33rd European Modeling & Simulation Symposium
18th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0029 ISBN 978-88-85741-57-7 © 2021 The Authors.
doi: 10.46354/i3m.2021.emss.015

Comparing Physics Effects through Reinforcement Learning in the ARORA Simulator

Troyle Thomas[1], Armando Fandango[1], Dean Reed[1,*], Clive Hoayun[1], Jonathan Hurter[1], Alexander Gutierrez[1], and Keith Brawner[2]

[1]Institute for Simulation and Training, 3100 Technology Pkwy, Orlando, FL 32826, USA
[2]U.S. Army Combat Capabilities Development Command Soldier Center (CCDC SC), 12423 Research Pkwy, Orlando, FL 32826, USA

*Corresponding author. Email address: dean.reed@ucf.edu

Abstract

By testing various physics levels for training autonomous-vehicle navigation using a deep deterministic policy gradient algorithm, the present study fills a lack of research on the impact of physics levels for vehicle behaviour, specifically for reinforcement-learning algorithms. Measures from a PointGoal Navigation task were investigated: simulator run-time, training steps, and agent effectiveness through the Success weighted by (normalised inverse) Path Length (SPL) measure. Training and testing occurred in the novel simulator ARORA, or A Realistic Open environment for Rapid Agent training. The goal of ARORA is to provide a high-fidelity, open-source platform for simulation, using physics-based movement, vehicle modelling, and a continuous action space within a large-scale geospecific city environment. Using four physics levels, or models, to create four different curriculum conditions for training, the SPL was highest for the condition using all physics levels defined for the experiment, with two conditions returning zero values. Future researchers should consider providing adequate support when training complex-physics vehicle models. The run-time results revealed a benefit for experimental machines with a better CPU, at least for the vector-only observations we employed.

Keywords: Autonomous vehicles; machine learning; reinforcement learning; virtual environments

1. Introduction

A basic task for future autonomous vehicles in a real-life urban setting includes navigating from one location to another while following roads and avoiding obstacles. Researching autonomous urban vehicles not only helps prepare for real-world, everyday driving, but for vehicle models within constructive simulations, such as wargaming. Autonomous vehicles may be rigorously trained for such a basic task, while reducing logistical and safety issues, through computer simulations. In other words, a vehicle agent is able to gain artificial intelligence (AI) through machine learning in a virtual environment (VE). The aim of the present study is to provide a foundation on how various levels of vehicle physics in such training simulations can impact the effectiveness and efficiency of a machine-learning algorithm: the approach of reinforcement learning (RL) was used to train an agent represented as a vehicle with dynamic physics, including simple-, intermediate-, and complex-physics car models.

A question remains as to what types of physics matter for the aforementioned task. If we are to find physics aspects that are not important, their usage within a simulation becomes moot, adding

unwarranted time and effort to training. In contrast, if certain physics do matter, they should be considered for training future autonomous vehicles. Ultimately, this study is directed toward understanding the link between physics fidelity and agent behaviour (i.e., through different physics models).

Challenges exist in obtaining timely and realistic real-world data to train AI. Creating a set of AI-driven behaviours for advanced, ad hoc, and aggregate navigation behaviours is of specific interest to the U.S. Army One World Terrain (OWT) mission by contributing to mission needs, such as mission rehearsal (PEO STRI, n.d.). Effective indoor agent navigation behaviours have been implemented through VEs (Chaplot et al., 2020), but expansive outdoor agents capable of realistic urban and cross-country navigation are elusive. A continuing goal of the authors is to provide support that AI learning algorithms can be reliably trained using data provided through a high-resolution VE; and further, that time and valuable research funds will be saved by leveraging training data produced by VEs, as previously supported (Reed et al., 2018).

1.1. Background

1.1.1. Novel Simulator for Current Work

We present A Realistic Open environment for Rapid Agent training, or ARORA, a geospecific RL simulator. ARORA was developed to train agents to complete navigational tasks in a large-scale city environment with the complexities of physics-based movement, vehicle modelling, and a continuous action space. The goal of ARORA is to provide an open-source platform where scientists can explore AI tasks that simulate agents and the real world with high fidelity. ARORA is built on top of the Unity game engine, allows for headless training (i.e., without any graphical interface), and provides a flexible application programming interface (API) with several sensors.

In terms of other simulators for training autonomous agents, AI2-THOR (Kolve et al., 2019), iGibson (Shen et al., 2020), and Habitat (Savva et al., 2019) have provided 3D indoor environments, whereas CARLA (Dosovitskiy et al., 2017) and AirSim (Shah et al., 2017) have provided outdoor environments. See Appendix A for a comparison of these simulators and ARORA.

In ARORA, CARLA, AirSim, and Habitat, physically simulated agents are used for navigation, which can result in collisions that impede movement. Other simulations use simulated physics only for agent interactions with objects, such as with faucets in AI2-THOR and cabinets in iGibson. The ARORA simulator leverages realistic movement for navigating a geospecific physical environment with dynamic vehicular traffic. AI2-THOR, iGibson, and Habitat are limited to 3D-scanned panograph environments and discrete navigation with teleportation-style movements. CARLA and AirSim target realistic movement, but operate in geotypical environments. CARLA provides dynamic scenes due to autonomous vehicle and pedestrian traffic, whereas AI2-THOR and iGibson provide some complexity by allowing the agent to modify the environment with object interactions.

1.1.2. Reinforcement Learning (RL)

In RL, an agent is not given any kind of training data beforehand. Instead, the agent is provided with an environment in which it acts, and the environment provides two things in response to this action: a reward and an observation. The observation is the next state of the environment, and the reward is a positive or negative value assigned to the result of the action. The agent selects an action by following a policy, or set of guiding rules. Thus, when repeating actions several times, the agent generates sequences of experiences and learns the best actions to maximise the long-term sum of the rewards.

Training an agent for autonomous driving consists of three distinct stages: perception, planning, and control. Perception refers to an agent sensing the characteristics of the world and itself, the egocentric vehicle. The vehicle may have multiple sensors, such as cameras, radars, lidars, and a GPS. From the perception module, an agent can know its location, a map, and other information. Following the perception module, the planning module is used by the agent to build a plan or set of plans that define the route from the agent's current location to a goal location. The information from the planning module is then fed to a control module that controls the vehicle movement and collects feedback from motion; this feedback is then provided back to the perception module.

When training agents as robots or autonomous vehicles, deep reinforcement learning (DRL) within a simulation allows an agent to use the simulation to generate the experience sequences and deep learning algorithms to learn from the generated sequences. Popular algorithms for DRL fall into three broad algorithmic families: policy gradients, policy optimisation, and actor-critic methods. We used a Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al., 2016) in our study; the DDPG is based on policy gradient and actor-critic methods. The DDPG is an adaptation of Deep Q-Networks (Mnih et al., 2013, 2015) for the continuous action domain. In our context, DRL can be applied in two ways: end-to-end or separately to each part of the autonomous vehicle pipeline; DDPG follows the former approach.

1.1.3. Related Work With Vehicle Physics

There are several levels of physics fidelity that can be used when simulating the physical actions of an agent in a VE. Kinematic models describe a system in terms of its motion, without reference to the forces causing that motion. Dynamic models, on the other hand,

describe a system through those forces. These forces include friction, gravity, and momentum. A kinematic model has the advantage of a simpler physics calculation, helping improve a simulator's speed and possibly allowing an agent to learn behaviours in fewer training steps by limiting the number of factors the agent must account for. Although work has looked at various vehicle physics, a lack of literature was found comparing different levels of dynamic physics for training an RL agent rendered as a vehicle.

In a recent work, the kinematics bicycle model for trajectory planning was used (Polack et al., 2017), where the authors compared this model with a more complex 9 degrees-of-freedom (9-DoF) model and concluded that using the simpler kinematics bicycle model was enough to generate feasible trajectories for a low-level controller, to an extent: errors skyrocketed at high acceleration, whereas switching to a more complex dynamic model resulted in better performance at higher acceleration levels. In relation, dynamic models were also advantageous in situations where the behaviour of the vehicle was affected by the nonlinear factor of tire traction during a turn (Kang et al., 2014; Kong et al., 2015). Yet, these works do not inform RL researchers in terms of the impact of model complexity on RL-algorithm training.

Other conceptually similar work used different physics models for a vehicle and studied their impact on simultaneous localisation and mapping (SLAM) algorithms (Lehto & Hedlund, 2019), with the indication that for a known environment, the accuracy of pose estimation did not differ greatly for any of the models used; however, for unknown environments, the dynamic models resulted in better accuracy for pose estimation when the vehicle was exposed to large slip angles. An additional conclusion from the latter authors was similar to that of Polack et al. (2017): switching to complex physics models resulted in better performance at higher acceleration levels.

When comparing various motion models for vehicle tracking, having a certain complex model did improve performance as compared to a simpler model; but at a certain point, adding more complex motion models did not greatly improve performance (Schubert et al., 2008). The present study's physics models differed from the latter work's variations of physics.

This study's physics implementation has ignored factors with values that could vary, were unknown, or were expected to have a minimal impact on the system. Some of these factors were also ignored by another work: underlying brake and engine dynamics (Polack et al., 2017). Other ignored factors in our study were the exact aerodynamic properties of the vehicle, the weight distribution of the vehicle, and certain tire properties (e.g., tire tread).

1.2. Research Question

The following exploratory research question was defined for the study: are there any differences between levels of dynamic physics for a PointGoal Navigation (PointNav) task, in terms of simulator run-time, training steps, and agent effectiveness through the Success weighted by (normalised inverse) Path Length (SPL) measure?

2. Method

For our experiment, we included three main components. First, the primary component was the ARORA simulation engine, which has physics and a replica of real-world entities. Second, the NavSim API: a Python API that gave access to the functionality of the simulation engine and exposed Unity's OpenAI Gym interface for interacting with RL components. Third, the DDPG algorithm.

2.1. Model Framework

2.1.1. Task

The present study employed an absolute PointNav task (Kadian et al., 2020), the latter synonymous with the PointGoal task (Anderson et al., 2018). For this task, the agent was placed at a starting location and had the ultimate objective of reaching a goal location. Exploration points were placed at multiple locations within the VE. Both the start and goal locations were placed in the VE based on a given environment seed. An episode ended only if the agent fell off the map, reached the goal location, or reached a maximum number of training steps.

2.1.2. Observations

In the present study, the environment provided only vector observations, in the form = [agent position, agent rotation, agent velocity, goal position, proximity sensor]. However, the environment could be configured to also provide visual observations, additionally or exclusively. Visual observations = [raw agent camera, depth agent camera, segmentation agent camera] taken from the simulator are provided in Figure 1 for illustration, with the agent's visual input from an RGB-D camera system as perception. Figure 2 shows the photorealistic visualisation that produced the Figure 1 images.

Figure 1. Visual observation examples taken from the simulator (from left to right): raw agent camera, depth agent camera, and segmentation agent camera.

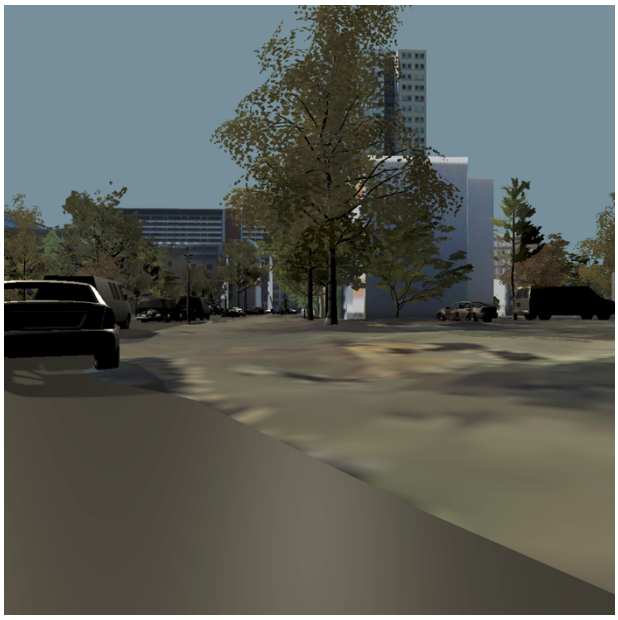

Figure 2. The photorealistic visualisation that produced the Figure 1 images.

2.1.3. Reward

The agent received negative rewards for collisions and falling off the map; and positive rewards for colliding with the exploration points and goal location. Values for these rewards from the environment are shown in Table 1.

Table 1. Rewards from the environment.

Event	Type	Value
Collided with goal	Positive	50
Collided with exploration point	Positive	0.005
Fell off map	Negative	-50
Collided with other objects	Negative	-0.1

2.1.4. Training and Testing

The four physics models used in the experiment are shown in Table 2. Each physics level was additive to the previous physics level, leading to additions of increased complexity (further physics implementation is discussed in Section 2.2). These four physics models formed the bedrock of four experimental conditions.

Table 2. Features per each physics model used in the experiment. Each model was additive to its predecessor, starting from simple physics and adding down the list.

Physics Model	Features
Simple	Collisions and gravity
Intermediate 1	Addition of wheel torque
Intermediate 2	Addition of suspension, downforce, and sideslip
Complex	Addition of traction control and varying surface friction

For each of the four conditions, an agent was trained in ARORA for a total training-length of 15000 episodes, with a maximum of 1000 steps per episode. A step consisted of an agent making observations and implementing an action based off those observations. An episode consisted of all of the steps taken and ended only when the goal was achieved, the agent fell off the map, or the maximum number of 1000 steps had passed. The distribution of the 15000 training episodes for each condition is shown in Table 3, with each percentage denoting a percentage of the 15000 training episodes. Although some conditions had skipped training stages, training patterns were sequentially consistent: a model with more physics features was never trained before a model with fewer physics features. The total time taken to train was recorded. The resulting trained agents were tested in ARORA using only the complex-physics model, with the SPL measure for agent performance recorded; the SPL was based on previous work (Anderson et al., 2018) and is shown in Equation 1. The training and experimental results were collected on two machines: a high-end server machine with multiple AI-specific GPUs (i.e., the V100) and a midrange quad-video-card machine (i.e., the Quad). Identical collection between machines was performed to offer a machine efficiency comparison. The machine specifications are listed in Table 4.

Table 3. Distribution of training episodes (hyphens denote a skipped training stage).

Condition	Training Stage			
	Simple	Intermediate 1	Intermediate 2	Complex
1	25%	25%	25%	25%
2	33.3%	33.3%	-	33.4%
3	50%	-	-	50%
4	-	-	-	100%

Table 4. Machine specifications for the experiment.

Machine Aspect	Aspect Specifications	
	V100	Quad
Operating System	Ubuntu 18.04	Ubuntu 20.04
Graphic Processing Unit(s)	8x V100 SXM2 32 GB with NVLINK (Total: 40960 Cuda Cores)	4x NVIDIA Quadro RTX 6000 24 GB
Central Processing Unit(s)	Dual Intel Xeon Gold 6248	Dual Intel Xeon Gold 5220R
Random Access Memory	768 GB, DDR4 ECC	256 GB, DDR4
Primary Storage	8 TB-SAS (12 Gbps) SSD storage, RAID 10 configuration	Single 1 TB SSD storage
Secondary Storage	8 TB (12 Gbps) Enterprise drives (rotating platter, 10 K) in a RAID 10 configuration	Single 8 TB-SAS (12 Gbps)

$$\mathrm{SPL} = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)} \quad (1)$$

In Equation 1, l_i was the shortest path length between

a goal and an agent's starting location, per episode; p_i was the path length taken by the agent per episode, S_i was a binary indication of success for episode i, and N was the total number of episodes.

2.2. Physics Implementation

The simple-physics model for the vehicle followed a basic rigid body using Unity's Rigidbody class. There were no forces used to propel the vehicle for travel. Instead, the Rigidbody's MovePosition member method was invoked, which moved the vehicle smoothly through the environment during simulation. The vehicle was still affected by gravity and collision with any objects in the environment, such as terrain, trees, and buildings. Thus, the vehicle followed a kinematic model with the addition of gravity to keep the agent on the ground.

The complex-physics model mimicked a simplified model of a real vehicle and served as the base from which the intermediate-physics models were incorporated. Each vehicle wheel was implemented using Unity's native Wheel Collider component, with the entire vehicle configured as four-wheel drive. The agent could increase throttle, which controlled the amount of torque produced by the vehicle; the amount of torque was then divided equally between the vehicle's drive wheels. Braking was also controlled by the agent and was applied to the wheels as brake torque.

Wheel suspension of the vehicle was enabled by Unity's Wheel Colliders and was configured to allow for a small amount of compression and bounciness by setting the spring coefficient to 70000 N · m and the damper coefficient to 3500 N · s/m. For the configuration of the Intermediate 1 physics model, suspension was disabled by zeroing out the distance at which the springs could travel. Downforce was applied by adding a downward force on the vehicle that was affected by the current velocity of the vehicle.

When considering elements of tire grip and traction for the vehicle, tire sideslip, traction control, and friction were included as varying elements. Though tire sideslip was included in Unity's Wheel Colliders, this interaction was ignored for the Intermediate 1 physics model by rotating the direction of the vehicle's velocity while turning to keep the vehicle moving in the direction the vehicle was facing. A simple traction control system was implemented by monitoring the tire slip in motion and adjusting the wheel torque to mitigate slippage. The effects of varying surface friction were simulated by modifying each Wheel Collider's stiffness property (i.e., friction curve coefficient), for both forward and sideways friction, depending on whether the Wheel Collider was either on or off a road.

2.3. Simulation Environment

The simulation environment used for ARORA was based on a dense and publicly available dataset utilising the city of Berlin, Germany. The initial source data obtained from the 3DCityDB project (Technical University of Munich Chair of Geoinformatics, n.d.) was provided as a well-formed dataset derived from a series of satellite images, tiled (lidar-based) terrain data, and correlated building meshes. The base terrain was encapsulated in the CityGML (Kolbe, 2012) format, the latter maintained through the Open Geospatial Consortium (OGC). The Institute for Simulation and Training, a part the University of Central Florida (UCF), augmented the aforementioned dataset to include other important environmental features, such as individual trees, vehicles, roadway networks, walkway networks, and railroads. A database of individual point-features, which was carefully maintained and cultivated by UCF, is available to the end user of the ARORA simulator.

The terrain's import process was achieved with an original terrain-import tool: the Unity Terrain Importer (UTI; Thomas et al., 2020). The UTI was augmented to handle geospatial feature data for points and vectors through a series of special Unity scripts that imported feature details and preserved geospatial attribution that is commonly lost through traditional mesh-only import methods. For example, point trees in ARORA maintain attributes that can be dynamically queried, such as a plant's species, genus, trunk size, leaf cycle, age, width, and height. The ARORA developers preserved as much significant detail of the source data as possible for the Unity rendering environment. A specified goal of ARORA is to support the widest possible potential learning algorithms, some of which may depend on attribution that is generally left out of other systems.

The ARORA interface provides users with a key capability to query individual object attribution to a granular detail. For example, using a Unity Machine Learning Agents (ML-Agents) custom side-channel, a developer can issue a query to the simulation to retrieve the attribution of a current object in view. A concrete example is the following custom side-channel call, which requests attribution from the location defined by the first three elements of the second parameter:

> "position_scan_side_channel.send_request("positionScan", [177.9137,35.11.13,28.1348,100])"

An example return of the previous query is a vector formatted as follows:

> "X:390364.9843,Y:5818783.567, LATITUDE:52.50813726, LONGITUDE:13.38458, COMMONNAME:SAND BIRCH; WHITE BIRCH, AGE:30,HEIGHT:7.83, LEAF_CYCLE:deciduous, NATURAL:tree, AVG_SPREAD:6.9, MAX_SPREAD:8.5, FOREST:NO, MODEL_PATH:Assets/Trees/TreePrefabs/Betula_Birch_Deciduous,MODEL_NAME:Betula_Birch_Deciduous_10M.prefab,MAX_HGT_FR:0.783,MIN_HGT_FR:0.783".

3. Results and Discussion

Results are given in terms of effectiveness and efficiency measures. The SPL constituted

effectiveness, whereas the simulator run-time and number of training steps constituted efficiency. Objective measures were taken to support discussion points. Conditions 1, 2, 3, and 4 are represented by C1, C2, C3, and C4, respectively.

3.1. Effectiveness

Table 5 shows the SPL measure collected across a combination of parameters: each agent's condition and each condition's training and testing sets. Table 5 is used to quantify the effectiveness of each condition's agent given the SPL measure (the closer the value was to the maximum of 1.0, the more effective the agent was in reaching the goal using the shortest path). Figure 3 shows each agent's total reward for each episode across conditions. Figure 3 is used to show how successful an agent was throughout its training timeline.

Table 5. The Success weighted by (normalised inverse) Path Length (SPL) measure for agents, across conditions for the training and testing sets.

Condition	SPL	
	Training Set	Testing Set
1	.230637175	.19
2	.0	.0
3	.0	.0
4	.003816329	.007921156

The results from Table 5 show that the C1 agent's SPL measures were meaningfully higher than those for C2, C3, and C4. These differences suggest that the curriculum provided by C1 provided a tangible benefit on navigational performance with reduced training time. Given that C1 was the only condition that introduced all physics levels during training (where each physics level corresponded to a specific physics model), we expect the exposure to each physics level during training provided the agent time to learn from each level and hence produced a higher SPL measure during testing.

When comparing C4 to C2 and C3, we can see a small uptick in the SPL measure within C4. This might be due to the agent exclusively training and testing on the same physics level (i.e., the complex-physics model). It is expected that when trained and tested on the same domain, the agent will eventually learn an optimal strategy. Still, this SPL measure uptick was small for C4 and thus could be due to chance. Figure 3 shows that the C2 agent had sparse successful rewards throughout training, and the C3 agent had reduced successful rewards once it started training on the complex-physics model. Figure 3 supports that the change in physics levels for C2 and C3 was too great for the agent to adapt to new physics levels and maintain a successful strategy.

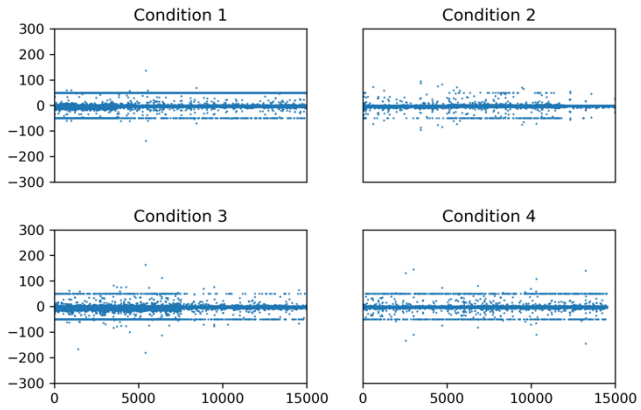

Figure 3. Agent's total reward per each condition. Each x-axis shows the episode number, and each y-axis shows the total reward.

3.2. Machine Efficiency

An identical set of the experimental conditions were run on the V100 and Quad machines (see Table 4 for each machine's specifications). Since the experiment was constrained to leverage vector observations, we can see from Figure 4 that the machines were primarily bound by CPU performance. Figure 4 also shows that the overall run-time was faster on the Quad machines than the more graphically capable V100 machines. This benefit may be explained when looking at specific aspects of the CPU for the Quad (Intel Corporation, n.d.-a) and V100 (Intel Corporation, n.d.-b) machines: the Quads had a higher maximum frequency of 4.0 GHz, whereas the V100 server had a maximum 3.9 GHz; and (more importantly) the CPU in the Quad machines supported eight more concurrent threads per CPU than the V100 machines. Ultimately, if only vector observations are needed, additional GPU power may become marginal for agent performance.

The total run-time between conditions was affected greatly by how well the agent performed within the condition. The overall number of steps was lower within a condition if the agent localised prior to the maximum number of allowed steps; this link is illustrated in Figure 4 when comparing run-times with the number of training steps, the latter an average of both machines per condition.

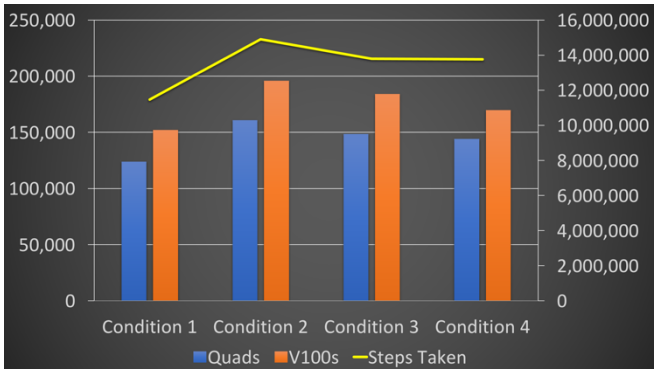

Figure 4. Run-time vs. training steps taken for Quad and V100

machines. The leftmost y-axis shows run-time in seconds for each vertical bar. The rightmost y-axis shows the number of training steps via the yellow line, using the average of both machines per condition.

As always, there is a trade-off between models: higher fidelity means higher computational cost, with lower fidelity meaning lower computational cost. For agent training, however, lower-fidelity models could be used: policies can be built with greater speed on lower-fidelity models, with fidelity scaled upwards to develop the final policies. In Figure 4 we see the computational overhead associated with each of the conditions: the highest computational load being placed on C2 and C3, with a lower overall load on C1 and C4. The run-time and training steps for C1 indicates that initially learning within the simple-physics realm before scaling the policies accordingly is best.

4. Conclusion

Overall, this study showed that levels of physics, when integrated into an agent-training curriculum, can have an impact on a virtual vehicle's navigation effectiveness. Specifically, the most effective agent was produced from the condition that gradually elaborated on complexity through every physics level used in the experiment: the results indicate that a comprehensively staged curriculum can provide the agent with a learning benefit with reduced training time. Removing less complex physics levels within the curriculum made navigation more challenging. Interestingly, not every curriculum led to the same performance, despite the fidelity progression of simpler to more complex physics levels being maintained. Future researchers should consider the need for providing adequate support when training complex-physics vehicles using an RL methodology. The efficiency behaviour of different machines used for identical experiments showed that training solely with vector observations places more demand on the CPU than the GPU, thereby informing agent-training requirements for reducing run-time. Run-time was also impacted by agent performance, whereby agents being localised before reaching a max-step limit reduced run-time. The effectiveness and efficiency results worked in associative harmony, where a beneficial curriculum of scaling policies provided improvements in agent effectiveness and machine efficiency.

4.1. Limitations

The technology used for the study presented limitations. Training time in Unity was limited, thus affecting how long agents could be trained. Further, scaling with multiple simulations on one machine was unavailable, as only one simulation was allowed per machine; this was caused by the way Unity ML-Agents depended on an X-Server in order to provide visual observations. As a potential future limit, there was also an issue with scaling one simulation instance for multiple agents: the Unity ML-Agents Gym wrapper limited one agent per simulation.

4.2. Next Steps

This study provides an exploratory foundation for agent behaviour as affected by physics when using DRL in a VE. Still, further research may elaborate on this foundation. Adding visual observations, more episodes, longer goal distances (with a higher number of maximum steps), a larger training-and-testing environment, various weather effects, various vehicle types, new physics types, new performance measures, and other RL algorithms are potential avenues to explore. For the latter, a hierarchical and modular algorithm with DRL applied to each part of the autonomous vehicle pipeline may be used. In terms of performance, we may also explore the measurement of Success weighted by Completion Time (SCT), which looks at the fastest path (that is, not necessarily the shortest path) as dependent on vehicle dynamics (Yokoyama et al., 2021). Finally, the use of fine-grain attribution (e.g., the plant properties previously discussed) as semantic annotation may be investigated for its effect on agent behaviour through DRL, although not necessarily through physics levels.

Funding

This research was funded by the U.S. Army Combat Capabilities Development Command Soldier Center (CCDC SC), grant number W911NF-19-2-0104.

Acknowledgements

This research was funded by the SFC Paul Ray Smith Simulation and Training Technology Center (STTC) Soldier Effectiveness Directorate (SED) U.S. Army Combat Capabilities Development Command Soldier Center (CCDC SC). However, the views, findings, and conclusions contained in this presentation are solely those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

This paper is dedicated to a dear friend and coworker, Mr Nathan A. Hubbard. Through his tireless attention to detail and expert skillset, the highest quality geospecific, feature-rich environment was realised. His presence and inspirational personality will be missed. All the authors are indebted to his willingness to push the limitations and demolish expectations on what is the status quo.

Appendix A. Comparison of Simulators

Figure 5 provides a select list of similar simulators and how they compare with ARORA.

ENVIRONMENT	ENVIRONMENT REALISM	AGENT MOVEMENT RESTRICTIONS	PHYSICS INTERACTION • MASS • FORCES	INDOOR/ OUTDOOR	DYNAMIC SCENE	NAVIGATION	AGENT MODELLING	ENGINE	FRAMES PER SECOND (FPS)	CITATION
ARORA	GEOSPECIFIC	CONTINUOUS	AGENT AND ENVIRONMENT DYNAMICS	OUTDOOR	TRAFFIC SIMULATION	CONTINUOUS	PHYSICALLY MODELLED CAR	UNITY		
AI2-THOR			AGENT ACTS ON INDOOR OBJECTS	INDOOR	INDOOR OBJECTS INTERACTIONS	DISCRETE	CAPSULE-SHAPED ENTITY	UNITY	70	Kolve et al. (2017)
CARLA	GEOTYPICAL			OUTDOOR	TRAFFIC & PEDESTRIANS		VEHICLE	UNREAL ENGINE 4	10	Dosovitskiy et al. (2017)
AirSim		CONTINUOUS	HIGH-FIDELITY AGENT & ENVIRONMENT	OUTDOOR	STATIC	CONTINUOUS	PHYSICALLY MODELLED DRONE & CAR	UNREAL ENGINE 4		Shah et al. (2017)
Habitat		TELEPORT MOVEMENT	ONLY COLLISIONS ON MOVEMENT	INDOOR	STATIC	DISCRETE	LoCoBot		10000	Savva et al. (2019)
iGibson				INDOOR			LoCoBot			Shen et al. (2020)

Figure 5. Select comparison between ARORA simulator and similar simulators. A blank cell indicates information that was not found.

References

Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., & Zamir, A. R. (2018). On evaluation of embodied navigation agents. *ArXiv:1807.06757v1*.

Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., & Salakhutdinov, R. (2020). Learning to explore using Active Neural SLAM. *ArXiv:2004.05155v1*.

Dosovitskiy, A., Ros, G., Codevilla, F., López, A., & Koltun, V. (2017). CARLA: An open urban driving simulator. *1st Conference on Robot Learning (CoRL 2017)*, 1–16.

Intel Corporation. (n.d.-a). *Intel® Xeon® Gold 5220R Processor (35.75M Cache, 2.20 GHz)*. Retrieved July 23, 2021, from https://www.intel.com/content/www/us/en/products/sku/199354/intel-xeon-gold-5220r-processor-35-75m-cache-2-20-ghz/specifications.html

Intel Corporation. (n.d.-b). *Intel® Xeon® Gold 6248 Processor*. Retrieved July 22, 2021, from https://ark.intel.com/content/www/us/en/ark/products/192446/intel-xeon-gold-6248-processor-27-5m-cache-2-50-ghz.html

Kadian, A., Truong, J., Gokaslan, A., Clegg, A., Wijmans, E., Lee, S., Savva, M., Chernova, S., & Batra, D. (2020). Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, *5*(4), 6670–6677.

Kang, C. M., Lee, S.-H., & Chung, C. C. (2014). Comparative evaluation of dynamic and kinematic vehicle models. *53rd IEEE Conference on Decision and Control*, 648–653.

Kolbe, T. H. (2012). *CityGML*. http://www.citygml.org/citygml.org.html

Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Gordon, D., Zhu, Y., Gupta, A., & Farhadi, A. (2019). AI2-THOR: An interactive 3D environment for visual AI. *ArXiv:1712.05474v3*.

Kong, J., Pfeiffer, M., Schildbach, G., & Borrelli, F. (2015). Kinematic and dynamic vehicle models for autonomous driving control design. *2015 IEEE Intelligent Vehicles Symposium (IV)*, 1094–1099.

Lehto, H. S., & Hedlund, R. (2019). *Impact of vehicle dynamics modelling on feature based SLAM for autonomous racing: A comparative study of the kinematic and dynamic vehicle models*. KTH Royal Institute of Technology.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016*.

https://arxiv.org/abs/1509.02971

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. *ArXiv:1312.5602v1*, 1–9.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533.

PEO STRI. (n.d.). *One World Terrain (OWT)*. Retrieved May 17, 2021, from https://peostri.army.mil/one-world-terrain-owt

Polack, P., Altche, F., D'Andrea-Novel, B., & de La Fortelle, A. (2017). The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? *2017 IEEE Intelligent Vehicles Symposium (IV)*, 812–818.

Reed, D., Thomas, T., Eifert, L., Reynolds, S., Hurter, J., & Tucker, F. (2018). Leveraging virtual environments to train a deep learning algorithm. *17th International Conference on Modeling and Applied Simulation, MAS 2018*, 48–54.

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., & Batra, D. (2019). Habitat: A platform for embodied AI research. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9338–9346.

Schubert, R., Richter, E., & Wanielik, G. (2008). Comparison and evaluation of advanced motion models for vehicle tracking. *2008 11th International Conference on Information Fusion*.

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2017). AirSim: High-fidelity visual and physical simulation for autonomous vehicles. *ArXiv:1705.05065v2*, 1–14.

Shen, B., Xia, F., Li, C., Martín-Martín, R., Fan, L., Wang, G., Buch, S., D'Arpino, C., Srivastava, S., Tchapmi, L. P., Tchapmi, M. E., Vainio, K., Fei-Fei, L., & Savarese, S. (2020). iGibson, a simulation environment for interactive tasks in large realistic scenes. *ArXiv:2012.02924v2*.

Technical University of Munich Chair of Geoinformatics. (n.d.). Semantic 3D City Model of Berlin. *The CityGML Database 3D City DB*. Retrieved May 18, 2021, from https://www.3dcitydb.org/3dcitydb/visualizationberlin/

Thomas, T., Hurter, J., Winston, T., Reed, D., & Eifert, L. B. (2020). Using a virtual dataset for deep learning: Improving real-world environment re-creation for human training. *19th International Conference on Modeling and Applied Simulation, MAS 2020*, 26–33.

Yokoyama, N., Ha, S., & Batra, D. (2021). Success weighted by Completion Time: A dynamics-aware evaluation criteria for embodied navigation. *ArXiv:2103.08022v1*.