Investigating Determinism of Gaming Engines for Autonomous Vehicle Verification

Abanoub Ghobrial[1,3], Greg Chance[1,3], Severin Lemaignan[2,3], Tony Pipe[2,3], and Kerstin Eder[1,3]

[1]Trustworthy Systems Lab, University of Bristol, Bristol, UK
[2]University of West of England, Bristol, UK
[3]Bristol Robotics Lab, Bristol, UK

***Abstract*—The connected and autonomous vehicle (CAV) community are adopting the use of gaming engines to develop vehicle control systems and testing environments using simulation. These engines must be deterministic for finding and resolving defects (bugs) in software and ensuring auditable and repeatable tests can be executed for verification and validation (V&V). This work highlights the requirements these engines need (to fulfil this purpose/for CAV development) and proposes a method to determine *simulation variance* for specific configurations and resource utilisation. A case study is used to illustrate the proposed method and results from a small scenario indicate the presence of deterministic operational domains.**

***Index Terms*—Autonomous Driving, Determinism, Physics Engines, Connected Autonomous Vehicles (CAV), Verification and Validation(V&V)**

I. Introduction

Verification is the process used to gain confidence in the correctness of a system with respect to its requirements [1]. Simulation based verification can be used to test autonomous driving functions under development benefiting from full control over the road network and the actors within it. These simulated tests aim to provide evidence to regulators of the functional safety of the vehicle or its compliance with commonly agreed upon road conduct [24], national rules [19] and road traffic laws [22]. There have been a number of fatalities with autonomous vehicles which could be attributed to a lack of verification and validation (V&V), e.g.[17]. Simulation is an obvious domain choice to explore the vast parameter space in a safe and efficient manner [12] without millions of miles of costly on-road testing [9] where the rare event can be made to happen more often [11] such as unpredictable actor behaviour [7].

For games engines to offer a suitable testing environment they must: operate with a minimal *reality gap* in the physical domain (vehicles, actors), be realistic enough [11] including suitable rendering for perception stack testing, be easy to setup, run and manipulate the environment and the temporal development of actors [23], include the provision for sensor analogues to a suitable accuracy (e.g. video, LIDAR, IR), allow hardware-in-the-loop development and provide a suitable real-time test-bed for cyber-security testing [8]. Game engines offer a simulation domain solution for the development and testing of CAVs that meet many of these requirements but many challenges still exist.

For games engines to be useful as a verification tool then they must be deterministic. Deterministic, in this context, refers to the property of causality given a temporal development of events such that any state is completely determined by prior states. Herein the term *simulation variance* is used to refer to how simulation runs vary when repeated many times and specifically the variance of agent paths within a given scenario. We will adopt the terminology defined in [23], where *scene* refers to all static objects including the road network, street furniture, environment conditions and a snapshot of any dynamic elements. Dynamic elements are the elements in a scene whose actions or behaviour may change over time, these are considered actors and may include other road vehicles, cyclists, pedestrians and traffic signals. The *scenario* is then defined as a temporal development between several scenes which may be specified by specific goals and values. A *situation* is defined as the subjective conditions and determinants for behaviour at a particular point in time. Determinism is important for verification. Any software defects that are detected but not repeatable at a later time or by another party does not allow for suitable resolution. Furthermore, defects not found on the verification testing system may arise on a deployed system with a different hardware architecture resulting in untested and potentially unsafe vehicle behaviour. Our main research question is to what extent are gaming engines deterministic so that they can provide a reliable and repeatable testing environment for CAV verification? The game engine, in this context, must also comply with the requirements to provide a suitably realistic environment and real-time processing for perception and security analysis.

This paper is structured as follows. In Section II a background on gaming engines and sources of potential non-determinism are explored. Section III presents a method to design and execute an experiment to assess suitability of a system for CAV verification. A case study to illustrate the level of simulation variance for a number of scenarios is explored in Section IV. We conclude in Section V and give an outlook on future work.

II. Background

There are numerous game engines with their associated development environments which could be suitable for CAV

[1]{abanoub.ghobrial, greg.chance, kerstin.eder}@bristol.ac.uk
[2]{severin.lemaignan, tony.pipe}@brl.ac.uk

development, e.g. Unreal, Unity, CryEngine. Specific autonomous driving research tools have been created to abstract and simplify the development environment, some of which are based on existing engines, e.g. Carla, AirSim, Apollo, and some have been developed specifically, e.g. the could based Nvidia Drive Constellation simulator. **TODO: ref each of these** Investigating the determinism of gaming engines has not attracted much research interest since performance is more critical for playing games than accurate and repeatable runtime execution. However, when these engines are used for CAV development then deterministic behaviour is required. This section gives an overview of a gaming engine and what sources or settings in the engine may affect *simulation variance*.

A. Overview of a Game Loop

The game loop is responsible for the interaction between the maths, physics and rendering engines. Figure 1 depicts a basic representation of the process flow in a game engine loop. A game loop is broken up into three distinct phases: processing the inputs, updating the game world (Physics Engine), and generating outputs (Rendering). [6]

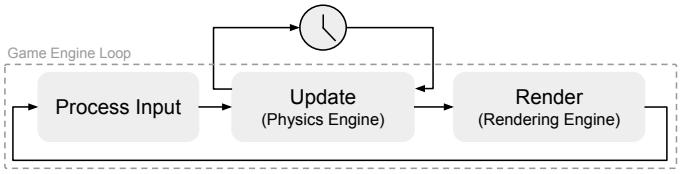


Fig. 1: Game engine loop block diagram

The first part of the game loop is to process the user inputs which may take the form of user keyed inputs or, in the case of CAV development, the resultant actions of the autonomous vehicle given the current state of the environment. For example, the throttle and braking inputs from the PID controller controlling the speed of an AV. The update interprets the intended actions of the AV along with all other dynamic actors in the scene and performs any necessary physics calculations and updates the actor states. **TODO: For AG: what does the clock symbolise, need to explain it?** The rendering engine then takes the updated actor states and renders the scene [16]. **TODO: For AG: you cite this book 4 times which might be too much as a general reference, is there something specific you want to draw the reader towards in the book? What specifically about the rendering part of this cycle does the book help to illustrate the point?**

It is necessary to have an understanding of how rendering and physics engines operate together in order grasp some of the sources of non-determinsim that will be mentioned.Briefly, the game loop operates as follows:[16] (see Figure 1)

- At the beginning of each tick (scene), the lag between the game clock and the real world is updated based on how much real time passed. This measures how far the game's clock is behind compared to the real world.
- Then the user inputs are processed.
- There is then an inner loop to update the physics engine (clock symbol in Fig. 1), incrementing at fixed step until the game clock is equal to the real world. The physics engine uses a fixed time step, because it makes everything simpler and more stable for physics and AI. The shorter this fixed time step is, the more processing time it takes to catch up to real time and the more deterministic the engine becomes and vice versa[6][16], as will be explained in section II-B. Thus, ideally the time step should be as short as possible, so that simulations run with high fidelity on fast machines. However, if the fixed time step is too short i.e. less than the time it takes to process an "Update" inner loop on some (slow) machines, then the simulation will simply never catch up on these slow machines.
- Rendering occurs once the physics engine catches up. The process then starts again.

The time step, dt, in any physics calculation is important. To use a fixed physics time step the user's display refresh rate needs to be known in advance. This requires an update loop to take less than one frame of real world time [4]. Given the different capabilities of game players hardware a variable delta time can be implemented taking the previous frame time as the next dt. However, variable dt can lead to different physical results and in some cases unrealistic physics. Semi-fixed or limited frame rates ensure dt does not exceed some user-defined limit to meet a minimum standard of physical representation but allows computational headroom for slower hardware. Some engines provide sub-stepping which does multiple physics calculations per frame at a greater CPU cost [1]. If the engine tries to render between updates *residual lag* can occur [6][16] and extrapolation between frames is performed to smooth transition between scenes.

B. Sources of non-determinism

After establishing an understanding of how physics engines work, the focus will now be on what causes these engines to be non-deterministic. The main reasons that could cause non-determinism are discussed below.

Floating Points: A generic issue with computers are floating points precision. Various errors occur when doing arithmetic manipulations using floating points, especially when operating between large and small numbers due to rounding, memory limitations and so forth.[5][15]. Translating to the usage of CAV simulations for V&V in physics engines, this could result in many precision issues resulting in complex non deterministic behaviours.

For instance, consider a simulation where the world is infinite and progressively generated in the physics engine. After a few hundred meters, precision issues will start to occur, and will get progressively worse the further from the origin the simulation gets. This is due to the values of the coordinates being used in calculations are getting bigger and hence will not cope with small values that are critical for V&V testing purposes. A solution to this problem is perhaps

[1] https://docs.unrealengine.com/en-US/Engine/Physics/Substepping/index.html

every time the simulations moves 100 meters away from the origin the entire world will move by 100m in the opposite direction. Thus, avoiding getting into floating point issues.[5]

There are many scenarios that could be addressed, and solutions to them will entirely depend on the characteristics of that specific scenario. There is no generalised solution for the floating problem, regardless of how good computers get there will always be a limitation. However, floating points as they stand could just be sufficient for V&V testing purposes. Whenever they are not, smart solutions can be found to accommodate for these limitations.

Navigation meshes and AI: Some CAV simulator developers claim that the built-in physics engines' AI is non-deterministic[2]. The validity of this depends on what are the built in AI algorithms being used by the engine. Most of them tend to use the A* algorithm[10], which is an algorithm with deterministic behaviour if the environment is deterministic[14][18]. Therefore the determinism of the AI depends on the determinism of the engine and how its navigation meshes are created or modelled. It is interesting to note that changes can occur to meshes every time the simulation is loaded. This is mainly resorted to the different scheduling of threads.

Physics and Rendering clocks: The inherent way of how these physics engines operate (i.e.game loops) causes the engine to be non-deterministic. A follow up from section II-A, these engines use a variable frame rate, which is good for hardware scalability but creates a challenge for the physics engine which works best with small fixed time steps. The problem of having a variable frame rate could also be a reason for the non-deterministic behaviour of these engines. This issue could perhaps be of negligible significance if the time steps are small enough.

Scheduling of threads: Scheduling of threads is believed to be the main reason for the non deterministic behaviour of these engines. Given the same hardware the output should always be the same, unless scheduling of the threads change, this would then cause non-deterministic behaviours. To solve this one would have to control all of the threads, which is a very involved and to achieve that one would need to replace or control the whole run-time system to allocate the same threads in the same order.

III. Methodology

A method for determining the level of simulation variance of an engine is presented in this section and the working flow of the method is shown in Figure 2. Each of the steps in the method is elaborated in the following subsections.

A. Design experiment

The experiment should be a simple scenario that can be easily repeated with the same initial conditions. Logs should be used to record the actor positions at fixed time intervals throughout the simulation. For each actor the variance in trajectory can be determined and the deviation from the mean, S_t, of the trajectory monitored at each time step. Averages over time and for different actors can be used to condense results down to single figures which is done for some results.

We use game engine actors to simulate the AV and pedestrians. Actors can either follow a series of trajectory waypoints or can plan a route to a further destination using some path planning and obstacle avoidance algorithms, e.g. A* search algorithm. To simulate the demands of realistic rendering and complex physics calculations in a simple scenario, stress routines are used to artificially occupy system resources. Further to this, the experiment should be designed to create collision callbacks to the physics engine by allowing actors to collide during the scenario. The user should also design the experiment to monitor and record system CPU and GPU utilisation. Additionally, if the user is interested in real-time assessment of the simulation then FPS should also be monitored and logged. Where possible a fixed dt should be used and any random numbers used should be seed controlled.

It is crucial to also make sure that any randomisation factors are eliminated in the experiment.

B. Internal settings

There are several error and physics collision internal settings in physics engines that can be tweaked to enhance the simulation variance of the engine.

Increasing the physics time-step calculations, which is increasing the number of calculations per unit time, would improve the level of simulation variance since the physics sequence will be more finely defined; but on the other hand this would be more computationally expensive. However, in such applications of V&V of CAV simulations, the computational cost should be of less concern compared to the importance of repeatability of tests.

If the experiment setup includes the usage of the physics engine's AI then altering the navigation mesh settings in the engine, like increasing the granularity of the mesh or changing the mesh type; can improve the level of simulation variance as a result of allowing the AI to navigate in a more well defined space.

Disabling rendering or running in headless mode is a way of attempting to entirely decouple the rendering engine from the physics engine, which theoretically should affect positively the repeatablility of tests. The downside of most physics engines is that they do not provide a headless mode in the editor setup, thus this could make running in headless mode a real challenge for verification engineers.

Running in editor or deployed mode can sometimes cause massive differences in performance, with deployed mode being worse, mostly due to the way settings get packaged when in deployed mode.

C. External settings

Running other programs in the background, like having a web browser open, or running the physics engine in the background while performing other tasks on the same machine does also have an effect; because it alters the CPU and GPU utilisation of tasks.

Trying to run stress experiments on computers by running other programs makes it difficult to control experiments since the utilisation would be inconsistent, nonetheless, there exists dedicated stress utilisation programs which provide consistent CPU and GPU stressing. Note that during running these tests computers should be left alone and not to be used for any other purposes apart from running the experiments, else the CPU + GPU stressing will lose its consistency through out a given test.

D. Running experiment

When it comes to running the experiment itself, things that one should consider are the problem space exploration; the number of runs that should be executed to get reliable results; the frequency of logging data of actors; and possibly having a program to monitor the hardware utilisation.

E. Output and analysis

Once the experiments are run, the logged data is post processed to find the variance in the logged data between the different runs. Note that bifurcation effects can cause a jump in the variance, so it is worth plotting the different logged paths in order to determine if there are any bifurcation effects.

This whole process is repeated for various internal and external settings (if needed). Then various plots can be created to draw the engine's simulation variance. Thereon, it would be up to the verification engineers to determine where they want to define the line below which the simulation variance would be acceptable, and thus know at which levels of computational utilisation they can guarantee to run repeatable experiments.

IV. Case Study

Given that our interest is on CAV simulations, this study presents a number of experiments run on typical agents used in CAV simulations and their interactions with each other. These agents are classified into two, either vehicles(cars, motorcycles, bikes etc.) or pedestrians (can include animals).

Here we will walk through our hypothesis, description of our tests and the analysis of results from the different tests. We will point out at each stage where we sit in Fig 2 as we go through this case study.

A. Problem And Hypothesis

We want to see whether using Unreal Engine 4 (UE4) along with CARLA [3] for V&V of CAV simulations is deterministic or not; if not then what is the level of simulation variance that we will operate with, is it acceptable or not, and whether it worsens with stressing GPU and CPU.

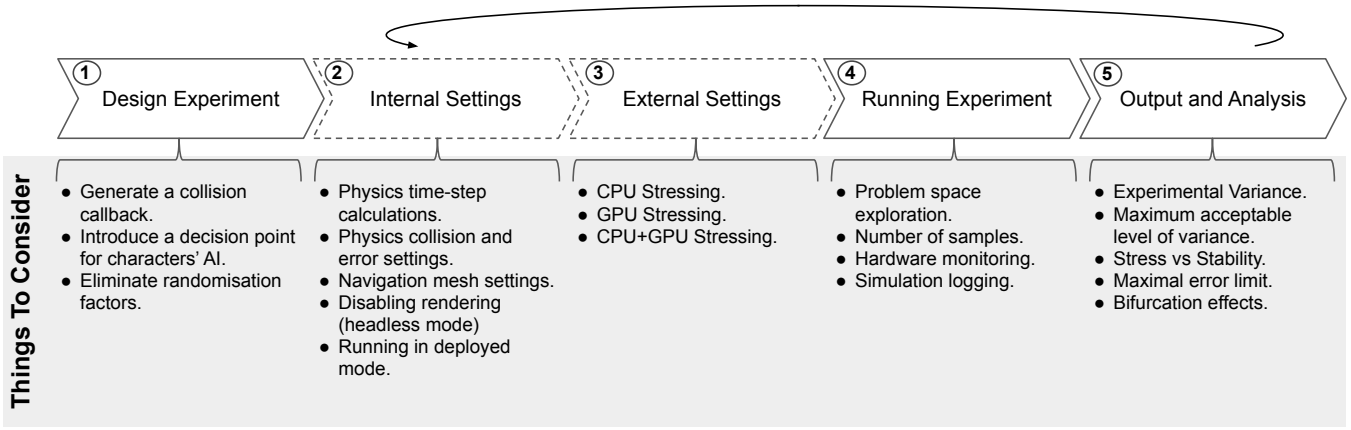


Fig. 2: Shows a flow diagram of the methodology proposed to define the level of non-determinism of a physics engine.

Test ID	Test description (See Fig 3a and 3b)	Collision/Intersection	Collision Type	Look ahead distance (m)	No. of repeats
1	Two vehicles driving	No	N/A	2	1000
2	Two vehicles driving	Yes	Vehicle and Vehicle	2	1000
3	Two vehicles driving and a pedestrian	No	N/A	2	1000
4	Two vehicles driving and a pedestrian	Yes	Vehicle and Pedestrian	2	1000
5	Two pedestrians	No	N/A	2	1000
6	Two pedestrians	Yes	Pedestrian and Pedestrian	0.4	1000
7	Two pedestrians	Yes	Pedestrian and Pedestrian	2	1000
8	Two pedestrians	Yes	Pedestrian and Pedestrian	20	1000

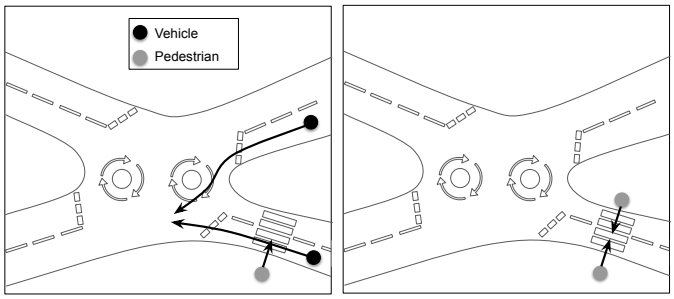
TABLE I: Set of experiments

Our hypothesis is that since UE4 is a gaming engine it is not deterministic, however, the level of simulation variance would be very low to acceptable limits that allows verification engineers to use it confidently and assume that tests would be repeatable. This behaviour is expected not to hold as the level of computing stress increases.

B. Tests Description And Technicalities

The vehicles in this simulator follow a trajectory of waypoints using a PID controller and they have a look ahead distance that they target for from a given list of trajectory waypoints. Similarly with the pedestrians, but they don't have a PID controller and they can do dynamic path planning using A* algorithm to reach their destination or in this case a trajectory waypoint.

Starting from step 1 in the methodology diagram (Design Experiment); we are using a double mini roundabout for our experiments. The list of tests shown in Table I were chosen to cover the mandatory interactions between the different types of agents. These are summed up in the following: **i)**Agents (vehicles and pedestrians) moving with no collisions or interactions with each other **ii)**Collision between vehicles **iii)**Collision (intersection) between pedestrians **iv)**Collision between vehicles and pedestrians.



(a) (b)

Fig. 3: Shows the setup of the different tests in Table I (a)Test IDs 1 to 4 (b)Test IDs 5 to 8.

Tests 1, 3and 5 are similar and there are no collisions in them, they were done as separate tests (instead of combining them into one) for the sake of simplicity and to use them as base lines for the tests-collision versions of them. The tests for collisions and non-collisions differ by a very slight change in the trajectory of the agents just to make them avoid colliding or intersecting with each other.

Test 2 is a collision between two vehicles and no pedestrians involved and is depicted by 3a, where the two vehicles will approach the roundabout and crash.

Tests 6,7 and 8 are tests with pedetrians having the same trajectory waypoints but walking in opposite directions to each other (see Fig 3b). Pedestrians have the functionality of dynamic path planning i.e. they can avoid obstacles to reach their waypoints and so they can avoid bumping into each other. Hence, to make sure they intersect (collide) with each other then we have set tests 6,7 and 8 to be the same but with different look ahead distances; the smaller the look ahead distance the less chance they will get to path plan around each other, and thus making sure they intersect.

In V&V of CAV simulations, collisions of pedestrians together is not of much interest, nevertheless we decided to include these tests for the sake of completeness and to obtain a full image of the simulation variance of the simulator.

Test 4, is where a vehicle hits a pedestrian and runs over it on a zebra line, as shown in Fig 3a.

Proceeding to step 2; there is about 20 different internal settings related to our context that can be tweaked in UE4; such as *Enable Enhanced Determinsim*, *PhysX Tree Rebuild*, *Max Physics Delta Time* etc. By doing various inspections around them we did not notice much improvement or worsening in the simulation variance between test runs, therefore the internal setting option was disregarded.

On the other hand the external settings (step 3), where CPU + GPU are stressed, did show significant variations in the results as will be discussed in section IV-C. Different levels of stress were applied to the computer; using the `stress` function [21] to stress the CPU and `gpu-burn` from github [20] to stress the GPU.

In terms of running the experiments (step 4), **i)** the computer used is an Alienware Area 51 R5 with an i9 9960X CPU chip, NVIDIA GeForce RTX 2080 graphics card and 64GB of RAM; **ii)** `nvidia-smi` and `htop` were used to monitor the levels of GPU and CPU stress respectively before and while the experiments were executed [13]; **iii)** the computer was left alone and not used for any other tasks while the runs were executed.

C. Results And Discussion

We start by analysing (step 5) the summary from all of the tests plotted in Fig 4; where the x-axis represents the percentage of CPU + GPU utilisation and the y-axis shows the total mean devaition. This was calculated by averaging the variance between all of the different runs through out the simulation time for all of the agents, then taking the square root to obtain the total mean devaiton.

Note the dotted red line in the plot shows the millimeter (mm) level. Below this level we assume that the level of simulation variance is very low for V&V of CAV simulations purposes, that it is sufficient to assume the tests are deterministic and repeatable. This line can be moved up or down and it is entirely up to verification engineers where they want to define the level below which tests' variance is negligibale and it is suffcient to assume that tests are deterministic, but just for the sake of argument we chose it to be the millimeter level.

First, the tests with no collisions are considered, these represent tests 1, 3 and 5. Tests 1 and 3, which involves vehicles only and vehicles with pedestrians respectively, shows very low simulation variance for stress levels below 50% and as the stress level increases they surpass the red line crossing our defined deterministic level but still being sufficiently low (i.e. with in the centimeter level). This clearly shows how the load on a computer (i.e. stressing CPU and GPU) can have drastic effects on the repeatablitiy of tests. This is more specific to vehicle agents rather than pedestrians since the behaviour of pedestrians is consistent and remarkably below the mm level for all of the different levels of stress shown by test 5, where only pedestrians are involved with no collisions, from which it can be deduced that the mean deviation change with the stress level in test 3 was solely due to the vehicles.

Next, given the pedestrians were showing a very robust and low level of simulation variance, we ran tests for pedestrians colliding (i.e. with paths intersection) for different look ahead distances to exploit whether that would cause any changes, but yet as it can be seen by lines Test IDs' 6,7 and 8 that their performance is consistent for all of the different tests.

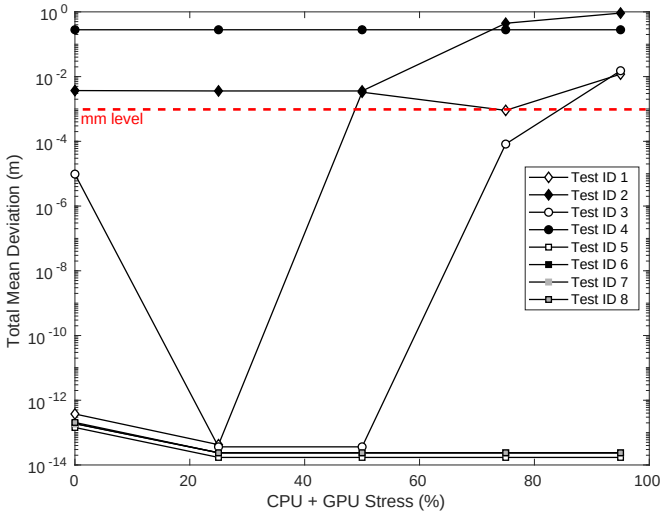


Fig. 4: Summary of total mean deviation for the different tests runs plotted against the different CPU + GPU stresses.

Finally, we look at the other collisions tests, tests 2 and 4 where the tests correspond to a collision between two vehicles and a collision between a vehicle and a pedestrian respectively. We observe that in both of these tests the mean deviation is always above our red line regardless of the stress level, nevertheless still increasing the stress levels shows worsening effects.

Thus, we resorted to plotting the mean deviation over the number of runs of these two tests vs simulation time with stress level %25 chosen arbitrarily (see Fig. 5a and 6a). Doing that we discovered that after collisions occur there is a jump in the mean deviation, which is what causes the averaged mean deviation through the whole simulation time in Fig 4 to always surpass our red line.

Plotting Figs 5b and 6b, which shows the mean deviation pre and post collision for the different stress levels; it can be seen that there is two discrete levels for the pre and post collison devaition from these plots and as the stress increases these two discrete levels start to merge. However, in V&V of CAV simualtions we are normally interested in the point until a collision occur after which the simulation will be terminated. Thus, the post collision effect should not be of much concern in the context of CAVs.

It is important to note, however, that in test 4 a delayed effect of increase in simulation variance was noticed to Car ID 1, which interestingly was not involved in the collision, this was attributed to how the physics engine was programmed to calculate collisions which might cause other obscure behaviours to other agents. The importance of this outcome is that it shows even if an ego vehicle is not involved in a collision, but other agents are, this will still have effects on the simulation variance of the ego vehicle. Thus failing to stay below our red line and the test should be terminated.

In summary, from the tests carried out we can deduce that for a computer of the following specifications, an Alienware Area 51 R5 with an i9 9960X CPU chip, NVIDIA GeForce RTX 2080 graphics card and 64GB of RAM, that all tests carried out using CARLA in UE4 are gauranteed to be deterministic and repeatable (according to our definition by the red line) as long as the CPU + GPU stress does not exceed 50% and tests are terminated as soon as a collision of any type is detected in the simulation.

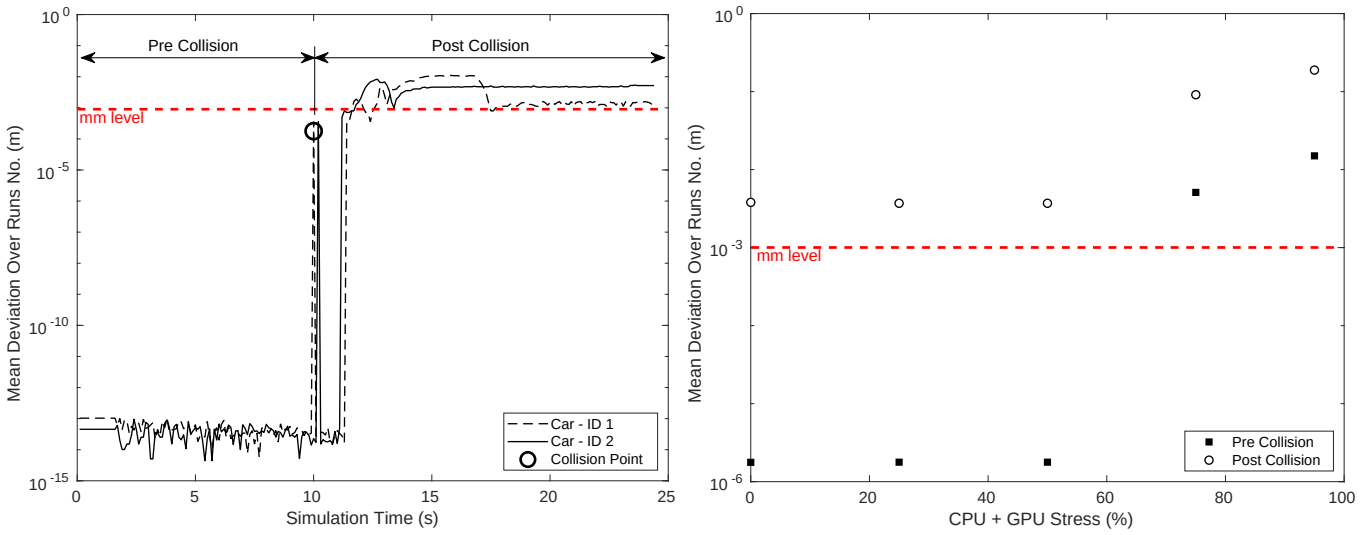
V. Conclusions

We have presented a method by which one can determine the level of non-determinism of a physics engine used in V&V application of CAV simulations. The method consists of five stages, starting with designing the experiment, then suggesting the various settings that can improve or worsen the performance and finally focuses on running the experiment and analysing the output.

A case study is then used to show the implementation of the method introduced, where several tests were setup and run to explore the different interactions between the different agents in a simulator called CARLA. It was found that for the computer used (with specs of—-) the level of non determinism is sufficiently low to assume the simulator to be deterministic as long as the CPU + GPU utilisation is below 50% and that tests are terminated as soon as a collision of any type is detected.

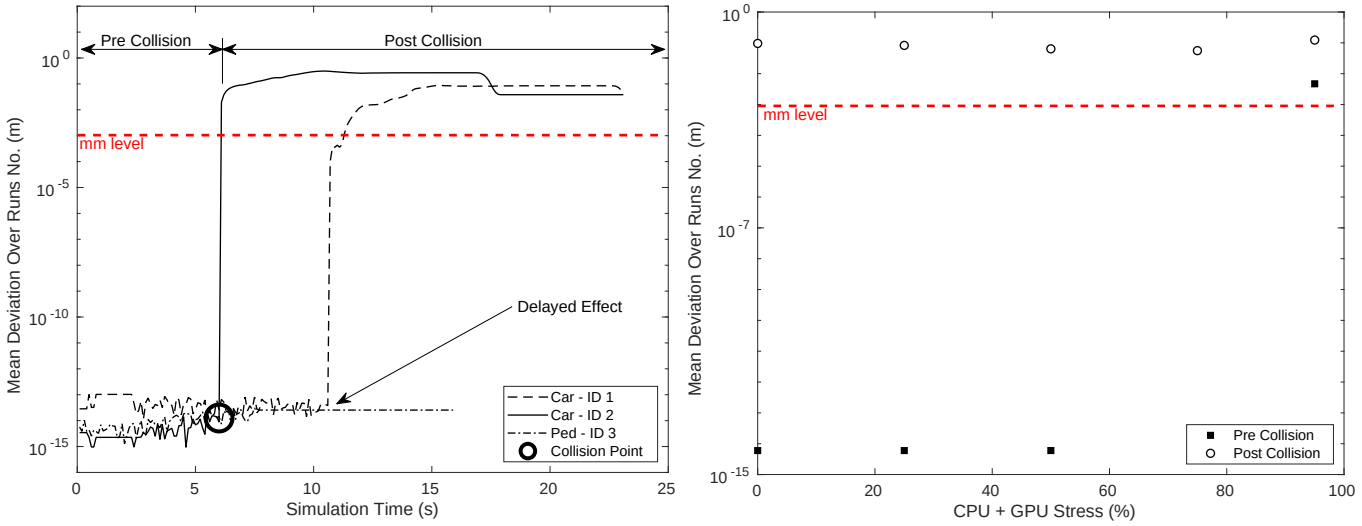
Acknowledgements

This work is part of the ROBOPILOT and Capri projects funded by Innovate UK. **Competition Code:** 1608_CRD1_TRANS_CAV2S1



(a) (b)

Fig. 5: Test ID 2 (a)Mean deviation over runs no. vs simulation time for 25% stress. (b)Pre and post collision mean deviation over runs no. for different CPU + GPU stresses.



(a) (b)

Fig. 6: Test ID 4 (a)Mean deviation over runs no. vs simulation time for 25% stress. (b)Pre and post collision mean deviation over runs no. for different CPU + GPU stresses.

References

[1] J. Bergeron. *Writing testbenches functional verification of HDL models*. Springer Science Business Media, 2012.

[2] F. Codevilla. *CARLA 0.8.2 Driving Benchmark*. http://carla.org/2018/04/23/release-0.8.2/. [Accessed: 26-03-2019].

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. "CARLA An Open Urban Driving Simulator". In: *1st Conference on Robot Learning (CoRL)* (2017).

[4] G. Fiedler. "Fix Your Timestep". In: (2004). Accessed: 2019-12-18.

[5] D. Goldberg. "What Every Computer Scientist Should Know About Floating-Point Arithmetic". In: *ACM Computing Surveys* (1991).

[6] J. Gregory. *Game Engine Architecture*. 2nd ed. CRC Press, 2017. Chap. 7.

[7] C. Hutchison et al. "Robustness Testing of Autonomy Software". In: *International Conference on Software Engineering Software Engineering in Practice Track* (2018).

[8] A. Y. Javaid, W. Sun, and M. Alam. "UAVSim A simulation testbed for unmanned aerial vehicle network cyber security analysis". In: *2013 IEEE Globecom Workshops, GC Wkshps 2013* (2013), pp. 1432–1436.

[9] N. Kalra and S. M. Paddock. “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?” In: *Transportation Research Part A Policy and Practice* 94 (2016), pp. 182–193.

[10] L. Kliemann and P. Sanders. *Algorithm Engineering Selected Results and Surveys*. 2nd ed. Springer, 2016.

[11] P. Koopman and M. Wagner. “Toward a Framework for Highly Automated Vehicle Safety Validation”. In: *SAE Technical Paper Series* 1 (2018), pp. 1–13.

[12] K. Korosec. *Waymo’s self driving cars hit 10 million miles*. https://techcrunch.com/2018/10/10/waymos-self-driving-cars-hit-10-million-miles. Accessed: 2019-11-26. 2019.

[13] L. labs. *Perform GPU, CPU, and I/O stress testing on Linux*. [Accessed: 17-12-2019]. URL: https://lambdalabs.com/blog/perform-gpu-and-cpu-stress-testing-on-linux/.

[14] S. Miglio. *AI in Unreal Engine learning through virtual simulations*. https://www.unrealengine.com/en-US/tech-blog/ai-in-unreal-engine-learning-through-virtual-simulations. [Accessed: 26-03-2019].

[15] J.-M. Muller et al. *Handbook of Floating-Point Arithmetic*. 2nd ed. Springer International Publishing, 2018.

[16] R. Nystrom. *Game Programming Patterns*. 1st ed. Genever Benning, 2011.

[17] “Preliminary Report Highway HWY18MH010”. In: *National Transportation Safety Board* (2018).

[18] U. S. Team. *AI and Behavior Trees*. https://docs.unrealengine.com/en-us/Gameplay/AI. [Accessed: 26-03-2019].

[19] *The Highway Code*. https://www.gov.uk/guidance/the-highway-code. Accessed: 2019-11-25. Department of Transport, UK, 2015.

[20] V. Timonen. *GPU burn*. [Accessed: 17-12-2019]. URL: https://github.com/wilicc/gpu-burn.

[21] Ubuntu. *Ubuntu Manuals*. [Accessed: 17-12-2019]. URL: http://manpages.ubuntu.com/manpages/bionic/man1/stress.1.html.

[22] *UK Road Traffic Act 1988*. http://www.legislation.gov.uk/ukpga/1988/52/contents. Accessed: 2019-10-03.

[23] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer. “Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving”. In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* 2015-Oct (2015), pp. 982–988.

[24] *Vienna Convention on Road Traffic*. https://treaties.un.org. Accessed: 2019-10-03.