Aaron Young
Department of Mechanical Engineering,
University of Wisconsin-Madison,
Madison, WI 53706
e-mail: aryoung5@wisc.edu

Jay Taves
Department of Mechanical Engineering,
University of Wisconsin-Madison,
Madison, WI 53706
e-mail: jtaves@wisc.edu

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

Simone Benatti
Department of Mechanical Engineering,
University of Wisconsin-Madison,
Madison, WI 53706
e-mail: benatti@wisc.edu

Alessandro Tasora
Department of Industrial Engineering,
Universita degli Studi di Parma,
Parma, Italy
e-mail: alessandro.tasora@unipr.it

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

Enabling Artificial Intelligence Studies in Off-Road Mobility Through Physics-Based Simulation of Multiagent Scenarios

We describe a simulation environment that enables the design and testing of control policies for off-road mobility of autonomous agents. The environment is demonstrated in conjunction with the training and assessment of a reinforcement learning policy that uses sensor fusion and interagent communication to enable the movement of mixed convoys of human-driven and autonomous vehicles. Policies learned on rigid terrain are shown to transfer to hard (silt-like) and soft (snow-like) deformable terrains. The environment described performs the following: multivehicle multibody dynamics cosimulation in a time/space-coherent infrastructure that relies on the Message Passing Interface standard for low-latency parallel computing; sensor simulation (e.g., camera, GPU, IMU); simulation of a virtual world that can be altered by the agents present in the simulation; training that uses reinforcement learning to "teach" the autonomous vehicles to drive in an obstacle-riddled course. The software stack described is open source. Relevant movies: Project Chrono. Off-road AV simulations, 2020[2]. [DOI: 10.1115/1.4053321]

1 Introduction

Computer simulation has been extensively used in the design and analysis of various automation aspects tied to on-road mobility, see, for instance, Ref. [1]. A similar statement cannot be made for off-road mobility owing to a smaller market and a set of stiff challenges brought along by the unstructured nature of the task at hand. However, a predictive simulation platform for off-road mobility analysis of autonomous agents (AAs) is very desirable since it can accelerate the engineering design cycle, reduce costs, perform more thorough testing, and produce more performant and safer designs. Simulation has its limitations, first of all related to the issue of simulation-to-reality transfer [2], which pertains to the failure of control policies derived in simulation to work well in the real world. Furthermore, models are difficult to set up and calibrate, the validation process can be tedious and time consuming [3]. Open source simulation tools that are both predictive and expeditious are not readily available. This contribution addresses this last point. It describes a simulation environment whose stated purpose is to allow the practitioner to gain insights into the operation of AAs (robots and autonomous wheeled or tracked vehicles) in off-road conditions with an eye toward: improving mechanical designs of AAs; and, producing and testing control policies that govern the operations of the AAs.

There are several ongoing efforts that seek to address the AA simulation issue. In robotics, Gazebo [4,5] is a widely used 3D multirobot simulator with dynamics. It is not a simulation engine per se, but a platform that exposes several engines: ODE,[3] Bullet [6], DART [7], and Simbody [8]. Unlike Gazebo, which is open source, CoppeliaSim (formerly V-REP) [9] is a commercial multirobot simulation solution that also exposes a set of simulation engines: MuJoCo [10], Vortex Dynamics [11], Bullet, and Newton Dynamics [12]. ROAMS [13], and ANVEL [14] are two other simulation engines for off-road AAs. The former is used for mission planning by NASA and draws on an in-house dynamic engine [15]; the latter relies on the ODE simulation engine and has been used in the past for off-road military applications [16] in combination with a sensor simulation package [17]. MAVS is an off-road

[1]Corresponding author.

Manuscript received October 1, 2021; final manuscript received December 3, 2021; published online March 8, 2022. Assoc. Editor: Sachin Goyal.

[2]https://uwmadison.box.com/s/glbpqxpomgyiomt2ydctpe35avrh44vd
[3]http://www.ode.org/

AA simulation environment that is currently under active development [18]. It provides an in-house developed, sophisticated sensor simulation module [19,20], has a ROS [21] bridge, and uses Chrono as its dynamic engine. Recently, CM-Labs has embedded in its Vortex Studio commercial solution comprehensive support for off-road autonomous mobility [11]. USARSim [22] is an AA simulation platform, not under active development, that draws on a game engine (Unreal Engine [23]), a choice with pluses (scalability, ability to create complex worlds) and minuses (the simulation engine is designed for plausibility rather than accuracy). For autonomous vehicle (AV) simulation, Carla [1] and AirSim [24] are two often used open-source simulators, the former designed for on-road AV driving scenarios simulation, the latter originally designed for drones but now also including support for on-road traffic of AVs. Carla and AirSim rely on Unreal Engine but several other engines are used for AA simulation, e.g., Unity [25] and TORCS [26]. One platform using Unity is the LGSVL Simulation platform [27], which is open-source and has similar goals to Carla's. For a survey of other solutions for on-road mobility please see Refs. [28,29].

The AA off-road mobility simulation platform discussed herein is Chrono-centric [30,31]. In its purpose, it is similar to the ANVEL-VANE environment as it seeks to simulate robots and wheeled/tracked vehicles operating in off-road conditions. Compared to the ANVEL-VANE solution, the Chrono environment is different in several respects: it is open source and available for unfettered use under a BSD3 license; it uses its own multibody dynamics engine; it is scalable and deployable on supercomputers, clusters, or multicore architectures owing to its reliance on the Message Passing Interface (MPI) standard [32]; and is under active development. Chrono is an ecosystem of modules and toolkits. It has support for rigid and flexible body dynamics (Chrono::Engine), fluid-solid interaction (Chrono::FSI), and granular dynamics (Chrono::Multicore and Chrono::GPU) applications. It has Python bindings, support for sensor simulation in Chrono::Sensor, an API for ROS bridging, as well as facilities for: rapid vehicle modeling via parameterized templates with Chrono::Vehicle [33]; control policy design with GymChrono; and scalable control policy testing with SynChrono. Chrono relies on GPU computing for fluid-solid interaction and certain granular dynamics simulations, multicore for most of the other modules, and MPI-enabled parallel computing for cosimulation when handling large terramechanics applications or collections of AAs. Although for vehicle-on-rigid-terrain simulation Chrono provides faster than real-time performance, there are numerous applications that lead to long run times, e.g., deformable terrain mobility, nonlinear flexible body dynamics, fording scenarios, etc.

This contribution highlights the Chrono components that support the design and testing of control policies through simulation: PyChrono, GymChrono, Chrono::Sensor, and SynChrono. To show these components at work, a Reinforcement Learning (RL) approach is used herein to produce a control policy. There is nothing special about the RL approach; other techniques to design control policies could be used equally well, a point touched upon in more detail in Sec. 2. Section 3 describes the Chrono infrastructure that facilitates artificial intelligence studies in off-road, multi-agent mobility scenarios. Section 4 covers simulation experiments that highlight two aspects: the scalability of the SynChrono testing environment, and the process of designing the RL control policy along with an evaluation of the policy's robustness. We close with concluding remarks and directions of future work.

2 Deriving Control Policies Through Simulation

Derived using an accurate simulation framework, control algorithms have been shown to bridge the sim-to-reality gap successfully [34,35]. The use of vehicles with Level 1 and Level 2 autonomy has grown considerably [36,37], and the automotive industry is making big strides in the transition to Levels 3 and 4 autonomy [38,39]. The use of simulation for on-road AVs is an area of intense research and development as this technology is seen as an important catalyst of the aforementioned transition.

One active area of research is focused on sampling-based methods. These approaches generate many candidate trajectories a vehicle can follow, selecting and executing the controls associated with the best candidate [40,41]. Graph search methods are commonly associated with the selection of each trajectory. The approach is real-time challenged, since achieving robust results requires a high number of samples to be analyzed [42]. Algorithms such as Dijkstra's, A-Star (A*), or the Rapidly-exploring Random Tree-Star (RRT*) sample the state space either deterministically or stochastically [41]. Depending on the complexity of the traffic scenario, these algorithms can prove computationally expensive and provide suboptimal results.

Model Predictive Control (MPC) is another common AV control approach [43]. Using a dynamic model of a vehicle, the MPC algorithm computes trajectories over the state space and determines an optimal trajectory using gradient-descent optimization techniques [42,44]. A limited time horizon is employed to reduce unneeded computation for times too far out into the future. In comparison to sampling algorithms, the MPC approaches display improved performance owing to their use of gradient fields in the underlying optimization problem [43].

The accuracy of the simulation platform plays a critical role both for MPC as well as sampling-based controllers. To adequately validate and subsequently verify a controller, the simulation must be of high enough fidelity to carry over successfully to reality [45]. For instance, when using traditional PID controller solutions, for which gains must be carefully selected, an inaccurate simulation platform could yield a poor design that leads to undesired consequences when deployed on a real vehicle.

The design of a robust controller that performs adequately in complex environments using the aforementioned strategies has proven difficult when aiming for a generalized policy [46]. An emerging approach that has gained momentum in recent years is based on Machine Learning (ML) [47]. ML has shown promise in producing efficient and robust models that generalize well in a variety of situations. The three pillars of ML include supervised learning, unsupervised learning, and reinforcement learning. In the AA problem, deep reinforcement learning (DRL) has been very successful, as it displays the ability to learn and respond in complex scenarios without the need for preprocessed or labeled data [35].

Since its introduction [48], DRL has proven successful in robotics applications [49,50]. At its core, DRL is an iterative learning process in which an *agent* interacts with an *environment*; at each iteration the agent collects an *observation* (or *state*), then performs an *action* based on the previous observation and gets a *reward* which is tied to its performance. The goal of RL is to find a policy that maximizes the sum of the collected reward.

RL allows for complex control policies viable in unstructured and stochastic environments; is model-free in that it does not require a model that predicts environment transients; and can learn from scratch. RL's major flaw is its need for a massive amount of training data to infer a robust policy. The role of simulation is to produce this collection of samples. *Policy Gradient Algorithms* are a subset of RL algorithms whose goal is to directly learn an optimal stochastic policy $\pi_\theta(a|s)$, where s, a, and θ are the state, action, and a set of learnable parameters, respectively. If π is a Neural Network (NN), the parameters are the weights and biases of the NN, the state is the NN input, and the action is its output. Proximal Policy Optimization (PPO) [51] is one of the most widely used algorithms for continuous state and action environments and is the algorithm of choice in this contribution. PPO is a policy gradient algorithm whose goal is to optimize a stochastic policy. It is also an actor-critic method since another NN is trained and used to estimate the Value Function [52] employed to estimate the Advantage Function [53]. The Advantage Function is used to maximize the objective function.

3 Simulation Infrastructure

The purpose of the simulation environment described is twofold. First, it is used to produce the data needed to design a control policy. Second, it is used for testing purposes. To this end, it exposes the control policy produced in a model-based or model-free approach to tests that gauge its correctness and robustness. This section outlines the five components of this Chrono-centric simulation environment that are leveraged in this research: Chrono::Vehicle, Chrono::Sensor, PyChrono, GymChrono and SynChrono. More established Chrono components or functionality will be touched upon in passing; more details are provided in Refs. [31,33,54].

Chrono. Under active development for over two decades, Chrono [31] is a multibody dynamics simulation engine distributed as open-source under a permissive BSD license. Its core module, Chrono::Engine, provides support for rigid multibody dynamics, nonlinear finite element analysis, and frictional contact dynamics. Chrono is modular, with optional modules providing support for additional classes of physics simulation (e.g., fluid-solid interaction or large-scale granular dynamics), for modeling and simulation of specialized mechanical systems (e.g., ground vehicles), for interfaces to external solvers (e.g., sparse direct linear solvers), or for dedicated parallel algorithms targeting different computing architectures (multicore, distributed, and GPU) for large-scale simulations.

Written almost entirely in C++, Chrono is middleware in that it is called from user code or a third-party software. Chrono is portable and can be built on different platforms, under different operating systems, and using various compilers. Chrono has a continuous integration process, an active user forum, and is managed through GitHub [55]. Its latest release is 6.0, available as of March 2021. Chrono is used by academic, industrial, and government research and development groups and projects, e.g., NASA, U.S. Army, and European Space Agency.

Chrono::Vehicle. Chrono::Vehicle [33] is a specialized Chrono module that exposes a collection of templates (fully parameterized models) for various topologies of both wheeled and tracked vehicle subsystems. It provides facilities for modeling rigid, deformable, and granular terrain; support for closed-loop and interactive driver models; and run-time and off-line visualization of simulation results. Chrono::Vehicle leverages and works in tandem with other Chrono modules for run-time visualization or finite element, granular dynamics, and parallel computing support.

Chrono::Vehicle provides *subsystem* templates for tires, suspensions, steering mechanisms, drivelines, sprockets, track shoes, etc.; templates for external systems such as powertrains, drivers, and terrain models; and additional utility classes and functions for vehicle visualization, monitoring, and collection of simulation results. As a middleware library, Chrono::Vehicle requires the user to provide C++ classes for a concrete instantiation of a particular template. An optional Chrono library provides complete sets of template instantiations for several concrete ground vehicles (e.g., a Sedan, HMMWV, SUV), both wheeled and tracked, which can serve as examples for developing more customized vehicle models. An alternative mechanism for defining concrete instantiation of vehicle system and subsystem templates is based on input specification files in the JSON format [56]. For additional flexibility and to allow integration of third-party software, Chrono::Vehicle is designed to permit either monolithic simulations or cosimulation where the vehicle, powertrain, tires, driver, and terrain/soil interaction can be simulated independently.

Chrono::Vehicle provides several classes of terrain and soil models, of different fidelity and computational complexity, ranging from rigid, to semi-empirical Bekker-Wong type models, to complex physics-based models based on either a granular or finite-element based soil representation. For simple terramechanics simulations, Chrono::Vehicle implements a customized Soil Contact Model (SCM), based on Bekker theory, that is lightweight, scales well to arbitrary terrain size and incorporates bulldozing effects [54]. Second, Chrono provides an FEA continuum soil model based on multiplicative plasticity theory with Drucker–Prager failure criterion and specialized brick elements. Finally, leveraging Chrono support for large-scale granular dynamics and for multicore, GPU, and distributed parallel computing, off-road vehicle simulations can be conducted using fully-resolved, granular dynamics-based complex terramechanics, using a Discrete Element Method approach, see Fig. 1 [57,58]. Recently, a continuum methodology has also been implemented for terramechanics, see Ref. [59].

Chrono::Sensor. Chrono::Sensor provides sensor simulation support for software-in-the-loop testing. Cameras, lidars, radars, GPS, IMUs (gyroscope, accelerometer, magnetometer) and tachometers can be placed within a Chrono simulation to generate synthetic data based on user-defined sensor parameters and attributes of the virtual world hosting the AA simulation experiment. The goal of the module is to allow realistic data generation based on sensor characteristics such as noise, distortion, and lag. Sensors can be attached to objects within the simulation and configured to match corresponding real sensors. For modeling convenience, sensors can be defined through a JSON file [56]. Additionally, custom sensors and postprocessing filtering can be implemented, leveraging the existing rendering framework or physics interface. Where possible, the sensors leverage physics-based models. Where physics-based approaches are infeasible to model or simulate (e.g., noise, or MEMS) data is augmented by in part by phenomenological models. More details on the framework and models can be found in Ref. [60].

Sensors provided by Chrono::Sensor can be divided into interoceptive (IMU, tachometer) and exteroceptive (GPS, camera, lidar, radar). For interoceptive sensing, the module utilizes the internally computed physical quantities from the Chrono system and can augment this ground truth with drift, Gaussian noise, lag, and filtering characteristics from finite collection time. For exteroceptive sensors that provide information about scene characteristics, Chrono::Sensor leverages hardware accelerated ray tracing through the OptiX library [61] and implements physically based rendering techniques. The ray tracing approach allows for the physical reconstruction of the light-based data acquisition process and thus controls the attributes of the synthetically generated sensor data. For camera, lens models and postprocessing noise augmentation are supported, with an interface to extend or implement custom models. For lidar, the framework expands on work from Ref. [19] to provide a beam divergence model that supports multiple modes of lidar return and reduced intensity during partial beam reflectance. The camera and lidar can also be parameterized by update rate, time over which to collect data, and lag. All sensors and capabilities are written in C++, but can also be accessed from Python through the PyChrono interface. The entire module can be run headless without the requirement of a render context, allowing for ease of deployment in machine learning applications on remote servers or in the cloud [62].

PyChrono. While the main Chrono API is expressed in C++, we recently implemented Python wrappers for much of the Chrono functionality. The purpose was twofold: provide a lower-entry point to Chrono simulations for users less familiar with C++; and facilitate interfacing to various machine learning platforms, e.g., TensorFlow [63], PyTorch [64], Theano [65], and CAFFE [66]. The Python wrapping relies on using automated technology provided by SWIG [67] to generate the interface between Python user-code and the underlying Chrono C++ libraries. Presently, a large set of Chrono functionality is exposed to Python users, including the core multibody and FEA module, the interface to CAD systems (like SolidWorks), run-time

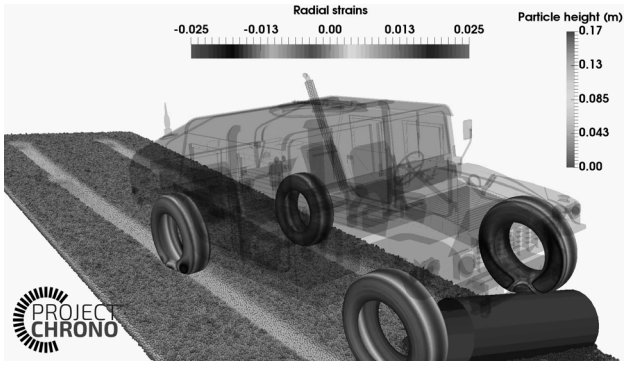


Fig. 1 Chrono::Vehicle HMMWV with flexible tires navigating granular terrain demonstrating vehicle dynamics, flexible body dynamics, and parallel computing support in Chrono [57]

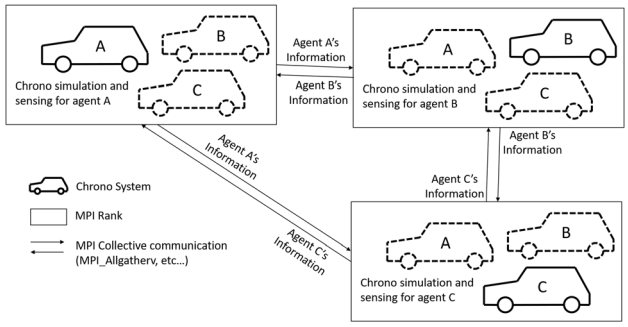


Fig. 2 Schematic of the SynChrono framework. Dynamics simulations are done in separate Chrono systems and the outcome of the dynamics simulation is synchronized between ranks using MPI.

visualization with Irrlicht, etc. In particular, full support is available in Python for the Chrono::Vehicle-based modeling, simulation, and visualization of wheeled and tracked ground vehicles, as well as the use of sensor models provided by Chrono::Sensor. PyChrono for Python 3 can be built from sources on Linux, Windows, and MacOS. Alternatively, prebuilt conda PyChrono packages are available on the project's Anaconda page [68] (note that Chrono::Sensor is not available yet via the conda PyChrono packages).

GymChrono. This is an extension of OpenAI Gym [69]. It exposes a set of environments providing continuous control tasks for physics and sensor simulation run by the Chrono backend. These environments inherit from OpenAI Gym classes. As such, they can be used out of the box with any algorithm or DRL framework made for gym environments. They can also draw on gym's environment parallelization for learning acceleration.

SynChrono. SynChrono is a software component that uses Chrono to implement a distributed-memory execution model when simulating scenarios that include multiple AAs. By leveraging the Message Passing Interface (MPI) standard [70], SynChrono can manage multiple instances of Chrono running together in a *single* mobility analysis on a supercomputer, cluster, or multicore setup thus supporting the scalable and distributed simulation of multiple agents (robots, tracked vehicles, wheeled vehicles, etc.) The paradigm embraced is that of running the dynamics of one AA as one MPI rank, with the ranks/AAs communicating through MPI messages to maintain space and time coherent state for all agents participating in the study. As an example, if there are two agents, SynChrono makes it possible to synchronously run the two agents on two different compute ranks in a supercomputer. By the same token, if there are 50 agents and 50 compute ranks in a cluster, SynChrono provides the infrastructure to keep the 50 agents operating in a coherent (time-wise and space-wise) virtual world. The time coherence aspect prevents some agents from racing into the future while other agents lag behind in the past. The global synchronization mechanism in SynChrono ensures that all agents march forward in simulation time in a coherent fashion so that mutual interaction (a vehicle crossing the ruts of a different one, a vehicle sensing another vehicle, etc.) happens as it would in a monolithic simulation.

A schematic of the structure of SynChrono's MPI framework is shown in Fig. 2. SynChrono manages multiple AAs as multiple processes via as many MPI ranks. Each AA runs in its own SynChrono process (an MPI rank) and interfaces with its dedicated control stack for software-in-the-loop or human-in-the-loop control. The control stack is fed synthetic data generated by Chrono::Sensor and acts upon the environment through Chrono::Vehicle control inputs (throttle, steering, braking). The control algorithm for each agent is also configurable and can vary from complex algorithms that fuse sensor feeds/data streams, to controls based on empirical models, and on to inputs provided by a human-in-the-loop in scenarios that are simple enough to allow real-time simulation.

Each SynChrono process is responsible for the dynamics of a single agent. At a slower frequency (relative to the simulation time-step), all SynChrono processes communicate via the SynChrono daemon to exchange state information. State data is intended to be minimal, sufficient to enable a SynChrono process to reconstruct a "ghost" version of outside agents in its own world for visualization and sensing purposes. In an example where each agent is a vehicle, the state information consists of the vehicle location and orientation along with pose information for each wheel. This information is packaged for transmission using the FlatBuffers serialization library.[4]

As a justification for choosing an MPI-based communication mechanism, the reader is referred to Fig. 3. As reported in Ref. [71], there is no other standard for communication that is better than MPI in terms of latencies with the exception of the Data Distribution Service (DDS) solution, which comes on par with MPI. Additionally, SynChrono supports the DDS standard [72] as well, yet the simulations reported herein were all carried out using the MPI standard.

One limitation of the implementation is that two agents running as two SynChrono processes cannot participate in an operation that couples their dynamics, e.g., jointly lifting a heavy object. Such a scenario should be run in Chrono, since no force information is synchronized in SynChrono. This will make the simulation longer to run since more agents will have to be handled within one Chrono process. However, if the agent coupling happens via sensing or through the virtual world, e.g., one agent sensing another one, or one vehicle crossing over and being jolted by the ruts left by a different agent, then SynChrono can be relied on, thus ensuring scalability.

Interface to an External Controller. For testing of control algorithms that are intended to be easily transferred to real-life vehicles or robots, the simulation platform provides an external control interface that is exposed in SynChrono. An agent in SynChrono can send messages (i.e., sensor data packets) to the external autonomous controls framework which can then send a message back (i.e., control inputs). The control stack is independent of the SynChrono platform (e.g., a bridge has been developed for ROS/ROS2), and can be tested with inputs replicating those from reality, such as sensor and/or V2X communication data.

4 Technology Demonstration

All simulation scenarios considered in this section use a Chrono::Vehicle HMMWV model. Chrono::Vehicle was benchmarked

[4]https://google.github.io/flatbuffers/flatbuffers_white_paper.html

as part of the Next Generation-NRMM (NG-NRMM) exercise [73]; Chrono::Vehicle-specific benchmark findings are detailed in Refs. [74] and [75]. All results reported herein were obtained using a simulation time-step of $\Delta t = 2 \times 10^{-3}$ seconds, both for rigid and deformable terrain. This time-step information is relevant when discussing real-time performance and scalability aspects.

4.1 Synchrono Scaling Analysis. SynChrono uses N processes executing on a supercomputer or workstation to simulate the dynamics of N agents handled as N independent Chrono simulations. The numerical experiments described here answer the following questions: (*i*) How does the time to complete a simulation change as N increases? (*ii*) How fast is SynChrono in mobility studies on rigid terrain? (*iii*) How fast is it on SCM deformable terrain?

The handling of a virtual world that has SCM terrain is challenging since each of the N vehicles alters the terrain at the same time and these changes must be space and time coherent. The key component of the SCM terrain is the deformation of each vertex in the underlying mesh. All other terrain properties can be computed based on the height of each vertex alone. At each simulation step, a SynChrono rank that is associated with an agent moving on SCM terrain collects a list of the deformed vertices that the agent produced during the course of that time-step. Mesh deformation data may not be sent at every simulation time-step (as agents do not synchronize every time-step, but only at a slower rate, sufficient to provide smooth sensor data), so this collection of mesh changes is, in general, persistent across simulation time steps. Once an agent reaches a SynChrono synchronization point, the cumulative mesh deformations produced by one agent are sent via the MPI network to every other agent's node. Each agent then applies the deformations to their own copy of the SCM mesh and resets their collection of mesh deformations. This means that two agents should not come close enough to the point where they deform the same vertices during the same synchronization period, as there would be no "source of truth" for those deformations. This is not a matter of much concern, as it is just as restrictive as SynChrono's assumption that any two agents will not interact by crashing. Book-keeping for the SCM mesh uses an integer grid to localize each vertex, and as noted earlier, since vertices are only displaced vertically, the information needed to synchronize a single vertex is two integers for the position and a double for the displacement. While this is not much per vertex, of the order of thousands of nodes can be impacted per synchronization step per vehicle; this can affect overall performance.

The scenario discussed herein is that of many vehicles crossing perpendicularly on a rectangular patch of SCM terrain, see Fig. 4 and online movies [76]. In this setup, one can easily scale up the number of vehicles and verify that the SCM terrain deformation is properly synchronized across multiple ranks. The scaling metric used was the Real Time Factor (RTF), representing the amount of wall-clock computation time divided by the amount of time simulated. Running in real-time corresponds to a factor of 1.0, while slower than real-time corresponds to factors larger than 1.0. The tests were run on the Euler computing cluster at the University of Wisconsin-Madison. Each node has an Intel Xeon E5 2650

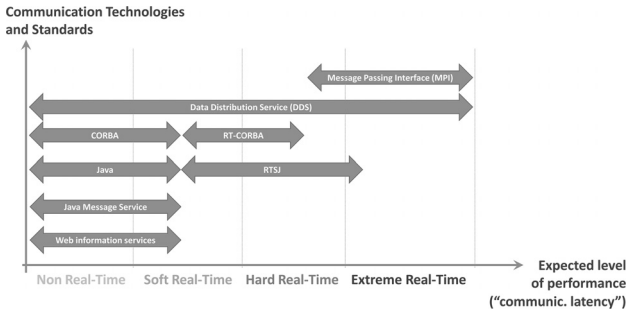


Fig. 3 The MPI standard was chosen owing to its low communication latency, see [71], thus positioning SynChrono for human-in-the-loop and hardware-in-the-loop simulation

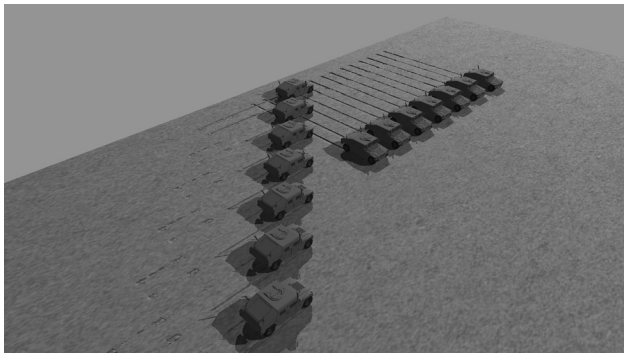


Fig. 4 Environment used for SynChrono scaling analysis for agents operating on SCM terrain. Two lines of vehicles move across a rectangular patch, crossing orthogonally and making ruts in the SCM soil.

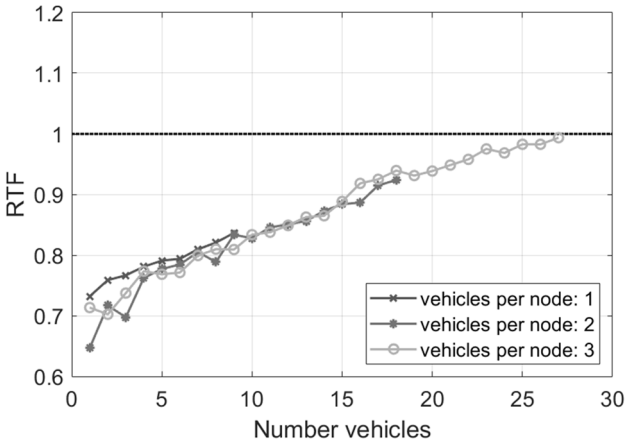


Fig. 5 SynChrono scaling analysis for SCM terrain, with a 5 cm grid resolution. SynChrono simulations on nine cluster nodes, using one, two, or three MPI ranks per node (corresponding to simulations with 9, 18, and 27 vehicles, respectively) and three threads per rank show the real-time capability of simulating multiple vehicles on deformable terrain (RTF ≤ 1).

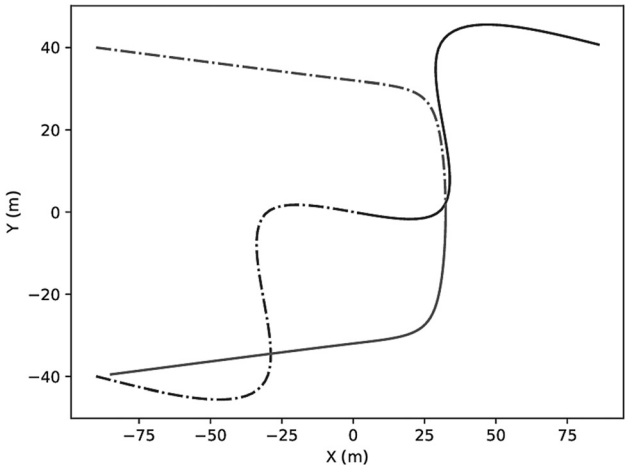


Fig. 6 The double S and C paths used during training. Each one of these is randomly flipped and rotated, resulting in 16 different possible paths. The dashed lines represent the segment of the path in which the simulation episode can start.

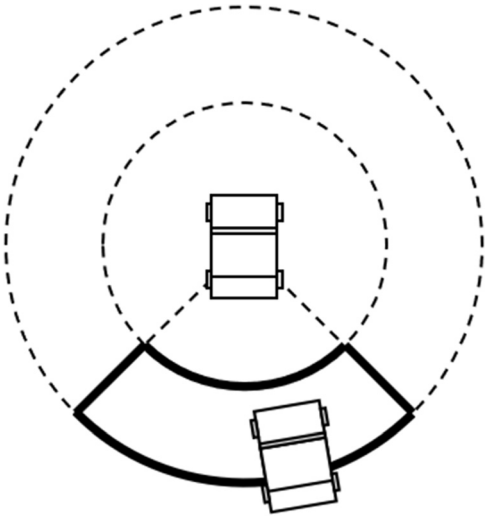


Fig. 7 The maximal reward is given to the agent when it is in the region shown, with the reward decreasing hyperbolically the further the agent is from the desired area. Note that the figure is not to scale. More concretely, the reward is provided to the agent when the angle between the heading of the leader and the follower is in the range $[-\frac{\pi}{4}, \frac{\pi}{4}]$, and this reward is maximal when the agent is an optimal distance from the lead vehicle.

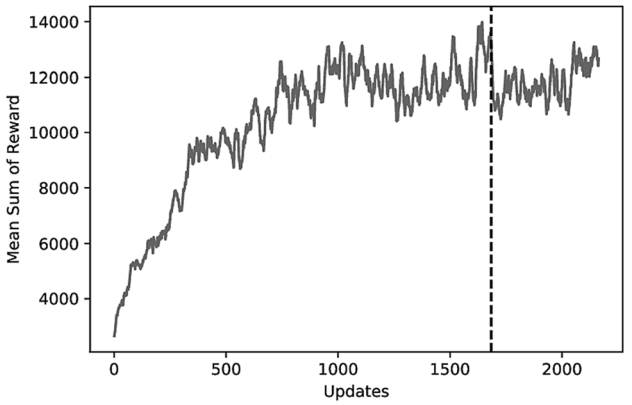


Fig. 9 Plot of the moving average of the sum of collected rewards with respect to the policy updates (updated every 1500 interactions). The vertical dashed line represents the switch from the reduced to the full HMMWV model. (*a*) Rigid terrain; (*b*) SCM-Hard terrain;and (*c*) SCM-Soft terrain.

v3, 2-socket 10-core processor; internode communication is facilitated via a Gigabit Ethernet interconnect.

The scaling analysis reported herein drew on nine nodes and up to 27 SynChrono processes (each running a single vehicle), assigning one, two, or three SynChrono vehicles per node. Simulations conducted on rigid terrain show practically constant scaling, with RTF values around 0.6 (i.e., faster than real-time). Using three OpenMP threads per MPI rank (for parallel ray-casting in the SCM calculations), the simulations on SCM deformable terrain can also achieve real-time or better, as shown in Fig. 5. The RTF value for SCM terrain is independent of the SCM soil parameters (i.e., soft versus hard), but is highly dependent on the processor performance, MPI setup, number of OpenMP threads assigned to each rank, compiler, and compilation optimization. For example, simulations with too many ranks per node can exhibit a slowdown due to cache contention. SynChrono has been used to simulate even more vehicles (up to 128) on a different cluster. The scaling analysis presented here was limited by the number of identical nodes available on the Euler cluster.

4.2 Learning to Drive in a Convoy. Chrono helps with two tasks: learning a control policy, and testing a policy, the latter designed in Chrono or elsewhere. In this example, PyChrono and GymChrono are used to design a policy and SynChrono is subsequently used to test it. The RL-based learning is done on rigid terrain using a nondescript texture. The goal is to enable a vehicle to move as part of a convoy. To test it, the policy is deployed on vehicles that are part of a four-vehicle convoy driving on rigid or SCM deformable terrain. Up to three of the convoy vehicles use this policy while driving in a platoon. Thus, the possible scenarios

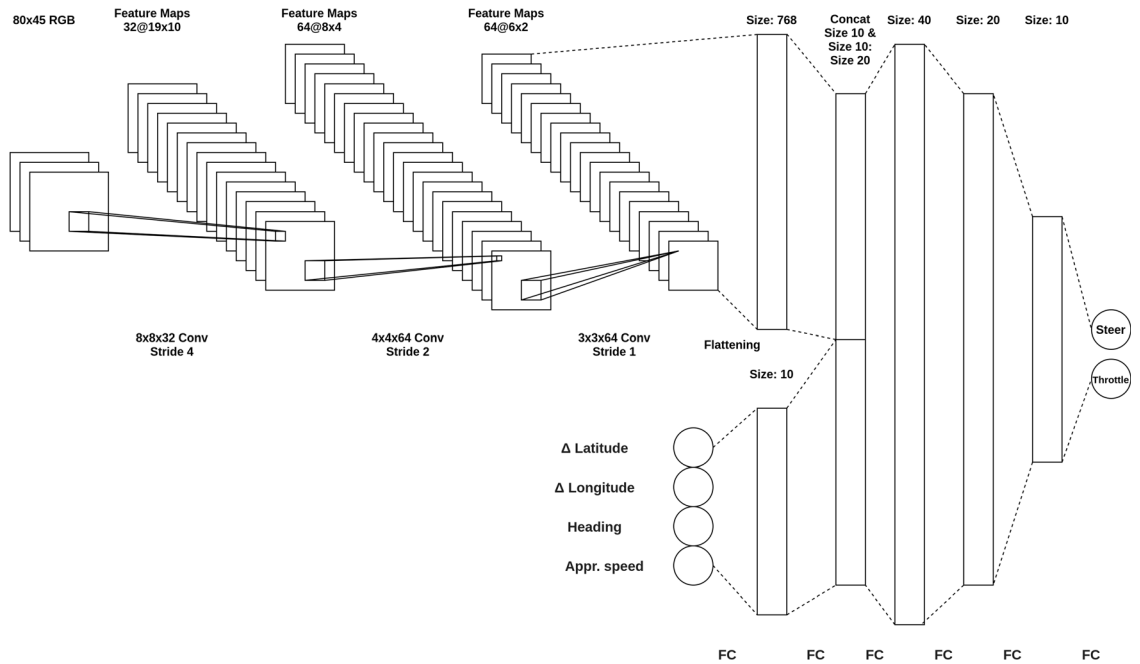
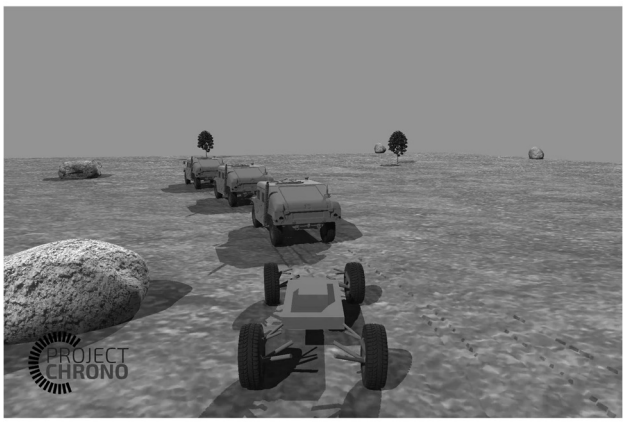


Fig. 8 Sensor Fusion NN architecture

are: three lead vehicles and one following vehicle (3 L+1F), two lead and two followers (2 L+2F), and one lead and three followers (1 L+3F). The *lead* vehicles are programed to follow a path defined by way-points; for all purposes, these can be considered human driven. A follower vehicle is autonomous and uses the learned policy to follow the vehicle in front of it. In doing so, it should (*i*) not crash into the vehicle ahead of it, and (*ii*) avoid hitting obstacles in the vicinity of the path. To this end, it relies on a camera sensor, location acquired through GPS sensing, and compass heading. Note that communication allows a vehicle to find out the GPS location and velocity of the vehicle in front. Given that four vehicles are involved in this platooning experiment, SynChrono is subsequently used to test the policy to reduce simulation times. Indeed, this validation could be run in Chrono::Engine, but it would take more than four times as long to complete. The salient points of this experiment are as follows: although vehicles are run in different SynChrono processes, there is time and space coherence between them to the point where the vehicles sense each other; the learning occurs using rigid terrain with nondescript texture, yet the policy is tested on deformable terrain that is white (snow-like) or brown (silt-like); this is an end-to-end policy that uses sensor data fusion to control both the steering and acceleration/deceleration of the vehicle.

(*a*)



(*b*)

(*c*)

Fig. 10 Still frames from attached third person camera. NOTE: last vehicle shown without cabin to better see how steering takes place in the movies associated with the simulation [76].

Designing a Policy. The policy was obtained through training using a custom implementation of the Proximal Policy Optimization (PPO) reinforcement learning algorithm leveraging PyTorch [64] as the Deep Learning framework. The agent is a HMMWV vehicle modeled in Chrono::Vehicle. The goal of the training process is to develop a control policy that enables an agent (in this case, the vehicle) to drive in a convoy. For training, to increase the randomness of the path and thus the robustness of the control policy, two different path types were used on a 90 m × 90 m area. The first is S-shaped, starting from one corner and finishing in the opposite corner; the second is C-shaped, starting and finishing on the same side of the driving area. To further increase the randomness, these paths are mirrored along the east-west and north-south axes to obtain 16 different possible paths. The starting point is picked randomly within the first half of the path as shown in Fig. 6. Eight obstacles placed near the path are randomly selected from various rock, tree, and bush assets.

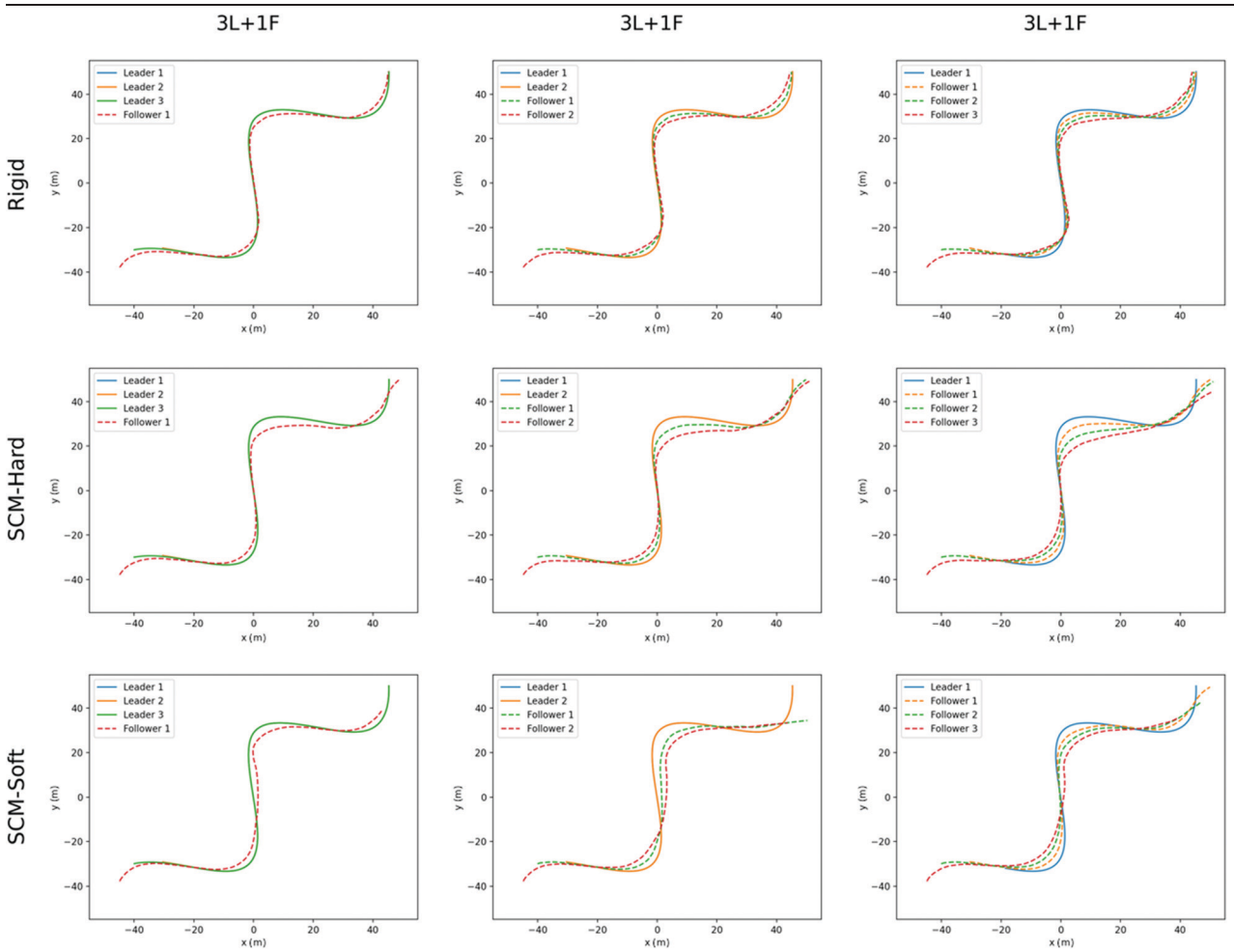
In order for the agent to accomplish its task, the vehicle must be aware of its surroundings. To that end, the HMMWV used two sensors simulated in Chrono::Sensor, a GPS sensor and an RGB camera placed on the front bumper. The camera, which updates at 30 Hz, has a resolution of $80 pixels \times 45 pixels$; note that resolutions as high as $3840 pixels \times 2160 pixels$ can be simulated but for the test considered herein resolutions this high would slow down both the simulation and learning without any clear gain. This level of resolution suffices, since the detailed features that could be extracted from higher resolution images are likely not needed by the control policy. Furthermore, given that the dataset contains one image per interaction, images of too high resolution can quickly deplete the available GPU memory due to the increased memory footprint of the NN update process. As such, large images that are contained in observations are typically avoided.

The target for the vehicle to earn its reward is illustrated schematically in Fig. 7. As with any other RL environment, an observation and a reward are provided to the ML algorithm at each time-step. Subsequently, the agent must perform an action prescribed by the ML algorithm in order to maximize the reward collected. The action is a two element array with the first a steering value and the second element a combined throttling and braking value. The choice of collapsing throttle and brake control into the same action was taken to avoid simultaneous braking and throttling as they both directly control the vehicle's acceleration.

The learning draws on information from several sensors, whose output is organized into two tuples. The first element is an 80 × 45 × 3 RGB image. The second is a vector of four values: the latitude and longitude difference between the leader and the follower, the heading according to the compass, and the speed at which the follower is approaching the leader. This multisensor observation required the NN architecture to incorporate an input composed of a 3D and a 1D tensor. The image is processed in a convolutional neural network as in Ref. [48]. Its output is then concatenated with the output of the one Fully Connected (FC) hidden layer deep neural network which takes the 1D tensor as input. Their concatenated output is then processed by three FC hidden layers. The architecture of the model is shown in Fig. 8.

RL-based training requires a very large number of iterations. Three decisions helped speed up the training process: (*i*) the lead

Table 1 2D Positions of each vehicle in each simulation configuration



vehicles were not simulated, but only rendered at the correct location and orientation (as this has no bearing for sensing purposes); (*ii*) a reduced-order model of the HMMWV vehicle was used in the first stage of training, with the more computationally demanding full vehicle model substituted during the training process (see Fig. 9) to further refine the NN parameters; and (*iii*) the learning process was accelerated using the OpenAI Baselines tool for environment parallelization [69], thus allowing several simulations running simultaneously to speed up the collection of the dataset samples.[5]

Learned Policy Testing. The AA control policy derived in PyChrono and GymChrono was tested in SynChrono for various convoy setups while operating on three terrain types. The platoons were 3 L+1F, 2 L+2F, and 1 L+3F. The terrains were rigid, SCM hard (silt-like), and SCM soft (snow-like). This led to a set of nine platooning scenarios. In the following discussion, the leader and follower vehicles are numbered starting from the head of the convoy; for example, the order of the vehicles in a 1 L+3F configuration is: Leader, Follower 1, Follower 2, Follower 3.

For each platooning scenario, data recorded from simulations included position, velocity, and acceleration for each of the four vehicles. In addition, a high definition camera sensor, from a third-person perspective, was attached to the last vehicle in the convoy in order to visualize the simulation. Full-length videos of representative simulations are available online [77]. Different ground textures and colors were used to further differentiate between the three terrain types (rigid, SCM-Hard, and SCM-Soft) and test robustness of the control policy (Fig. 10).

Table 1 shows top-down views of the convoy trajectories for each platooning scenario, with solid and dashed lines representing leader and follower trajectories, respectively. As these images indicate, different simulation configurations lead to different reactions of the follower vehicles. The training process described in Sec. 4.2 rewards a follower vehicle for being within a certain angle and distance of the leader; as a result, this allows for deviations between the paths of follower and leader vehicles as well as deviations in the speed at which leader and follower vehicles negotiate a certain path segment.

In an effort to quantify the performance and robustness of the derived platooning policy, we next present results from a statistical analysis using ensemble convoy simulations. This study includes results from 128 independent simulations for each one of the three terrain types mentioned above. In order to allow relative comparisons among different terrain types as well as between follower positions in the convoy, all simulations used the 1 L+3F convoy configuration and each one of the three sets of 128 simulations used the same set of randomly generated trajectories. The performance metrics used in this analysis measure the path and speed deviation of a follower vehicle from that of the convoy's leader.

It is hoped that the control policy obtained is general and capable of handling some degree of randomness in its environment. While in some machine-learning tasks the randomness in the

[5]Training an end-to-end policy requires a large training dataset, and despite the aforementioned strategies the two parts of the training took approximately 15 and 7 hours 15 hours, respectively.

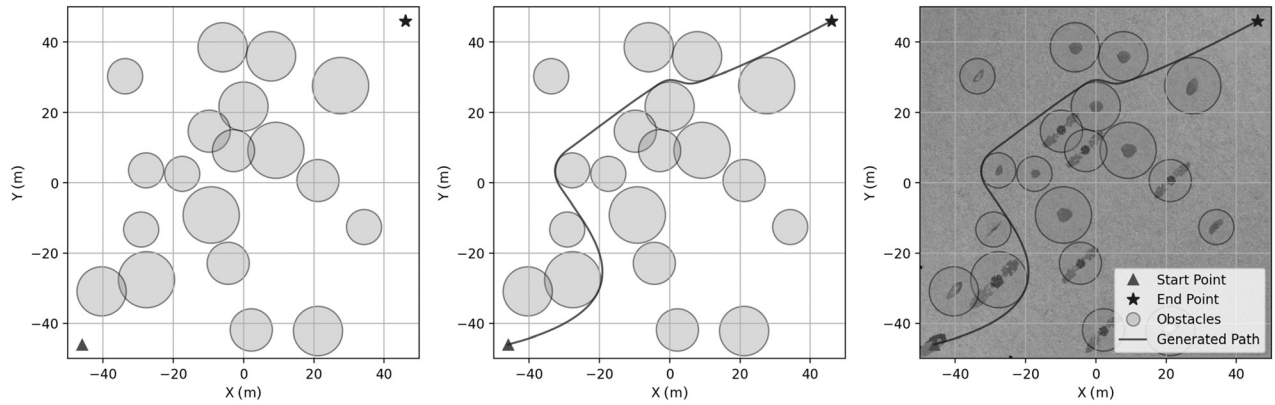


Fig. 11 Sample obstacle field, PSO-based path planning, and the corresponding SynChrono environment setup

training environment may be sufficient to guarantee generality, in our case the training environment was relatively crude since its physics-based nature is relatively performance intensive. For this reason we made many changes to the testing environment relative to the training environment. First, the 128 distinct scenarios were created on a 100m × 100m swath of flat terrain. A script generated 30 randomly placed circular obstacles with sizes uniformly distributed between 5 and 7 m. The obstacle positions were drawn from a uniform distribution with an additional restriction to allow overlap of no more than half their radius [78]. To increase the complexity of the resulting paths and reduce the likelihood of a straight path, *each* configuration included four additional fixed obstacles with radius of 8 m placed equidistantly along the diagonal from start to end (one sample obstacle placement can be seen on the left of Fig. 11). Next, a Particle Swarm Optimization (PSO) algorithm [79] was used to generate a shortest distance path connecting the start (southwest) and end (northeast) locations, as shown in the middle image of Fig. 11. The path generation was constrained to produce trajectories that remained in the domain and did not intersect obstacles. To produce paths that are feasible for simulated HMMWV vehicles, the generation of the corresponding environment setup in SynChrono involved scaling obstacle meshes (rocks, trees, bushes) such that their bounding sphere allows 2 m (the width of the vehicles) of separation from the generated path centerline; in other words, in each of the 128 environments, the path prescribed for the leader vehicle ensures a

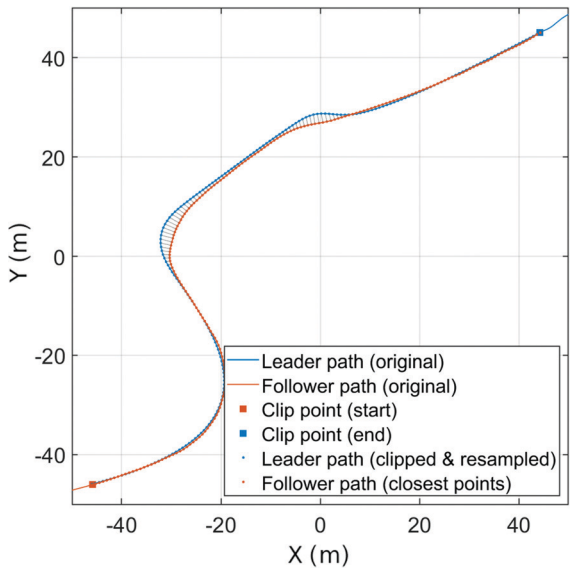


Fig. 13 Deviation in path between Leader and Follower 3. The lateral deviation metrics for this particular scenario (rigid terrain) were $m_p^{avg} = 0.473$ m, $m_p^{max} = 2.484$ m.

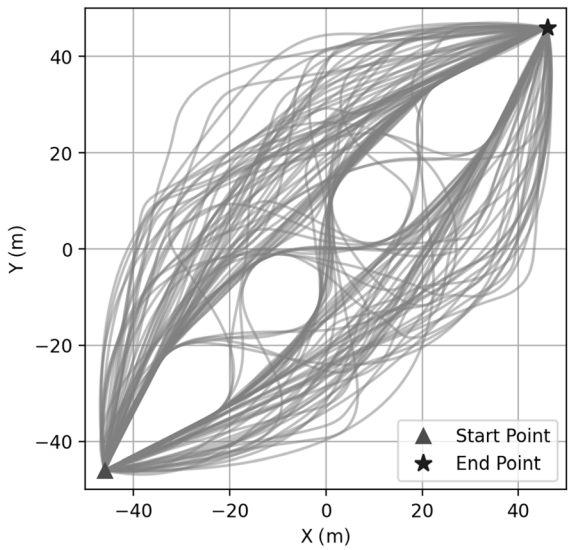


Fig. 12 Ensemble of the 128 paths used in the statistical analysis

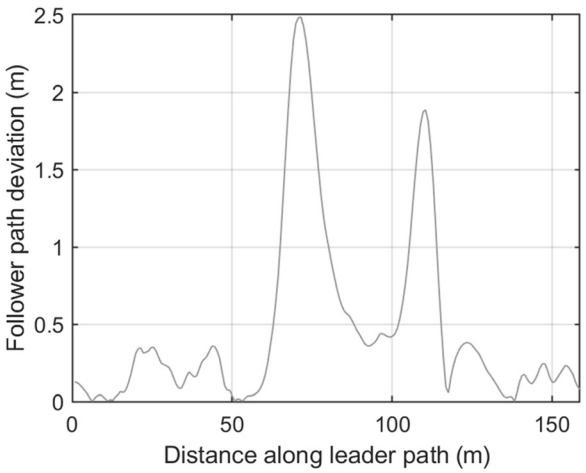


Fig. 14 Deviation in speed between Leader and Follower 3. The vehicle speeds are evaluated at the same location along the leader's path. The speed deviation metrics for this particular scenario (rigid terrain) where $m_{as}^{avg} = 0.505$ m/s, $m_{as}^{max} = 2.154$ m/s and $m_{rs}^{avg} = 12.2\%$, $m_{rs}^{max} = 69.2\%$. (*a*) Average and (*b*) maximum.

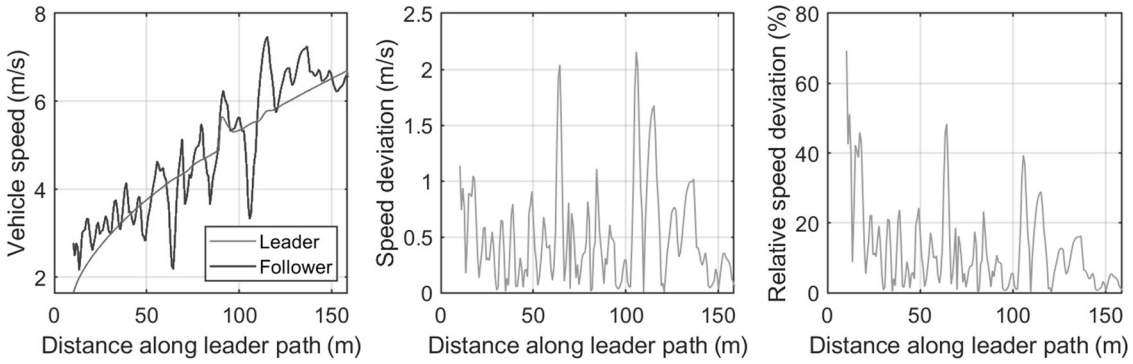


Fig. 15 Statistics of average and maximum follower path deviations. (*a*) Average and (*b*) maximum.

minimum width of 4 m. An overhead view of the resulting SynChrono simulation environment is shown on the right in Fig. 11.

Each of the 128 resulting paths shown in Fig. 12 were then used as the prescribed trajectory for the path-follower PID-based lateral controller implemented on the leader vehicle, in conjunction with a PID-based longitudinal controller that prescribes a target speed linearly increasing in time (corresponding to a constant acceleration of 0.47 m/s^2).

In order to perform a statistical analysis of the performance of the platooning policy, we define a set of six performance metrics that measure the deviations of a follower vehicle from that of the convoy leader and encode both lateral path deviation and deviations in the vehicle speed at a given location along the leader's path. These metrics are defined in such a way as to allow

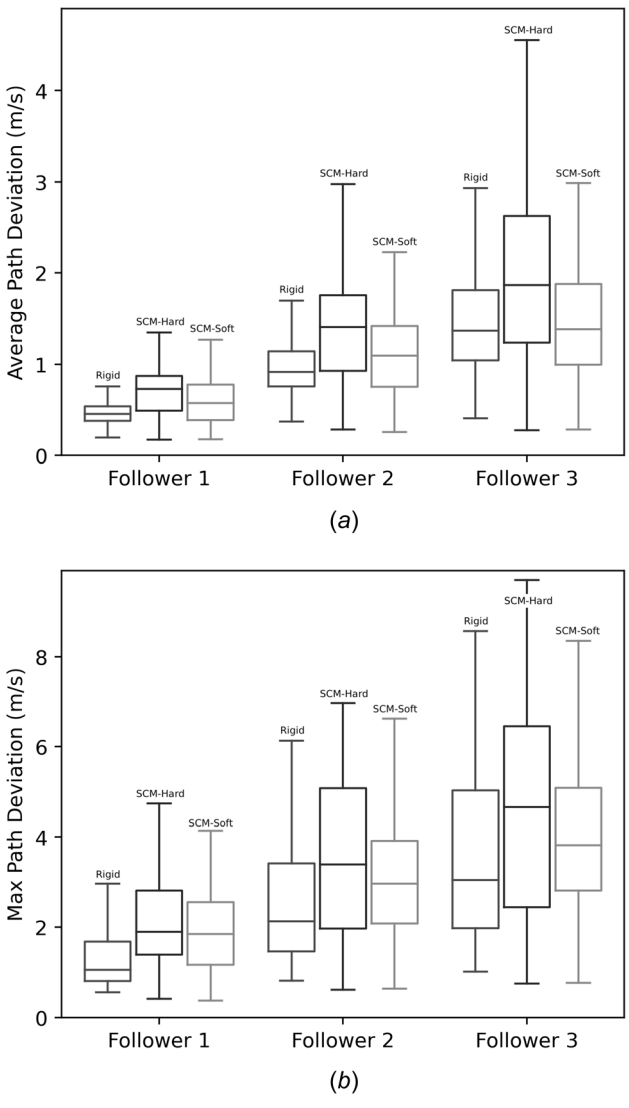


Fig. 16 Statistics of average and maximum follower absolute speed deviations. (*a*) Average and (*b*) maximum.

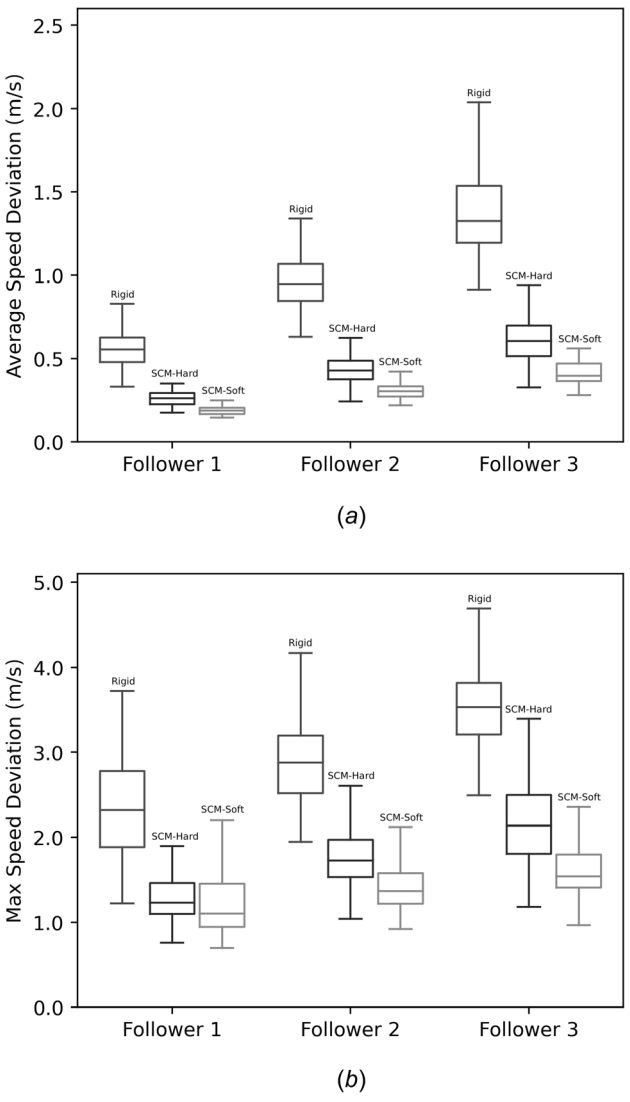


Fig. 17 Statistics of average and maximum follower relative speed deviations. (*a*) Rigid terrain; (*b*) SCM-Hard terrain; and (*c*) SCM-Soft terrain.

comparisons between the performance of followers at different positions in the convoy, as well as across our three different terrain types.

To eliminate differences due to the fact that vehicles in a convoy are inherently staggered along the path, the evaluation of the performance metrics is based on a common path segment. Figure 13 illustrates this process for the same trajectory used as an example before. The sample results used in this description are outcomes of a simulation on rigid terrain and focus on the last vehicle in the convoy (Follower 3). The start clip point is defined as the point on the follower path closest to the leader's initial location. Similarly, the end clip point is defined as the point on the leader's path closest to the follower's final location. The resulting segment on the leader's path is then sampled at intervals of equal arc-length and the closest point on the follower's path to each such sampled point is identified. The distance between corresponding points on the leader and follower paths are then used to define a *follower path deviation* as a function of distance traveled along the leader's path (see Fig. 13). This allows us to define the first two performance metrics: m_p^{avg}, the average follower path deviation, and m_p^{max}, the maximum path deviation.

Next, we compare the vehicle speeds at corresponding points on the follower and leader paths as shown in the left plot of Fig. 14 from which we derive the absolute and relative speed errors (the latter being the speed difference scaled by the leader's speed at that location). The underlying assumption in these definitions is that a follower's speed should match as closely as possible the speed of the leader vehicle at the same location on the path (rather than at the same point in time); this is also why, when calculating the subsequent speed deviation metrics, we discard the values corresponding to the first 10 m of travel (to allow the vehicles to accelerate from rest to the desired convoy speed). With these, we define two more pairs of performance metrics: m_{as}^{avg}, m_{as}^{max}, for the average and maximum absolute speed deviation between leader and follower, and m_{rs}^{avg}, m_{rs}^{max}, for the average and maximum relative speed deviation of the follower.

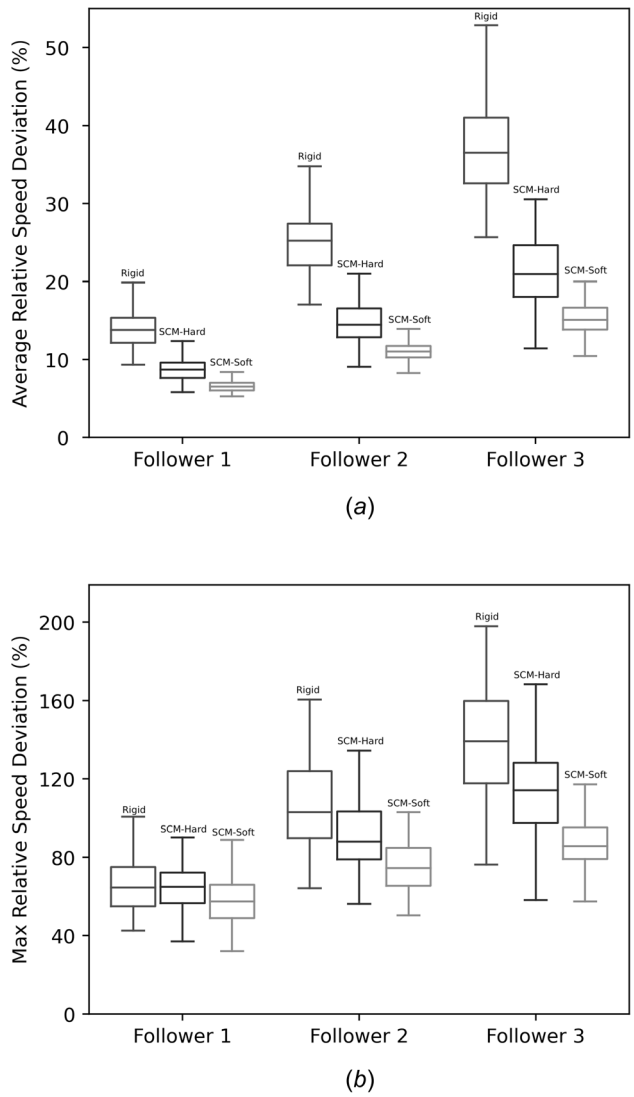


Fig. 18 Paths of every vehicle on each terrain type. (*a*) Rigid terrain; (*b*) SCM-Hard terrain; and (*c*) SCM-Soft terrain.

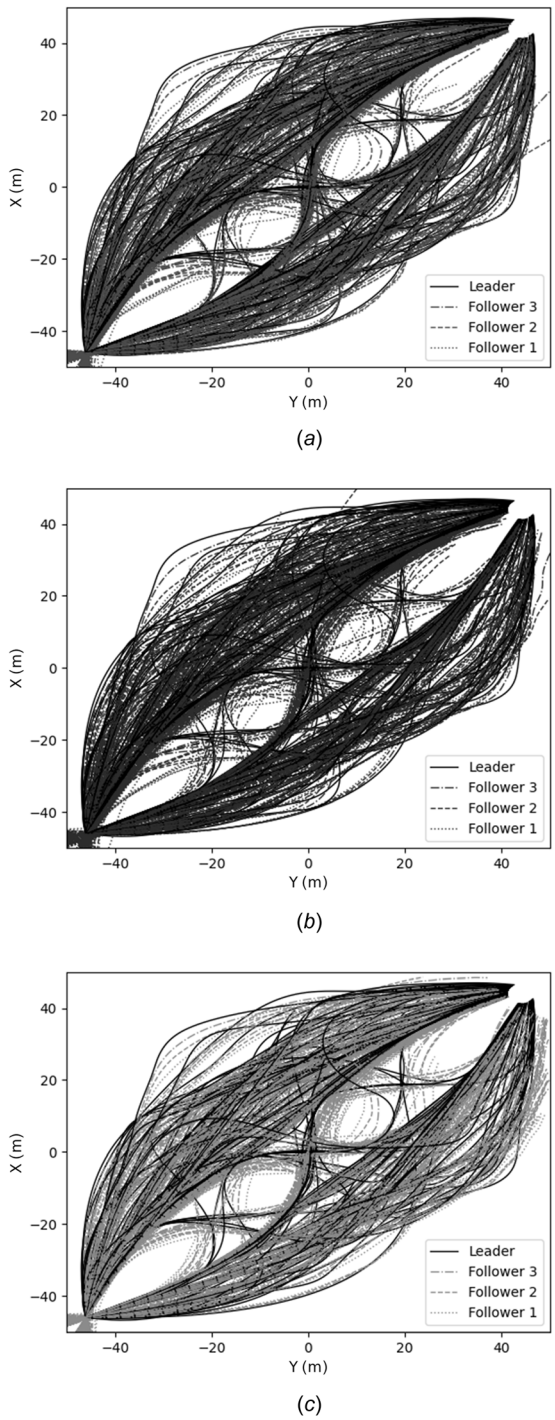


Fig. 19 Path deviation metrics calculation (rigid terrain, Follower 3). The segment of the leader path used in calculations for this particular scenario (rigid terrain) was 158.2m.

The six metrics defined above were evaluated for each of the three follower vehicles in each scenario in the three sets of 128 environments on rigid, SCM-Hard, and SCM-Soft terrain, respectively. The resulting statistics are presented as box-and-whisker diagrams in Figs. 15, 16, and 17, providing measures of the variability in the three statistical populations without any assumption on their underlying statistical distribution (which is unknown, due to the manner in which the sample trajectories were constructed). For each metric, on each terrain type and for each of the three follower vehicles in the 1 L+3F configuration, these standard box plots provide information on their mean, second and third quartiles, as well as minimum and maximum values.

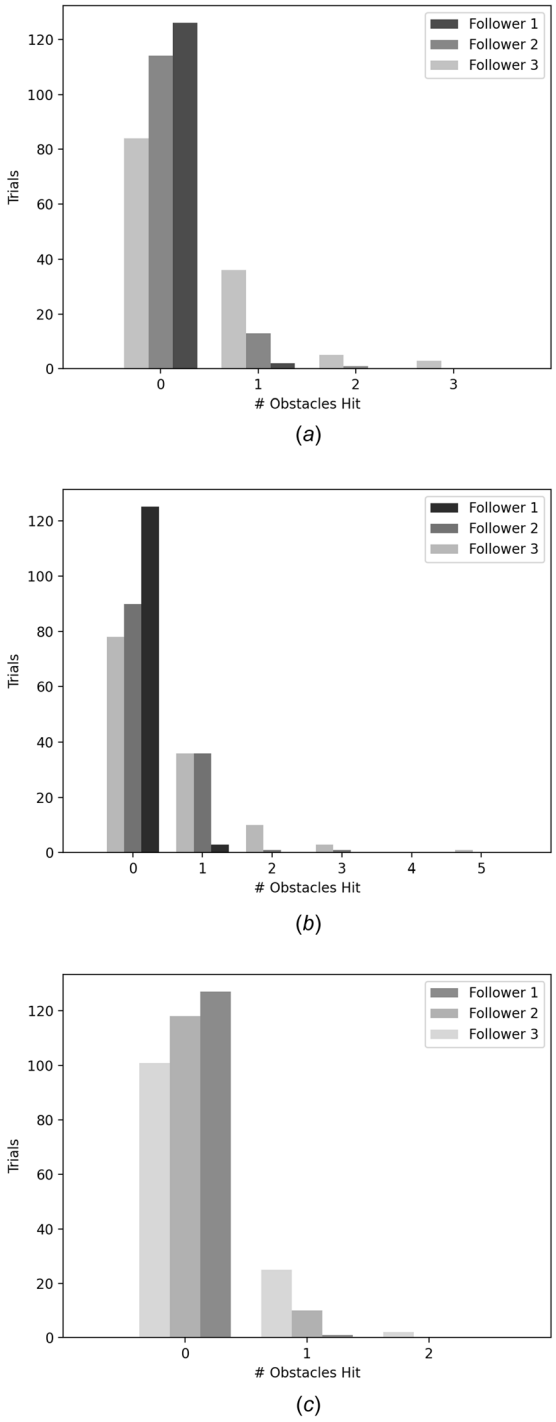


Fig. 20 Number of obstacles hit by each vehicle on the three terrain types

We assume that a perfect control policy for a follower vehicle would result in a convoy in which each follower vehicle runs exactly in the tracks of the vehicle preceding it and achieves the exact same speed at any given location. Given their definition, this ideal case corresponds to zero values for each of the six performance metrics.

The results of the statistical analysis presented herein confirm the intuition that, in a 1 L+3F configuration, a less than perfect control policy will lead to worse performance for the trailing follower vehicles and better performance on harder surfaces (especially taking into account that the training was performed exclusively on rigid terrain). However, as the results for both path and speed deviation show, this is not the case when comparing performance on SCM-Hard and SCM-Soft terrains, with the latter showing consistent lower metrics values (i.e., better performance in terms of maintaining position in the convoy). The explanation for this behavior is likely a combination of several factors. First, even though the target speed profile for the leader vehicle was set identically for all three terrain types, the increased terrain resistance in the SCM-Hard and SCM-Soft cases resulted in the leader vehicle being unable to continuously increase its speed to the specified value; on both deformable terrain types, the vehicles were unable to shift in the higher gears and their speed limited to lower levels than on rigid terrain (an effect more pronounced on SCM-Soft terrain than on SCM-Hard). The ensuing overall lower convoy speed results in driving scenarios where the control policy can adapt better. Second, as seen in Fig. 14 and typical of all simulations conducted as part of this analysis, current deficiencies in the RL-based control policy result in relatively jerky motion of the follower vehicles and noisy speed profiles. These spurious accelerations and decelerations are less pronounced on the SCM-Soft due to the increased motion resistance. Finally, the largest deviations in a follower's path (see Fig. 19) always occur at tight turns where the leader vehicle must go around an obstacle. In these situations, the tendency of the control policy is to "cut corners" and thus direct the vehicle to increase steering input. However, these control steering inputs are more difficult to follow in a deformable terrain soft enough to result in deep ruts, thus resulting in the follower vehicles more closely matching the leader's vehicle path around obstacles.

While the path planning procedure and the path-following PID-based control policy implemented for the leader vehicles ensures that a leader vehicle always avoids obstacles, this is not the case for the RL-based control policy implemented for the follower vehicles, which occasionally are unable to avoid an obstacle (in a few situations, a follower vehicle, especially one in position 2 or 3, may end up going on the opposite side of an obstacle). This behavior has multiple compounding causes, including the particular reward system used in the current training as well as configurations where perception of the leader vehicle is obstructed by an obstacle or the leader vehicle is out of the camera sensor's field of view while negotiating a tighter turn. The paths of all vehicles on each terrain type are shown in Fig. 18. The path information therein is used to gauge the robustness of the control policy in terms of obstacle avoidance. Figure 20 provides the cumulative statistics in terms of number of obstacles hit, over all ensemble simulations, for all three terrain types and for each of the three follower vehicles. These results show the same relative performance trends observed before, with the trailing follower on SCM-Hard terrain exhibiting worst performance.

5 Conclusion and Future Work

This contribution discussed a simulation platform designed to facilitate the design and testing of control policies for AAs operating in off-road conditions. The platform draws on a multibody dynamics simulation engine; has templates for wheeled and tracked vehicles; enforces space and time coherence; allows for human-in-the-loop scenarios; provides sensor simulation capabilities; has a bridge to ROS/ROS2; can simulate mobility on fully

resolved, continuum, or SCM representations of the terrain; is open source; and is cluster-deployable to support multi-AA mobility studies. This software framework is used here to design an end-to-end, RL-based control policy that allows AAs to follow in a convoy formation. The learning took place on rigid terrain but was demonstrated to work when deployed on AAs that operate on deformable SCM soils. The virtual environments used in testing differed in textures and colors from the ones used in the training, thus demonstrating robustness of the inferred policy that relies on inputs from an RGB camera sensor. Unsurprisingly, the fewer AAs in the platoon, the tighter it managed to follow a prescribed path. Looking ahead, we plan to augment the sensing simulation support; improve scalability; and use this simulation infrastructure to derive new control policies for off-road AA mobility.

Funding Data

- Army Research Office DURIP instrumentation grant W911NF1810476 (Funder ID: 10.13039/100000183).
- US Army Research Office project W911NF1910431 (Funder ID: 10.13039/100000183).
- National Science Foundation project CISE1835674 (Funder ID: 10.13039/100000001).
- National Science Foundation project CPS1739869 (Funder ID: 10.13039/100000001).
- U.S. Army GVSC grant W56HZV-08-C-0236 (Funder ID: 10.13039/100006751).
- SAFER-SIM program (Funder ID: 10.13039/100000140).

Acknowledgment

The Euler supercomputer used in this work contains hardware procured through the Army Research Office DURIP instrumentation grant W911NF1810476. Support for terramechanics research is provided through U.S. Army Research Office project W911NF1910431. Ongoing support for core Chrono development is provided by National Science Foundation project CISE1835674. Ongoing support for SynChrono development is provided by National Science Foundation project CPS1739869. Support for the development of Chrono::Vehicle was provided by U.S. Army GVSC grant W56HZV-08-C-0236. Support for the development of Chrono::Sensor and SynChrono has been provided by the SAFER-SIM program, which is funded through a grant from the U.S. Department of Transportation's University Transportation Centers Program (69A3551747131).

References

[1] 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.

[2] Jakobi, N., and Husbands, P., and Inman 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, Springer, pp. 704–720.

[3] Choi, HSun., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., Hearl, F., Hodgins, J., Jain, A., Leve, F., Li, C., Meier, F., Negrut, D., Righetti, L., Rodriguez, A., Tan, J., and Trinkle, J., 2021, "On the Use of Simulation in Robotics: Opportunities, Challenges, and Suggestions for Moving Forward," Proc. Natl. Acad. Sci., **118**(1), p. e1907856118.

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

[5] Koenig, N. P., and Howard, A., 2004, "Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator," Iros, pp. 2149–2154.

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

[7] 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 Software, **3**(22), p. 500.

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

[9] Rohmer, E., Singh, S. P., and Freese, M., 2013, "V-REP: A Versatile and Scalable Robot Simulation Framework," 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, Nov. 3–7, IEEE, pp. 1321–1326.

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

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

[12] Newton Dynamics, 2020, "Newton Dynamics: A Cross-Platform Life-Like Physics Simulation Library," http://newtondynamics.com/forum/newton.php

[13] Jain, A., Balaram, J., Cameron, J., Guineau, J., Lim, C., Pomerantz, M., and Sohl, G., 2004, "Recent Developments in the ROAMS Planetary Rover Simulation Environment," 2004 IEEE Aerospace Conference Proceedings, Vol. 2, Big Sky, MT, Mar. 6–13, IEEE, pp. 861–876.

[14] Quantum Signals, "Anvel:AE - ANVEL UGV Simulator," accessed May 13, 2020, https://quantumsignalai.com/

[15] Abhi, J., "DARTS Dynamics Algorithms for Real-Time Simulation," https://dartslab.jpl.nasa.gov/DARTS/index.php

[16] Durst, P., Goodin, C., Cummins, C., Gates, B., Mckinley, B., George, T., Rohde, M., Toschlog, M., 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, IEEE, pp. 7–10.

[17] 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 Space, epub, pp. 1441–1447.

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

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

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

[21] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y., 2009, "ROS: An Open-Source Robot Operating System," ICRA Workshop on Open Source Software, Vol. 3, Kobe, Japan, May 12–17, p. 5.

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

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

[24] Shah, S., Dey, D., and Lovett, C., and Kapoor, A., 2018, "Airsim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," Field and Service Robotics, Springer, Berlin, pp. 621–635.

[25] Unity3D, Main Website, 2016, accessed Jun. 9, 2016, https://unity3d.com/

[26] Espié, E., Guionneau, C., Wymann, B., and Dimitrakakis, C., 2020, "TORCS – the Open Racing Car Simulator," https://sourceforge.net/projects/torcs/

[27] Rong, G., Shin, B. H., Tabatabaee, H., Lu, Q., Lemke, S., Mozeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., Agafonov, E., Kim, T. H., Sterner, E., Ushiroda, K., Reyes, M., Zelenkovsky, D., and Kim, S., 2020, "Lgsvl Simulator: A High Fidelity Simulator for Autonomous Driving," IEEE 23rd International Conference on Intelligent Transportation Systems, Rhodes, Greece, Sept. 20–23, pp. 1–6.

[28] Kang, Y., Yin, H., and Berger, C., 2019, "Test Your Self-Driving Algorithm: An Overview of Publicly Available Driving Datasets and Virtual Testing Environments," IEEE Trans. Intell. Veh., **4**(2), pp. 171–185.

[29] Rosique, F., Navarro, P. J., Fernández, C., and Padilla, A., 2019, "A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research," Sensors, **19**(3), p. 648.

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

[31] 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*, T. Kozubek, ed., Springer, Berlin, pp. 19–49.

[32] Message Passage Interface Forum, 1994, "MPI: A Message-Passing Interface Standard," University of Tennessee, Knoxville, TN, Report, http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail\&id=oai

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

[34] Amini, A., Gilitschenski, I., Phillips, J., Moseyko, J., Banerjee, R., Karaman, S., and Rus, D., 2020, "Learning Robust Control Policies for End-to-End Autonomous Driving From Data-Driven Simulation," IEEE Rob. Autom. Lett., **5**(2), pp. 1143–1150.

[35] Pan, X., You, Y., Wang, Z., and Lu, C., 2017, "Virtual to Real Reinforcement Learning for Autonomous Driving," Proceedings of the British Machine Vision Conference (BMVC), Sept., Article No. 11.

[36] SAE, 2014, "Taxonomy and Definitions for Terms Related to on-Road Motor Vehicle Automated Driving Systems," SAE Paper No. J3016-201806.

[37] Joshi, A., 2017, "SAE 2017-01-1994: Hardware-in-the-Loop (Hil) Implementation and Validation of Sae Level 2 Automated Vehicle With Subsystem Fault Tolerant Fallback Performance for Takeover Scenarios," Proceedings of SAE Congress.

[38] Folkers, A., Rick, M., and Buskens, C., 2019, "Controlling an Autonomous Vehicle With Deep Reinforcement Learning," 2019 IEEE Intelligent Vehicles Symposium (IV), pp. 2025–2031.
[39] Dikmen, M., and Burns, C. M., 2016, "Autonomous Driving in the Real World: Experiences With Tesla Autopilot and Summon," Proceedings of the 8th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, Automotive-UI 16, New York, NY, USA, Association for Computing Machinery, pp. 225–228.
[40] Qian, X., Navarro, I., de La Fortelle, A., and Moutarde, F., 2016, "Motion Planning for Urban Autonomous Driving Using Bézier Curves and Mpc," 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, Nov. 1–4, pp. 826–833.
[41] Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., and How, J. P., 2009, "Real-Time Motion Planning With Applications to Autonomous Urban Driving," IEEE Trans. Control Syst. Technol., **17**(5), pp. 1105–1118.
[42] Qian, X., Altché, F., Bender, P., Stiller, C., and de La Fortelle, A., 2016, "Optimal Trajectory Planning for Autonomous Driving Integrating Logical Constraints: An Miqp Perspective," 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, Nov. 1–4, pp. 205–210.
[43] Borrelli, F., Falcone, P., Keviczky, T., Asgari, J., and Hrovat, D., 2005, "Mpc-Based Approach to Active Steering for Autonomous Vehicle Systems," Int. Journal Vehicle Autonomous Systems, **3**(2/3/4), pp. 265–291.
[44] Liniger, A., Domahidi, A., and Morari, M., 2017, "Optimization-Based Autonomous Racing of 1: 43 Scale rc Cars," *ArXiv*, abs/1711.07300.
[45] Huang, W., Wang, K., Lv, Y., and Zhu, FHua., 2016, "Autonomous Vehicles Testing Methods Review," 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, Nov. 1–4, pp. 163–168.
[46] Kuutti, S., Bowden, R., Jin, Y., Barber, P., and Fallah, S., 2019, "A Survey of Deep Learning Applications to Autonomous Vehicle Control," ArXiv, abs/1912.10773,
[47] Kober, J., Bagnell, A., and Peters, J., 2013, "Reinforcement Learning in Robotics: A Survey," Int. J. Rob. Res., **32**(11), pp. 1238–1274.
[48] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A., 2013, "Playing Atari With Deep Reinforcement Learning," *CoRR*, abs/1312.5602,
[49] Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D., 2016, "Learning Hand-Eye Coordination for Robotic Grasping With Deep Learning and Large-Scale Data Collection," CoRR, abs/1603.02199,
[50] Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W., 2020, "Learning Dexterous in-Hand Manipulation," Int. J. Rob. Res., **39**(1), pp. 3–20.
[51] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., 2017, "Proximal Policy Optimization Algorithms," CoRR, abs/1707.06347,
[52] Sutton, R. S., and Barto, A. G., 1998, *Introduction to Reinforcement Learning*, 1st ed., MIT Press, Cambridge, MA.
[53] Kakade, S., and Langford, J., 2002, "Approximately Optimal Approximate Reinforcement Learning," Proceedings of 19th International Conference on Machine Learning, San Francisco, CA, July 8–12, pp. 267–274.
[54] Tasora, A., Mangoni, D., Negrut, D., Serban, R., and Jayakumar, P., 2019, "Deformable Soil With Adaptive Level of Detail for Tracked and Wheeled Vehicles," Int. J. Veh. Performance, **5**(1), pp. 60–76.
[55] Project Chrono Development Team, "Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems," accessed Dec. 7, 2019, https://github.com/projectchrono/chrono
[56] ECMA, 2013, "The JSON Data Interchange Format," ECMA International, Report No. ECMA-404.
[57] Recuero, A. M., Serban, R., Peterson, B., Sugiyama, H., Jayakumar, P., and Negrut, D., 2017, "A High-Fidelity Approach for Vehicle Mobility Simulation: Nonlinear Finite Element Tires Operating on Granular Material," J. Terramech., **72**, pp. 39–54.
[58] Serban, R., Negrut, D., Recuero, A. M., and Jayakumar, P., 2019, "An Integrated Framework for High-Performance, High-Fidelity Simulation of Ground Vehicle-Tyre-Terrain Interaction," Int. J. Veh. Performance, **5**(3), pp. 233–259.
[59] Hu, W., Rakhsha, M., Yang, L., Kamrin, K., and Negrut, D., 2021, "Modeling Granular Material Dynamics and Its Two-Way Coupling With Moving Solid Bodies Using a Continuum Representation and the SPH Method," Comput. Methods Appl. Mech. Eng., **385**, p. 114022.
[60] Elmquist, A., and Negrut, D., 2020, "Methods and Models for Simulating Autonomous Vehicle Sensors," IEEE Trans. Intell. Veh., **5**(4), pp. 684–692.
[61] 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 Trans. Graph., **29**(4), pp. 1–13.
[62] Elmquist, A., Serban, R., and Negrut, D., 2021, "A Sensor Simulation Framework for Training and Testing Robots and Autonomous Vehicles," J. Auton. Veh. Syst., **1**(2), p. 021001.
[63] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yuan, Y., and Zheng, X., 2016, "Tensorflow: A System for Large-Scale Machine Learning," 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283.
[64] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A., 2017, "Automatic Differentiation in Pytorch," NIPS 2017 Workshop Autodiff. Report.
[65] Theano Development Team, 2016, "Theano: A Python Framework for Fast Computation of Mathematical Expressions," arXiv e-prints, abs/1605.02688.
[66] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Caffe, T. D., 2014, "Convolutional Architecture for Fast Feature Embedding," Proceedings of the 22nd ACM International Conference on Multimedia, 22, pp. 675–678.
[67] Beazley, D. M., 2003, "Automated Scientific Software Scripting With SWIG," Future Gener. Comput. Syst., **19**(5), pp. 599–609.
[68] Project Chrono Development Team, "PyChrono: A Python Wrapper for the Chrono Multi-Physics Library," accessed Apr. 29, 2020. https://anaconda.org/projectchrono/pychrono
[69] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., 2016, "OpenAI Gym," CoRR, abs/1606.01540.
[70] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard Version 3.0, 09 2012," Chapter author for Collective Communication, Process Topologies, and One Sided Communications, Report.
[71] Gonzales, D., and Harting, S., 2014, "Designing Unmanned Systems With Greater Autonomy: Using a Federated, Partially Open Systems Architecture Approach," *RAND Corporation*, CA, Santa Monica.
[72] Pardo-Castellote, G., 2003, "OMG Data-Distribution Service: Architectural Overview," Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops, Providence, RI, May 19–22, pp. 200–206.
[73] BallingMcCullough, O., Hodges, M., Pulley, H. R., and Jayakumar, P., 2018, "Tracked and Wheeled Vehicle Benchmark – a Demonstration of Simulation Maturity for Next Generation NATO Reference Mobility Model," Ground Vehicle Systems Engineering and Technology Symposium, Novi, MI.
[74] Serban, R., Gerike, R., Elmquist, A., and Negrut, D., 2017, "NG-NRMM Phase II Benchmarking: Chrono Wheeled-Vehicle Platform Simulation Results Summary," Simulation-Based Engineering Laboratory, University of Wisconsin-Madison, Report No. TR-2017-05. http://sbel.wisc.edu/documents/TR-2017-05.pdf
[75] Serban, R., Taylor, M., Melanz, D., and Negrut, D., 2016, "NG-NRMM Phase I Benchmarking: Chrono Tracked Vehicle Simulation Results Summary," Simulation-Based Engineering Laboratory, University of Wisconsin-Madison, Report No. TR-2016-08, http://sbel.wisc.edu/documents/TR-2016-08.pdf
[76] Project Chrono, 2020, Off-road AV simulations, https://uwmadison.box.com/s/glbpqxpomgyiomt2ydctpe35avrh44vd
[77] Project Chrono, 2020, GVSETS paper simulations, https://uwmadison.box.com/s/glbpqxpomgyiomt2ydctpe35avrh44vd
[78] Adam, D., "Draw Randomly Centered Circles of Various Sizes," accessed June 17, 2020, https://www.mathworks.com/matlabcentral/fileexchange/70348-draw-randomly-centered-circles-of-various-sizes
[79] Yarpiz, Path planning using PSO in MATLAB, accessed June 17, 2020, https://www.mathworks.com/matlabcentral/fileexchange/53146-path-planning-using-pso-in-matlab