The upgrade of an advanced driving simulator: vehicle, visuals, sound and software stack

Albert Solernou✉, Anthony J Horrobin, Peter T Woodthorpe, Jorge Garcia de Pedro, Hsuan Chao, Michael R Daly, Natasha Merat

Institute for Transport Studies, University of Leeds, United Kingdom, e-mail: a.solernou@leeds.ac.uk

Abstract - *Fourty years after Daimler-Benz came up with vehicle dome integrated to a 6-DOF hexapod motion system, driving simulation has become a well established tool used in a number of fields, including fundamental research, vehicle development and training. However, time has also shown that simulators age eventually reaching their end of life. This has happened to the University of Leeds Driving Simulator, put together in 2006 by a small academic team. Aiming to build a flexible tool with up-to-date technology that supports industry standards and makes us more productive while creating the best immersive experience, we have upgraded our driving simulator. Everything from the motion platform up was renewed, including the interface vehicle, the projection system, the enclosure and the video capture system. In addition, we wrote a whole new software stack, Simulator5, that covers simulation, 3D scenery and scenario generation. In this paper we present its design, and discuss the technical decisions taken during this endeavour, with the hope that it can help similarly small-sized teams.*

Keywords: *Driving simulator architecture, simulator design, simulator software.*

https://doi.org/10.82157/dsa/2025/24

Introduction

Driving simulation has become an accepted scientific tool to study the multiple aspects of driving, from vehicle design to human factors, from fundamental psychology to civil engineering. Since its launch in 2006, the University of Leeds Driving Simulator (UoLDS) has been used in many studies, successfuly contributing to all these aspects and with its 8 DoF motion system it has remained a world-reference simulator (Bruck, Haycock, and Emadi, 2020). However, over this 18-year period technological advances have changed every aspect of driving and driving simulation. Indeed, the Jaguar S-type interface vehicle is not representative of the latest vehicle capabilities and interfaces, and improvements in hardware, software, graphics and immersive technologies have brought our driving simulator to an effective end-of-life state.

Thus, a business case was put together, seeking to refurbish the motion system by overhauling motors and actuators, and integrating a set of computers, while acquiring, modifying and integrating a new vehicle. Given the success of the driving simulator in attracting research funds, the comprehensive simulation capabilities of the current team, and a forecasted 18 to 24 months of (declining) life-time the Faculty and University boards supported our proposal. Ultimately, we secured a budget of nearly £1M and scheduled a downtime period of one year to undertake the upgrade of our driving simulator.

Initial discussions about the scope of the upgrade started with the choice of the new interface vehicle. As a result of a long-standing series of collaborations with Nissan we were offered a car to be used as the new vehicle interface, for which we are very grateful. An electric vehicle was initially considered as representative of the technologies to come but the required modifications to remove the battery from the underfloor structure in order to decrease its overall weight discouraged us. The Juke and the Qashqai were the two compact vehicles available, amongst which we chose the latter after being one of the top 3 best-selling cars over the last 4 years in the UK, thus making it most suitable for the kind of driver behaviour studies we are interested in.

Improvements on the motion cueing system were also considered. The motion cueing of our driving simulator is provided by a 8-degrees-of-freedom system composed of a x-y table of 5 x 5 metres and a 2500 kg payload hexapod (Jamson, Horrobin, and Auckland, 2007), its physical capabilities described in Tab.1. Introducing a dedicated yaw table would improve the yaw cueing of the motion system, currently limited by the stroke of the actuators. However, it would come with a significant payload increase, requiring an upgrade of the hexapod to the latest and larger VHT eMotion-2700, which in turn would require an upgrade of the x-y table. This was incompatible with the time-scale and budget of the project,

Figure 1: The modified Qashqai with the driver's seat placed at the centre of the motion platform to be used as interface vehicle. Both Brunner units powering the brake pedal and the steering wheel are connected to the driver's side electrical cabinet (shown with its door open); the air compressor can be seen on the front right.

so pre-positioning the platform will still be required to increase the motion cueing envelope.

Finally, we considered the possibility of supporting different interface vehicles, using an approach similar to that used by Jansson, et al., 2014. However, space constraints together with budget limitations forced us to abandon this idea.

On the software side, action was needed too. Simulator3, the in-house written software powering our driving simulator had been designed upon a series of libraries that were about to or had already reached its end of life. Indeed, GTK+ 2.x reached its end of life by the end of 2020 (Clasen, 2020), and OpenSceneGraph was discontinued that same year. Although we had already integrated Unity in SimulatorD, a later evolution of Simulator3Solernou and Horrobin, 2022 we chose to use this opportunity to develop Simulator5, with the aim to refactor and simplify the code, build upon modern libraries, and support industry standards like OMG DDS, ASAM OpenDRIVE and OpenSCENARIO.

Ultimately, the upgrade consisted of renewing everything from the motion platform up, including the interface vehicle and its enclosure, the projection system, every computer, the video capture system, and the whole software stack covering simulation, scenario design, 3D scenery generation and rendering. In the following sections we present its design, and discuss the technical decisions taken during this endeavour.

Methods

Vehicle modifications

Multiple modifications were made to the Qashqai with a three-fold objective: to reduce its weight, to reduce reflections off its bodywork, and to allow reading and setting its controls. As can be seen in Fig.1, shell and chassis were cut to allow for the driver to sit at the centre of the motion platform, every heavy internal component removed, and car exterior was wrapped in matt black vinyl.

Further changes were required to collect and set the multitude of signals that constitute the vehicle controls into a single computer *control*, a Raspberry Pi 4B running a dedicated Codesys project on a 32-bit real-time patched Linux kernel.

Signals from and to the gear lever, gear select dial, parking brake and auto hold, stop / start button, hazard lights button, gear select paddles, left and right stalks, (including indicators, multiple light settings and wipers) wheel buttons (including volume, next, prev, cruise control, etc) and horn (original system still audible), window controls, interior lights, and seat belt and door closed sensors go through a PLC in the front right electrical cabinet and into *control*.

The original accelerator was fully digital and it only required the monitoring of its position which was achieved through a resistive input on the PLC.

The steering wheel was replaced with a 3D printed substitute with touch sensitive pads to track the position of the driver's hands, its output going through the PLC and into *control*.

The steering wheel was connected to a Brunner CLS-P Direct Drive 180 unit via a connecting rod, universal joint and gearing system. The unit can deliver a peak torkque of 180Nm and be driven to a maximum speed of 200 rpm and allows us to read data from the wheel for the participants to drive and to provide realistic force feedback when connected to our software. Similarly, and with the same aim, the original brake pedal was connected to a Brunner CLS-P Linear Motion 1000, with a stroke of 220 mm and a maximum force of 1000 N. Both units were connected to a Brunner Motion Control System, ultimately connected to *control*.

A compressor was bolted to a plate in the engine bay area to pump air into the air conditioning system. Two 0.96" screens were installed on the climate control system substituting its original digital readout screen to display the cabin temperature and the fan speed. The backend unit, which is also connected to *control*, has 3 selectable settings (low, medium and high) changing the intensity of the air blown through the vents.

The driver seat controls were unchanged, providing limbar support, seat tilt, seat pan raising and linear movement on the seat rails.

The dashboard was replaced by a stretched 12.3" monitor and the media unit by a 10.1" touchscreen, each connected to a different Raspberry Pi 4B system: *dashboard* and *infotainment*. Each system runs on a 64-bit real-time patched Linux kernel, the latter connected to the four existing speakers through a JBL amplifier. The wing mirrors were replaced by 8" monitors and mounted on the wing mirror structure by 3D printed covers, both connected to a dedicated rendering unit, *wingmirror*.

An interphone system was also set, consisting of a laptop with a wireless headset located in the control room, that communicates with the driver either through an second wireless headset or through a full hands free-system in the Qashqai cabin. Calls to the headset or to the hands-free system are initiated from the laptop, with the option of recording available. Finally, a full surround sound speaker system was installed to simulate driving and ambient noises.

Table 1: Motion cueing provided by the motion system EMotion-2500-8DOF-500-MK2-XY

System	Motion	Excursion	Speed	Acceleration	90°phase lag bandwidth
Hexapod	surge	−408 / +307 mm	±0.8m/s	±6.5 m/s^2	7.2Hz
	sway	± 318 mm	±0.8m/s	±6.0 m/s^2	7.1Hz
	heave	−300 / +333 mm	±0.6m/s	±6.0 m/s^2	9.5Hz
	roll	−20 / +22°	±40°/s	±300 °/s^2	6.8Hz
	pitch	± 21°	±40°/s	±300 °/s^2	6.7Hz
	yaw	± 23°	±50°/s	±350 °/s^2	9.2Hz
XY Table	surge	± 2500mm	±2.0m/s	±5.0 m/s^2	5.3Hz
	sway	± 2500mm	±3.0m/s	±5.0 m/s^2	5.5Hz

Table 2: Weight comparison (in kg) between the old and the new system, not including the weight of the pedal and steering drives and mounts (88.3 kg on the new system).

System	Dome & Gantry	Car	Projectors	Electrics	Total
Old	845	640	31	61	1577
New	643	667	214	54	1578

Visual system

The projection system is mounted on top of the platform within a relatively small space and thus it needs to be light and to dissipate a relatively low amount of heat, while providing a high resolution. Currently, two different technologies can provide high resolution images: LED and laser projectors. Laser projectors tend to be larger and brighter and they are ideal for large open venues. Instead, LED-based projectors are more energy efficient, and more compact so they were the obvious choice for this setup.

Our existing AC unit did provide a power envelope of 7 kW. An upgrade was considered, since higher dissipation power correlates with larger brightness. However, the resulting AC unit would have been too big and too heavy to be mounted at the bottom of the hexapod. Thus, we chose to mount we chose to mount six Norxe P55 projectors runing at 4096 x 2176@120fps and 2700 ansi lumens, which is a very good resolution and considerable brightness for just 2.64 kW.

The figures in Fig.2 show the arrangement of the projectors and the coverage of the projection around the driver's seat on a digital representation of the Qashqai interior. Even if the coverage is only continuous for the 5 front channels, the rear gaps are unnoticeable from the driver's seat since the wing mirrors are LED displays.

Coverage, mapping and blending was done by STE Antycip using ProjectionTools Mapper3D and Creator, with great accuracy. Finally, while the rendering happens on each image generator (IG) computer, the warping happens directly on the projectors which lowers the workload on the GPUs and allows us to monitor the simulation from the control room more easily.

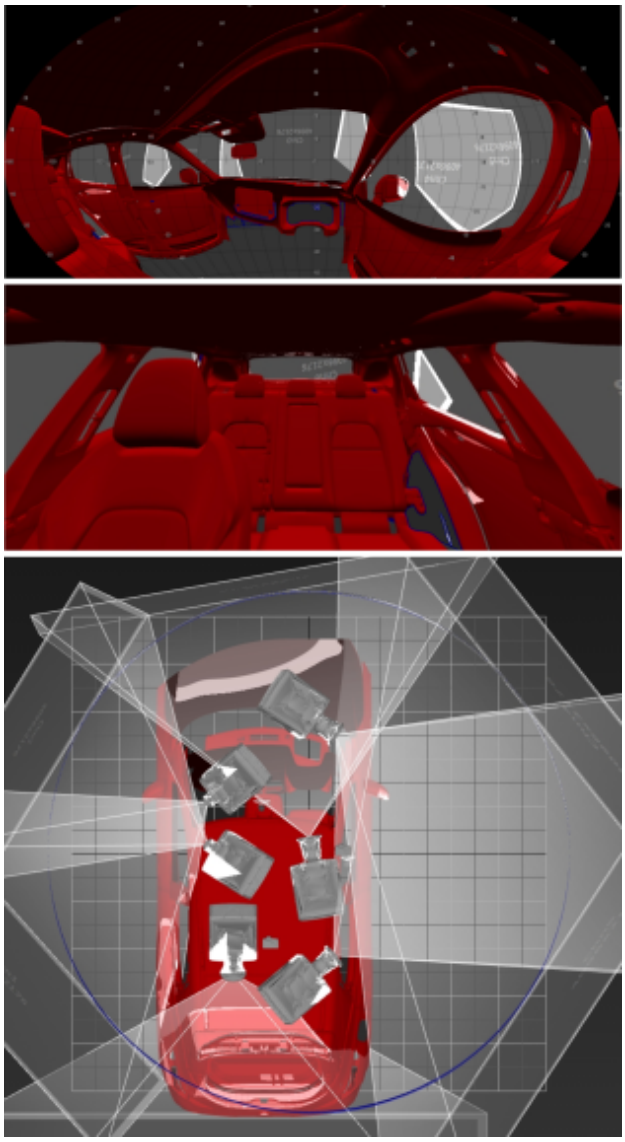

Figure 2: Projection system. Top and middle figures demonstrate full coverage around the driver's seat, while bottom shows the arrangement of the 6 projectors and their projection span.

Enclosure

The enclosure of our driving simulator needs to be large enough to accomodate the vehicle and to be used as the projection wall, resistant to an accidental full-speeed crash against the Enidine heavy duty shock absorbers and as light as possible. As shown

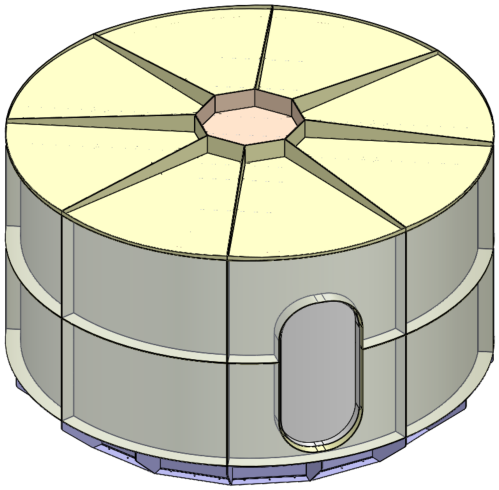

Figure 3: 3D model for the enclosure and projection wall of the driving simulator used to perform the structural finite element analysis to ensure its safety.

in Tab.2, the projector system is more than 180 kg heavier than its previous counterpart and thus we knew from the start that a new and lighter enclosure would be needed capable of withsanding a larger stress.

With no changes on the motion system, the physical requirements for the new enclosure were the same than for its previous encarnation, that is, a natural frequency larger than 15Hz and resistance to a force of 2.5g when hitting the deceleration buffers at maximum speed, decelerating from 3 m/s to stop in 0.122 seconds.

Rather than designing a dome-shaped enclosure, this time we chose it to be cylindrical so that the required projector warping would be on a single dimension, thus easing the rendering effort and resulting in better image quality. Using the structure shown in Fig.3, structural analysis were made considering it built of GRP (Glass-reinforced polymer or fibreglass), with an aluminium frame on its ceiling and masses of 23.5 kg for each of the six projectors.

The enclosure was first envisioned with a 4 m diameter, like the old dome-shaped enclosure. However, it had to be increased to 4.8 m to accommodate the projection which resulted in a heavier wall than anticipated. Indeed, the Nissan is 181 mm taller than the Jaguar, and was mounted on a higher mounts (150 mm vs 70 mm) that are similar ground clearance to the real Qashqai and convenient for ergonomic reasons, but that resulted in shadows casted on a 4 m diameter enclosure.

Moreover, its weight was also increased with the need to use a fire-retardant material. This resulted in an unexpected challenge in which we needed multiple iterations to design a compliant enclosure, resulting in delays and budget concerns. Ultimately, the structural requirements were met using a reduced density of aluminium, panels with variable thickness, bonding the frame to the ceiling, and using extra ribs to the roof to increase its stiffness.

Software architecture

Simulator5 is an inherently distributed piece of software for which a series of `stations` communicate over a messaging system. The messaging system of choice was the OMG Data Distribution Service (DDS), which has become an accepted industry standard for real-time data communication, currently used as the middleware communication layer in robotics (ROS2) and vehicle platforms (AUTOSAR, GVA and NGVA), and by Volkswagen and Audi to test and develop autonomous vehicles (rti.com, 2025).

The 20-year-old DDS open standard describes an easy to integrate publish-subscribe model with quality of service that targets performance efficiency and scalability. A recent study (Bode, et al., 2023) benchmarked four popular DDS implementations and showed that FastDDS had the best end-to-end latency. Released under an Apache Licence and supporting Linux and Windows, it was the ideal choice to build Simulator5 upon, releasing us from maintaining our previous communication bus (Solernou and Horrobin, 2022) and easing future collaborations.

With this messaging system, Simulator5 runs a series of nodes or `stations` distributed over a number of machines:

- `control` reads from the device, publishes `control inputs`, subscribes to `control feedback` and sets the feedback to the device.
- `controlGUI` is an optinal station for debugging purposes, which subscribes to `control inputs` and `control feedback`.
- `dynamics` subcribes to `control inputs` and `controller outputs`, calculates a time step of the vehicle dynamics, sends `motion inputs` to the motion system over a UDP Linux socket, and publishes `vehicle state`.
- `traffic` subscribes to `vehicle state`, `control inputs` and `Turner updates`, runs a time step of the scenario updating the traffic and pedestrian state, and publishes `traffic updates`, `dashboard inputs` and `controller outputs`.
- `dashboard` subscribes to `dashboard inputs` and displays the vehicle information on the vehicle dashboard.
- $\texttt{renderer}_1$ to $\texttt{renderer}_7$, subscribe to `traffic updates`, render the corresponding cameras (one per projector, two in the case of the wing mirrors) and publish `Turner updates`.

Managing each `station` there is an `angel station` spawining and stopping the corresponding `child` upon the request of the `coordinator`. The latter is in charge of starting and stopping the experiments, passing the corresponding details to the `angels`

The stations are written in C++ and have access to a Lua interpreter, while the Lua side has access some to C++ classes and methods exposed through SWIG. This allows Simulator5 to be able to load configuration data from Lua files, thus benefiting the fact that is a lightweight interpreted language. Moreover, it allows us to write scenarios in a programming language that can query the `traffic` station at runtime without the need to re-compile.

Each `renderer` station runs an independent instance of an in-house written Unity project, Turner, which renders a camera as specified through the command-line. Unity exposes its scripting API

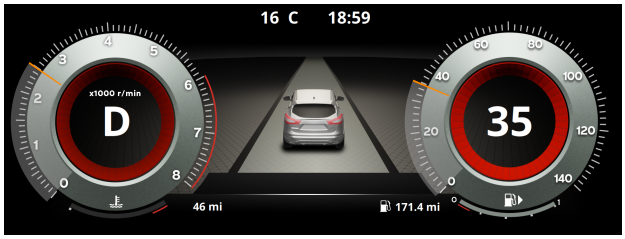

Figure 4: A close replica of the original Qashqai dashboard was built using Qt QML, running on a Raspberry Pi 4B.

through C#, so we gave Turner access to most of Simulator5 through a set of C# bindings automatically generated with SWIG. Amongst other benefits, it grants Turner access to the base stations making it capable of publishing and subscribing to the Simulator5 topics, while having access to many more classes including its road network system.

The choice of the engine game happened in mid 2017, when we started developing our first experiments in VR (Lee, et al., 2019) and Unity and Unreal Engine were the obvious contenders. At the time, Unreal excelled in photorealistic graphics but Unity suited our needs better, with greater support for VR headsets, a larger user community and a much larger Asset Store. The choice has served us well since, and even though Unreal still leads on the rendering side, our past experience in Unity had more value to us as a small team. A rendering alternative that we also considered was VulkanSceneGraph, a modernised version of OpenSceneGraph that builds upon Vulkan rather than OpenGL. Our past experience with OpenSceneGraph and the fact that it was written by the same author made it an interesting option. However, Unity is more flexible and provides further capabilities, so we decided against it.

When windows are needed on other stations, they are created using Qt as it seemed more flexible to us than GTK+ which we dropped. While a station like `controlGUI` that only needs values displayed in rows and columns would have been possible using any of the two GUI toolkits, Qt allowed us to create a replica for the dashboard using its QtQuick framework as can be seen in Fig.4. Without pulling in further library dependencies, Unity would have been the obvious alternative. However, it would have been complicated to run it on the Raspberry Pi that runs the `dashboard` station, since the Unity does not support Linux on ARM architecture and using workarounds such as running it on a web server through WebGL seemed too convoluted.

The file format of choice to store and load road files was ASAM OpenDRIVE. One of the most successful standards to describe road networks, it has an XML based file format describing the geometry, the characteristics and the features of the roads. A number of different free software tools do exist supporting the standard to a varying degree. For instance, libOpenDRIVE is strong on its visualisation, plugins do exist for Blender and Unreal, CARLA has put efforts to support the standard and most significantly esmini provides a road management library as part of its OpenSCENARIO XML implementation. However, their coverage of the (still growing) standard is sometimes unclear and the computational geometry support sometimes limited. Instead, we chose to use the OpenDRIVE Road Network System (ODRoNeS) Solernou, 2023, a simulation-ready C++ library that provides visualisation and computational geometry tools both for roads and lanes. Thus, points can be converted from Cartesian to road coordinates, projected to specific lanes, intersections detected and `conflicts` created with priorities established automatically. Visualisation is possible on a Qt window, and their bindings for Lua, C# and Python allow us to use it in our scenario scripts, in Unity and in Blender which we use to create 3D scenery.

Computers

The `stations` presented in the previous section can run all on the same computer for development purposes, including `control` for which we had a couple of variations: keyboard-mouse and racing wheel. However, they are designed to work on the following set of computers to power the new driving simulator.

`Control` and `dashboard` run on separate Raspberry Pi 4B machines, both with real-time patched kernels. The former needed a 32bit architecture to support the Codesys project that collects and sets the data from the vehicle, while the latter runs on a 64bit one. Another station, `infotainment` is capable to run on a 64-bit third Raspberry Pi powering the touchscreen that substitutes the original media system and will be developed when a study requires it.

`Dynamics`, `traffic` and `coordinator` run on a Rocky 9 machine (a Red Hat 9 clone) with an Intel Core i7-7700K CPU, 32GB of memory and an Nvidia Quadro P5000 16GB GPU also running a real-time kernel. Another machine with the same specifications running Windows has the `renderer` that powers both wing mirrors.

The rest of the six `renderer`s run on separate Windows 11 virtual machines (VMs), these in turn on on three physical machines each with an AMD Threadripper PRO 5955WX, 128 GB of memory and two Nvidia RTX 4090 GPUs. Having three machines instead of six reduced the overall space, power consumption and noise, and creating the VMs using UNRAID, allowed us to allocate half of the physical resources to each VM solving the known problem of dedicating a specific GPU to a Unity process.

The image signal exits the computers through DisplayPort cables, travels 25 m over fibre cables through the cable tray and is converted back to DisplayPort and connected to the corresponding projector and wing mirror monitor.

A dedicated Windows 11 machine with large storage and fast IO is connected to a camera system described later (Fig.5), USB cameras using fibre optic extrensions, OBS Studio used as recording software. Finally, a Windows laptop is used for the interphone system, and the VHT motion system computer running a real-time Linux kernel creates the motion cueing from the inputs received from `dynamics`.

Graphical environment

In a recent work (Pedro, Horrobin, and Solernou, 2024), we presented a methodology to create 3D scenery using procedural approaches that was suitable for the geotypical models needed in driving simulation. Our approach allows us to create a series of

Figure 5: A set of synchronous frames taken with the driving monitoring system, built with five cameras (one using IR light) connected to a computer with fast and large storage system.

road tiles with consistent characteristics that, with a varying size and shape, can be put together to build a specific road layout.

The creation of 3D models was done in Blender, which was chosen for being the most popular open-source tool in computer graphics supporting modeling, rigging, animation, simulation, rendering, compositing and motion tracking. Moreover, starting in Blender 2.92 2021, Blender introduced their Geometry Nodes, a system for modifying the geometry of an object with node-based operations and the fundamental ingredient for procedural generation.

In order to simulate traffic and pedestrians, every road tile needs a corresponding road layout. While this is not difficult for straights and arcs, a computational approach is needed for complex road geometries. As a first attempt we tried to build tiles using the Blender driving scenario creator add-on (Schmitz, 2021), but found a number of limitations. Instead, using the Python bindings in ODRoNeS, we wrote a Blender plugin that allows us to export an OpenDRIVE file out of a 3D scenery with relatively little effort.

Thus, once the 3D model is built and loaded in Blender, the user creates a reference line for each road using a Bezier curve with no textures. For each of the roads, the same details that describe the OpenDRIVE lanes are specified in text boxes, and once the all the roads are included, a click to `Export` results in a OpenDRIVE file that can be loaded by ODRoNeS and used for simulation under Simulator5.

The resulting OpenDRIVE file contains a series of `connection points` in the `userData` section, which is used later to create the full road layout out of the individual tiles tiles. Indeed, a Simulator5 Lua script specifying the requested tiles and their linkage is processed both by the `renderer` stations and by the `traffic` station, the former loading both the 3D models and the xodr files, the latter just creating the logical road network system.

However, two further changes make the resulting xodr files not fully portable: straight and arc geometries in the network are identified and saved as such, but the rest are stored as 3rd order Bezier geometries (and OpenDRIVE only supports straights, arcs, clothoids and paramPoly3). Besides, the information about the junctions is not present. This is not a problem for ODRoNeS, which supports Beziers and has the option to link lanes exhaustively using geometric considerations. Still, writing fully compliant OpenDRIVE files should not be difficult. Indeed, the Bezier curves can be easily translated into paramPoly3 geometries (being both parametric curves of the same order), and the ODRoNeS automatic geometric linkage can be written down into the required `junction` section. We intend to release the add-on to the public when this work is finished.

Vehicle dynamics

Our vehicle dynamics model has not changed significantly since the previous incarnation. The Sayers and Han (1996) model describing the braking and handling behaviour of a four wheeled motor vehicle with independent suspension is still used, the 36 ordinary differential equations solved at every time step using a Runge-Kutta integrator.

The longitudinal model consists of the same composition of models (Salaani and Heydinger, 1998), in which engine torque is derived from a look-up table, transferred to the driveline axes as a function of the relative angular speed between the crankshaft and the gearbox input shaft. Next, this torque enters an automatic gearbox model for which the ratio is selected as a function of throttle position and engine speed, and ultimately goes through a drive ratio providing the tractive torque at each of the front wheels, finally providing rotational speed.

Similarly, the lateral model, uses the same rack and pinion steering system for the power assistance and torque feedback.

Tyre dynamics is governed by the Pajecka's Magic Formula (Pacejka, 2005) with the protection from low speed instabilities by Bernard and Clover (1995).

Motion cueing

The resulting acceleration of the vehicle dynamics model is fed into the motion system. With that, the 8 DoF system presented earlier provides the corresponding motion cueing to the driver using tilt-coordination, in which high frequency response is provided moving the motion platform along the x-y table while sustained acceleration is provided by tilting it. To achieve such purpose, our driving simulator uses the Classical Washout Algorithm (Nahon and Reid, 1990), a relatively simple but well-established approach.

Indeed, tilting rate during tilt-coordination should happen below the perception threshold, that is $3^{\circ}/s^2$ according to Groen and Bles (2004). In general, vehicle accelerations can build faster than this, and thus we use a careful parameterisation of the cueing algorithm which depends on the overall motion envelop.

Because our driving simulator is mainly dedicated to driving behaviour and human factor studies, we can also address the limitations of our motion system envelope (Tab.1) by adjusting the road layout and the

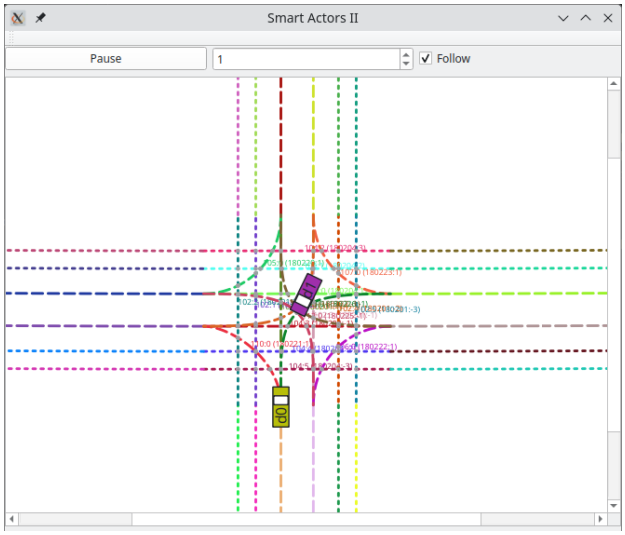

Figure 6: The `traffic` station displays the road layout and the actors in a Qt window. The station can also run in standalone mode, easing the development of scenarios.

scenario development. Thus, such studies are implemented in close collaboration with their research leaders ensuring that their scientific questions are correctly addressed.

Traffic and Scenario

The `traffic` station generates the traffic around the ego vehicle, both creating realistic ambient traffic and scripted scenarios in which drivers are put under controlled conditions.

Once the road network is loaded through ODRoNeS, `traffic` will build routes for the actors (vehicle and pedestrians) as required, given the shortest path between their entry and exit points. Ambient traffic is then assigned to fill in the routes with the required density, while scripted actors can be spawned and driven to fulfill the scenario needs.

Internally, six main tasks are performed sequentially at every time step. First, one or more leaders are assigned to every actor. Second, the scenario ticks. Then, ambient vehicles consider whether they should change lanes, and depending on the model used surrounding actors will be responsive to this potential action. Next, their acceleration for the next time step is set, the equations of motion integrated, cartesian and (possibly multiple) road coordinates updated, and finally the introduction of new actors to the simulation is considered.

Scenarios are defined on Lua scripts, executed by a Lua interpreter in `traffic`. We started implementing OpenSCENARIO XML before changing our minds back to Lua, which makes us more productive because is a programming language. As can be seen in Fig.6, these scripts can be run on a standalone `traffic` station which is very convenient at development time.

Still, we appreciate that using standards ease collaborations, reduce portability costs and ultimately accelerates research. Thus, we borrowed almost every concept from the standard itself to define our scenarios, so that it should be relatively easy to support OpenSCENARIO XML when a future project requests it.

Figure 7: The driving simulator mounted onto the motion system

Figure 8: The modified Nissan Qashqai on the motion platform with the driving simulator in action.

Finally, building upon previous works (Paschalidis, et al., 2021; Solernou, et al., 2021), we implemented several models (some using theoretical games (Kang and Rakha, 2020) some threshold distibution models (Giles, et al., 2019)) to recreate responsive behaviour in merging and lane changing manoeuvres, roundabouts, T-junctions and zebra crossings.

Results and Discussion

When the modified car by EDM Ltd arrived in Leeds, it took about 12 weeks to have all the hardware in place and connected, every issue resolved.

Ultimately, the driving simulator rests on the motion system, as can be seen in Fig.7, the projectors rendering smoothly at 120 Hz, the images projected nicely blending over the projection wall (see Fig.8).

The interior of the vehicle is shown in Fig.9, with three eye-tracking cameras on top of the dashboard, together with two of the driving monitoring cameras.

Finally, a capture from the set of video cameras can be seen in Fig.5 for which there is a context view, a front camera focusing on the face of the participant,

Figure 9: The cabin of the interface vehicle, from which two of the driver monitoring cameras can be seen, together with a second set of cameras part of the eye tracking system.

one at the top of the front left pillar taking the top half of the driver, another one recording the steering wheel area, and a last infrared camera facing the pedals area. Each panel is recorded synchrously at 1920 x 1080 p and 60 Hz.

Conclusions

In this paper, we detailed the work taken to upgrade the University of Leeds Driving Simulator when it was reaching its end-of-life. After 18 years of continuated use, the interface vehicle was not representative of this time, the hardware that it was built with was reaching obsolescence, and the software needed a complete re-write.

Our effort to renew everything from the motion platform up, including the interface vehicle, the projection system, the enclosure, the video capture system, and the whole software stack, resulted in a very flexible scientific tool suitable for driving behaviour studies when we are entering the era of autonomous vehicles.

While we had to make choices restricted by time and budget constraints, we aim to keep enhance its immersive experience over the next years. We hope that this paper serves as inspiration to those small teams thinking of improving or building a new driving simulator.

References

Bernard, J. E. and Clover, C. L., 1995. Tire modeling for low-speed and high-speed calculations. *SAE transactions*, pp. 474–483.

Bode, V., Buettner, D., Preclik, T., Trinitis, C., and Schulz, M., 2023. Systematic analysis of DDS implementations. In: *Proceedings of the 24th International Middleware Conference*, pp. 234–246.

Bruck, L., Haycock, B., and Emadi, A., 2020. A review of driving simulation technology and applications. *IEEE Open Journal of Vehicular Technology*, 2, pp. 1–16.

Clasen, M., 2020. *GTK 4.0, GTK Development Blog*. https://blog.gtk.org/2020/12/16/gtk-4-0.

Giles, O., Markkula, G., Pekkanen, J., Yokota, N., Matsunaga, N., Merat, N., and Daimon, T., 2019. At the Zebra Crossing: Modelling Complex Decision Processes with Variable-Drift Diffusion Models. In: *Proceedings of the 41st Annual Meeting of the Cognitive Science Society*. Cognitive Science Society, pp. 366–372.

Groen, E. L. and Bles, W., 2004. How to use body tilt for the simulation of linear self motion. *Journal of Vestibular Research*, 14(5), pp. 375–385.

Jamson, A. H., Horrobin, A. J., and Auckland, R. A., 2007. Whatever Happened to the LADS? Design and development of the new University of Leeds driving simulator. *DSC 2007 North America*.

Jansson, J., Sandin, J., Augusto, B., Fischer, M., Blissing, B., and Källgren, L., 2014. Design and performance of the VTI Sim IV. In: *Driving simulation conference*, pp. 128–138.

Kang, K. and Rakha, H. A., 2020. A Repeated Game Freeway Lane Changing Model. *Sensors*, 20(6), pp. 1554–1187.

Lee, Y. M., Madigan, R., Garcia, J., Tomlinson, A., Solernou, A., Romano, R., Markkula, G., Merat, N., and Uttley, J., 2019. Understanding the messages conveyed by automated vehicles. In: *Proceedings of the 11th international conference on automotive user interfaces and interactive vehicular applications*, pp. 134–143.

Nahon, M. A. and Reid, L. D., 1990. Simulator motion-drive algorithms-A designer's perspective. *Journal of Guidance, Control, and Dynamics*, 13(2), pp. 356–362.

Pacejka, H., 2005. *Tire and vehicle dynamics*. Elsevier.

Paschalidis, E., Solernou, A., Hasan, M., Markkula, G., Wang, H., and Romano, R., 2021. Estimation and safety validation of a roundabout gap-acceptance model in a simulated environment. In: *Proceedings of the Road Safety and Simulation Conference conference*.

Pedro, J. G. de, Horrobin, A., and Solernou, A., Sept. 18, 2024. Procedural improvement of models and textures of 3D urban scenery: A case study. In: A. Kemeny, J.-R. Chardonnet, F. Colombet, and S. EspiÃ© eds. *Proceedings of the Driving Simulation Conference 2024 Europe VR*. Driving Simulation Association. Strasbourg, France, pp. 227–232. ISBN: 978-2-9573777-5-6. [Accessed Sept. 18, 2024].

rti.com, 2025. Whitepaper: DDS in Autonomous Car Design.

Salaani, M. K. and Heydinger, G. J., 1998. Powertrain and brake modeling of the 1994 Ford Taurus for the National Advanced Driving Simulator. *SAE Transactions*, pp. 1812–1826.

Sayers, M. W. and Han, D., 1996. A generic multibody vehicle model for simulating handling and braking. *Vehicle system dynamics*, 25(S1), pp. 599–613.

Schmitz, J., 2021. *Blender driving scenario creator add-on*. `https://github.com/johschmitz/blender-driving-scenario-creator`. [Accessed Apr. 24, 2025].

Solernou, A and Horrobin, A. J., 2022. University of Leeds: Pedestrian in the loop. In: *Interoperable Simulation*.

Solernou, A., 2023. *ODRoNeS (OpenDRIVE Road Network System)*. `https://github.com/ITSSimulators/ODRoNeS`. [Accessed Apr. 23, 2025].

Solernou, A., Paschalidis, E., Hasan, M., Wang, H., Markkula, G., and Romano, R., 2021. Performance comparison of lane-changing models for merging scenarios in traffic simulation for driving simulators. In: *Proceedings of the Driving Simulation Conferences*. Driving Simulation Association, pp. 109–116.