End-to-end learning for off-road terrain navigation using the Chrono open-source simulation platform

End-to-end learning for off-road terrain navigation using the Chrono open-source simulation platform

Simone Benatti[1], Aaron Young[1], Asher Elmquist[1], Jay Taves[1], Alessandro Tasora[2], Radu Serban[1*] and Dan Negrut[1]

[1*]Department of Mechanical Engineering, University of Wisconsin-Madison, Madison, 53706, WI, USA.
[2]Dipartimento di Ingegneria ed Architettura, Università di Parma, Parma, I-43100, Italy.

*Corresponding author(s). E-mail(s): serban@wisc.edu;
Contributing authors: benatti@wisc.edu; aryoung5@wisc.edu; amelmquist@wisc.edu; jtaves@wisc.edu; alessandro.tasora@unipr.it; negrut@wisc.edu;

Abstract

This contribution (***i***) describes an open source, physics-based simulation infrastructure that can be used to learn and test control policies in off-road navigation; and (***ii***) demonstrates the use of the simulation platform in an end-to-end learning exercise that relies on simulated sensor data fusion (camera, GPS, and IMU). For (***i***), the 0.5 million lines of open source code support vehicle dynamics (wheeled/tracked vehicles, rovers), deformable & non-deformable terrains, and virtual sensing. The library has a Python API for interfacing with existing Machine Learning frameworks. For **(*ii*)**, we use a Gator off-road vehicle to demonstrate how a policy learned on non-deformable terrain performs when used in hilly conditions while navigating around a course of randomly placed obstacles on deformable terrain. The hilly terrain covers an 80×80 m patch and the soil can be controlled by the user to assume various behavior, e.g., non-deformable, deformable hard (silt-like), deformable soft (snow-like), etc. To the best of our knowledge, there is no other open source, physics-based engine that can be used to simulate off-road mobility of autonomous agents operating on deformable terrains. The results reported herein can

be reproduced with models and data available in a public repository [1]. Animations associated with the tests run are available online [2].

Keywords: Simulation, Reinforcement Learning, Off-road Autonomous Vehicles, Deformable Terrain

1 Introduction

There are many applications in which controller design can benefit substantially from the use of simulation. Off-road navigation is arguably one such application. Indeed, it is difficult to test autonomous rovers, light robots, and heavy-duty vehicles in off-road conditions for several reasons. Many times, they do not exist at the time the controller is designed. If they do exist, it is costly (in terms of time and money) to take them out in the field and test candidate control policies. Even if this can be done in principle, it can be daunting since a multitude of testing scenarios may be necessary. For instance, unlike on-road driving, off-road navigation takes place in very unstructured environments, e.g, rocky terrain, mud, sand, obstacles (ditches, fallen trees, etc.), snow, etc. In addition to cost savings, shortening of design time, and ability to do exhaustive testing, there are two other advantages for simulation use in controller design: repeatability and safety. If a scenario is problematic, it can be recreated in simulation to gauge whether a better control policy addresses the issue at hand. As for safety, simulation eliminates hazards (both to the human and hardware) that are sometimes associated with physical testing. However, simulation is not a silver bullet. Many times, control policies designed in simulation fail to transfer to the real world owing to the so called simulation-to-reality gap [3]. It is also a source of frustration to anticipate when the gap is insurmountable, and when simulation is helpful. Enhancing the transferability attribute of simulation-learned control policies represents an area of active research [4–6]; however, this rich topic falls outside the scope of this contribution.

Herein, we describe a physics-based simulation platform and demonstrate it in conjunction with the task of producing an end-to-end policy for controlling autonomous vehicle (AV) navigation directly from raw sensory data. The AVs operate in hilly, off-road conditions with randomly placed obstacles (rocks) obstructing safe navigation. Training is based on a curriculum learning approach; the complexity of the environment is increased as the policy converges. The training is exclusively done with non-deformable terrain since these simulations run faster than real time [7]. The deformable terrain implementation used here is approximately 4-5$\times$ slower than the non-deformable terrain simulation counterpart. As such, learning on deformable terrain is expensive. The control policy derived is tested on deformable soils that have different textures and soil deformation attributes. The deformable soils are of two categories: deformable but hard (silt-like) and deformable but soft (snow-like). The end-to-end approach to navigation is certainly not new; see for instance [8, 9].

However, to the best of our knowledge, (*a*) this is the first example of off-road navigation (driving control + reaching a goal) with reinforcement learning; and (*b*) the simulation environment developed is the first *open-source*, physics-based platform that brings together tracked/wheeled vehicle dynamics, sensor, and terrain simulation.

Our goal is to demonstrate that *off-road* mobility of AVs can rely, in principle (see the sim-to-real caveat above), on simulation for the development of control policies and to report on tests that assess the effectiveness of these policies. Thus, in Section 2 we provide an overview of similar ongoing efforts in the simulation-in-robotics area. Section 3 provides an outline of the simulation environment developed by this group. Section 4 presents the end-to-end learning approach for off-road AV mobility. Simulation results are described in Section 5. Concluding remarks and directions of future work round up the contribution.

2 Related Work

This section provides a summary of the state of the art in simulation environments (see Subsection 2.1), and learning techniques for autonomous navigation (see Subsection 2.2). The discussion of simulation environments is restricted to those commonly used for training reinforcement learning algorithms.

2.1 Simulation Environments for Reinforcement Learning

Gazebo [10], one of the most broadly used simulators in robotics, has been used for reinforcement learning by leveraging the open-source nature and tight integration with the Robot Operating System (ROS). Gazebo exposes an environment that wraps multiple dynamics engines and sensors. However, it lacks specific support for vehicle modeling and deformable terrain for off-road scenarios.

Widely used for "in-doors robotics" reinforcement learning, MuJoCo [11] is a dynamics engine that supports Universal Robot Description Format (URDF)-based modeling. MuJoCo does not support vehicle dynamics. The sensing support is also limited due to the noise models applied to the sensors, most of which are restricted to be dynamics-based and interoceptive. An alternative to MuJoCo is PyBullet [12], which provides an interface to generate the specific sensor data desired by the user. The strengths of PyBullet are in rigid-body dynamics and ease of use, making it a convenient choice for Reinforcement Learning (RL) applications outside the realm of autonomous vehicles. For AV simulation, CARLA [13], AirSim [14], LGSVL [15], and Torcs [16] support vehicle simulation for training and testing of control algorithms. While these are vehicle-focused, they cannot perform off-road simulation and have limited sensor fidelity. Torcs, which builds off a racing game, provides limited support for sensing, allowing access to camera and simplified lidar, with dynamic information available directly from the physics engine. CARLA, LGSVL, and AirSim are designed for on-road applications

and support an array of sensors. The sensors include basic distortions and noise. Due to limited geometric fidelity and time-resolution of collision-based ray-casting, the sensor data is typically overly clean or has obvious discontinuities or modeling artifacts. None of these on-road simulation environments support complex off-road navigation. CARLA and AirSim build off Unreal Engine [17], while LGSVL is based on Unity [18]. Unreal and Unity are video gaming engines that provide the aforementioned simulators with high quality rendering and basic physics simulation tools. In turn, both Unreal and Unity internally use NVIDA's PhysX physics engine [19]. Vehicle dynamics are thus simulated very quickly, as PhysX was designed for speed. The drawback is that CARLA, LGSVL, AirSim all have the level of fidelity associated with video gaming. For instance, tire models are basic and display a level of fidelity good enough for gaming. Adding, for instance, new bushing elements to better capture suspension dynamics, or a more accurate terrain model is difficult and not available with the PhysX capabilities exposed by Unreal and Unity.

2.2 Learning techniques

The use of Deep Reinforcement Learning (DRL) has met with great success since its introduction [20]. In particular, DRL has found a niche in vision-based robotic manipulation tasks. Robots controlled by DRL-trained neural networks (NN) have been shown to solve complex tasks in unstructured environments with [21] or without [22] the use of imitation learning. End-to-end DRL approaches have also been successfully applied to on-road autonomous driving. One of the major challenges in this area is the gap between RGB images generated by simulators and real world camera images, which can cause autonomous driving policies trained in simulation to perform poorly in the real world. This has been addressed in various ways, e.g. using synthesized realistic images [23] or tools to generate images directly from real-world sampling [24]. Sensor fusion with DRL techniques has shown promising results in controlling small indoor robots with camera and lidar [25, 26]. RL in conjunction with imitation learning has been used in off-road driving to teach a vehicle to race quickly on a course [27]. However, to the best of our knowledge, there has been no demonstration of an end-to-end, off-road driving policy capable of reaching a target position while avoiding randomly placed obstacles on deformable soil and hilly terrain.

3 Chrono Simulation Environment

The physics-based simulator used in conjunction with this work is called Chrono. It is actively developed, is open source, and is released under a permissive BSD3 license for unfettered use, change, and distribution [28]. A full description of the simulation platform falls outside the scope of this document; for an overview, see [29]. Chrono provides support for multi-body dynamics (multi-core), nonlinear finite element analysis (multi-core),

fluid solid interaction (GPU), granular dynamics (multi-core/GPU), terramechanics (multi-core/GPU/MPI), sensing (GPU), and the simulation of large collections of AVs running in one joint scenario (MPI). The hardware support includes multi-core CPUs via OpenMP, GPU computing via CUDA, and distributed memory (clusters/supercomputers) via Message Passing Interface (MPI). The four Chrono components relevant herein are: Chrono::Engine, Chrono::Vehicle, terramechanics, and Chrono::Sensor. Chrono::Engine is the solver that advances the simulation in time. Chrono::Vehicle provides support for rapidly setting up and analyzing vehicles (tracked or wheeled) via a library of templates for vehicle subsystems [30]. The terramechanics support comes in three flavors: semi-empirical expeditious approaches [31], continuum representations [32], and discrete element method approaches (fully resolved granular terrain) [33].

Setting up vehicle models quickly is facilitated via Chrono::Vehicle, which provides vehicle subsystem templates such as tires, suspensions, steering mechanisms, drivelines, sprockets, track shoes, and powertrains. For instance, there are 15 types of suspensions supported, e.g. double wishbone, multi-link, MacPherson strut, leaf spring etc. There are several tire models available, e.g., Pacejka, Fiala, TMeasy. Chrono::Vehicle works in conjunction with a variety of terrain models, ranging from rigid, to semi-empirical Bekker-Wong type models, and on to complex physics-based solutions that draw on either a granular or a continuum representation of the soil.

Sensing support in Chrono is provided as an additional module that builds on top of Chrono::Engine to provide measurement data from within the simulation and virtual environment. Currently, there is support for RGB cameras, lidar, GPS, and IMU [34]. The purpose of the Chrono::Sensor module is to provide realistic data for training and testing autonomous controls. For GPS and IMU, ground truth data, queried from Chrono::Engine is augmented to introduce noise commonly found on accelerometers and gyroscopes [14] as well as GPS receivers. For camera and lidar, the visual environment is ray-traced using custom GPU kernels that model the acquisition process of the specific sensor. The ray-traced data is then augmented to introduce noise and distortion to model the true sensor output. All sensors are parameterized by their update frequencies, noise characteristics, and lag.

The Chrono lidar model augments ground-truth data with noise (based on the measurements of range, intensity, and angular precision) to produce the final point cloud. The lidar leverages ray-tracing to create a point cloud based on the visual scene. This, in combination with supersampling for beam divergence, allows Chrono::Sensor to generate high-fidelity point clouds of complex environments. The beam discretization model extends that proposed in [35] to allow a user-defined number of rays per lidar beam. By incorporating beam divergence, we can model multiple return modes and encountered objects. In addition to beam divergence, the ray-tracing method allows the temporal sampling of a scanning lidar to be based on modern motion blurring techniques resulting in realistic and continuous distortions that are not possible with

large time steps in video gaming collision detection systems employed by other learning environments, e.g. [13, 14].

The implemented camera simulator introduces lens and image sensor models to improve the realism of the data. The camera is parameterized based on the frequency, resolution, field of view, exposure time, and lag. Based on the exposure time, motion blur that accounts for object and camera movement is introduced. The camera lens model draws on work from [36] to allow for wide angle lenses. The noise model is based on modified version of the EMVA standard [37], which introduces intensity-dependent noise based on the image sensor characteristics. Additional components of the image signal processor (ISP) are in development since the ISP introduces additional sensing artifacts such as compression, demosaicking, and color correction.

The Python API of Chrono, known as *PyChrono*, provides access to the vast majority of Chrono API from Python, including Chrono::Vehicle and Chrono::Sensor. This allows for a simulation to be directly interfaced to the Python API of popular ML frameworks. By using the SWIG wrapper [38] to directly interface with the C++ binaries, minimal overhead is introduced when running a simulation from Python. As an example, large data from sensor simulations (such as RGB images or lidar) are cast to NumPy arrays without instantiating new memory by means of SWIG typemaps.

4 End-to-end learning approach

The control policy employed in this work is end-to-end: the NN takes as inputs raw sensor data and directly outputs the control values for steering and throttle. The policy is trained from scratch. The objective for the navigation algorithm is to control a John Deere Gator to reach a target destination given by GPS coordinates. The algorithm uses a GPS and IMU that provide the NN with the current vehicle location and orientation. The vehicle is also equipped with a down-sampled RGB camera, which the network leverages in order to avoid obstacles; i.e., rocks of various shapes, sizes, and textures. Since the impact of the sensor models is outside the scope of this paper, all sensor data is idealized with no noise.

For training, the environment consists of a 120×120 m patch of terrain on which obstacles are randomly placed. The vehicle's initial position is picked randomly in a 80 m diameter circle while the goal is placed on the opposite side of the same circle; in polar coordinates, given α the angle of the vehicle initial position, the angle of the goal will be $\alpha + \beta$ with β randomly picked in $[\frac{\pi}{2}, \frac{3\pi}{2}]$. The vehicle must navigate to within 10 m of the destination and the reward is proportional to the vehicle's approach speed. The episode is terminated with a reward penalty if the vehicle hits an obstacle, goes outside the terrain boundary, or the timeout is reached, while it is terminated with a reward bonus if it reaches the goal.

The observation consists of a two-element tuple: an 80×45 pixel RGB image and a five-element array containing the components (x,y) of the distance from

the goal (in the vehicle frame, based on GPS measurements), the vehicle orientation (compass angle), the heading with respect to the goal, and the vehicle speed. Based on these inputs, the policy controls the steering (-1 to 1) and the throttle/brake value (-1 to 1, where a negative value implies braking).

To train the NN, we adopted the constrained version of the Proximal Policy Optimization (PPO) algorithm [39], using two separated NN for the actor and the critic. PPO is known as one of the best performing DRL algorithms for continuous control [6]. The NN model inputs come from the GPS, IMU, and RGB camera. Through PyTorch [40], the NN model was implemented as follows. A five-element array was fed to a fully connected layer into 10 neurons, while the RGB image was processed by a CNN as in [20] through 3 Convolutional Layers of kernel size 8×8, 4×4 and 3×3 and stride 4, 2 and 1, respectively (no padding), then flattened into 768 features which were processed by a fully-connected layer into 10 neurons. The output of the CNN and the single fully-connected hidden layer were concatenated and then processed through three fully-connected hidden layers, as shown in Fig. 1. All layers used rectified linear unit (ReLU) activation function.

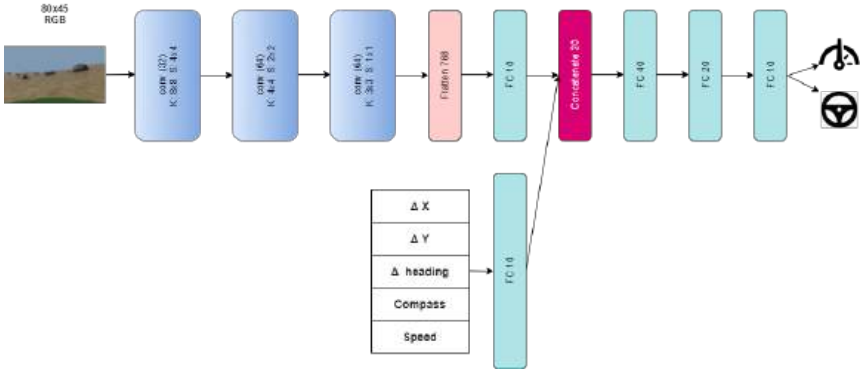


Fig. 1: Actor neural network architecture.

Given the GPS coordinates of the vehicle and the goal, along with the orientation of the vehicle from the IMU, it is straightforward to evaluate the distance between the vehicle and the objective. This being said, there are several ways to pass this information to the NN as an input: directly feeding the GPS coordinates of the vehicle and of the objective, the coordinate difference, the relative distance in a frame oriented along the cardinal points or the distance in the local frame of the vehicle. The last has proven to be the most effective, even though in principle the ML algorithm should be able to infer the correlation. The direct global coordinate input approach proved to be inefficient, as shown in Fig. 2. It can be seen that, in terms of convergence, simply rotating the position of the goal with respect to the vehicle in the vehicle reference frame (called *local*) dramatically improved the policies performance.

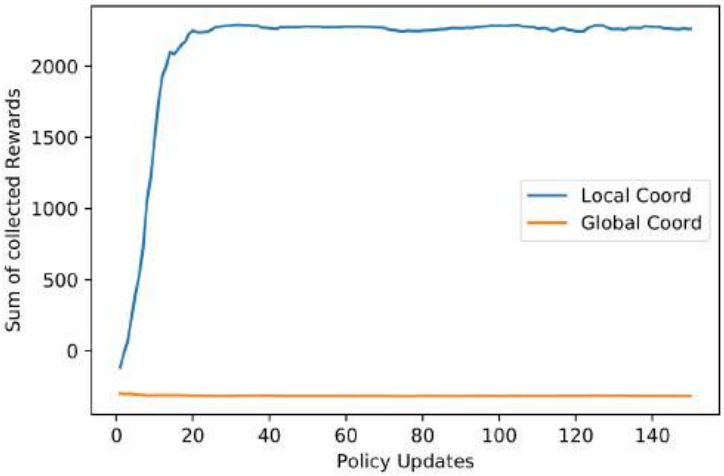


Fig. 2: On flat terrain *without* obstacles, the algorithm converges immediately when feeding the position of the goal rotated in the vehicle reference frame. When the position is given in the global frame, training does not converge.

We adopted a curriculum learning approach [41], progressively increasing the complexity of the task, as shown in Fig. 3. The first part of the training was performed on flat terrain with a random number of obstacles (from 0 to 30). Once convergence was reached after approximately 200 policy updates, the obstacle number was fixed at 30. After a visible drop off, convergence was reached again quickly. Then, after 376 policy updates, the flat terrain was replaced with hilly terrain (while keeping the same number of obstacles). The agent initially struggled and many updates were necessary in order to converge again. In the third and last stage of the training, the obstacle count was increased to 50 and the terrain texture was randomized. Curriculum learning was deemed necessary since convergence could not be directly reached from scratch on hilly terrain. Investigating multiple training approaches, we found that: (*i*) Irregularities in terrain height caused policies trained exclusively on flat terrain to perform poorly; (*ii*) This problem can by solved by undergoing further training on hilly terrain; and, (*iii*) The curriculum learning approach (see Fig. 3) was instrumental in eventually handling the complex tasks, namely hills with many random obstacles.

Training relied on the Adam algorithm [42] with a learning rate of 10^{-4}. The training set at each update included 6000 tuples (timesteps), fed by 1000 element mini-batches to the optimizer, which performed eight epochs per update. Since Chrono is compatible with OpenAI gym [43] environments, the dataset was collected by running six parallel episodes leveraging OpenAI baselines [44] multiprocessing tools.

To complete the first phase, pertaining to navigation on a flat terrain with sparse obstacles, the algorithm took 200 policy updates, each of which required 6000 steps for collecting samples over 6 parallel simulation environments. Considering that the frame rate was 10 fps (even though the physics was carried out at 500 Hz) it comes out that it would take more than 33 real world hours for the policy to converge (considering the use of 6 parallel environments, 200 updates, 1000 steps per update, and a 0.1,s time step). Given the massive amount of samples needed, the computational overhead introduced by simulating terrain deformation strongly increases the training time. In addition, preliminary experiments did not always show a consistent improvement when adding terrain deformation during the training: other factors, such as terrain texture randomization, play a much important role. The feedback nature of the policy can correct for deviation in response due to a different terrain, and the agent has no means (in terms of perception) to fully estimate the trafficability of the path ahead. This being said, thanks to recent improvements in our deformable terrain simulation performance, we plan to investigate this further in the future, possibly combined with additional sensing information (e.g., wheel slip).

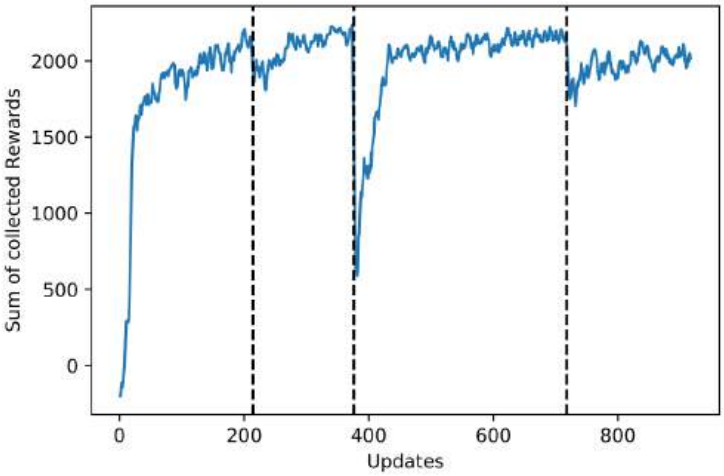


Fig. 3: Reward progression. Vertical lines represent the changes introduced to make the environment more challenging. In order of occurrence, the dotted lines mark: fix obstacle count at 30; change from flat to hilly terrain; and increase of obstacle count to 50.

5 Simulation Experiments

To demonstrate and analyze the capabilities of the control policy, we used a model of the John Deere Gator utility vehicle driving on an 80×80 m patch of terrain. The reduced scale of the test environment allowed for rapid evaluation and inclusion of tests using deformable terrain based on the Soil Contact Model (SCM) [31].

The Gator vehicle model is constructed by instantiating and combining the appropriate subsystem templates from the Chrono::Vehicle library. In particular, it uses a single wishbone suspension in the front, connected to a rack-and-pinion steering mechanism, and a rigid rear suspension, connected to the rear-wheel driveline consisting of a conical gear and a differential. Power is provided by an electrical motor (modeled with a simple linear torque-speed curve). The Gator driveline contains no torque converter nor transmission box and provides fixed gear rations for forward and reverse operation. The resulting Chrono model consists of 10 bodies, 11 kinematic joints, 2 translational spring-damper force elements (for the front suspension), and 4 1D shaft elements (used in modeling the driveline) and has 14 degrees of freedom. The total vehicle mass is 906.2 kg (note that this is a model of a Gator vehicle instrumented for autonomous driving and as such includes the mass of an instrumentation tower rigidly attached to the chassis).

Multibody systems in Chrono are modeled using a body-coordinate formulation based on Newton-Euler equations, with orientation parameterized using unit quaternions. For more details, as well as additional information on the solver options available in Chrono, see [29, 45].

The deformable terrain model used in this study is SCM which relies on the semi-empirical Bekker [46, 47] and Janosi-Hanamoto formulae [48]. The two sets of deformable soil parameters used in this study are provided below (see [31] for the SCM formulation and model parameters used herein).

		SCM hard	SCM soft
K_ϕ	[N/m^{n+2}]	$5.3 \cdot 10^3$	$2.0 \cdot 10^5$
K_c	[N/m^{n+1}]	$1.0 \cdot 10^3$	0
n		0.793	1.1
c	[Pa]	$1.3 \cdot 10^3$	0
ψ	[deg]	31	30
K	[m]	0.012	0.010
k_e	[Pa/m]	$4.0 \cdot 10^8$	$4.0 \cdot 10^7$
r	[Pa/m·s]	$3.0 \cdot 10^4$	$3.0 \cdot 10^4$

For all tests, the vehicle is placed at world location (-35,35); to be deemed successful, it must navigate safely to within a 10 m radius of world location (35,-35). This setup was a matter of convenience and is not a limitation of the policy. Since training used a larger patch of 120×120 m, it is important to note that parameters such as number of obstacles and height of terrain cannot be compared directly with testing; this was done on purpose. For the terrain patch, an equivalent number of obstacles to the 50 used in training

is approximately 22. Additionally, the maximum height difference of 10 m in training is equivalent[1] to a height of approximately 7.25 m on the testing terrain. Furthermore, in testing, to prevent the vehicle from successfully navigating purely along the diagonal straight toward the destination, five obstacles were placed randomly near the diagonal to force non-trivial trajectories. Overall, the tests were conducted with a higher-complexity environment than that used in training. They were designed to probe the robustness of the algorithm and understand to what extent the policy could be used on never-before-seen terrain. We looked at (*i*) increased levels of hilliness; (*ii*) increased number of obstacles; and (*iii*) alterations of the soil including non-deformable, hard deformable (silt-like), and soft deformable (snow-like).

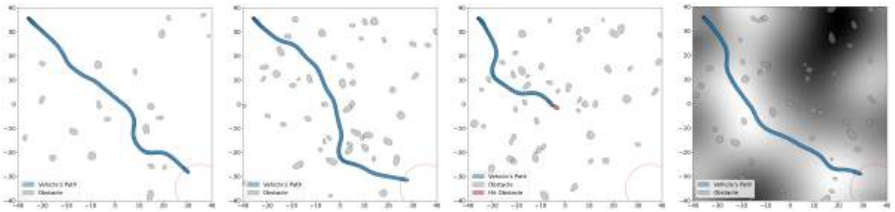
Figure 4 shows a snapshot of one scenario: the soft deformable terrain used for testing robustness; a capture of the image from the camera sensor used by the NN; and the position of the vehicle along its current trajectory with the obstacles and height map overlaid for context (in the height map, black indicates valleys and white peaks).

Fig. 4: Snapshot of a scenario used for testing. Left: third-person perspective of the vehicle; center: image from the camera's view at the resolution used in training 80×45 pixels; right: the current progress of the vehicle amount the hills and obstacles corresponding to the images on left.

Figure 5 shows an example set of paths navigated by the Gator in the test environment. In Fig. 5a, the vehicle avoids a sparse environment, limited to 20 obstacles. By increasing the obstacles to 50, as shown in Fig. 5b, the complexity of the test environment forced the policy into a correspondingly complex path. An example failure is illustrated in Fig. 5c where the vehicle did not reach the destination owing to a collision. Any scenario where the vehicle collides with an obstacle was deemed a failure, regardless of how directly the vehicle collided with the obstacle. The last example, shown in Fig. 5d, demonstrates the vehicle navigating a hilly terrain, based on a programmatically-generated random height map.

In an attempt to assess the practicality of the vehicle's chosen path, a Particle Swarm Optimization (PSO) algorithm [49] with global knowledge of the environment was used to generate reference trajectories in the environment; these were used in post-processing only, for comparison purposes. Figure 6a

[1] By "equivalent" we mean the number-of-obstacles/surface ratio is the same; for terrain height difference, "equivalent" means that the maximum slope is the same.



(a) (b) (c) (d)

Fig. 5: Example test scenarios: (a) 20 obstacles, (b) 50 obstacles, (c) failure with 50 obstacles, (d) hilly terrain with 50 obstacles.

shows a comparison between the trained vehicle's path and a trajectory generated using the PSO path planner. In this example, 40 obstacles were present on a rigid flat terrain.

To quantify the end-to-end learned navigation, the primary metric is success rate. This is simply a measure of the vehicle's ability to reach the destination without colliding with any obstacles. For this metric, any collision, regardless of directness of collision, is considered a failure and the simulation is terminated. In addition to the primary metric, the length of the path is analyzed relative to the PSO path. Since the PSO path would generally be able to find a shorter path than the trained algorithm, which only has local information, the comparison is used to show that the path taken by the trained algorithm is reasonable, and not simply a path that avoids obstacles yet produces bizarre trajectories.

When testing the effect of hilliness on navigation, the maximum height difference of the random height map was increased from 0 m (flat) to 12 m in increments of 2 m. At each level, 200 simulations on random height maps generated using simple noise were performed to gauge the success rate of the algorithm. In all experiments, the terrain used one of the textures used in training. The results are shown in Fig. 7. As expected, the trained network's ability to safely navigate the environment decreases with increased hilliness. Additionally, as the number of obstacles is increased, the task becomes more difficult; we note that tests conducted with 20 obstacles were the most similar to the training environment.

To investigate the policy robustness and assess the relative importance of including different mechanisms of terrain variability in the training set, we estimated next the success rate for different terrain topographies and soil properties, with varying numbers of obstacles, and using both terrain textures that were included in the training test as well as entirely new textures. The results of Fig. 8a correspond to tests in which different textures were used for each soil type; however, the texture used for 'Rigid' terrain was one of those included in the training set, while those used for the two SCM deformable cases were not. In contrast, the results in Fig. 8b were generated using a single

texture (from the set of textures included in training) across all terrain and soil types.

For a proper evaluation of these results, we note that (*i*) the deformable 'SCM Hard' soil properties model a relatively rigid terrain (that is, close in behavior to the 'Rigid' type) and (*ii*) the terrain textures for 'Rigid' terrain in Figs. 8a and 8b are different, although both coming from the set of textures used in training. The trends shown here confirm the expected outcomes:

- First, they confirm the strong influence of terrain texture (for autonomous navigation relying on camera sensors) and the fact that terrain texture is much more important on uneven terrain where it represents a larger portion of the background for each obstacle. This is best illustrated by the differences in success rates for the 'SCM Hard Hilly' case in the two sets of experiments. This is not observed for the 'Rigid Hilly' terrain since both sets of experiments used textures included in the training process.
- As before, increased obstacle density results in lower success rate.
- On very soft soils and uneven terrain, the ensuing effects on vehicle mobility (side slip while steering, increased longitudinal slip on sloped terrain, etc) end up being the main performance limitation. This indicates that the current policy, derived through training on rigid soil only, is not suitable for this type of scenario. The goal of the current study was to assess the extent to which such a policy is applicable and identify this limit.

The second metric, which compares the length of the chosen path to the length of a path generated by the PSO, is discussed for a single environment setup. This is shown in Fig. 9a for flat, rigid terrain with 50 obstacles, which is more than double the obstacle density used in training. The length difference is computed as $length(PSOpath) - length(NNpath)$. These results show that the mode of the distribution is around -5 m, meaning the NN path is most often 5 m longer than the PSO path. While a direct comparison is not feasible since the PSO takes into account global knowledge and does not make any claims about optimality, the path taken by the NN appears to be moderately close in length to a global planner. This means that the policy is appropriately weighting the directness of the path as expected. Only successful paths were included in the calculation of this metric.

To further understand and analyze the success rate of the algorithm, the full set of collisions were evaluated based on the directness of collision. The directness of collision was computed by measuring the overlap of the projection of the vehicle and obstacle onto a plane perpendicular to the vehicle heading. This metric quantifies scraping collisions near 0% and direct collisions near 100%. This percentage can be interpreted as the percent of the frontal area of the vehicle that collided with the obstacle. While this cannot directly assign severity to the collision, it can hint at the type of collisions experienced by the policy. The distribution of results in Fig. 9b show that while the mode is near 100%, there is also a significant portion of the collisions that have low directness.

6 Conclusion and Future Work

In this paper, we briefly describe the Chrono infrastructure, including support for vehicle dynamics, sensor simulation, and terramechanics, to allow comprehensive off-road AV mobility studies and focus the study on end-to-end learning as enabled by the Chrono environment which anchors both the learning and testing phases. The end-to-end policy is used in flat and hilly landscapes, with deformable terrain that can be hard (silt-like) or soft (snow-like). We noted the following: policies learned on flat terrain are insufficient for navigating hilly scenarios; policies learned on rigid terrain transfer quite well to deformable terrain when the terrain is flat; the hillier the landscape, the harder it is to navigate it (Fig. 8); the more obstacles are randomly placed on the course, the less likely it is for the policy to see the vehicle through; the control policy led to trip trajectories that came rather close to those generated with PSO, a third party trajectory planning tool (Fig. 9a); there is a sizable number of head-on collisions that point to room for improvement in the derived policy (Fig. 9b). Note that the testing conditions, in terms of average number of obstacles per unit area, were more harsh than the learning conditions.

Looking ahead, we plan to pursue several research thrusts and simulation platform development avenues. One direction is to investigate more complex control stacks that would combine the end-to-end with more traditional strategies such as model predictive control. The current work should be expanded to understand how tracked vehicles perform under similar conditions given that their traction and turning radius differ significantly from their wheel counterparts. A recent reformulation of our SCM deformable soil algorithm resulted in speedups of more than 50×, opening the door to real-time simulation on deformable soil. This will allow for training on deformable soil which we expect to result in more robust autonomous navigation. Not analyzed in this contribution is the steering control input to the vehicle, which can often be noisy based on the output of the NN. Finally, we plan to investigate approaches that enhance the chance of simulation-derived policies transferring effectively to the real world.

References

[1] UW-Madison Simulation Based Engineering Laboratory: Supporting models, scripts, data. https://go.wisc.edu/arflqq (2021)

[2] UW-Madison Simulation Based Engineering Laboratory: Supporting simulations. https://go.wisc.edu/256xb9 (2021)

[3] Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: The use of simulation in evolutionary robotics. In: European Conference on Artificial Life, pp. 704–720 (1995). Springer

[4] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.:

Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 23–30 (2017). IEEE

[5] Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., Fox, D.: Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In: 2019 International Conference on Robotics and Automation (ICRA), pp. 8973–8979 (2019). IEEE

[6] 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., Zaremb, W.: Learning dexterous in-hand manipulation. The International Journal of Robotics Research **39**(1), 3–20 (2020). https://doi.org/10.1177/0278364919887447

[7] Negrut, D., Serban, R., Elmquist, A., Taves, J., Young, A., Tasora, A., Benatti, S.: Enabling Artificial Intelligence studies in off-road mobility through physics-based simulation of multi-agent scenarios. In: NDIA Ground Vehicle Systems Engineering and Technology Symposium (2020)

[8] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)

[9] Amini, A., Rosman, G., Karaman, S., Rus, D.: Variational End-to-End Navigation and Localization (2018). http://arxiv.org/abs/1811.1011

[10] Koenig, N.P., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 4, pp. 2149–2154 (2004). Citeseer

[11] Todorov, E., Erez, T., Tassa, Y.: MuJoCo: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5026–5033 (2012). IEEE

[12] Matas, J., James, S., Davison, A.J.: Sim-to-Real Reinforcement Learning for Deformable Object Manipulation (2018). https://arxiv.org/abs/1806.07851

[13] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An open urban driving simulator. In: Proceedings of the 1st Annual Conference on Robot Learning, pp. 1–16 (2017)

[14] Shah, S., Dey, D., Lovett, C., Kapoor, A.: Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: Field and Service Robotics, pp. 621–635 (2018). Springer

[15] Rong, G., Shin, B.H., Tabatabaee, H., Lu, Q., Lemke, S., Možeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., Agafonov, E., Kim, T.H., Sterner, E., Ushiroda, K., Reyes, M., Zelenkovsky, D., Kim, S.: LGSVL simulator: A high fidelity simulator for autonomous driving. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), pp. 1–6 (2020). IEEE

[16] Espié, E., Guionneau, C., Wymann, B., Dimitrakakis, C.: TORCS – The Open Racing Car Simulator. https://sourceforge.net/projects/torcs/ (2020)

[17] Epic Games: Unreal Engine. https://www.unrealengine.com. Accessed: 2021-11-23 (2020)

[18] Unity3D: Main Website. https://unity3d.com/. Accessed: 2021-11-23 (2016)

[19] NVIDIA: PhysX simulation engine. Available online at http://developer.nvidia.com/object/physx.html (2019)

[20] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR **abs/1312.5602** (2013) https://arxiv.org/abs/1312.5602

[21] Zhu, Y., Wang, Z., Merel, J., Rusu, A., Erez, T., Cabi, S., Tunyasuvunakool, S., Kram??r, J., Hadsell, R., de Freitas, N., Heess, N.: Reinforcement and Imitation Learning for Diverse Visuomotor Skills (2018)

[22] Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. CoRR **abs/1504.00702** (2015) https://arxiv.org/abs/1504.00702

[23] You, Y., Pan, X., Wang, Z., Lu, C.: Virtual to real reinforcement learning for autonomous driving. CoRR **abs/1704.03952** (2017) https://arxiv.org/abs/1704.03952

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

[25] Bohez, S., Verbelen, T., Coninck, E.D., Vankeirsbilck, B., Simoens, P., Dhoedt, B.: Sensor Fusion for Robot Control through Deep Reinforcement Learning (2017). http://arxiv.org/abs/1703.04550

[26] Patel, N., Choromańska, A., Krishnamurthy, P., Khorrami, F.: Sensor

modality fusion with CNNs for UGV autonomous driving in indoor environments. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1531–1536 (2017)

[27] Pan, Y., Cheng, C., Saigol, K., Lee, K., Yan, X., Theodorou, E.A., Boots, B.: Agile Autonomous Driving Using End-to-End Deep Imitation Learning (2017). http://arxiv.org/abs/1709.07174

[28] Project Chrono Development Team: Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems. https://github.com/projectchrono/chrono. Accessed: 2022-01-10

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

[30] Serban, R., Taylor, M., Negrut, D., Tasora, A.: Chrono::Vehicle template-based ground vehicle modeling and simulation. Intl. J. Veh. Performance **5**(1), 18–39 (2019)

[31] Tasora, A., Mangoni, D., Negrut, D., Serban, R., Jayakumar, P.: Deformable soil with adaptive level of detail for tracked and wheeled vehicles. International Journal of Vehicle Performance **5**(1), 60–76 (2019)

[32] Hu, W., Rakhsha, M., Yang, L., Kamrin, K., Negrut, D.: Modeling granular material dynamics and its two-way coupling with moving solid bodies using a continuum representation and the SPH method. Computer Methods in Applied Mechanics and Engineering **385**, 114022 (2021). https://doi.org/10.1016/j.cma.2021.114022

[33] Kelly, C., Olsen, N., Negrut, D.: Billion degree of freedom granular dynamics simulation on commodity hardware via heterogeneous data-type representation. Multibody System Dynamics **50**, 355–379 (2020)

[34] Elmquist, A., Serban, R., Negrut, D.: A sensor simulation framework for training and testing robots and autonomous vehicles. ASME Journal of Autonomous Vehicles and Systems **1**(2), 021001 (2021)

[35] Goodin, C., Doude, M., Hudson, C., Carruth, D.: Enabling off-road autonomous navigation-simulation of lidar in dense vegetation. Electronics **7**(9), 154 (2018)

[36] Tang, Z., von Gioi, R.G., Monasse, P., Morel, J.-M.: A precision analysis of camera distortion models. IEEE Transactions on Image Processing **26**(6), 2694–2704 (2017)

[37] EMVA Standard: 1288, standard for characterization of image sensors and cameras. European Machine Vision Association **3** (2010)

[38] Beazley, D.M.: SWIG: an easy to use tool for integrating scripting languages with C and C++. In: Proc. 4th Conf. on USENIX Tcl/Tk Workshop, vol. 4. USA, p. 15 (1996)

[39] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017) https://arxiv.org/abs/1707.06347

[40] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NIPS 2017 Workshop Autodiff (2017)

[41] Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning. ICML '09, pp. 41–48. Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1553374.1553380. https://doi.org/10.1145/1553374.1553380

[42] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2017) https://arxiv.org/abs/1412.6980 [cs.LG]

[43] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. CoRR **abs/1606.01540** (2016) https://arxiv.org/abs/1606.01540

[44] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., Zhokhov, P.: OpenAI Baselines. https://github.com/openai/baselines

[45] Project Chrono: Chrono Documentation and API Reference. http://api.projectchrono.org/. Accessed: 2021-11-24

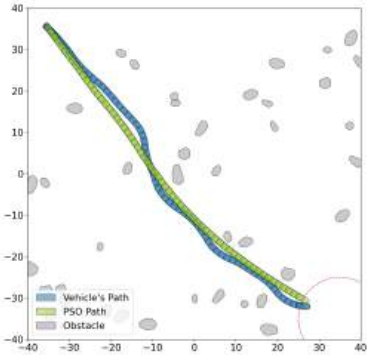
[46] Bekker, M.G.: Introduction to Terrain-vehicle Systems. University of Michigan Press, Ann Arbor, MI (1969)

[47] Wong, J.Y.: Theory of Ground Vehicles, 4th edn. John Wiley & Sons, New York, N.Y. (2008)

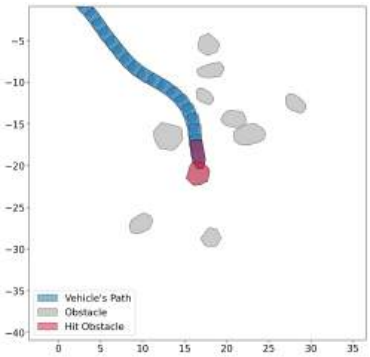
[48] Janosi, Z., Hanamoto, B.: The analytical determination of drawbar pull as a function of slip for tracked vehicles in deformable soils. In: Proc of the 1st Int Conf Mech Soil–vehicle Systems. Turin, Italy (1961)

[49] Yarpiz: Path Planning using PSO in MATLAB. https://www.mathworks.com/matlabcentral/fileexchange/

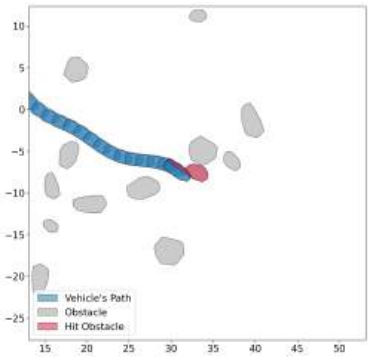
53146-path-planning-using-pso-in-matlab. Accessed: 2020-06-17



(a) Example comparison between the Gator's path and PSO path.



(b) Example collision where the directness of the impacts was 100%.



(c) Example collision where the directness of the impacts was 34%.

Fig. 6: Example paths showing comparison with global path planner and two levels of collision directness.

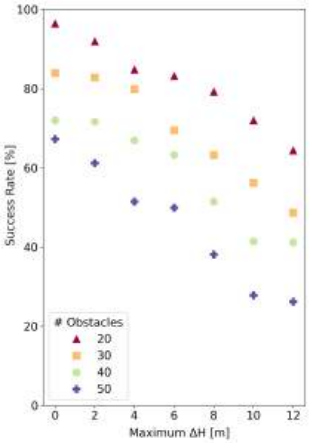
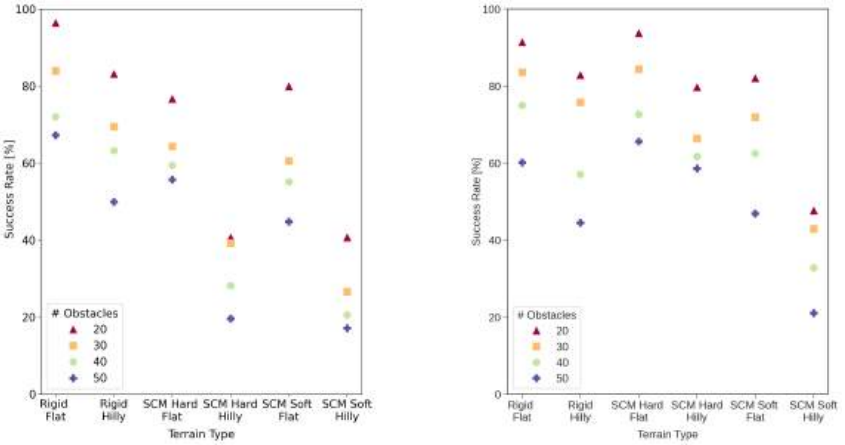


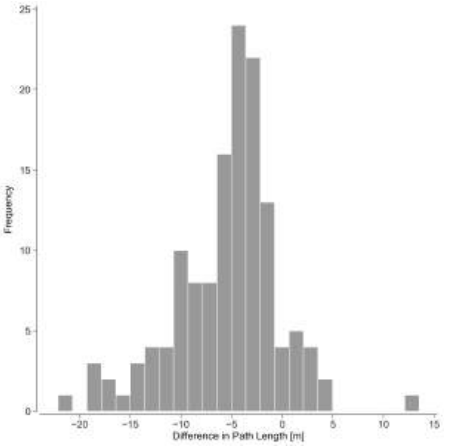
Fig. 7: Success rate as function of hilliness and obstacle density on rigid terrain.



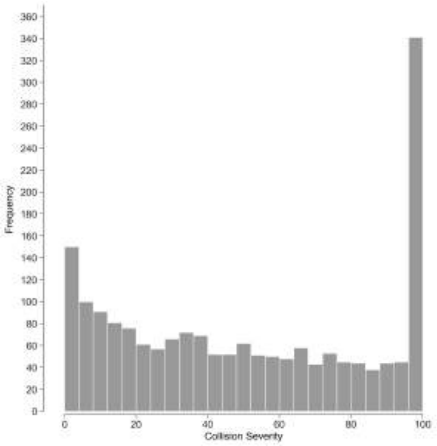
(a) Using new (different) textures for each scenario.

(b) Using same texture (from training set) for all scenarios.

Fig. 8: Success rate as measure of policy robustness.



(a) Difference in path length between the PSO trajectory and vehicle path for flat rigid terrain with 50 obstacles.



(b) Impact Directness on flat rigid terrain for all obstacle configurations.

Fig. 9: Additional metrics analyzing path length and collision directness.