Asher Elmquist
Department of Mechanical Engineering,
University of Wisconsin-Madison,
Madison, WI 53706
e-mail: amelmquist@wisc.edu

Radu Serban
Department of Mechanical Engineering,
University of Wisconsin-Madison,
Madison, WI 53706
e-mail: serban@wisc.edu

Dan Negrut[1]
Professor
Department of Mechanical Engineering,
University of Wisconsin-Madison,
Madison, WI 53706
e-mail: negrut@wisc.edu

A Sensor Simulation Framework for Training and Testing Robots and Autonomous Vehicles

Computer simulation can be a useful tool when designing robots expected to operate independently in unstructured environments. In this context, one needs to simulate the dynamics of the robot's mechanical system, the environment in which the robot operates, and the sensors which facilitate the robot's perception of the environment. Herein, we focus on the sensing simulation task by presenting a virtual sensing framework built alongside an open-source, multi-physics simulation platform called Chrono. This framework supports camera, lidar, GPS, and IMU simulation. We discuss their modeling as well as the noise and distortion implemented to increase the realism of the synthetic sensor data. We close with two examples that show the sensing simulation framework at work: one pertains to a reduced scale autonomous vehicle and the second is related to a vehicle driven in a digital replica of a Madison neighborhood. [DOI: 10.1115/1.4050080]

Keywords: exteroceptive sensors, proprioceptive sensors, simulation and design, vehicle autonomy

1 Introduction

1.1 Motivation and Contribution. In the fields of robotics and autonomous vehicles (AVs), simulation can play a key role in understanding how candidate designs respond when placed in a myriad of operational scenarios. The motivating factor of the research described herein is that of improving the realism of the simulation, so that the behavior of the virtual robot in simulation is similar to that of the physical robot in the real world. In other words, we seek to reduce the so-called simulation-to-reality gap [1].

Herein, we outline a sensing simulation module that has been built into a multi-physics simulation framework called Chrono [2,3]. This new module, called Chrono::Sensor, generates synthetic data for camera, lidar, GPS, and IMU when any of these sensors is immersed in a virtual environment whose time evolution is simulated via Chrono. By combining dynamics simulation and sensing simulation, we seek to establish a comprehensive analysis platform for complex robotic systems operating in unstructured, complex, and dynamic environments.

The primary objective of Chrono::Sensor is to provide realistic synthetic sensor data for use in the investigation, development, testing, and optimization of robots through simulation. The interest is in generating synthetic data streams that reflect the distortions and noise levels encountered in the actual sensor data. For instance, while *photorealism* can be an important aspect of a virtual environment, Chrono::Sensor camera simulation is concerned primarily with *data realism* such that the perception and control algorithms, not humans, find the synthetic data indistinguishable from real data. It is important to note that photorealism and data realism are not mutually exclusive, but unlike video gaming, the emphasis here is on data realism. Moreover, we seek a solution that is versatile (supports multiple sensors); efficient (soft-real time targeted); and accurate in order to provide a simulation-in-robotics experience that demonstrates effective sim-to-real transferability attributes.

In this contribution we (*i*) introduce a sensing library built alongside an open-source, multi-physics simulation framework for the purpose of simulating complex robotic systems and AVs; (*ii*) describe an extensible and robust sensing framework for implementing sensor noise and distortion models to improve synthetic data realism; and (*iii*) outline the methods and models currently supported in the Chrono::Sensor for generating synthetic sensor data from a dynamic virtual environment. To that end, the article is structured as follows. Section 1.2 overviews other simulation frameworks used for autonomous agent analysis. Our sensor simulation framework is described from a software implementation point of view in Sec. 2. The sensor models are outlined in Sec. 3. The document concludes with two demonstrations of Chrono::Sensor usage in Sec. 4. Specifically, we use a detailed vehicle model and equip it with multiple sensors to either evaluate a navigation algorithm or generate data for offline testing of autonomous systems. Ultimately, this contribution concerns sensor simulation models, and not the specific use of sensor simulation in particular applications, a very important task which nonetheless is too broad to be approached here. Section 5 presents a conclusion and future work is discussed.

1.2 Related Work. One of the most widely used robotics simulation platforms is Gazebo [4–6] owing to the fact that it is free, open source, it is closely integrated with the robot operating system (ROS), it offers a breadth of sensors, and it interfaces to several dynamics engines (Bullet [7], ODE [8], DART [9], and Simbody [10]). On the downside, it lacks depth in relation to: physics modeling (i.e., flexible bodies, fluid–solid interaction, deformable terrains, etc.); and sensor sophistication. CoppelliaSim (previously known as V-REP) [11] follows a similar paradigm to Gazebo, with which it shares a similar set of strengths and weaknesses. The major additional weakness of CoppelliaSim is that it is not provided free and open source. Recently, NVIDIA has released a robot simulation platform called ISAAC Sim [12], which leverages the PhysX dynamics engine [13] to support robot simulation. While ISAAC Sim is perhaps the most modern simulation software for robotics, its closed-source nature introduces a barrier to adoption and extension. Additionally, since little is published on their sensor models and dynamics validation, it is difficult to judge its ability to simulate complex scenarios. Additional frameworks exist for robot simulation, including Webots [14] and USARSim [15], but they largely lack the feature set and/or fidelity of the aforementioned frameworks, particularly so for unstructured, off-road conditions. Beyond this list, many frameworks focus on dynamics and have sought to add sensing capabilities to broaden

[1]Corresponding author.

Contributed by ASME for publication in the JOURNAL OF AUTONOMOUS VEHICLES AND SYSTEMS. Manuscript received August 11, 2020; final manuscript received February 3, 2021; published online February 23, 2021. Assoc. Editor: Lutz Richter.

the scope of their simulation capabilities. These include frameworks such as PyBullet [16] and MuJoCo [17].

For on-road AV navigation simulation, the most broadly utilized platforms are Carla [18] and AirSim [19]. Both are developed on top of Unreal Engine [20], with AirSim also supporting use of Unity three-dimensional (3D) [21]. Unity 3D and Unreal Engine are game development platforms that have seen broad use in AV and robot simulation. In video gaming, vehicle dynamics model fidelity is of less concern than simulation speed. Since they build on gaming engines, Carla and AirSim can provide photo-realistic images and support complex and artistically generated virtual environments. In terms of sensing, photorealism is important but not necessarily sufficient for camera simulation. Moreover, the ray-casting implementations in Carla and AirSim used for lidar simulation are low-fidelity methods that limit the realism of the synthetic point clouds. Beyond visual sensors, AirSim does provide a detailed approach for generating realistic noise and distortion for dynamics-based sensors including GPS, IMU, barometer, and magnetometer. While these simulators represent some of the most capable on-road frameworks, they are unable to support off-road environments with more complex dynamics, e.g., deformable terrain, which is important in capturing the vehicle–environment interplay for terrains with mud, sand, snow, etc. Two simulators for off-road scenarios are VANE [22] and MAVS [23], with the later leveraging Chrono for vehicle dynamics support. VANE is closed source, but published literature [24–27] illustrates the capabilities of the framework for military applications. MAVS provides high-fidelity lidar simulation [28,29] but it is not freely available as open source. A real-time desktop framework for robotic simulation, called ANVEL [30], is also designed for military applications and is tightly developed alongside VANE. Finally, Vortex [31] offers a real-time simulation solution with support for AV simulation.

Other frameworks for AV simulation include AutonoVi [32], PreScan [33], SynCity [34], rfPro [35], VIRES [36], dSPACE [37], and NVIDIA DRIVE Sim [38]. Most of these frameworks are commercial and closed-source, with limited breadth of published use to show simulation results. Relatively little information is available for the closed-source platforms to determine the level of sophistication and realism of the synthetic sensor data and dynamics support.

2 Simulation Framework

Chrono::Sensor extends the Chrono dynamics engine to provide a sensing simulation solution for robotic systems operating in complex and unstructured environments. Chrono::Sensor generates synthetic data based on the choice of number and type of sensors (RGB camera, lidar, GPS, and IMU) and passes these synthetic data to the robot control stack to determine the next set of robot commands. Based on this set of commands, the dynamics engine determines the evolution of the robot and its interaction with the environment. This sequence: sensing → command → dynamics, loops for the duration of the simulation, with specific frequencies of the control stack, sensing, and dynamics maintained according to real-world hardware (sensing and control) and physical requirements (dynamics). An example is shown in Fig. 1, where an instrumented all-terrain vehicle is interacting with the environment by deforming the terrain on which it operates. The scene is shown from the perspective of a third-person, Chrono::Sensor camera.

Fig. 1 Example of an off-road autonomous vehicle navigating a complex and deformable environment

2.1 Simulation of Physics with Chrono. Herein, the discussion of Chrono is limited to covering the components relevant to the sensor simulation. An overview of Chrono can be found in Ref. [2]. The Chrono dynamics engine supports rigid and flexible body dynamics with additional simulation modules for granular mechanics and fluid–solid interaction. Through the Chrono::Vehicle module [39], Chrono provides template-based support for specifying the topology of a vehicle from subsystems such as suspension, tires, steering mechanism, powertrain, and track subsystem. This allows a user to build a model from designed or measurable parameters rather than requiring construction of each individual body and kinematic constraint. Figure 2 illustrates the components of a Chrono::Vehicle suspension model. All of these components, including other moving obstacles, robot joints, vehicles, or agents, are visible to the sensors in Chrono.

Chrono can simulate vehicle–terrain interaction in complex off-road environments with deformable terrains [40]. While Chrono's primary objective is high-fidelity modeling, which might come at a high computational cost, simulations with trivial-to-compute dynamics; i.e., a vehicle in an on-road maneuver operating on rigid terrain, can be performed faster than real-time. This paradigm of "as fast as possible" simulation carries over to Chrono::Sensor, where there is no hard constraint on the simulation to be real time—instead, it is expected to run as fast as possible with real-time being an arbitrary line that comes into play for hardware/human-in-the-loop scenarios. In fact, for many applications in machine learning there is significant benefit to faster-than-real-time simulation. An example of reinforcement learning for off-road navigation is discussed in Ref. [41]. Therein, off-road navigation using a military convoy was learned in Chrono using reinforcement learning, and then demonstrated in simulation on deformable soil in off-road environments with silt-like and snow-like terrains.

2.2 Simulation of Sensing with Chrono::Sensor. Chrono::Sensor communicates with the Chrono dynamics engine to collect information about how the robot changes the environment and the environment changes the robot. Then, based on the set of sensors endowed upon the robot by the user, Chrono::Sensor generates synthetic data from the virtual world. In the case of the camera and lidar sensors, the synthetic data are generated via GPU-based ray-tracing, leveraging hardware accelerated support [42], and the headless rendering capabilities provided by OptiX [43]. Ray tracing allows the sensor models to closely mimic the data acquisition of light-detecting sensors (i.e., lidar and camera). While our solution relies on OptiX as a ray-tracing library, OptiX itself does not simulate cameras or lidars, but rather allows this simulation framework to model physics-based data acquisition using ray tracing. For IMU and GPS data, the Chrono system is queried and the ground-truth simulated data are augmented to form the distorted and noisy data that would be generated from a real sensor. The sensor framework is closely coupled with Chrono, but is extensible through a filter-graph that is applied to each sensor in a post-process step represented in Fig. 3. The figure shows the general implementation applied to each sensor, which follow similar flows to generate data, augment the data, and then return the data to the user. Each step can include specific models for the sensors, with an interface to implement and add custom filters to the framework.

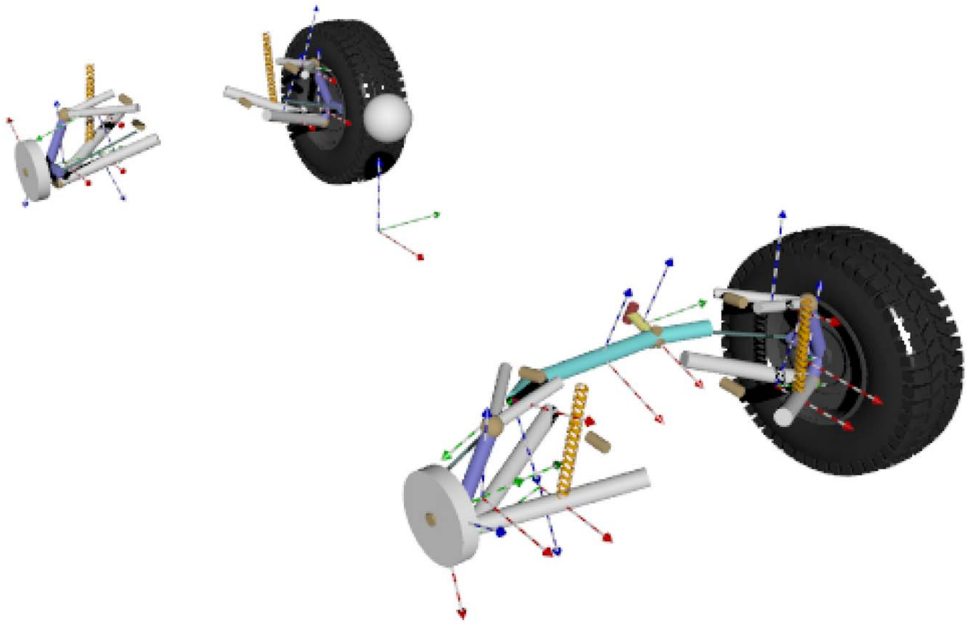

Fig. 2 Vehicle assembly showing the simulated components of a Chrono::Vehicle suspension

This filter graph is used for implementing specific models, as discussed in Sec. 3.

Due to the dynamic simulation requiring a much higher update frequency than sensor simulation (dynamics: order 1 kHz; sensors: order 10–100 Hz), the sensor framework uses a separate thread to manage the rendering and synthetic data curation. This thread manages a process in which multiple dynamics engine simulation steps complete during a single render step, yet there is time coherency in that neither the dynamics nor the sensing drift apart in time. An illustration of this is shown in Fig. 4. In this paradigm, a defined frequency of a sensor, e.g., 50 Hz, will be guaranteed to maintain 50 Hz in simulation time regardless of whether the computational bottleneck is sensing or dynamics related. When a system includes many sensors, and when more graphics hardware is available, Chrono::Sensor can leverage multiple render threads each managing a separate GPU for simulating a subset of the sensors present in the virtual test. This is particularly useful for scenarios with many agents with numerous sensors that also operate at various update frequencies in a highly complex virtual environment. Results that characterize that performance and scaling of Chrono::Sensor can be found in Ref. [44].

For camera and lidar simulation, Chrono::Sensor does not need to push an image to the display when simulating robots or AVs. Since no image needs to be rendered on a screen, the screen-based graphics contexts are bypassed altogether, resulting in "headless rendering." Due to the headless nature of Chrono::Sensor, many parallel simulations can be launched in unison on remote compute hardware. In addition, the framework is developed to be middleware, allowing the user to link their C++ application directly to Chrono or make direct calls to Chrono through the PyChrono module rather than set up a client-server model as would be needed for game engines like Unity and Unreal. These features are particularly relevant for reinforcement learning problems, when the computation associated with the simulation component is intensive and one would like to launch tens of simulations at the same time, using a multi-core chip or multiple nodes in a cluster.

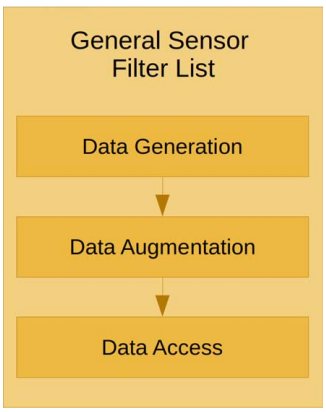

Fig. 3 Visualization of the filter graph implemented in Chrono::Sensor. The basic components are data generation (using ray tracing for camera and lidar), data augmentation (e.g. noise, point cloud conversion, color space conversion, etc.), and data access (including lag). Custom filters can be applied in any of the three above steps.

3 Sensor Models Implemented in Chrono::Sensor

3.1 The Lidar Model. The lidar sensor available in Chrono::Sensor is based on scanning time-of-flight sensors frequently used in automotive and robotic applications. The model implementation leverages the GPU OptiX library since lidar is fundamentally a ray tracing measurement. Each ray in the lidar is launched in parallel using the GPU and is traced with its own origin acting as its sole light source. The diffuse reflectance model based purely on geometry is used to calculate the relative intensity of the return. The implementation is flexible enough to allow custom materials that encode lidar reflectance on a per-object or per-triangle basis, analogous to any camera-based graphics model. For this to be useful, the reflectance of the object must be measured in the lidar-specific wavelength and attached to the Chrono body.

To accurately characterize the range and intensity of the lidar returns, and to generate physically realistic artifacts in the measurements, modeling and configurable quantities are provided by the user. Foremost are the basic parameters that define the quantity and update frequency of the lidar data: horizontal field of view, vertical field of view, scanning frequency, channels, angular resolution, and maximum distance. For modeling temporal artifacts, lag and collection time parameters are included. Lag defines the delay between when the data are measured and the time at which the

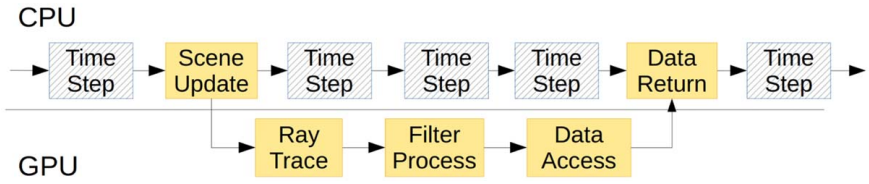

Fig. 4 Since camera and lidar are simulated on the GPU at lower frequencies than dynamics, GPU computation is done in parallel with dynamics time-steps, and data are returned between time-steps as defined by the lag setting of the sensor. The hashed boxes refer to Chrono dynamics time-steps, solid boxes refer to Chrono:: Sensing processes.

user receives these data. While this importance is often overlooked by sensor simulation environments, it can play a critical role when understanding time-evolutionary behavior of a control stack. The lidar collection time is the time window over which the data are collected. For a 360 deg scanning lidar, this is typically equal to the update period or 1/UpdateFrequency.

When the collection window is greater than 0, the movements of the sensor and scene relative to the sensor are taken into account using interpolation between keyframes. Building the scene from keyframes and using interpolation of the sensor position allow each lidar beam to be traced at the correct instant resulting in fine precision artifacts such as lengthening and shortening of quickly moving objects such as vehicles. This interpolation generates a continuous scan, avoiding non-physical artifacts that are often introduced when using ray-casting from within game engines. A comparison of the lidar data from an instantaneous model and from a continuous model is shown in Fig. 5, with data generated using Chrono::Sensor. The data correspond to a simulated experiment where a lidar is moved at 20 m/s along a path lined with cylindrical obstacles of 1 m diameter at 2 m intervals. Since the lidar is scanning in a counter-clockwise direction, the cylinders on the right appear shorter than the cylinders on the left.

In addition to temporal distortion modeling, the lidar implementation allows for beam discretization when the divergence angle of the lidar rays plays a significant role in the acquired data. This is the case in many automotive applications where medium to long range lidar are common. Furthermore, manufacturers can leverage beam divergence to detect multiple returns, effectively increasing the number of points in the resulting point cloud. An illustration of this process is shown in Figs. 6 and 7, as modified from Ref. [45]. This is an effect that is not possible to reproduce effectively with single ray casting. The sampling model is illustrated in Fig. 8, where the radius (R) of the samples is user-defined. Since Chrono::Sensor uses GPU-accelerated ray tracing, the discrete samples for each beam are all launched in parallel and reduced after launch, resulting in a sub-linear scaling of simulation time with respect to discrete samples.

With the raw data generated, noise is modeled based on the measurement uncertainty. Rather than applying a Gaussian noise independently to x, y, z and intensity values of the point cloud, the noise is applied to the range, angular, and intensity measurements of the lidar, and mapped into noise on the point cloud. The noise parameters can be estimated from the accuracy presented on most lidar data sheets or through calibration data collected from a known object.

As a final step, the data are provided to the user based on a parameterized lag. This models the processing interval of the lidar sensor such that the user will obtain data at an appropriately delayed time. This is a physical artifact with which perception and navigation algorithms must contend, so to accurately train or evaluate a control stack, the simulation should account for the time delay of received data.

The lidar model is set up to allow the physics-based characteristics to be based on quantities found in a datasheet. These can include frequency, range, channels, angular resolution, and beam divergence. To further improve the model, noise on the distance and angular measurements would need calibration data to be estimated. The accuracy of these models can significantly change the generated point clouds. Accurately capturing noise can improve the realism of the sensor, but quantification of that improvement remains an open research question.

Since the virtual environment plays an important role in the simulation of lidar data, the resolution of the 3D environment should be fine enough to allow geometric characteristics down to the scale of the beam divergence. Additionally, material properties, which are

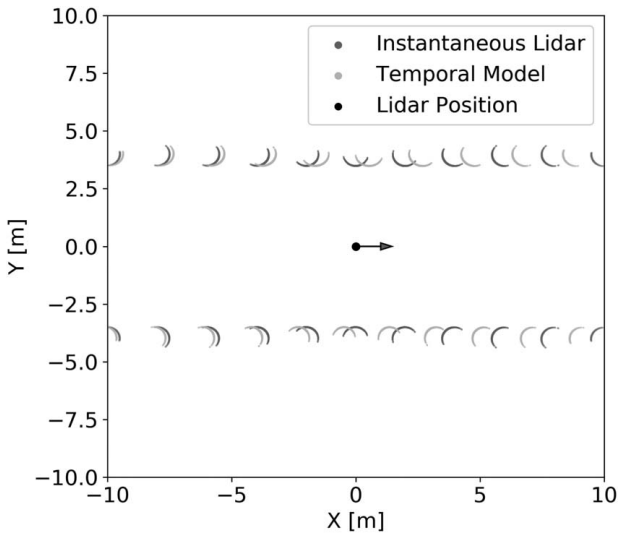

Fig. 5 Example of difference in point cloud when temporal distortions are considered. The lidar parameters are set for a Velodyne HDL-32E with a frequency of 20 Hz. The lidar is moved at a constant velocity of 20 m/s in the positive x direction (direction of arrow) with cylinders of 1 m diameter at two meter intervals placed on either side.

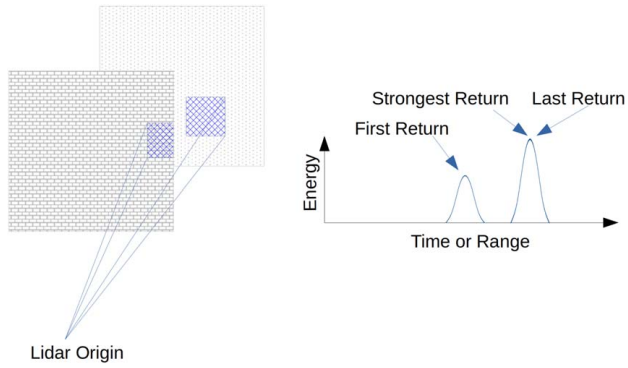

Fig. 6 Beam divergence used to detect multiple objects depending on return mode. Single ray cast would return a single measurement detecting the more distant wall. Modified from Ref. [45].

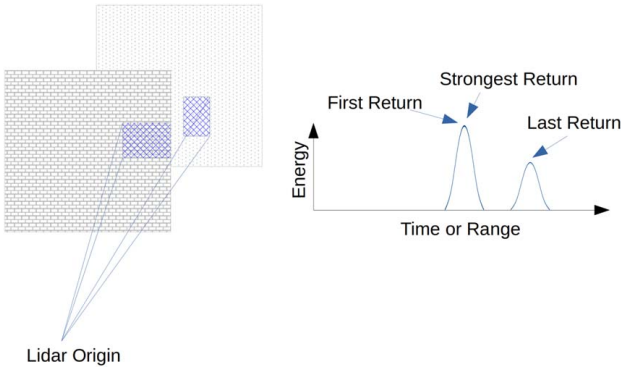

Fig. 7 Beam divergence used to detect multiple objects depending on return mode. Single ray cast would return a single measurement detecting the nearer wall. Modified from Ref. [45].

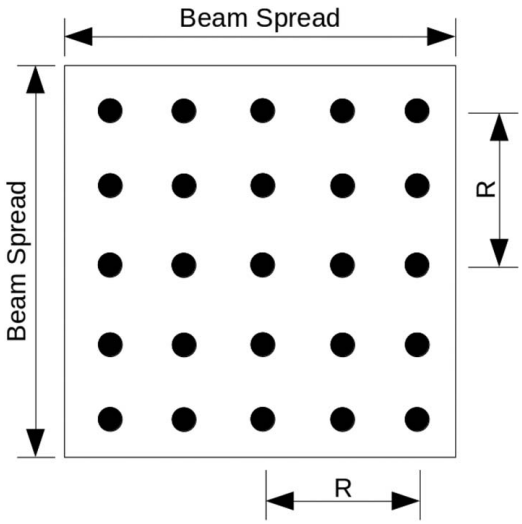

Fig. 8 Beam discretization model in Chrono::Sensor allows for a single beam to be traced by a user-defined number of rays, parameterized by the radius (*R*) of the samples in the sampling pattern

the subject of ongoing development, will be introduced to allow the user to specify reflectance in the lidar wavelength.

3.2 Camera Model. For simulating a typical RGB camera, Chrono::Sensor takes in several user-defined parameters: image size (height, width), frequency, horizontal field of view (FOV), exposure time, lag, and lens model type. These parameters are chosen for simple lens models as they are typically specified in camera data sheets, allowing the model to be physics-based rather than empirically based on calibration data. Additionally, the user can define computer graphics settings such as super-sampling factor and ray-tracing depth to improve the rendering fidelity or render time. Just like the lidar simulator, the camera simulation method uses the OptiX library, leveraging real-time-ray-tracing techniques. The rendering accounts for material and lighting of the environment. The simulation pipeline is built to mirror the image acquisition process [46], see Fig. 9. Specifically, in terms of computer graphics models, Chrono::Sensor implements physics-based rendering which includes reflectance, refractance, and Cook-Torrance [47] lighting models.

For the lens component of the pipeline, beyond the classical pinhole model, Chrono::Sensor implements a wide-angle-lens model based on the geometric considerations of a single spherical lens. The model draws on an approach discussed in Refs. [48,49], and follows from

$$r_2 = \frac{\tan(r_1 \tan(\omega))}{\tan \omega} \tag{1}$$

where ω is equal to half of the FOV of the camera, r_1 is the undistorted radius of the pixel, and r_2 is the distorted radius of the pixel. To allow this model to be based on an empirical FOV, this equation is modified to form the following

$$\begin{aligned} r_1 &= \sqrt{(x_1^2 + y_1^2)}, \quad r_2 = \frac{\tan(r_1 \tan(\omega))}{\tan \omega} \\ s &= \frac{\tan(\tan(\omega))}{\tan \omega}, \quad x_2 = \frac{x_1 r_2}{r_1 s}, \quad y_2 = \frac{y_1 r_2}{r_1 s} \end{aligned} \tag{2}$$

where x_1 and y_1 are the horizontal and vertical coordinates of undistorted radius r_1, and x_2 and y_2 are the horizontal and vertical coordinates of distorted radius r_2. Since the model augments a ray-tracing operation, the distortion is applied directly to the direction of launched rays to minimize post-processing overhead. An example of the FOV distortion is shown in Fig. 10 with ω equal to 0.704. The example shows that the horizontal FOVs for both lens models are identical.

To account for another important distortion, motion blur is factored in at the ray-tracing level. To mimic the cause of the distortion, the interval over which data are collected is determined by the user-defined exposure time. Using keyframes for the scene and camera position, each pixel is super-sampled with random launch times within the exposure window. Based on the ray's random time, the scene can be queried through the entire exposure time, interpolating the entire set of object positions from the scene and camera keyframes. This method introduces blur that captures the effects of a moving camera, as well as changes or movements in the environment.

To model the image measurement noise, Chrono::Sensor can add pixel-wise intensity-dependent noise, motivated by the EMVA standard [50] and subsequent variations, including the model described in Ref. [51] by

$$\sigma^2(p) = \overbrace{\frac{\Phi(p)t}{g^2} + \frac{\sigma_{read}^2}{g^2}}^{\text{pre-amplifier}} + \overbrace{\sigma_{ADC}^2}^{\text{post-amplifier}} \tag{3}$$

where p is the individual pixel; $\sigma^2(p)$ parameterizes the zero mean Gaussian distribution used for the specific pixel; $\Phi(p)$ is the radiant flux for the pixel; t is the exposure time; σ_{read}^2 is the read noise variance; σ_{ADC}^2 is the quantization variance produced by analog to digital conversion; and g is the user defined or computed

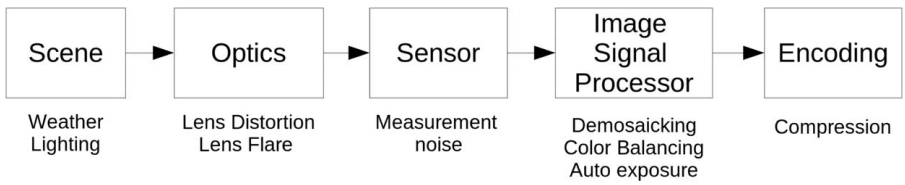

Fig. 9 The image acquisition process, according to Ref. [46]. The camera simulation pipeline mirrors the same components, and builds off models of each component.

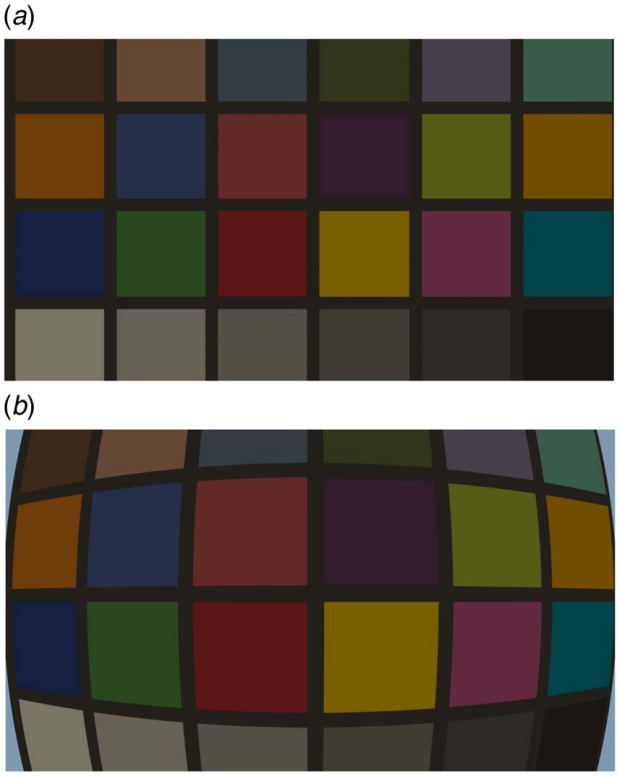

Fig. 10 Example of the difference between images generated with pinhole model and FOV lens model in Chrono::Sensor: (*a*) image generated with the pinhole lens model and (*b*) image generated with the field-of-view lens model

gain. The product $\Phi(p)t$ is then the photon intensity measured by the pixel and proportional to the preprocessed image intensity. With constant gain g and fixed exposure time t, the model reduces to a zero-mean Gaussian parameterized by the intensity of the pixel and two user-defined or measured parameters:

$$\sigma^2(p) = \sigma_1^2 I(p) + \sigma_2^2 \tag{4}$$

where σ_1 and σ_2 model the linear dependence of variance on image intensity. Equation (4) is the explicit model implemented in Chrono::Sensor. The parameters to this model can be calibrated from data measurements. Development is ongoing to implement an EMVA-compliant model which can be parameterized based on manufacturer-specified values provided in the camera data sheet.

While these noise models account for the sensor-level measurement error, they do not factor in the effects of the image signal processor (ISP), which often correlates the noise spatially, temporally, and chromatically [52–54]. Figure 11 illustrates why the EMVA noise model, while accurate at the image sensor level, does not fully model the final noise on a processed image such as an RGB JPEG image. Further work will focus on implementing components of the ISP to correctly augment the standard noise, and validate the final model.

As a final step in the camera simulation process, the data are made available to the user only after the lag time has elapsed. Similar to a lidar, the camera processes the measured data and provides sensed information to the user after a finite amount of time. This is modeled in Chrono::Sensor to account for inevitable temporal error that may factor into perception algorithm choices and pose estimation.

Many of the camera model parameters can be found in sensor data sheets including field-of-view, frequency, resolution, and exposure time. For other parameters such as noise levels for the implemented models, calibration experiments often need to be performed. A common noise calibration method is the mean-image method to estimate a noise distribution that can be used to fit a Gaussian or intensity-dependent model. An example noise estimation was performed in a related work [44].

The camera model relies on having a sufficiently realistic virtual environment, with 3D geometry of a resolution and fidelity similar to that found in the gaming community. For both lidar and camera, the visual properties must be specified to the same level as a video game, including colors, reflectance, refractance, texture and normal maps, and other lighting characteristics. The precise level of sensor realism is a function of both the virtual environment and the sensor model. Additionally, the intended use case (i.e., perception training, development testing, algorithm certification, etc.) will drive the level of realism required.

3.3 GPS and IMU. The GPS and IMU models are parameterized by a set of intrinsic values including update frequency, collection time, and lag. These sensors can be attached to any object within the Chrono simulation and can have an arbitrary attachment position and orientation. Therefore, these sensors can be placed on or in any location of a vehicle or robot including wheels, robotic arms, etc. These sensors make use of the dynamic quantities computed for the body by the dynamics engine to determine the proprioceptive data of interest. Even though GPS and IMU do not have a direct equivalence to the exposure time of a camera, a collection window is still used as a parameter for the IMU and GPS sensor models. This helps in two respects. First, when the sensor does signal collection over a finite amount of time as with GPS, the simulation can capture this trait. Second, because the simulation steps through time using numerical integration, the acceleration values can often be much noisier than in reality. Then, the collection

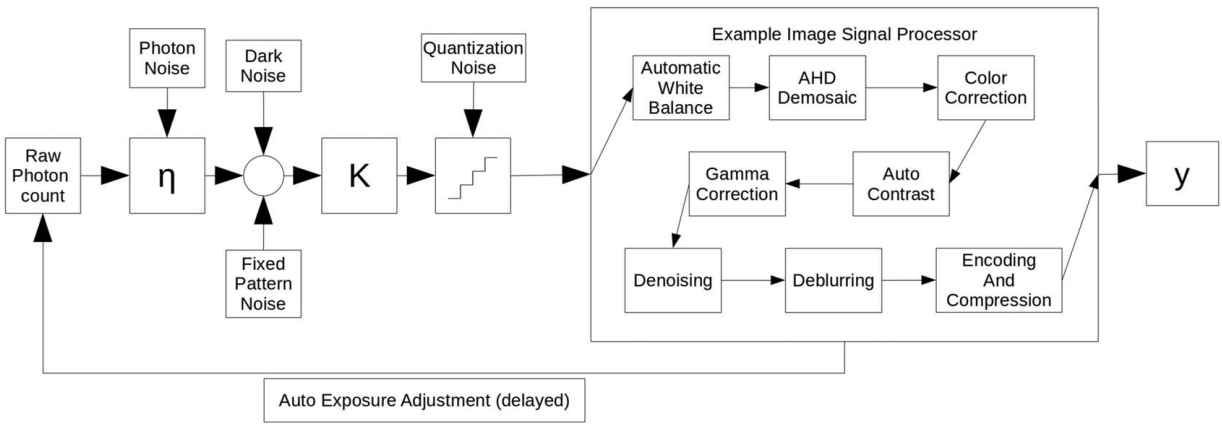

Fig. 11 Full noise factors including ISP which introduces significant noise correlation. Modified from the EMVA noise model [50] and the ISP example described in Ref. [55].

Fig. 12 Demonstration of sensors used to control a scaled autonomous vehicle in simulation. The vehicle uses lidar data to navigate a closed course without prior knowledge of the track.

Fig. 13 Demonstration of sensors used to control a scaled autonomous vehicle in simulation. While this image is not used in control, it represents a gray-scale image from a front-mounted camera that could be used in navigating the course.

Fig. 15 The vehicle shown is operated by a human in the loop and serves as a data collection vehicle within the virtual environment. The third-person perspective is for reader context; it is also used by the student driving the vehicle in the virtual neighborhood of Madison, WI.

Fig. 16 The front facing camera, in combination with other cameras and lidars, can be used as an input for software-in-the-loop simulation to test autonomous control algorithms

time provides an opportunity to smooth the ground truth data used in the sensor model.

While all sensors allow the user to implement a customized post-processing noise model, both the GPS and IMU allow for the addition of simple parameterized Gaussian drift models following

$$\omega_{out} = \omega + \eta_a + b_t, \quad \eta_a \sim N(\mu_a, \sigma_a^2)$$
$$b_t = b_{t-1} + \eta_b, \quad \eta_b \sim N\left(\mu_b, b_0\sqrt{\frac{dt}{t_b}}\right) \tag{5}$$

where ω_{out} is the final measure of angular velocity produced by the gyroscope, ω is the ground truth angular velocity, and μ_a, μ_b, σ_a, b_0, dt, and t_b are user-defined parameters of the noise model. The noise model for the accelerometer follows that for the gyroscope while additionally accounting for the gravitation offset. The model is given by

$$a_{out} = (a - g) + \eta_a + b_t, \quad \eta_a \sim N(\mu_a, \sigma_a^2)$$
$$b_t = b_{t-1} + \eta_b, \quad \eta_b \sim N\left(\mu_b, b_0\sqrt{\frac{dt}{t_b}}\right) \tag{5}$$

where a_{out} is the final measure of translational acceleration produced by the accelerometer, a is the ground truth translational acceleration, g is the gravitational constant, and μ_a, μ_b, σ_a, b_0, dt, and t_b are user-

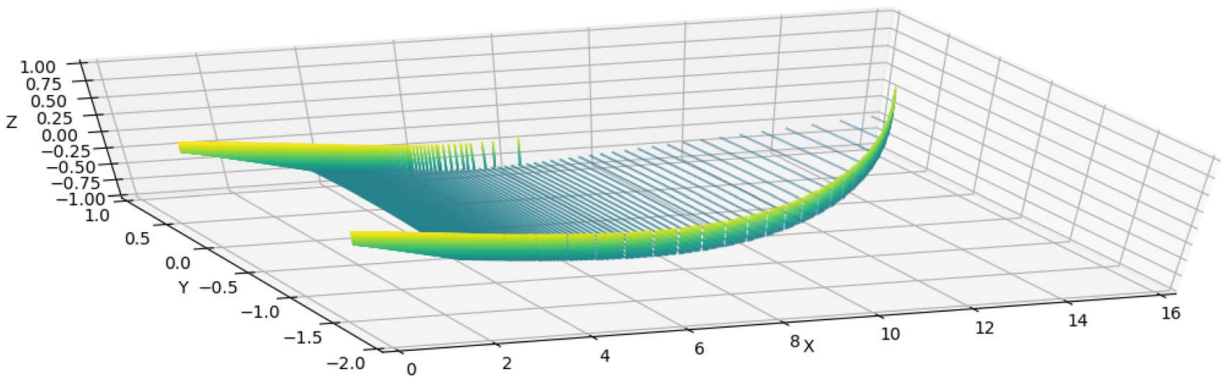

Fig. 14 Scaled autonomous vehicle navigation demonstration. The lidar-generated point cloud shown is used by the vehicle to navigate the course. For user readability, the intensity encodes height in the point cloud (the intensity used for visualization is not associated with the lidar model).

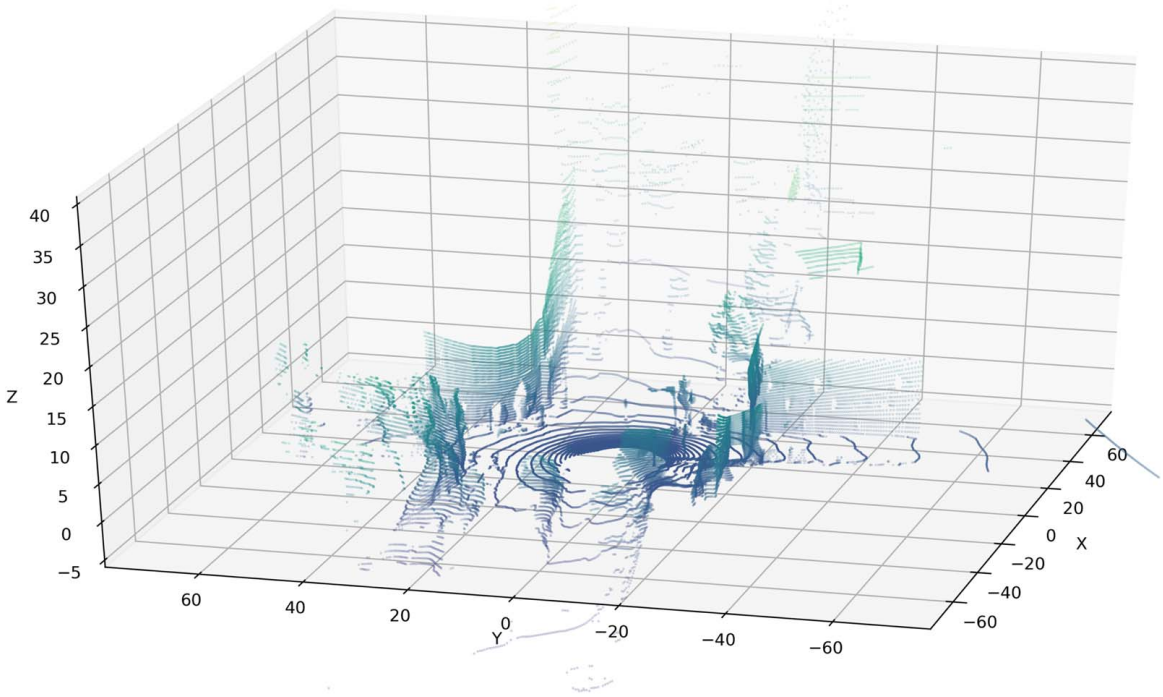

Fig. 17 The illustrated point cloud shows data collected from the intersection and same point in time as in Figs. 15 and 16. The height of the cloud is encoded as color in the image for readability. The intensity of the returned signal is also collected, but not plotted here.

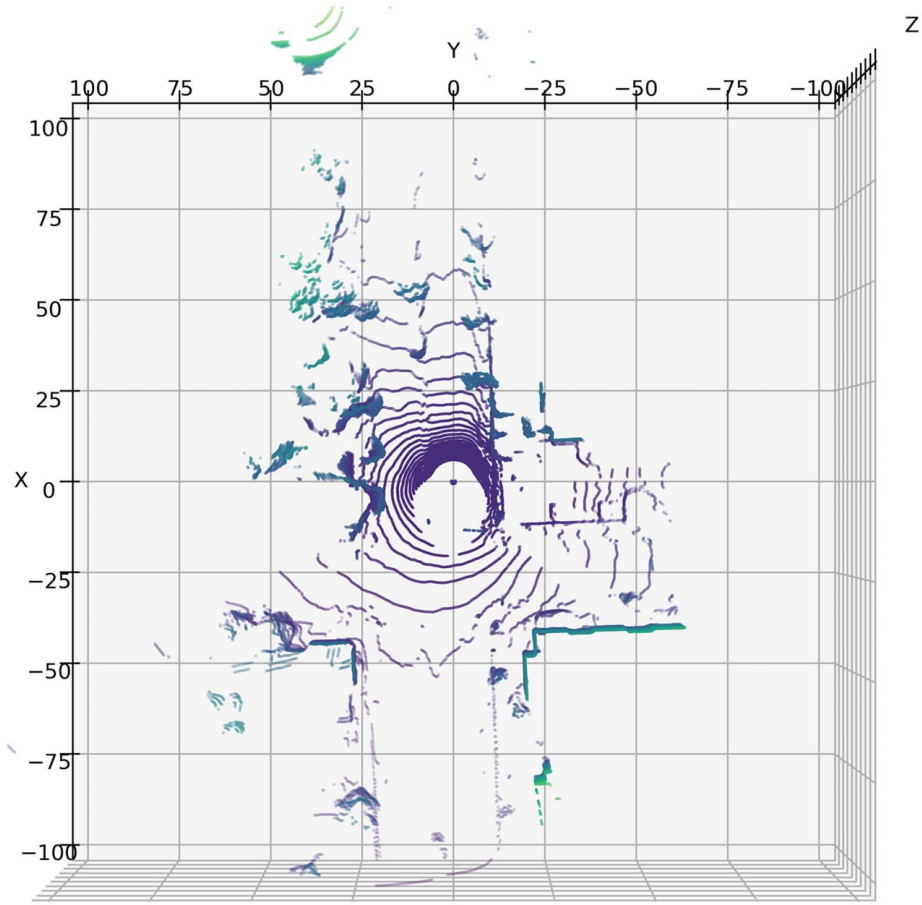

Fig. 18 The lidar point cloud is shown from a birds eye view and displays the information from the environment that can be seen by the sensor including trees, buildings, and roads

defined parameters for the accelerometer's noise. As show in Ref. [19], this drift model closely accounts for the noise in an IMU as recorded on both the accelerometer and gyroscope. While GPS noise can be significantly more complex than a Gaussian distribution, the use of OptiX allows for continued development of a more sophisticated model that includes the number satellites visible to the receiver as determined by path tracing methods similar to the model discussed in Ref. [56]. However, since many correction algorithms are often applied to positional estimates, the higher-fidelity model discussed in Ref. [56] could provide artifacts that have been removed by an on-sensor chip.

The sensor model parameters such as update frequency can be found in the sensor datasheet. For GPS and IMU noise models, these are inherently empirical and require calibration or noise estimation. Steady-state noise can be calibrated using stationary sensors, and using statistical methods and curve fitting to approximate noise components such as standard deviation and time constants.

4 Demonstration of Chrono::Sensor

Two examples are provided herein to show Chrono::Sensor at work. The first is a scaled version of an autonomous vehicle that navigates a closed course using only a lidar mounted on the top of the vehicle. The car has no prior knowledge of the course and can navigate safely between the barriers. A third person view of the car and course can be seen in Fig. 12. A front facing gray-scale camera is mounted on the front of the car (Fig. 13), yet in this setup its output is not used by the control algorithm (the control policy relies on the lidar information only). The implemented control algorithm parsed the point cloud into clusters representing the right and left barriers, found a center point as a target location, and used a PID steering controller to navigate to the target. This is not intended to demonstrate a state of the art controller, but rather flex the sensor simulation framework for demonstration purposes. The point cloud generated by the lidar is shown in Fig. 14 and matches the same frame shown by both cameras. Both the cameras and lidar are simulated in Chrono::Sensor. The vehicle is modeled using Chrono::Vehicle with parameters based on a remote control car that has been modified to drive autonomously.

The second demonstration pertains to a sedan in a replica environment. Specifically, through collaboration with Continental Mapping [57], an airplane flying over Madison, WI, generated a replica of a neighborhood in which the virtual vehicle was driven around (this is not an AV, details below). The human-driven vehicle is equipped with a simulated camera and lidar for generating and saving data that can be used offline for evaluating perception algorithms. The collected data are shown in Figs. 16–18.

Figure 15 provides a third-person perspective of the ego vehicle. The vehicle was driven by a student via a Logitech console using the data stream from this very camera for human-in-the-loop control. The data shown in Fig. 16 represent data that could be used for software-in-the-loop control of the vehicle, or for offline training or evaluation of perception algorithms. This camera is placed in the simulation and attached to the hood of the ego vehicle.

Data from a lidar attached to the roof of the virtual vehicle are shown in Figs. 17 and 18. The point cloud is colored by vertical position of each point (height of the point) for readability. The data are aligned with the vehicle coordinate system with X-forward, Z-up, and Y-left. The full videos for both demonstrations can be viewed at [58] under the "Autonomous Vehicles" heading.

5 Conclusion and Future Work

This contribution outlines the Chrono::Sensor framework for software/hardware/human-in-the-loop simulation of robots and AVs. The sensor models represent a foundational component of the Chrono simulation platform endowing it with an expandable sensing solution that complements its multi-physics simulation engine. By leveraging ray-tracing techniques for light-based sensing, high-fidelity models and methods are implemented that mimic the light-acquisition process of real sensors. The IMU and GPS models can augment ground-truth data that comes from the Chrono dynamics engine with realistic noise and lag. The lidar and camera models discussed herein include noise, distortion, and lag. Through Chrono, Chrono::Sensor is available in a public repository as open-source under a BSD3 license for unfettered use/modification/redistribution. The included demonstrations show example data generation for autonomous vehicles simulated in Chrono with Chrono::Vehicle and Chrono::Sensor. Chrono::Sensor has also been used in an off-road AV navigation scenario in Ref. [41]. The source code is freely available [3,59] along with documentation [60] and tutorial/demo examples [61]. The implementation and models herein can be modified and used in external software, but are designed primarily to be a simulation component in Chrono. Chrono as a whole, including Chrono::Sensor are designed as middleware and can be used or called from external frameworks.

The ongoing research associated with this framework seeks to tackle both the breadth and depth of sensing in order to facilitate the simulation of robotic systems and on/off-road autonomous vehicles. To support a broader set of robotic systems, future work will seek to expand the collection of available sensors, to include stereo and omnidirectional cameras (commonly used in robotic systems), encoders, odometers, and magnetometers. For on/off-road AVs, we plan to add exteroceptive sensors such as radar and infrared.

Future work also includes the improvement of the realism of the sensors discussed herein, an effort motivated by a desire to reduce the simulation-to-reality gap. Characterizing and closing the sim-to-real transferability gap will continue to be a paramount challenge to ours and any other simulation-in-robotics effort. Important effects that could significantly impact sensing realism, particularly for off-road scenarios, are tied to environmental attributes, e.g., weather conditions, dust, smoke, foliage, haze. These are beyond the scope of this paper and regarded as directions of future research.

While the underlying ground-truth measurements of the sensor models have been verified within the simulation framework [44], an important path of future work pertains to validating the sensor models against real-world data, and comparison with other sensor simulation frameworks. Presently, this calls for more research to understand and develop methods to meaningfully compare sensor data in a manner that accounts for the intended use of that data (sometimes sloppy sensing can be good enough). In addition to basic modeling research, ongoing work aims at improving the performance and scalability of the simulation framework to provide "as-fast-as-possible" simulation; and increasing the support for deformable objects (the geometries that need to be rendered change in time) to allow for robotic and AV simulation that includes, for instance, soft-robots or deformable terrains.

Acknowledgment

This work was partially supported by National Science Foundation grant CPS–1739869. Additional support came from the Safety Research using Simulation (SAFER-SIM) program. SAFER-SIM is funded by a grant from the U.S. Department of Transportation's University Transportation Centers Program (69A3551747131). The U.S. Government assumes no liability for the ideas expressed in this document or the use thereof. The authors would like to thank Continental Mapping of Sun Prairie, WI, for providing the virtual world replica of Park Street in Madison, WI used herein.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

Data provided by a third party listed in Acknowledgment.

References

[1] Jakobi, N., Husbands, P., and Harvey, I., 1995, "Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics," European Conference on Artificial Life, Granada, Spain, June 4–6, pp. 704–720.

[2] Tasora, A., Serban, R., Mazhar, H., Pazouki, A., Melanz, D., Fleischmann, J., Taylor, M., Sugiyama, H., and Negrut, D., 2016, "Chrono: An Open Source Multi-Physics Dynamics Engine," *High Performance Computing in Science and Engineering–Lecture Notes in Computer Science*, Kozubek, T., ed., Springer, Berlin, pp. 19–49.

[3] Project Chrono Development Team. "Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems," https://github.com/projectchrono/chrono, Accessed December 7, 2019.

[4] Koenig, N. P., and Howard, A., 2004, "Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator." International Conference on Intelligent Robots and Systems, Senda, Japan, Sept. 28–Oct. 2, pp. 2149–2154.

[5] Open-Source-Robotics-Foundation, "A 3D Multi-Robot Simulator With Dynamics," http://gazebosim.org/, Accessed March 9, 2015.

[6] Aguero, C., Koenig, N., Chen, I., Boyer, H., Peters, S., Hsu, J., Gerkey, B., Paepcke, S., Rivero, J., Manzo, J., Krotkov, E., and Pratt, G., 2015, "Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response," Auto. Sci. Eng., IEEE Trans., **12**(2), pp. 494–506.

[7] Simulation, R.-T. P., 2015, "Bullet Physics Library," http://bulletphysics.org

[8] Smith, R., "Opend Dynamics Engine," http://www.ode.org/ode.html, Accessed September 13, 2016.

[9] Lee, J., Grey, M. X., Ha, S., Kunz, T., Jain, S., Ye, Y., Srinivasa, S. S., Stilman, M., and Liu, C. K., 2018, "DART: Dynamic Animation and Robotics Toolkit," J. Open Source Soft., **3**(22), p. 500.

[10] Sherman, M. A., Seth, A., and Delp, S. L., 2011, "Simbody: Multibody Dynamics for Biomedical Research," Proc. IUTAM, **2**, pp. 241–261.

[11] Coppelia Robotics, 2020, "CoppeliaSim."

[12] NVIDIA, 2019. "Isaac SDK, " https://developer.nvidia.com/isaac-sdk

[13] NVIDIA, 2019, "PhysX Simulation Engine," http://developer.nvidia.com/object/physx.html

[14] Michel, O., 2004, "Cyberbotics Ltd. Webots™: Professional Mobile Robot Simulation," Int. J. Adv. Rob. Syst., **1**(1), p. 5.

[15] Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C., 2007, "Usarsim: A Robot Simulator for Research and Education," 2007 IEEE International Conference on Robotics and Automation, Rome, Italy, Apr. 10–14, Silver Spring, MD, pp. 1400–1405.

[16] Coumans, E., and Bai, Y., 2016–2019, "Pybullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning." http://pybullet.org

[17] Todorov, E., Erez, T., and Tassa, Y., 2012, "Mujoco: A Physics Engine for Model-Based Control," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, Portugal, Oct. 7–12, IEEE, Silver Spring, MD, pp. 5026–5033.

[18] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V., 2017, "CARLA: An Open Urban Driving Simulator," Proceedings of the 1st Annual Conference on Robot Learning, Mountain View, CA, Nov. 13–15, pp. 1–16.

[19] Shah, S., Dey, D., Lovett, C., and Kapoor, A., 2018, "Airsim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," *Springer Proceedings in Advanced Robotics*, Vol. 5, M. Hutter, and R. Siegwart, eds., Springer, Berlin, pp. 621–635.

[20] Epic Games, 2020, "Unreal Engine," https://www.unrealengine.com

[21] Unity3D, 2016, "Main Website," https://unity3d.com/. Accessed June 9, 2016.

[22] Goodin, C., George, T., Cummins, C., Durst, P., Gates, B., and McKinley, G., 2012, "The Virtual Autonomous Navigation Environment: High Fidelity Simulations of Sensor, Environment, and Terramechanics for Robotics," Earth and Space 2012: Engineering, Science, Construction, and Operations in Challenging Environments, Pasadena, CA, Apr. 15–18, pp. 1441–1447.

[23] Carruth, D. W., 2018, "Simulation for Training and Testing Intelligent Systems," World Symposium on Digital Intelligence for Systems and Machines (DISA), Kosice, Slovakia, Aug. 23–25, pp. 101–106.

[24] Goodin, C., Kala, R., Carrrillo, A., and Liu, L. Y., 2009, "Sensor Modeling for the Virtual Autonomous Navigation Environment," SENSORS 2009 IEEE, Christchurch, New Zealand, Oct. 25–28, IEEE, Silver Spring, MD, pp. 1588–1592.

[25] Goodin, C., Durst, P. J., Prevost, Z. T., and Compton, P. J., 2013, "A Probabilistic Model for Simulating the Effect of Airborne Dust on Ground-based Lidar," SPIE Defense, Security, and Sensing, Baltimore, MD, May 23.

[26] Goodin, C., Carrillo, J. T., McInnis, D. P., Cummins, C. L., Durst, P. J., Gates, B. Q., and Newell, B. S., 2017, "Unmanned Ground Vehicle Simulation With the Virtual Autonomous Navigation Environment," 2017 International Conference on Military Technologies (ICMT), Brno, Czech Republic, July 24, IEEE, Silver Spring, MD, pp.160–165.

[27] Goodin, C., Kala, R., Carrillo, A., and Liu, L., 2009, "Sensor Modeling for the Virtual Autonomous Navigation Environment," SENSORS, 2009 IEEE, Christchurch, New Zealand, Oct. 25–28, IEEE, Silver Spring, MD, pp. 1588–1592.

[28] Goodin, C., Doude, M., Hudson, C., and Carruth, D., 2018, "Enabling Off-Road Autonomous Navigation-simulation of Lidar in Dense Vegetation," Electronics, **7**(9), p. 154.

[29] Goodin, C., Carruth, D., Doude, M., and Hudson, C., 2019, "Predicting the Influence of Rain on Lidar in Adas," Electronics, **8**(1), p. 89.

[30] Durst, P. J., Goodin, C., Cummins, C., Gates, B., Mckinley, B., George, T., Rohde, M. M., Toschlog, M. A., and Crawford, J., 2012, "A Real-Time, Interactive Simulation Environment for Unmanned Ground Vehicles: The Autonomous Navigation Virtual Environment Laboratory (anvel)," 2012 Fifth International Conference on Information and Computing Science, Liverpool, UK, July 24–25, Silver Spring, MD, pp. 7–10.

[31] CM-Labs, 2020, "Vortex Studio," https://www.cm-labs.com

[32] Best, A., Narang, S., Pasqualin, L., Barber, D., and Manocha, D., 2018, "AutonoVi-Sim: Autonomous Vehicle Simulation Platform With Weather, Sensing, and Traffic Control," IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, June 18–22.

[33] Siemens – TASS, 2018, "PreScan: Simulation of ADAS and Active Safety." https://tass.plm.automation.siemens.com/prescan, Accessed January 1, 2018.

[34] CVEDIA, 2020, "Syncity," https://www.cvedia.com/syncity/. Accessed February 6, 2020.

[35] rFpro, 2018, "Driving Simulation, " http://www.rfpro.com/driving-simulation/, Accessed February 6, 2018.

[36] MSC Software, 2020, "VIRES Virtual Test Drive, " https://www.mscsoftware.com/product/virtual-test-drive, Accessed July 29, 2020.

[37] dSPACE, 2020, "dSPACE." https://www.dspace.com/en/inc/home/products/sw/sensor˙sim.cfm, Accessed November 6, 2020.

[38] NVIDIA, 2018, "NVIDIA DRIVE Constellation," https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/, Accessed February 6, 2018.

[39] Serban, R., Taylor, M., Negrut, D., and Tasora, A., 2019, "Chrono::Vehicle Template-Based Ground Vehicle Modeling and Simulation," Intl. J. Veh. Perform., **5**(1), pp. 18–39.

[40] Tasora, A., Magnoni, D., Negrut, D., Serban, R., and Jayakumar, P., 2019, "Deformable Soil With Adaptive Level of Detail for Tracked and Wheeled Vehicles," Intl. J. Veh. Perform., **5**(1), pp. 60–76.

[41] Negrut, D., Serban, R., Elmquist, A., Taves, J., Young, A., Tasora, A., and Benatti, S., 2020, "Enabling Artificial Intelligence Studies in Off-road Mobility Through Physics-Based Simulation of Multi-Agent Scenarios," Ground Vehicle Systems Engineering and Technology Symposium (GVSETS) & Advanced Planning Briefings for Industry (APBI), Novi, MI, Nov. 3–5.

[42] NVIDIA Corporation, 2018. "NVIDIA Turing GPU Architecture". WP-09183-001 v01.

[43] Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M., 2010, "OptiX: A General Purpose Ray Tracing Engine," ACM Transactions on Graphics, **29**(4). Article No. 66.

[44] Elmquist, A., and Negrut, D., 2020, "Sensor Modeling and Simulation for Evaluation of Connected and Autonomous Vehicles." Technical Report TR-2020-07, Simulation-Based Engineering Laboratory, University of Wisconsin-Madison.

[45] 2018. "Velodyne Lidar, " http://velodynelidar.com/, Accessed March 17, 2020.

[46] Farrell, J. E., and Wandell, B. A., 2015, Image Systems Simulation, American Cancer Society, Wiley Online Library, pp. 1–28.

[47] Cook, R. L., and Torrance, K. E., 1982, "A Reflectance Model for Computer Graphics," ACM Trans. Graphics (ToG), **1**(1), pp. 7–24.

[48] Tang, Z., von Gioi, R. G., Monasse, P., and Morel, J.-M., 2017, "A Precision Analysis of Camera Distortion Models," IEEE Trans. Image Process., **26**(6), pp. 2694–2704.

[49] Sturm, P., Ramalingam, S., Tardif, J.-P., Gasparini, S., and Barreto, J., 2011, "Camera Models and Fundamental Concepts Used in Geometric Computer Vision," Foundat. Trends ® Comput. Graphics Vision, **6**(1–2), pp. 1–183.

[50] European Machine Vision Association, 2010, "Standard for Characterization of Image Sensors and Cameras," *EMVA Standard 1288*, Vol. 3, European Machine Vision Association, Barcelona, Spain.

[51] Hasinoff, S. W., Durand, F., and Freeman, W. T., 2010, "Noise-Optimal Capture for High Dynamic Range Photography," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Francisco, CA, June 13–18, IEEE, Silver Spring, MD, pp. 553–560.

[52] Guo, S., Yan, Z., Zhang, K., Zuo, W., and Zhang, L., 2019, "Toward Convolutional Blind Denoising of Real Photographs," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, CA, June 16–20, pp. 1712–1722.

[53] Nam, S., Hwang, Y., Matsushita, Y., and Joo Kim, S., 2016, "A Holistic Approach to Cross-Channel Image Noise Modeling and Its Application to Image Denoising," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, June 26–July 1, pp. 1683–1691.

[54] Jaroensri, R., Biscarrat, C., Aittala, M., and Durand, F., 2019, "Generating Training Data for Denoising Real rgb Images Via Camera Pipeline Simulation." arXiv preprint arXiv:1904.08825.

[55] Choi, S.-H., Cho, J., Tai, Y.-M., and Lee, S.-W., 2014, "Implementation of An Image Signal Processor for Reconfigurable Processors," 2014 IEEE International Conference on Consumer Electronics (ICCE), Da Nang, Vietnam, July 30–Aug. 1, IEEE, Silver Spring, MD, pp. 141–142.

[56] Durst, P. J., and Goodin, C., 2012, "High Fidelity Modelling and Simulation of Inertial Sensors Commonly Used by Autonomous Mobile Robots," World J. Modell. Simul., **8**(3), pp. 172–184.

[57] 2019, "Continental Mapping," https://www.continentalmapping.com/, Accessed April 19, 2019.

[58] Simulation-Based Engineering Lab (SBEL), "Movies, Physics-Based Modeling and Simulation." http://sbel.wisc.edu/Animations, Accessed June 9, 2018.

[59] Project Chrono, 2020, "Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems, " http://projectchrono.org, Accessed March 3, 2020.

[60] Project Chrono, "Project Chrono API Web Page, " http://api.projectchrono.org/. Accessed October 20, 2017.

[61] Project Chrono Tutorials, "Project Chrono Tutorials Web Page," http://api.projectchrono.org/tutorial˙root.html, Accessed June 22, 2017.