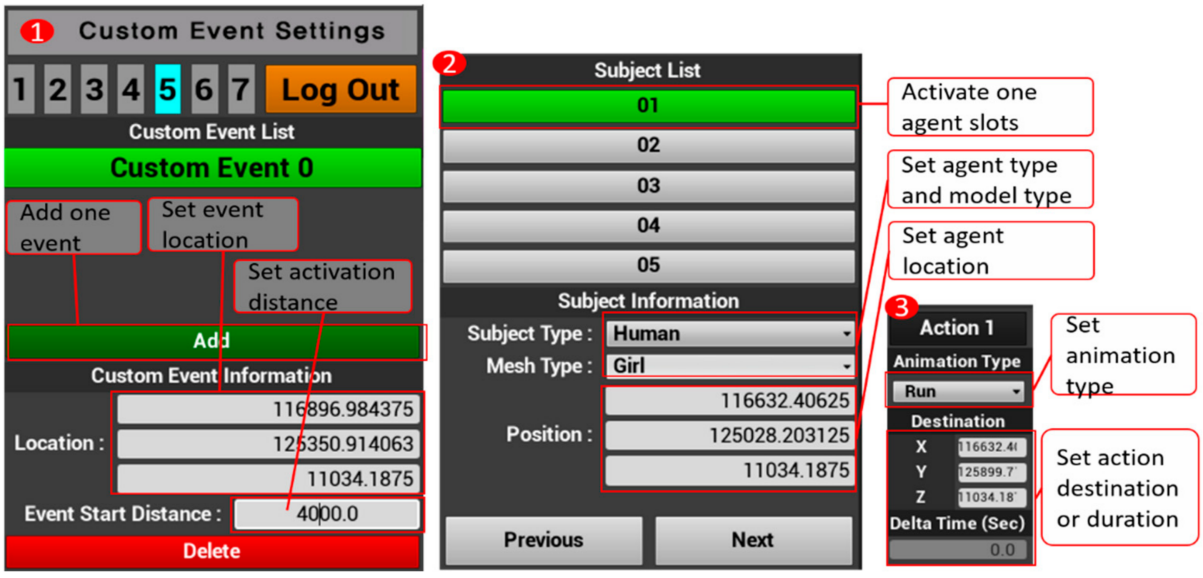

Figure 15. Detailed editing UI in the custom event mode.

Figure 14 depicts a case in which a person crossed the road and started with a running animation, ended with a walking animation, and had two vehicles driving on the road. Figure 16 depicts the execution results. The left figure shows the human starting to run and two vehicles going to arrive the destination. The right figure depicts the human going to arrive the second destination after arriving the destination of first action and the vehicle having arrived the destination.

Figure 16. Execution in the custom event mode.

4.5. Interrupt Event Function Results

Unlike the custom event mode, scenario design in the interrupt event function occurs while the simulation is underway. After pausing the simulation, users can create events, as depicted in Figure 17. The detailed editing UI is presented in Figure 18. This figure depicts a scenario in which an old woman is walking to the opposite side of the road. The execution results are presented in Figure 19.

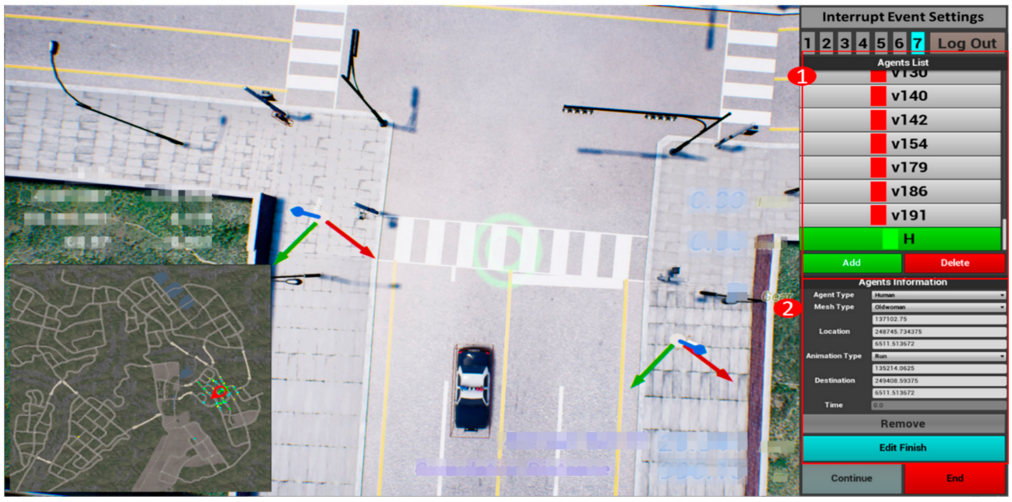

Figure 17. The editing UI of the interrupt event function.

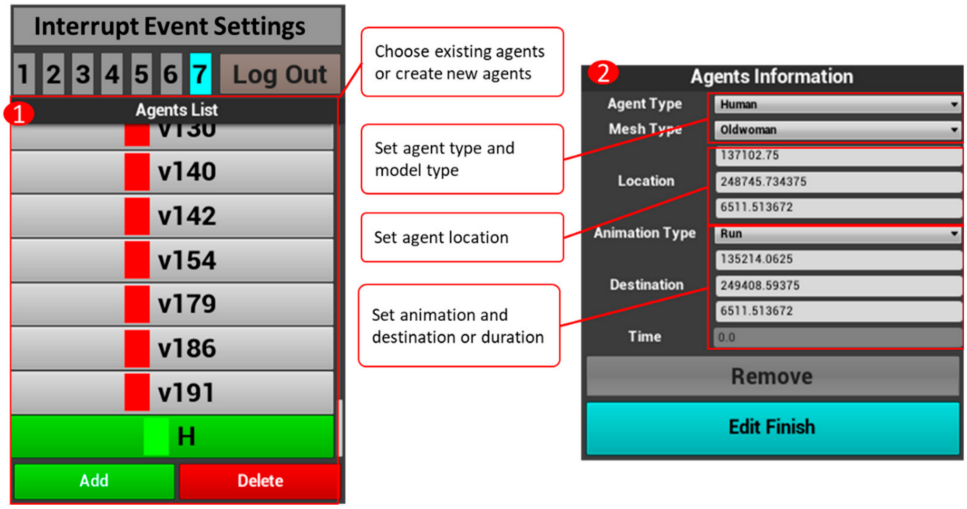

Figure 18. The detail editing UI of the interrupt event function.

Figure 19. The running of the interrupt event function.

4.6. Analysis

The most significant difference between our framework and existing simulators for autonomous vehicle is the scenario provided by the automatic event mode. We simulated five kind of hazardous events. One example scenario map in the automatic event mode is visualized in Figure 20, where the different color points represent different kind of hazardous events described in Table 2 and all event types were enabled while setting the scenario.

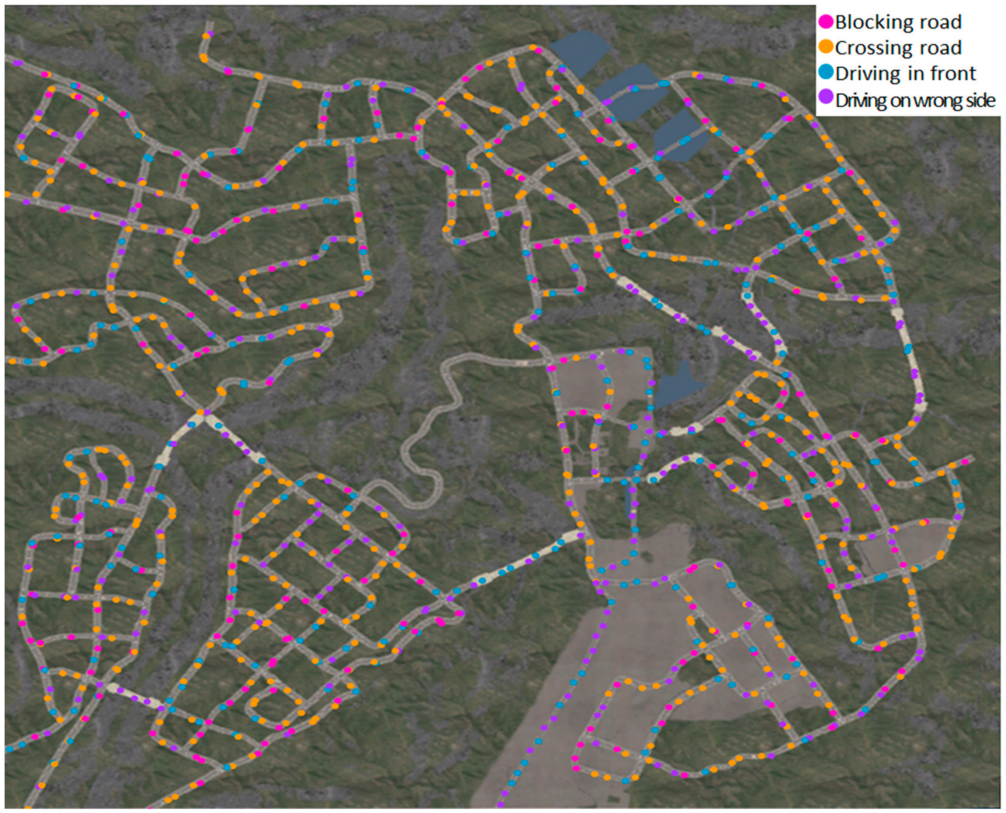

Figure 20. An example of generated scenario map in the automatic event mode.

Because the purpose is to let hazardous event occur in front of autonomous vehicle, the positions of all events are generated on roads, as shown in Figure 20. When we generate the events of one scenario, we consider the surrounding area of the road. For example, when there are no sidewalks for a road in the city map, human- and animal-related events will not be generated, as shown in Figure 20, where the white part of roads represents tunnels where no sidewalks are inside. Therefore, only last two types of events—driving in front and driving on the wrong side—were generated inside tunnels, as represented by blue and purple points, respectively.

In Figure 21, we also show the ratio of event types and agent types generated in the scenario shown in Figure 20. In total, there were 1132 events generated in the scenario.

Figure 21a shows that the crossing the road event occupied highest ratio, which was because the events of crossing the road (from left to right) and crossing the road (from right to left) were added together. The ratios of other events did not differ obviously. Three kind of agents are involved in automatic event mode. Vehicle agent-related events were found to occupy highest ratio because of driving front and driving reversely events only contain vehicle agents. Because human and animal agents are both involved in the blocking the road and crossing the road events, their ratio was closer compared to the ratio of vehicle agents. But the results could differ significantly if a user enables only specified event types in the scenario settings.

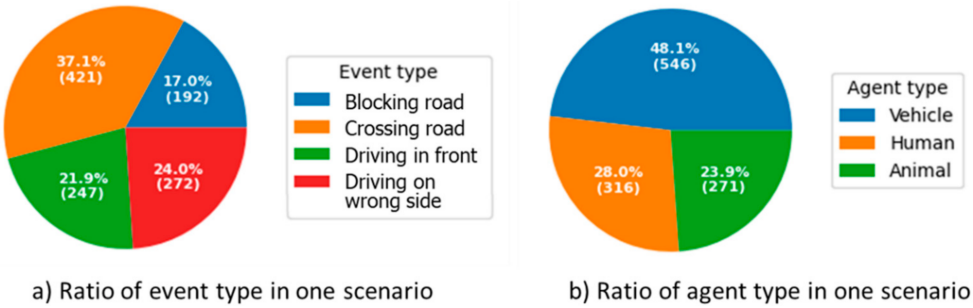

Figure 21. Ratio of event type and agent type in scenario shown in Figure 20.

To validate that events occurring near autonomous vehicles, we drove the autonomous vehicle in same path by running four scenarios. The path of autonomous vehicle and agent in each scenario is visualized in Figure 22. The red point in the figure represents the destination of one path. The path of autonomous vehicle is represented by the yellow line. The paths of five kind of events are represented by lines of different colors. Crossing the road (from left to right) and crossing the road (from right to left) are visualized by lines of the same color. Because event type 1 is blocking the road, whose agent does not move, its destination has the same start position. Thus, it is only represented by one red point. In Figure 22, we can see that the paths of different types of events in the simulation framework were found to have different patterns. The initial location and destination of event type 1 were always at the same location on the road; the initial location and destination of event type 2 were on different sides of the road; the destination of event type 3 was always closer to the autonomous car than the initial location, which was the opposite situation to event type 4. The revealed pattern was consistent with the events that have happened in real world. Figure 22 shows that most events occurred on the path of autonomous vehicle; this achieved our goal to let the events occur near autonomous vehicle. Several events did not occur on the path, e.g., event type 5 in Figure 22b,c. This is because the framework cannot anticipate the path of autonomous vehicles near junctions, and it needs to initialize events before the autonomous vehicle approaches too close to prevent the autonomous vehicle from capturing the scene in which agents are suddenly generated.

To validate that the events were executed in front of autonomous vehicle in the four scenarios of Figure 22, Figure 23 shows the closest distance between autonomous vehicle and agent of events during the autonomous vehicle driving to its destination in four scenarios. The x axis represents the frame index, and the y axis represents distance to autonomous vehicle. Each point represents one event, and its color represents the type of agent. According to the description in Figure 23, agents of most events can approach autonomous vehicle at a very close distance (0–20 m). To prevent the generation of agents in the virtual environment being captured by the autonomous vehicle, the agent of an event is generated when the autonomous vehicle is still far from the event location. In the junction area, because the framework cannot anticipate the driving path of the autonomous vehicle, events in multiple paths that are close to the junction are activated. When the autonomous vehicle does not go the path of the event, the agent cannot approach the autonomous vehicle. This is the reason that some event agents were far from the

autonomous vehicle in this simulation, as shown in Figure 23. Some agents were over 20 m away from autonomous vehicle.

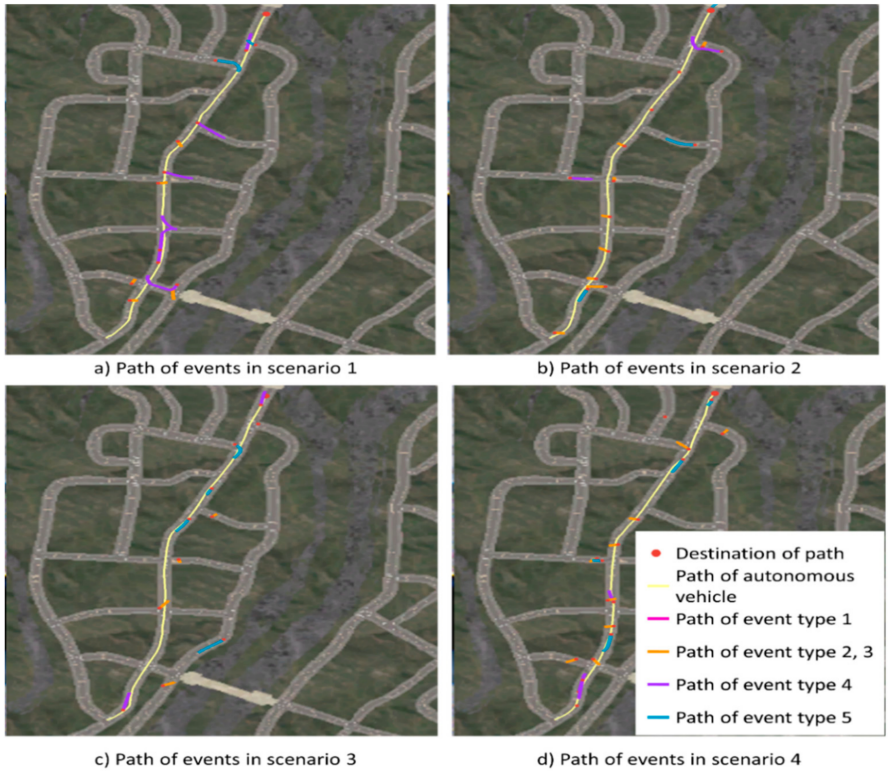

Figure 22. Path of events in four scenarios.

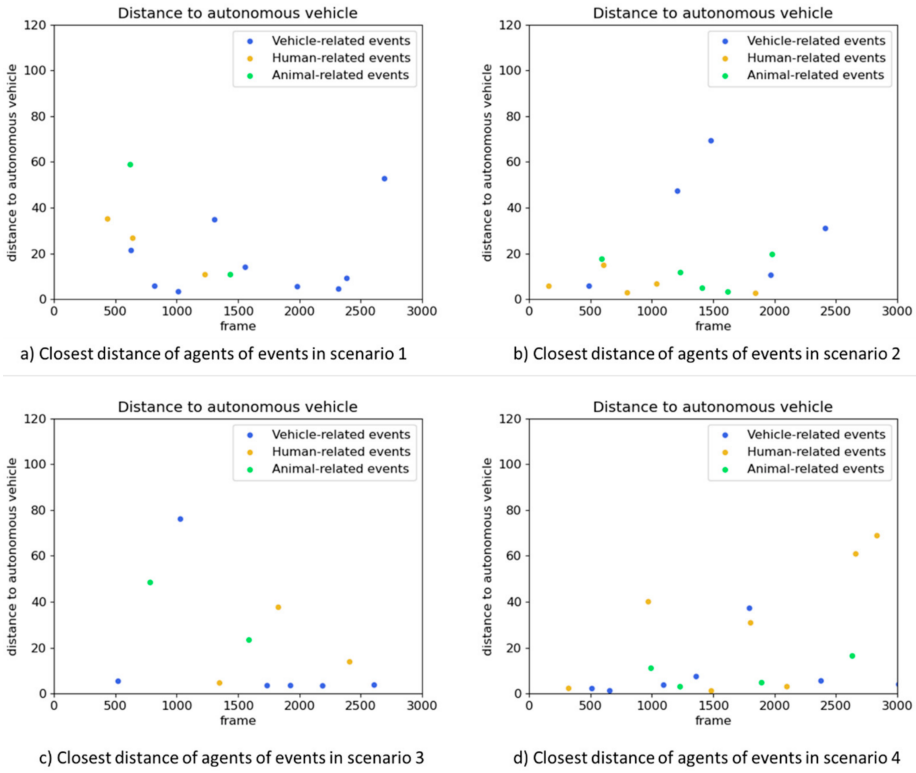

Figure 23. The analysis of distance between autonomous vehicle and agents in events of four scenarios in Figure 22.

The target of the framework is to generate scenarios on the road in autonomous driving simulators. A qualitative comparison results regarding the scenario effects with other simulators is infeasible due to the fact that the performance of an autonomous driving model trained in different simulators will be different due to uncontrollable variables, like different road networks and graphical quality. Thus, we show the advantages of our framework in respect to scenario generation abilities. Because Torcs and Deepdriving simulate scenarios with only the driving vehicle agents on the road (and were released a very long time ago), we compared our simulator with recently popular simulators for autonomous driving—AirSim, CARLA, LGSVL, and Apollo. Additionally, we compared our simulator with CarSim and CarMaker. As far as we know, there are no evaluation metrics for comparing scenarios in autonomous driving simulators, so we compared the scenario functions with a table. Table 4 shows the comparisons concerning the scenario functions. AirSim, LGSVL, and Apollo could can generate normal scenarios in which vehicle and human agents follow traffic rules in an automatic way. CARLA can generate only one kind of hazardous scenario that human or cyclist crosses the road. Our simulator is capable of generating more hazardous scenarios. To create more complex scenarios, our simulator has a GUI for user to create customized scenarios before and during the simulation. CARLA and LGSVL allow the user to create customized scenarios by script only, demanding the programming ability of users. CarSim and CarMaker only support customized scenarios. Moreover, all simulators do not support scenario generation and editing during the simulation process, which highlights the advantages of our simulator in that it allows users to check the learning results of handling specific scenarios of autonomous driving algorithms without terminating the training process. Compared to existing simulators, our simulator provides more flexible scenario generation functions for autonomous driving simulators with the automatic normal/hazardous scenario generation function, as well as an easy-use scenario generation GUI tool.

Table 4. The supported scenario functions in different simulators.

Supported Functions	AirSim	CARLA	LGSVL	Apollo	CarSim	CarMaker	Ours
Automatic normal scenario	√	√	√	√	×	×	√
Automatic hazardous scenario	×	√	×	×	×	×	√
Custom scenario by Script(S)/GUI(G)	×	√ (S)	√ (S)	√ (G)	√ (S/G)	√ (G)	√ (G)
Interrupt scenario	×	×	×	×	×	×	√

5. Discussion

In this study, we propose a virtual scenario simulation and modeling framework for simulators used to train and evaluate autonomous vehicles. The proposed framework operates in three modes—normal mode, automatic event mode, and custom event mode—and provides the interrupt event function. It was demonstrated that the proposed framework exhibits the advantage of being capable of simulating scenarios involving multiple types of agents and various 3D models for each kind of agent. In addition to normal scenarios that are common in the real world, our framework is capable of simulating hazardous scenarios. On the other hand, in addition to the automatic generation of scenarios, the proposed framework is equipped with a GUI to enable users to manually design their custom scenarios. The GUI supports scenario generation both before and during simulation to allow for user improvisation. Additionally, the LOD technique implemented in this study allows the framework to simulate scenarios involving a greater number of agents near autonomous vehicles with a higher efficiency compared to existing counterparts. The experimental results establish the potential of the proposed framework with regard the training and evaluation of autonomous driving. One limitation of the framework is its inability to simulate interactive actions, such as humans entering or exiting cars. Currently, the only difference in scenarios with different weather conditions is the amount of humans

performing the running action. In the future, we will design different actions for agents while considering different weather conditions.

Author Contributions: Conceptualization, methodology, and writing—original draft preparation, M.W. and J.P.; writing—review and editing, Y.S. and K.C.; software, M.W. and J.P.; investigation, J.P.; supervision, Y.W.P.; project administration and funding acquisition, K.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by a grant from Defense Acquisition Program Administration and Agency for Defense Development, under contract #UE171095RD.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kanada, T.; Thorpe, C.; Whittaker, W. Autonomous land vehicle project at CMU. In Proceedings of the 1986 ACM Fourteenth Annual Conference on Computer Science, Cincinnati, OH, USA, 4–6 February 1986; pp. 71–80.
2. Hullett, K.; Mateas, M. Scenario generation for emergency rescue training games. In Proceedings of the 4th International Conference on Foundations of Digital Games, Orlando, FL, USA, 26–30 April 2009; pp. 99–106.
3. Zook, A.; Lee-Urban, S.; Riedl, M.O.; Holden, H.K.; Sottilare, R.A.; Brawner, K.W. Automated scenario generation: Toward tailored and optimized military training in virtual environments. In Proceedings of the International Conference on the Foundations of Digital Games, Raleigh, NC, USA, 29 May–1 June 2012; pp. 164–171.
4. Luo, L.; Yin, H.; Car, W.; Lees, M.; Zhou, S. Interactive scenario generation for mission-based virtual training. *Comput. Animat. Virtual Worlds* **2013**, *24*, 345–354. [CrossRef]
5. Luo, L.; Yin, H.; Zhong, J.; Cai, W.; Lees, M.; Zhou, S. Mission-based scenario modeling and generation for virtual training. In Proceedings of the Ninth Artificial Intelligence and Interactive Digital Entertainment Conference, Boston, MA, USA, 14–18 October 2013.
6. Bhatti, G.; Brémond, R.; Jessel, J.P.; Dang, N.T.; Vienne, F.; Millet, G. Design and evaluation of a user-centered interface to model scenarios on driving simulators. *Transp. Res. Part C Emerg. Technol.* **2015**, *50*, 3–12. [CrossRef]
7. Luo, L.; Yin, H.; Cai, W.; Zhong, J.; Lees, M. Design and evaluation of a data-driven scenario generation framework for game-based training. *IEEE Trans. Comput. Intell. AI Games* **2016**, *9*, 213–226. [CrossRef]
8. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*; Springer: Cham, Switzerland, 2018; pp. 621–635.
9. ApolloAuto/Apollo. Available online: https://github.com/ApolloAuto/apollo (accessed on 22 November 2020).
10. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. *arXiv* **2017**, arXiv:1711.03938.
11. Rong, G.; Shin, B.H.; Tabatabaee, H.; Lu, Q.; Lemke, S.; Možeiko, M.; Boise, E.; Uhm, G.; Gerow, M.; Mehta, S.; et al. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. *arXiv* **2020**, arXiv:2005.03778.
12. Bella, F. Driving simulator for speed research on two-lane rural roads. *Accid. Anal. Prev.* **2008**, *40*, 1078–1087. [CrossRef] [PubMed]
13. TASS International. Available online: https://tass.plm.automation.siemens.com/ (accessed on 22 November 2020).
14. dSPACE. Available online: https://www.dspace.com/ (accessed on 22 November 2020).
15. Engineering Simulation & 3D Design Software | Ansys. Available online: https://www.ansys.com/ (accessed on 22 November 2020).
16. Driving Simulation for Autonomous Driving, ADAS, Vehicle Dynamics and Motorsport. Available online: http://www.rfpro.com/ (accessed on 22 November 2020).
17. Cognata | Autonomous and ADAS Vehicles Simulation Software. Available online: https://www.cognata.com/ (accessed on 22 November 2020).
18. Metamoto. Available online: https://www.metamoto.com/ (accessed on 22 November 2020).
19. Mechanical Simulation. Available online: https://www.carsim.com/ (accessed on 20 January 2021).
20. CarMaker: Virtual Testing of Automobiles and Light-Duty Vehicles. Available online: https://ipg-automotive.com/products-services/simulation-software/carmaker/ (accessed on 20 January 2021).
21. Lopez, N.G.; Nuin, Y.L.E.; Moral, E.B.; Juan, L.U.S.; Rueda, A.S.; Vilches, V.M.; Kojcev, R. gym-gazebo2, a toolkit for reinforcement learning using ROS 2 and Gazebo. *arXiv* **2019**, arXiv:1903.06278.
22. NVIDIA Isaac Sim. Available online: https://developer.nvidia.com/isaac-sim (accessed on 22 November 2020).
23. Wymann, B.; Espié, E.; Guionneau, C.; Dimitrakakis, C.; Coulom, R.; Sumner, A. Torcs, the Open Racing Car Simulator. 2000, Volume 4, p. 2. Available online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.710.6672&rep=rep1&type=pdf (accessed on 15 March 2021).
24. Chen, C.; Seff, A.; Kornhauser, A.; Xiao, J. Deepdriving: Learning affordance for direct perception in autonomous driving. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 2722–2730.
25. Jang, H.; Hao, S.; Chu, P.M.; Sharma, P.K.; Sung, Y.; Cho, K. Deep Q-network-based multi-criteria decision-making framework for virtual simulation environment. *Neural Comput. Appl.* **2020**, 1–15. [CrossRef]

26. Nguyen, H.T.; Chu, P.M.; Park, J.; Sung, Y.; Cho, K. Intelligent motivation framework based on Q-network for multiple agents in Internet of Things simulations. *Int. J. Distrib. Sens. Netw.* **2019**, *15*, 1550147719866385. [CrossRef]
27. Zhang, W.; Chu, P.M.; Park, J.; Sung, Y.; Cho, K. Driving data generation using affinity propagation, data augmentation, and convolutional neural network in communication system. *Int. J. Commun. Syst.* **2021**, *34*, e3982. [CrossRef]
28. Wen, M.; Park, J.; Cho, K. A scenario generation pipeline for autonomous vehicle simulators. *Hum. Cent. Comput. Inf. Sci.* **2020**, *10*, 1–15. [CrossRef]