Investigating Determinism of Physics Engines for usage in CAV and V&V simulations

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

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

***Abstract*—The industry and certification bodies for connected autonomous vehicles is adopting the use of physics and gaming engines to develop, train, verify, validate and certify the software of these autonomous systems in simulation. It is important for these engines to be deterministic in order for one to carry out these different processes. However, such engines are inherently non-deterministic. We propose a method by which one can identify the region where they can operate in these engines and guarantee that the level of non determinism is sufficiently low that it can be assumed to be deterministic. We also implement this method on a case study and show how we identified the region (encompassing computational utilisation and behaviours in simulation) at which one can guarantee performance to be sufficiently deterministic.**

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

I. Introduction

The evolution and inevitable implementation of connected and autonomous vehicles (CAVs) on roads is a delicate if not utterly fascinating subject. Such systems operate in environments marked by inaccessibility, unexpected weather conditions, or unpredictable behavior by surrounding humans[5]. Public safety is clearly a prime concern for putting autonomous vehicles into play, which means the meticulous and thorough process of developing, training, verifying, validating and certifying the artificial intelligence (AI) and software of the vehicle is a task to be handled with care. It requires lots of flexibility and room to iterate to ensure the algorithms are given the sharpest understanding for maximum performance as well as to ensure that they are hazard-free in order for them to be deployed[8].

There has been a number of catastrophic fatalities with self-driving vehicles due to lack of verification and validation (V&V) of software e.g.[11]. As such, there appears the need for finding ways to V&V the safety of such software, ranging from AI development to cyber-security, while maintaining minimal risks in testing.

Existing vehicle manufacturers use vehicle dynamic simulators to model the physics of each component in a system. Similarly, the use of simulation for traffic modelling is well established. Despite these proven simulation tools, non of them provide all of the required functionalities together to fully simulate CAVs[12].

While traffic-level simulators can model road layouts, they do not require as much realism or detail as CAVs require. On the other side, vehicle dynamics simulators include some of the high fidelity physics functionalities needed for CAV testing, but they only consider the vehicle itself and perhaps the road surface and gradient; they do not model any of the road layout, signs, traffic/street lights, other street furniture or surrounding infrastructure. Most critically, neither types of simulators model sensors such as cameras, lidars or radars[12].

Consequently, one of the major routes CAVs developers are moving towards is using physics engines, these are often games engine development frameworks, since they tick all of the boxes needed to fully simulate CAVs [12]. Robopilot, Capri, Carla, Apollo, Airsim, Udacity etc. are all examples of driving simulator projects that use physics engines to support development, training, and validation of autonomous driving systems software[7].

Test-retest reliability, is a key factor in V&V of software. Tests need to be repeatable in order for one to eliminate bugs. In other words physics engines has to be deterministic, or the level of non-determinism of the engine should be low and clearly defined, for them to be used in the context of V&V of CAV software.

TODO: Remove this paragraph? Determinism of gaming (physics) engines has not attracted much attention by engines' developers nor researchers. This previously was not important since in gaming, determinism is not critical. Performance, on the other hand, is a subject of more interest in that field, where developers want games to work on all platforms and still be fast to provide a positive user experience. On the contrary, determinism becomes vital and performance not when such engines are used in autonomous vehicles' AI development and testing.

This paper aims to investigate how deterministic physics engines are for usage in V&V for CAV simulations; and will

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

cover the list of following points:

- Necessary background on gaming engines used in CAV simulations.
- Review of sources of non-determinism in such engines.
- Provide a method of how the level of non-determinism can be investigated and defined.
- A case study showing the implementation of this methodology in an engine widely used in CAVs simulations (Unreal Engine).

II. Background

This section overviews important information and concepts in physics engines, which are relevant to their usage in CAV simulations. Particularly, on how physics engines operate and the sources of non-determinism in these engines necessary to understand the contribution of this paper.

A. Overview of a Game Loop

The game loop is responsible for the interaction between the math, physics and rendering engines[4]. Figure 1 depicts a very basic diagram showing the flow in a game loop. A traditional game loop is broken up into three distinct phases: processing the inputs, updating the game world (Physics Engine), and generating outputs (Rendering). The first part of

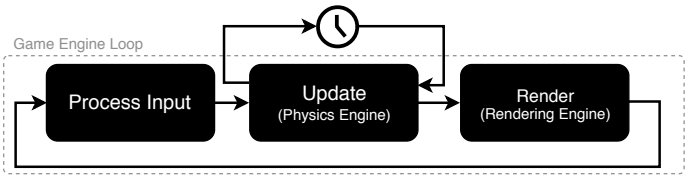


Fig. 1: Game engine loop block diagram

the game loop is to process the user input. Then comes update, which advances the AI and physics engine. Finally the rendering engine draws the game[10]. It is crucial for this paper to understand how the rendering and physics engines operate together. Briefly, the game loop operates as follows:[10] (see Figure 1)

- At the beginning of each tick (frame), 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, one fixed step at a time until it catches up with 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[4][10], 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.

This process allows a game to simulate across a range of hardware, but the rendering will become of jerky quality on slower machines.

These engines update at fixed intervals and render whenever they can, which is not steady. This results in what is so called residual lag[4][10], where the engine is trying to render between two consecutive updates. In this case, the engine will use extrapolation techniques to give a rough estimate of where it thinks the object should be. This often is sufficient for gaming purposes and unnoticeable to the user, in fact it improves the stuttery motion.

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.[3][9]. 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 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.[3]

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[1]. 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[6], which is an algorithm with deterministic behaviour if the environment

is deterministic[8][13]. 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 non-determinism 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

When designing the experiment(s) one needs to bare in mind the following two questions: i) Is the engine deterministic? ii) If not, how does its level of non-determinism vary?

In simulation, actors can either follow a series of trajectory way points or they get given the destination and their AI path plans to that destination. Thus, given that an engine is non-deterministic one needs to set up an experiment that would stress the engine in order to determine how its level of non-determinism vary. This is best done by generating collision callbacks, because that is when engines' do a lot of calculations to determine the response after the collision. Another approach, if AI is used for path planning instead of just following predefined trajectory way points, is to introduce a decision point for the AI.

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 level of non-determinism.

Increasing the physics time-step calculations, which is increasing the number of calculations per unit time, would improve the level of non-determinism 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 non-determinism 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 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

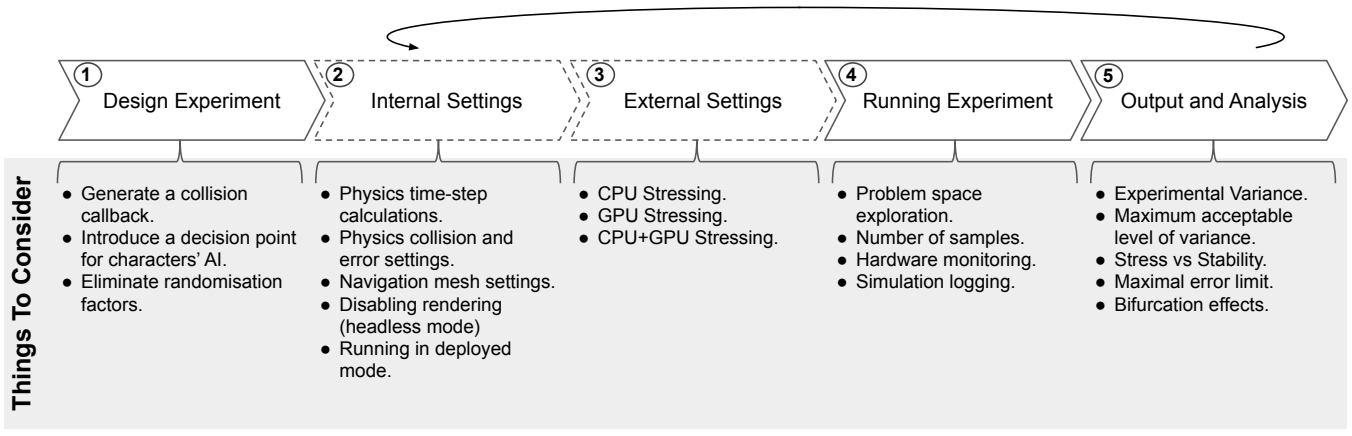


Fig. 2: Shows a flow diagram of the methodology proposed to define the level of non-determinism of a physics engine.

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, however, that during running these tests computers should be left alone not be used for any other purposes apart from running the experiments.

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. Then various plots can be created to identify and engine's level of non-determinsim. Thereon, it would be up to the verification engineers to determine where they want to define the line below which the level of non-determinism 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

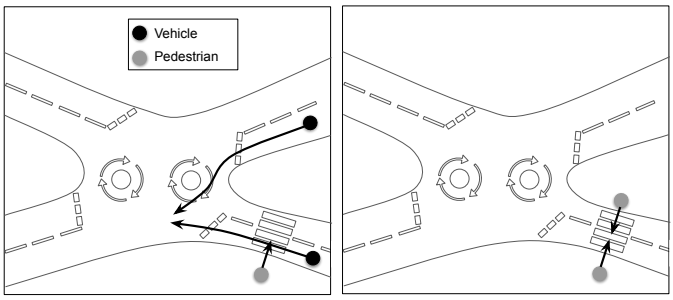
We want to see whether using Unreal Engine 4 (UE4) along with CARLA [2] for V&V of CAV simulations is deterministic or not; if not then what is the level of non-determinism we will operate with, is it acceptable or not, and whether it worsens with stressing GPU and CPU.

Our hypothesis is that since UE4 is a gaming engine it will not be deterministic, however, the level of non-determinism would be very low to acceptable limits that allows verification engineers to use it confidently. 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 level of non-determinism 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; by doing various inspections around the different internal settings of this simulator we did not notice any improvement or worsening in the results. On the other hand the external settings (step 3) did show significant variations on the results as will be discussed in the next section.

Different levels of stress were applied to the computer; using the —— to stress the CPU and —— for the GPU.

In terms of running the experiments (step 4), **i)** the specs of the computer used are —-; **ii)** —— was used to monitor the levels of GPU and CPU stress before and while the experiment was executed; **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. Note the dotted red line in the plot shows the millimeter (mm) level. Below this level we assume that the level of non-determinism is very low for V&V of CAV simulations purposes, that it is sufficient to assume the tests are deterministic. 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 are assumed to be sufficiently 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 to be deterministic 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 non-determinism behaviour, 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.

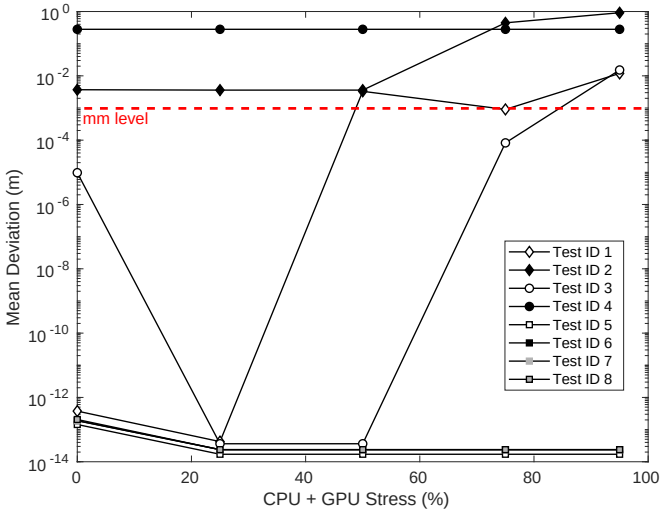
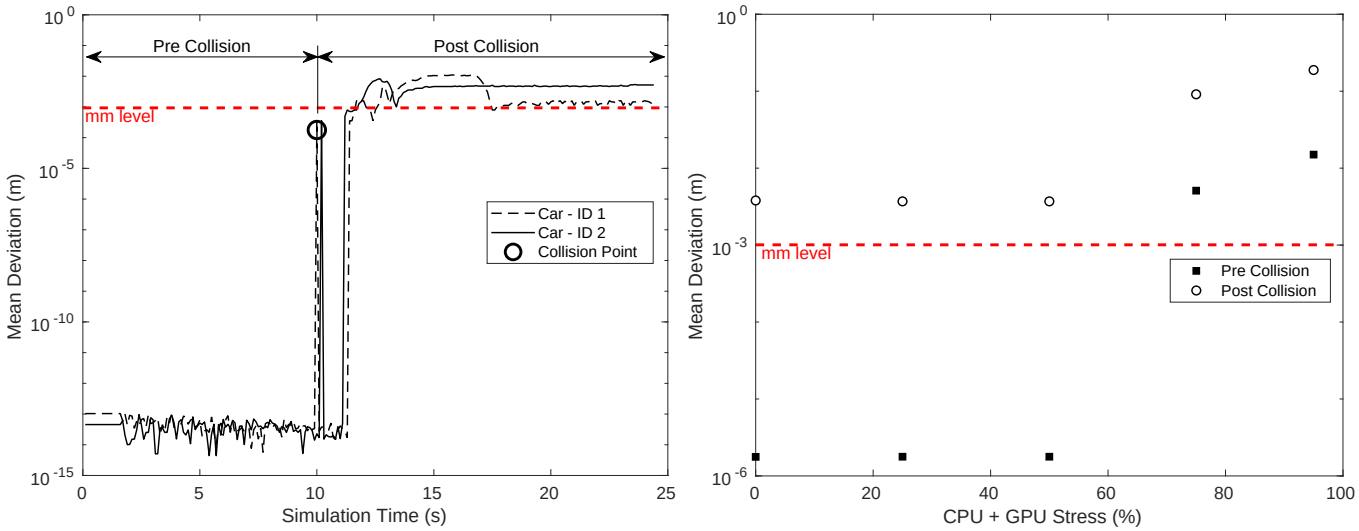


Fig. 4: Summary of mean deviation for the different tests runs plotted against the different CPU+GPU stresses.

Finally, we look at the other collisions tests, tests 2 and 4 where the tests correspond to a collision between two

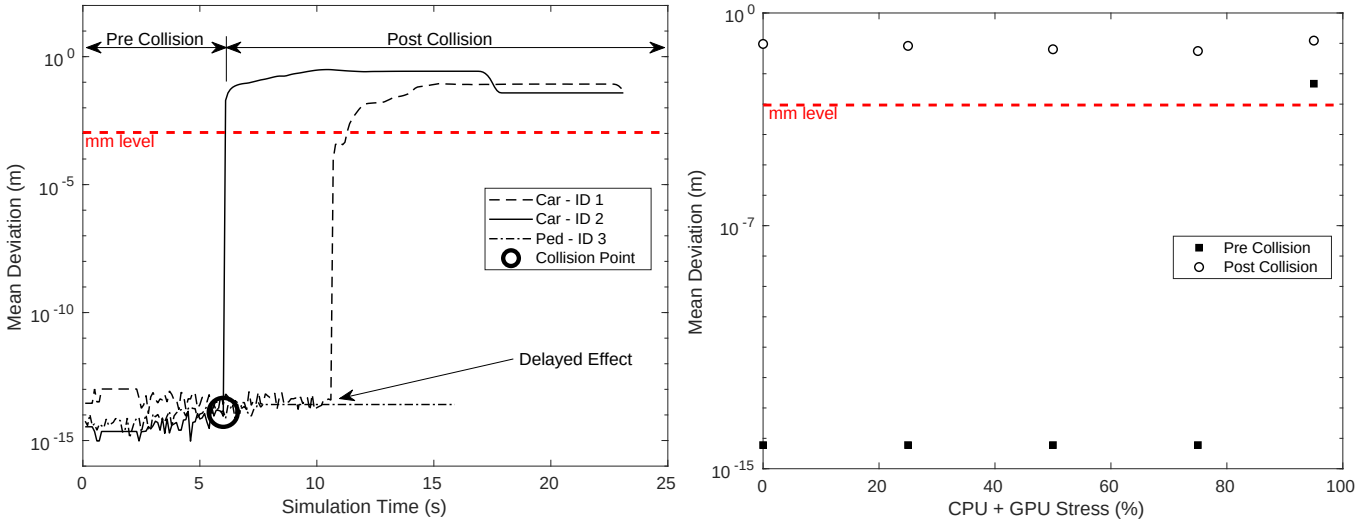
Test ID	Test description (See Fig 3a and 3b)	Collision/Intersection	Collision Type	Look ahead distance (m)	No. of repeats
1	Two vehicles driving	No	N/A	2	1000
2	Two vehicles driving	Yes	Vehicle and Vehicle	2	1000
3	Two vehicles driving and a pedestrian	No	N/A	2	1000
4	Two vehicles driving and a pedestrian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



(a) (b)

Fig. 5: Test ID 2 (a)Mean deviation vs simulation time for 25% stress. (b)Pre and post collision mean deviation at different CPU+GPU stresses.



(a) (b)

Fig. 6: Test ID 4 (a)Mean deviation vs simulation time for 25% stress. (b)Pre and post collision mean deviation at different CPU+GPU stresses.

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 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 the level of non determinism 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 the ego vehicle is not involved in a collision, but other agents are, this will still have effects on the determinism of the ego vehicle. Thus failing to stay below our red line and the test should be terminated.

TODO: We might need to change the y axes nameing betweent he summary plot (fig 4) and the fig 5 and 6, to show that one is mean deviation of all runs and thourgh out simulation time and the other is only between the different runs.

In summary, from the tests carried out we can deduce that for a computer of the following specifications,——..–, 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

References

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

[2] 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).

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

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

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

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

[7] A. Lakrintis. “2018: The Year of the Simulation for Autonomous Vehicles”. In: *Strategy Analytics* (2018). URL: https : / / www . strategyanalytics . com / access - services / automotive / autonomous - vehicles / reports / report - detail / simulation - for - autonomous - vehicles ? Related&langredirect=true.

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

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

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

[11] “Preliminary Report Highway HWY18MH010”. In: *National Transportation Safety Board* (2018). URL: https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf.

[12] Z. Saigol and A. Peters. “Verifying automated driving systems in simulation: framework and challenges”. In: *25th ITS World Congress, Copenhagen* (2018). URL: http : / / zeynsaigol . com / ITS2018VerifyingADSinSimulation.pdf.

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