2020 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
Autonomy, Artificial Intelligence & Robotics (AAIR) Technical Session
August 11-13, 2020 – Novi, Michigan

ENABLING ARTIFICIAL INTELLIGENCE STUDIES IN OFF-ROAD MOBILITY THROUGH PHYSICS-BASED SIMULATION OF MULTI-AGENT SCENARIOS

D. Negrut, R. Serban, A. Elmquist, J. Taves, A. Young
University of Wisconsin – Madison
Madison, WI

A. Tasora, S. Benatti
Dipartimento di Ingegneria ed Architettura
University of Parma, Italy

ABSTRACT

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 design and assessment of a reinforcement learning policy that uses sensor fusion and inter-agent communication to enable the movement of mixed convoys of conventional and autonomous vehicles. Policies learned on rigid terrain are shown to transfer to hard (silt-like) and soft (snow-like) deformable terrains. The enabling simulation environment, which is Chrono-centric, is used as follows: the training occurs in the GymChrono learning environment using PyChrono, the Python interface to Chrono. The GymChrono-generated policy is subsequently deployed for testing in SynChrono, a scalable, cluster-deployable multi-agent testing infrastructure that uses MPI. The Chrono::Sensor module simulates sensing channels used in the learning and inference processes. The software stack described is open source. Relevant movies: [1].

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 [2]. 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 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 is not a silver bullet as it has its limitations, first of all the issue of simulation-to-reality transfer [3], 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, and open source simulation tools that are both predictive and expeditious are not readily available. This contribution addresses the third 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 towards: improving mechanical designs of AAs; and, producing and testing control policies that govern the operations of the AAs. These are topical goals for the Army, see, for instance, [4, 5, 6, 7], or, for a historical perspective on early contributions, [8].

There are several ongoing efforts that seek to address the AA simulation issue. In robotics, Gazebo [9, 10] is a widely used 3D multi-robot simulator with dynamics. It is not a simulation engine per se, but a *platform* that exposes several engines: ODE [11], Bullet [12], DART [13], and Simbody [14]. Unlike Gazebo, which is open source, CoppeliaSim (formerly V-REP) [15] is a commercial multi-robot simulation solution that also exposes a set of simulation engines: MuJoCo [16], Vortex Dynamics [17], Bullet, and Newton Dynamics [18]. ROAMS [19] and ANVEL [20] are two other simulation engines for off-road AAs. The former is used for mission planning by NASA and draws on an in-house dynamics engine [21]; the latter relies on the ODE simulation engine and has been used in the past for off-road military applications [22] in combination with a sensor simulation package [23]. MAVS is an off-road AA simulation environment that is currently under active development [24]. It provides an in-house developed, sophisticated sensor simulation module [25, 26], has a ROS bridge, and uses Chrono as its dynamics engine. USARSim is an AA simulation platform, not under active development, that draws on a game engine (Unreal Engine [27]), 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 [2] and AirSim [28] are two often used open-source simulators, the former for on-road AV driving scenarios simulation, the latter originally designed for drones but now including support for on-road traffic of AVs as well. Carla and AirSim rely on Unreal Engine but several other engines are used for AA simulation, e.g. Unity [29] and TORCS [30]. For a survey of other solutions for on-road mobility the reader is referred to [31, 32].

The AA off-road mobility simulation platform discussed here is Chrono-centric [33, 34]. 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 multi-physics engine; it is scalable and deployable on supercomputers, clusters, or multi-core architectures owing to its reliance on the MPI message passing standard for parallel computing [35]; 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::Parallel and Chrono::Granular) applications. It has Python bindings, support for sensor simulation in Chrono::Sensor, an API for ROS [36] bridging, as well as facilities for: rapid vehicle modeling via parameterized templates [37]; control policy design with GymChrono; and scalable control policy testing with SynChrono. Chrono relies on GPU computing for fluid-solid interaction and select granular dynamics simulations, multi-core for most of the other modules, and MPI-enabled parallel computing for co-simulation when handling large terramechanics applications or collections of AAs. Real-time simulation is not one of Chrono's priorities. Although for vehicle-on-rigid-terrain simulation it provides faster than real-time performance, there are numerous applications that lead to long run times in Chrono, 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 Section §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. Concluding remarks and directions of future work round off the contribution.

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 [38, 39]. The use of vehicles with Level 1 and Level 2 autonomy has grown considerably [40, 41] and the automotive industry is making major strides in the transition to Levels 3 and 4 autonomy [42, 43]. 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 [44, 45]. Graph search methods are commonly associated with each trajectory. The approach is real-time challenged, since achieving robust results requires a high number of samples to be analyzed [46]. Algorithms such as Dijkstra's, A-Star (A*), or the Rapidly-exploring Random Tree-Star (RRT*), sample the state space either deterministically or stochastically [45]. Depending on the complexity of the traffic scenario, these algorithms can prove computationally costly and provide sub-optimal results.

Model Predictive Control (MPC) is another common AV control approach [47]. Using a dynamic model of a vehicle, the MPC algorithm computes trajectories over the state space and determines an optimal trajectory using gradient-descending optimization techniques [46, 48]. 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 show improved performance that is tied to the use of the gradient fields in the optimization problem that comes into play [47].

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 such as MPC, the simulation must be of high enough definition to carry over successfully to reality [49]. When using the more pedestrian 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 [50]. An emerging approach that has gained momentum in recent years is Machine Learning (ML) based [51]. 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 suc-

cessful, as it displays the ability to learn and respond in complex scenarios without the need for preprocessed or labeled data [39].

Since its introduction [52], DRL has proven successful in robotics applications [53, 54]. 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 a measure of 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; it 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), θ are the weights and biases of the NN, the state is the NN input, and the action is its output. Proximal Policy Optimization (PPO) [55] is one of the most widely used algorithms for continuous state and action environments and will be 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 [56] employed to estimate the Advantage Function [57]. The Advantage Function is used in the objective function to be maximized.

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 a battery of tests to gauge its correctness and robustness. This section outlines the components of this Chrono-centric simulation environment. The emphasis is placed on four components: Chrono::Sensor, PyChrono, GymChrono and SynChrono, that anchor the AA design and testing process and have not been discussed elsewhere in detail. More established Chrono components or functionality will be touched upon in passing; more details are provided in [34, 37, 58].

Chrono. Under active development for over two decades, Chrono [34] is a multi-physics 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 (multi-core, distributed, and GPU) for large-scale simulations.

Written almost entirely in C++, Chrono is middleware and therefore supports customized solutions that involve user code and potentially third-party software. The software is portable and can be built on different platforms, under different operating systems, and using various compilers. Chrono is actively developed, has a continuous integration process, an active user forum, and is managed through GitHub [59]. It's latest release is 5.0.1, available as of March 2020. Chrono is used by academic, industrial, and government research

and development groups and projects.

Chrono::Vehicle. Chrono::Vehicle [37] is a specialized Chrono module that makes available 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 vehicle *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, 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 [60]. For additional flexibility and to allow integration of third-party software, Chrono::Vehicle is designed to permit either monolithic simulations or co-simulation 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

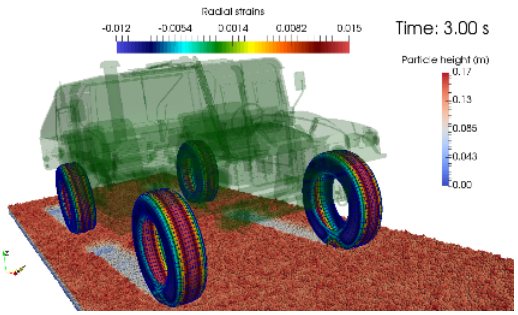


Figure 1: Chrono::Vehicle HMMWV with flexible tires navigating granular terrain demonstrating vehicle dynamics, flexible body dynamics, and parallel computing support in Chrono [61].

finite-element based soil representation. For simple terramechanics simulations, Chrono::Vehicle implements a customized Soil Contact Model (SCM), based on Bekker theory, with extensions to allow non-structured triangular grids, adaptive mesh refinement, and incorporation of bulldozing effects [58]. 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 multi-core, 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 [61, 62].

Chrono::Sensor. Chrono::Sensor provides sensor simulation support for software-in-the-loop testing. Cameras, lidar, GPS, and IMU sensors 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 corre-

sponding real sensors. For modeling convenience, sensors can be defined through a JSON file [60]. Additionally, custom sensors and post-processing filtering can be implemented, leveraging the existing render framework or physics interface.

For interoceptive sensing (GPS and IMU), 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 (camera and lidar), Chrono::Sensor leverages hardware accelerated ray tracing through the OptiX library [63] 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 post-processing noise augmentation are supported, with an interface to extend or implement custom models. For lidar, the framework expands on work from [25] 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.

Chrono::Sensor is currently in development with planned expansion of sensor support and additional distortion model implementations. 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 (in the cloud).

PyChrono. While the main Chrono API is expressed in C++, in recent years a concerted effort was dedicated to providing automatically-generated Python wrappers for much of the Chrono functionality. The purpose was twofold: to provide a lower-entry point to Chrono simulations for users less familiar with C++; and, provide a bridge to various machine learning platforms, e.g. TensorFlow [64], PyTorch [65], Theano [66], and CAFFE [67].

The Python wrapping relies on using automated technology provided by SWIG [68], which generates 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 visualization with Irrlicht, etc. In particular, full support is available for both modeling, simulation, and visualization of wheeled ground vehicles and use of the Chrono::Vehicle existing wheeled vehicle models, as well as using the sensor models provided by Chrono::Sensor. PyChrono for Python 3 can be built from sources on any of the supported platforms (Linux, Windows, MacOS). Alternatively, pre-built conda PyChrono packages are available on the project's Anaconda page [69] (note that Chrono::Sensor is not available yet via the conda PyChrono packages).

GymChrono. GymChrono is an extension of OpenAI Gym [70]. 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 leverag-

ing the Message Passing Interface (MPI) standard [71], SynChrono can manage multiple instances of Chrono running together *one* mobility analysis on a supercomputer, cluster, or multi-core 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 nodes in a supercomputer. By the same token, if there are 50 agents and 50 compute nodes in a cluster, SynChrono provides the infrastructure to keep the 50 agents operating in a coherent (timewise and spacewise) virtual world. Although each agent represents a Chrono simulation, SynChrono makes it possible for ruts generated by agent 27 to be picked up on a camera sensor on agent 31 if the camera points to agent 27. 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 50 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 in real life.

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 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

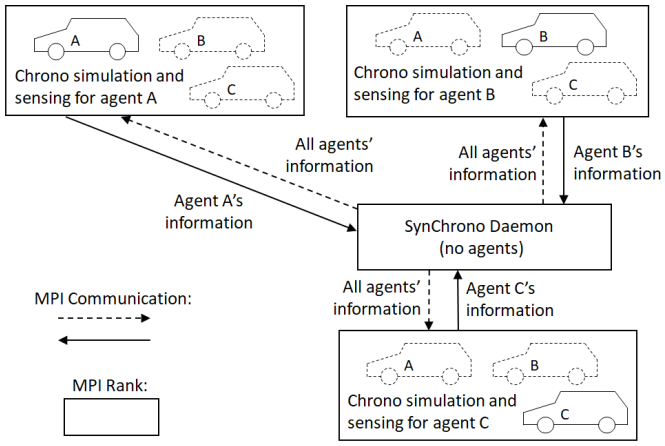


Figure 2: Schematic of the SynChrono framework. Dynamics simulations are done in separate Chrono systems and the outcome of the dynamics simulation is synchronized between ranks using MPI.

vary from complex algorithms that fuse sensor feeds/data streams, to controls based on empirical models, and on to pre-recorded driver inputs from a human or human-driven control in scenarios that are simple enough to allow real-time simulation. SynChrono supports human-in-the-loop experiments as well.

Each SynChrono process is responsible for the dynamics of a single agent. At a slow frequency (relative to the simulation time-step), all SynChrono processes communicate via the SynChrono daemon to exchange state information. State information is intended to be brief, 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 via the FlatBuffers serialization library [72].

One limitation of the implementation is that currently two agents running as two SynChrono processes cannot crash with each other or participate in an operation that couples their dynam-

ics, e.g. lifting together a heavy object. Such a scenario should be run in Chrono, since the coupling is too tight to be handled by 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 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. The goal of SynChrono is to have virtually constant scaling up to a high threshold where the data passing overhead becomes the simulation bottleneck.

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 [36]), 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 as part of the Next Generation-NRMM (NG-NRMM) exercise [73]; Chrono::Vehicle-specific benchmark findings are detailed in [74, 75]. Note that: the NG-NRMM benchmarking effort involved Chrono::Vehicle but no the particular model used in this study; and, 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 $N + 1$ processes executing on a cluster/supercomputer to simulate the dynamics of N agents handled as N independent Chrono simulations. Each Chrono simulation is a process; the extra process is involved in maintaining the time and space coherence for the virtual world shared by the N AAs. The SynChrono daemon executes as the extra process and it coordinates via MPI the dynamics of the N agents. 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 when an SCM deformable terrain model is considered in the simulation scenario? The Linux cluster used exposed 15 nodes. Thus, one set of experiments was conducted with up to $N = 14$ AAs, using rigid, SCM hard (silt-like), and SCM soft (snow-like) terrain, with one agent per compute node. However, the experiment also included scenarios with up to 56 AAs. In this case, four AAs ran on the same compute node. On the upside, this allowed more AAs to participate in the benchmark; on the downside, the simulation times went up since the hardware allocation per AA went down by a factor of four.

The handling of a virtual world that has SCM terrain is challenging since each of the N vehicles changes the terrain at the same time and these changes must be space and time coherent. This is facilitated and managed by the SynChrono daemon. The key component of the SCM terrain is the deformation of each vertex in the underlying SCM mesh. All other terrain properties can be computed based on the height of each vertex alone [58]. 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, 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 daemon to every other agent's world. Each agent then applies the deformations to their own copy of the SCM mesh and resets their collection of new mesh deformations. This means that two agents should not come close enough that they deform the same vertices during the same synchronization period, but this is just as restrictive as SynChrono's assumption that any two agents will not interact by crashing. While Chrono's SCM implementation allows for meshes that auto-refine, this is currently not supported in SynChrono due to the significant increase in algorithmic complexity for relatively little gain in performance. The main concern for scalability is message size, since for synchronizing SCM terrain there is about 15 times as much data to send for each agent relative to the rigid terrain case.

The scenario discussed herein is that of many vehicles crossing perpendicularly on a rectangular patch of SCM terrain, see Fig. 3 and online movies [1]. 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 Fraction (RTF), representing the amount of wall-time required to finish a simulation divided by the amount of time simulated. Running in real-time corresponds to a factor of 1, while slower than real-time corresponds to factors larger than one. The tests were run on the Euler computing cluster at the University of Wisconsin-Madison. Each node has an eight core, Broadwell-generation Intel processor; inter-node communication is facilitated via a Gigabit Ethernet interconnect.

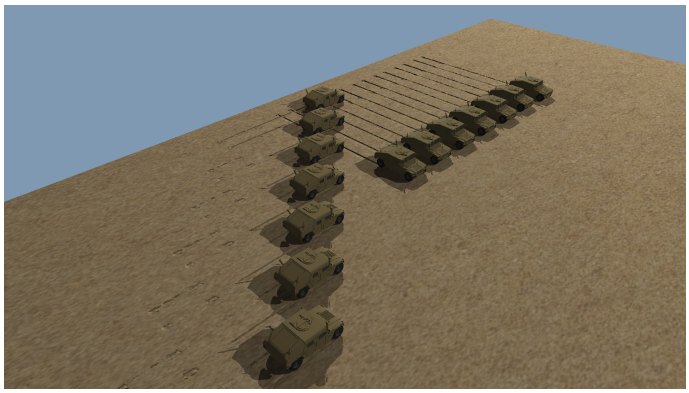


Figure 3: Environment used for SynChrono scaling analysis for agents operating on SCM terrain. Two lines of vehicles move across a rectangular patch, crossing orthogonally and making ruts in the SCM soil. Simulations were run on both soft and hard SCM terrain, as well as on rigid terrain.

In a first sub-experiment, we assign each SynChrono process (which runs one vehicle) to distinct cluster nodes. The scaling analysis results are presented in Fig. 4. The plot shows on the y-axis the log_{10} of the RTF. Thus, any dot below the horizontal x-axis is associated with an experiment that ran faster than real time. Simulations on the rigid terrain ran with an RTF of 0.75 and simulations on either type of SCM terrain (hard or soft) ran with an RTF of approximately 110. The RTF value of 110 is independent of the of SCM soil parameters (i.e. soft vs. hard), but is highly dependent on the processor performance, MPI implementation, compiler, and compilation optimization. These results confirm weak scaling behavior: no significant performance penalty is observed for including additional agents in the simulation when each SynChrono process is assigned to a different compute node.

Using the same scenario as above, scaling analyses were also conducted for a larger number of agents on rigid terrain to ensure that the same weak scaling continued to hold. To this end, two other sub-experiments were conducted: two Syn-

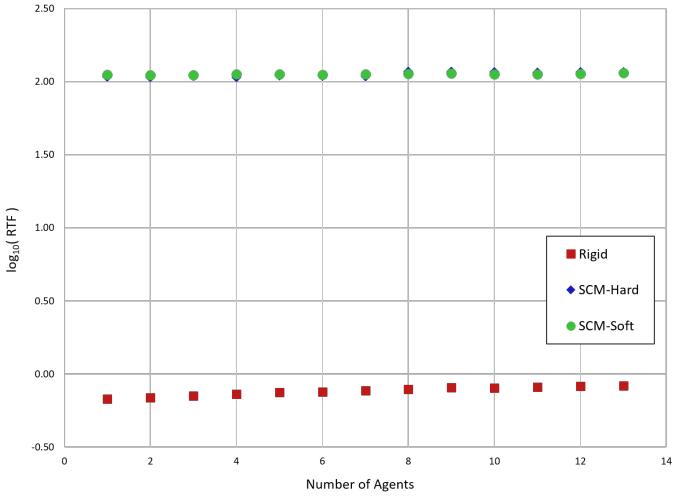


Figure 4: SynChrono scaling analysis for rigid, SCM hard, and SCM soft terrains. Logarithmic scale used.

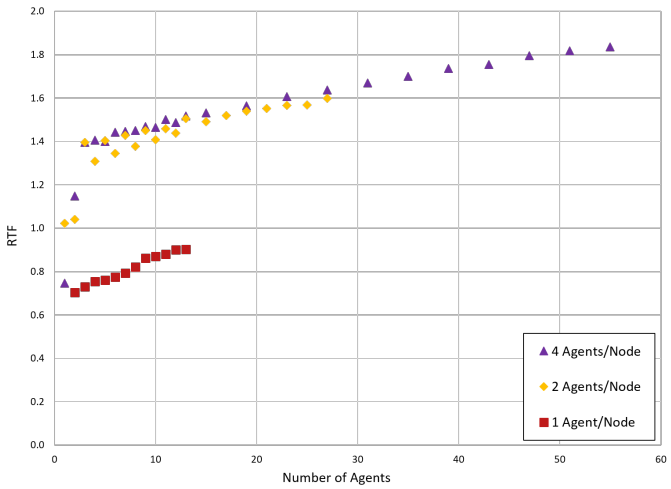


Figure 5: SynChrono scaling analysis on rigid terrain for a large number of agents.

Chrono processes were deployed per node; and, four SynChrono processes were deployed per node. Given the number of nodes made available on the cluster (15), for two-per-node, SynChrono handled 28 AAs; for four-per-node, it managed a joint simulation of 56 AAs. Timing results are shown in Fig. 5, which shows that 56 AAs run in slightly more than 1.8× real time. Given a supercomputer or larger cluster, SynChrono opens the possibility of simulating swarms of AAs.

4.2 Learning to drive in a convoy

Chrono helps with two tasks: learning a control policy, and testing a policy that was designed in Chrono or brought from outside. In this example, PyChrono and GymChrono are used to design a policy. 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 are: three lead vehicles and one following vehicle (3L+1F), two lead and two followers (2L+2F), and one lead and three followers (1L+3F). The *lead* vehicles are programmed 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 not crash in the vehicle ahead of it, and avoid hitting obstacles in the vicinity of the path. To this end, it relis 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 would roughly require four times longer to complete it. 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 but the policy is tested on deformable terrain that is white (snow-like); this is an end-to-end policy that uses sensor data fusion to control both the steering and acceleration/braking of the vehicle.

Designing a policy. The policy was obtained through training using a custom implementation of the PPO reinforcement learning algorithm leveraging PyTorch [65] 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 HMMWV) to drive in a convoy. For training, to increase the randomness of the path and thus the robustness of the control, two different path types were used on a 90x90 meters area. The first is S-shaped, starting from a 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 can be flipped along the y-axis (left-right) and rotated about the vertical z-axis 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 $\times$ 45 pixels. This level of resolution suffices, since detailed features that can be extracted from higher resolution images are 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.

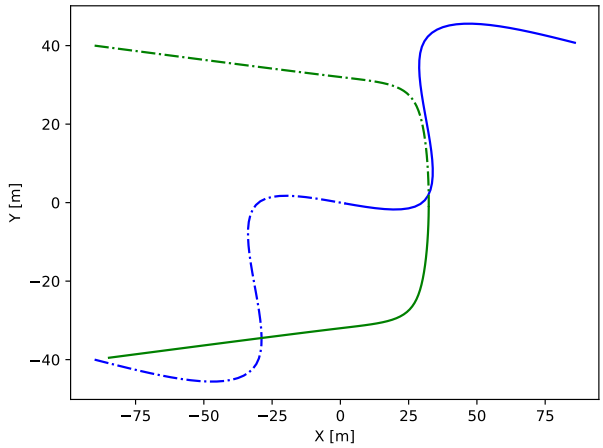


Figure 6: The double S and C paths used during training. Each one of these is randomly flipped and rotated, resulting in 16 different possible paths. The dashed lines represent the segment of the path in which the simulation episode can start.

As with any other RL environment, an observation and a reward is provided to the ML algorithm at each timestep. 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: the first is a steering value; the second element is 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 and because throttling and braking both directly control the vehicle's acceleration.

The agent is rewarded only when it successfully stays behind the leader, meaning a reward is provided when the angle between the heading of the leader and the follower is in the range $[-\pi/4, \pi/4]$. When this condition is met, the agent receives the maximum reward if it keeps an optimal distance (within a prescribed tolerance) from the lead vehicle. In other words, the follower is rewarded when

it stays in a sector of an annulus centered at the leader. When the vehicle is outside the desired area, the reward decreases hyperbolically.

The learning draws on information from several sensors, whose output is organized into two tuples. The first element is a $80 \times 45 \times 3$ 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 multi-sensor 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 (CNN) as in [52]. Its output is then concatenated with the output of the one Fully Connected (FC) hidden layer Deep Neural Network (DNN) 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. 7.

RL-based training requires a very large number of iterations. Three decisions helped speed up the learning process: (*i*) the lead 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. 8) to further refine the NN parameters; and (*iii*) the learning process was accelerated using the OpenAI [70] Baselines tool for environment parallelization, thus allowing several simulations running simultaneously to speed up the collection of the dataset samples.

Testing of learned policy. 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 3L+1F, 2L+2F, and 1L+3F. The terrains were rigid, SCM hard (silt-like), and SCM soft (snow-like). This leads 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 1L+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 was attached to the last vehicle in the convoy in order to visualize the simulation. Still frames captured with this camera are shown in Fig. 9 and full-length videos of representative simulations are available online [1]. Different ground textures and colors were used to further differentiate between the three terrain types (rigid, SCM-Hard, and SCM-Soft).

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 §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 present next 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 1L+3F convoy configuration and each one of the three sets of 128 simulation used the same set of randomly-generated trajectories. The performance metrics used in this analysis measure the

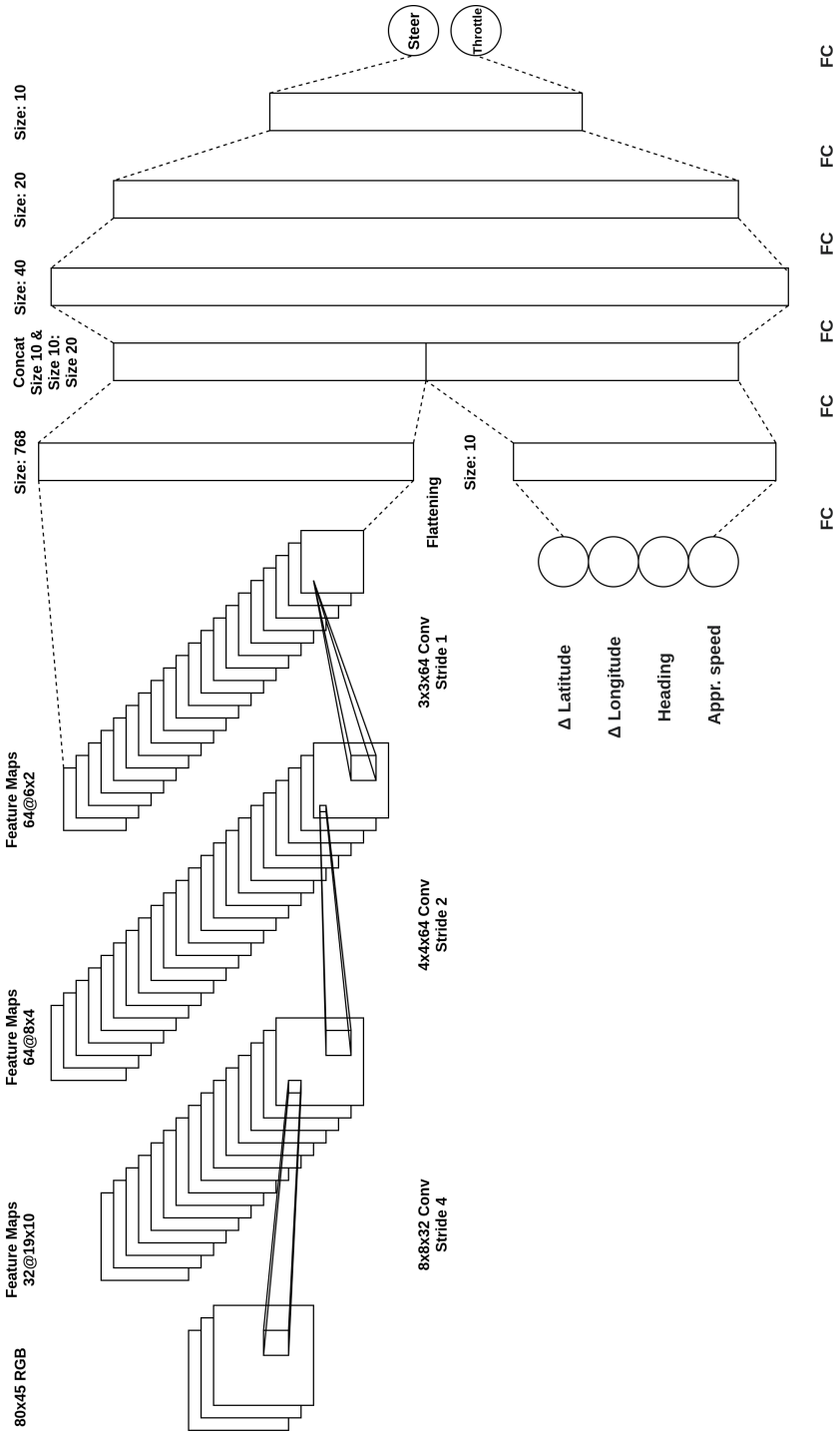
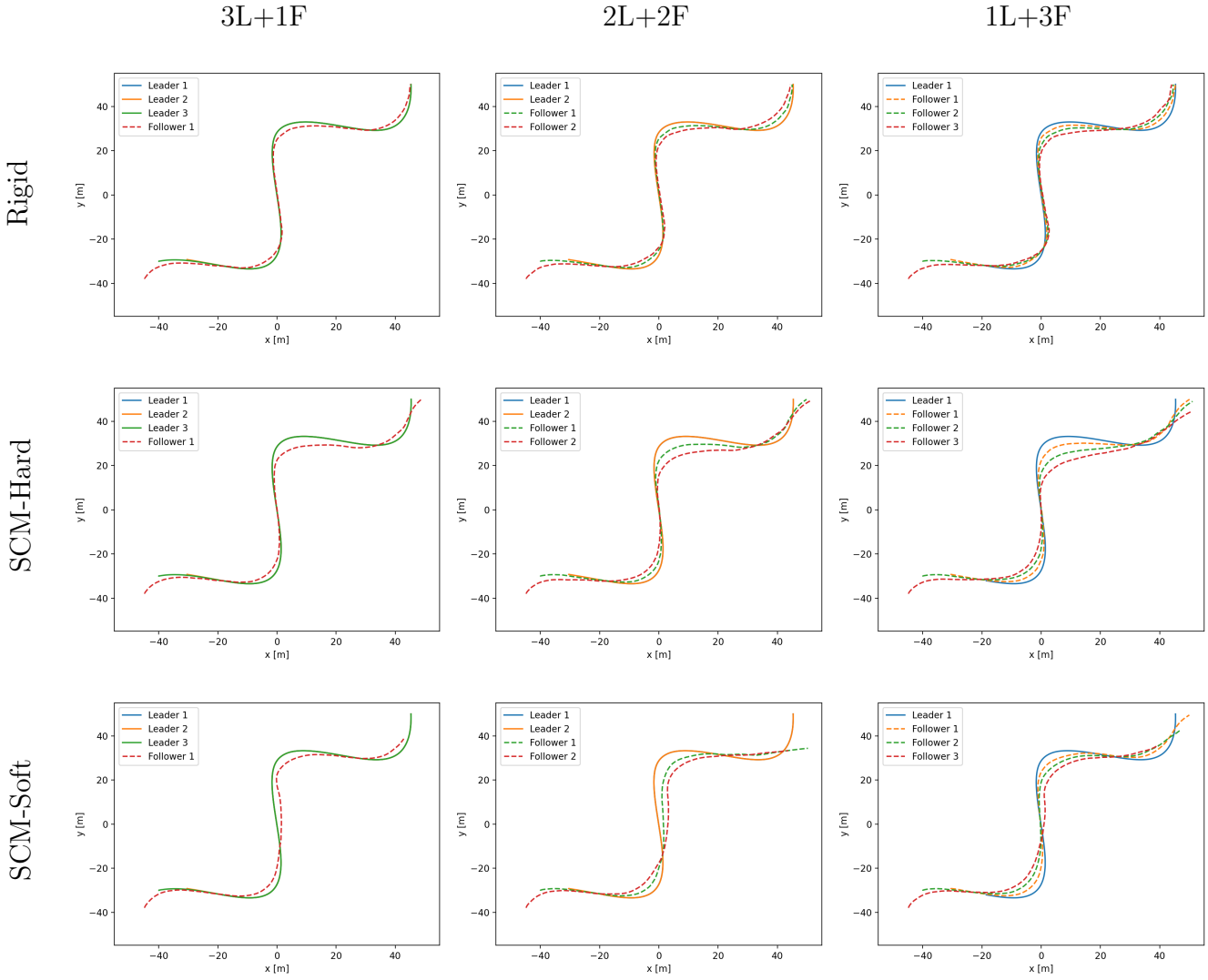


Figure 7: Sensor Fusion NN architecture.

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



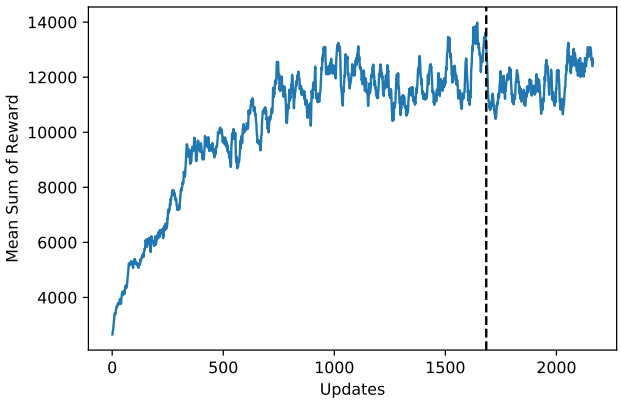


Figure 8: Plot of the moving average of the sum of collected rewards with respect to the policy updates (updated every 1500 interactions). The vertical dashed line represents the switch from the reduced to the full HMMWV model.

path and speed deviation of a follower vehicle from that of the convoy's leader.

The 128 distinct scenarios were created in a 100 m $\times$ 100 m swath of flat terrain. A script generated 30 randomly placed circular obstacles with sizes uniformly distributed between 5 m 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 [76]. Finally, 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 equidistant along the diagonal from start to end.

A path was designed to get from start to finish while avoiding the obstacles. The path start and end positions were fixed to the south-west and north-east corners, respectively. A sample obstacle field is shown on the left in Fig. 10. A Particle Swarm Optimization (PSO) algorithm [77] was used to generate a shortest distance path connecting the start and end locations, as shown in the middle image of Fig. 10. The path generation was constrained to produce trajectories that remained in the domain and did not intersect obstacles.

Each of the 128 resulting paths shown in Fig. 11 was 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).

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 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 minimum width of 4 m. An overhead view of the resulting SynChrono simulation environment is shown on the right in Figure 10.

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

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 12 illustrates this process for the same trajectory used as an example before. The sample results used in this description are results of a simulation on rigid terrain and consider 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 de-

(a) (b) (c)

Figure 9: Still frames from attached third person camera: (a) rigid terrain; (b) SCM-Hard terrain; (c) SCM-Soft terrain.

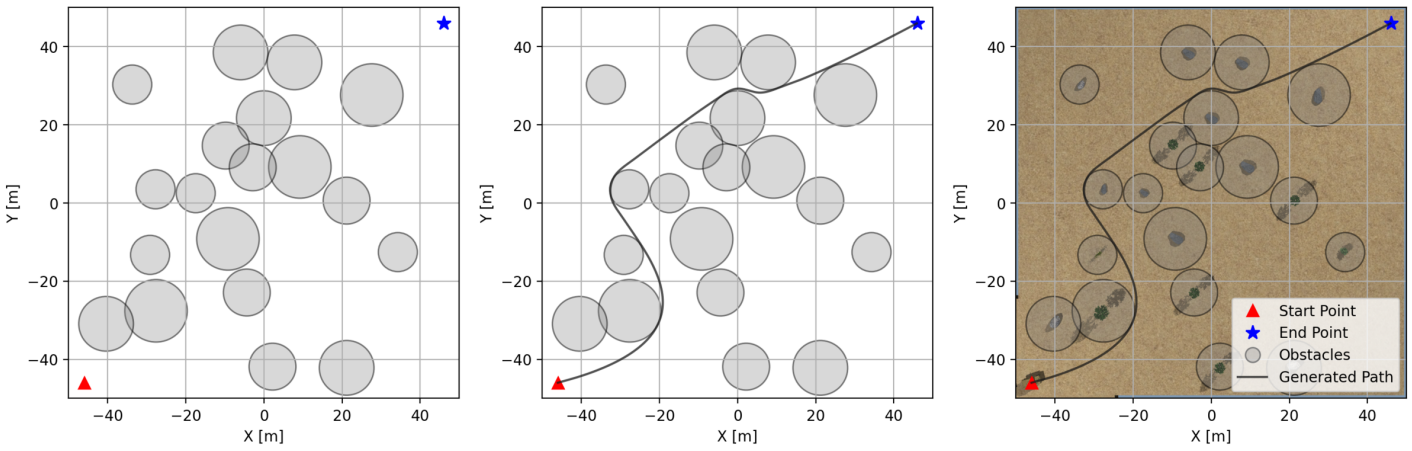


Figure 10: Sample obstacle field, PSO-based path planning, and the corresponding SynChrono environment setup.

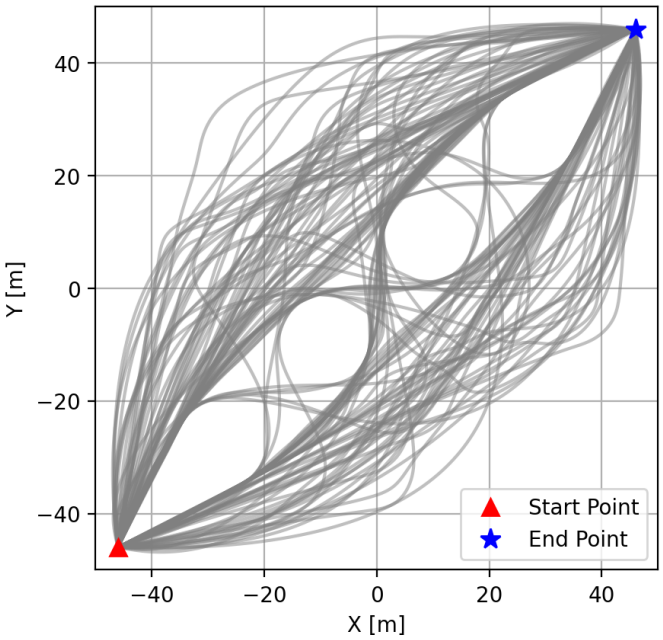


Figure 11: Ensemble of the 128 paths used in the statistical analysis.

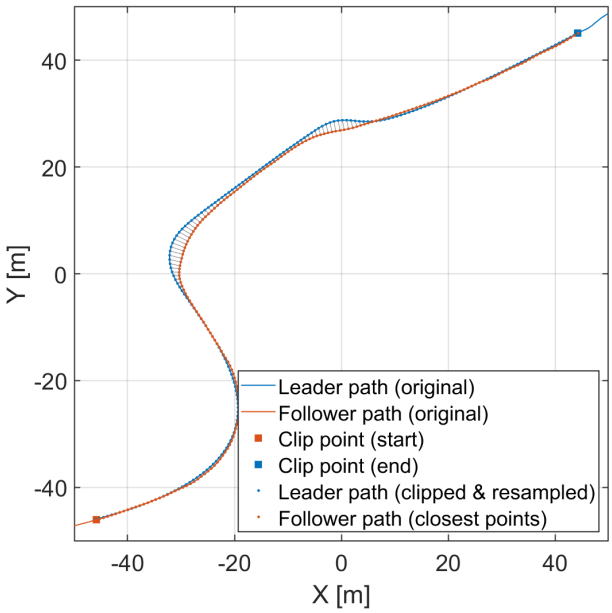


Figure 12: Path deviation metrics calculation (rigid terrain, Follower 3). The segment of the leader path used in calculations for this particular scenario (rigid terrain) was $L = 158.2$ m.

fined 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 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 (and not 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.

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

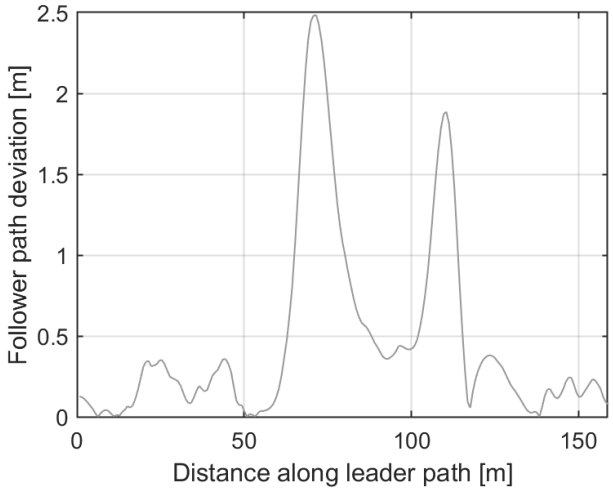


Figure 13: Deviation in path between Leader and Follower 3. The lateral deviation metrics for this particular scenario (rigid terrain) were $m_p^{avg} = 0.473$ m, $m_p^{max} = 2.484$ m.

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 1L+3F configuration, these standard box plots provide information on their mean, second and third quartiles, as well as minimum and maximum values.

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 1L+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 inhibited on the SCM-Soft due to the increased motion resistance. Finally, the largest deviations in a follower's path (see Fig. 12) 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 few situations, a follower

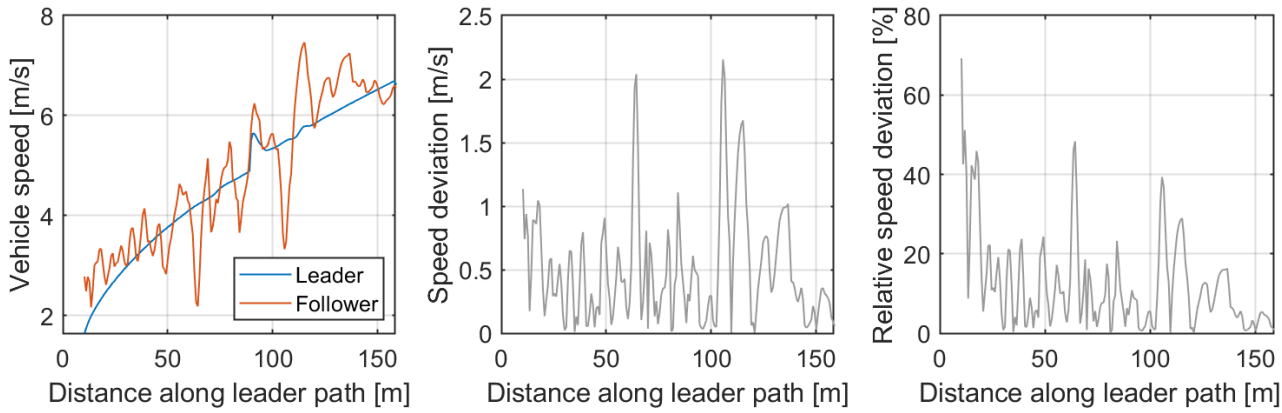
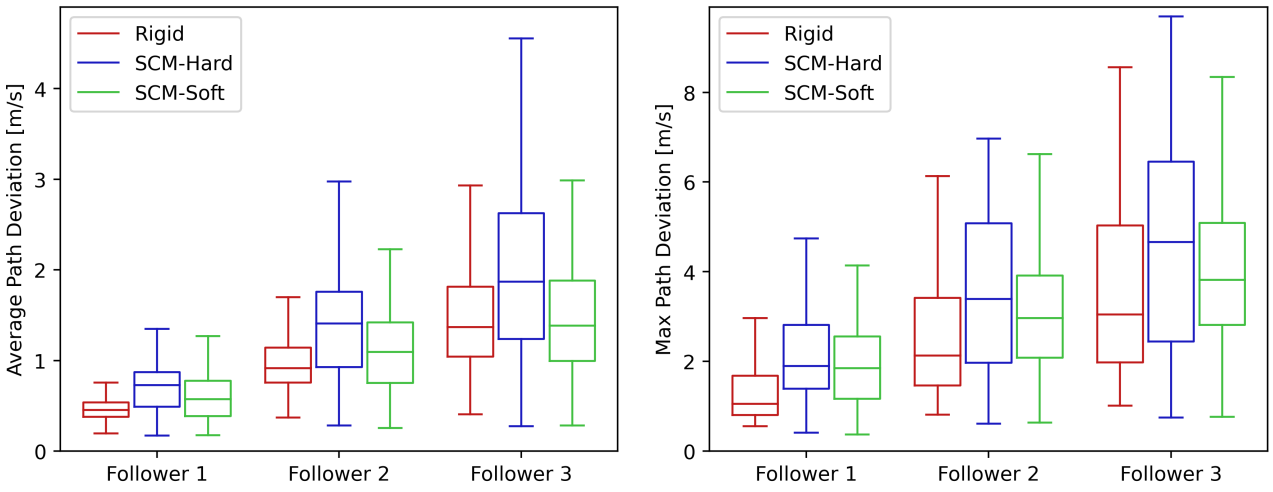


Figure 14: Deviation in speed between Leader and Follower 3. The vehicle speeds are evaluated at the same location along the leader's path. The speed deviation metrics for this particular scenario (rigid terrain) where $m_{as}^{avg} = 0.505$ m/s, $m_{as}^{max} = 2.154$ m/s and $m_{rs}^{avg} = 12.2\%$,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

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 19 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.

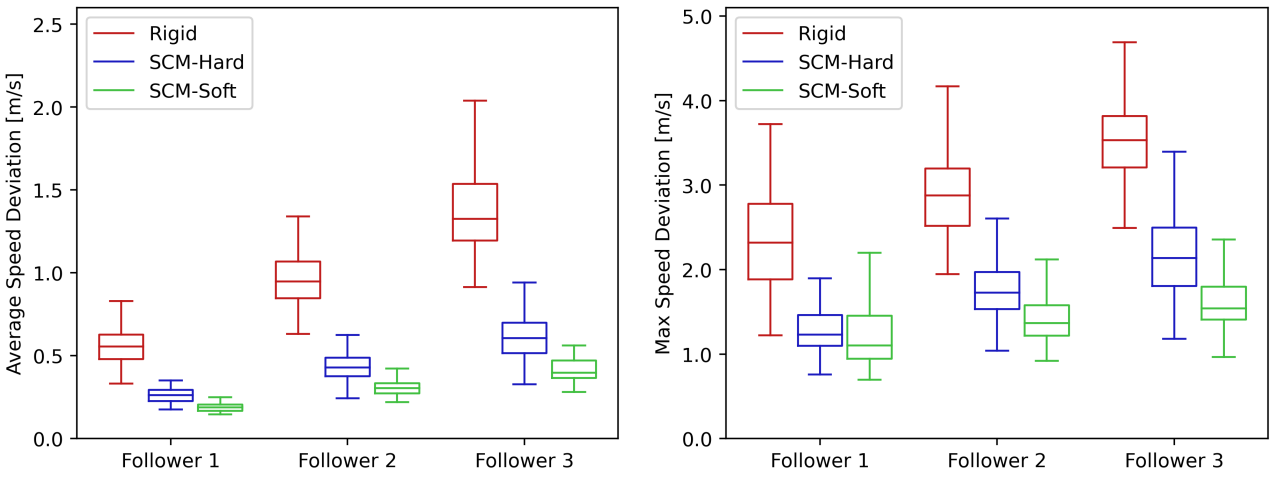
5 CONCLUSIONS. FUTURE WORK

This contribution discussed a Chrono-centric simulation platform designed to facilitate the design and testing of control policies for AAs operating in off-road conditions. The platform draws on a physics-based 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; 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 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 speed up the SCM implementation; augment the sensing simulation support; improve scalability; and use this simulation infrastructure to derive new control policies for off-road AA mobility.



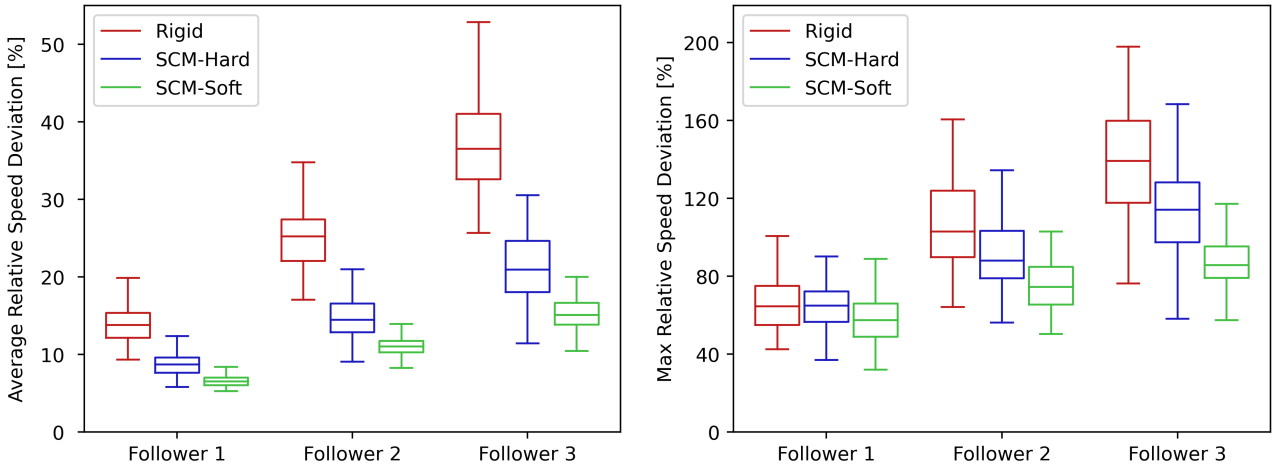
(a) (b)

Figure 15: Statistics of average and maximum follower path deviations.



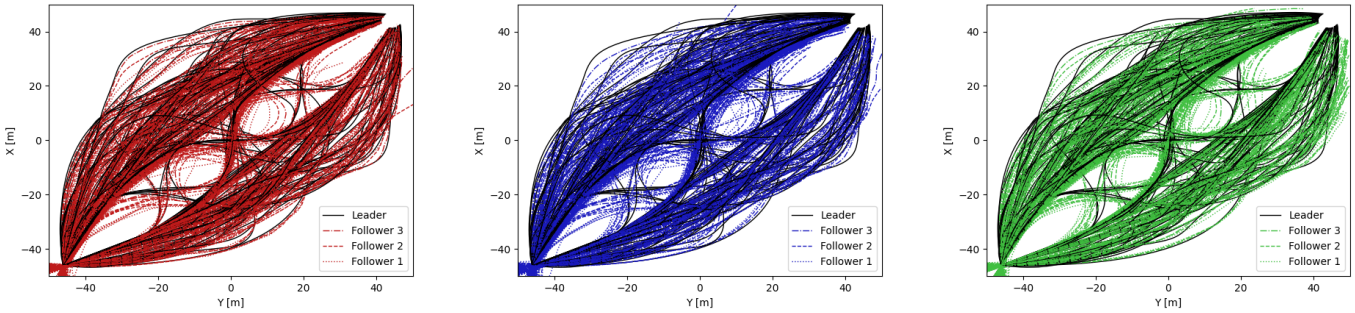
(a) (b)

Figure 16: Statistics of average and maximum follower absolute speed deviations.



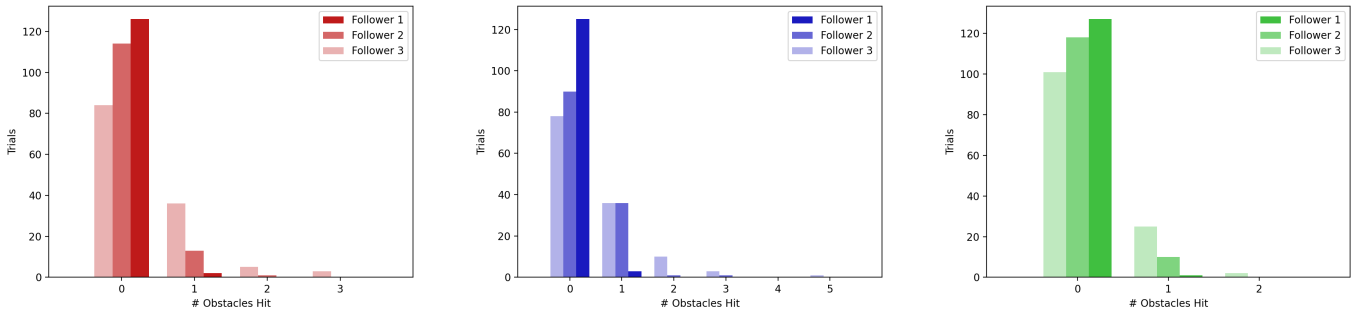
(a) (b)

Figure 17: Statistics of average and maximum follower relative speed deviations.



(a) (b) (c)

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



(a) (b) (c)

Figure 19: Number of obstacles hit by each vehicle on the three terrain types: (a) Rigid; (b) SCM-Hard; (c) SCM-Soft.

Acknowledgments

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 US 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] Project Chrono. GVSETS paper simulations. `https://uwmadison.box.com/s/glbpqxpomgyiomt2ydctpe35avrh44vd`, 2020.

[2] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[3] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer, 1995.

[4] Paramsothy Jayakumar, William Smith, Brant A Ross, Rohit Jategaonkar, and Krystian Konarzewski. Development of high fidelity mobility simulation of an autonomous vehicle in an off-road scenario using integrated sensor, controller, and multi-body dynamics. Technical report, Army Tank Automotive Research Development and Engineering Ceter, Warren-MI, 2011.

[5] Paramsothy Jayakumar, Abhinandan Jain, James Poplawski, Marco Quadrelli, Jonathan Cameron, and Joseph Raymond. Advanced mobility testbed for dynamic semi-autonomous unmanned ground vehicles. Technical report, Army Tank Automotive Research Development and Engineering Ceter, Warren-MI, 2015.

[6] David J Gorsich, Paramsothy Jayakumar, Michael P Cole, Cory M Crean, Abhinandan Jain, and Tulga Ersal. Evaluating mobility vs. latency in unmanned ground vehicles. *Journal of Terramechanics*, 80:11–19, 2018.

[7] Michael Cole, Cesar Lucas, Kumar B Kulkarni, Daniel Carruth, Christopher Hudson, and Paramsothy Jayakumar. Are M&S tools ready for assessing off-road mobility of autonomous vehicles? In *Ground Vehicle Systems Engineering and Technology Symposium*, pages 13–15, 2019.

[8] Philip Frederick, Mike Del Rose, David Pirozzo, Robert Kania, Bernard Theisen, and Stephanie Roth. Implementing robotic control algorithms in open source and government virtual environments. In *NDIA Ground Vehicle Systems Engineering and Technology Symposium*, 2014.

[9] Open-Source-Robotics-Foundation. A 3D multi-robot Simulator with Dynamics. `http://gazebosim.org/`. Accessed: 2015-03-09.

[10] Nathan P Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *IROS*, volume 4, pages 2149–2154. Citeseer, 2004.

[11] Russell L. Smith. Open Dynamics Engine. Available online at http://www.ode.org/, 2015.

[12] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[13] Jeongseok Lee, Michael X Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S Srinivasa, Mike Stilman, and C Karen Liu. DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3(22):500, 2018.

[14] Michael A Sherman, Ajay Seth, and Scott L Delp. Simbody: multibody dynamics for biomedical research. *Procedia IUTAM*, 2:241–261, 2011.

[15] Eric Rohmer, Surya PN Singh, and Marc Freese. V-REP: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013.

[16] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

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

[18] Newton Dynamics. Newton Dynamics: a cross-platform life-like physics simulation library. `http://newtondynamics.com/forum/newton.php`, 2020.

[19] A Jain, J Balaram, J Cameron, J Guineau, C Lim, M Pomerantz, and G Sohl. Recent developments in the ROAMS planetary rover simulation environment. In *2004 IEEE aerospace conference proceedings (IEEE Cat. No. 04TH8720)*, volume 2, pages 861–876. IEEE, 2004.

[20] Quantum Signals. Anvel:AE - ANVEL UGV simulator. `https://quantumsignalai.com/`. Accessed: 2020-05-13.

[21] Abhi Jain. DARTS dynamics algorithms for real-time simulation. `https://dartslab.jpl.nasa.gov/DARTS/index.php`.

[22] Phillip Durst, Christopher Goodin, Chris Cummins, Burhman Gates, Burney Mckinley, Taylor George, Mitchell Rohde, Matthew Toschlog, and Justin Crawford. A real-time, interactive simulation environment for unmanned ground vehicles: The autonomous navigation virtual environment laboratory (ANVEL). In *2012 Fifth International Conference on Information and Computing Science*, pages 7–10. IEEE, 2012.

[23] C. Goodin, T. George, C. Cummins, P. Durst, B. Gates, and G. McKinley. The virtual autonomous navigation environment: High fidelity simulations of sensor, environment, and terramechanics for robotics. In *Earth and Space*, pages 1441–1447, 2012.

[24] D. W. Carruth. Simulation for training and testing intelligent systems. In *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*, pages 101–106, 2018.

[25] Christopher Goodin, Matthew Doude, Christopher Hudson, and Daniel Carruth. Enabling off-road autonomous navigation-simulation of lidar in dense vegetation. *Electronics*, 7(9):154, 2018.

[26] Christopher Goodin, Daniel Carruth, Matthew Doude, and Christopher Hudson.

Predicting the influence of rain on lidar in adas. *Electronics*, 8(1):89, 2019.

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

[28] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[29] Unity3D. Main Website. https://unity3d.com/, 2016. Accessed: 2016-06-09.

[30] Eric Espié, Christophe Guionneau, Bernhard Wymann, and Christos Dimitrakakis. TORCS – The Open Racing Car Simulator. https://sourceforge.net/projects/torcs/, 2020.

[31] Yue Kang, Hang Yin, and Christian Berger. Test your self-driving algorithm: An overview of publicly available driving datasets and virtual testing environments. *IEEE Transactions on Intelligent Vehicles*, 4(2):171–185, 2019.

[32] Francisca Rosique, Pedro J Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3):648, 2019.

[33] Project Chrono. Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems. http://projectchrono.org, 2020. Accessed: 2020-03-03.

[34] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut. Chrono: An open source multi-physics dynamics engine. In T. Kozubek, editor, *High Performance Computing in Science and Engineering – Lecture Notes in Computer Science*, pages 19–49. Springer, 2016.

[35] Message Passage Interface Forum. MPI: A message-passing interface standard. Technical report, University of Tennessee, Knoxville, TN, USA, 1994. http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail&id=oai%3Ancstrlh%3Autk_cs%3Ancstrl.utk_cs%2F%2FUT-CS-94-230.

[36] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

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

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

[39] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. 01 2017.

[40] SAE. *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*, Jan 2014.

[41] Adit Joshi. Hardware-in-the-loop (hil) implementation and validation of sae level 2 automated vehicle with subsystem fault tolerant fallback performance for takeover scenarios. 1:13–32, 07 2018.

[42] A. Folkers, M. Rick, and C. Bskens. Controlling an autonomous vehicle with deep reinforcement learning. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2025–2031, 2019.

[43] Murat Dikmen and Catherine M. Burns. Autonomous driving in the real world: Experiences with tesla autopilot and summon. In *Proceedings of the 8th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI 16, page 225228, New York, NY, USA, 2016. Association for Computing Machinery.

[44] X. Qian, I. Navarro, A. de La Fortelle, and F. Moutarde. Motion planning for urban autonomous driving using bzier curves and mpc. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 826–833, 2016.

[45] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.

[46] X. Qian, F. Altch, P. Bender, C. Stiller, and A. de La Fortelle. Optimal trajectory planning for autonomous driving integrating logical constraints: An miqp perspective. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 205–210, 2016.

[47] Francesco Borrelli, Paolo Falcone, Tamás Keviczky, Jahan Asgari, and Davor Hrovat. Mpc-based approach to active steering for autonomous vehicle systems. 2005.

[48] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *ArXiv*, abs/1711.07300, 2017.

[49] W. Huang, Kunfeng Wang, Yisheng Lv, and FengHua Zhu. Autonomous vehicles testing methods review. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 163–168, 2016.

[50] Sampo Kuutti, Richard Bowden, Y. Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *ArXiv*, abs/1912.10773, 2019.

[51] Jens Kober, Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[52] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[53] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016.

[54] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[55] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[56] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT

Press, Cambridge, MA, USA, 1st edition, 1998.

[57] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *IN PROC. 19TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, pages 267–274, 2002.

[58] A. Tasora, D. Magnoni, D. Negrut, R. Serban, and P. Jayakumar. Deformable soil with adaptive level of detail for tracked and wheeled vehicles. *Intl. J. Veh. Performance*, 5(1):60–76, 2019.

[59] Project Chrono Development Team. Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems. `https://github.com/projectchrono/chrono`. Accessed: 2019-12-07.

[60] ECMA. The JSON data interchange format. Technical Report ECMA-404, ECMA International, 2013.

[61] A. M. Recuero, R. Serban, B. Peterson, H. Sugiyama, P. Jayakumar, and D. Negrut. A high-fidelity approach for vehicle mobility simulation: Nonlinear finite element tires operating on granular material. *Journal of Terramechanics*, 72:39 – 54, 2017.

[62] R. Serban, D. Negrut, A. M. Recuero, and P. Jayakumar. An integrated framework for high-performance, high-fidelity simulation of ground vehicle-tyre-terrain interaction. *Intl. J. Vehicle Performance*, 5(3):233–259, 2019.

[63] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. OptiX: A general purpose ray tracing engine. *ACM Transactions on Graphics*, August 2010.

[64] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[65] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[66] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, 2016.

[67] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[68] David M Beazley. Automated scientific software scripting with SWIG. *Future Generation Computer Systems*, 19(5):599–609, 2003.

[69] Project Chrono Development Team. PyChrono: A Python wrapper for the Chrono multi-physics library. `https://anaconda.org/projectchrono/pychrono`. Accessed: 2020-04-29.

[70] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016.

[71] 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.

[72] Google. Flatbuffers white paper: https://google.github.io/flatbuffers/flatbuffers_white_paper.html.

[73] O. Balling, M. McCullough, H. Hodges, R. Pulley, and P. Jayakumar. Tracked and wheeled vehicle benchmark – a demonstration of simulation maturity for next generation NATO Reference Mobility Model. In *Ground Vehicle Systems Engineering and Technology Symposium*, Novi, MI, 2018.

[74] R. Serban, R. Gerike, A. Elmquist, and D. Negrut. NG-NRMM Phase II Benchmarking: Chrono Wheeled-Vehicle Platform Simulation Results Summary,. Technical Report TR-2017-05: http://sbel.wisc.edu/documents/TR-2017-05.pdf, Simulation-Based Engineering Laboratory, University of Wisconsin-Madison, 2017.

[75] R. Serban, M. Taylor, D. Melanz, and D. Negrut. NG-NRMM Phase I Benchmarking: Chrono Tracked Vehicle Simulation Results Summary. Technical Report TR-2016-08: http://sbel.wisc.edu/documents/TR-2016-08.pdf, Simulation-Based Engineering Laboratory, University of Wisconsin-Madison, 2016.

[76] Adam Danz. Draw randomly centered circles of various sizes. https://www.mathworks.com/matlabcentral/fileexchange/70348-draw-randomly-centered-circles-of-var Accessed: 2020-06-17.

[77] Yarpiz. Path planning using PSO in MATLAB. https://www.mathworks.com/matlabcentral/fileexchange/53146-path-planning-using-pso-in-matlab. Accessed: 2020-06-17.