Dipartimento di Ingegneria e Scienza dell'Informazione

Master in
Computer Science

Thesis

Integration via Networks

Integrating a third-party traffic software into a proprietary driving simulator

Supervisor
Luigi Palopoli

Graduand
Mattia Affabris

Academic year 2017/2018

Never trade luck for skill.

— aviation humour

Thanks

Contents

List of Figures

List of Tables

Abstract

In the view of the overall advancement of ground-based transportation research through simulation, both in an academic and enterprise setting, this document describes the work done on behalf of *ioTech*, *Slightly Mad Studios*, and the *University of Trento* on the integration of a third-party vehicle traffic software in a proprietary game engine. In the final product, the latter acts as the on-screen visualization for the former's underlying traffic simulation. Data is exchanged through a typical network stack, either using UDP or TCP: two networking socket solutions (one based on low-level operating system libraries, the other on an higher level framework) are implemented, analysed, and compared. The results show that, with some performance tradeoffs, pre-existing network frameworks like *0mq* provide interesting advantages in the development process over home-grown solutions.

Introduction

The concept of "simulation" is becoming increasingly relevant in all the branches of ground-based transport: whether developing and testing sensors, using deep learning methodologies for Autonomous Driving (AD), or analysing driver behaviour in arbitrary traffic situations; simulating the required process allows for cheaper, faster, and less rigid production loops. Since both the academic and the enterprise research environments push for similar long-time goals (barring some expected differences), a tool tailor-made for one can, with the appropriate changes, be fruitfully used by the other. As such, collaborations between universities such as *University of Trento* (UNITN) and software companies such as *ioTech* may prove remarkably successful, by promoting an healthy exchange of workforce, technologies, and know-how.

UNITN has been looking into either developing, co-developing, or downright acquiring a fully modular driving simulator, to replace the software currently used in a number of different research projects. To that end, an analysis was made on some of the competitors in the simulators market: by properly weighing the features specifically requested informed decisions may be taken for future collaborations. This analysis and its results are presented in Part I, Chapter 1.

A related opportunity arose for a collaboration with the newly formed *ioTech*, a subsidiary of *Slightly Mad Studios* (SMS). This company the developer, owner, and distributor of *Project Cars*, a consumer-grade driving simulator and one of the leading videogames in the genre, thanks to the team's ten-plus years experience, a focus on realistic physics, and state-of-the-art graphics. *ioTech*'s aim is to branch the videogame's underlying engine in the commercial and enterprise scene, providing high-fidelity visuals (unmatched by the current enterprise solutions) on a modular and extensible simulation core.

An internship period of six months was agreed upon with *ioTech*: the goal was to develop an agnostic, plug-and-play, and network-based integration of a (any) third-party vehicle traffic system into *Project Cars*' game engine. More specifically, the state and actions taken by the agents in a traffic system were to be displayed to the user in the game engine; the user was to drive a car in that game engine, interacting in real time with those same traffic agents. As a proof-of-concept, the traffic software produced by *Vires Simulationstechnologie GmbH*, a fully featured enterprise solution to simulate realistic car traffic flows in arbitrary road topologies, was successfully plugged into the system. The work produced during this collaboration is detailed in Part II: learning and familiarizing with the various software involved (Chapter 2 – Software Descriptions), designing and coding a middleware (Chapter 3 – Middleware Development), and successfully integrating it into the game engine (Chapter 4 – Engine Integration).

Whereas the initial implementation of the middleware leveraged low-level raw Operating System (OS) functions for its network sockets (specifically User Datagram Protocol (UDP)), a second implementation (unrelated from the internship) was also developed, that utilizes the existing middleware framework *0mq*. Details on this framework, its implementation, and the key differences can also be found in Chapter 3 – Middleware Development.

Finally, a performance-driven analysis was done on the middleware, concentrating on two main aspects: its impact on overall system performance (based on a number of metrics, some related to the game engine), and code development concerns. The two socket solutions (OS vs. *0mq*) were compared: ultimately *0mq*, with only a small drop in performance, allows for greater code modularity and expansion, easier maintainability, and less complex code requirements; still, other considerations apply, showing that both approaches are equally viable. Part III is entirely devoted to this analysis.

PART I

Motivation

1 Feasibility Analysis

Keywords — ATTRIBUTE, EASINESS, FEASIBILITY, FEATURE, FRAMEWORK, PRACTICABILITY

This chapter describes the analysis performed on behalf of *University of Trento* (UNITN) on some of the main competitors in the driving simulator market. Informed decision may then be made with regards to which simulation framework might best suit the ongoing research projects: more specifically, the results show the reasons behind the internship collaboration with *Slightly Mad Studios* (SMS).

Section 1.1 – Features lists and describes the set of features required by UNITN to deem the simulator feature-complete. Section 1.2 – Frameworks lists and describes a number of existing simulation frameworks, analysing the main advantages and shortcomings of each. Section 1.3 – Comparison compares the various frameworks by scoring two attributes for each feature, providing an overall weighted score used to numerically compare the different frameworks.

The following analysis was written without taking into account the information and experience collected during the collaboration with *ioTech*.

1.1 Features

> FEATURE: An option, behaviour, or capability that a software must provide for it to be considered feature-complete.

In the scope of feasibility analysis, detailing specific features eases the comparison between the different frameworks.

Features are separated based on their domain, either software or hardware; in each section, the features are listed in order of importance. An associated weight w is also given to every feature[1] (see Table 1.1), as to numerically quantify the relative importance of each: this weight is then used when comparing the various frameworks in Section 1.3 – Comparison. The values were chosen in accordance with UNITN requirements, based on their current and future research projects.

Note that there is no requirement that any of these features be developed from the ground-up, and in-house: a valid alternative might be to integrate pre-existing solutions in the framework.

software					input hardware		output hardware		
model	map	world	traffic	sensor	FF	touch	VR	wheel	seat
1.0	0.8	0.6	0.4	0.4	1.0	0.6	1.0	0.6	0.4

Table 1.1: features weights

1.1.1 Software

Model

The simulator must be capable of accepting a mathematical model as an input, that must then be used by the game engine to properly simulate the behaviour of the car. Any changes made on this model should reflect in the simulation without requiring any modifications of the source-code (plug-and-play paradigm).

[1] $w \in \mathbb{Q} : 0 \leqslant w \leqslant 1$

The purpose of this feature is to allow developing, altering, or fine tuning the behaviour of a car via physical considerations alone. Separating the mathematical description from the coding implementation allows direct alterations to the behaviour of the simulator, without requiring any level of knowledge of the underlying code. Different models of cars, tire conditions, track conditions, and more may then be tested with the simulator without explicit additions or alterations.

Map

The simulator must be capable of recreating arbitrary street maps, or racetracks. The data used to create the map may come from different sources.

- 3D laser scanning is a process that uses laser-based technologies to convert real-life surfaces to clouds of virtual points. The resulting 3D point cloud can then be modified and rendered with millimetric precision. [1] This technique is usually used to faithfully recreate small unchanging scenarios like racetracks, thanks to the high-precision, highly automated, time-saving approach.
- Manually building the scenario with an editor grants total control on its characteristics since the presence, properties, and placement of every element can be freely fine tuned. This is the easiest data source to implement, although faithfully recreating real-life locations is time-consuming, and devising practical non-existing locations requires significant design skills.
- Data about a geographic location (such as roads, buildings, trails, train tracks) can be collected in independent bundles of structured, parsable data. By mapping each entity of the bundle to an asset of the engine the bundle can be programmatically analysed, and the entire location recreated by the engine. Any arbitrary road grid can thus be generated automatically, although the precision and soundness of the final result directly depends on the quality of the mapping algorithm and of the original bundle.

World

The simulator must be capable of rendering and handling world objects, both static and mobile; these objects should interact with the simulated car as realistically as possible, by altering the behaviour of the car (e.g. puddles, curbs), or by interacting with simulated smart sensors (e.g. pedestrians, road lines). A distinction should be made between generic world objects, and specific traffic related entities, to ease the implementation of the traffic feature.

Mobile entities may be controlled by either simple or non-trivial Artificial Intelligence (AI) systems, depending on their role within the simulation.

Traffic

AI traffic must be simulated with an acceptable degree of precision. Although traffic entities are technically world objects, the added complexity governing their behaviour warrants a specific feature. The AI should be formed by a core capable of providing basic behaviour, and an extendible architecture to allow future expansions such as:

- entities following different rules based on their type (e.g. emergency vehicles);
- entities behaving with a degree of unpredictability;
- entities altering their behaviour based on the output of simulated sensors;
- entities dynamically changing their path based on an overall set goal.

Any smart sensors should be able to properly interact with the moving traffic, and ideally with similar simulated sensors on the AI cars.

Sensors

Smart sensors may alter the behaviour of the simulated car (by providing data to its self-drive systems), provide the driver with information about the surrounding environment, or give feedback on the car's state. Generally, two paradigms can be used to simulate smart sensors.

- Use engine Application Programming Interfaces (APIs) (e.g. raycasting) to directly collect the data handled by the sensor: this allows an easier development process, but creates an unrealistic simulation due to the reliance on functions not available to the real sensor.
- Fully recreate the physical phenomena (e.g. reflections) that the real sensor uses to collect its data: this requires in-depth development of both the sensor and the associated physics, but ultimately provides an highly realistic simulation of its inner workings.

Properly simulating smart sensors is a massive undertaking, and existing solutions use remarkably complicated technologies to achieve results. As such, integrating rather than developing may be the only realistic way of providing this feature.

1.1.2 Input Hardware

FF

Force Feedback (FF) effects must be felt on the steering wheel and the pedals. This feedback reflects the forces acting on the car at any given time, and not the force transmitted by the car's smart systems.[1]

FF requires both a software component (that calculates the feedback forces based on the current forces acting on the simulated car), and an hardware component (a motor delivering the force to the steering wheel and the pedals). More advanced software and hardware components allow enhanced mappings, by e.g. providing individual feedback based on the status of each wheel.

Touch

The virtual 3D space should be aware of the position of the hands and fingers of the real-life driver, ideally by camera recognition or Virtual Reality (VR) gloves. The advantages are twofold: systems can be handled with hand gestures without requiring additional input systems (e.g. keyboard, mouse); the driver has a visual representation of the position of his hands even whilst wearing a VR headset.

1.1.3 Output Hardware

VR

The simulation must be compatible with VR headsets. Full integration requires a number of VR-specific options such as head positioning, scaling, supersampling, 3D cockpits, eye distance, and more. VR enhances the experience primarily by providing depth perception and complete immersion (currently at the cost of narrow Field of View (FOV)), limiting the requirement for multiple screens or expensive wide-angle projector setups.

Wheel

The steering wheel must be able to provide steering or counter-steering force based on any input given by the car's smart system. These forces are separate from the typical simulation FF, since they do not represent the physical state of the car (what in real-life is controlled by mechanics alone), but rather represent the output of some smart system (a deliberate software-controlled motor output on the steering wheel).

For simulating reasons the steering wheel already has a motor used for FF purposes: that same motor may also be used by the sensors. Having two separate dedicated motors would enhance the simulation, by allowing directly counteracting forces to be felt by the driver.

[1] In a real-life car there exist no FF motor in the steering wheel; the forces felt are rather of purely mechanical origin.

Seat

FF effects must be felt on the simulator's setup seating. This feedback should reflect the forces acting on the car at any given time, in a similar manner to FF.

A number of technologies make this possible, normally cantered around a motor that acts on one or more moving elements, matching as close as possible the forces applied to the car with forces felt by the real-life driver.

- A moving platform with an immobile seat allows limited 2-axis or 3-axis movement: simulated g-forces are reproduced using gravity.
- A moving seat allows less limited 3-axis movement that a moving platform: similarly, gravity is leveraged to reproduce g-forces.
- Inflatable sacks can be worn or placed on the seat: g-forces are then reproduced by filling the sacks with air, providing pressure on the body of the driver. [2]
- Specific signals may be translated into matching low frequency sounds, that then drive a weighted motor: this may give remarkably precise and distinct rumbling effects. [3]

1.2 Frameworks

> FRAMEWORK: A software abstraction that allows additional functionalities to alter and/or extend its original behaviour.

The simulation software listed here can all be considered frameworks due to their general capability of being modified (at different depth levels) by the user.

OpenDS

> *The flexible open source driving simulation.* [4]

OpenDS is an open-source driving simulation, modifiable under the GNU general public license. The core architecture provides a basic rendering engine (capable of low-quality graphics), positional audio support, and notably a `Java` fork of the *Bullet* real-time physics engine [5]: the latter provides *"mesh-accurate collision shapes, experience of acceleration, friction, torque, gravity and centrifugal forces"* [6]. Other relevant features include:

- automated scenery generation;
- a 3D editor;
- scenery-aware AI cars and pedestrians;
- a traffic lights system;
- real-time data export;
- *Oculus Rift* VR headset support;
- motion chair support.

OpenDS also provides high-level analysis and scenery creation tools, geared towards evaluating the performance of a driver in an arbitrary driving scenario. It thus lacks an accurate simulation of both the behaviour of the car, and the physical interactions it may have with the surrounding environment.

Six different purchase tiers are available, ranging from a 99 $ "Student" option to a 3150 $ "Pro Complete CE" option: notably, only the latter pricier tier includes the automated scenery generation tools. As such, the analysis of this framework in Section 1.3 – Comparison is based on the "Pro Complete CE" tier.

VDrift

> *VDrift is a cross-platform, open source driving simulation made with drift racing in mind.* [7]

VDrift is centred around simulation-grade driving physics. Unlike *OpenDS* the focus is on racing and proper physics simulation, with little to no effort put into systems and analysis. Features of interest are limited:

- numerous tracks and vehicles
- limited racing AI
- limited FF support

Additionally, the last release dates back to 20/10/2014, with slow development times, and a small number of contributors.

Unity

> *The world's favourite game engine.* [8]

Unity is a powerful and widespread game engine, capable of rendering 2D and 3D environments alike; a host of modules allows complete integration with existing technologies such as VR.

Being a mature and fully customizable game engine, the main advantage is the possibility of implementing every feature without having to rely on the support of 3rd-party APIs. The lack of high-level APIs specific to driving simulations though forces a longer and more involved development process, since most features must be coded from the ground up.

Three monthly subscription tiers are available, ranging from a free "Personal" option to a 125 $ "Pro" option; the latter allows negotiating source code access, and obtaining advanced support. Further analysis of this framework is based on the "Plus" tier.

Panthera

> *Panthera uses high-end physics and an excellent rendering engine. It contains controllers for motion platforms, steering feedback, pedals, dashboard, audio etc. as well as a scripting engine to define and customize the simulation.* [9]

Panthera is a full-stack, modular software suite by *Cruden*; initially tailored for in-house Hardware In the Loop (HIL) and Software In the Loop (SIL) solutions, it has been since ported as a stand-alone desktop simulator. It *"uses high-end physics and* [...] *It contains controllers for motion platforms, steering feedback, pedals, dashboard, audio etc. as well as a scripting engine to define and customize the simulation."* [9] More specifically it includes:

- a generic interface that allows custom built simulation models
- interfaces for scenarios and traffic
- extensive output hardware customization
- a dedicated tire simulation
- HIL and SIL support

Visuals are unmatched by any other professional engineering simulation available. It is unclear whether VR support is available or not: a reference is made about *"Helmet Mounted Displays (HMDs) [being] a factor in our next Panthera Free simulator software upgrade"* [10], but no mention can be found in the features lists; it is apparent though that VR can be supported by the rendering engine.

No pricing listings are publicly available; additionally the proposed free software suite lacks, among other features, the FF module and multiple monitor or projection support.

Project Cars

> *Project CARS 2 is the second installment in the best-selling and critically-acclaimed Project CARS motorsport franchise developed by Slightly Mad Studios.* [11]

Project Cars is a consumer-grade driving simulation geared towards racing and hyper-realistic car handling. Being a direct port of a consumer-grade software, both visual and audio rendering are state of the art: as such, the immersion potential is the highest of all analysed frameworks, especially when considering VR support.

Although the current available engine is closed-source and not modifiable, the software company SMS is in the process of developing a professional and enterprise version of their simulation and rendering engine. All details regarding functionalities, APIs, and cost are unknown at the time: all further analysis is thus done based on the capabilities of the current consumer product.

Project Cars 2 has full support for most external peripherals, including VR, FF steering wheels, and moving seats. A strong focus is given to tire-simulation, driven by a technology named *Over the Limit*: the developers were helped by both *Pirelli* and a number of real-life drivers. [12] AI and available tracks are limited to racing only, with no pre-existing support to city driving.

SCANeR

> *SCANeR studio is a complete software tool meeting all the challenges of driving simulation* [...] *it is a genuine evolving simulation platform, extendable and open, answering the needs of researchers and engineers.* [13]

SCANeR is a modular simulation framework that provides dedicated Graphical User Interfaces (GUIs) to create, alter, or configure its different parts; visual and audio rendering are basic, following a "function before form" paradigm. Relevant features are:

- dynamic traffic and pedestrians
- tunable vehicle model
- `XML`-based scenario creation

Available information of the full set of capabilities is limited; the strong focus on the GUI of the sales pitch though seems to imply limited expandability out of the scope of what the original developers intended. Development might be forced to occur not on the source code of the software itself, but rather on a limited set of APIs or on separate parsers and configurators.

No pricing is publicly displayed.

STISIM Drive

> *Whether for occupational therapy assessment and rehabilitation, driver training, or driver research, the versatile STISIM Drive brings you a new level of flexibility, ease and analytical insight.* [14]

STISIM Drive is, similarly to *OpenDS*, a driving simulation focused on driver training, research and assessment, and scenario conception:

The possibilities of meaningfully extending the framework seem limited by the scenario-centred list [15] of capabilities, that contains no mentions to peripherals, physics simulation and interaction, on-board systems, or smart sensors:

- proprietary scenario definition language
- modular programming
- realistic roadway environments
- data collection and analysis

Minor importance is clearly given to the realism of the physical simulation, the quality of the graphics and audio, and the integration with input and output peripherals.

No pricing is publicly listed, although numerous pre-built solutions are proposed by the developing company, in the form of bundles of software and hardware.

1.3 Comparison

1.3.1 Attributes

For each feature two attributes are scored, in order to objectively assess the capabilities of a framework:

- PRACTICABILITY: The capability of a given framework to implement a given feature.
- EASINESS: The effort required to implement a given feature on a given framework.

For example, a framework with a large number of APIs but limited access to the source code would have an high easiness score but a low practicability score.

1.3.2 Scoring

Scoring is based on both objective and subjective observations on the capabilities and tools of the analysed framework. A score between 1 and 5 is given to the practicability (s_p) and easiness (s_e) attribute of each feature: Table 1.2 lists these scores.

	model		map		world		traffic		sensors	
	s_p	s_e	s_p	s_e	s_p	s_e	s_p	s_e	s_p	s_e
OpenDS	2	1	4	4	4	4	5	4	3	2
VDrift	2	1	3	2	3	3	2	2	2	1
Unity	5	1	5	2	5	3	5	2	5	1
Panthera	5	3	5	3	4	3	5	3	4	2
Project Cars	3	2	3	2	4	4	3	2	4	2
SCANeR	2	1	4	4	4	4	5	5	2	2
STISIM Drive	2	1	3	3	3	3	5	4	2	2

	FF		touch		VR		wheel		seat	
	s_p	s_e	s_p	s_e	s_p	s_e	s_p	s_e	s_p	s_e
OpenDS	3	3	2	2	5	5	3	2	5	4
VDrift	3	3	2	2	2	1	2	2	2	2
Unity	5	3	5	2	5	4	5	3	5	3
Panthera	5	4	4	2	3	3	4	3	5	4
Project Cars	5	4	4	2	5	5	5	4	5	4
SCANeR	3	2	2	2	1	1	2	2	2	2
STISIM Drive	3	3	2	2	2	2	2	2	3	2

Table 1.2: frameworks scoring

1.3.3 Analysis

To aid in the ordering of frameworks, a third attribute (s_f) is used:

FEASIBILITY: A weighted score calculated from practicability and difficulty.

Weighing allows importance to be shifted between practicability and easiness.[1] Additionally, an overall score (s) is given to each framework by applying a weighted mean to the feasibility values[2]: see Section 1.1 – Features for the weights of the features.

Three different rankings are provided, each in order of overall score (from the best to the worst):

- in Table 1.3 equal weight is given to practicability (0.5) and easiness (0.5);
- in Table 1.4 larger importance is given to practicability (0.7) than easiness (0.3), highlighting those frameworks that would best provide a complete implementation of the simulator;
- in Table 1.5 larger importance is instead given to easiness (0.7) than practicability (0.3), highlighting those frameworks that would best provide easier to use tools and faster development times.

As expected, *Unity* scores the highest (the only framework to score above 4.0 across all analyses) when the feasibility score is calculated with a practicability bias: this reflects its highly adaptive and extensible nature, it being a generic game engine and not a pre-existing driving simulation engine. At the same time though, *Unity* scores poorly (3.2) when importance is given to easiness, due to the lack of high-level APIs and the need to build the simulator from the ground up.

At 4.0 and 3.8 respectively *Panthera* and *Project Cars* are close seconds in the practicability bias classification, also scoring the highest (3.5 both) when easiness is concerned: the availability of high-level APIs, along with the pre-existing implementation of a number of highly weighted features make for an ideal combination.

Panthera, *Project Cars*, and *Unity* are the highest scorers 3.7 when no bias is given in the feasibility calculation, further proving their edge over the other options.

STISIM Drive scores low (2.5 or below) on all classifications, due to its lack of focus on aspects relevant to the simulator; *SCANeR* scores identically, although it does so due to code access concerns. *VDrift* on the other hand suffers its open-source nature (2.2 or below), with slow development times and the lack of a supporting company further undermining its possibilities.

Both *Project Cars* and *Panthera* appear to be the ideal solutions: the choice of the former though depends on the access level to the engine, as determined by the developers SMS; the latter is a more mature and supported professional option, although with less potential capabilities. *Unity* becomes the third choice only if enough time and work-hours are available to develop the simulator from the outright basis: everything from the physics engine to the peripherals support will have to be coded in their entirety. *OpenDS* is thus the realistic third choice, in case *Unity* is deemed a too time-intensive solution: it scores right behind *Project Cars* and *Unity* in the non-biased calculations, and is on par with *Unity* w.r.t. easiness. It provides high-level APIs, existing scenario building options, and basic peripherals integration; the immersion potentiality is lower though, due to the basic graphics and audio rendering.

1.4 Recap

In Section 1.1 – Features ten different features were presented, listing all those requirements the simulator must adhere to. In Section 1.2 – Frameworks five frameworks were analysed. In Section 1.3 – Comparison the value of each framework were quantified through weighting and means, to generate three classifications: from those, a general analysis was performed and PROJECT CARS or PANTHERA were chosen as the best overall framework for the development of the simulator, with UNITY and OPENDS as the fall-back solutions.

[1] See Appendix A.1 – Weighted Feasibility for the mathematical description of the function.
[2] See Appendix A.2 – Overall Score for the mathematical description of the function.

	tot	mod.	map	wor.	tra.	sen.	FF	tou.	VR	whe.	sea.
	s_f	s_f	s_f	s_f	s_f	s_f	s_f	s_f	s_f	s_f	$\boldsymbol{s}$
Panthera	**3.7**	4.0	4.0	3.5	4.0	3.0	4.5	3.0	3.0	3.5	4.5
Project Cars	**3.7**	2.5	2.5	4.0	2.5	3.0	4.5	3.0	5.0	4.5	4.5
Unity	**3.7**	3.0	3.5	4.0	3.5	3.0	4.0	3.5	4.5	4.0	4.0
OpenDS	**3.3**	1.5	4.0	4.0	4.5	2.5	3.0	2.0	5.0	2.5	4.5
STISIM Drive	**2.5**	1.5	3.0	3.0	4.5	2.0	3.0	2.0	2.0	2.0	2.5
SCANeR	**2.4**	1.5	4.0	4.0	5.0	2.0	2.5	2.0	1.0	2.0	2.0
VDrift	**2.1**	1.5	2.5	3.0	2.0	1.5	3.0	2.0	1.5	2.0	2.0

Table 1.3: overall frameworks ranking (no bias)

	tot	mod.	map	wor.	tra.	sen.	FF	tou.	VR	whe.	sea.
	s_f	s_f	s_f	s_f	s_f	s_f	s_f	s_f	s_f	s_f	$\boldsymbol{s}$
Unity	**4.2**	3.8	4.1	4.4	4.1	3.8	4.4	4.1	4.7	4.4	4.4
Panthera	**4.0**	4.4	4.4	3.7	4.4	3.4	4.7	3.4	3.0	3.7	4.7
Project Cars	**3.8**	2.7	2.7	4.0	2.7	3.4	4.7	3.4	5.0	4.7	4.7
OpenDS	**3.4**	1.7	4.0	4.0	4.7	2.7	3.0	2.0	5.0	2.7	4.7
STISIM Drive	**2.5**	1.7	3.0	3.0	4.7	2.0	3.0	2.0	2.0	2.0	2.7
SCANeR	**2.5**	1.7	4.0	4.0	5.0	2.0	2.7	2.0	1.0	2.0	2.0
VDrift	**2.2**	1.7	2.7	3.0	2.0	1.7	3.0	2.0	1.7	2.0	2.0

Table 1.4: overall frameworks ranking (practicability bias)

	tot	mod.	map	wor.	tra.	sen.	FF	tou.	VR	whe.	sea.
	s_f	s_f	s_f	s_f	s_f	s_f	s_f	s_f	s_f	s_f	$\boldsymbol{s}$
Panthera	**3.5**	3.6	3.6	3.3	3.6	2.6	4.3	2.6	3.0	3.3	4.3
Project Cars	**3.5**	2.3	2.3	4.0	2.3	2.6	4.3	2.6	5.0	4.3	4.3
Unity	**3.2**	2.2	2.9	3.6	2.9	2.2	3.6	2.9	4.3	3.6	3.6
OpenDS	**3.2**	1.3	4.0	4.0	4.3	2.3	3.0	2.0	5.0	2.3	4.3
STISIM Drive	**2.4**	1.3	3.0	3.0	4.3	2.0	3.0	2.0	2.0	2.0	2.3
SCANeR	**2.4**	1.3	4.0	4.0	5.0	2.0	2.3	2.0	1.0	2.0	2.0
VDrift	**2.0**	1.3	2.3	3.0	2.0	1.3	3.0	2.0	1.3	2.0	2.0

Table 1.5: overall frameworks ranking (easiness bias)

PART II

Development

2 Software Descriptions

2.1 Madness Engine

2.2 VTD

2.3 0mq

2.4 Recap

3 Middleware Development

3.1 Concept

3.2 Structure

3.3 Networking

3.3.1 OS sockets

3.3.2 0mq

3.4 VTD implementation

3.5 Testing

3.6 Recap

4 Engine Integration

4.1 Approach

4.2 Modularization

4.3 Refinement

4.4 Limitations

4.5 Recap

5 Further Work

5.1 Integration

5.2 Recap

PART III

Analysis

6 Performance Analysis

6.1 Framerate

6.2 Network

6.3 Hardware

6.4 Bridge

6.5 Outcome

6.6 Recap

7 Code Analysis

7.1 Functionalities

7.2 Coding ease

7.3 Expandability

7.4 Recap

PART IV

Appendix

1 Math

A.1 Weighted Feasibility

$$f : \mathbb{N} \mapsto \mathbb{Q}$$

$$\forall w_p, w_e \in \mathbb{Q}^+ : w_p + w_e = 1$$
$$\forall s_p, s_e \in \mathbb{N} : 1 \leqslant s_p, s_e \leqslant 5$$

$$s_f = (s_p \cdot w_p) + (s_e \cdot w_e)$$

A.2 Overall Score

$$f : \mathbb{Q} \mapsto \mathbb{Q}$$

$$\forall w_i \in \mathbb{Q} : 0 \leqslant w_i \leqslant 1$$
$$\forall s_{f_i} \in \mathbb{Q} : 0 \leqslant s_{f_i} \leqslant 5$$

$$s = \frac{\sum_{i=1}^{n} (s_{f_i} \cdot w_i)}{\sum_{i=1}^{n} w_i}$$

2 Code

PART V

References

Bibliography

[1] iRacing track technology. http://www.iracing.com/cars-and-tracks/track-technology/ (last access 21/09/2017).

[2] Gametrix jetseats. https://www.gametrix.eu/ (last access 17/07/2018).

[3] buttkicker transducers. https://thebuttkicker.com/ (last access 17/07/2018).

[4] OpenDS. https://www.opends.eu/home (last access 20/09/2017).

[5] Bullet. http://bulletphysics.org/wordpress/ (last access 22/09/2017).

[6] OpenDS features. https://www.opends.eu/software/features (last access 22/09/2017).

[7] VDrift. https://vdrift.net/ (last access 20/09/2017).

[8] Unity. https://unity3d.com/ (last access 20/09/2017).

[9] Panthera. http://www.cruden.com/#innerShape483__Shape483_Shape1165711 (last access 01/10/2017).

[10] Panthera virtual reality. http://www.cruden.com/virtual-reality-article.html (last access 01/10/2017).

[11] Project Cars. https://www.projectcarsgame.com/ (last access 20/09/2017).

[12] Project Cars over the limit. https://www.projectcarsgame.com/project-cars-2-faq (last access 25/09/2017).

[13] SCANeR. http://stisimdrive.com/ (last access 29/09/2017).

[14] STISIM Drive. http://stisimdrive.com/ (last access 25/09/2017).

[15] STISIM Drive features. http://stisimdrive.com/research/ (last access 25/09/2017).

[16] OpenDS tiers. https://www.opends.eu/ (last access 22/09/2017).