Soft-body Simulation and Procedural Generation for the Development and Testing of Cyber-physical Systems

Alessio Gambi
University of Passau
Passau, Germany
alessio.gambi@uni-passau.de

Pascale Maul, Marc Mueller,
Lefteris Stamatogiannakis,
Thomas Fischer
BeamNG GmbH
Bremen, Germany
{pmaul,mmueller,estama,tfischer}@beamng.gmbh

Sebastiano Panichella
Zurich University of Applied Science
Winterthur, Switzerland
panc@zhaw.ch

***Abstract*—Industry and research organizations increasingly rely on simulation platforms to facilitate the development and validation of Cyber-physical Systems (CPSs). The main factors for this trend are simulation's cost-efficiency and the possibility of evaluating the system's performance early on and throughout the development cycle in a fully controlled environment. However, simulations need to meet stringent functional and non-functional requirements to benefit development and debugging activities. In particular, high simulation accuracy and the ability to systematically generate relevant test scenarios are paramount for effectively assessing CPSs' behaviors in nominal and critical test scenarios. This paper (i) discusses soft-body simulation and procedural content generation relevance to achieve the systematic generation of physically accurate virtual tests; (ii) and presents BeamNG.tech a novel simulation framework featuring both soft-body simulation and procedural content generation. Hence, we report on the main advantages and research results in testing self-driving car software enabled by BeamNG.tech. Finally, we reflect on the central role of simulation-based continuous integration and testing pipelines to improve current CPSs development practices.**

***Index Terms*—Software Testing, Cyber-physical Systems, DevOps, Virtual Environments.**

I. Introduction

Cyber-physical Systems (CPSs) are systems in which algorithms analyze sensor data collected from the surrounding environment to control physical actuators. Emerging CPS—from medical monitoring systems and devices, industrial robots, and autonomous transportation systems—are expected to play a crucial role in future generations' quality of life and the global economy [8]. These CPSs typically interact with humans as well as other systems in highly dynamic and unpredictable environments. Thus, guaranteeing CPSs' reliability and safety are critical challenges to address, primarily when CPSs operate in critical scenarios, such as driving in traffic or detecting wildfires [40].

Current CPS development practices have several limitations and drawbacks, including (i) the limited ability to repeat tests under the same conditions due to ever-changing environmental factors [26]; (ii) the difficulty to test the systems in safety-critical scenarios, with the goal to avoid irreversible damage causes by dreadful outcomes [22], [38], [39]; (iii) not being able to guarantee the system's reliability and safety in its operational design domain due to a lack of testing under a wide range of execution conditions [25]. Consequently, new development practices, environments, and frameworks are needed to address the fundamental development challenges of *observability*, *testability*, and *predictability* of CPSs and support their foreseen wide-spread adoption.

Simulation-based testing of cyber-physical systems. CPSs require flexible development and verification strategies to account for Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), and Hardware-in-the-Loop (HiL) paradigms [2]. CPSs are also more difficult and expensive to test and integrate than traditional software systems [19], [40]. Common reasons for this are that the final version of the hardware is often available only late, and integrating hardware components requires a great deal of manual effort. A typical approach to dealing with CPSs safety and testing requirements is to develop *hardware proxies*, such as system simulators and digital twins [23] (see Figure 1). CPSs safety requirements are nowadays evolving under the direction of organizations responsible for software system and development standardization practices and policies (e.g., consider the *ISO21448*[1] and *ISO26262*[2] standards). According to these organizations, simulations can facilitate testing of CPSs safety requirements, as they are inexpensive and less dangerous than running the systems in real-life [7], [11]. Consequently, the development and validation of CPSs heavily rely on them.

In this context, we argue that simulators need to meet stringent functional and non-functional requirements (e.g., accuracy, efficiency, photo-realism) to be beneficial for developing CPSs. Additionally, they must also support the systematic generation of test scenarios to effectively assess the behavior of CPSs in nominal and critical conditions [13], [15].

[1]https://www.iso.org/standard/70939.html
[2]https://www.iso.org/standard/43464.html

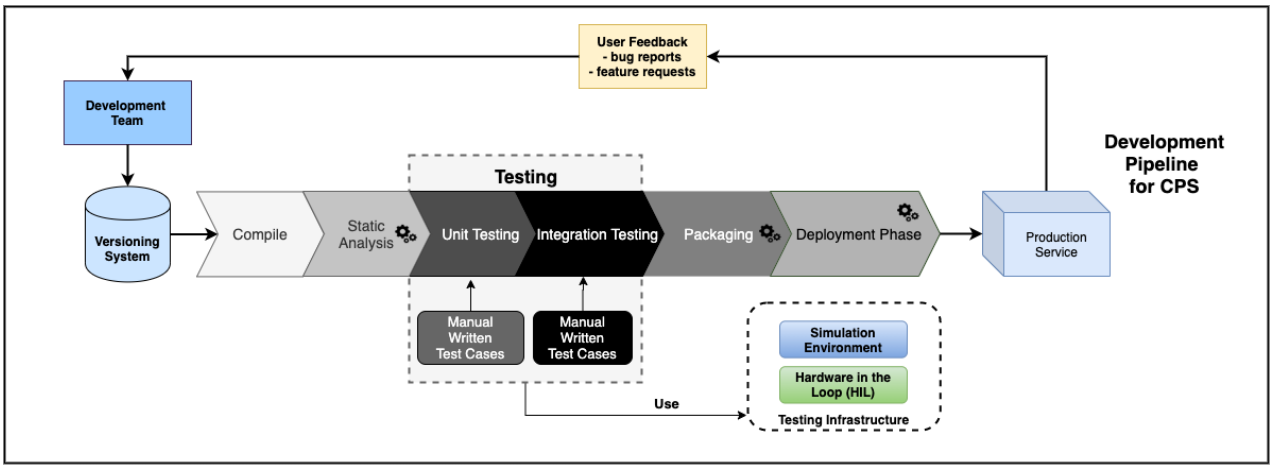

Fig. 1. (Idealized) Development pipelines for Cyber-physical Systems.

Paper contributions and organization. This paper makes a case for soft-body simulation as a necessary complement to mainstream rigid-body simulations (Section II) and remarks on the importance of enabling automation in both test generation and execution also in the context of Cyber-physical Systems. In detail, the main contributions of this paper are:

(1) we introduce BeamNG.tech, a novel simulation framework provided by BeamNG GmbH featuring a highly accurate and faster-than-real-time soft-body simulator and an intuitive API for automatically generating virtual environments, collecting training and validation data, and executing test scenarios (Section III).
(2) We summarize the latest results in research, education, and industry enabled by BeamNG.tech (Section IV), and discuss the most relevant state-of-the-art approaches on CPSs simulation (Section V).
(3) We critically reflect on the path towards the future development of BeamNG.tech and simulation-based continuous integration and testing for sustaining the realization of reliable and safe CPS evolution (Section VI)

II. Simulation Approaches for CPSs Development

This section discusses the importance of simulation environments in developing and testing CPSs and the main simulation approaches. We contextualize the discussions in the domain of self-driving cars.

X-in-the loop activities in CPSs development. Several researchers and practitioners advocate DevOps as a promising approach for CPSs development [9], [40]. However, both in traditional [9] and CPSs application domains, the state-of-the-art of DevOps is still forming [40], and emerging practices need validation in the wild. A recent survey by Törngren and Sellgren [40] discusses how CPSs' engineering deals with the inner complexity of CPSs' design and the challenges that arise from the environments in which CPSs operate. According to Törngren and Sellgren, while (semi-automated) integration happens through software, there are ***several distinguishing characteristics*** between software and physical systems that make co-designing hardware and software hard. Those characteristics include entirely different approaches, techniques, abstractions, platforms, faults & failure modes, and development practices [40]. Törngren and Sellgren conclude that to cope with the foreseen demand of CPSs at scale and in multiple domains, CPS development and testing need rapid prototyping, code/test generation, and various testing phases [34] encapsulating X-in-the Loop (XiL) activities. In XiL, the 'X' indicates the target of development and testing and typically refers to model (MiL), software (SiL), and hardware (HiL). Hence, a typical CPS development pipeline needs to efficiently and effectively integrate various XiL activities to support development and evolution (see Figure 1) [34], [40]. Simulations are becoming one of the cornerstones in developing and validating CPSs, as they are heavily utilized in various XiL activities and across the entire development life-cycle. For example, simulations are currently used to support the initial inception and requirement analysis (MiL), Hardware/Software Co-design (MiL), design and testing of software components (SiL), training and validation of Machine Learning components (SiL), and testing and validation of the deployed system (HiL).

Main simulation approaches. Many simulation technologies have been developed to support developers in various stages of design and validation. For instance in the self-driving cars domain, developers resort to basic simulation models [17], [36], rigid-body simulations [29], [42], and soft-body simulations [13], [33] among others.

Basic simulation models, like MATLAB and Simulink models, implement fundamental abstractions (e.g., signals) but target mostly non-real-time executions and generally lack photo-realism. Consequently, their usage as a means for integration and system-level software testing is limited, and they are mainly utilized for model-in-the-loop simulations and Hardware/Software co-design.

Rigid-body simulations approximate the physics of bodies by modeling entities as undeformable bodies or as compositions of a limited number of rigid parts. Basic simulation entities are three-dimensional objects such as boxes, cylinders, and convex meshes [3]. Rigid-body simulations implement a coarse approximation of reality; hence, they are not computationally demanding and can scale well in the number of simulated entities. They can capture object motions and rotations but cannot deform or break the rigid simulation elements. To illustrate this aspect better, let us consider a

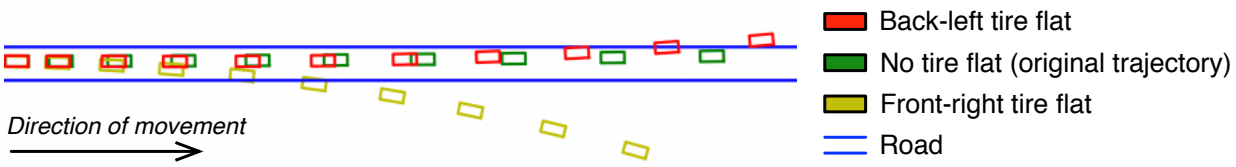

Fig. 2. An illustrative example of how BeamNG.tech can easily simulate the effect of flat tires. The plot shows the position of a simulated vehicle driving on a straight segment of the road when the steering control is fixed and one tire is flat (in red and yellow). For comparison, the plot reports the trajectory of the car when no tires are flat (in green).

scenario in which a car hits the curb while driving down a road generating a sudden lateral force acting on the car's parts (e.g., tires, suspensions). Assuming the impact's intensity is not too high, in real life, the vehicle absorbs the impact as its parts dissipate energy, possibly deforming. Consequently, the resulting movement of the vehicle would be relatively smooth. On the contrary, a standard rigid-body simulation of this scenario would result in the (simulated) car having a "unnatural" (i.e., discontinuous) behavior due to the curb's side wheel suddenly "jumping" from road level to the curb's top surface in whole.

Soft-body simulations can simulate deformable and breakable objects and fluids; hence, they can handle a wide range of simulation cases. Mass-spring systems and finite element method (FEM) are the main approaches for solid body simulations, while finite volume method (FVM) and finite difference method (FDM) are the main approaches for simulating fluids [30]. In CPSs, specifically self-driving cars, mass-spring systems and FEM are the most suited soft-body simulation approaches since they target solid objects. They approximate solid objects as sets of atomic elements that react to forces and interact with each other. Therefore, soft-body simulations follow a bottom-up approach: they model the behavior of the atomic elements that compose each body and the high-level behavior of that body *emerges* from the atomic element interactions. Mass-spring systems model bodies as structures composed of nodes (i.e., points with mass) connected by beams (i.e., elastic elements); they simulate body deformations and fractures by controlling the beam's characteristic attributes such as spring, damping as well as deformation and strength thresholds. FEM, instead, simulates bodies' elastic properties by solving the partial differential equations that define the stress and strain acting upon discrete hexahedral or tetrahedral elements that compose the bodies. FEM is more accurate than mass-spring systems but has a higher computational demand due to its complexity [30]. Consequently, the achievable simulation efficiency of FEM is usually significantly lower than mass-spring systems, strongly limiting FEM's applicability for real-time simulations.

Both rigid- and soft-body simulations can be effectively combined with powerful rendering engines to implement photo-realistic simulations [5], [7], [11], [41]. However, soft-body simulations can simulate a wider variety of physical phenomena compared to rigid-body simulations. For example, soft-body simulations can model body deformations, fracture, vibrations, anisotropic mass distributions, and inertia, essential in many CPSs scenarios. Soft-body simulations are also very versatile. As stated by Dalboni and Soldati, *"using the elementary description of target systems as collections of nodes and beams, it is possible to contemplate all the laws of mechanics that rule the physical world"* [10]. Consequently, soft-body simulations can easily simulate different materials and other phenomena, such as aerodynamics and pressured volume changes relevant in many CPSs domains. For example, to simulate how changes in the tires' pressure impact vehicles' response to driving commands, soft-body simulators "simply" deflate the tires (i.e., they do not change how they simulate physics nor implement any heuristic to handle this specific case). To illustrate this point, we report in Figure 2 the results of one experiment that we ran using BeamNG.tech. For this experiment, we created a straight road segment (blue in the figure). Next, we programmed a vehicle to drive on it without flat tires (in green). Finally, we repeated the same experiment but deflated a different tire each time (red and yellow). In Figure 2, we show the position and direction of the vehicle during the simulations. As shown in the figure, the vehicle keeps to the road when no tires are flat, but its trajectory drastically changes as soon as one of its tires is flat. Noticeably, the trajectory consistently changes depending on which tire (i.e., front or rear, left or right) is flat. In contrast, to implement these scenarios, rigid-body simulations would need to adapt the wheel model and the friction coefficient to "brute-force" replicate the effects of the deflated tires.

Soft-body simulators require more complex models than rigid-body simulators. Those models are not trivial to create [10] and impose higher computational demand, but enable soft-body simulators to achieve higher simulation accuracy. Therefore, rigid-body simulations are standard tools for simulating complicated traffic scenarios where simulated entities' movement is mostly unrestricted (e.g., trajectory planning). In contrast, soft-body simulations are a better fit for implementing safety-critical scenarios (e.g., car crashes [13]) and focused scenarios in which high simulation accuracy, even in extreme situations, matters the most.

III. BeamNG.tech in a Nutshell

BeamNG.tech is a framework specializing in autonomous driving and driver's training. It has been recently released under a mix of commercial and open-source licenses by BeamNG GmbH.[3] The framework features a powerful soft-

[3]BeamNG.tech is available at https://beamng.gmbh/research/ and is free for non-commercial use.

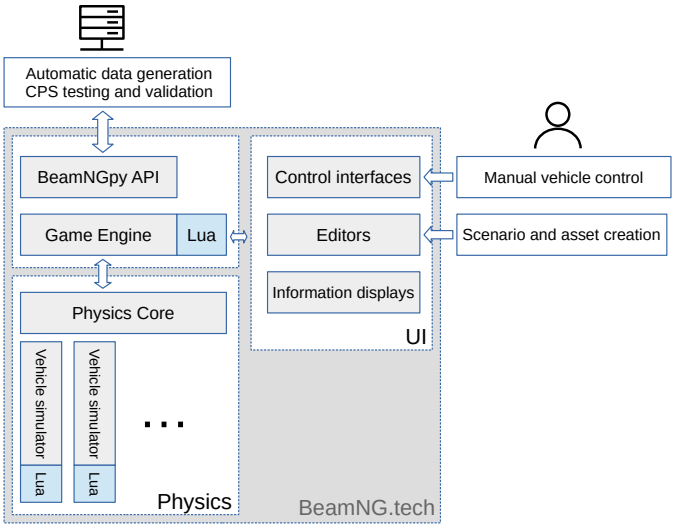

Fig. 3. Overview of BeamNG.tech

body simulator, several editors (e.g., road editors, track generator, world editor), and an extensive API that enables procedural content generation, automatic collection of simulation data for training and validation, and control of simulated vehicles using external programs (e.g., driving AIs). Additionally, since BeamNG.tech is derived from a video-game developed by the same company, the framework (i) comes with an unmatched set of assets, including more than twenty vehicles, seven trailers, hundreds of obstacles, elements of scenery and types of vegetation, and seventeen detailed maps scoring forty-four square kilometers of urban and suburban environments; (ii) it is well maintained and meets stringent requirements on real-time execution and visual realism; (iii) the majority of its components are open-source[4] and extensible; and, (iv) it has a large user base that continuously provides fresh and diverse content, extensions, *mods*, and add-ons.

In this section, we briefly describe BeamNG.tech's software architecture and main components (Section III-A), its approach to soft-body simulation (Section III-C), and how part of its APIs (i.e., BeamNGpy) can be used to procedurally generate environments and test scenarios (Section III-D).

A. BeamNG.tech's Architecture

BeamNG.tech is organized around a central *game engine* that communicates with the *physics simulation*, the *UI*, and the *BeamNGpy API*[5] as exemplified by Figure 3. CPS' developers and testers can use BeamNG.tech via its UI or API. The UI can be used for game control, and its diverse editors enable assets and content creation (e.g., *gameplay*, *scenarios*). For example, developers can use the world editor to create or modify the virtual environments that are used in the simulations, while testers can create test scripts (*gameplay* in the figure) to setup dynamic scenarios. The API, instead, allows the automated generation and execution of test scenarios, the collection of simulation data for training as well as testing and validating CPSs (e.g., camera images, LIDAR point clouds, g-forces), and the possibility to interact with the running simulation (e.g., drive vehicles, move objects around). The *game engine* manages the simulation setup, camera, graphics, sounds, gameplay, and overall resource management. The *physics core* handles resource-intensive tasks such as collision detection, basic physics simulation and orchestrates the multi-threaded execution of the vehicle simulators (see Section III-C). The *vehicle simulators* —one for each of the simulated vehicles— simulate the high-level driving functions and the various vehicle sub-systems (e.g., drivetrain, ABS, ESC, Turbo).

[4]The Lua code is open-source under the CDDL or MIT licenses but the C++ core simulator is not.

[5]BeamNGpy is available as a package from PyPI, the Python Package Index, while its source code is available at https://github.com/BeamNG/BeamNGpy

B. BeamNG.tech's Code Base

BeamNG.tech is implemented using various technologies but primarily consists of C++ and Lua code. The framework code base contains more than 350k lines of C++ code and more than 200K lines of Lua code (without considering third-party libraries). The physics simulation alone accounts for some 10k lines of C++ code.

C. BeamNG.tech's Soft-body Simulation

This section describes how BeamNG.tech implements the physics simulation.

The Core Physics Model. At its very core, BeamNG.tech implements the fundamental spring-mass model (see Section II). In its basic form, this model consists of *nodes* with mass connected by weightless elastic elements, called *beams*. Beams implement a parallel mass-spring-damper circuit and obey Hooke's law extended with damping:

$$F_s = -kx - c\dot{x} \tag{1}$$

According to this law, the force F_s necessary to extend or compress a spring by some distance x is proportional to that distance by a positive factor k that characterizes the spring's stiffness. Damping, instead, is a force that opposes the spring's motion and is proportional to the spring' velocity $(\dot{x})$ by a positive factor c that characterizes the spring's internal friction. Therefore, the mass-spring model can simulate how forces applied to the nodes propagate through the beams causing them to expand, compress, deform, or even break.

Nodes are dimensionless mass-points that occupy a position in space and are affected by the sum of forces propagated by the elastic elements that connect them. Since BeamNG.tech does not model nodes' rotation, nodes do not sum any momentum; hence, they act as hinges free in all axes. Nevertheless, nodes have frictional properties and collide against other simulated entities. Nodes can be organized along one dimension to form chains, two dimensions to form polygonal meshes modeling the edges of objects' surfaces, or three dimensions to form complex networks modeling the internal structure of bodies [10]. Figure 4 (left panel) exemplifies how the body of a vehicle is modeled as a three-dimensional network in BeamNG.tech.

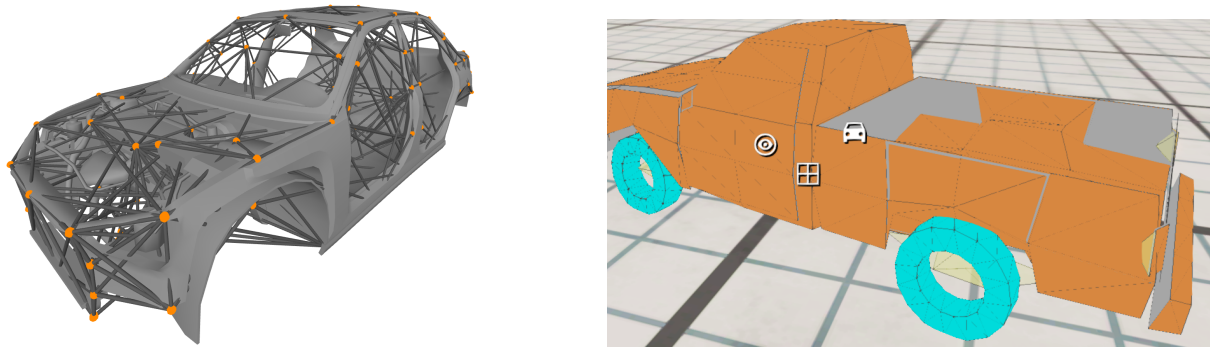

Fig. 4. An example of how BeamNG.tech models the internal (left panel) and the external (right panel) structure of vehicles. The internal structure is modeled as a three-dimensional network of nodes and beams, while the external structure consists of triangles.

Beams and triangles connect nodes. Beams enable the simulation of bodies' deformation, while triangles enable the simulation of aerodynamic' drag, lift and the simulation of enclosed pressured surfaces. Additionally, triangles enable detecting collisions and computing collision response. Figure 4 (right panel) highlights how the triangles contribute to the vehicle's body modeling in BeamNG.tech. To help developers build accurate models that involve linear and non-linear behaviors typical of real-life components, BeamNG.tech defines various types of beams. Noticeably, this is one of the most distinctive novelties of the framework. *Basic beams* have constant spring and damping rates, hence implement the linear behavior of Hooke's law (see Equation 1). They can stretch, shrink, break, or permanently deform and can be used to model various materials by adjusting their deformation and break thresholds. *Bounded beams*, instead, have different spring and damp rates depending on how fast —not how much— they are compressed or expanded. They simulate materials with non-linear properties, like real-life car suspension's spring and dampers, and might have a defined length to limit suspension's travel. *Anisotropic beams* implement different behaviors during compression and expansion [16]. They can be used to model structures that are easy to compress but very hard to expand, such as a tire's sidewall, and "supporting" structures that resist compression but are free to expand. Finally, *L-beams* and *torsion bars* model angular springs and can be used to simulate the leaf springs in real-life suspension systems.

The Vehicle Models. The diversified types of BeamNG.tech's beams allow developers to accurately model complex systems, such as complete vehicles. To facilitate reusability and maintainability, BeamNG.tech models vehicles as ensembles of individual components that correspond to real-life vehicle's part (see Figure 5). Not all the components are modeled at the same level of detail. Some components, such as wheels and suspensions, are modeled as beams and nodes to accurately simulate their physics. Other components, instead, are modeled only in abstract form; for instance, this is the case of vehicles' engine. Figure 5 illustrates the main components that form a BeamNG.tech car's body (left panel) and how wheels and suspensions are composed of nodes and beams (right panel). The vehicle models encode the rules that specify how parts are constructed, how they fit together, and how they interact with each other. For example, vehicle models specify how the tires are constructed using nodes and beams and how wheels are attached to the wheel' hubs. Similarly, they describe how the hubs are attached to the suspensions and the chassis' suspensions. The vehicle models also describe how torque propagates from the engine through the car's powertrain (e.g., gearbox, differential, axles) to the wheels. The vehicle models are comprehensive and make BeamNG.tech extremely versatile as they allow to model structurally different vehicles in various configurations by directly picking the right parts to assemble them, as it happens in reality. For instance, according to these models, motorcycles, cars, and trucks need an engine, but trailers do not; motorcycles have two wheels, cars have four, while trucks have more. These models also contain all the necessary logic to simulate headlights, sounds, materials, and the various driving-assistance systems (e.g., ABS, ESC) that contribute to the final driving simulation and data for realistically render the vehicles.

Simulation execution model. BeamNG.tech implements a two-layered simulation: at high-level, it simulates components that generate forces (e.g., vehicles engines, external controllers) without representing them explicitly as beams and nodes; at low-level, it simulates how the forces propagate across soft-bodies composed of nodes and beams. The simulation execution follows the *actor model* [20]: different vehicles are independent and concurrently executed, and they communicate by passing messages using event queues. Consequently, all the interactions between them happen asynchronously, enabling BeamNG.tech to parallelize the physics simulation's execution, the game logic, and the vehicle models on multiple threads, and simulating the force propagation across a large number of elements efficiently. On top of this layered, "distributed" execution model, BeamNG.tech can run simulations faster than real-time using gaming setups as well as commodity PCs. Specifically, on the reference benchmark [6] BeamNG.tech runs the simulations up to five times faster than real-time for one simulated vehicle. We report the results of

[6]The reference benchmark is available along with the BeamNG.tech's distribution.

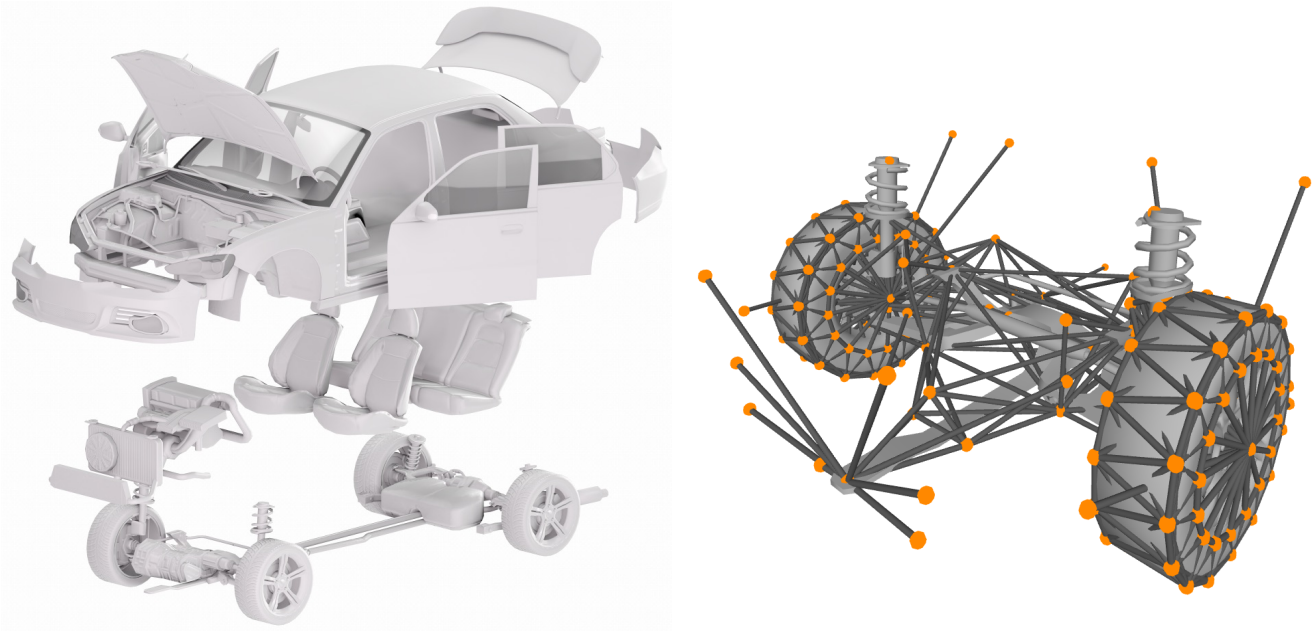

Fig. 5. An example of a BeamNG.tech's vehicle model. The vehicle is composed of different parts that are assembled to resemble real-life settings. The 3d graphic meshes of these parts are presented in (left panel). The vehicle physics design contains 714 nodes that are connected by 4479 beams of various types and 976 triangles. The rear suspension part is presented in (right panel).

TABLE I
RESULTS OF BEAMNG.TECH'S REFERENCE BENCHMARK IN WHICH AN INCREASINGLY LARGE NUMBER OF VEHICLES IS CONCURRENTLY SIMULATED.

# Cars	MBeams / sec	Real-time speed up
1	46.723	513%
2	87.503	480%
3	114.311	418%
4	139.884	384%

We ran the benchmark using a non-dedicated machine with 16 GB of memory, and an i7 CPU with 4 physical and 4 virtual cores, running at 3.10GHz.

the reference benchmark in Table I.

Low-level and high-level simulations are tightly coupled: the components continuously communicate with each other and propagate the forces or torques to the nearby components. Force propagation is bi-directional and happens across both the simulation layers. For example, the low-level physics core calculates the forces acting on a wheel's hub and tire nodes. Next, it passes the angular velocity of the wheel's hub to the high-level drivetrain. Then, the drivetrain propagates the wheel's angular velocity up to the engine that computes the torque. Finally, the engine propagates the torque back to the wheels via the drivetrain, and the physics core applies the torque to the wheel's hub nodes causing the beams that connect it to the tire's nodes to pull them, effectively making the tires move. Similarly, acting on the high-level steering wheel causes a force that propagates through the steering rack and eventually makes the wheels turn.

Noticeably, when developers use real steering wheels to drive the simulated vehicles, the simulated wheels' reaction forces are fed back to the steering wheel and realize the so-called force feedback. Since the simulation continuously propagates forces back and forth across connected components, BeamNG.tech ensures the simulation's stability by running the simulation loop at a very high-frequency (up to 2KHz). Additionally, because BeamNG.tech simulates physics at high-frequency, it can model subtle effects, such as the high-frequency micro-vibrations that happen to the wheels as the tires approach their grip limits, which are notoriously hard to simulate accurately.

BeamNG.tech builds on top of the mass-spring approach to soft-body simulation but extends it in many ways to accurately simulate the physics, pressure, aerodynamics, and collisions of entire vehicles (see Figure 6). Additionally, the use of comprehensive vehicle models and massively parallel simulation execution enable faster than real-time execution of a large variety of vehicles (see Table I).

D. Procedural Content Generation with BeamNGpy

As with many other simulations, BeamNG.tech allows users to manually provide additional content to the platform like three-dimensional models, material definitions, and maps. However, differently than most of the state-of-art simulations, it comes with a set of editors that allow the modification or creation of environmental assets directly inside the platform. It also provides an API that automates scenarios generation, test execution, and data collection and allows external programs to control the simulation. A fundamental requirement for testing driving AIs and implementing test oracles.

BeamNGpy organizes simulations around the concept of *scenarios*, driving tasks that the car under tests, (i.e., the *ego-car*) must solve; a common scenario is to drive from

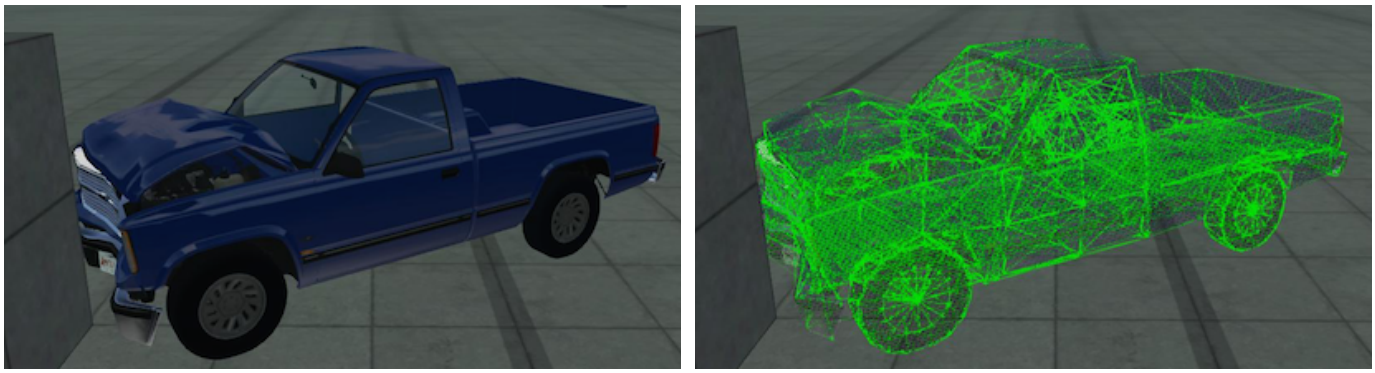

Fig. 6. Simulation of the physics, pressure, aerodynamics, and collision of a vehicle in BeamNG.tech

a starting location to a target area without driving off-road nor crashing [28]. Scenarios take place in *environments* that are defined by a *terrain*, environmental parameters such as *time of day*, and contain *roads* and *static objects* (e.g., props, obstacles). BeamNGpy allows controlling environmental conditions and lighting, the generation of roads, lane markings, curbs, and props' placement and sizing. Like parked cars and traffic signs, complex static objects require the availability of three-dimensional mesh models that define their geometries. Instead, simple objects, like cones and cubes, are procedurally generated.

Scenarios involve at least one *ego-car* equipped with sensors, like cameras and *lidars*, and controlled by the driving agent under test. Scenarios might include additional vehicles that can be programmed using BeamNGpy. Those non-ego cars implement regular traffic, adversarial scenarios (e.g., cut-in-lane), or critical situations (e.g., car crashes). As for the ego-car, various sensors can be attached to non-ego vehicles to monitor their status. Thanks to those sensors, BeamNG.tech fulfills the fundamental "observability" requirement of testing, allows the definition of test oracles, and can quickly collect large datasets for training and validation.

IV. BeamNG.tech Usage in Research, Industry & Education

BeamNG.tech in autonomous systems research. Autonomous driving is expected to improve transportation safety drastically. However, recently reported fatal crashes involving self-driving cars are signaling that, at least in the short-term, these expectations appear to be too optimistic. This situation calls for further research in the field to avoid releasing self-driving cars equipped with defective software, which might become erratic and possibly life-threatening. Given the danger and ineffectiveness of physical testing [25], researchers and practitioners devised simulation-based testing approaches for CPSs, which provides the opportunity to generate test scenarios automatically. Additionally, in simulations, the systems under test can face scenarios that might be otherwise too hard, too risky, or impossible to recreate in real life. However, automatic test generation comes with several open challenges of *what* constitutes useful tests and *how* to systematically generate them. The approaches reported in this section leverage BeamNG.tech to address those challenges. To the best of our knowledge, a comprehensive list of approaches include AsFault [14], [15], AC3R [12], [13], [21], and DeepJanus [33].

Testing CPSs with search-based procedural content generation. Testing a CPS requires the generation of test scenarios that effectively stress the system under tests. To systematically find critical test scenarios or test scenarios particularly challenging for the system under test, Gambi, Mueller, and Fraser combined search-based software testing and procedural content generation into a novel approach called AsFault. AsFault generates road models that stress self-driving cars' lane-keeping and leverages BeamNG.tech to implement them in automatically executed soft-body simulations. AsFault uses a genetic algorithm to mutate and recombine the road models and create increasingly challenging test scenarios that eventually cause the ego-car to drive off-road.

Generating critical test cases from police reports. Search-based test generation is a valid alternative to traditional scenario-based testing. It can generate challenging test scenarios that expose safety-critical issues in CPSs. However, it struggles to generate test scenarios with peculiar features, e.g., a rear-end car crash, that developers might need to validate or debug their implementations. Manually creating such test scenarios is time-consuming and cumbersome, so an automated solution is needed. To address this problem, Gambi, Huynh, and Fraser devised AC3R, an approach to derive scenario-based tests from simulations of real car crashes. AC3R leverages natural language processing and a custom ontology to extract information from police reports that describe car crashes. Next, it uses basic kinematics to plan the intercepting vehicles' trajectories. Finally, it uses BeamNG.tech to automatically simulate the environments, roads, and vehicles' trajectories that re-enact the car crashes. A large user study confirmed that the simulations generated by AC3R using BeamNG.tech accurately match the police reports' descriptions. Furthermore, an empirical evaluation using a vision-based driving AI showed the effectiveness of critical driving scenarios as test cases.

Automatically exploring Deep Learning systems' behavior frontier. Deep Learning (DL) systems learn from data, and traditional approaches to software quality cannot assess their reliability and generalization capabilities. Analyzing the behavior frontier is a viable solution to characterize DL systems

TABLE II
MAIN MODEL-BASED SIMULATIONS USED IN CPS RESEARCH AND THEIR MOST DISTINGUISHING FEATURES.

Simulator	Domain	Approach	Vehicles	Pedestrians	Environment	Deployment
AirSim	autonomous driving drones	rigid-body	cars drones	no	weather lighting	multi-OS docker
Carla	autonomous driving	rigid-body	cars	yes	weather lighting	multi-OS docker
BeamNG.tech	driving	soft-body	cars/trucks trailers/drones	no	weather lighting	windows
Gazebo	robotics	rigid-body	cars drones	no	N.A.	multi-OS docker
Udacity AV	autonomous driving	rigid-body	car	no	N.A.	multi-OS docker
Baidu Apollo	autonomous driving	rigid-body	cars	yes	weather lighting	multi-OS docker

by quantifying how much valid inputs can be modified before these systems misclassify them. Finding DL systems' behavior frontier is not easy. It requires identifying the boundary of the parameter space where the DL systems behave as expected and outside of which start to diverge. To address this problem, Riccio and Tonella proposed a model-based approach, DeepJanus, that identifies pairs of similar inputs that trigger different behaviors of the DL systems under test. Riccio and Tonella successfully applied DeepJanus to explore the behavior frontiers of a digit classifier trained on the MNIST dataset [27] and the NVidia's Dave2 [6] end-to-end steering controller. For exploring the steering controller's behavior frontier, DeepJanus relies on BeamNG.tech to procedurally generate the pairs of similar roads for which the steering controller keeps the car on the road and drives off the road.

BeamNG.tech in industrial projects and education. BeamNG.tech is a simulation framework for autonomous driving applications and driver training. It supports testing, data generation, and driving simulations all the way to research and commercial deployment. As a result of that, BeamNG GmbH has participated in various research projects and industrial collaborations, involving major players in the CPSs domain, such as Audi Electronics Venture GmbH and the German Research Center for Artificial Intelligence (DFKI). BeamNG.tech is also an emerging reality in education.[7] It is currently used as a learning tool at the University of Passau (Germany) and at the University of Zurich (Switzerland).

In summary, BeamNG.tech has already shown promising results from research and industrial enablers and as a tool for education, supporting our expectations that soft-body simulation will continue to be considered in future research initiatives in CPSs domains. In Section VI, we summarize the path to the future of soft-body simulation and BeamNG.tech in CPSs development and evolution.

[7]BeamNG.tech has been used in seminars, several students' projects, and theses. An incomplete list of submitted theses using BeamNG.tech is available at https://staff.fim.uni-passau.de/~gambi/#submitted-theses

V. RELATED WORK

In recent years we observed a growing number of CPS simulation environments for research and industry. These environments allow to perform model-based [24] and data-driven [4] simulations, and to pair them as co-simulations [31]. In this paper, we mainly discuss model-based simulations. Specifically, we consider model-based simulators that are released under a non-commercial license, hence promoting open research.

Model-based simulations rely on abstractions (i.e., the models) to simulate only the physical phenomenons that are relevant for their intended application (e.g., physics of solid objects for simulating vehicles). To our knowledge, the form of model-based simulation most widely adopted in industry and academia is rigid-body simulation (see Table II). In Table II, we list model-based simulators commonly used in academia and industry along with their most distinguishing features. This list does not aim to be exhaustive and contains the following model-based simulators: AirSim [7], Carla [11], BeamNG.tech [5], Gazebo [43], Udacity AV Simulator [35], and Baidu Apollo [41]. As it is possible to observe from Table II, most simulators target the autonomous driving domain. The only exceptions are AirSim, which includes drones, and Gazebo, traditionally used in Robotics. Additionally, BeamNG.tech targets the general domain of driving since it has been used to research autonomous and manual driving. All the simulators, but BeamNG.tech, implement rigid-body simulation, can be installed in multiple environments (e.g., Windows, Linux, Mac), and provide a dockerized version for quick use. In contrast, BeamNG.tech implements soft-body simulation and can be used only in Windows-based environments.[8] All the simulators come with at least one car model and allow for using custom vehicles. AirSim, Gazebo, and BeamNG.tech provide drone models, but only BeamNG.tech also includes trucks, trailers, and other vehicle models. The simulation of pedestrians is not yet widely supported. In contrast, the simulation of environmental conditions, such as

[8]A release of BeamNG.tech for Linux is under development.

weather and time-of-day (lighting in the table), is available in most simulators. All the simulators also provide an API that enables external programs to interact with the running simulations (e.g., controlling vehicles). However, only BeamNG.tech enables the procedural generation of test scenarios and creating assets and contents via editors.

VI. The Path Forward

We reflected on the advantages and disadvantages of soft-body and rigid-body simulations toward developing and testing CPSs. We discussed using soft-body simulations as a necessary complement to mainstream rigid-body simulations for simulating focused and safety-critical test scenarios. In addition to simulation aspects, we considered the procedural generation of virtual environments and test scenarios. We concluded that automation in simulation frameworks is vital to improving current CPS development and testing practices. We identified some aspects that require enhancements for realizing the vision of maintainable, evolvable, and reliable CPSs. We discuss below some critical innovations to achieve this vision:

- **DevOps testing pipelines for CPS**: minimizing the number of post-release failures experienced by CPSs requires understanding, modeling, and monitoring the specific failures occurring in them. The challenge of this activity concerns the definition of tools that allow identifying CPS behaviors in real-world scenarios. Thus, a related challenge to this problem is mitigating the risks of experiencing unexpected CPSs' behaviors in complex and dynamic environments. Our investigation highlighted the need to design DevOps testing pipelines that combine different simulation approaches (e.g., rigid- and soft-body) and support the whole development in an agile fashion.
- **Modularity & Serviceability**: novel strategies are needed to facilitate the adoption of continuous integration, testing, and deployment. Likewise, modularity and serviceability require better support during development and maintenance [23]. In the CPS domain, modularity refers to decoupling the (hardware/software) components and identifying small CPS units of functionality that can be developed, maintained, and tested in isolation. We observed how soft-body simulations could assess the behavior of those CPS units more realistically.
- **Test generation and procedural content generation**: research in Cyber-physical Systems already addressed automated test execution and reporting [1]; however, practitioners in the field also require means for automated verification and validation (V&V) [19], [22]. Novel V&V solutions must handle open challenges such as the automated generation of test scenarios for CPSs and their optimization [18], non-deterministic behaviors in CPSs and their environments, and humans-in-the-loop [23], [32], [37]. We argue that ensuring the reliability and safety of CPSs requires tailoring test generation around specific CPS domains. This way, the generated tests can meet complementary and necessary testing criteria (e.g., fault detection, code coverage, execution costs) [18].
- **Multi-domain support**: the research and industrial communities have produced simulation environments that could serve the purpose of testing CPSs. However, such environments (as discussed in Section II and Section V) come with severe limitations. One critical aspect concerns the need for those simulation environments to support application domains other than automated driving. Drones, mobile work machines, and, more in general, robotics systems are increasingly becoming relevant. Hence, research in those fields deserves the right support from state-of-art rigid-body and soft-body simulators.
- **Scenarios concerning users' needs**: research in autonomous driving systems has discussed the need to include user-relevant criteria in current simulations other than safety.[9] For instance, *motion sickness*, a phenomenon that frequently occurs in vehicles' passengers —not drivers— might become a show stopper for the acceptance of self-driving cars. Consequently, to avoid or reduce the effect of motion sickness on passengers, self-driving cars' design must also consider their comfort. Motion sickness is caused by sudden later and vertical forces acting on the passengers; it also depends on where and how the passengers sit inside the vehicles. So, dealing with it requires very accurate simulations (i.e., soft-body simulations) to study how self-driving car driving styles affect passengers' comfort.

VII. Acknowledgements

Sebastiano Panichella gratefully acknowledges the Horizon 2020 (EU Commission) support for the project *COSMOS* (DevOps for Complex Cyber-physical Systems), Project No. 957254-COSMOS).

References

[1] Automatic testing with ROS. http://wiki.ros.org/Quality/Tutorials/UnitTesting. Accessed: 2019-11-27.

[2] S. Abbaspour Asadollah, R. Inam, and H. Hansson. A survey on testing for cyber physical system. In K. El-Fakih, G. Barlas, and N. Yevtushenko, editors, *Testing Software and Systems*, pages 194–207, Cham, 2015. Springer Intern. Publishing.

[3] R. B. Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter. Testing autonomous cars for feature interaction failures using many-objective search. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 143–154. IEEE, 2018.

[4] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters*, 5(2):1143–1150, 2020.

[5] BeamNG GmbH. BeamNG.research. https://www.beamng.gmbh/research. Accessed: 2018-10-11.

[6] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End-to-end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[7] E. Bondi, D. Dey, A. Kapoor, J. Piavis, S. Shah, F. Fang, B. Dilkina, R. Hannaford, A. Iyer, L. Joppa, and M. Tambe. AirSim-w: A simulation environment for wildlife conservation with uavs. In E. W. Zegura, editor, *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies, COMPASS*, pages 40:1–40:12. ACM, 2018.

[9] http://dsc2018.org/Docs/Keynotes/BOS_KEYNOTE.pdf

[8] H. Chen. Applications of cyber-physical system: A literature review. *Journal of Industrial Integration and Management*, 02(03):1750012, 2017.

[9] CollabNet. Anti-patterns in the continuous delivery (cd) practice, 2014.

[10] M. Dalboni and A. Soldati. Soft-body modeling: A scalable and efficient formulation for control-oriented simulation of electric vehicles. In *IEEE Transportation Electrification Conference and Expo (ITEC)*, pages 1–6, 2019.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun. CARLA: an open urban driving simulator. In *1st Annual Conference on Robot Learning, CoRL 2017*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 2017.

[12] A. Gambi, T. Huynh, and G. Fraser. Automatically reconstructing car crashes from police reports for testing self-driving cars. In J. M. Atlee, T. Bultan, and J. Whittle, editors, *International Conference on Software Engineering: Companion Proceedings, ICSE*, pages 290–291. IEEE / ACM, 2019.

[13] A. Gambi, T. Huynh, and G. Fraser. Generating effective test cases for self-driving cars from police reports. In M. Dumas, D. Pfahl, S. Apel, and A. Russo, editors, *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 257–267. ACM, 2019.

[14] A. Gambi, M. Müller, and G. Fraser. Asfault: testing self-driving car software using search-based procedural content generation. In J. M. Atlee, T. Bultan, and J. Whittle, editors, *International Conference on Software Engineering: Companion Proceedings, ICSE*, pages 27–30. IEEE / ACM, 2019.

[15] A. Gambi, M. Müller, and G. Fraser. Automatically testing self-driving cars with search-based procedural content generation. In D. Zhang and A. Møller, editors, *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*, pages 318–328. ACM, 2019.

[16] V. Giavotto, M. Borri, P. Mantegazza, G. Ghiringhelli, V. Carmaschi, G. Maffioli, and F. Mussi. Anisotropic beam theory and applications. *Computers & Structures*, 16:403–413, 12 1983.

[17] C. A. González, M. Varmazyar, S. Nejati, L. C. Briand, and Y. Isasi. Enabling model testing of cyber-physical systems. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, page 176–186, New York, NY, USA, 2018. Association for Computing Machinery.

[18] G. Grano, C. Laaber, A. Panichella, and S. Panichella. Testing with fewer resources: An adaptive approach to performance-aware test case generation. *IEEE Trans. on Soft. Engineering*, pages 1–1, 2019.

[19] P. Helle, W. Schamai, and C. Strobel. Testing of autonomous systems - challenges and current state-of-the-art. *INCOSE International Symposium*, pages 571–584, 2016.

[20] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, page 235–245, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[21] T. Huynh, A. Gambi, and G. Fraser. AC3R: automatically reconstructing car crashes from police reports. In J. M. Atlee, T. Bultan, and J. Whittle, editors, *International Conference on Software Engineering: Companion Proceedings, ICSE*, pages 31–34. IEEE / ACM, 2019.

[22] F. Ingrand. Recent trends in formal validation and verification of autonomous robots software. In *3rd IEEE International Conference on Robotic Computing, IRC 2019, Naples, Italy, February 25-27, 2019*, pages 321–328, 2019.

[23] D. S. Johnson, H. Koehneman, D. LaFortune, D. Leffingwell, S. Magill, S. Mayner, A. Ofer, R. Stroud, A. Wallgren, and R. Yeman. *INDUSTRIAL DEVOPS - Applying DevOps and Continuous Delivery to Significant Cyber-Physical Systems*. National Academies Press, 2017.

[24] K. N. Junejo and D. K. Y. Yau. Data driven physical modelling for intrusion detection in cyber physical systems. In A. Mathur and A. Roychoudhury, editors, *Proceedings of the Singapore Cyber-Security Conference (SG-CRC) 2016 - Cyber-Security by Design, Singapore, January 14-15, 2016*, volume 14 of *Cryptology and Information Security Series*, pages 43–57. IOS Press, 2016.

[25] N. Kalra and S. Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 12 2016.

[26] J. Kim, S. Chon, and J. Park. Suggestion of testing method for industrial level cyber-physical system in complex environment. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, Apr. 2019.

[27] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[28] L. Li, W. Huang, Y. Liu, N. Zheng, and F. Wang. Intelligence testing for autonomous vehicles: A new approach. *IEEE Transactions on Intelligent Vehicles*, 1(2):158–166, 2016.

[29] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2020.

[30] J. Mesit and R. K. Guha. A general model for soft body simulation in motion. In S. Jain, R. R. C. Jr., J. Himmelspach, K. P. White, and M. C. F. 0001, editors, *Winter Simulation Conference*, pages 2690–2702. IEEE, 2011.

[31] D. Nalic, A. Eichberger, G. Hanzl, M. Fellendorf, and B. Rogic. Development of a co-simulation framework for systematic generation of scenarios for testing and validation of automated driving systems. pages 1895–1901, 10 2019.

[32] S. Panichella, A. D. Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In R. Koschke, J. Krinke, and M. P. Robillard, editors, *International Conference on Software Maintenance and Evolution, ICSME*, pages 281–290. IEEE Computer Society, 2015.

[33] V. Riccio and P. Tonella. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE '20, page 13 pages. Association for Computing Machinery, 2020.

[34] H. Shokry and M. Hinchey. Model-based verification of embedded software. *Computer*, 42(4):53–59, April 2009.

[35] M. V. Smolyakov, A. I. Frolov, V. N. Volkov, and I. V. Stelmashchuk. Self-driving car steering angle prediction based on deep neural network an example of carnd udacity simulator. In *2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–5, 2018.

[36] S. Sontges and M. Althoff. Computing the drivable area of autonomous road vehicles in dynamic road scenes. *IEEE Trans. Intell. Transp. Syst.*, 19(6):1855–1866, 2018.

[37] A. D. Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall. What would users change in my app? summarizing app reviews for recommending software changes. In T. Zimmermann, J. Cleland-Huang, and Z. Su, editors, *International Symposium on Foundations of Software Engineering*, pages 499–510. ACM, 2016.

[38] The Guardian. Self-driving uber kills arizona woman in first fatal crash involving pedestrian, 2018.

[39] The Washington Post. Uber's radar detected elaine herzberg nearly 6 seconds before she was fatally struck, but "the system design did not include a consideration for jaywalking pedestrians" so it didn't react as if she were a person., 2019.

[40] M. Törngren and U. Sellgren. *Complexity Challenges in Development of Cyber-Physical Systems*, pages 478–503. Springer International Publishing, Cham, 2018.

[41] J. Xu, Q. Luo, K. Xu, X. Xiao, S. Yu, J. Hu, J. Miao, and J. Wang. An automated learning-based procedure for large-scale vehicle dynamics modeling on baidu apollo platform. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 5049–5056. IEEE, 2019.

[42] E. Zapridou, E. Bartocci, and P. Katsaros. Runtime verification of autonomous driving systems in carla. In J. Deshmukh and D. Ničković, editors, *Runtime Verification*, pages 172–183, Cham, 2020. Springer International Publishing.

[43] M. Zhang, H. Qin, M. Lan, J. Lin, S. Wang, K. Liu, F. Lin, and B. M. Chen. A high fidelity simulator for a quadrotor UAV using ROS and gazebo. In *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society, Yokohama, Japan, November 9-12, 2015*, pages 2846–2851. IEEE, 2015.