

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Collaborative Driving in Virtual Reality

Nuno Pereira



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Ricardo Baptista

Co-Supervisor: Carlos Campos

January 31, 2021

Collaborative Driving in Virtual Reality

Nuno Pereira

Mestrado Integrado em Engenharia Informática e Computação

January 31, 2021

Resumo

Aumentar a segurança nas estradas é uma necessidade urgente visto que o número de condutores está a aumentar cada vez mais, assim como a necessidade de cada vez as viagens sejam mais longas, quer na rotina diária porque as pessoas trabalham mais longe de casa, quer nas viagens de lazer em que as pessoas cada vez mais têm sede de conhecer outros destinos. De forma que esse objetivo seja atingido é importante estudar as características e o comportamento dos condutores. Muitas vezes, os casos a ser estudados são perigosos ou acarretam demasiados custos. Como tal, para que estas situações possam ser testadas e adequadamente avaliadas são usados simuladores automóveis, onde múltiplos fatores podem ser facilmente isolados devido à possibilidade de repetir os testes as vezes que forem necessárias, sem nunca comprometer as condições de segurança mantendo os custos baixos. Simuladores de condução são sistemas de realidade virtual que permitem reproduzir cenários do mundo real num ambiente controlado. O que se vai melhorar está localizado no laboratório de análise de tráfego do Departamento de Engenharia Civil da Faculdade de Engenharia da Universidade do Porto e é intitulado de DriS. A partilha dum ambiente rodoviário colaborativo entre diferentes arquiteturas de sistemas foi o principal problema em que este trabalho se focou, noutras palavras, com este trabalho foi acrescida a possibilidade de várias pessoas, conduzirem no mesmo ambiente partilhado e interagindo uma com a outra usando diferentes tipos de tecnologia. A solução desenvolvida é baseada no princípio master/slave e permite que duas máquinas distintas corram a simulação no mesmo ambiente, comunicando uma com a outra. O computador que corre como master envia a posição e orientação de todos os carros existentes no ambiente de simulação, incluindo o próprio que o master controla, para todas as máquinas que estejam a correr como slave o mesmo ambiente rodoviário nesse momento. Por sua vez, o computador que esteja a executar o DriS como slave também é capaz de enviar a posição e orientação de apenas o próprio carro que controla, para a máquina que esteja a funcionar como master, para que essa máquina possa distribuir as posições e orientações desse carro por todas as máquinas que estejam a correr como slaves, na próxima iteração, assegurando desta forma o sincronismo entre todas as instâncias do simulador de condução DriS existentes no momento. O cenário experimental idealizado é constituído por um condutor, usando para tal a estação de condução principal representada pelo Volvo 440, localizado na sala da simulação, e por um outro condutor usando a segunda estação de condução, caracterizada pelo conjunto de volante e pedais Logitech, montada com o propósito de validar o projeto, conduzindo simultaneamente no mesmo ambiente rodoviário. É expectável que os estudos realizados no simulador a partir de agora tenham um alcance superior e uma gama mais vasta de possíveis aplicações visto que foi conseguido um ambiente mais próximo da realidade, em que é também possível estudar a interação entre os diferentes condutores.

Palavras-chave: Condução colaborativa, Ambientes partilhados , Simulação de condução, Ambientes imersivos

Email: nuno1997pereira@hotmail.com

Abstract

Increase road safety is an urgent need as the number of drivers is increasing, as well as the need of travel more and more kilometers whether in daily routine or on weekend trips. In order to pursue that goal, it is important to study the characteristics and behaviour of drivers. Often, the cases that are meant to be studied are highly dangerous or entail many costs. So that these situations can be tested and adequately evaluated are used driving simulators, where various factors are easily isolated owing to the possibility of repeat the tests over and over meeting all safety conditions with reduced costs. Driving simulators are virtual reality systems that allow reproducing real-world scenarios in a controlled environment. The one we are going to improve is in the Traffic Analysis Laboratory of the Faculty of Engineering of the University of Porto and is called DriS. The sharing of collaborative road environments by different system architectures was the main problem addressed in this work, that is, it was enabled the possibility of multiple people to drive in the same shared environment and interacting with each other using different types of technology. The developed solution was based on the master/slave principle and it enables two different machines to run the DriS simulator in the same environment and communicating with each other. The machine working as master is able to send all the existing cars position and orientation on the road environment, including the one that it controls, to all the slave machines running at the moment the same road environment. On the other hand, the machine working as slave is also able to send the position and orientation of only the car that it controls to the master working machine, so that the master can distribute it across all the slaves in the next iteration, assuring this way the synchronism between all the running instances of the DriS driving simulator. The idealized experimental scenario was having one person using the Volvo's 440 driving station located at the simulation room of the laboratory previously mentioned and another using the second driving station, composed by a Logitech G920 set of steering wheels and pedals, created by us with the purpose of validating the project, driving simultaneously in the same road environment. It is expectable that the studies conducted from now on in the simulator to be a lot richer as it was accomplished a closer environment to the reality, not forgetting that the range of studies that can be made in the simulator was widen as it is now possible to study the interaction between different drivers.

Keywords: Collaborative driving, Shared environment, Driving simulation, Immersive environments

Email: nuno1997pereira@hotmail.com

Acknowledgments

This section is dedicated to thank all of those who made it possible for me to reach this stage of my life, and to become the man that this journey made me be.

I want to thank the Prof. Ricardo Baptista for his orientation during this project, for always putting confidence in me and for giving me suggestions and advice so that I could deliver my best possible work. I also would like to thank the Prof. Carlos Campos for being present and having an active role in the discussion, providing me with valuable inputs and tips. Last but not least, Prof. Miguel Leitão must be thanked as well since he was fully available to clarify doubts and to guide me to the best course of action.

My greatest thanks are indisputably for my mother Lurdes Outeiro, my father José Manuel and my sister Núria Pereira for always being there for me, for believing in me and making me believe in myself and in my skills. Thank you for always being on my side, helping me with whatever I needed and the best you could, because without you none of this could come true.

To my friends, I would like to show gratitude for motivating me and for the companionship and adventures lived during this experience.

Finally, my last piece of appreciation goes to my girlfriend Maria Pires for pushing me to do my best, never doubting my abilities and for supporting me in all moments, whether they were good or not.

Thank you,

Nuno Pereira

*“The good news is that virtual reality is here.
The bad news is that something is still missing”*

MYCHILO STEPHENSON CLINE,
Virtual Reality: a Catalyst for Social and Economic Change

Contents

1	Introduction	1
1.1	Context	1
1.2	Main Problem	1
1.3	Motivation	2
1.4	Objectives	2
1.5	Document Structure	2
2	Distributed Systems	5
2.1	Architectures	5
2.1.1	Peer to Peer	6
2.1.2	Client/Server	7
2.1.3	Master/Slave	10
2.1.4	Architectures comparison	10
2.2	Communication	11
2.2.1	Internet Layers	11
2.2.2	Communication Protocols	11
2.2.3	Sockets	18
2.2.4	Mutexes	18
3	Virtual Reality	19
3.1	Illusion	19
3.1.1	Immersion	20
3.1.2	Sense of presence	20
3.2	Exposure to Virtual Reality	21
3.3	Collaborative Virtual Environments (CVE)	21
3.3.1	Collaborative applications	22
3.3.2	Asseto Corsa Competizione	22
3.3.3	MASSIVE	23
3.3.4	Hyperverses	23
3.3.5	Zero Latency	23
3.3.6	Sandbox VR	23
3.3.7	Networked collaborative work environments	24
3.3.8	Collaboration Challenges	24
3.4	Driving simulators	25
3.4.1	University of Leeds Driving Simulator (UoLDS)	25
3.4.2	Audi's Dynamic Driving Simulator	26
3.4.3	Lander's simulator	27

4	DriS - Driving Simulator	29
4.1	Characteristics	30
4.1.1	DriS Room	31
4.1.2	Image and Sound	31
4.1.3	Adaptive Cruise Control	32
4.2	Important uses	34
4.3	Simulation	34
4.3.1	Simulated traffic	36
4.4	Architecture	36
4.5	Start up procedures	41
5	Project - Distributed Collaborative DriS	43
5.1	Architecture	43
5.1.1	Requirements	43
5.1.2	Use Cases Diagram	46
5.1.3	Possible solutions	48
5.2	Implementation	53
5.2.1	Stream used and data flow diagram	67
5.3	Validation	69
6	Conclusions and Future Work	71
6.1	Future Work	72
A	Profile Characterization Questionnaire	73
B	Experience Evaluation Questionnaire	81
	References	87

List of Figures

2.1	Peer to Peer information delivery	6
2.2	Functionality and processing load	8
2.3	3-tier architecture [4]	9
2.4	Internet Layers - TCP/IP vs OSI	11
2.5	TCP 3-way handshaking	12
2.6	Unicast	14
2.7	Broadcast	15
2.8	Multicast	15
2.9	Fluxo Unicast	16
2.10	Fluxo multicast	17
3.1	Zero Latency Experience	24
3.2	Leeds simulator exterior	25
3.3	Leeds simulator dome	26
3.4	Leeds simulator dome interior	26
3.5	Audi Driving Simulator	27
3.6	Lander's car and moto simulators	28
4.1	DriS road environment	29
4.2	DriS Cockpit	30
4.3	DriS room (side view)	31
4.4	Control Room	32
4.5	Adaptive Cruise Control	33
4.6	Simulation room - view from behind	35
4.7	DriS initial architecture (Adapted from [15])	37
4.8	Simulation's peripheral connections	39
4.9	DriS - Single mode	41
5.1	DriS Use Cases Diagram	46
5.2	Implemented Collaborative Physical Architecture	47
5.3	Peer-to-Peer possible architecture	49
5.4	Master-Slave possible architecture	50
5.5	Collaborative DriS diagram proposal	52
5.6	Collaborative DriS chosen implementation diagram	53
5.7	Definition of a movement as internal movement	57
5.8	Class diagram	59
5.9	Possible command loss	62
5.10	Sequence diagram - SimulationMaster to ViewSlave	65
5.11	Sequence diagram - SimulationSlave to ViewMaster	67

5.12 Command example	67
5.13 Data Flow diagram	68
A.1 Immersive Tendency Questionnaire - page 1	74
A.2 Immersive Tendency Questionnaire - page 2	75
A.3 Immersive Tendency Questionnaire - page 3	76
A.4 Immersive Tendency Questionnaire - page 4	77
A.5 Immersive Tendency Questionnaire - page 5	78
A.6 Immersive Tendency Questionnaire - page 6	79
B.1 Experience Evaluation Questionnaire - page 1	82
B.2 Experience Evaluation Questionnaire - page 2	83
B.3 Experience Evaluation Questionnaire - page 3	84
B.4 Experience Evaluation Questionnaire - page 4	85
B.5 Experience Evaluation Questionnaire - page 5	86

List of Tables

2.1	Architectures comparison	10
4.1	CMD command flags	39
4.2	Turning on the equipment	42
5.1	Requirements	44

Listings

4.1	Speedometer Data Frame	39
5.1	Volvo's Cockpit Data Frame	54
5.2	How all the objects are updated	55
5.3	ObjectType Update Function	55
5.4	Load Movement Function Excerpt	58
5.5	Send object(car) to slaves	60
5.6	Receive function	61
5.7	Add Command to CommandBuffer	61
5.8	Process one command	63
5.9	Updates the object matrix values for position and orientation	64

Abbreviations

UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IP	Internet Procotol
AI	Artificial Intelligence
CSG	Collective Sport Games
API	Application Programming Interface
ACC	Adaptive Cruise Control
UI	User Interface
VR	Virtual Reality
SAV	Simulated Autonomous Vehicles
IPTV	Internet Protocol Television
TTL	Total Time To Live
IGMP	Internet Group Management Protocol
P	Position
R	Orientation

Chapter 1

Introduction

Virtual reality had already proved its benefits and various field of application. It is possible to improve almost every system with the introduction of some Virtual Reality concepts. Nevertheless, when the question is about exploring the collaboration using virtual reality systems, the procedure raises some concerns. This is a very active field of research, in which much brands seem to continuously invest, in the attempt to integrate or improve such technology in their products or services. This dissertation is about exploring the collaboration advantages in a virtual reality driving simulator where all the players will interact with the system using different types of system architectures.

1.1 Context

Increase road safety is an urgent need as the number of drivers is increasing, as well as the need to travel more and more kilometers whether in daily routine or on weekend trips. In order to pursue that goal, it is crucial to study the characteristics and behaviour of drivers. Often, the cases that are meant to be studied are hazardous or entail many costs. So that these situations can be tested and adequately evaluated are used driving simulators.

1.2 Main Problem

The sharing of collaborative road environments by different system architectures is the main problem to solve in this work, that is, enable multiple people to drive in the same shared environment and interacting with different types of technology. The experimental scenario could for example be: having one person using the DriS driving simulator mentioned before, another driving with a Logitech steering wheel with pedals, and leave open the possibility for, in the future, adding more users. This will require the creation of that second driving station and establishing the communication between the computers used to run the simulation, as each driving station has to be in contact with the other, so that it can share its actions across the system, enabling by that way the simultaneous presence at the same road environment. There is more than one possible outcome

for this problem. Several architectures were considered and after analysing what each one had to offer, it was taken the approach that was thought to have the most favourable outcome to the global system.

1.3 Motivation

The motivation to solve this problem was born of the will to make possible the interaction between multiple drivers in the same shared environment and willingly using different devices. Using the simulator to conduit studies is definitely a fair use of the technology since it turns possible to repeat the tests over and over and with all the safety conditions met. Moreover, using a driving simulator to research purposes allows performing the desired tasks keeping the costs low. Therefore, having in mind all the significant impact of addressing this project with the will of improving it and increasing the number of studies in which the simulator could be used, made us want to embrace this dissertation. In addition, here is one quote that I found particularly interesting while I was surfing and browsing about virtual reality on internet.

“The good news is that virtual reality is here. The bad news is that something is still missing”[10]

That caught my attention because you feel the same way when you are driving in the DriS simulator. It gets very predictable and it is noticeable that the simulator lacks one big thing: human interaction. After collaboration being at stake, the whole experience is expected to be more immersive as well as catchier and more appealing. That will cause significant differences in the way people behave because they get engaged and start act as they would normally do, turning their behavior closer to reality.

1.4 Objectives

The main objectives to fulfill by addressing this work are to enable at least two persons to drive the simulator in the same shared environment and being able to test it. Secondly, significant data must be extracted to verify if the immersion felt by the users increased, and if they enjoyed more the collaborative experience rather than the individual one. With this work it is desirable that the tests made with the simulator get a lot more reliable as it will be tested in a closer scenario to reality. It will also widen the range of studies that can be made with the simulator as many new possibilities arise with the surge of the interaction between human drivers.

1.5 Document Structure

In the first chapter, there is a brief overview around of all the project. It starts by defining which kind of problem this dissertation is about, and details the context, the motivation and the general objectives of this project. In Chapter 2, some background on the Distributed Systems is layed

down, either about architectures and communication. This topic finishes with a comparison between a centralised system and a distributed one. In the third chapter (3), virtual reality will be addressed and topics like immersion and presence will be discussed. Furthermore, there are also mentioned collaborative virtual environments, presenting existing technologies and related work in this field.

Concerning chapter 4, the DriS is presented in detail, starting by his main features and explaining how each integrated part works, since the disposition of the room, until how image and sound are being generated. It is also explained some of his features like the Adaptive Cruise Control. Also, there are some references to important projects developed around DriS. Besides DriS, were approached some competitive hardware solutions. The chapter ends with a diagram explaining the DriS architecture when we first met the project. To visually understand the DriS architecture changes, this diagram must be compared with the following diagrams on the next chapter.

It is in the chapter 5 that the implementation will be explained in great detail. Following a logical thread, this chapter starts by approaching the project objectives, moving one to the requirements needed to made that happen. Then, the use cases of the implemented project are presented, giving the reader a better understanding of how the simulator can be used after this project being delivered. Naturally, next are presented and debated the possible solutions to achieve the desired goal. After, exhaustively explained all the architectures and their constraints, logically follows the chosen solution indication and starts its detailed and thorough explanation of how it was implemented. To aid the reader's understanding, there are presented multiple kinds of diagrams. This chapter ends by addressing the validation section in which the experiences are explained and a reference to the questionnaires elaborated to validate them is pointed.

Last but not least, in the chapter 6 are summarized the contributions of this work to the DriS driving simulator and discussed some future and possible developments on top of the project.

Chapter 2

Distributed Systems

A distributed system is one in which components located at networked computer communicate and coordinate their actions only by passing messages [13]. In other words, it is a set of independent components located on different machines that share messages with each other in order to achieve common goals. The first widespread distributed system appeared around 1970's when ethernet and local-area networks were invented. Then, there was an evolution to the major example of a distributed system that it is possible to get, the internet. Other example of a distributed system is the e-mail.

Not all distributed systems have the same characteristics and ones can be more distributed than others. Following, there are presented some usual characteristics of this type of systems.

Openness - Accessible from different machines, with different systems, locally or remotely.

Concurrency - Some systems can take advantage of concurrent processing to enhance performance.

Scalability - In a distributed system we can easily increase the number of participants.

Fault tolerance - The system's ability to continue in service even if a fault occur.

Complexity - Usually, this systems are more complex than a centralised one.

Security - More susceptible to external attacks.

Manageability - It is harder to manage the system.

Unpredictability - Depending on the network load, the system may have different responses.

2.1 Architectures

There are plenty of different system architectures that can be mounted to represent a distributed system. Some of them are present in things we use every day without knowing.

2.1.1 Peer to Peer

When Internet first appeared, in the late 60's, it was a peer-to-peer system. It was called ARPANET and its main goal was to share computing resources around the US. The first hosts known were UCLA, SRI, UCSB and the Utah university that managed to be independent computing sites with equal status. In order to achieve that, they were connected as equal computing peers, rather than in a master/slave or client/server architectures. However, with the internet rising popularity, some architectural changes occurred as we are going to see further. A peer-to-peer system distributes uniformly the responsibilities among machines in the system, known as peers, which can serve as either client or server. (Website reference). There are no additional machines used to provide services or manage resources. In this approach, there is no distinction between publishers, subscribers and brokers. In this type of system architecture, each peer communicate with other peers, not only with one as it used to happen in a centralised system, in order to get the information that he needs. It works this way because there is not a single peer with all the information. Instead, the data is spread amongst several peers, resulting in each one having a small part of the information. In addition, there are peers with replicated information so that, in case of one of them not being available/reachable, there are others with the same information to ensure that all the system requests are fulfilled. One important aspect of this type of architectures is the placement of the data objects across the several hosts and subsequent provision for access to them in a balanced way between the workload and the availability without generating under overheads. [13] For a better understanding of how peers can communicate in a peer-to-peer system network, an example is presented below.

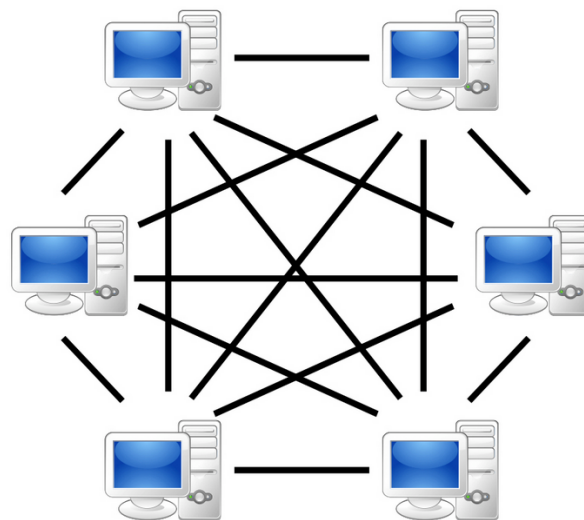


Figure 2.1: Peer to Peer information delivery

Some more recent peer-to-peer systems can share not only data but also computing resources and storage among the multiple hosts. They take advantage of existing naming, routing, data

replication and security techniques to build a reliable resource-sharing layer over an unreliable and untrusted collection of computers and networks.[13] The moment we take more advantage of this architecture is when there is the need to store very large collections of immutable data. Whereas, his effectiveness is not so high for applications that store and update mutable data objects.

Another major advantage of peer-to-peer systems when compared to centralised ones is the easier scalability. The leeway that a centralised system has to expand his services by adding more computers hosting them is limited when all the hosts have to be owned and managed by the service provider.

Despite that they may differ in the resources they contribute, all the nodes in a peer-to-peer system have the same functional capabilities and responsibilities. This type of systems can be designed to offer a limited degree of anonymity to the providers and users of resources.

2.1.2 Client/Server

In this computer network architecture many clients (remote processors) request and receive service/information from a centralized server (host computer). Clients contact the server for data, then format it and display it to the end-user. The end-user can also make a change from the client-side and commit it back to the server to make it permanent. So that this communication may be perceptible and the information easy to interact with, the client side has an interface thought and designed to the end-user. The client may be located on a workstation or on a personal computer, whereas the server normally is located elsewhere on the internet and in a powerfull machine, in order to be able to respond to a large number of requests simultaneously and respond in the quick-est time possible. A client/server system may be implemented of several ways, an so it can have different architectures:

1-tier architecture - It is used to describe systems in which all of the processing is done on a single host. An example of this is a user access to a mainframe through a display/dumb terminal. However, what is displayed and how it appears is controlled by the mainframe.

2-tier architecture - In this type of client-server environment, the user interface is stored at client machine and the database is stored on the server. Database logic and business logic may be filed at either client or server but it needs to be maintained. If Business Logic and Data Logic are collected at a client side, it is named as fat client thin server architecture. If Business Logic and Data Logic are handled on the server, it is called thin client fat server architecture. This is considered as affordable. So, what to choose?

Benefits of thin-client systems:

- Less chance to have viruses, spyware, spam, thefts, etc...
- Easy to keep the software properly configured and patched
- Lower TCO (Total Cost of Ownership)
- Fewer points of failure

Shortcomings of thin-client systems:

- Servers are the central point of failure
- Systems tend to be proprietary
- Multimedia-rich applications require a significant amount of bandwidth to function to their maximum potential

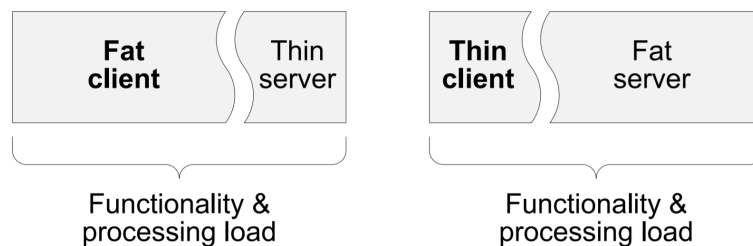


Figure 2.2: Functionality and processing load

In our days, there are some factors driving IT towards thin clients. Some of them are the desktop virtualization, the increase of web-based applications and the high-bandwidth LAN and WAN connections provided by the competent services. Data security and energy saving concerns also help to explore this methodology rather than the fat client. [28]

In two-tier architecture, client and server have to come in direct incorporation. If a client is giving an input to the server there should not be any intermediate. This is done for rapid results and to avoid confusion between different clients. For instance, online ticket reservations software use this two-tier architecture.

3-tier architecture - In this type of architecture we can name the 1st tier as presentation, the 2nd as business logic and the 3rd as data or resource. The 1st is responsible for displaying the information and taking care of the interaction of the user with the client, the 2nd is represented by an application server that contain all the business logic of the application and processes commands, makes logical decisions, performs calculations, and coordinates the application and the 3rd is a resource manager that stores and manages data. For a better understanding see the figure 2.3.

The benefits of a 3-tier architecture are clear concerning topics as flexibility. Changes to the presentation/application/database logic are largely independent from one another. Furthermore, we can not forget that this allows increased security as security can be defined for each service and at each tier. Yet, it also helps to increase performance as tasks are shared between tiers. This systems also have shortcomings like the need for a extra load balance and fault tolerance, and the predicted extra complexity to develop and test.

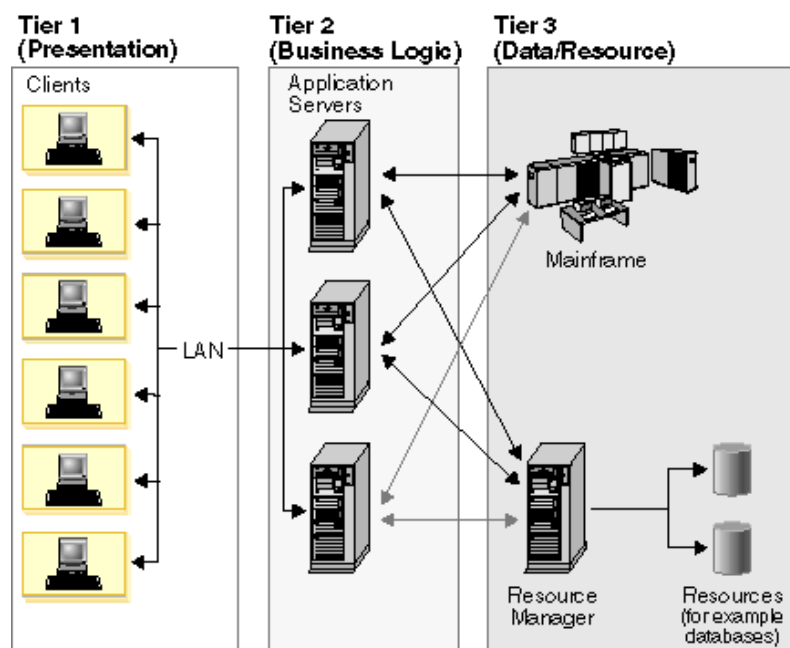


Figure 2.3: 3-tier architecture [4]

In this type of architecture client and server need different amounts of hardware/software resources but they may be not permanent, because at any time the system can horizontally scale with the increase of client machines. The same happens when it comes to the server that may need to be upgraded, which can be done by the migration to a more powerful server or to a multiserver solution (vertical scalability).

One of the reasons why this architecture is so used is that the client is abstracted from knowing anything from the resource manager. It is possible to change the database without the client knowledge. This is clearly an advantage because usually there are a lot more clients than servers, and normally servers are on easier locations to update, which simplifies a lot all the process. Another strength of this model is related to the security provided, since only the servers need to access the data controlled by the resource manager.

Clients can access several different servers, and the servers themselves can act as clients to other servers. Function distribution across servers is an application design decision, for example, a single server can provide all the services that a client needs, or the client can access multiple servers to perform different tasks. Before the application designer decide must take into account items such as scalability, location, and security. Some questions that can help him to better understand what he needs can, for example, be: Are the clients and servers local or is the application distributed over a wide geographic area? Do the servers need to be on machines that are physically secure? It is important to understand that there is not a perfect general solution, there are solutions that contribute to a better fit with the requirements.

2.1.3 Master/Slave

As distributed systems have become more commonplace, the need to assure high capacity distributed databases also increased. With this grown, it got more difficult to preserve the ACID guarantees. This resulted in the use of master-slave architecture. When dealing with this databases only the master has the right to write on them, whereas the slaves can only access the database, because they have read-only access. The main goal of this architecture when dealing to databases is to assure that the same data cannot be modified via concurrent mutation requests. The expression that can more accurately classify a master/slave system is a model of asymmetric communication, where a device or process (master) controls one or more devices or processes (slaves). The master process works as a communication hub. The master usually processes information and then replicates it to all the slaves. The example of the databases presented above is good to understand the concept, as slave processes can be used to show information sent by the master but not change them. It is an helpful architecture to follow if the purpose is to have some machines that are only able to visualise and show data.

In addition, a curiosity that we think that can be interesting to leave here, is the recent disputes opened to debate about the replacement of the terms "Master" and "Slave" to avoid the slavery reference. It will not change anything in the way the architecture works, it is just related to the terms used but it is interesting to observe that this kind of debates do not leave the tech world aside [23][17].

2.1.4 Architectures comparison

Table 2.1: Architectures comparison

Comparison	Peer-to-peer	Client-Server
Concept	Clients and server are not distinguished, each node act as client and server	The server is specified and the clients identified
Service	Each peer can request for services and can also provide the services	The client request for service and server respond with the service.
Storing	Each peer has its own data	The data is stored in a centralized server
Bottlenecks	Can distribute data and balance request across the network avoiding bottlenecks.	When a server is being simultaneously flooded with requests it can get bottlenecked
Speed	The connections between the peers are normally not designed for high throughput rates	The host must provide storage and bandwidth.
Cost	The entire network must be able to provide storage and bandwidth	The host must provide storage and bandwidth.
Privacy	Difficult to determine who is inserting or requesting content	Easy to determine who is inserting or requesting content.

The table 2.1 was added with the purpose of summarise the main characteristics of both peer-to-peer and client-server architectures, and to intuitively spot the differences. The master-slave was not included in that table as it has not many differences when compared to the client-server architecture, mainly because it is based on that architecture with the addition of being based on the underlying principle above explained.

2.2 Communication

2.2.1 Internet Layers

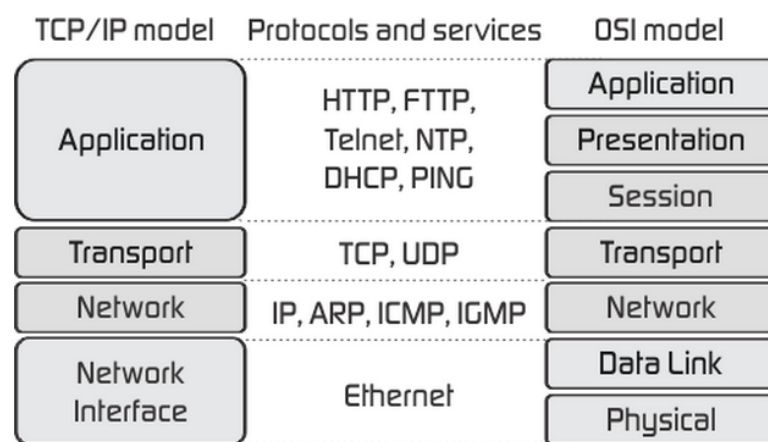


Figure 2.4: Internet Layers - TCP/IP vs OSI

2.2.2 Communication Protocols

2.2.2.1 TCP/IP

The TCP/IP protocol stack has become the standard method of transmitting data over the Internet, and increasingly it is becoming a standard in virtual private networks (VPNs). The TCP/IP protocol stack includes not only internet protocol (IP), but also transmission control protocol (TCP), user datagram protocol (UDP), and internet control message protocol (ICMP).

TCP is an end-to-end protocol that operates at the transport and sessions layers of OSI, providing delivery of data bytes between processes running in host computers via separation and sequencing of IP packets. TCP can control transmission rate by “sensing” when packet loss occurs. If any packet is lost and TCP proves to be due to network congestion, TCP assumes that the transmission rate exceeded the capacity of the network, and responds by slowing the rate of transmission.

IP is part of the TCP/IP protocols. It is used to recognize incoming messages, route outgoing messages, and keep track of Internet node addresses (using a number to specify a TCP/IP host on the Internet). IP corresponds to the network layer of OSI.

TCP/IP is a protocol that provides communications between interconnected networks. The TCP/IP protocol is widely used on the Internet, which is a network comprising several large networks connected by high-speed connections.

It is believed that TCP/IP (including user datagram protocol (UDP)) is the overwhelming protocol of choice, and will continue to be in the foreseeable future.

However, how does it work? In order to a client get information from a server must have place a connection setup, that is several times referred as three-way handshaking. First, the client sends a synchronize (SYN) packet to the server. Secondly, the server answers with a SYN acknowledgement (SYN-ACK) packet, which means that the server has allocated resources to deal with the request that the client is about to send, and to inform the client that he received his SYN message. At last, the client responds back with an ACK packet.

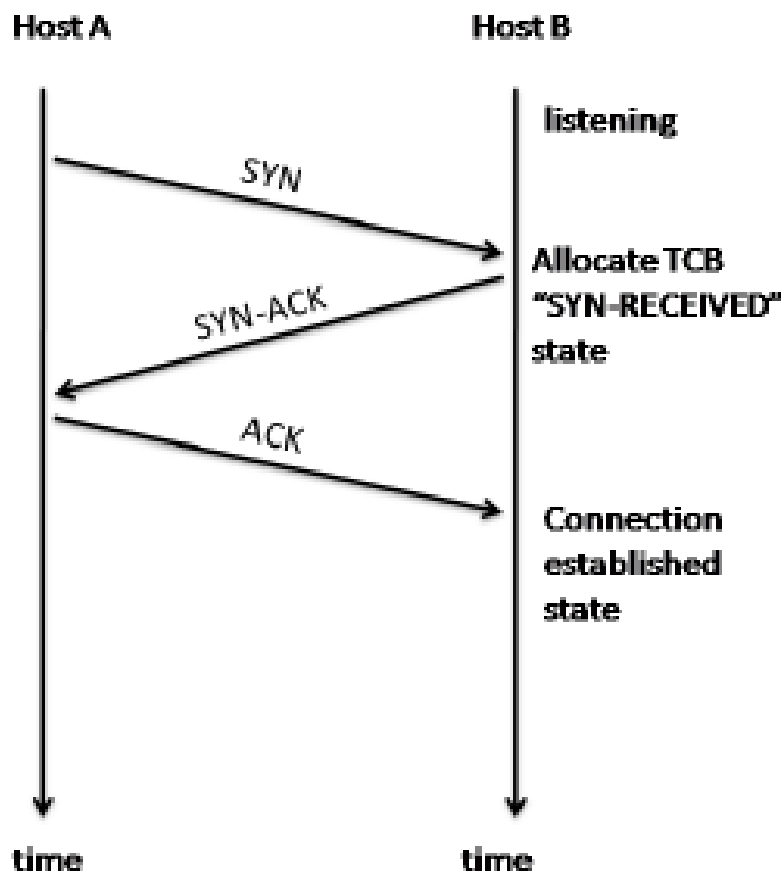


Figure 2.5: TCP 3-way handshaking

TCP/IP divides the user data to be sent between the various systems on the IP network in packets. So, when a large file is sent down the protocol stack, the IP function needs to do the segmentation and packetization of the data. A header is placed on the packet for delivery to the data link. Then, the network layer performs the routing and switching of this data. When a packet is prepared for transmission, IP protocol does not specifically route the call across a specific channel.

Instead, it places a header on the packet and lets the network deal with it. Therefore, the outward bound packets can take various routes to get from a source to a destination. More precisely, the packets are in a datagram form and not sequentially numbered as they are in other protocols. IP makes its best attempt to deliver the packets to the destination network interface. However, no assurances that data will arrive and that data will be free of errors are given. When something goes wrong the IP protocol does nothing about it.

When doing IP routing, one packet can be sent along the network in a loop, and to fight this, IP has a counter mechanism that decrements every time the packet passes through a network node. It can be called a "time to live" on the network and its placed in the packet's header information to allow only a certain number of "hops". That way, it prevents an undeliverable pack to loop around the network. If the counter reaches 0, the node will discard the packet.

To complement this IP protocol and fix the points where it do not perform so well, we have TCP which provides controls to ensure that a reliable data stream is sent and delivered.

At the sending end, TCP puts a byte count header on information that will be delivered to the IP protocol layer and encapsulates it as part of the packet.

When the receiving end gets the packets it resequences them and ensures its accuracy. If all of the IP flow is not received accurately, the byte count acknowledgment or non-acknowledgment message can be sent back to the sending end, triggering it to resend the needed bytes to fill in the missing portions of the packet flow. [1]

2.2.2.2 UDP

UDP stands for User Datagram Protocol and it is a communications protocol which is essentially used for establishing low-latency and loss-tolerating connections between applications on the internet.

It enables the transfer of data before an agreement is guaranteed by the receiving party, which speeds up the transmissions. As a result, UDP is preferable in time-sensitive communications such as, for example, DNS, Network Time Protocol (NTP), gaming, video or audio playback, which can suffer some data loss without causing an impact in the perceived quality.

Both TCP and UDP run on top of IP and are sometimes referred to as TCP/IP or UDP/IP. Nonetheless, there are an alternative to each other and show important differences when compared. For example, UDP enables process-to-process communication, while TCP supports host-to-host communication. While TCP sends individual packets and is considered a reliable transport medium, UDP sends messages, called datagrams, and is considered a best-effort mode of communications, insofar as it tries the best possible to send all the datagrams but does not give any guarantees that the data will be delivered and if it is not, this protocol will not bother to retransmit lost or corrupted messages, turning in lower bandwidth overhead and latency.

The UDP header has 4 fields, each of which uses 2 bytes:

Source port number - which is the number of the sender;

Destination port number - the port the datagram is addressed to;

Length - The length in bytes of the UDP header and any encapsulated data;

Checksum - Which is used in error checking (Its use is required in IPv6 and optional in IPv4.).

2.2.2.3 IP

In this section, it will be more clearly explained the IP protocol than above. Conventional IP networks allow any device to send a flow of data to another device, usually using unicast communication. Each connected device whether it is a desktop, notebook, smartphone or server has an identifier called IP. The unit of transfer of this protocol is a package or datagram composed by a header and a block of data. Accordingly to the newest version of IP (IPv6), the header of an IP packet include: a version, traffic class, flow label, payload length which is the total size of the packet, the next header, hop limit - which represents the maximum number of times that a router can send the same packet, a source address and a destination address. These fields are important insofar as the routers do not look at the TCP/UDP messages. Instead, they handle the IP header each time they send a datagram. It is the TCP at the destination that handles the TCP header.

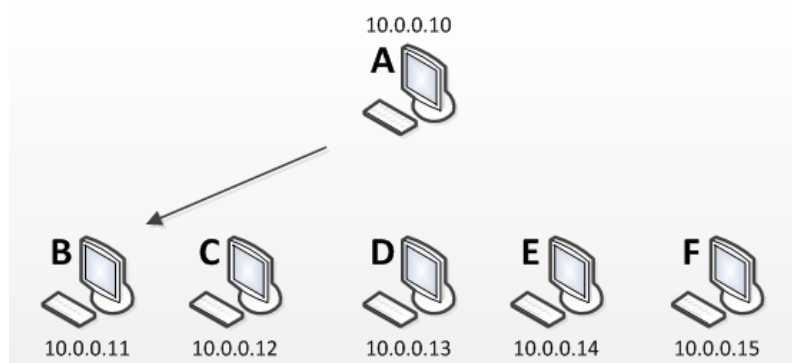


Figure 2.6: Unicast

2.2.2.4 IP Broadcast

In this type of communication, data is sent to every device connected to the network specified. By convention, the IP address reserved to perform this type of communication is the last IP address of the network. This means, to perform the communication illustrated in the following picture, the data was sent to the IP address 10.0.0.255. The A station sent a copy of the data to B, C, D, E, F devices without having to individually identify them.

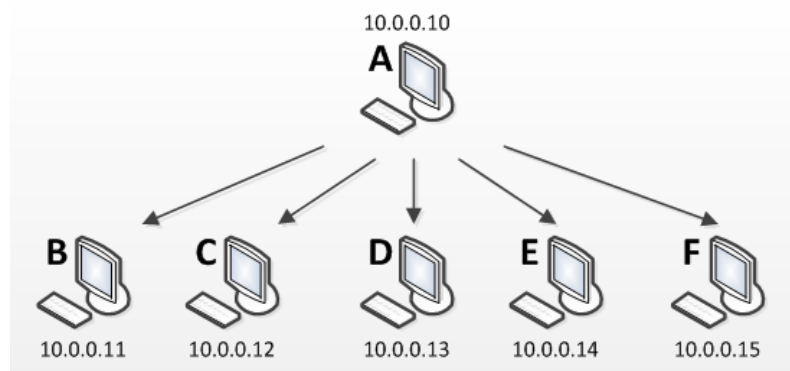


Figure 2.7: Broadcast

2.2.2.5 IP Multicast

The same does not succeed if our desire is to use IP Multicast. In this type of communication, a specific IP address between 224.0.0.0 and 239.255.255.255 must be used, and instead of just delivering the data to anyone connected to the network, it will only send the data to the devices that have demonstrated their interest to receive the data.

Accordingly Luís Henrique M. K. Costa and Otto Carlos M. Bandeira Duarte in their cited article [12], a IP multicast services faces a group like an open communication insofar as:

- Any network connected device can take part without the sender's authorization;
- There is no limitation regarding the number of groups of which a device belongs;
- The sender is not required to belong to the group;
- Each device can join/leave any group at any time;
- The members of the group are not known neither by the sender neither by the other members.

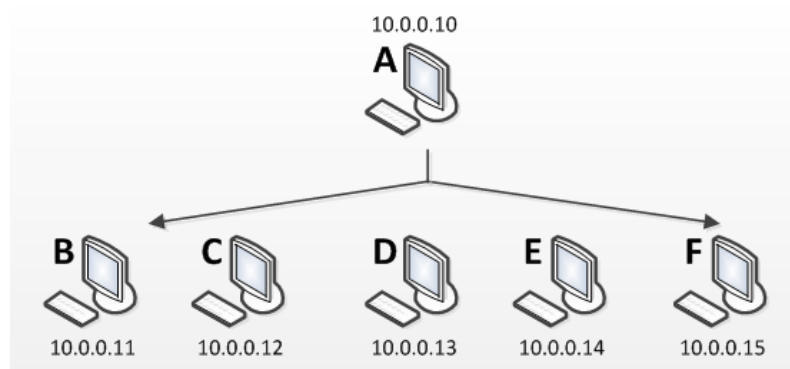


Figure 2.8: Multicast

This solution is of special interest to this project because it may be essential to guarantee that the whole system works harmoniously and efficiently. It is used as it has been said in IPTV, and other real time data deliveries like teleconferences, webinars, distributed computing, the so on demand distance learning that we were forced to implement due to the pandemic reasons and most streaming apps. This dissertation project lays on similar principles as accuracy and speed is needed to assure a usable system from which we can take advantage.

2.2.2.6 Comparison

In order to visibly understand the difference between this protocols it is going to be used a simple base scenario of communication between a server and some clients. For example, a IPTV server dealing with 5 clients, using unicast flows with approximately 1 Mbps of bandwidth each as we see in the figure 2.9.

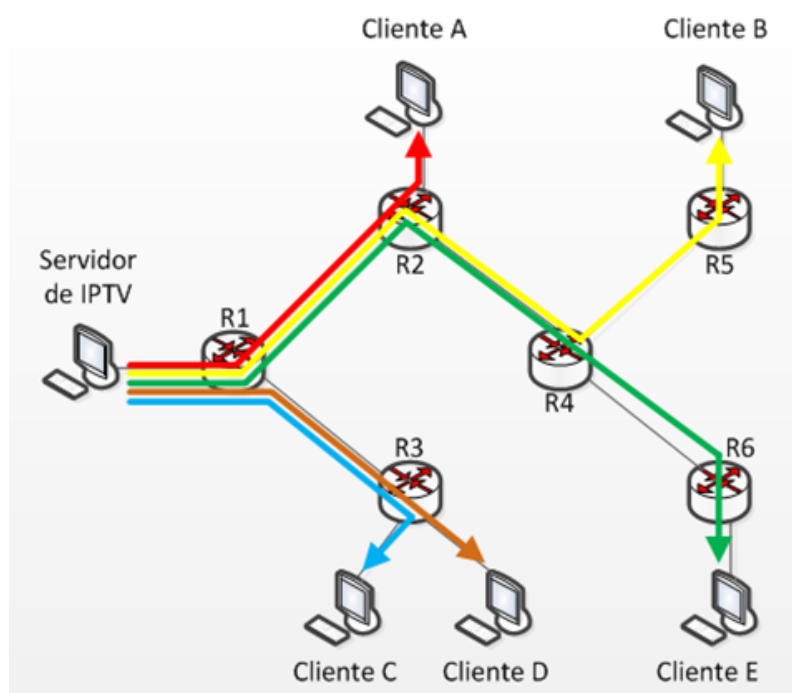


Figure 2.9: Fluxo Unicast

In this scenario, between the server and the router R1 there are used 5 Mbps of bandwidth, in the link between the R1 and R2 would be used 3 Mbps, and the ones between R1 and R3, and R2 and R4, would use 2 Mbps each. This seems unnecessary since routers R1, R2, R3 and R4 are sending multiple flows of the exact same data. If we escalate this scenario to a real case scenario with thousands of users, the inefficient use of bandwidth is astonishing.

To face this problem, a multicast solution, having as basis the exact same scenario, will be shown in the next picture 2.10.

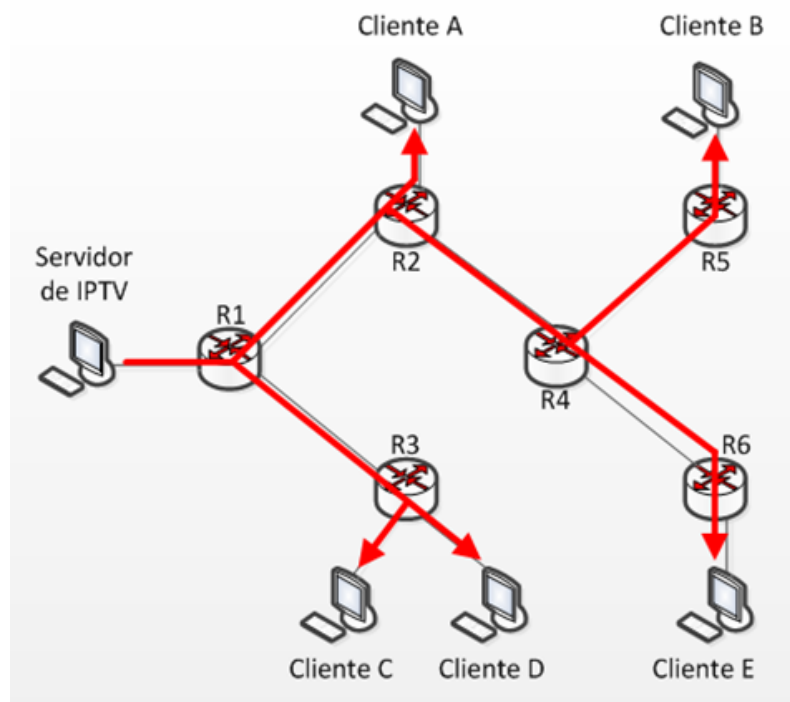


Figure 2.10: Fluxo multicast

The major improvement we get by using this solution is that, regardless of the number of users interested in receiving the data, it is possible to guarantee that a unique flow goes through all the network. In this case the bandwidth used between each node will be around 1 Mbps.

So, with this comparison it is possible to confirm that one of the major advantages in using multicast to this type of scenarios is the lower consume of bandwidth between the server and the clients. That happens due to the fact that there is not unnecessary packet sending, they are just sent to the pretended users. With the massive grown of internet, and application users, this solution offers scalability, allowing the apps to fulfill a large number of users without overloading the network.

Now that I mentioned it, it is important to say that error management, flow controls and congestion flows are essential to assure a reliable delivery of data but are not dealt in this layer. Instead, they are handled at the layer above, the transport layer.

With this implementation, it is also reduced the amount of processing done on the network devices, due to the lower number of packages disseminated through the network and it lacks no investment in the network infrastructure as all the modern equipment allows the forwarding of multicast packages and implementation of routing protocols.

In contrast, the not so good aspects of this solution may be indicated as the following: the sender not being able to know the identity and the number of clients that are receiving the data, and the fact that the security of networks using this feature can easily be explored by attackers, since it is possible to connect and disconnect to groups anonymously.

2.2.3 Sockets

Accordingly Oracle, a socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number in order that the TCP layer can identify the application from which data is destined to be sent to. An endpoint is composed by an IP address and a port number. The two endpoints of a TCP connection uniquely identify it. Even that we do not know that, sockets are

The usual functioning of socket-based communication is the following: A server is running in some computer with a socket bound to a specific port number, and it is listening on that port, waiting for a client to establish a connection. On the other hand, the client knows the hostname of the server and the port number on which he is listening to. So, the client sends a connection request to that machine in the specified port number. The client needs to identify himself so that the server knows where to send, this means, it has to specify a port number where it is going to receive the information. If the server accepts the connection, then it will create another socket that will be associated with the new communication channel, leaving his original socket free to keep receiving connection requests. A file descriptor relative to the new socket is returned as output parameter of the accept function. In other words, the server process creates a sub-process to handle the client connection, preserving the reception of new connection requests in the initial socket [27].

In order to a client creating a socket all it needs to do is ask for a connection establishing through the connect function represented below.

- int connect (int sockfd, struct sockaddr *nome, int dim);

This function receives as a parameter, the name of the server socket, and by name it is meant the server address and port number, and it returns, in case of success, a file descriptor pointing to the communication channel just established.

2.2.4 Mutexes

A mutex is a very used technique in concurrent programming to avoid that two or more processes or threads simultaneous have access to the same resource, preventing data inconsistencies due to race conditions. A race condition happens when, as we said above, two or more threads have to realize operations on the same memory area, but the results of computations depend on the order in which these operations are performed. Therefore, the application gives a mutex to threads in the order that they are requested. Nonetheless, the process schedule can be adapted to rearrange the order when faced with higher-priority threads. [2]

So, mutexes give the program sureness that it is the only one executing an operation in the critical section, because that section is protected by the mutex object. However, the use of mutexes rises other problems like performance, since they decrease the portion of the program that can run in parallel. Dealing with multiple mutexes also means the potential for deadlocks to happen. Deadlocks surge if overlapping mutexes are locked by multiple threads in a different order. [16]

Chapter 3

Virtual Reality

Virtual Reality [40] has increasingly been the field of research of more and more studies since his first "discovery" in the 60's. It was Ivan Sutherland [34] in 1968 who created what was widely considered to be the first head-mounted display system. Despite that, the "virtual reality" term was only coined by Jaron Lanier [3] in 1987, in a period of intense research activity into this area. It was also the same man that 2 years before had created a company branded as VPL Research, which was pioneer in selling the first virtual reality gear equipment such as glasses, gloves and data suits. Meanwhile, technology has evolved in so many different ways, either hardware as software, which enable the development of VR applications. Besides, as the interest in exploring this area and their applications has been increasing, alongside with the interest, the investment in research and development of VR compatible devices has increased too. All of this looked quite promising but then it had not evolved as the public would expect mostly due to technology limitations. That caused an hype decrease as technology was not prepared to create worlds as believable as reality. However, in the late 2000s virtual reality solutions were creating acceptable representations of the world for entertainment, training and design. In our days, the concept is increasingly gaining notoriety and more uses cases are appearing.

3.1 Illusion

The predominant perceptual sense of most human beings is the sight, being as well the principal means of acquiring information. Though, so that the human being perceives the real world, he explores other senses as well, and the same must be addressed when creating a virtual reality experience. So far, VR experiences manage to create sound and haptic inputs as well, whereas smell and taste are still in a pioneering stage, due to the complexity of the required technology. So, the goal of an VR experience is to give the user the illusion of being in an environment that can be perceivable as a believed place and create the possibility of the user to interact with that environment efficiently and comfortably. According Gutierrez [22], in order to achieve that we must take into account two important factors: immersion and presence.

3.1.1 Immersion

Immersion is a measure of evaluating an experience and depends on how much the user can perceive (see, hear, touch) the real world during the simulation. A VR experience can be categorised in 3 categories: fully-immersive if it takes advantage of a Head Mounted Display (HMD), semi-immersive if it is used a large projection screen or nonimmersive if it is only in a desktop. Unexpectedly, the first VR systems were fully-immersive and were used several kinds of head mounted displays, with the belief that it would help the believability and efficiency of the simulation. They got different rates of success but some were difficult to use, or at least not practical. It is common, when using HMDs, to experience motion/cyber sickness, which is explained in better detail in the next topic 3.2. It can also occur with other VR experiences that not use HMD, but it does not happen so often.

Semi-immersive systems guarantee 3D sound and quite good resolution graphics and depending on the type of experience, if the user does not need his own Point of View (POV), sometimes it is possible to share the same simulation with several users, which make it a good solution when collaborative work is the intended goal.

Regarding the non-immersive experiences, contrary to what we may expect, having a small screen (usually desktop), catch the user's attention for longer periods of time. It is the type of experience that we can more easily find, since it is the more affordable one. Here we are talking about video games for example, and if it is a good one, it is a tough opponent when it comes down to psychological isolating the user from the real world.

Immersion is directly dependent of the physical extent of the sensorial information simulated, and increases with isolation from the physical environment. The more the user feels isolated from the physical environment, the more he feels included in the virtual simulation. That way, the user can naturally interact with the environment which helps to improve the stimulus that support the perception of self-motion [38].

This engagement factors, among many other psychological VR reactions, are yet object of research as it is not clear the relation between factors and responses or degree of interest. Another important factor that influences the whole experience is the sense of presence that we will discuss in the next chapter 3.1.2.

3.1.2 Sense of presence

The term started to appear as “telepresence” and it was coined by Marvin Minsky in 1980 as a phenomenon that a human operator develops a sense of being physically present at a remote location through interaction with the system's human interface [11]. VR was responsible for transforming the users into active participants in a computer generated three-dimensional virtual world, and therefore generating the illusion of taking part in a virtual computerised world rather than the external observation of the environment around the user. By making use of the existing sensory stimulation devices, the users are allowed to enter that three-dimensional worlds where they can see, hear, touch, move and explore. That virtual space where the user is placed immerses the user

into an egocentric position similar to the real world. Of course, the level of immersion will depend on the diversity and quality of these impressions. The sense of presence generated will be determined by the insertion of the subject in the system. Immersion is directly dependent of the physical extent of the sensorial information. Alongside with immersion, the sense of presence also increases when the sensorial information increases. As Coelho said [11], VR is all about misleading the senses, inducing a feeling of reality through an immersive synthetic system. However, our senses do not are misled easily, there are a lot of factors that contribute to perform this effectively. One of them, is the system response time, which takes an very important role in our project, as the time adjustment between the user's actions and the perceived effects of those actions on the environment must be low, otherwise the experience is ruined. Another, major factor is the field of vision, that must be was wider as possible, owing to the fact that being larger produces two desired effects at the same time, provides larger and more compelling visual motion cues, and simultaneously reduces the field of vision of the real-world environment, which helps reducing the distractions from the real world.

3.2 Exposure to Virtual Reality

When what the user sees, feels and senses do not match, the user experiences sensory mismatch, which can leads to the motion sickness. One example of this can for example be, when the sensory channels get confused about body orientation and position as they may not coincide with the virtual world. Cybersickness is another possible effect of the virtual reality exposure, it is similar with the motion sickness, but it is only triggered by the visual senses. There is yet the simulator sickness which is produced by vehicle simulators as Stanney [32] insisted to distinguish from the cybersickness. Both cause the same negative effects: nausea, sweating, dizziness and fatigue [20]. Those are the most common but it can also provoke eye strain, headache, pallor, dryness of mouth, fullness of stomach, disorientation, vertigo, ataxia and vomiting [24]. One aspect that must be taken into account is the after-effects that may surge in the hours following the experience.

As a consequence of these, in case of a VE experience in a flight simulator, many air force bases have mandatory grounding polices demanding that a pilot cannot fly an aircraft from anywhere from 12 to 24 hours after the exposure. When it comes to more common VR experiences, many VR entertainment centers require that users not drive for at least 30 to 45 minutes after exposure.

It were already identified some possible factors of cause, but there is not yet an efficient method of fully prevent that.

3.3 Collaborative Virtual Environments (CVE)

The shortest definition for a CVE is a virtual world shared by participants across a computer network. Each user must be represented by some embodiment able to reveal his presence, location, identity, movements and activities. Normally, the embodiment is an avatar or an object, in case the

user is controlling for example a car or a moto, in which representing the user itself does not have so much importance. These embodiments are used to interact with the contents of the world and to communicate with the other users. CVEs represent a technology that may support some aspects of social interaction not so well assured by technologies such as audio and videoconferencing and shared desktop applications. So that social interaction may be established, a proper environment is needed, highlighting the important role of a physical space to potentiate peripheral awareness on the participants.¹

3.3.1 Collaborative applications

In driving simulation, if reality and high fidelity are wanted, the realistic effects of driver-driver interaction should be a concern [19]. As such, there is a need to study how collaborative applications work, what exists at the moment and how they are operating to enlight us of possible system architectures. So that this real time shared environment can be successful, low latency communications between the clients of the server is needed. Latency is the name given to the time it takes for an instruction sent by the client to the server, to have its response broadcasted to all other users in the play [14]. One way to guarantee a proper real-time shared environment is to make sure that latency is lower than the frame rate. There are multiple ways of minimizing the latency but not completely eliminate [30]. To accomplish that, were developed many latency compensation algorithms, such as Bernier's Half-Life algorithm [7] which deals with the real-time goal. Other common problem in distributed collaborative virtual environments [9] are the abrupt state transitions due to communication delays, which turn one user movement only apparent for other user after the delay. All of this makes that the environment showed to one user different from the one showed to another during the period of delay, followed by an abrupt synchronization which causes a big jump in the user's view of the world. This jumps do not happen in the real world, which upset the users. In the following of this, the partnership composed by Roberts, Sharkey and Ryan developed a perception model, which with the aid of some filters, solve this problem by ensuring a continuous version view of the environment for each user [31]. Although this is not the focus of this thesis, it is important to be aware of such problems that certainly will arise. Mark [26] believed that in order to have multiple users interacting in the same environment, must exist information transparency, which could not be more true when talking about driving simulators, where there is a need to all the users know each other's position and which actions are doing in real time.

3.3.2 Asseto Corsa Competizione

Asseto Corsa Competizione is a multi-player racing simulation video game of great quality used by a lot of simulators owners. It uses Unreal Engine 4² to ensure photo-realistic weather conditions, car materials and graphics, night races, motion capture animations, reaching a really good standard

¹ACM: Digital Library: Communications of the ACM Last accessed in 06/03/2020

²Unreal Engine - URL = <https://www.unrealengine.com/en-US/> Last accessed in 2020-02-10

in terms of driving realism and immersion. It feels very real due to the fact that it uses sophisticated mathematical models that accurately replicates a lot of factors such as tyre grip, aerodynamic impact, engine parameters, suspensions and electronics systems that determine vehicle balance, as well as the influence of mechanical damage on the car's drivability [36].

3.3.3 MASSIVE

Greenhalgh and Benford [21] developed a collaborative virtual environment for teleconferencing called MASSIVE in 1995. It allowed multiple users to communicate using combinations of audio, graphics, and text media over local and wide area networks. The spatial environment that it created provided a flexible and natural atmosphere for a conversation. The system was prepared to be scaled to a large number of users. Years later, in 2009, [18] decided to implement a P2P based avatar interaction, turning all the system in a hybrid network, as the P2P was only used for tracking the avatar.

3.3.4 Hyperverse

It is an immersive multiplayer virtual reality experience in a free-roam VR park for groups of people within 2 to 6 persons. The users are physically in the epicenter of the action. The difference between Hyperverse and other similar experiences as we are about to see ahead is that it uses markers placed on the ceiling, which are referenced by cantilevered cameras atop the backpack PCs to create accurate tracking of the hands and body. This kind of experiences are getting some notoriety and there are arising more competitors to the scene.³

3.3.5 Zero Latency

It is also a multiplayer virtual reality experience with plenty of space to freely walk around in, there are no cables or wires, which makes all thee experience very immersive. The VR Gear that users carry is composed by a VR headset, a backpack and a controller. There are 7 possible games to choose. Zero Latency has greatly minimized the risk of player-on-player or player-on-wall collisions as each player has access to an on-screen indicator which appears whenever they could potentially get too close to a physical barrier or to another player. This one has already arrived to Portugal, as there is one of these free-roam VR parks in Lisboa [37].

3.3.6 Sandbox VR

As the name may reveal, this is one more entire body virtual reality experience similar to the previous ones. The users must be equipped with haptic suits, 3D precision body trackers and a headset. This experience is up to 6 players and enables the user to act like he really were on the game.⁴

³HyperVerse - URL = <http://hyperverse.io/> Last Accessed in 10/02/2020

⁴Sandbox VR - URL = <https://sandboxvr.com/> Last Accessed in 10/02/2020



Figure 3.1: Zero Latency Experience

3.3.7 Networked collaborative work environments

Concerning Collaborative Work Environments, or CWEs, as many prefer to call them, a bunch of software solutions are known to satisfy the client's expectations. Most of them include as their top functionalities, shared project workspaces, document management, TO DO boards and calendars management. One CWE developed in 1996 by [8] was a worldwide application focused on manipulating an object defined by a three-dimensional model. The major feature was the possibility to each one of the remote participants in the collaborative work environment to manipulate the object.

Salas [29]'s article was a illustrative artifact about systems and methods for providing a collaborative work environment over a network, more specifically, in this paper, the goal was to enable multiple users to view, edit and create common documents. Data objects were transmited by network which then were combined with stored templates to render the HTML pages. Manolescu [25] and Anupam [5] also developed distributed CWE's.

3.3.8 Collaboration Challenges

The requirement to support real-time interaction between large numbers of simultaneous participants distributed over a network is a great challenge. Concerning the scalability of a CVE, it is important to define which is the number maximum of simultaneous participants that can be supported. It is also important to have in mind the graphical and behavioural complexity of the virtual environments and their embodiments. Large numbers of active participants generate high volumes of network traffic, especially movement updates and audio packets. The servers on the network may have to process this data, but even considering that they are able to deal with all the requests, bottlenecks may occur in other processes of the network connection, especially when the users are in different networks. Besides, good computers are also needed as after receiving the information, they must be able to process it and render the shared virtual world quick enough to maintain a

satisfactory quality. There are some techniques that decrease the amount of data transferred via network such as, for example, constraining a user from receiving audio packets from users that are too far from him. However, that is not the focus of this research, at least, by now. [6]

3.4 Driving simulators

In this section there are presented three different types of driving simulators. There is one within the academic scope, one representing the professional scope and another simpler made by a company that addresses simulation solutions.

3.4.1 University of Leeds Driving Simulator (UoLDS)



Figure 3.2: Leeds simulator exterior

This simulator was launched in 2006 and is composed by the dome visible in the picture, and inside it was a car. It also has a curve screen in front of the car, and exterior mirrors are functional with correct images from behind. The virtual environment projected in the interior of the dome allows for a 250° horizontal vision range in the front of the car and 45° in the vertical. The physic system is layed on rails with 10 meters each. It has 8 degrees of freedom as it can moves a bit in almost every direction and lean itself as well. It also allow rotation movements as well. The projection system is composed by five computers using graphical motherboards FX3000G and FX4500G. It delivers 8 channels with 1024x768px resolution at 60FPS.

It has a wrap-around projection of a driver's eye view and as such it provides a realistic and familiar driving environment which enhance the illusion of driving a real car. [35]



Figure 3.3: Leeds simulator dome



Figure 3.4: Leeds simulator dome interior

3.4.2 Audi's Dynamic Driving Simulator

Audi's driving simulator has a simpler or lighter appearance, as it is simplified to the strictly essential. It is constituted by a driver's cabin mounted on a hexapod structure and a 4 meters tall, 180 degrees wide screen placed in front of the cabin. That structure has 6 electrically driven cylinders that are responsible for moving the cabin precisely within the room by 60 centimeters in any direction. As in DriS simulator there is a steering actuator which generates steering

resistance/force-feedback. Seven LED projectors are in charge of drafting the road environment on the big screen.



Figure 3.5: Audi Driving Simulator

Richard Uhlmann, the project manager and development engineer from the “Chassis Concept Properties” department enhances that the "driving cabin moves on the hexapod according to the steering input and the image automatically adjusts in response to this movement", which is a crucial fact to create the impression that the user is actually driving in a car. It is also important to highlight that the driving performance needs to be replicated as precisely as possible so that the drivers do not feel motion sickness. Another similarity that this simulator has with the DriS is the darkness around the driver as the Audi’s room is painted black and the lights are turned off while the simulator is operating. They start developing this driving simulator in 2012 but it was only recently that it has been put in use [33].

One of the main purposes of this simulator is for enabling the Audi engineers to access the driving characteristics of the new models right from the beginning of the development process. The sooner they can test and experiment that, the more errors are avoid in the development of a new model, and with that, the more money that can be saved in the process. As you may have guessed, the simulator is prepared to be configured with vehicle data such as weight, suspension, wheelbase, among others. Uhlmann dare to say that the driving simulator was already part of the comprehensive virtual development chain used to design new chassis.

One thing that this simulator has that DriS does not is the possibility of losing control of the car, which can happen in real life. For example, if you drive too fast going into a curve, the rear of the car will begin to skid, the image rotates, and the simulator gives you a good shake, which is a realistic element that misses on DriS.

3.4.3 Lander’s simulator

This one is not so complex, it seems like a carousel car, it has no roof which may be not so immersive, however, it has three screens, one for the front view and the other two, one for each side views which helps a lot in the immersiveness. It is developed by a company named Lander,

which makes many kinds of simulators including custom ones. The one we see on the picture is their car simulator, it has 3 degrees of freedom of motion. It is possible to sense the motion when driving on curves or experience the acceleration and breaking sensations. ⁵



Figure 3.6: Lander's car and moto simulators

⁵LANDER Simulation and Training Solutions - URL = <http://www.land simulation.com/en> Last Accessed in 10/02/2020

Chapter 4

DriS - Driving Simulator

In this topic, it is presented the driving simulator DriS, developed by several northern portuguese institutions since the end of the 90s and located at the traffic laboratory of the Civil Engineering department of the Faculty of Engineering of the University of Porto.

DriS make possible a real driver enclosed in a real vehicle to drive around virtual realist roads generated by a graphical station. By using a a real car fixed in the center of the lab, the immersion and the sensation of realism of the experience is a lot higher. Since the 90's, the DriS has been upgraded through the years. This work will contribute to one of the improvements made over the years.



Figure 4.1: DriS road environment

4.1 Characteristics

The simulator has a real car (Volvo 440 Turbo), with all the working features like the steering wheel, gauges, a gearbox, ignition, among others. All of this controls are functional, and pass the information read to the simulation environment. An example of that is the steering wheel, which is disconnected from the front wheels of the car, but still has the same movement range. The difference is that the values are transmitted to the simulation, and as a response it is able to simulate the force feedback accordingly to the actions performed in the simulation. The same happens with the accelerator, brake and clutch pedals which are no longer connected to the engine, but instead connected to a mechanism which measures the pressure that it is being made on them and transmit that values to the simulation engine, which in turn, apply them on the car of the simulation. There is a sensor in the gearbox to detect in which gear the selector is and to transmit that information to the simulation. There are also similar sensors to detect and monitor in real time the position of the ignition, the handbrake, the horn, the infotainment devices, the turning signals and of the lights enabler. The infotainment device was added to the car and installed in the center console. The driving station is also instrumented with a set of video cameras to visually record the actions made by the driver during an experience. The picture 4.2 shows the cockpit of the Volvo.



Figure 4.2: DriS Cockpit

In a more detailed and technical way the simulator architecture can be divided in the following elements:

- Simulation Room
- Control Room and Experience Preparation
- Simulation System and Image Synthesis
- Image Projection System and 3D Sound Projection System
- Virtual Instrument Panel System
- Infotainment System

4.1.1 DriS Room

The lab where the DriS is located has around 75 square meters and can be divided in 3 different areas. Beginning with the simulation room, it can be said that it has an image projection system composed by a 180° curve screen and several projectors. The 3D sound system is spread over the room so that it is possible to distinguish a sound from the back or the front of the car, which is located in the middle of the room. The room provides a black false roof and it is completely isolated from the outside lights. When it comes to the control room number 1, it is equipped with several computers and monitors, one to perform the simulation, another to enable to instrument panel (gauges), another for the sound system, and some more to perform experiences. The control room number 2 is used to keep the main hardware and peripherals that do not require frequent direct access.

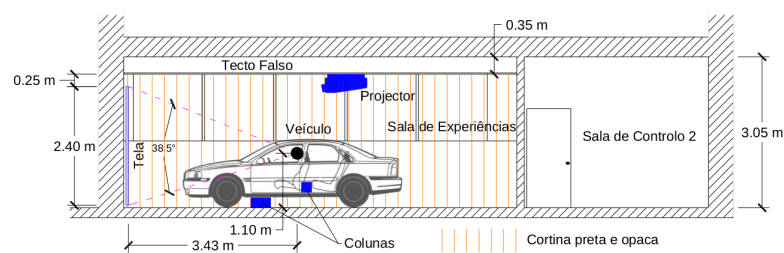


Figure 4.3: DriS room (side view)

4.1.2 Image and Sound

The images presented in the curve screen are generated in the principal simulation system through a graphic board GeForce 1050, with a 128bit Nvidia processor and 2GB of internal dedicated memory, whereas image rendering is done by a dedicated genes framework, developed on top of OpenSceneGraph. Then, that image is sent to the Barco Data 801s projector using separated RGB

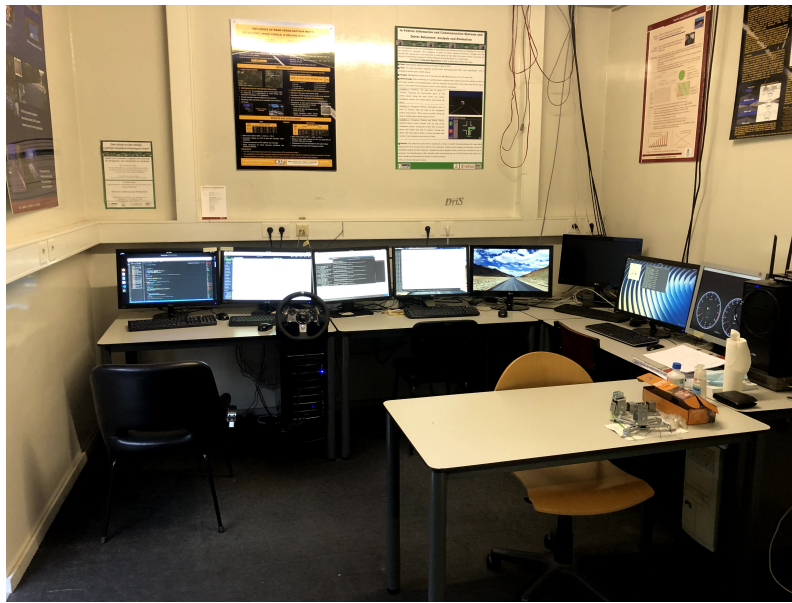


Figure 4.4: Control Room

coaxial cables. This RGB independent system allows a balanced focus calibration, enabling to show images with 1280x1024 pixel resolution at a frame rate of 60 frames per second. Although the screen where the images are presented be curve, right now, it is only used part of it. Sound is a valuable asset when it comes to provide immersion and realism to the driving simulation. The sound generation system is composed by a computer to generate and simulate the sound, two audio amplifiers and four loudspeakers. The sound computer is equipped with two soundboards Sound Blaster AWE32 from Creative Labs, allowing the sound reproduction in four different channels. One of the channels is used to play the engine rotation sound, while the others are used to surround sound reproduction, providing the driver with different location stimuli. The amplifiers are placed inside the control room, allowing the professionals to change the volume depending on each specific situation. In order to better reproduce the sound heard by the driver in a real situation and to be the most close possible to a real situation the loudspeakers were distributed within the room and inside the vehicle. Two of them are placed on the front corner of the room and another in the center back. The other speaker is placed under the vehicle to simulate the engine sound.

4.1.3 Adaptive Cruise Control

The major role of this functionality is to help the driver to maintain a safe distance between himself and the the vehicle in his front. The driver chooses the speed pretended and also the minimum distance that he allows to have between himself and the closest car in front of him. If the radar identifies that the vehicle right ahead is going lower than itself, then it automatically adapts his own speed so that safety do not be compromised. As soon as the road in front of the car gets clear, it immediately returns to the pre-established speed. Nonetheless, it is important to say that the Adaptive Cruise Control do not break when faced to pedestrians, animals or small vehicles like

motorcycles and bicycles. The same applies to vehicles which have zero and near zero speed or circulate in opposite direction. In order not to neglect comfort to the occupants, the ACC tries to regulate the speed in a smooth way, and so, when presented to situations where an emergency break is needed, that must be driver's responsibility. However, if we are presented to one of this cases, where the vehicles needs to break more than the ACC allows, the ACC turn up a light in the instrument panel and a warning sound. Despite that, it is advisable that the driver must rely more on himself and perform the braking before the system warning. Besides, the ACC can be activated with the preposition of following another vehicle with a speed comprehended between 30-200 km/h, but one more time there are some restrictions as it is the case of the speed going under the 30 km/h or the rpm of the engine going too low, which in this cases will cause the disable of the speed control and all the systems associated like the automatic braking.

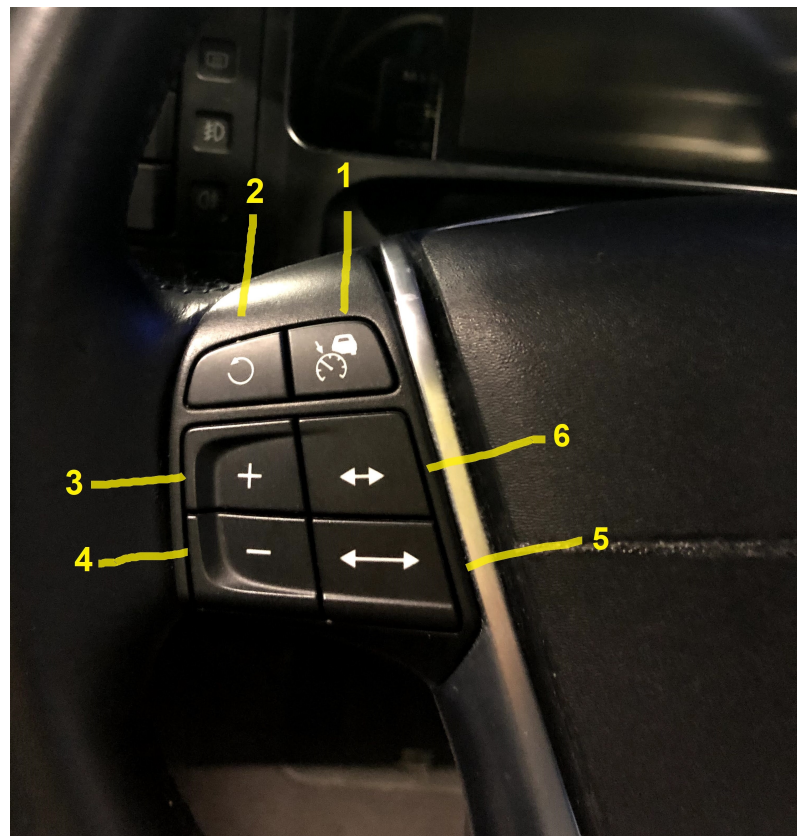
How to use:

Figure 4.5: Adaptive Cruise Control

- 1 - Enabling/Disabling the system
- 2 - Return to the pre-established speed
- 3 - Increase the speed (+ 5km/h)
- 4 - Decrease the speed (- 5km/h)
- 5 - Increase the time interval between the car and the car in his front (+ 0.5s)

- 6 - Decrease the time interval between the car and the car in his front (- 0.5s)
- 7 - Target speed
- 8 - Time interval target
- 9 - Real distance to the vehicle in front
- 10 - System enabled indication

Regarding the number 7, 8, 9 and 10, those are indications that are represented in the panel, along with the speedometers.

4.2 Important uses

There are multiple purposes of a simulator, it can be used to perform, test and evaluate experiences, which when practised in the real world would be very dangerous. Besides, it also may be used to different purposes such as entertainment or training. It is not new that F1 drivers use simulators to practise their expertise and skill. Concerning the DriS simulator, it already has been used to perform multiple scientific studies regarding driver behaviour in specific driving experiences, including driving with alcohol effect or somnolence. It was already used to study the ergonomics of some road infrastructures and also the human-computer interaction. Regarding road safety, there were developed several projects, among them the European project Haste (Human Machine Interface And the Safety Traffic in Europe, Haste) and the research project “Tempo de Colisão e Sinistralidade Rodoviária” (A. M. G. Jacob and others, 2012; A. Jacob Rodrigues, 2008), whereas in the ergonomics field were conducted studies to evaluate the impact of multiple audio and visual stimulus created by the information systems of the vehicle in the driving performance (Eg. In Vehicle Information System, IVIS, Global Positioning System, GPS)) (Pereira, Simões, Rodrigues, Leitão, Campos, Bruyas, 2008 a; Pereira, Simões, Rodrigues, Leitão, Campos, Bruyas, 2008 b). Moreover, about road engineering it can be highlighted the SafeSpeed project (Melo, Lobo, Couto, Rodrigues, 2012) which study how the speed drivers chose to practise was influenced by the geometric characteristics of the road. More recently were conducted studies in the psychology field, where it was observed the driver’s behaviour when submitted to the alcohol, drugs and tiredness effect (Durães, 2016) and also regarding the treatment of driver phobias in tunnels (Claudio, Carmo, Gaspar, 2014).

4.3 Simulation

To conduct scientific studies in driving simulators it is required a beforehand preparation of the road environments. This, accordingly the type of the experimental work, must strictly obey to the study specifications. An example of such specifications may for example be the signalization included, how many and how to dispose road crossings and roundabouts, the pavement type, among others. Furthermore, the study may demand a pretty long road extension, reaching several dozens of kilometres and it should be paid special care in the preparation of this environments with special

and complex specifications. In addition, must be created a set of information regarding the simulation environment well structured and organized. In order to perform experimental tests there were implemented road environments, with road models obtained of project data, traffic events and autonomous vehicles in both sides of the road. Therefore, there are different existent road environments available to use and to validate the improvements/changes made on the simulator. If the existing models do not fulfill the requisites needed to the experience, it can be easily modified. Simulation environments can be specified from data acquired from existent road networks, and defined using WDF (Word Description Format) files. One of the greatest advantages when compared to the real driving is the ease that the information about all the trip can be monitored. It is possible to save all the information about the experience, not only about the self vehicle but also about all the other intervenients in the simulation, like the autonomous cars. The data is recorded in real time and writted to a binary file than can be converted to a text file.



Figure 4.6: Simulation room - view from behind

4.3.1 Simulated traffic

Simulated environment traffic is composed of a many instance of autonomous vehicles. The implementation of autonomous vehicles is carried out using an integrated model, which allows these vehicles to work on three different modes:

Being absolutely autonomous: The vehicle has a t-moves polyline movement, which is composed by a polyline and a chosen speed to the vehicle perform.

Following a route imposed by an external entity: In this case, the car has a slave polyline movement which is characterised by a polyline and a pointer to one master vehicle. Then, there is a file that indicates the position of this car relatively to the master car in the polyline, in other words, in that file there is the distance that they must keep between each other. So, if the master car (usually the driven car) increases speed, then all the vehicles of this type increase speed in the same way.

Having any intermediate behavior between the 2 previous modes: This one is a hybrid mode in which sometimes the car behaves one way and other times may have a different behaviour.

4.4 Architecture

Images presented on the screen are generated by a graphic workstation that also holds the scene database and performs the simulation tasks. The graphical workstation is running a GNU/Linux OS in a PC desktide computer with the following requisites:

Processor - Intel Core i7

Memory - 16GB DDR5

Graphics card - Nvidia GeForce 1050

Disk - 2x 240GB SSD

Operating System - Fedora 31 GNU/Linux

This workstation is connected to the sound and cockpit PC by a private local area network, using an Ethernet Switch D-link 1010G. The virtual environment simulation toolkit used to develop DriS is Genes, which allows real-time image and sound generation, together with some movement and behavior controllers.

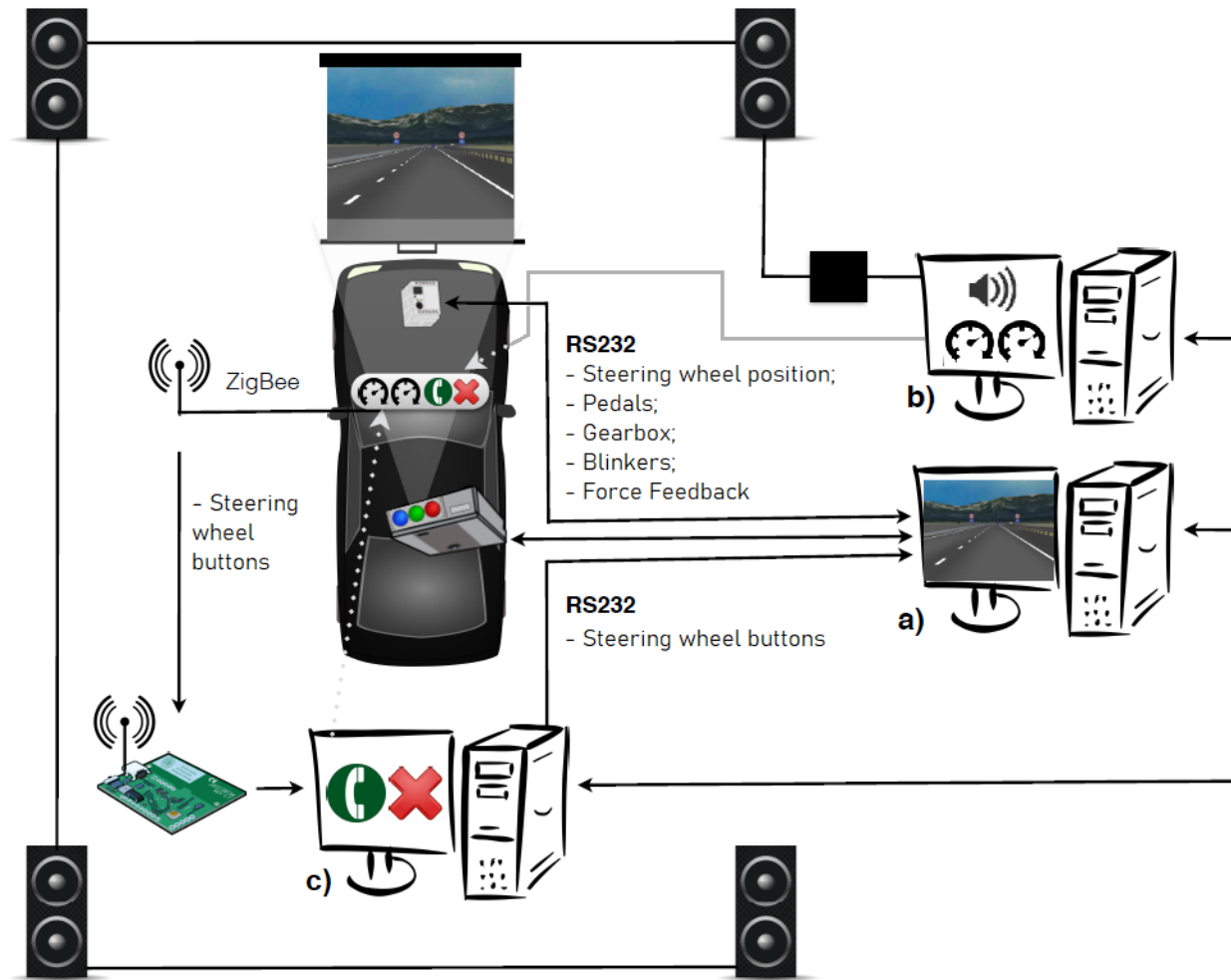


Figure 4.7: DriS initial architecture (Adapted from [15])

Under the hood of the car it is located an equipment to simulate the force feedback and a PLC to gather information about :

- Pedals position
- Steering wheel rotation
- Gearbox position
- Blinkers

This information is delivered to the computer a) by an RS232 cable. In addition, there is also communication in the opposite direction since the computer tells the PLC what it needs to know to perform the accurate force feedback on the steering wheel. This has a huge realistic impact on the DriS performance, since the force-feedback generated is very similar to the one felt in our cars

and vary with pretty much the same factors. This is not the only information gathered from the cockpit, it is also collected information about the ignition key position and about the steering wheel buttons, more precisely, about the adaptive cruise control (ACC). This ACC info is transferred to the the computer c) through ZigBee and then to the computer a) by a RS232, where that info is going to be processed. The image that the driver sees in the canvas is generated by the computer a) and it is sent by a VGA cable until it reaches a conversor where the 3 RGB channels are selected and from there sent to the projector by coaxial independent cables.

Whereas in the interior of the car there are two different digital displays, one to represent the speedometer and the rpm gauge, and another to act as the infotainment system of the car. Regarding the first one, it is in the computer b) that the dash application is launched and its output is sent by VGA to the display placed in the car. Concerning the second, the infotainment is reproduced in computer c) which is located in the car's trunk, and it allows the driver to perform/answer calls or to receive notifications. The computer b) is also responsible to launch the soundserver application, reproducing the sounds of the experience such as the engine sound or braking sound, for example.

All the computers communicate with each other, and that communication is established by ethernet network, using TCP/IP and UDP protocols. As it has been said, the computer a) is responsible for all the simulation logic, whilst the computers perform their assigned tasks as explained above, accordingly the information received by the computer a).

This processes, such as, the car controllers, the sound or the cockpit instrumentation are connected with the main simulation through the pretty much same architecture design. They are implemented following a client/server architecture, in which, the main simulation is the client and the sound application, cockpit instrumentation and car controllers work as server applications.

Each time we run the simulation in the terminal we can add options by introducing some flags as it will be shown in the table 4.1. Based on this options the due connections are established.

Table 4.1: CMD command flags

Flags	Action
-T	To add options following the T
-s	To innit the server/client sound connection
-p	To innit the server/client panel connection
-w	To innit the serial port connection with Volvo's cockpit
-j	So that the view window opened do not get fullscreen
-S	To run in slave mode
-M	To run in master mode
-W	To innit the serial port Logitech controller connection

Through the following client/server connections represented in the figure 4.8, it is sent the needed data to maintain the sound and panel applications synchronised with the simulation at the moment.

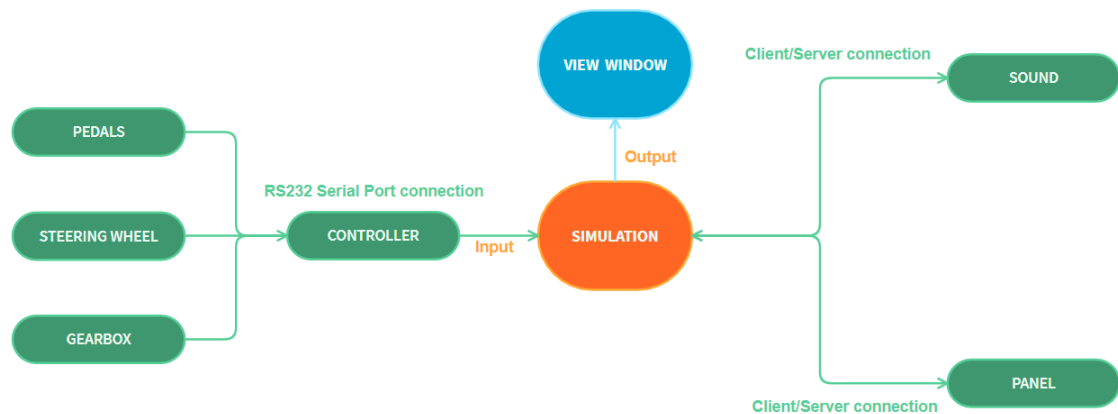


Figure 4.8: Simulation's peripheral connections

As an example, the data sent through the client/server connection to the panel, which means to the gauges is grouped in this struct 4.1.

```

1 typedef struct tactualstate {
2     int speed, rotation, temperature, fuel, indicators, gear;

```

```
3  int acc_on, acc_speed, acc_dist, acc_fcdist;  
4  } tactualstate;
```

Listing 4.1: Speedometer Data Frame

So, if in the command line, if after the -T was added a p, then a socket was open and as long as the simulation is running, the connection to the panel is still on and exchanging the data specified above in the struct tactualstate, the actual speed, engine rotation, engine temperature, how much fuel the car has, if it has any indicator on and in which gear it is at the moment. The panel application is continuously rendering a virtual panel that shows that information.

In the command line it is also pointed a configuration file, which specifies what road has to be loaded, which cars are going to exist in the simulation, their positions and how they will behave. There, are loaded as well other underlying characteristics such as the hills, the sky, the rails, the warning signs and other simulation elements. That file is read and parsed, initializing elements and building the simulation world and its inter-dependencies like, for example, autonomous cars that move accordingly the operated car. How is that made? In the definition of that autonomous car is given a file that contains some points representing relative distances to the operated car.

Basically, the simulation is a representation of objects and its positions. The focus of that representation is the controllable car, which is also our point of view.

The simulation opens up a graphic window that has two modes: a predefined view pointing to the road from the car's hood, which give us the driver perspective, and a free mode where we can change the position of the camera with the aid of the mouse. To exchange between this two modes must be pressed the C key on the keyboard.

According the inputs reproduced in the steering wheel, pedals and gearbox of the Volvo, and received by the simulation application, through some logic and physic conditions programmed, the simulation will start moving the car and visually reproduce the inputs in the graphic window, which means we are driving.

The logic and physic mentioned right above include some rules and attenuation factors so that the experience feels the closest possible to a real driving experience. In order to perform that, abrasion is considered, as well as the car's weight among other factors that are not of our interest. The acceleration performances are closest to that real Volvo as they were used similar engine torques to all the rpm, and were considered the gearbox ratios. It is also considered if the car is going up or going down, so that gravity has an effect in the simulation as well. Moreover, there is also a device simulating the force-feedback of the steering wheel. This was all made in name of getting the closest possible to a real experience.

In order to the readers better understand how the simulator worked when we first met this project, we are uploading here the picture 4.9 that graphically exemplifies is.

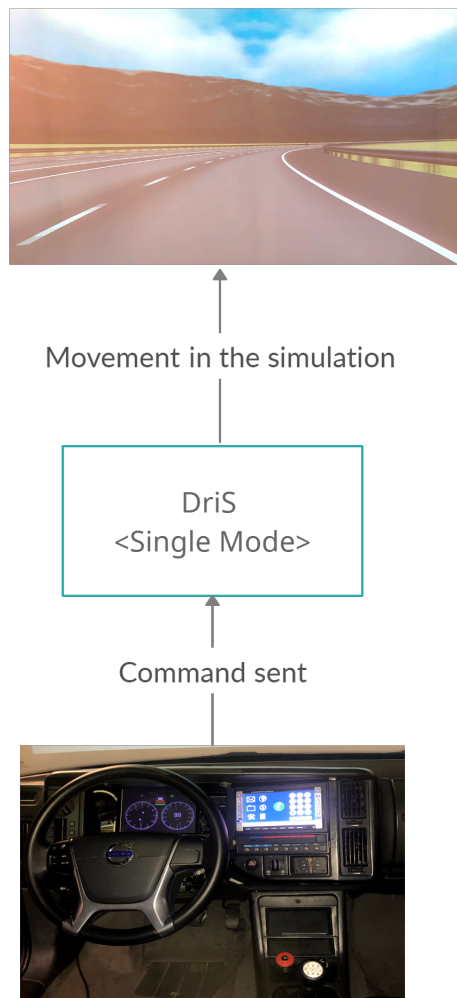


Figure 4.9: DriS - Single mode

The picture portrays what happens when we launch the simulator in the terminal in the single mode, typing -Tw to open the serial port for the controllers. A view window was open representing the simulation environment with our car stopped in the road. If we went to the Volvo and drive it, that information was passed as it was already explained before, a movement in the simulation was provoked and consequently the car moved. The computer running the <Single Mode> was responsible for doing whether the iteration of all the cars on the simulation environment whether of the rendering of the point of view.

4.5 Start up procedures

In this topic there is presented a resume of how to start up the simulator in the single mode.

Table 4.2: Turning on the equipment

Task	Explanation
Turn on the projector	Press the on button
Turn on the screen in the instrument panel	Press the button on and in one computer launch the dashboard program
Turn on the pedals, steering wheel and gearbox sensors	Couple the plug that comes out of the front of the car in the socket Steer the wheel all the way to the left Pop the hood of the car Turn the 3 visible switchers on, which will not only enable the pedal, steering wheel and gearbox sensors but also the car alimentation which is located in the trunk In the controller, chose the parameter 6.30 using the buttons and put it on. By doing that you are enabling the steering wheel force feedback.
Turn the computers on	In the control room, turn on the interface box (beige) to the current, turn the speakers and the 3 computers (a), b) and c) on: Computer a) - This computer has as his main goal to generate the road environment. Computer b) - This computer is connected to the instrument panel and used to reproduce it. What reproduces in this computer, reproduces in the instrument panel. Besides, it also is in charge of the sound. Computer c) - This computer has as his main goal to generate and manage the infotainment system. Computers d) and e) - Initially used to perform developments, but now the d) is used to run the DriS in the slave mode, associated to the Logitech controller. After that, in the computer a), you should surf into the folder where the road model pretended is, and type "driS model.wdf", model being the name of the road environment desired. If data recording is pretended must add this parameter "-U" to the command above mentioned. Then, this data recorded since the moment the driver begins his driving experience, will be saved in the following directory "/home/driSdev/onyx/DriSData". The simulation data are saved in a binary file that must be converted using the "makeDataDriS NameOfbynariFile NameOfTxtFile" command in order to be legible. In the computer b), you should run "./dash" to initialize the speedometer and execute the soundserver application as well.

Chapter 5

Project - Distributed Collaborative DriS

The distributed collaborative DriS is a project developed on top of the existing DriS driving simulator to improve it and take it to the next level when it comes to immersiveness and reality of the experience. The DriS was a driving simulator able to support just one user at a time, which would seat on the instrumented Volvo 440 and drive in the road environment chosen with other simulated autonomous vehicles. The DriS after the conclusion of this project is a driving simulator able to support more than one user driving in the same road environment at the exact same time alongside with the simulated autonomous vehicles. This is a major improvement because it will turn possible the coexistence of two or more human drivers able to interact and see each other in the environment, which will lead to a much more real experience rather than the previous monotonous experiences with some SAV.

5.1 Architecture

Having in mind what was said in the intro of this chapter, I think it was perceptible that such task is not easy to perform and requires a very good understanding of the whole project, and due to its complexity, must be very well analysed so that a solution that meets all the requirements can be found. As this was a project with very little documentation, a lot of time was disposed to understand the project and to gain the confidence to start doing improvements. So, the next subsections are organized in a way in which everything plays along and that makes sense to reader's comprehension. It will be presented the requirements, followed by a use cases explanation, moving on to a set of possible solutions and finishing by explaining in great detail the implementation.

5.1.1 Requirements

So that no feature falls by the wayside and also to enlight the reader a bit about what to expect, a table of requirements 5.1 is presented below with the the major ones.

Table 5.1: Requirements

Requirements	Description
R1	The simulator must be able to be driven in the singular mode
R2	Enable the simulator to be driven by a set of wheel and pedals (Logitech)
R3	The simulator should be driven with any of the equipment in spite of the mode chosen
R4	Having two machines running the simulator, the master must be able to send the info about all the moving objects to the machine working in slave mode
R5	The machine acting as slave should be prepared to receive and parse the information sent by the master machine
R6	The slave mode running machine must update the object list with the received information
R7	The slave machine must be able to drive his associated drivable car
R8	The slave machine must send the information (P and R) about its drivable car to the master working machine
R9	The master must be prepared to receive and parse the info about the the car controlled by the slave execution
R10	The master must update the object list with the received information
R11	Both machines must open a view window with a camera located at the POV of the respective drivable car

Basically, after all the implementations, the simulator has to maintain the option of being driven in the single mode, just like it was before (**R1**). This requirement is fulfilled as if in the run command we do not specify any flag -S or -M to chose whether Slave or Master mode, it runs in Single mode.

Regarding the **R2**, it was needed to configure and extract the values of the position of the pedals and of the steering wheel, so that we could use them in the DriS, by injecting them on the same spot where we used to inject the Volvo's ones, enabling that way, to drive with the Logitech set regardless of the mode chosen, whether being in the Single mode or in the Slave mode (used when collaborative driving is intended) **R3**.

Here **R4**, we are saying that having two computers running the DriS simultaneously, the machine working as Master should be able of sending the data frame containing all the cars and their respective P's and R's to the one working as a slave.

Consequently, the slave should be able to receive that data frame and parse it so that the info can be successfully used in the simulation (**R5**).

After that, it is intended that the slave updates his object list with the new P's and R's received for each of the objects (**R6**).

Concerning the **R7**, and having present that the **R2** is already fulfilled, it is important that the DriS launched in slave mode has a car with a type of movement that includes being driven by a cockpit.

The car will start being driven but now, his new position and orientation must be communicated to the master DriS (**R8**), so that it can distribute it across all the "slave machines" in the network, assuring that all of the machines running the DriS simultaneous can be synchronised, allowing the users to drive in the same collaborative shared environment.

Besides distributing it, the master also has to receive the information sent by all the slaves in the network, parse the information and update his own list of objects (**R9** and **R10**).

Then, in order to the users to interact with the environment, each machine that has a controllable car must has its own view window representing the POV of the correspondent drivable car (**R11**), abstracting the user from all of the previous proceedings.

5.1.2 Use Cases Diagram

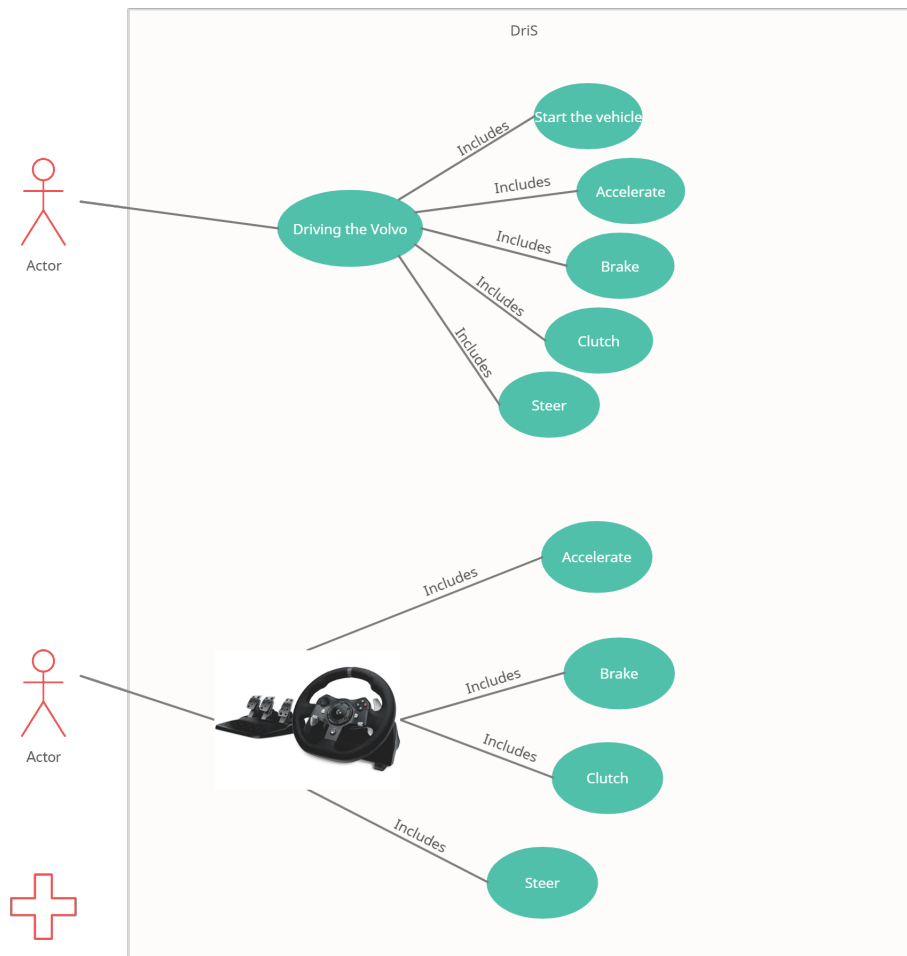


Figure 5.1: DriS Use Cases Diagram

In the diagram shown in the picture 5.1 we can notice a very simple diagram that intends to explain that how multiple persons (at least two) can interact with the driving simulator in the collaborative mood. The idea was that the simulator stayed the most flexible possible, and so, it is also possible for the user to drive alone using whether the instrumented Volvo, or using the Logitech wheel and pedals. I think, although simple, it successfully shows that when there are two users in action, one drives with the Volvo and the other with the Logitech cockpit. Also, we leave a plus sign there to represent that it is possible to add more users, if in the future, the DriS purpose take that direction. However, in case of a different set of wheels and pedals, it would require to create or find some drivers so that Linux can read his values, as it was done for this Logitech set. It is also possible to notice that when using the set of wheel and pedals there is no need to start the ignition, and we do not have so many features as we have in the Volvo, but so that we can understand this differences there is a better diagram below 5.2.

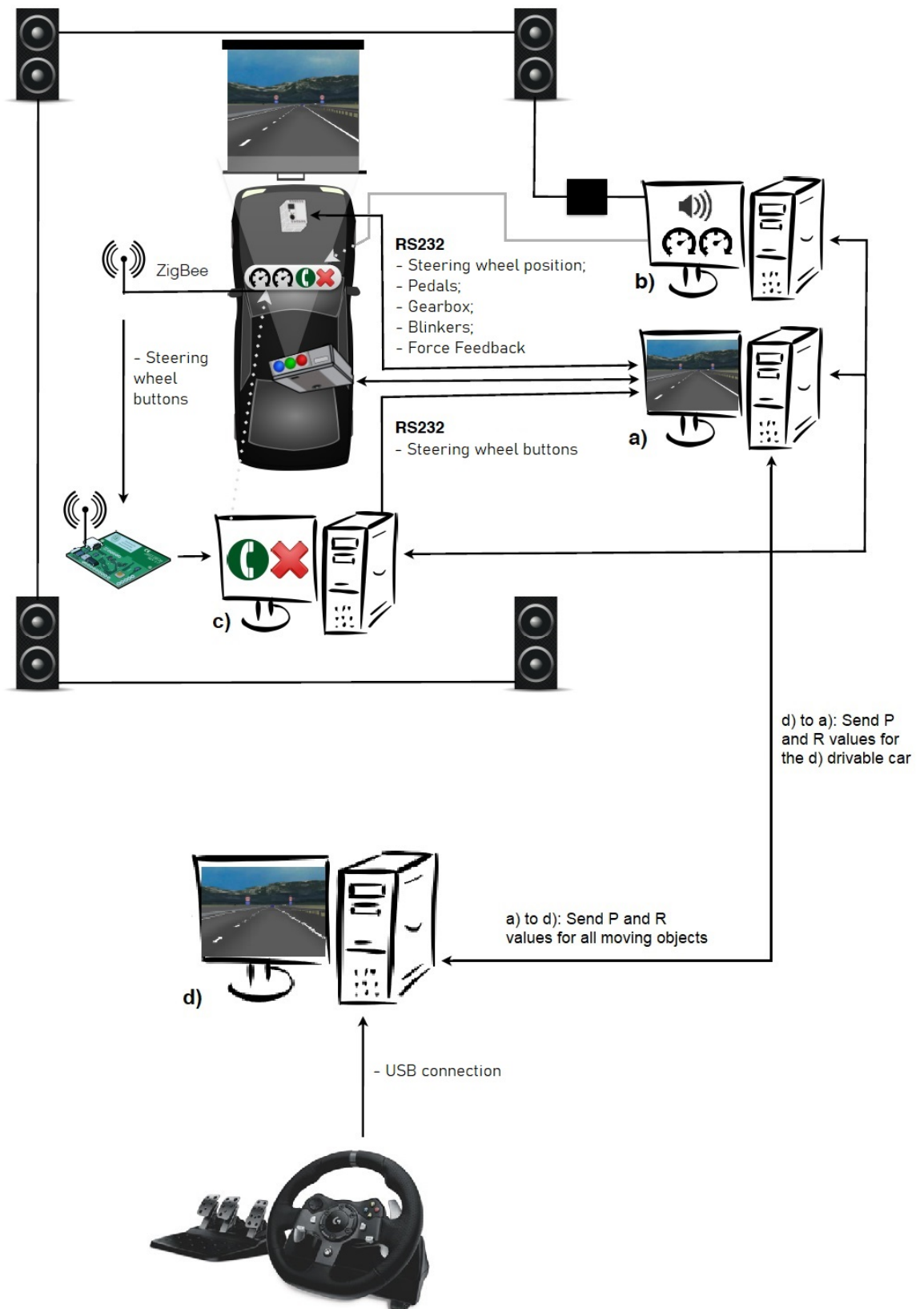


Figure 5.2: Implemented Collaborative Physical Architecture

Through this diagram, it is possible to understand all the links between all the physical components of the DriS essential to perform a simulation with two human drivers, one in each driving station.

In that diagram, it was considered that the computer a) launched the DriS running as master and the computer d) other instance of DriS running in the slave mode. Relatively to the notes about the connections between the computer a) and d), the a) sends the P and R values for all moving objects existing in the simulation environment, whereas the computer d) sends the P and R values regarding the car that it controls in the simulation to the computer a).

As you may notice, only the Volvo's driving station has sound, speedometers, blinkers, ignition, gearbox and adaptive cruise control.

So that there is no mistake, the view on the screen of the computer a) and in the screen in front of the car are the same, because as I said before they are connected. However, the view on the screen of the computer d) is another point of view, it is the one from the other drivable car, but it should be enlightened that the simulation environment is the same.

5.1.3 Possible solutions

Since the main objective was to implement a real-time shared environment where multiple human users could interact, which could arise multiple problems, it was essential to think and balance well the architecture to choose. We could think of 3 main possible architectures to implement as it is possible to observe in the following figures.

These architecture solutions make use of concepts and themes approached in the chapter 2 as here we have a peer-to-peer solution, and two master-slave-based architectures. It was important to study those concepts as understanding them better helped us choosing the best path to follow.

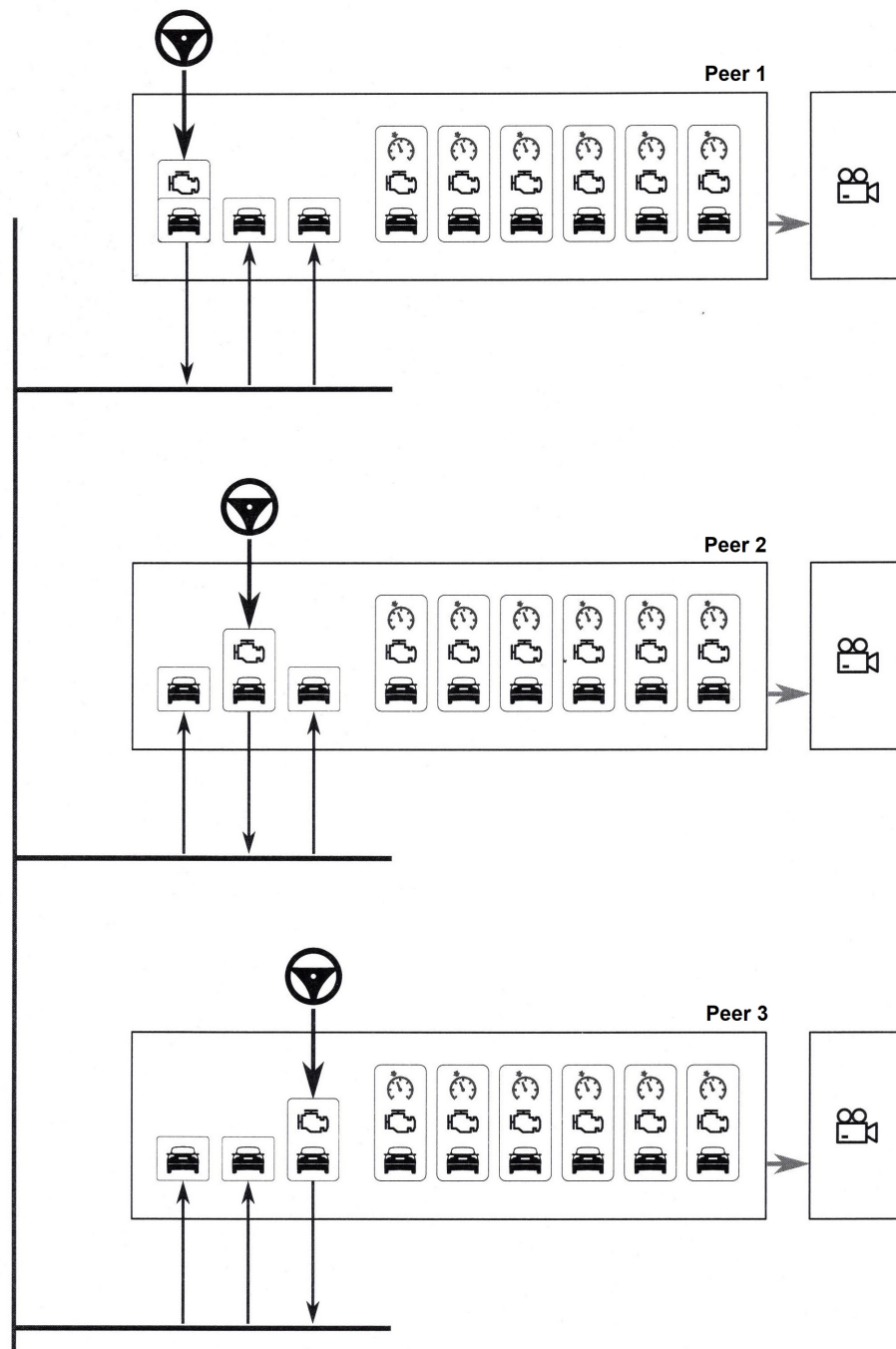


Figure 5.3: Peer-to-Peer possible architecture

Regarding the solution presented in the diagram 5.3, it was a strong possibility of implementation, mounting a peer to peer system, in which each peer had a drivable car, a set of autonomous

vehicles and info about the cars controllable by the other peers. As we can confirm in the diagram, each peer was sending his own drivable car (P and R) to the other peers on the network and receiving similar commands about others peer's cars, having each peer working as server and client at the same time. So far so good, however, we can also notice that no arrows come in or out of the simulated autonomous vehicles, which means that the peers are not sending the SAV vehicles between them. There is a reason for that, a peer-to-peer network is a decentralised system and as such, it has not a central or principal server. As a result of that, which of the peer's simulated autonomous vehicles should prevail? None, because if it is a peer to peer network, in the purity of the concept, that will not happen. The other question that yourselves may have in mind is why any SAV should be sent across the peers, or sent by one peer? The need to synchronise the simulated autonomous vehicles comes from the fact that if that does not happen, each of the peers may have their SAV's at different positions. Why does this happen? As we said in sub-chapter 4.3.1, simulated traffic can be of multiple kinds. The ones that follow a route imposed by an external entity, being the external entity usually the drivable car, depend on that car, as their behaviour is correlated with that car by a table of points representing the distances between that SAV and the controllable car. Therefore, as the different human users would drive differently, their correspondent SAV's would be at different locations. That can not succeed, otherwise the users would not be driving in the same collaborative shared environment and the whole point of this project would vanish.

Consequently, this architecture solution had to be dropped, and having in mind the problem that made us disregard the previous solution, another one was designed.

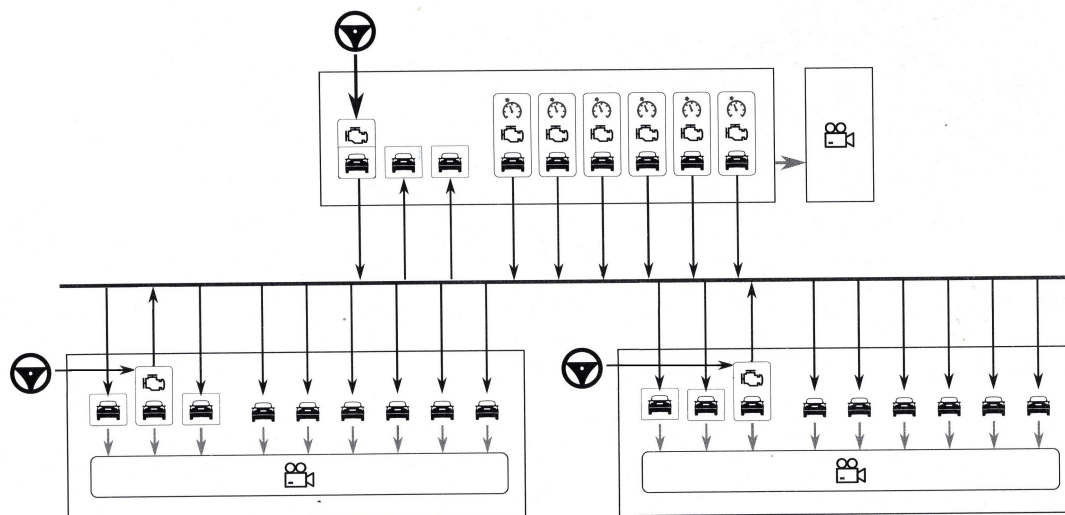


Figure 5.4: Master-Slave possible architecture

The solution presented in 5.4 is based on the master-slave principle in which one process is in

charge of controlling one or more other processes, which are denominated slaves. By analysing the diagram we can line off that there are three machines represented, one working as master and two other as slaves. Each of them has a controllable car, and each of the slaves is responsible for sending his car's new position and orientation values to the machine denominated as master. If close attention was given to that diagram, you might have understood that before reading this excerpt, as the only two arrows going up are the ones arising from the drivable cars. Then, the master machine distributes the position and orientation of all the drivable cars, including the one that it controls, across all the slaves in the network. Nevertheless, these are not the only cars propagated to the network, the master also sends the simulated autonomous vehicles to the slaves present on the networked, and by doing that, assures the synchronism of the simulation for all the participants and avoids the problem of the SAV's being in distinct positions for each of the users. After balancing the pros and cons of this architecture, we decided that this was the course to go.

It is important to say that, despite using the terms master and slave, this architecture does not represent the purity of the master-slave concept, since the slave is also able to send information. However, as the master coordinates and guarantees the system's harmony, we decided to still use this terminology throughout the document.

Another aspect interesting to discuss is the possibility of the master having an associated drivable car that he can control and send to the slave machines as well when distributing the info. In the diagram above, this possibility is graphically represented. Further down, that will be discussed and the option chosen will be declared along with the reasons of that choice.

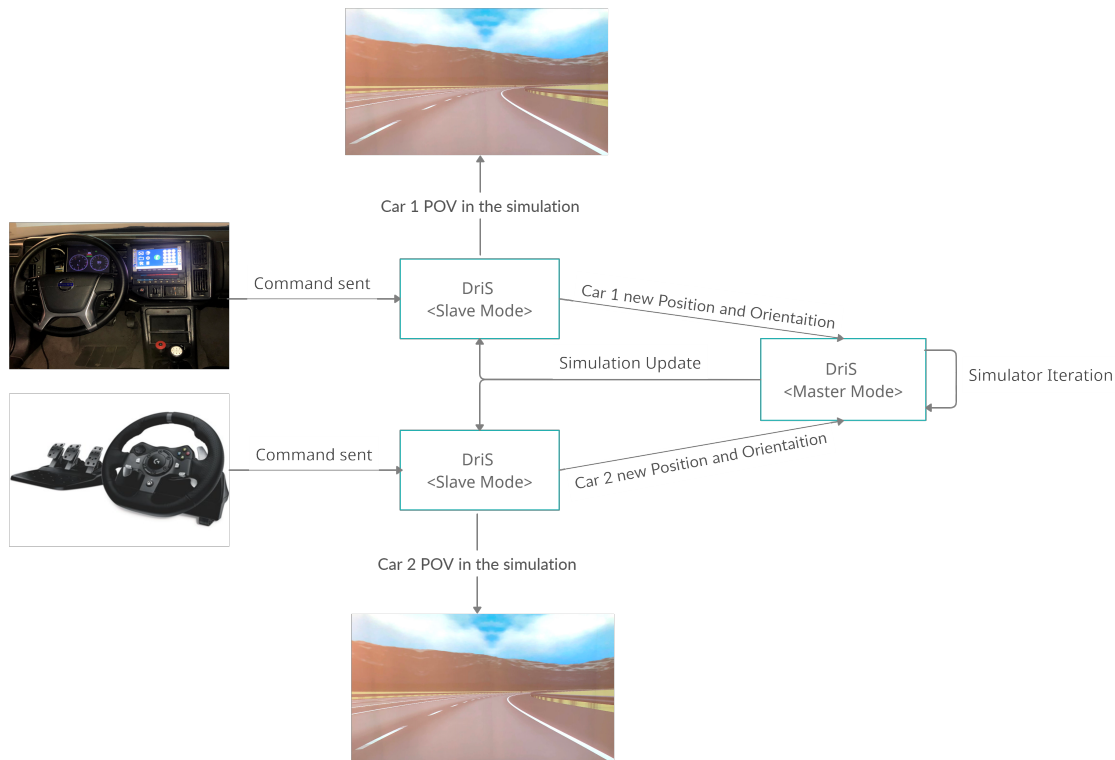


Figure 5.5: Collaborative DriS diagram proposal

In the figure 5.5, there is a representation of how the simulator would work in case the master did not have a controllable car. We would have one machine running the simulator in the <Master-Mode> and two other machines running in the <Slave Mode>. Each Slave Mode running machine had its controller connected, whether it be the Volvo or the Logitech set of wheels and pedals. In this scenario, the master machine would not open a view window because it would only be entitled to synchronise all the moving objects and distribute them.

It is important to mention that, the simulation iteration represented in the diagram 5.5 only refers to the update of the position and orientation of all the vehicles on scene, because all the image rendering is done by the machines working in slave mode and regarding its own point of view.

Whereas in the figure 5.6, a representation of the simulator's operation by having the master machine also controlling a drivable car is shown. In this architecture, the master machine, performs exactly the same actions as it does in the previous architecture with the plus of also being responsible for controlling a car. Basically, if we chose this architecture we may have more implementation work but we are simplifying the needed structure to achieve our goal, or at least reducing by one, the number of machines needed to perform a simulation in the collaborative mode.

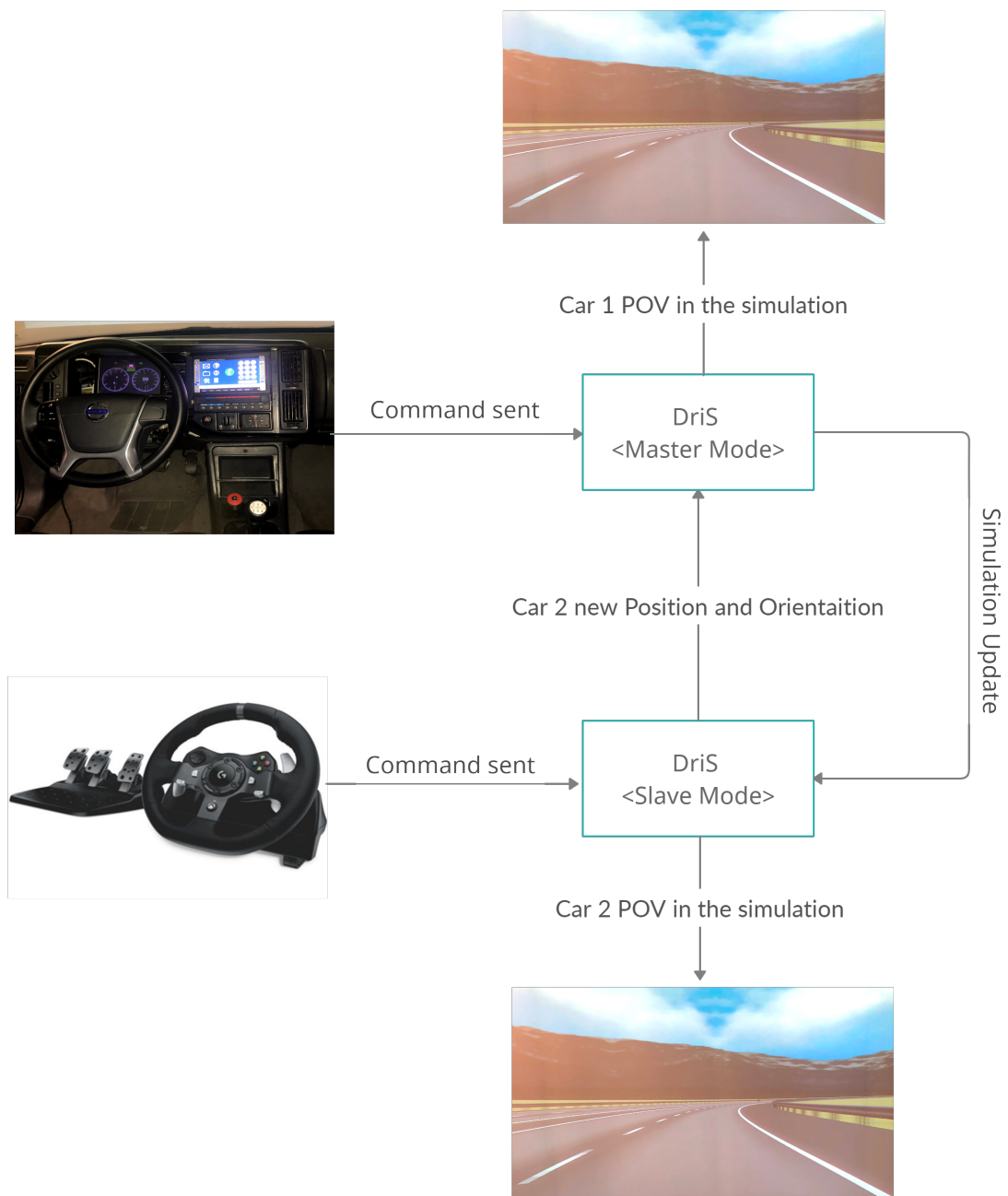


Figure 5.6: Collaborative DriS chosen implementation diagram

5.2 Implementation

Having the previous master-slave solutions in mind, the one where the master also controls a car was chosen because it contributes to simplify the resources needed to perform the exact same operations, achieving the same goals, exchanging a smaller amount of information, therefore helping

us improving the performance of the simulator. And, the performance of this simulator is very important and we must think on her every choice we made, otherwise the final result can be severely negatively affected by that.

Therefore, the master machine when doing the simulation iteration, besides updating P and R of all the vehicles on scene, also has to do the image rendering of its associated drivable car, as the slave machine, regarding its car's point of view.

However, this implementation can not be straightforward, due to his huge complexity, involving practically the whole simulator structure/code, and to simplify that, some intermediary steps were chosen. We are talking about implementing a Master-Slave architecture that was able to start the communication between 2 machines running the simulator in one of the directions. In other words, we started by considering having a Master computer/process and a Slave computer/process. There was initialized a UDP client in the "Master" machine responsible for sending the data to the "Slave" machine, which has a UDP Server listening at a specified port. So, first we worried about the master being able to send all the vehicles P's and R's to the slave. After achieving this stage, we moved on to the implementation of the communication in the other direction, so that the slave could send the info regarding its own controllable car.

Since the main goal was to enable two humans in different driving stations to drive in the same shared environment within the lab, the implementation made is communicating by unicast, because there are only two machines exchanging information. In the future, to fulfill the possibility of enabling the communication between more driving stations, it only has to be used a multicast address rather than the IP address of the machine to which we want to send the information.

Based on the figure 5.6, it is possible to understand that a lot changed, the DriS architecture's complexity raised. In that diagram we see a machine running the simulator in the <Master Mode> and another machine running in the <Slave Mode>. Both machines have an associated controller connected, whether it is the Volvo or the Logitech set of wheels and pedals. In the case represented we decided to pair the master with the Volvo's cockpit and the slave with the Logitech. Each of the controllers sends his own data for his connected computer. In the case of the Volvo's cockpit, the data frame sent is described in the following code excerpt 5.1.

```
1 typedef struct CockpitData_t {
2 int wheel;
3 int accelerator;
4 int brakes;
5 int clutch;
6 int gear;
7 int digitals;
8 } CockpitData_t;
```

Listing 5.1: Volvo's Cockpit Data Frame

The Logitech set, by now, is sending less data, as we are considering that the car associated to this controller is automatic and for that reason there is no need to send data concerning gears. Nonetheless, we are still sending the clutch pedal, so that if someday someone wants to start using the steering wheel paddles, they just need to worry about these two inputs. So, regarding the

Logitech it is being sent steering wheel position, accelerator, brakes and clutch values. Besides the case of someone wanting in the future, to make use of this set to drive a manual car, there is a lot of other buttons that can be used to simulate blinkers, ignition, the adaptive cruise control and other stuff they may remember.

Then, the computer executing the slave mode, by receiving the parameters enunciated before, calculates the next position (P) and orientation (R) of the respective car. After that, sends to the computer running in the master mode, sending a command that specifies its own drivable car and the values accordingly the coordinates x,y,z for the position (P) and orientation (R).

The DriS running in the master mode calculates the P and R of all the simulated traffic, and updates in the object list the position and orientation of all the cars, such as the simulated autonomous vehicles and also the two cars controlled either by the "slave machine", with the values received by the UDP request, either the car controlled by itself (Volvo controllers).

```

1 void Update(void) {
2     for(ObjLinkedList *obj_ptr=this ; obj_ptr!=NULL ; obj_ptr = obj_ptr->Next()
3     ) {
4         if ( obj_ptr->ExistElement() ) {
5             if ( ! obj_ptr->body ) fprintf(stderr,"Void obj body\n");
6             obj_ptr->body->Update();
7         }
8     }
9     if (UseSounds)
10        oSom3D.SendObjSounds (NULL);
11 }

```

Listing 5.2: How all the objects are updated

In the piece of code above enunciated 5.2, it is shown how the master does it, it has a linked list of objects that is traversed and for each object it is called an Update() function that as the name advances, updates the object position and orientation.

This update() function has a vital role in this project as it is responsible for not only updating the objects one by one by calling the Move() function that accordingly the movement type of the selected object performs a determined action, but also for calling the function that will send to the "slave machine" the position and orientation of each of the objects, as the linked list is traversed. About the Move() function it can be said that there are eighteen different types of movements, but it is not our focus of enlight right now.

```

1 void ObjectType::Update()
2 {
3     #ifdef _DEBUG_
4         char *name = GetName();
5         if (name) printf(" Updating Object '%s'\n",name);
6         fflush(stdout);
7     #endif
8     if ( movement != NULL )
9     {
10        movement->Move(*this);

```

```

11  #ifdef _DEBUG_
12      printf("      Object '%s' moved. Movement name='%s', type=%d\n",
13          GetName(), movement->GetName(), movement->type;
14          fflush(stdout);
15  #endif
16  #ifdef V_ANIM
17      vanim->wheelsAngSpeed = 1.;
18      vanim->steeringAngle = 0.;
19      vanim->updateWheels();
20  #endif
21  if ( setupMode==Master || setupMode==Both ) {
22      sendToSlaves();
23  } else if( setupMode==Slave && movement->internal_movement){
24      SendToMaster(); //req1
25  }
26  //printf(" Updating ObjectType '%s' sound  \n",GetName());
27  if ( UseSounds && sound )
28      oSom3D.SendObjSounds(this);
29  #ifdef _DEBUG_
30      printf("      Object '%s' Updated\n",GetName());
31      fflush(stdout);
32  #endif
33  }

```

Listing 5.3: ObjectType Update Function

It is also reason of detachment, that when the machine that call this function 5.3 is working in Slave mode, it is verified if the movement of the selected object at the moment has the bool `internal_movement` as true. It is only in this case, that the machine running in <Slave mode> performs the sending of its respective drivable car position and orientation to the machine working as master. So, that bool when true represents that the associated movement is from a drivable car controlled by a slave mode running DriS. These are the only movements that the "slave machine" has to send to the master one, that is, the movements of that drivable car. All the other movements are responsibility of the master and she receives them from the master.

So far, the reader must already know what is an internal movement, however does not know how each of the machines loading the same configuration file know which is the car that they must control, and that is what it is going to be exposed. In the picture 5.7 there are represented two movements, one to each of the drivable cars. The first called as Guiado is respective to the master's controllable car and the second named CarroMovE6 is the one associated to the slave's drivable car. Nevertheless, so that the slave's machine could read the CarroMovE6 as its drivable movement, and the master's machine as a movement that it is going to receive as an external command, some pre-processing directives was used. So, if it is the slave reading this excerpt of the configuration file, it is going to read the first definition of the CarroMovE6, which has the class defined as `TMOV_INTERNAL TMOV_DRIVE`. Whereas when the master loads this part of the configuration file it is going to parse the second definition which has `TMOV_EXTERNAL` defined as the class of this movement.

```

; Movimento guiado com o volante Volvo

DEF_MOVEMENT Guiado
  CLASS TMOV_DRIVE
  ROAD a25_2012
  MODEL toyota
  COCKPIT volvo
  lixo
END_MOVEMENT

; Movimento guiado com o volante Logitech

#ifdef SLAVE
DEF_MOVEMENT CarroMovE6
  CLASS TMOV_INTERNAL TMOV_DRIVE
  ROAD a25_2012
  MODEL toyota
  COCKPIT logitech
  lixo
END_MOVEMENT

#else
DEF_MOVEMENT CarroMovE6
  CLASS TMOV_EXTERNAL
  ROAD a25_2012
END_MOVEMENT

#endif

```

Figure 5.7: Definition of a movement as internal movement

Thanks to this workaround, we can use the same configuration file for both machines, and at the same time achieving the expected result. The master will load the Guiado movement as its associated drive movement, and for this reason being able to control the car that has the Guiado movement associated, and load the CarroMovE6 as an external movement, and for that reason it will not do anything to the car who has that movement associated but instead it will be expecting to receive a command about that car. The slave mode execution of DriS will load every movement that does not has TMOV_INTERNAL as class, as an external movement, that is, as movements it will receive from the master, and will load the CarroMovE6 movement as an internal drive movement, which will trigger the parser to mark that movement with the `internal_movement` as true, enabling the slave execution to drive the car associated to that movement. This will be also crucial to line off what is the object that must be sent to the master.

```

1 if ( (strmatch=strstr(line,"CLASS")) )
2 {
3     int classtype, cr, internal;
4     cr = sscanf(strmatch+strlen("CLASS "),"%d",&classtype);
5     if( classtype == 718){
6         cr = sscanf(strmatch+strlen("CLASS "),"%d %d", &internal, &classtype);
7         InternalMovement = true;
8         move_ptr->Element()->internal_movement = InternalMovement;
9     }
10    if(cr != 1 && cr !=2){ //|| strcmp(internal, 718) != 0 Verificacao caso a 1
        palavra nao seja TMOV_INTERNAL
11        fprintf(stdout,"Movement: Error reading ClassType for movement '%s'\n Error
        Line: '%s'\n",
12        name, line);
13    }

```

Listing 5.4: Load Movement Function Excerpt

The listing 5.4 shows a small part of the LoadMovement() function, which is the function in charge of parsing the movements from the configuration file and accordingly to each of the 18 movement types, identify which is the type of the movement being parsed at the moment, and perform the correspondent actions. In the part of the function that this excerpt exhibit, it is searched through the movement definition present in the configuration file, line by line for the word CLASS, because as we have seen before, it is through the class that the movements are identified. When found, the double in front of the word CLASS is stored in a variable named classtype. If you remember the previous figure 5.7 in front of the CLASS was placed a text and not a double. However, some macros were defined to replace that texts for numbers, and the TMOV_INTERNAL was substituted by the number 718. As you see in the line 5, it is verified if the classtype of the movement being analysed is the 718 (TMOV_INTERNAL), and if it is the case, it means that we are dealing with an internal movement. As such, that line is parsed again and 2 doubles are extracted, the first, that we already know that is the 718 is stored in the internal variable and the true type of movement is stored in the classtype variable, which is in this case TMOV_DRIVE. Besides, the InternalMovement flag is marked as true, and the internal_movement attribute of this Movement also enabled as true. Between the lines 10 and 13 it is just made an error check to assure that if there are more than 2 words in front of the CLASS to informs us that an error reading the movement happened.

That InternalMovement flag is the responsible for following in that function, the movement of the slave drivable car do not be treated as if it was an ExternalMovement. Because there is a condition specifying to proceed that way only when the setupMode of the execution is Slave and the InternalMovement is marked as false. When the setupMode is Master, Single or the InternalMovement is true (for the case of the setupMode being slave and the movement being an internal movement of the slave's drivable car), it is done a switch with the classtype and according to that classtype, the correspondent movement is created.

In this section it will be presented a class diagram that highlights the most important classes created to perform the implementations. It does not illustrate all of them, which means that were performed implementations in another classes, however these are the essential to explain how the communication was established and achieved. Regarding the `_ObjectType` class, it is represented with the attributes and methods that can have some relevance to the implementation but not with all of them as it is a very extensive class.

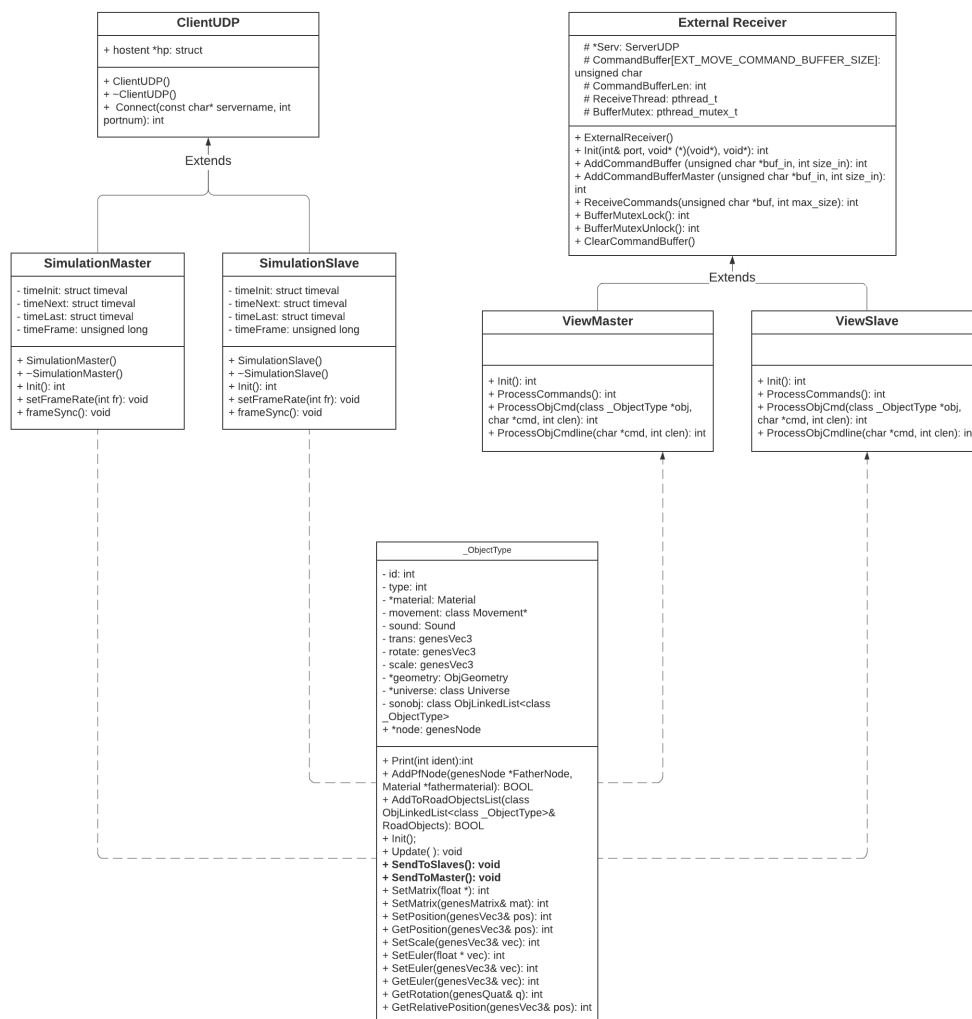


Figure 5.8: Class diagram

In the diagram we can see some of the components developed to achieve the solution, which we will see in more detail in the remaining course of this chapter:

1. **SimulationMaster Class** - Responsible for creating the connection to the slave and by calling the `Send()` function;

2. **ViewSlave Class** - In charge of receiving and parsing the information sent by the SimulationMaster, finishing with the update of all the objects;
3. **SimulationSlave Class** - Responsible for creating the connection to the viewMaster, to send the information about his own (slave) drivable car;
4. **ViewMaster Class** - Must receive and parse the information sent by the SimulationSlave, updating that object.

Regarding the **SimulationMaster Class**, here it is initialized a UDP client that establishes connection with the server specified at the specified port number.

```
1 int res = ClientUDP::Connect("192.168.81.24", SLAVE_PORT);
```

The IP mentioned in the above line of code corresponds to the IP of the computer used to launch the simulator as slave. Firstly, it is created and allocated a socket and then established the connection.

Through this connection, each one of the cars on the simulation environment will be sent to the ViewSlave class of the execution that took off as slave.

Following, in the listing 5.5, there is an excerpt of the most relevant code of this function.

```
1 void ObjectType::sendToSlaves()
2 {
3     #ifdef _DEBUG_
4     printf("      Enviando state do obj '%s' para Slaves\n",GetName());
5     fflush(stdout);
6     #endif
7     char *mBuffer;
8     mBuffer = (char *)malloc(200);
9     genesVec3 oPos, oEuler;
10
11     int mlen;
12     GetPosition(oPos);
13     GetEuler(oEuler);
14
15     snprintf(mBuffer,200,"O%s:P%f,%f,%f:R%f,%f,%f\n", name, oPos[0], oPos[1],
16     oPos[2], oEuler[0], oEuler[1], oEuler[2] );
17     mlen = strlen(mBuffer);
18     printf("\nComando enviado: %s", mBuffer);
19
20     simulationMaster.Send(mBuffer,mlen);
21     #ifdef _DEBUG_
22     printf("      Enviou state do obj '%s' para Slaves\n",GetName());
23     fflush(stdout);
24     #endif
25     free(mBuffer);
26 }
```

Listing 5.5: Send object(car) to slaves

In the line 15 of this code excerpt, there is build a command containing the name of the object, in other words the car to which the following values for position and orientation belong. That values represented in the code by oPos[0], oPos[1], oPos[2], oEuler[0], oEuler[1] and oEuler[2] are the position and orientation of the car in the x, y, and z axes, respectively, and they were consulted before in the code excerpt lines 12 and 13, through GetPosition() and GetEuler() functions, which consult the object matrix at the moment and get the most recent values. Then, the command is saved in the mBuffer.

After the command is ready, this class calls the Send() function, having the mBuffer, already filled with the command previously built, as parameter.

In the other side of this communication, there is the **ViewSlave Class**, which receives each of the commands sent by the simulationMaster, and parse them.

When a ViewSlave class is initialized, it is associated a port number in which this class is going to receive data, it is called the init() of a ExternalReceiver class instance and it is also launched the ViewSlaveReceiveFunc() on a thread. In this function 5.6 the spots where we should pay most attention are in the line 11 and 17.

```

1 void* ViewSlaveReceiveFunc(void* move_ptr)
2 {
3     printf("\n\nexecuting ViewSlaveReceiveFunc()\n\n");
4     // Executed in a concurrent pthread.
5     ExternalReceiver *Mptr = (ExternalReceiver *)move_ptr;
6     Mptr->ClearCommandBuffer();
7     while( ! SimDone() )
8     {
9         int BufLen = 0;           // Bytes recibidos
10        unsigned char LocalBuffer[8000];
11        BufLen = Mptr->ReceiveCommands(LocalBuffer, sizeof(LocalBuffer));
12        printf(" ViewSlaveReceiveFunc: cmd received, %d bytes\n", BufLen);
13        if (LocalBuffer[0]==TEXTMOV_THREAD_EXIT)
14            return (void *)TEXTMOV_THREAD_EXIT;
15            // return adds some extra cleanup over pthread_exit()
16        if (BufLen>0)
17            Mptr->AddCommandBuffer(LocalBuffer,BufLen);
18    }
19    return NULL;
20 }

```

Listing 5.6: Receive function

While the simulation is not over, this function will not end, and it will be continuously receiving the commands sent by the simulationMaster, and saving them in a LocalBuffer. Yet, in the same loop it is called the AddCommandBuffer function (5.7) that is responsible of adding the command received and stored in the LocalBuffer to the CommandBuffer, which is a protected attribute of the ExternalReceiver class.

```

1 int ExternalReceiver::AddCommandBuffer(unsigned char *buf_in, int size_in)
2 {

```

```

3  #ifdef _DEBUG_
4      buf_in[size_in] = '\0';
5      fprintf(stdout, "Adding command to buffer: (%d) '%s'\n", size_in, buf_in);
6  #endif
7      BufferMutexLock();
8      if ( CommandBufferLen+size_in+1<EXT_MOVE_COMMAND_BUFFER_SIZE ) {
9
10         memcpy(CommandBuffer+CommandBufferLen, buf_in, size_in);
11         CommandBufferLen += size_in;
12         BufferMutexUnlock();
13     }
14     else {
15         BufferMutexUnlock();
16         fprintf(stdout, "Error: CommandBuffer Overflow in
17 ExternalReceiver::AddCommandBuffer. %d+%d>%d\n", CommandBufferLen, size_in,
18 EXT_MOVE_COMMAND_BUFFER_SIZE);
19     return 0;
20     }
21     return CommandBufferLen;
22 }

```

Listing 5.7: Add Command to CommandBuffer

This CommandBuffer is of great importance owing to the fact that it contains several commands, working as a queue, and it is from this buffer that the functions responsible for processing the information remove command by command and parse them. In other words, this CommandBuffer is a queue where new commands are coming in and added in the back while the commands in the front of the queue are cleared as they are processed. So, what we are saying, is that this buffer is constantly being acceded by two functions, and we must assure that does not happen a simultaneous access. To guarantee that, we used mutexes as we can notice in the lines 7 and 12. The mutexes give us the guaranty that while locked, this is the only function writing on the CommandBuffer. If we did not guarantee that, we could end up losing commands. Following, in the picture 5.9, there is an example of a possible command loss if we did not have used mutexes.

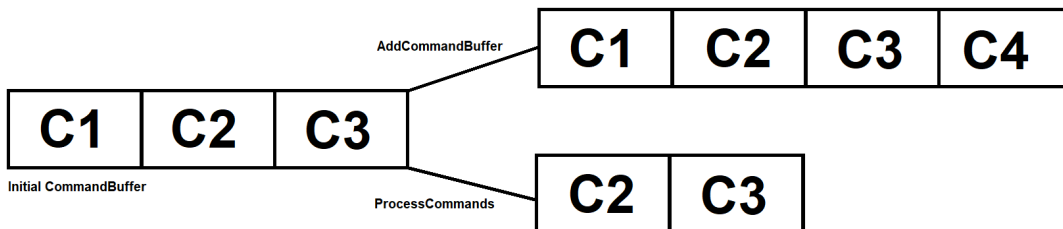


Figure 5.9: Possible command loss

To explanation purposes, lets consider that the CommandBuffer had 3 commands, C1, C2 and C3. The function AddCommandBuffer() starts writing on this buffer to add the command

C4, but before this function can finish, the function `ProcessCommands()` also starts writing in the `CommandBuffer` in order to process the first command, C1, and naturally removing it from the `CommandBuffer`. Meanwhile the `AddCommandBuffer()` finishes writing and immediately after the `ProcessCommands()` also finishes writing. What the `CommandBuffer` will contain in the end of this two actions is the result of the one finishing last, and as such, it will contain 2 commands, C2 and C3, and we have just lost the command C4. So, with the use of mutexes, each function can only access the `CommandBuffer` after the other has finished.

As we have advanced in the previous explanation, the next step is processing the commands received. The function responsible for that is the `ProcessCommands()` which is called within the `updateScene()` function.

In this function, the size of the `CommandBuffer` is verified and as long as it has commands, a break line is searched to segregate the commands from each other. Then, the first line corresponding to one command is copied to the `CmdLine`, string that will later be used by the `ProcessObjCmdline()` function. After, copying the first command of the `CommandBuffer` to the `CmdLine`, there is erased that first command from the `CommandBuffer`. To perform this operation, considering that we are writing on the `CommandBuffer`, the same procedure used in the `AddCommandBuffer()` function must be taken, which is using mutex locks to safely write on the buffer without occurring losses. Lastly, there is a call to the `ProcessObjCmdline()` function with the `CmdLine` as parameter, that is, with one isolated command. After this call, the `CmdLine` is cleaned, so that it can be used when the next command of the queue gets processed.

```

1 int ViewSlave::ProcessObjCmdline(char *cmd_line, int clen) {
2     assert(cmd_line[0]!='\0');    // Only implemented for ASCII protocol version.
3     cmd_line[clen] = 0;
4     char *oname = strtok(cmd_line+1, ":\n\r");
5     class _ObjectType *obj = world.FindObject(oname);
6     if ( obj == NULL ) {
7         obj = world.FindObjectType(oname);
8     }
9     if (obj == NULL) return -1;
10    genesVec3 v;
11    printf("    Processing commandline to object '%s':%s\n", obj->GetName(), cmd_line);
12    char *cmd = NULL;
13    while( cmd = strtok(NULL, ":\n\r") ) {
14        ProcessObjCmd(obj, cmd, strlen(cmd));
15    }
16    return 0;
17 }

```

Listing 5.8: Process one command

In the function 5.8, from the command received, it is identified the name of the object to which the values belong. Through the name, it is searched in the world which is the object which has that name, if not found, then it will search which is the `ObjectType` that has that name. After it is

found, the Object/ObjectType is stored in the obj variable. Lastly, it is called the ProcessObjCmd() function with the obj, the cmd and the lenght of the cmd.

The next function 5.9 is the really important one where the position and orientation values of the object are updated with the ones passed by the command. It is at this point that the object matrix is updated.

```

1 int ViewSlave::ProcessObjCmd(class _ObjectType *obj, char *cmd, int clen) {
2     if ( obj == NULL ) return -1;
3     genesVec3 v;
4     int res = 0;
5     switch (*cmd) {
6         case 'P':
7             res = sscanf(cmd+1, "%lf,%lf,%lf", &v[0], &v[1], &v[2]);
8             assert(res==3);
9             printf("      set pos %lf,%lf,%lf\n", v[0], v[1], v[2]);
10            obj->SetPosition(v);
11            break;
12
13            case 'R':
14                res = sscanf(cmd+1, "%lf,%lf,%lf", &v[0], &v[1], &v[2]);
15                assert(res==3);
16                printf("      set rot %lf,%lf,%lf\n", v[0], v[1], v[2]);
17                obj->SetEuler(v);
18                break;
19            default:
20                fprintf(stdout, "ViewSlave::ProcessObjCmd: Bad Object Command: '%c'\n", *cmd);
21            }
22            return 0;
23 }

```

Listing 5.9: Updates the object matrix values for position and orientation

Inside that switch, by analysing the cmd passed by the previous function, when a "P" is found we know that in front of that character there are 3 values describing the car's position according the x, y and z axes. Based on that, we store that 3 floats on a genes vector of 3 coordinates v. Then, we just need to call the _ObjectType method SetPosition() with the v as parameter. This method will update the object matrix with the new values for its position. The same happens when a "R" is found, however if that is the case, the _ObjectType method called is the SetEuler() with the genes vector v containg the 3 values for the orientation of that object accordingly the 3 axes. This method will upload the new orientation values of the _ObjetctType obj to its matrix.

By now, we have just explained how the communication was implemented in one of the directions, from the SimulationMaster to the ViewSlave. It was also added a sequence diagram where the sequence of function calls to perform a successful send and processing of one object command is represented 5.10. So far, how the master working machine sends the data about all the moving objects must be clear.

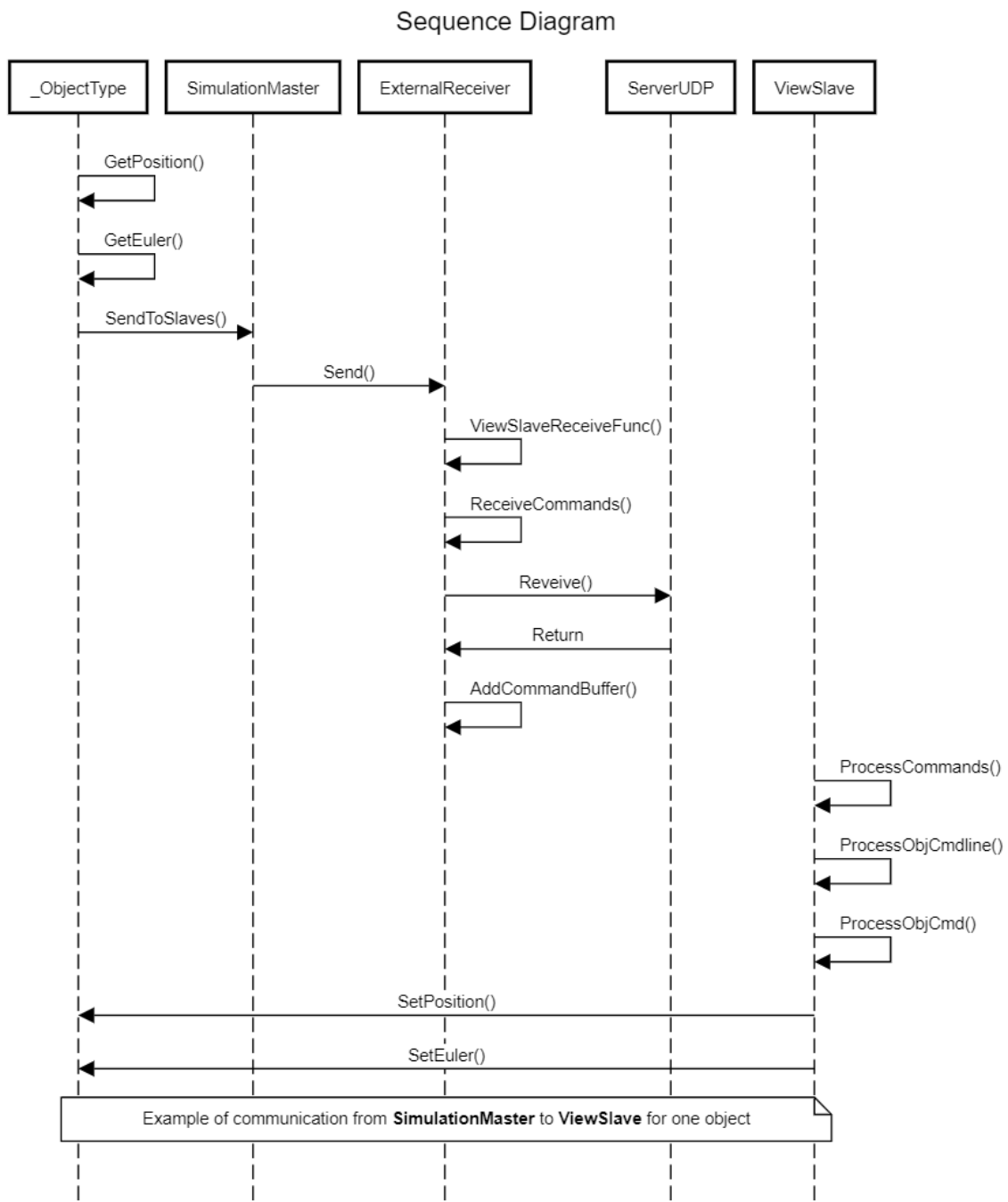


Figure 5.10: Sequence diagram - SimulationMaster to ViewSlave

Following it will be explained the other direction of the communication, how the slave working machine sends its drivable car to the machine working as a master.

Concerning the communication made in the opposite direction, we have the **SimulationSlave class** sending to the **ViewMaster class**. The logic of this communication is similar to the one previously implemented. The major difference is that, this is just used to send commands about one car, the one that the slave machine controls. So, the SimulationSlave class opens a socket with

the IP address of the computer running as master and its port number, so that further on this class can send commands to the ViewMaster class process. The key aspect of this communication is knowing when to communicate and that is made in the update function. If the computer is running the simulator in the slave mode, and the object to which the function Update() was called has a movement, and it has the bool internal_movement as true, then we send it to the master. This bool as we said before, it is only true when the movement refers to the slave's machine drivable car.

Otherwise, if any of the previous conditions is not fulfilled, the object is moved on that machine but not sent to the master, which is the case of all the other vehicles, the vehicle controlled by the master and the simulated autonomous vehicles which are all treated as external movements in what regards to the slave working machine.

Regarding that slave drivable car, it is called the SendToMaster() function, which like the SendToSlaves() function, gets the position and orientation of the object, builds a command containing the name of the object and the axes values for its position and orientation and sends it. In this case, the Send() call is made from the SimulationSlave class, which means it will be sent through the socket open by the SimulationSlave class. Then, the ViewMasterReceiveFunc which runs on a thread, will call 2 functions, one to receive the command and the other to add the command received to the CommandBuffer, which is an attribute of that instance of the ExternalReceiver. On a parallel thread, the ProcessCommands() function is processing all the commands from the receiving queue, that is the CommandBuffer, one command is passed to the CmdLine and that same command is erased from the CommandBuffer. It is then called the ProcessObjCmdline with the CmdLine, which is going to identify the object to which the name present in the command belongs, so that it can pass the object as parameter to the next function change his position and orientation. That function is the ProcessObjCmd(), which will update the object matrix with the new values received in the command, through the call of the object methods SetPosition() and SetEuler(). If you want a better understanding of this, please pay attention to the communication in the other direction, between the first two enumerated classes, already explained above, because the procedure is exactly the same and it was explained with better detail. Nonetheless, in the picture introduced below 5.11 a sequence diagram aims to clarify the sequence of function calls concerning the send of the slave's drivable car command.

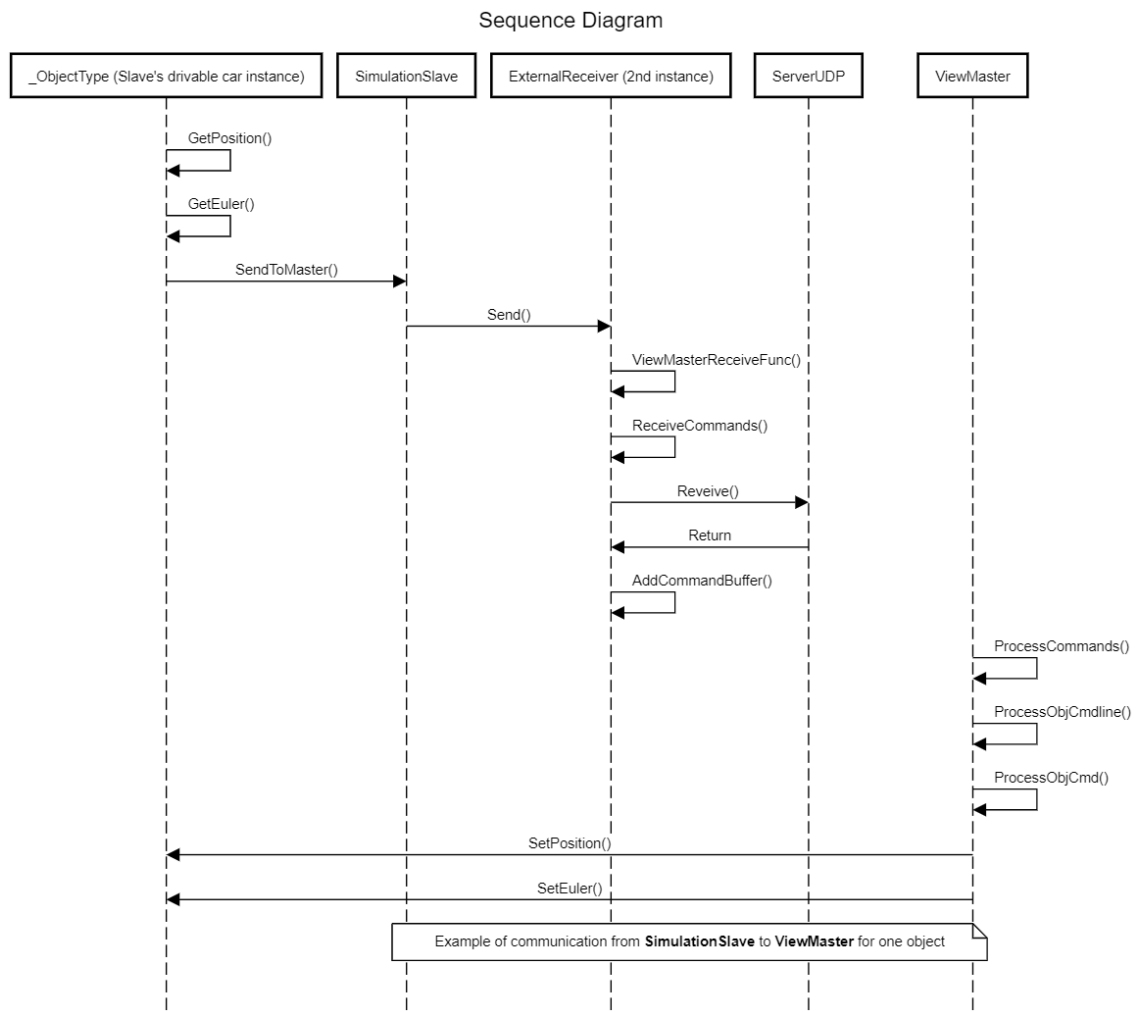


Figure 5.11: Sequence diagram - SimulationSlave to ViewMaster

5.2.1 Stream used and data flow diagram

So that it is more perceptible how are the commands sent between the machines an example of one is presented in the picture 5.12.

OCarro10Amb1A: P0.000045,0.000057,0.000000: R47.212464,42.787536,0.100000

Object name
Position
Orientation

Figure 5.12: Command example

The command is a string, that is analysed accordingly the punctuation signal : found, in charge of enabling the separation of the three fields. First, there is the name to which the values belong, then the axes values for position and lastly the axes values for orientation. The object name may seem odd, but the list of cars is defined in the configuration file, and some of them are grouped, in case of having the same movement type. So, the next command to be sent after the one used as example would be about the "OCarro10Amb2A" car.

In case we have 2 computers running the simulator, one in the master mode and another in the slave mode, there are lots of commands being sent by second. The amount of commands sent depend on the frame rate chosen. So that you can better understand who needs to send what, in one iteration of the program's main loop, below is introduced a data flow diagram 5.13 that aims to elucidate the reader on that subject.

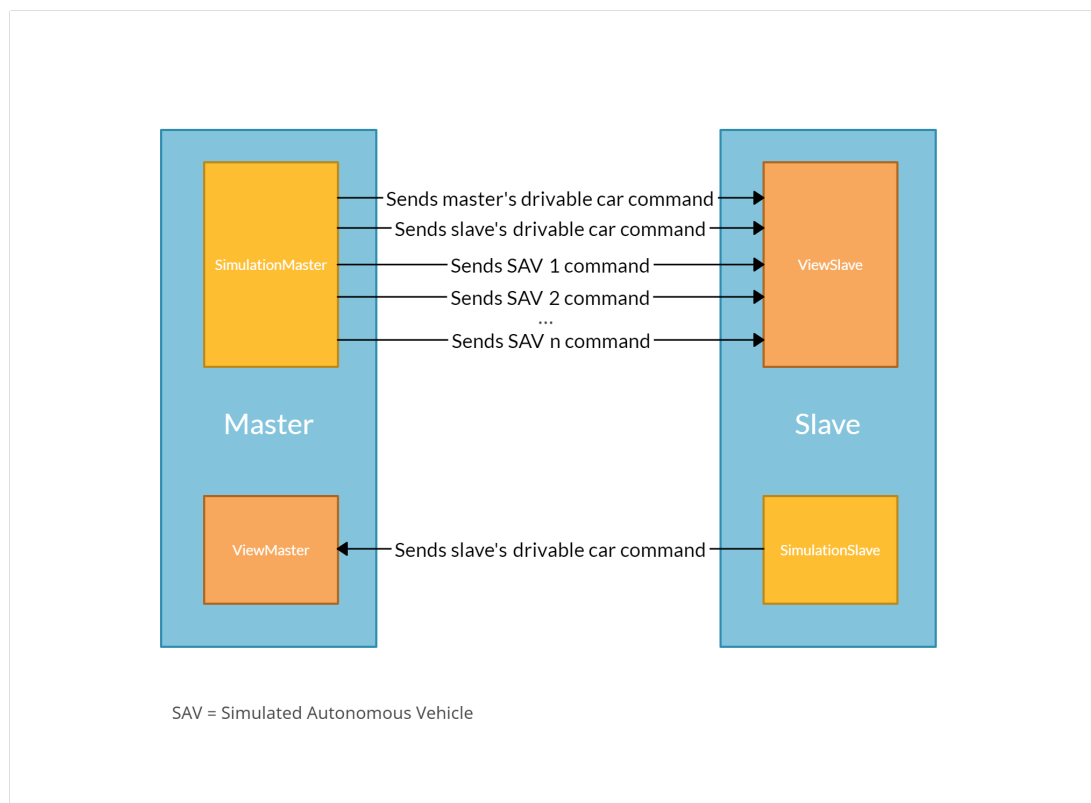


Figure 5.13: Data Flow diagram

In blue we have represented the 2 instances of the DriS running, one as master and the other as slave. Inside we have represented in rectangles the 2 major classes that each of them has. The master has a instance of the SimulationMaster class and another of the ViewMaster class. The slave has a instance of the SimulationSlave class and another of the ViewSlave class. Those classes communicate accordingly the arrows represented in the diagram.

As it is possible to verify, whilst the DriS working in the slave mode only sends one command, referring to its controllable car, the DriS working in the master mode sends multiple commands, depending on the number of existing cars on the simulation environment. It sends a command about his own controllable car, sends a command about the slave running DriS controllable car (if there were 2 machines running DriS in the slave mode, it would send 2 commands), and sends a command for each of the simulated autonomous vehicles.

5.3 Validation

In order to validate the contributions of this project to the enhancing of the DriS driving simulator, we thought of taking an experience in the traffic laboratory of the civil engineering department, by having multiple people driving using the DriS. The experience's intended case scenario was having one person driving using the Volvo's cockpit and another using the Logitech set of wheel and pedals in the same shared environment.

Regarding the number of participants in this experience, we have not set a maximum, but in order to get viable results, the more that take part, the better. Our course of action would pass by sending an email to all the FEUP community asking for interested people to participate in the experiment. However, that would not fulfill a varied range of volunteers, as they would be most predominant engineering students. To fight this, we would reach some friends and some persons we knew that could widen the spectrum of participants in fields like the age and area of research/activity. After some interested volunteers manifest their accordance to take part in the experiment, we would start to group them in groups of two and set an appointment for each of the groups.

Each test session would need the expertise of two volunteers and would last more or less 35 minutes. First, we would introduce ourselves and present the facilities. After they get familiar with the facilities, the volunteers would be reminded about the context and purposes of this dissertation. To start, the fulfillment of an anonymous demographic and immersive tendency revealing questionnaire presented in the appendix A, first used by Witmer [38], would be required, so that we could know our sample and understand if they are persons who easily get involved in a virtual experience.

Explanations about the controllers will be just barely reminded as the persons selected to perform the experiments must have a driver's license and know how to operate a car. When it comes to the Volvo, her daily driving experience must be very similar, whereas regarding the Logitech set we can say that it is pretty intuitive. The road environment selected for one group must remain the same in the whole experience so that it does not play a role in the immersion felt by the users. However, between test sessions other road environments can be chosen, widening the range of road structures used in the experience.

Initially, the users would be asked to drive for 5 minutes in the single mode, which means, driving in a environment where the user is driving alone, without any other human driving in the same environment, in each of the driving stations. So, whilst one was driving using the Volvo's

cockpit in single mode, the other was also driving in single mode in a different road environment using the Logitech set. After those 5 minutes they must change of driving station and repeat the experience.

At the end of that individual experience that has the purpose of getting the users familiarised with the equipment and establishing a comparison point, the users would be asked to fill a survey regarding the immersion felt in the individual experience just realized (Questionnaire B). In this questionnaire there are some questions to review the performance of the simulator in that experience and others that address the immersion and how involved the user was during the experience.

Then, the users would drive in the collaborative mode, with 2 instances of the DriS running the same shared environment, one running as master and another as slave. Therefore, one of the users would use the Volvo's cockpit and the other the Logitech one, however at this time, they would be driving in the same shared environment. This experiment must have the same duration as the previous one, which means they should drive for the same time. After reaching the 5 minutes, they must exchange of driving stations and repeat the experience, like they did previously in the other experiment.

After they finish this experience, they would be asked to fill the same questionnaire again, the one realized after the individual experience, this time regarding the collaborative experience they would have just finished. This questionnaire is presented in the appendix B and it had as base the Witmer [39] technical report. The main interest of this questionnaire would be to check if the immersion increased when driving with another human driver present in the same environment, if they found the collaborative experience better than the individual one and, of course, to spot flaws or to point if there was something that they did not like, or that they would like to see improved. By the end of the experience, the user's presence and participation would be thanked.

In spite of the experience had been planned, the same has not gone forward due to the pandemic reasons. This experience would need the participation of at least 20 persons, of multiple ages and convenient from multiple places and activity areas. Besides, multiple simultaneous presences were needed at the laboratory and the need to change of driving station multiple times would highly increase the sanitary risk taken during the experience. For those reasons, we considered that it was wise not to go ahead with the experience. Another experience was thought, and it passed by testing with persons at their own homes. However, this would require that the person had a set of wheels and pedals and the project is not ready to be remotely accessed in the desired conditions, which leads us to the following chapter 6.

Chapter 6

Conclusions and Future Work

Nowadays, much attention is devoted to simulation environments, as companies know they can improve their products and refine their specs based on multiple experiments conducted in high reliable simulators. They are not using simulators anymore only to test the final products, they are using them since the early stages of the product creation owing to the fact they realise this way they could reach a much better final product while saving a lot of money in the process. Based on this, all the companies that can take advantage of a simulator start doing it and the ones that already did, invested in improving their existing simulator or even to replace it by a newer and more capable one.

The most desirable aspect of a simulator is the similarity to a real-life scenario, the closer the experience provided to reality, the better is the simulator. However, this has to be achieved without neglect other important characteristics of a good simulator such as the ease of use.

It is in the same line of thinking that this project's contributions improved the DriS driving simulator. However, the improvements were not targeted to the existing Volvo's driving station by increasing the degrees of freedom of the platform but rather to the DriS global architecture, as this work made it possible for 2 human users to drive and interact in the same collaborative shared environment using 2 different driving stations. The complexity of this work is vast as the simulator was never thought to accommodate two users. Therefore creating that possibility from such a dense project that was not designed to achieve it raises some obstacles.

This project's main goal was accomplished as it was made possible the collaborative driving experience of 2 users that drive in the same environment, using different driving stations. In order to reach that goal, several architectures were considered and it was implemented the one we think that best complies with all the project requirements. Accordingly the chosen architecture, it was implemented a bidirectional communication between two machines running the DriS simulator, and left the code prepared to accommodate a possible third user in the future. Despite not so much emphasis being given to this, it was also created and configured a second driving station composed by the Logitech G920 set of wheels and pedals, as you may have guessed.

6.1 Future Work

Since we were not able to go forward with the planned and previously presented experiences due to pandemic reasons explained in the end of the chapter 5, and as it seems that the end of this unfortunate era we are living is not near, we would like to suggest that for future improvement the DriS driving simulator should be prepared to be usable in the implemented collaborative mode from anywhere outside the laboratory, in other words, the suggestion is to make the DriS accessible from the comfort of the possible user's homes.

That would have a particular interest because it would allow the test and validation with volunteered users, who could give us important information about the DriS collaborative performance regarding their point of view, despite still living in the pandemic context. Based on that feedback some adjustments could be made.

Appendix A

Profile Characterization Questionnaire

Questionário de Tendências de Imerssividade

Questionário de Tendências de Imerssividade

Indique a sua resposta selecionando o valor apropriado na escala linear de 7 valores. Por favor considere toda a escala ao responder.

***Obrigatório**

Idade *

A sua resposta

Sexo *

☐ Masculino

☐ Feminino

Profissão *

A sua resposta

Você fica facilmente envolvido de forma intensa em filmes ou séries? *

Nunca 1 2 3 4 5 6 7 Frequentemente

☐ ☐ ☐ ☐ ☐ ☐ ☐



<https://docs.google.com/forms/d/e/1FAIpQLScEcN79f81ih1z-8rFlmXs9KJqKCc2MsVFrGxGMzdrZ0DG32Q/viewform>

1/6

Figure A.1: Immersive Tendency Questionnaire - page 1

Questionário de Tendências de Imersividade

Alguma vez ficou tão envolvido num programa de televisão ou livro que pessoas tiveram dificuldade de obter a sua atenção? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Freqüentemente

Quanto se sente mentalmente alerta neste momento? *

1 2 3 4 5 6 7

Nada alerta ☐ ☐ ☐ ☐ ☐ ☐ ☐ Totalmente alerta

Alguma vez ficou tão envolvido num filme que deixou de estar consciente de coisas a acontecerem à sua volta? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Freqüentemente

Com que frequência se identifica com um personagem de um filme ou série? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Freqüentemente



Figure A.2: Immersive Tendency Questionnaire - page 2

Questionário de Tendências de Imersividade

Alguma vez se envolveu tanto num jogo que computador que sentia que estava dentro do jogo ao invés de estar apenas a mover os controlos e a olhar o ecrã? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Frequentemente

Quanto se sente hoje em boa forma física? *

1 2 3 4 5 6 7

Nada bem ☐ ☐ ☐ ☐ ☐ ☐ ☐ Extremamente bem

Quanto bom é a bloquear distrações externas quando está envolvido em alguma coisa? *

1 2 3 4 5 6 7

Nada bom ☐ ☐ ☐ ☐ ☐ ☐ ☐ Muito bom

Quando está a assistir a desporto, alguma vez fica de tal forma envolvido no jogo que reage como se fosse um dos jogadores? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Frequentemente



Figure A.3: Immersive Tendency Questionnaire - page 3

Questionário de Tendências de Imersividade

Alguma vez fica envolvido numa divagação que deixa de reparar em coisas a acontecer á sua volta? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Frequentemente

Alguma vez tem sonhos que são tão reais que fica desorientado quando acorda? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Frequentemente

Quando está a pratica um desporto, fica de tal forma envolvido que perde a noção do tempo? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Frequentemente

Quanto bem se concentra em uma atividade que gosta? *

1 2 3 4 5 6 7

Nada bem ☐ ☐ ☐ ☐ ☐ ☐ ☐ Muito bem



Figure A.4: Immersive Tendency Questionnaire - page 4

Questionário de Tendências de Imersividade

Com que frequência joga computador? (frequentemente significa todos os dias ou de 2 em 2 dias, em média) *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Frequentemente

Alguma vez ficou animado com uma cena de perseguição ou luta na TV ou num filme? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Frequentemente

Alguma vez ficou assustado com algo a acontecer numa série ou filme? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Frequentemente

Alguma vez ficou apreensivo ou com medo, muito tempo depois de ver um filme de terror? *

1 2 3 4 5 6 7

Nunca ☐ ☐ ☐ ☐ ☐ ☐ ☐ Frequentemente



Figure A.5: Immersive Tendency Questionnaire - page 5

Questionário de Tendências de Imersividade

Alguma vez ficou tão envolvido em alguma coisa que perdeu a noção do tempo?
*

	1	2	3	4	5	6	7	
Nunca	☐	☐	☐	☐	☐	☐	☐	Frequentemente

Submeter

Nunca envie palavras-passe através dos Google Forms.

Este formulário foi criado dentro de Faculdade de Engenharia da Universidade do Porto. [Denunciar abuso](#)

Google Formulários



<https://docs.google.com/forms/d/e/1FAIpQLScEcN79f81ih1z-8rFImXs9kJqKc2MsVFrGxGMzdrZ0DG32Q/viewform>

6/6

Figure A.6: Immersive Tendency Questionnaire - page 6

Appendix B

Experience Evaluation Questionnaire

Avaliação da Experiência

Carateriza a sua experiência no ambiente virtual selecionando a opção apropriada na escala linear de 7 valores. Por favor considere toda a escala ao responder, pois os valores intermédios podem ser aplicáveis. Não salte perguntas ou volte atrás para alterar uma resposta.

***Obrigatório**

Experiência Realizada *

☐ Individual

☐ Colaborativa

Quão foi capaz de controlar os eventos? *

1 2 3 4 5 6 7

Nada ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente

Como classifica a capacidade de resposta do ambiente às ações iniciadas (efetuadas) por si? *

1 2 3 4 5 6 7

Nada responsivo ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente capaz

Quanto natural lhe pareceram as suas interações com o ambiente? *

1 2 3 4 5 6 7

Extremamente artificial ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente natural

Figure B.1: Experience Evaluation Questionnaire - page 1

25/01/2021 Avaliação da Experiência

Quanto o envolveram os aspetos visuais? *

1 2 3 4 5 6 7

Nada ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente

Quão natural foram os mecanismos que controlaram os movimentos através do ambiente? *

1 2 3 4 5 6 7

Extremamente artificial ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente natural

Quão cativante foi a sua percepção dos carros a moverem-se no ambiente? *

1 2 3 4 5 6 7

Nada ☐ ☐ ☐ ☐ ☐ ☐ ☐ Muito cativante

Quão coerente foram as suas experiências no mundo virtual quando comparadas com o mundo real? *

1 2 3 4 5 6 7

Nada coerente ☐ ☐ ☐ ☐ ☐ ☐ ☐ Muito coerente

Foi capaz de antecipar o que iria acontecer após as ações que efetuou? *

1 2 3 4 5 6 7

Nada ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente

https://docs.google.com/forms/d/e/1FAIpQLSeymNlewAHAFbNqc1zHRQIV494sMI_gbYg-5nwWlaoFZYJVvW/viewform 2/5

Figure B.2: Experience Evaluation Questionnaire - page 2

25/01/2021

Avaliação da Experiência

Quão cativante foi a sua percepção do movimento dentro do ambiente virtual? *

1 2 3 4 5 6 7

Nada cativante

☐☐☐☐☐☐☐☐

Muito cativante

Quão envolvido esteve na experiência de ambiente virtual? *

1 2 3 4 5 6 7

Nada envolvido

☐☐☐☐☐☐☐☐

Completamente absorvido

Quanto atraso sentiu entre as suas ações e os resultados esperados? *

1 2 3 4 5 6 7

Nenhum atraso

☐☐☐☐☐☐☐☐

Longo atraso

Quão rápido se adaptou á experiência no ambiente virtual? *

1 2 3 4 5 6 7

Não me adaptei

☐☐☐☐☐☐☐

Menos de 1 minuto

Quanto interferiu ou o distraiu a qualidade de imagem de concluir a sua tarefa de condução? *

1 2 3 4 5 6 7

Nada

☐☐☐☐☐☐☐☐

Interferiu bastante

https://docs.google.com/forms/d/e/1FAIpQLSeymNiewAHAFbNqc1zHRQIV494sMI_gbYg-5nwWlacFZYJVvW/viewform

3/5

Figure B.3: Experience Evaluation Questionnaire - page 3

25/01/2021

Avaliação da Experiência

Quão bem conseguiu concentrar-se na atividade de condução em vez de nos mecanismos usados para realizar essas atividades? *

1 2 3 4 5 6 7
Nada ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente

O force-feedback gerado pelos volantes ajuda a confirmar as ações efetuadas? *

1 2 3 4 5 6 7
Nada ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente

Ficou tão envolvido no ambiente virtual que perdeu noção do tempo? *

1 2 3 4 5 6 7
Nada ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente

Ficou tão envolvido no ambiente virtual que não teve percepção do que se passava à sua volta? *

1 2 3 4 5 6 7
Nada ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente



https://docs.google.com/forms/d/e/1FAIpQLSeymNiewAHAFbNqctzHRQIV494sMI_gbYg-5nwWlacFZYJVvW/viewform

4/5

Figure B.4: Experience Evaluation Questionnaire - page 4

25/01/2021 Avaliação da Experiência

Sentiu-se confiante a utilizar as estações de condução= *

1 2 3 4 5 6 7

Nada ☐ ☐ ☐ ☐ ☐ ☐ ☐ Completamente

Pessoalmente, eu diria que esta experiência foi *

1 2 3 4 5 6 7

Desagradável ☐ ☐ ☐ ☐ ☐ ☐ ☐ Agradável

Sentiu algum efeito secundário negativo? Se sim, qual? *

A sua resposta

Falhas encontradas

A sua resposta

Sugestão de melhoria

A sua resposta

Submeter

Nunca envie palavras-passe através dos Google Forms.

Este formulário foi criado dentro de Faculdade de Engenharia da Universidade do Porto. [Denunciar abuso](#)

Google Formulários

https://docs.google.com/forms/d/e/1FAIpQLSeymNiewAHAFbNqc1zHRQIV494sMI_gbYg-5nwWiacFZYJVWw/viewform 5/5

Figure B.5: Experience Evaluation Questionnaire - page 5

References

- [1] US6862622B2 - Transmission control protocol/internet protocol (TCP/IP) packet-centric wireless point to multi-point (PTMP) transmission system architecture - Google Patents.
- [2] Process scheduler having multiple adaptive partitions associated with process threads accessing mutexes and the like. Technical report, mar 2006.
- [3] Who coined the term “virtual reality”? - virtual reality society, (accessed January 07, 2021). <https://www.vrs.org.uk/virtual-reality/who-coined-the-term.html>.
- [4] Three-tiered client/server architecture, accessed October 24, 2020. https://www.ibm.com/support/knowledgecenter/en/SSAL2T_9.1.0/com.ibm.cics.tx.doc/concepts/c_three_tierd_clnt_sevr_arch.html.
- [5] Vinod Anupam and Chandrajit Bajaj. SHASTRA-An architecture for development of collaborative applications. In *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE*, pages 155–166. IEEE Computer Society, 1993.
- [6] Steve Benford, Chris Greenhalgh, Tom Rodden, and James Pycock. Collaborative Virtual Environments. Technical Report 7, 2001.
- [7] Yahn W Bernier. Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization. Technical report, 2001.
- [8] Richard Carey. Collaborative work environment supporting three-dimensional objects and multiple remote participants. Technical report, jan 1996.
- [9] E. F. Churchill and D. Snowden. Collaborative virtual environments: An introductory review of issues and systems. *Virtual Reality*, 3(1):3–15, 1998.
- [10] M. Cline. *Virtual Reality: a Catalyst for Social and Economic Change*. Mychilo Cline.
- [11] Carlos Coelho, JG Tichon, Trevor J Hine, GM Wallis, and Giuseppe Riva. Media presence and inner presence: the sense of presence in virtual reality technologies. In *From communication to presence: Cognition, emotions and culture towards the ultimate communicative experience*, pages 25–45. IOS Press, Amsterdam, 2006.
- [12] Luís Henrique Maciel Kosmowski Costa, Serge Fdida, and Otto Carlos Muniz Bandeira Duarte. Incremental service deployment using the hop-by-hop multicast routing protocol. *IEEE/ACM Trans. Netw.*, 14(3):543–556, 2006.
- [13] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. Distributed Systems Concepts and Design 5ed. *Pearson*, 5th edition, 2012.

- [14] Jon. Crowcroft and Markus (Computer scientist) Hofmann. *Networked group communication : Third International COST264 Workshop, NGC 2001, London, UK, November 7-9, 2001 : proceedings*. Springer, 2001.
- [15] André Jesus de Carvalho Pinto. Atualização de simulador físico de condução automóvel por integração de sistema de realidade virtual. Master's thesis, 2016.
- [16] Ulrich Drepper. Parallel Programming with Transactional Memory. *Queue*, 6(5):38–45, sep 2008.
- [17] Leonard Ellis. It's time for ieee to retire 'master/slave' | ee times, (accessed January 19, 2021). <https://www.eetimes.com/its-time-or-ieee-to-retire-master-slave>.
- [18] Markus Esch, Jean Botev, Hermann Schloss, and Ingo Scholtes. P2P-based avatar interaction in massive multiuser virtual environments. In *Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2009*, pages 977–982, 2009.
- [19] DL Fisher, M Rizzo, J Caird, and JD Lee. Handbook of driving simulation for engineering, medicine, and psychology. 2011.
- [20] Alireza Mazloumi Gavgani, Frederick R. Walker, Deborah M. Hodgson, and Eugene Naliwaiko. A comparative study of cybersickness during exposure to virtual reality and “classic” motion sickness: Are they different? *Journal of Applied Physiology*, 125(6):1670–1680, dec 2018.
- [21] Chris Greenhalgh and Steven Benford. MASSIVE: A Collaborative Virtual Environment for Teleconferencing. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 2(3):239–261, sep 1995.
- [22] Mario A. Gutiérrez A., Frédéric Vexo, and Daniel Thalmann. *Stepping into virtual reality*. Springer London, 2008.
- [23] Elizabeth Landau. Tech confronts its use of the labels 'master' and 'slave' | wired, (accessed January 19, 2021). <https://www.wired.com/story/tech-confronts-use-labels-master-slave/>.
- [24] Joseph J Laviola. A Discussion of Cybersickness in Virtual Environments. Technical report.
- [25] Manolescu. United States (12) Patent Application Publication. Technical Report 10, aug 2007.
- [26] Gloria Mark and Alfred Kobsa. The effects of collaboration and system transparency on CIVE usage: An empirical study and model. In *Presence: Teleoperators and Virtual Environments*, volume 14, pages 60–80. MIT Press 238 Main St., Suite 500, Cambridge, MA 02142-1046 USA journals-info@mit.edu, feb 2005.
- [27] J.A. Marques and P. Guedes. *Tecnologia de sistemas distribuídos*. Tecnologias de informação. FCA-Editora de Informática, 1998.
- [28] Dmitri Moltchanov. Client/server and peer-to-peer models: basic concepts. Technical report, 2013.

- [29] Pito Salas, Jeffrey Beir, Glenn McDonald, Andy Dennie, Paul Kleppner, Neal Finnegan, Craig Morrissey, Alan Steele, and Patrick Crowley. (12) United States Patent (54) (75) (73) (21) (22) (51) (52) (58) (56) METHOD AND SYSTEM FOR PROVIDING A NETWORKED COLLABORATIVE WORK ENVIRONMENT. Technical report, jul 2001.
- [30] Cheryl Savery and T. C. Nicholas Graham. Timelines: Simplifying the programming of lag compensation for the next generation of networked games. *Multimedia Systems*, 19(3):271–287, jun 2013.
- [31] Paul M. Sharkey, Matthew D. Ryan, and David J. Roberts. Local perception filter for distributed virtual environments. In *Proceedings - Virtual Reality Annual International Symposium*, pages 242–249. IEEE Comp Soc, 1998.
- [32] Kay M. Stanney, Robert S. Kennedy, and Julie M. Drexler. Cybersickness is not simulator sickness. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 41(2):1138–1142, 1997.
- [33] Alexander Stiehle. Dynamic driving simulators: why they will make decisions about the audis of the future, (accessed January 07, 2021). <https://www.audi.com/en/career/working-world/dynamic-simulators-automotive-testing.html>.
- [34] SUTHERLAND IE. HEAD-MOUNTED THREE DIMENSIONAL DISPLAY. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*, volume 33, pages 757–764, New York, New York, USA, 1968. ACM Press.
- [35] Unknown. Car simulator (uolds), (accessed February 10, 2020). <https://uolds.leeds.ac.uk/facility/car-simulator/>.
- [36] Unknown. Unreal engine: The most powerful real-time 3d creation platform, (accessed February 10, 2020). <https://www.assettocorsa.it/competizione/>.
- [37] Unknown. Zero latency: The world's greatest virtual reality experiences, (accessed February 10, 2020). <https://zerolatencyvr.com/>.
- [38] Bob G. Witmer and Michael J. Singer. Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoperators and Virtual Environments*, 7(3):225–240, 1998.
- [39] Bob G. Witmer and Michael J. Singer. Measuring presence in virtual environments. Technical report, U.S. Army Research Institute, Nov, 1994.
- [40] Ning Ning Zhou and Yu Long Deng. Virtual reality: A state-of-the-art survey. *International Journal of Automation and Computing*, 6(4):319–325, oct 2009.