Faculdade de Engenharia da Universidade do Porto

An Approach to Simulation of Autonomous Vehicles

Miguel Cordeiro Figueiredo

Dissertation executed under the
Integrated Master in Electrotechnical and Computer Engineering
Major in Telecommunications

Supervisor: Prof. Dr. Rosaldo Rossetti
Co-Supervisor: Prof. Dr. Luís Paulo Reis

June 2009

Dedicated to

My grandfather Filipe, may he rest in peace.

Abstract

The most common cause of traffic accidents is driver error. This isn't going to change any time soon thanks to increasingly cell-phone usage, in-car entertainment systems, and more traffic. Autonomous vehicles, like driverless cars, would decrease traffic accidents and traffic jams.

Automakers are currently developing systems that will enable these cars to do their role. Some of these systems are already widespread. For example, Anti-lock brakes, a standard feature in most cars, are a basic form of driverless technology. But there's still much work to do in the field of autonomous vehicles.

Simulations are safer, more efficient, and cheaper than live testing on vehicles. Changes have to meet a certain level of operation before they are put to a live test.

This thesis is about the study and implementation of a simulator to test such vehicles. Included is a study of the State-of-Art in driverless car simulations, and the objectives that such simulators should aim for in order to help test driverless car operations. Also included is an implementation strategy for such a simulator, and the software used for it, as well as modifications made to some software, and perspectives for future development.

Resumo

A causa mais comum dos acidentes de trânsito é o erro do condutor. Isto não vai mudar tão cedo, graças ao crescente uso de telemóveis, uso de sistemas de entretenimento no interior do veículo e ao aumento do trânsito. Os veículos autónomos, tais como carros sem condutor, poderiam diminuir estes acidentes de trânsito e os engarrafamentos.

Os fabricantes de automóveis estão actualmente a desenvolver sistemas que permitem que esses veículos desempenhem o papel desejado. Alguns destes sistemas já estão amplamente difundidos. O da travagem anti-bloqueio, uma característica padrão na maioria dos automóveis, é um exemplo de uma forma básica dessa tecnologia. Mas há ainda muito trabalho a fazer no campo dos veículos autónomos.

Simulações são mais seguras, mais eficientes, e mais baratas do que testes ao vivo em veículos. Alterações nos veículos e no seu software devem chegar a um certo nível de qualidade antes de se fazerem testes ao vivo.

Esta dissertação é sobre o estudo e implementação de um simulador cujo propósito é testar estes veículos. Inclui uma panorâmica do desenvolvimento actual da área das simulações de carros autónomos e, também, os objectivos que estes simuladores devem tentar alcançar, a fim de ajudar a testar o funcionamento destes automóveis. Também está incluída uma estratégia de implementação para um simulador, o software utilizado, assim como as alterações feitas, e perspectivas para futuro desenvolvimento.

Acknowledgments

Special thanks go to my family and friends for their unconditional support.
Especially to my sister Filipa for her help in the final stages of the dissertation. Without it, I don't think I'd have done this.

I'd also like to thank my supervisor, Prof. Dr. Rosaldo Rossetti, for assisting me with choosing the subject of my dissertation and for his understanding, and Rodrigo Braga for his aid with the conceptual issues I had.

Finally, thanks to Pedro Malheiro and Paulo Ferreira for their support with the simulators they worked with. Their insight helped me set the appropriate scope for this thesis.

List of Contents

List of Figures

List of Tables

Abbreviations

List of abbreviations

CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DEEC	*Departamento de Engenharia Electrotécnica e de Computadores*
FEUP	*Faculdade de Engenharia da Universidade do Porto*
GPS	Global Positioning System
LIACC	*Laboratório de Inteligência Artificial e de Ciências de Computadores*
MAS	Multi-Agent System
MAS-Ter Labs	Laboratory for MAS-based Traffic and Transportation Engineering Research
UDP	User Datagram Protocol
V2I	Vehicle-To-Infrastructure
V2V	Vehicle-To-Vehicle
XML	Extensible Markup Language

Chapter 1

Introduction

In this first chapter, the subject of this study is introduced: Simulation of driverless cars. How important driverless cars will become, and how important it is to simulate them. Also mentioned, are my personal motivations behind this study. The general objectives of this dissertation are also listed, as well as a short explanation of this document's structure.

1.1 – Motivation

Autonomous vehicles are one of the possible solutions to our most common cause of traffic accidents: Driver error. According to several different studies, like the one from L. Craig Davis in 2004, Driverless cars will substantially decrease traffic accidents and traffic jams even if there's just a few of them driving among the regular cars to minimize the traffic waves [1].

The first attempts to make robot cars began in the 80s, in Germany. Some of these UniBW cars would drive as fast as 96 km/h on empty streets. Efforts continued and one of these cars autonomously drove 1678 km on public highways from Munich to Denmark and back, at up to 180 km/h, automatically passing other cars. And after that came the DARPA Grand challenge, in the USA. There was no traffic, but close none road markers as well. There was only a long course of desert. Eventually there was a similar competition with traffic, called DARPA Urban Challenge.

In the latest years, with computer technology advancing fast, simulations began to be used more and more for this kind of project. Because simulations are safer, more efficient, and cheaper than live testing on vehicles.

Having developed an interest in computer programming early in my childhood, everything that was computer related caught my attention. And robots (in Science Fiction) were no exception, and neither was that old TV Show, Knight Rider, where the star was K.I.T.T., the talking car that didn’t need a human driver. Later, during my university studies, one of my

appointments was to make a small and simple 3D Application: The idea of creating a 3D mini-videogame was irresistible. In the end, that's how programming, 3D graphics, and robot car simulation all came to be interests of mine, and how this subject found me.

Nowadays, the existing robotic simulators are still insufficient when it comes to simulating sophisticated driverless cars in scenarios with intense traffic, as we will see in Chapter 2. And this project aims to change that.

1.2 – Objectives

The dissertation's main purpose is the study and specification of a realistic simulator for analysis of autonomous driving and semi-assisted driving on networks of intense vehicular traffic. This will be based on the MAS-Ter Labs traffic simulator developed at LIACC [2][3][4] and on a modified version of Ciber-Rato software, developed by University of Aveiro [5]. Ciber-Rato is a robotic simulator for a competition where the goal is to develop an agent which will control the simulated robot and guide it through a virtual maze [6] using sensor signals themselves simulated by the simulator. This modified version of Ciber-Rato is the Intellwheels simulator.

The concept will be tested by implementing a prototype with some basic functions. The objectives of this work are the following:

i. Study and characterization of the simulation of autonomous vehicles and semi-assisted driving;
ii. Concept study for the integration of a traffic simulator with a detailed autonomous vehicle simulator;
iii. Communication of car positioning from MAS-Ter Labs traffic simulator to a modified version of Intellwheels simulator;
iv. Simple road network XML, for use in traffic simulation tests in MAS-Ter Labs;
v. Simple map XML, for use in modified version of Intellwheels simulator;
vi. 3D Models and Textures for better immersion in the tests;
vii. Functional playable type agent to demonstrate collision tests, physics and sensors;
viii. Modification of Intellwheels simulator to adapt to simulation of autonomous vehicles;
ix. Implement vehicle's sensor interaction with the vehicles whose positions are being communicated by the MAS-Ter Labs traffic simulator.

1.3 – Structure of the Document

This dissertation is organized in six chapters, the first of which is this introduction.

The second chapter discusses the state of art of the existing traffic simulators, robotic simulators, simulation of driverless cars, as well as the MAS-Ter Labs simulator, and the Intellwheels project, and the current limitations of the simulators in general.

In the third chapter, the solution design will be introduced: approaching the problems that this design aims to solve, and discussing how the design deals with the problems, as well as the tools used to do it.

The development of the prototype is discussed in the fourth chapter. Specifically, the chapter discusses the modifications done to the existing simulators in order to implement a prototype of the design introduced in the previous chapter, as well as the elements created to increase the realism of the simulation, such as the map.

Chapter five will briefly explain the methodology for the performance and functional tests, as well as expressing the results and demonstrating basic functionality.

The sixth and final chapter of this dissertation concludes the entire project with a few remarks followed by the most relevant test results. Further developments are also discussed, followed by potential future works, approaching possible paths of additional research and development from this point on.

Chapter 2

State of the Art

In this chapter, the state of the art of the simulation of driverless cars is briefly described. This forcefully includes a glimpse at the state of art of driverless cars and robotic simulation. The different kinds of simulators that exist to simulate driverless cars and their main characteristics are explained here as well. These are also compared against one another. The MAS-Ter Labs and Intellwheels project will also be briefly introduced due to their roles in the simulator developed, and finally, the chapter is concluded with the limits of today's simulators, and a short summary.

2.1 – Driverless Cars

The first attempts to make robot cars began in the 80s, in Germany, and were made by Ernst Dickmanns and his group at Univ. Bundeswehr Munich (UniBW). Some of these UniBW cars would drive as fast as 96 km/h on empty streets. This was followed by the largest robot car project ever: the pan-European Prometheus project worth almost $1 billion, it involved UniBW and many other groups. This started in 1987 and ended in 1995. One of these cars, the "VAmP" (a Mercedes 500 SEL) using vision-based sensors, drove in Paris traffic in 1994, tracking up to 12 other cars simultaneously. It drove more than 1000 km on the Paris multi-lane ring, up to 130 km/h, automatically passing slower cars in the left lane. And in 1995, a car made by the same group, a S-class car of Dickmanns and UniBW, autonomously drove 1678 km on public highway from Munich to Denmark and back, at up to 180 km/h, automatically passing other cars. A few years later, in the USA, year 2005, DARPA started its "Grand Challenge" in the desert. There was no traffic, but close none road markers as well. The course was 211 km long and the fastest team was Stanford's who did the whole course in almost 7 hours. This was followed in 2006 by a similar demonstration in Europe, called ELROB (European Land Robot Trials) that was also with autonomous off-road vehicles. In 2007 there was another DARPA Grand Challenge and another ELROB challenge as well, and finally in 2007

there was the DARPA Urban Challenge, which consisted of an urban scenario, with traffic, where the driverless cars would try and complete missions given to them in a given amount of time.

Figure 2.1 - Driverless car named "Junior" of Stanford University Racing Team [7]

Figure 2.2 - Driverless car named "Odin" of Victor Tango Team from Virginia Tech [8]

2.2 – Simulation of Driverless Cars

In the latest years, with computer technology advancing fast, simulations began to be used more and more for this kind of projects. Simulations are safer, more efficient, and cheaper than live testing on vehicles. Simulations also allowed testing more scenarios than those that would have been possible with real world testing, and they also allowed testing situations too dangerous to involve humans. Testing in the virtual world is the ideal solution to validate code quickly, with more possibilities, cheaply and with minimum risk.

Up to now, many of these projects used no simulators to test changes in the implementation of their cars. Some did, but using simulators with minimal functionality. And even those were not capable of simulating the cars in crowded roads.

Existing robotic simulators are insufficient when it comes to simulating driverless cars in scenarios with intense traffic, as we will see in the next section. And this project aims to change that.

2.3 – Types of Simulators

Simulators that are related to the study of traffic and autonomous vehicles are currently separated into two types: Large scale traffic simulators, and small scale robotic simulators.

2.3.1 – Traffic Simulators

Large scale traffic simulators simulate traffic flow over very large and/or complex road networks. In these simulators the movement of each car is so simplified that you almost can't distinguish between human drivers and robot drivers. This is also because the behavior of the drivers is a very simplified model, in order to allow for a simulation with thousands of vehicles to be able to run in a computer with reasonable resources, and within a reasonable timeframe. A good example of such simulators is the MAS-Ter Labs Traffic Simulator. [2][9]

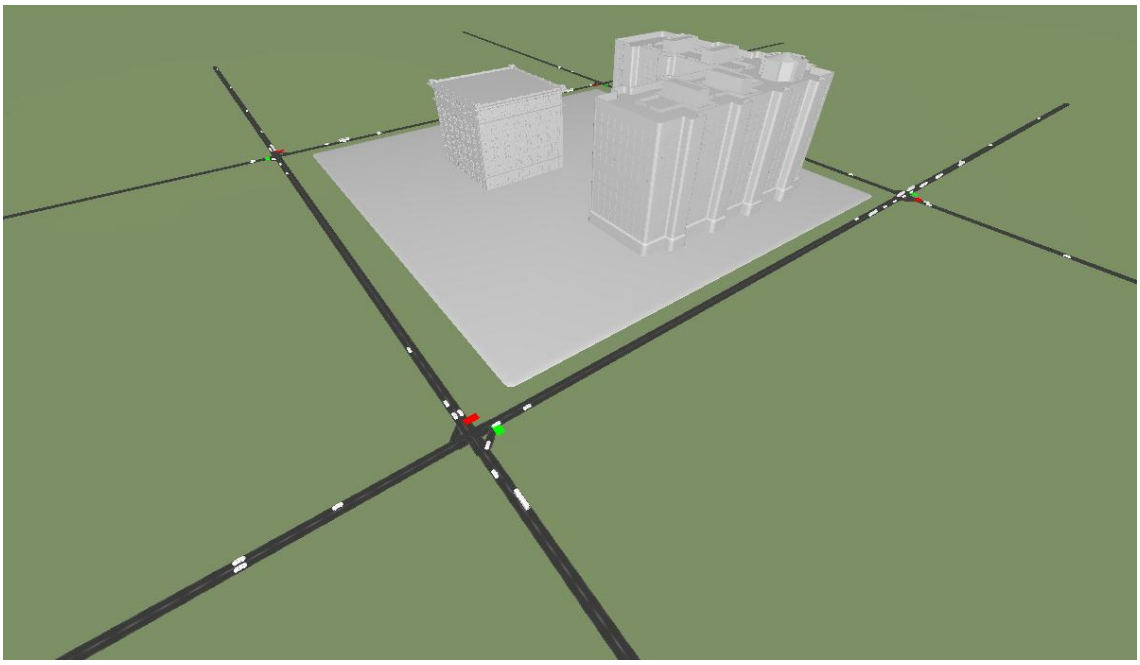


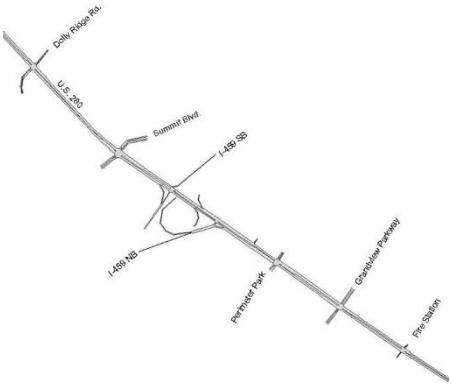
Figure 2.3 - Screenshot of MAS-Ter Labs Traffic Simulator (with the map designed for this thesis)

It should be noted that these simulators are not detailed enough to simulate autonomous cars when it comes to their behavior, sensors, actuators, surroundings, etc. The physics in these simulators are over-simplified, to the point that the movement of the cars that are changing lanes isn't continuum: A car is either in lane A, or lane B. There's no state where the car is partially occupying both lanes. The point of these simulations is to simulate traffic flows, average waiting times, average speeds, fuel consumptions, etc. [10].

The MAS-Ter Labs Traffic simulator is the one used in this project to perform the role of traffic simulator in our design solution, because it was developed here in LIACC as well, and it’s not a commercial solution. The reasons why this simulator was chosen are explained in the section 3.1.3 of this document.

There are other simulators with similar features [11], such as PTV’s VISUM and VISIM [12][13][14] (a commercial solution to simulate traffic road networks), SIMTRAFFIC [15][16][17], AIMSUN [18] and CORSIM [13][16][19] which is compared to the others in the references just mentioned. The best microscopic traffic simulators studied all rely in very

simple models (car-following model, lane-change model, and route-choice model [20][21][22][23] to simulate large numbers of independent, yet apparently intelligent drivers [24][25]. This model is pretty much the same for all of the microscopic type of traffic simulators, and they're similar in the way they display the information as well. For more information on comparisons between them, see [11].



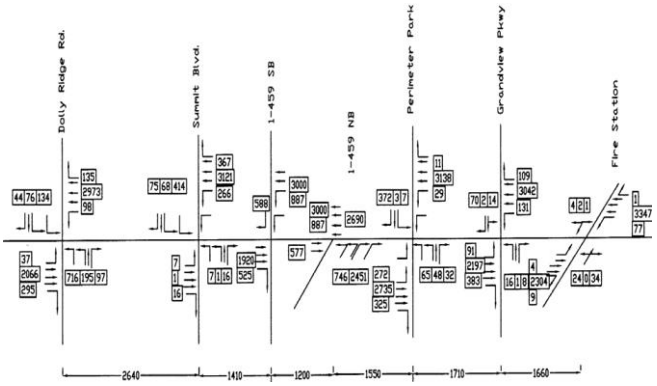


Figure 2.4 - U.S. Highway 280 study corridor at CORSIM Software [11]

Figure 2.5 - AM peak hour traffic volumes on U.S. Highway 280 at CORSIM Software [11]

Although some of them have more analysis tools than others, like graphics showing the traffic flow in different roads, waiting times, average speeds, etc, all this falls out of the scope of this dissertation, and so it is unnecessary to study and compare these differences. After all, the role of the traffic simulator in this project is meant to influence the simulation of the autonomous vehicle being simulated in detail in the robotic simulator. It is not the scope of this project to study road network traffic and its flows.

2.3.2 – Robotic Simulators

In order to simulate and test the performance of a driverless car, a more detailed simulator is required. Like a small scale robotic simulator. A lot of sensor and actuator detail is needed if we want to know how will the car throttle, break, and steer, when reacting to various things ranging from walls and other cars, to people, obstacles, animals, sidewalks, etc. Collision detection is also required, to know if the driverless car malfunctioned to the point of causing an accident. We might want to know which of the sensors are detecting (or not) which obstacles, and try to identify errors in its design (both hardware and software) from those tests. A large scale traffic simulator can't provide us with those details.

A look has been taken at the DARPA Urban Challenge teams and what tools they used to simulate their cars before live-testing. To be successful in such a competition, the use of a simulator is basically unavoidable. Overcoming safety concerns and strict time constraints is a must here. And testing in the virtual world is the ideal solution to validate code quickly and

with minimum risk. The teams were also able to test situations too dangerous to involve humans and test more scenarios than would have been possible in the real world. Simulation was used to find obvious problems with their software, but this was always followed by testing on the vehicle.

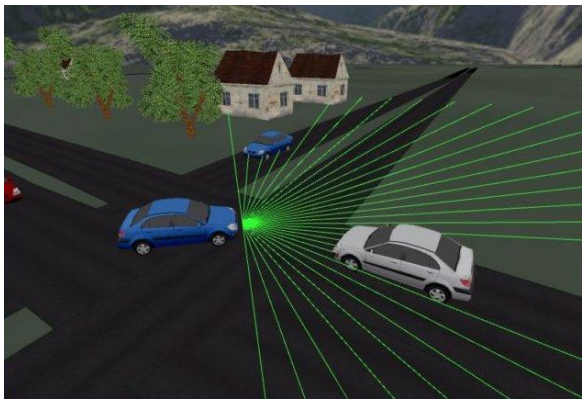


Figure 2.6 - Simulator of Victor Tango Team from Virginia Tech [26]

Figure 2.7 - Sensors Simulation at the Simulator of Victor Tango Team from Virginia Tech [8]

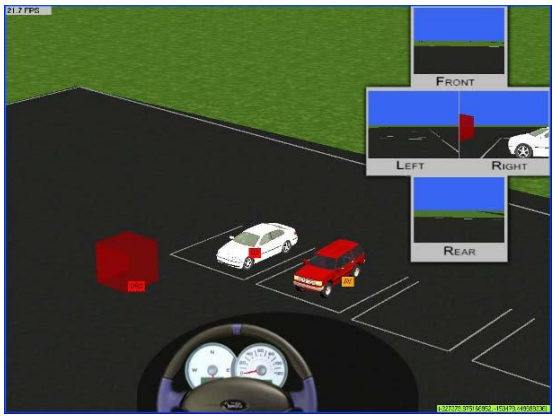


Figure 2.8 - Live Test with "Odin", the driveless car of Victor Tango Team from Virginia Tech [8]

Figure 2.9 - Screenshot of 3D GUI Software from The Princeton University team [27]

The Intellwheels simulator [28] is a simulator that was built based on the Cyber-Rato simulator[5][29]. It's one of these robotic simulators. When comparing to the others, Cyber-Rato is mentioned instead of Intellwheels simply because it is the original. But note that this modified version of Cyber-Rato, known as Intellwheels simulator, is the one used for this project: performing its role as a robotic simulator to take care of the sensor information, physics and other details in our simulations. The reasons why this simulator was chosen are also explained in the section 3.1.3 of this document.

2.3.2.1 – Robotic Simulators Characterization

These are some simulator features used as means to compare the different robotic simulators. Some are more important than others, depending in the field of study. Listed here are the ones that are more or less useful when it comes to testing driverless cars:

- **3D simulation** - Some simulators run full 3D physical simulations. The calculations become much more complex and resource-consuming, but this results in a more realistic simulation;
- **3D visualization** - Simulators with this feature allow the user to observe and better understand the events happening in the simulation by animating detailed 3D graphics and models to represent the different elements of the simulation;
- **Large scale traffic** - These simulators are able to calculate large amounts of elements with very simple behaviors and physics in order to be able to reproduce large-scale phenomena in a reasonable timeframe and using reasonable computer resources;
- **Multi Agent simulation** - Some simulators can have multiple and independent Agents interact in the same simulation. The nature of these simulations can be simply to test how agents react to one another, but they also enable testing of cooperative action, or competitive action;
- **V2V / V2I communication** - Simulators that can simulate communications between agents will allow for messages to be exchanged between the agents, and may simulate physical restrictions like the broadcasting radius of a certain robot;
- **Collision detection** - This is a very basic feature and is implemented in almost every simulator out there. The point of simulations is to test for anomalies and undesirable events, and a collision is the most common of such events, independently of the field of study;
- **Sensor noise** - Simulators that are able to calculate random noise at the outputs of sensors will allow for more realistic testing of the decision making systems that need to deal with non-ideal nature that real sensors have;
- **Failure simulation** - Failure simulation is another feature that will help test if the robots are fail-safe. This is when the simulator has the ability to corrupt, or completely suppress, the information coming from sensors to the agents, or from the agents to the actuators;
- **Environment affecting sensors** - Harsh weather conditions and hazardous terrains can affect sensors in various ways, for example, fog or darkness affecting the visibility of an optical camera, or intense weather causing echoes in laser scanners. Simulators might include these factors in the calculation of sensor values;

- **Environment affecting physics** - Weather and ground conditions can also affect the performance and control of the vehicle in various ways, for example, loose gravel, rain, or snow making the roads more slippery.

In the following list, the most relevant sensors and measures in this field of study are briefly introduced, so that we can later compare what sensors are driverless cars using, and which simulators are able to simulate such sensors.

- **GPS** - A GPS receiver is able to determine their current location, the time, and their velocity, thanks to the precise microwave signals transmitted by a constellation of between 24 and 32 Medium Earth Orbit satellites;
- **Luminosity** - There are a few kinds of sensors that simply detect the amount of light hitting them. A cheap and easy way of knowing if a car needs to turn the lights on due to the night, or tunnels and such;
- **Optical camera** - There are many types of optical cameras there. The point is to have it send a stream of images for the robot to analyze things like movement flow, colors, etc;
- **Infra-red camera** - It's essentially the same as a normal optical camera, except that it detects light in the infra-red spectrum instead of the human visible spectrum. Also known as night-vision camera;
- **Laser scanner (LIDAR)** - This sensor emits a laser that is constantly changing its angle, while listening to the laser reflections. It results in an array of points where the laser hit a target and got reflected. This happens as fast as 12.5 times per second in a typical one. If used correctly, it can measure the distance, size, shape and speed of multiple obstacles several times per second;
- **Ultrasound** - Typically used to measure short distances in a wide angle;
- **Infrared** - Typically used to measure short distances in a sharp angle;
- **Inertial Measurement** - An inertial measurement sensor is the main component of inertial guidance systems used in air-, space-, and watercraft. It works by sensing motion — including the type, rate, and direction of that motion — using a combination of accelerometers and gyroscopes. The data collected from these sensors allows a computer to track a craft's position, using a method known as dead reckoning[1];
- **Radar** - It emits either microwaves or radio waves that are reflected by the target and detected by a receiver, typically in the same location as the transmitter. Although the signal returned is usually very weak, the signal can be amplified.

[1] Dead reckoning - This is when upon known speed, elapsed time and course, one estimates his current position by advancing on a previously determined position. It's no longer considered a primary method of navigation, but is widely used in complement with more complex navigation systems.

This enables radar to detect objects at ranges where other emissions, such as sound or visible light, would be too weak to detect;

- **Speed** - The simple sensors that exist in every car that connect to the speedometer to let the driver know the instant car's speed. They're usually based in watching how many times the car's wheels complete a revolution in a given amount of time.

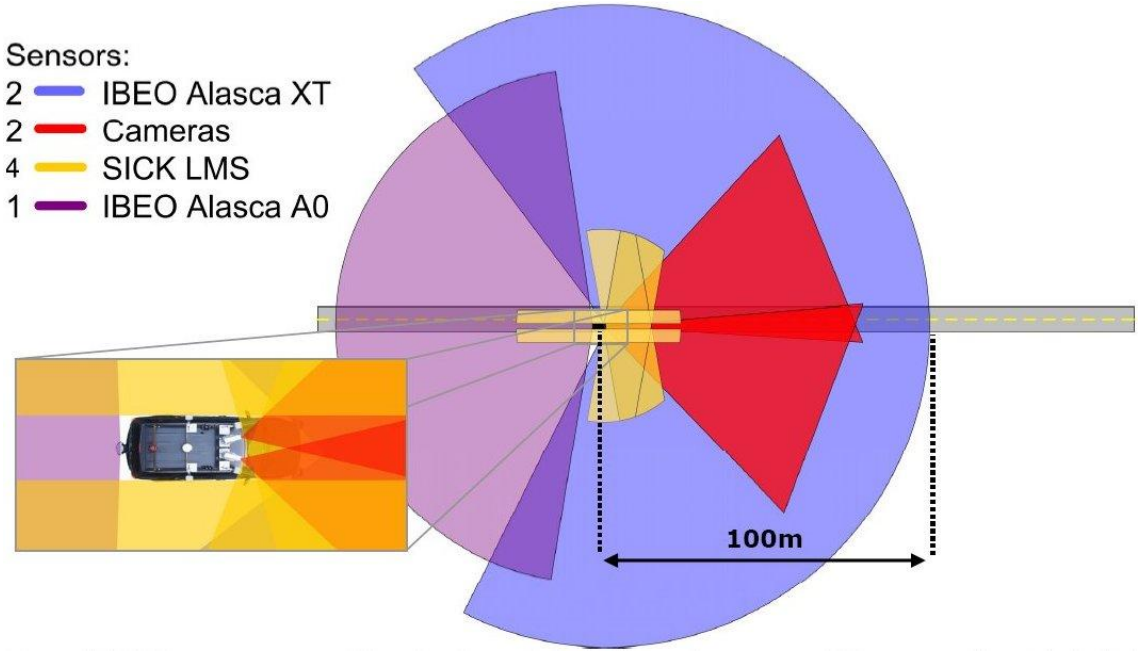


Figure 2.10 - Odin's sensor coverage [30]

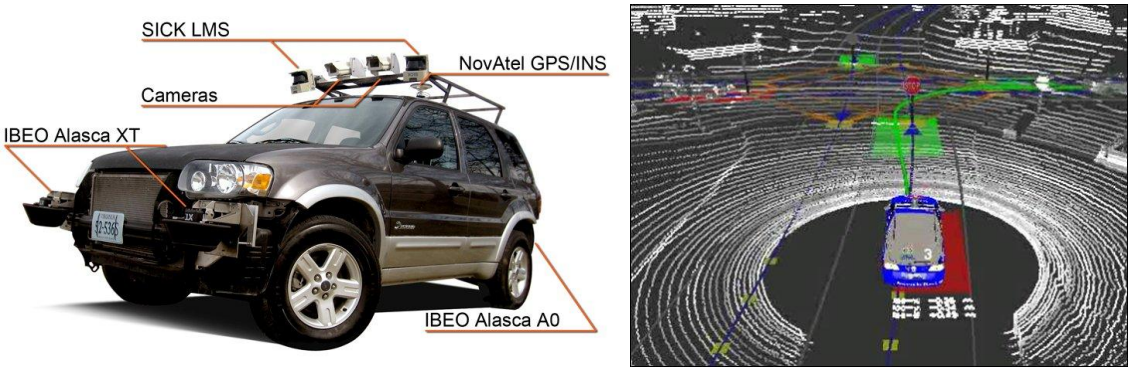


Figure 2.11 - External view of Odin with sensors labeled [30]

Figure 2.12 - Stanford's simulator replaying data from the Urban Challenge final event [26]

2.3.2.2 – Robotic Simulators Comparison

Next, the features of different Robotic simulators are compared. Some teams that participated in the DARPA Urban Challenge 2007 are also compared: Both their simulators'

features, and the sensors that the cars had equipped. A generic game engine with the typical features is compared as well, because with some modifications, game engines are functional enough to allow real-time simulation of real applications. USARSim [31] is an example of a simulator that was implemented by modifying a game engine: Unreal Engine, developed by Epic Games (from a First Person Shooter game called Unreal Tournament) [32].

Figure 2.13 - Simulations in the USARSim[2]

So we want to compare game engines to the generic robotic simulators and to the simulators made by the DARPA Urban Challenge teams. Included in the comparison are also two robotic soccer simulators known as ÜberSim [33] and EyeSim [34].

In Table 2.1, we compare the simulators, included those made by teams for use in the DARPA Urban Challenge 2007 [7][8][26][27][30][35][36][37][38][39][40][41][42].

While in table 2.2, these entries contain the information of which sensors their driverless cars had equipped during the competition. Detailed information about the different simulators that each team used is hard to find, and so, some of these features are marked as not determined. That's specified in the table's legend.

It's important to remember, at this point, that large scale traffic simulators are insufficient when it comes to simulation detail. They lacked in all the features mentioned in the previous section, except for that of "Large scale traffic" (and in some cases "3D visualization"), so they weren't included in the following table - Table 2.1. And since they don't simulate sensors at all, they weren't included in Table 2.2 either:

[2] Left image source: Dr. Alexander Kleiner and Dr. Sven Behnke. Institut Für Informatik Freiburg (IIF), Research Group on the Foundations of Artificial Intelligence, Software Laboratory. Accessed at: http://www.informatik.uni-freiburg.de/~ki/teaching/ss05/sopra.html

Right image source: OxfordRescue Team 2008, RoboCup Rescue - Virtual Robots Competition. Accessed at: http://www.oxfordrescue.co.uk/

Table 2.1 – Summary chart with the features of today's Robotic simulators

	Simulator features									
	3D simulation	3D visualization	Large scale traffic	Multi Agent simulation	V2V / V2I communication	Collision detection	Sensor noise	Failure simulation	Environment affecting sensors	Environment affecting physics
Game engines										
Generic	x	x	x			x			x	x
DARPA Teams' simulators										
1st place - "Boss"	x?	x?				x?				
2nd place - "Junior"	x?	x?				x?				
3rd place - "Odin"	x?	x?				x?				
"Talos"	x?	x?				x?				
"Little Ben"	x?	x?				x?				
"Skynet"						x?				
PAVE (Princeton's team)	x?	x?				x?				
Generic robot simulators										
Ms Robotics Studio	x	x		x	?	x				
Webots	x	x		x	?	x				
CyberRato				x	m	x	x			
USARSim	x	x					x			
ÜberSim (Robot Soccer)		x				x				
EyeSim (Robot Soccer)		x		x		x	x			

Table Legend:
x - Yes | m – Yes (considering modifications known to have been made already)
x? – Could not be verified, and the information was not found, but assumed as being a positive
? – Couldn't be determined | blank – No

The observation of this table reveals that most simulators nowadays lack important features for the simulation of autonomous vehicles.

Robotic simulators nowadays don't seem to simulate:

- Large scale traffic or pedestrians – We want to simulate how a driverless car responds to crowded areas, or traffic jams;
- V2V/V2I Communications – We want to simulate how effective can these communications be between vehicles and the road network infrastructures and how they can improve operations;
- Sensor Noise – There's a wide amount of sensor errors due to their non-ideal nature. We need to simulate as much as possible when it comes to those errors if we want to know if the decision-making of the driverless car can take these into account and avoid malfunctions. These include sensor noise, illusions, reflections

and echoes. For example light reflection with LADARs: Some surfaces do not reflect laser pulses well, where others reflect too well: Well painted road lines can be picked up with an LADAR system and reported as an obstacle, when in fact the vehicle can drive over them;

- Failures – Sensors and actuators can simply malfunction and stop playing their role in real life situations. If we want fail-safe driverless cars, we need to simulate if they can minimize damage, safely come to a halt, or even drive and function close to normality even if some of the equipment is malfunctioning. This can range from simple GPS failure to a more serious failure involving steering or breaking;
- Atmospheric condition, terrain condition, and environment - These are all real, and they affect sensors, actuators, and the behavior of cars as a whole. If these aren't taken into consideration when simulating driverless cars, we can't predict how well they'll operate in the non-ideal real-life situations.

Some of the teams that participated in the DARPA Urban Challenge implemented their own simulators, while others used already existing simulators. Some of their own simulators had very practical features that are worth mentioning [26].

The simulator used by Princeton's team allowed the user to test their code in the simulator and then transfer it to the vehicle without the need to recompile [27], thanks to their use of the Microsoft Robotic Studio framework [43].

MIT's simulator could play back data recorded in real life test runs, but the simulated obstacles reflected perfect data during those recorded runs, something that actual sensors didn't obtain during the real life test runs.

CarOLO also used their simulator to test new software implementations before adding them to the vehicle, as well as confirming bugs found during real world tests. That's something that the previous teams also did. But further development of this simulator has yielded a version in which multiple instances of their autonomous vehicle could be operated. In doing this, their software could learn efficient driving behavior in an environment in which multiple traffic vehicles exist. In addition, different versions of code could be run from the same starting point, running the same mission file, in order to compare their performance [35].

Tartan's simulator also had the ability to add virtual obstacles to a real world environment during testing. In doing this, the vehicle was made to think there were obstacles in front of it even though there were none. This is achieved with Augmented Reality and Mixed Realities, when the real world reacts to interactions between the real world and virtual objects, and different realities interact continuously. These features are also available in the Intellwheels Simulator, and explained further here [44][45].

All these features are very time-saving when simulating and testing, and most of them were not easy to implement. They were developed for these simulators because time was the most important factor during the DARPA Urban Challenge, since the teams only had a few hours to update and validate their code between the events.

In the next table, we can compare which sensors are simulated by these generic robot simulators, and the sensors equipped in the DARPA's driverless cars.

Table 2.2 – Summary chart with the sensors simulated by today's Robotic simulators

	Simulator sensors									
	GPS	Luminosity	Optical camara	Infra-red camera	Laserscanner (LIDAR)	Ultrasound	Infrared	Inertial Measurement	Radar	Speed
DARPA cars' sensors										
1st place - "Boss"	x				x			x	x	x
2nd place - "Junior"	x		x		x			x	x	x
3rd place - "Odin"	x		x		x			x		x
"Talos"	x		x		x					x
"Little Ben"	x		x		x					x
"Skynet"	x		x		x				x	x
PAVE (Princeton's team)	x		x						x	x
Generic robot simulators										
Ms Robotics Studio					x					?
Webots	x	x	x			x	x	x		?
CyberRato	x					x				?
USARSim					x	x				?
ÜberSim (Robot Soccer)										?
EyeSim (Robot Soccer)			x				x			?

Table Legend:
x - Yes | ? – Couldn't be determined | blank – No

After a quick observation, you'll notice that car game engines aren't good enough because they're simply not sensor based, and they also lack some important simulation features like the ability to use Multi-Agents, V2V/V2I Communication, and sensor noise. They lack exactly what the real projects like DARPA Teams need to simulate how their cars will behave before testing in real situations.

2.4 – Limits and Specialization of the Simulators

Even after so much effort in the development of simulation, we are still far away from having an ideal simulator. The more features they tend to have, and the more realistic they are, the more resources they need to do their calculations in a reasonable amount of time. But simulation won't help much if it's oversimplified. In other words, we need a balance between the realism of the simulation, and the simplicity of its calculations. There are lots of things that we'd love to be able to simulate, but the sheer amount of processing power those would require are simply too much. Let's talk about these limits.

Care must be taken with big quantities. When we're talking about large amounts of things in a simulation, they better be simple. You want to make some complex calculations, and many simple calculations. But you don't want to make many complex calculations. For such an example, let's look at a large scale traffic simulator. The behavior of those thousands of vehicles is simplified to the point of the simulator being able to make thousands of calculations almost instantly. That's because the decisions are simple, and their physics and movement are very simple as well.

But imagine if each one of these cars was aware of its environment through individual sensors, affected by generated noise. And all these cars would go through complex decision-making algorithms in order to send commands to their actuators, and then the simulation would have to calculate the next step based on what the actuators were ordered to do. If you multiply all these details by a thousand cars, it would take days to simulate a just a few seconds (if not less) of such a large scale scenario.

But it's very easy for a normal computer to simulate in real-time a few of these driverless cars. So the most obvious solution is to try and have more detail where it matters and less detail where it doesn't matter, depending on the specialization of the simulator. This means, in the case of this project, that we could try and have some detailed cars, with detailed decision-making and a more detailed physics simulation to study, and surround them with a crowd of less detailed objects, with their very own simple simulation.

3D simulation becomes a bit of a problem here, especially if you're adding a third dimension to the terrain. That's because in this case, there's a whole lot more calculations involving those thousands of cars, to account for the extra dimension in their dynamic. For highway systems and most road networks, the changes in elevation are relatively small. So the simulated results are similar even without a 3D simulation. But with some cities and road networks, there are very drastic elevation changes. A 3D simulation is very important in these cases because it must be taken into account that sensors can't always look around corners, or down hills.

The vehicle performance is affected by the third dimension as well. It needs more power to climb uphill, and more importantly, it needs to break sooner and harder if it's going downhill. Weather conditions such as rain and snow also affect the way the vehicle performs.

Dirt, loose gravel, and other different ground surfaces can also be simulated to test the vehicle's control system. Again, a normal computer can simulate effects like these for a few cars, but not for thousands.

A good example of something that would be very hard to realistically implement in a simulation is GPS signal loss. The satellite signal can be occluded by pretty much anything that is big enough and in the way, like buildings, or trees. Realistic simulation of such a blockage would require simulation of the satellites and their orbits. The line of sight between the GPS receiver and the satellites would need to be tested to see if there was a signal loss or not. You can go as far as considering signal reflections off large buildings. But all these details would take a tremendous amount of processing power, and a lot of effort to implement. A balance between the effort and the results is essential: We can try and simulate GPS signal loss in various ways, from simple random time intervals, to specifying areas in the map where the GPS signal would be lost. As long as the result is that the GPS receiver loses the signal every once in a while, we get a simulator that can test the vehicle’s ability to predict its position roughly even if he temporarily loses the GPS signal.

There's also a good example of something that was very hard to simulate a few years back, and now, with some developments in computer technology, has became easy on processing needs. That's the example of vision based sensors. Vision algorithms used to be tested by simulating the markings in the roads, but the results would differ a lot to the real-life, where there are shadows from objects that might be out of the picture, atmospheric conditions like fog, storms, or even direct sunlight. Vision algorithms can now be tested more realistically because 3D rendering has evolved a lot lately. Now it's very common to have applications (any common video-game) that has a 3D engine implementing all these features, and can render them all in real-time with a reasonably cheap computer. It's a matter of streaming the result video output into a virtual camera instead of streaming it into a monitor screen. And you end up with a camera receiving a video that features photorealistic weather effects [32].

An ideal simulator would have to be indistinguishable from the real world. That might only be accomplished in an utopian future. But we can take what exists and adapt it to our necessities when it comes to simulation, by re-balancing the capabilities of the simulator.

Figure 2.14 - Screenshots of Crysis game[3]

[3] Images source: TClms5400, "PROJECT OFFSET vs CRYSIS vs ALAN WAKE", 4 Nov 2007. Accessed at: http://www.gamespot.com/pages/forums/show_msgs.php?topic_id=26022609&page=0&prev_button=1

2.5 – Summary

Although attempts to build driverless cars are not new, only recently has it become an intensive effort. Only now people are worrying with the effectiveness of the testing and simulating of their projects, and the simulators used are far from being ideal. The more features they tend to have, and the more realistic they are, the more resources they need to do their calculations in a reasonable amount of time. But simulation won't help much if it's oversimplified. And in some cases, it is.

We need a balance between the realism of the simulation, and the simplicity of its calculations. When deciding the type of simulation that we need to run, we need to decide what aspects are important to simulate, and what aspects really don’t matter much: We don’t have an ideal simulator, but the next best thing would be a simulator that allows us to choose what we want it to simulate.

Or, a connection between different types of simulators, that would each simulate different aspects and areas of a scenario to complement each other’s strengths. This is what this study aims for.

Chapter 3

Solution Design

In this chapter, the methodology of this study is discussed: The problem, the methods, and the tools. We'll also describe the proposed architecture, its goals and functionalities, and at last, a brief analysis of which of those functionalities are implemented in the prototype.

3.1 – Methodology

In the following sections, the problem and the goal is described in detail, as well as the methods followed to reach the goals of this study. Also described are the tools and software used in the development, and the development environment itself.

3.1.1 – Problem Statement

The goal of this study is to achieve a simulator that, while being reasonably detailed with its physics calculations for the simulation of autonomous vehicles, can take into account influences from large amounts of entities, like the situation of driving in road network that is crowded with intense traffic.

The problem is, as concluded in the end of chapter 2, that a simulator that could both simulate huge amounts of traffic, while at the same time being very detailed with it all, would take a lot of computing resources to run in a reasonable machine. But a traffic simulator is not detailed enough to simulate autonomous vehicles, while a robotic simulator is not good enough influencing a detailed autonomous vehicle with the actions of thousands of other cars.

The goal is to overcome the limits of the two types of simulators by making them work together, cooperatively. The strengths of both types of the simulators are joined together to locally eliminate their flaws where we are analyzing the problem.

The traffic simulator achieves quantity, while the robotic simulator achieves detail. We don't need to know the details everywhere in a simulation, so we can have the simulators

cooperate in a way where the robotic simulator calculates the details where we need them, while being influenced by the larger, crowded, but simpler world outside that is being managed by the traffic simulator.

3.1.2 – Methods

First, the general characteristics of the concept project were sketched (Chapter 3.2). After that, a study of the state of art was done (Chapter 2), to confirm the strengths and weaknesses of the existing simulators and to further develop the concept project. The solution design was developed afterwards (Chapter 3), having confirmed what was needed to complement today's traffic and robotic simulators. A proposed architecture for the solution design was defined, and then the programs and tools that would be used to build it were chosen (Chapter 3.1.3). Afterwards, further detailed goals and functionalities were defined for this project (Chapter 3.2.1), from which a few were selected to be implemented in a prototype simulator (Chapter 3.2.2). Finally, the prototype was achieved with modifications and implementations of the simulators previously chosen (Chapter 4), performance and functional tests were made, and the results noted down (Chapter 5).

3.1.3 – Techniques

The MAS-Ter Labs traffic simulator was chosen to perform the role of traffic simulator in the proposed architecture. It was practically chosen before even the study of the state of art of traffic simulation was done. There were a few reasons for this: It was developed in LIACC as well, so we were in close contact with those who had developed it. Also, its source was immediately available, unlike the commercial solutions already available. Although some of them have more analysis tools than this one, like graphics showing the traffic flow in different roads, waiting times, average speeds, etc, all this falls out of the scope of this dissertation. The role of the traffic simulator in this project is meant to influence the simulation of the autonomous vehicle being simulated in detail, in the robotic simulator.

Similar to the choice of the traffic simulator, the decision of the robotic simulator was rather quick too. The reasoning behind this decision is connected to the proven performance and flexibility of this simulator, as it has been used in different applications and adaptations. It has been successfully used for various competitions: Micro-Rato [10][46][47][48] (2001-2008), CiberMouse@RTSS [49] (2007-2008) and CiberMouse@DCOSS [50] (2008). It is open-sourced, so, the source code was readily available for us, and it was previously used at LIACC in several research projects such as a computational study on emotions and temperament in Multi-Agent Systems [51][52], development of cooperative rescue operations [53], and the already mentioned Intellwheels Project [54][55][56]. So, again, we are in close contact with those who had already worked with Cyber-Rato, and with those who had adapted it to various situations.

Next, the development environment was chosen: The software and libraries used to implement the prototype. All the simulators that were chosen for this project have been and are being developed under Microsoft Windows operating systems, so this simulator project should be developed under the same family of operating systems. This ensures better compatibility between interacting software and reduces the combined diversity of programming software requirements.

As for the programming software suite for the Intellwheels simulator, the choice was Microsoft Visual Studio C++. The Intellwheels simulator is in C++ language [31], and uses the Qt 2.3 libraries [57] from Qt Software (formerly known as Trolltech, before being bought by Nokia) [58], a set of libraries, with special classes and functions. These libraries are cross-platform (this means that they can be used in various operating systems, including Windows and Linux) and provide various class libraries that aid in the low level functions, allowing a higher level of programming. This version of Qt has direct integration with Microsoft Visual Studio C++, and this software provides a simple to use programming environment. This development environment was easy to set up too. Because of all this, that was the software we used to code and compile simulator-related code.

Regarding the simulation Viewer, and the simulation Agents: The ones modified were developed in Borland Delphi 7 [59]. It is an integrated software development environment that allows visual, event-oriented programming through Pascal programming language [60]. For the Viewer in particular, it calls and uses external OpenGL libraries [61][62] to render the 3D views. So a development environment using Borland Delphi 7 and OpenGL libraries was set up to code and compile changes to everything that was related to the simulator Viewer and the Agents.

Finally, regarding the MAS-Ter Labs traffic simulator, it was programmed in C++, and developed using the Eclipse software suite. This development environment was already installed and configured at the start of the project, so we saw no reason to change it: Since the workplace was already ready. It also used Qt Software's Qt [58] libraries, but the version used here was more recent than the version used in Cyber-Rato. The Qt libraries were already integrated into Eclipse as well.

3.2 – Proposed Architecture

The architecture for this simulation software and its peripheral modules was sketched out while guessing a few desired characteristics, after considering what was thought to be the limits of today's simulators. The initially desired goal was to create a detailed robotic simulator that could simulate autonomous vehicles as well as vehicles with semi-assisted driving technologies, among a realistic scenario of intense road traffic. This goal of ours was confirmed after a study of the State of Art was done in the fields of traffic and robotic

simulation which showed us that this subject still had a long way to go. So we developed the architecture further until it became a rich composition of connected modular software, as described below.

The figure below (Figure 3.1) is a simple schematic for the proposed architecture, where the different modules of the project are represented. The 2 greyed out ones in the centre of the figure are the modules which are the main focus and the point of start for the prototype: The Simulator, and the Simulation Viewer.

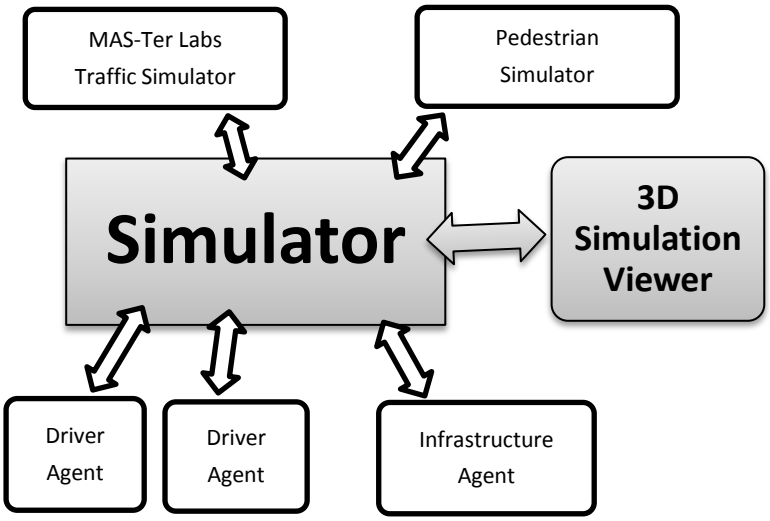


Figure 3.1 - Architecture of the simulator and its peripheral systems

The basics of the modules and their connections are as follows:

- The Simulator connects to MAS-Ter Labs Traffic Simulator and to the Pedestrian Simulator [63] and receives traffic information from both the simulators mentioned above, adding that information to its calculations. It is also able to send information of its agents back to those simulators, so that they, in turn, can take that information into consideration when calculating the outcomes of their own simulations. By default, the map is loaded from the MAS-Ter Labs Traffic Simulator, and a parser reformats the map into the format needed to the Simulator calculations, as well as sending the map to the remaining peripheral simulators, like the Pedestrian Simulator;
- The 3D Simulation Viewer connects to the simulator, and receives information from it to render the 3D representation of the simulation. It also has the functionality to send information of rendered images back to the Simulator for optical sensors like cameras (to test vision-based algorithms), including infrared cameras. Those rendered images, or video streams, are in turn sent to the relevant Agents so that they may analyze it. The 3D Viewer will also load the map from the Simulator for rendering purposes;

- Agents connect to the simulator and send all the information it takes for the simulation to know the necessary characteristics about the Agent. This includes component positioning and type of Agent. It can be a Driver Agent, in which case it will have a car assigned to it. It can also be an Infrastructure Agent, like traffic lights, intelligent street signs, communications access point, or obstacles.

3.2.1 – Modules' Functionalities

Below, the specific objectives and functionalities of the different parts of this architecture are briefly described.

3.2.1.1 – External Simulators

MAS-Ter Labs Traffic Simulator can be defined as a multi-agent traffic simulator following a microscopic approach, and in this section, its solution design is described. This content is largely based in the third chapter of [2], which resulted in two papers that were accepted into two different conferences: Readers are referred to [4] and to [64].

Note that this is not the implementation of the MAS-Ter Labs Traffic Simulator prototype that was made, but the concept behind it. The development of the prototype of the MAS-Ter Labs Traffic Simulator is approached in section 4.1 of this document, just before the modifications made to the prototype.

The Simulation Engine is at first described in [2] as the MAS-Ter Lab framework Virtual Domain sub-system. Its high level architecture and all its main components are illustrated in Figure 3.2. Each of the represented components can be run in a different processing unit, and can be interpreted as independent applications.

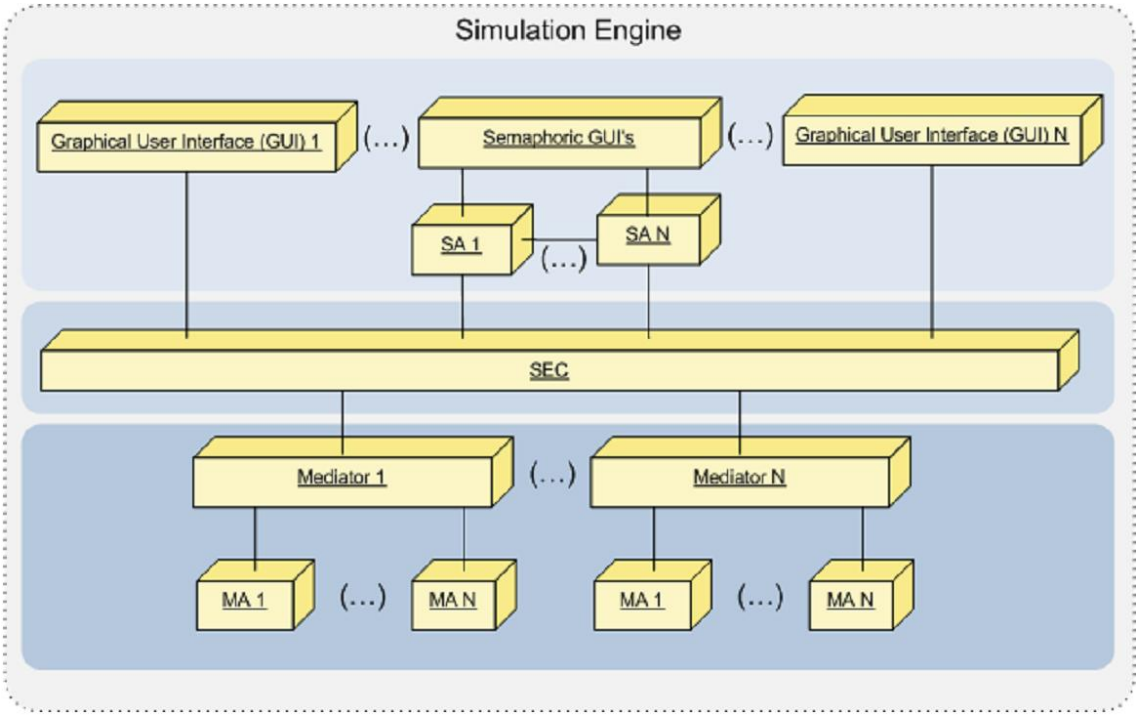


Figure 3.2 - Simulation Engine High Level Architecture [2]

As shown in the diagram of Figure 3.2, the simulation engine is highly distributed and is formed of several independent applications, making it very flexible.

The Simulator Engine Controller (SEC) illustrated in the middle of the diagram is responsible for receiving all allowed actions provided by all its features and for executing them. It is also responsible for sending simulation state updates every time it is necessary.

The Moveable Agent (MA) components are agents that control network entities that are allowed to move (for instance a driver of a car, or a pedestrian). Basically, they receive information in regular time intervals about its surrounding and decide what actions they wish to perform upon the entities that they control. Each Moveable Agent controls one single network entities and the control actions he wishes to perform on the respective entity must be sent to the Mediator application that will be responsible for executing them.

The Graphical User Interface applications are responsible for receiving the simulated network state in regular time intervals and render it in a human readable graphical interface.

The Semaphore Agent (SA) applications are basically agents that are responsible for controlling traffic light intersections with the objective of improving the overall traffic flow. Each Semaphore Agent only controls and regulates one single intersection. Their decisions are not solely based on the information they retrieve from their controlled intersection but also are based on information sent by other agents as well as negotiations between them. Instead of having as their only objective the improvement of the traffic flow of the downstream of a

traffic light intersection the Semaphore Agents also try to improve the overall traffic flow of the network by negotiating between each other before making decisions. These agents also provide, in regular time intervals, information about their decisions and intersection traffic flow state to the Semaphore Graphical User Interface Applications (Semaphore GUI's) that represent this information in a human readable graphical interface.

The Mediator application is responsible for coordinating a certain number of Moveable Agents that control a physical entity in the simulation. This coordination includes translating the received world state information sent by the Simulator Engine Controller in regular time intervals into the surrounding environment perceptions (objects that surround and influence the behaviors of the simulator entities co-habiting in a common environment) of each agent it coordinates. It also includes receiving agents' actions, aggregating them, calculating and updating the entities affected by those actions and sending the updated information back to the Simulator Engine Controller. The Mediator application is also responsible for launching new agents in the network entry points that populate its control area, as well as to kill agents that reach any of the network exit points (also inside its control area).

To achieve a desired perception of a part of the environment, an agent becomes a listener of all perceptible properties of entities that are inside an attention range (let us name it the agent's foci) that the agent sends to the mediator. All objects within the range of the listener foci become its casters. This means that the mediator must create a perception message describing all the features of those objects, in other words it must translate the main information that defines all the objects that are considered to be casters of the listener agent into agent understandable information.

After the Mediator sent all the required perceptions to all agents within its controlled zone, it sends updated information on entities affected by the agent's actions to Simulator Engine Controller. Finally, the Simulator Engine Controller can update all the changes that were provoked by the agents' actions into its own network representation.

The example network illustrated in Figure 3.3 describes the concept of the agent foci in a typical simulation time step.

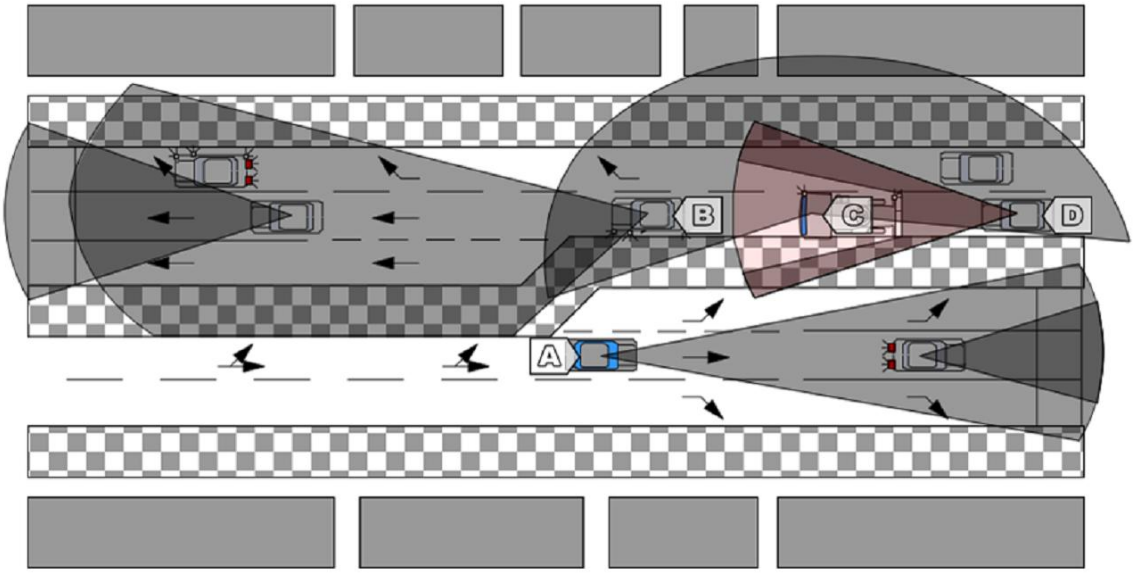


Figure 3.3 - Network Scenario exemplifying driver agents foci [2]

Imagine that in this example several time steps have already passed. So let us focus our attention on the driver of vehicle A. Based on the perception of the previous time step received, the driver decided to focus his attention on the front of his vehicle as illustrated by the gray sector of the figure. Now, assume that its desired action is to maintain its current velocity. The driver will send the mediator a message informing that it desires to maintain its velocity by inducing in its vehicle the necessary acceleration (assuming that friction exists) and that it desires to receive perceptions information about all objects inside its line of sight (foci).

In the described case, the only object that will be affected by the driver of vehicle A action is its own vehicle. The mediator then updates, in its control zone data structure, vehicle A's position based on the acceleration provided by its driver. After finishing updating vehicle A's position, it will search for all the objects within the received line of sight (foci) sent by the driver. Looking at the example scenario it is possible to identify four objects inside the agent foci:

- The "Go ahead" arrow sign on its lane;
- The "Turn left" arrow sign on its left lane;
- The "Turn right" arrow sign on its right lane;
- And a car at his front, on the same lane.

The characteristics that define these four objects are gathered by the mediator that translates them into a language that can be understood by the driver agent of vehicle A. Notice that the car in its front has the breaking lights on. Finally the mediator will send the constructed perception back to the driver of vehicle A.

The described sequence of actions that happened between vehicle A's driver and the mediator will happen with all other drivers. While another simulation time step is not started the driver of vehicle A will reason on received perception to decide its next action. Since it received the information that the front vehicle is reducing its velocity it will probably decide also to reduce its velocity. Whatever its decision, the action will only be executed in the next simulation time step.

This ends the explanation of how the MAS-Ter Labs Traffic Simulator runs the simulation and a suggestion to how the many agents in it take decisions. The suggested architecture is composed of several different types of applications that can be distributed across several processing units. This kind of modularity enables this sub-system with the capacity of being an extendable and scalable solution. It is also believed to be the best solution to guarantee a good system performance when simulating large networks highly populated.

The developed prototype described in section 4.1, in the next chapter, only contains a few of the described features. Nevertheless, the prototype presents itself as the first stage of the development of the conceptualized sub-system where some of the nuclear concepts introduced in this chapter were implemented.

So, to conclude, some details on how this simulator connects to our solution design is briefly explained now. The Simulator connects to MAS-Ter Labs Traffic Simulator and receives traffic information from it. Specifically, the positions and other characteristics of the Agents (Moveable Agents, Semaphore Agents, etc.) are sent to the Simulator. The Simulator will send back to the MAS-Ter Labs Traffic Simulator the information about the agents connected directly to it. And both the simulators include that information in their calculations. This way, the agents in both simulators will perceive the agents from the other simulator, and react accordingly.

So that we can run a very big and complex scenario with this solution design, we have to make sure that the Simulator won't get unnecessary information sent to it. The MAS-Ter Labs Traffic Simulator is very scalable, and so, can deal with the traffic simulation of a road network as big as the world's biggest metropolises [2]. So, a module should be created in between the simulators, that keeps information about the Agents connected to the Simulator, and that filters the information from the MAS-Ter Labs Traffic Simulator, only letting information pass to the Simulator if it's about Agents that are somehow in the range of the Agents connected to the Simulator.

In short, this connection will make the numerous traffic cars react to the autonomous cars that are being simulated in detail, without the need for many complex calculations involving sensors. In turn, the autonomous cars being simulated in detail will have their sensors detect the numerous traffic cars and react accordingly. Thus we achieve our main goal of being able

to simulate, in detail, one or a few complex autonomous vehicles in the middle of heavy traffic situations in huge cities.

3.2.1.2 – The Simulator

The main objective of the Intellwheels Simulator was to develop simulation software to function as a test board for control algorithms for intelligent wheelchairs. This following section will be based in Chapter 4 of [28]. It introduces the basics of the Intellwheels Simulator, in order to better explain its role in our global design solution.

The Intellwheels Simulator expanded the Ciber-Rato project that it was based on, acquiring important features which are critical for intelligent wheelchair simulation. As core functions, this application creates a virtual world, complete with map definition, where robotic agents can connect to. The simulator regulates the connection attempts, handles the communications and returns the perception of the world to the agents, similarly to what a real robot would get from the real environment around it, through its sensors.

The robot control software should treat the awareness information not discriminating it from real or simulated, therefore producing the result independently: it produces orders for every connected actuator, being real or virtual. This scenario leads to the subject of reality definitions. In fact, the usage of the same software for real situations as for virtual tests, suggests a leap forward into the augmented reality concept [44], in which virtual world objects interact with the real world.

This chapter will go through the simulator's conceptual architecture, including how the support for mixed reality [45] was implemented. It goes through the modifications made to Ciber-Rato simulation environment and the new algorithms implemented to correctly simulate Intellwheels' wheelchairs.

Being essentially based in the Ciber-Rato source code, the Intellwheels Simulator has its main basic architecture. Conceptually it is illustrated in Figure 3.4.

In a higher abstraction level, it consists of a central simulation server to which every agent, independently of its type, will connect to. Furthermore, to have a structure as modular as possible, the agents are external applications, developed in any kind of language and running in any type of operating system, must connect via IP and UDP protocols [65]. Through this obligation, the spectrum of possibilities for agent development is greatly broadened.

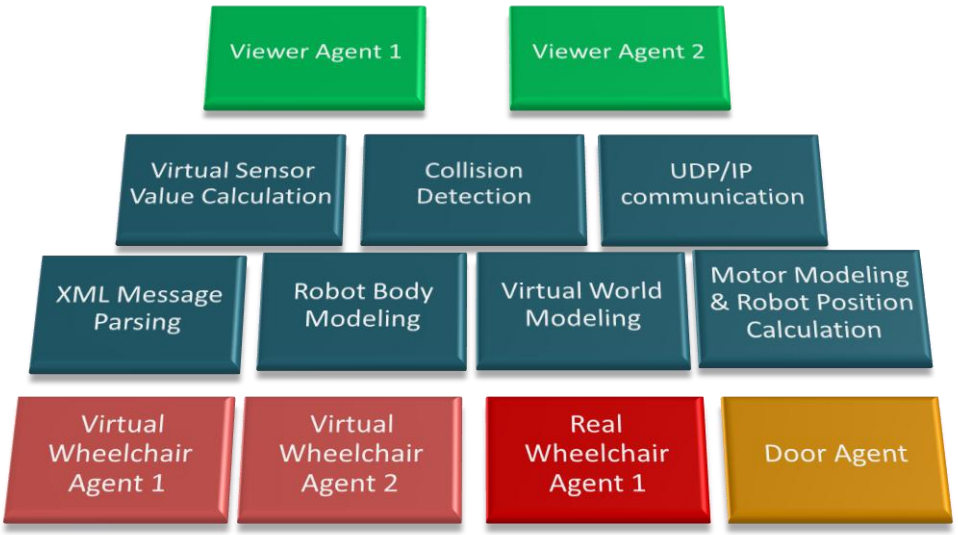


Figure 3.4 - Intellwheels Simulator's architecture [28]

The simulator server is responsible for all calculations concerning simulation (collision detection, position calculation, wheel motor emulation and world perception sensors' values). It is also the assurance of communications between every intelligent agent (independently of their type). Viewer agents are able to graphically draw the modeled world, as the simulator sends them map definition, and robotic agents' positions. These agents, on the other hand, have a more intense interaction with the server. They not only receive information concerning their virtual sensors' perception but also need to send power input orders to their virtual motors.

The physical implementation of this architecture resulted in the usage of laptop computers. They house the simulation and agent applications, which connect through a Wi-Fi wireless network under protocol 802.11g and cabled Ethernet connections, as illustrated in Figure 3.5.

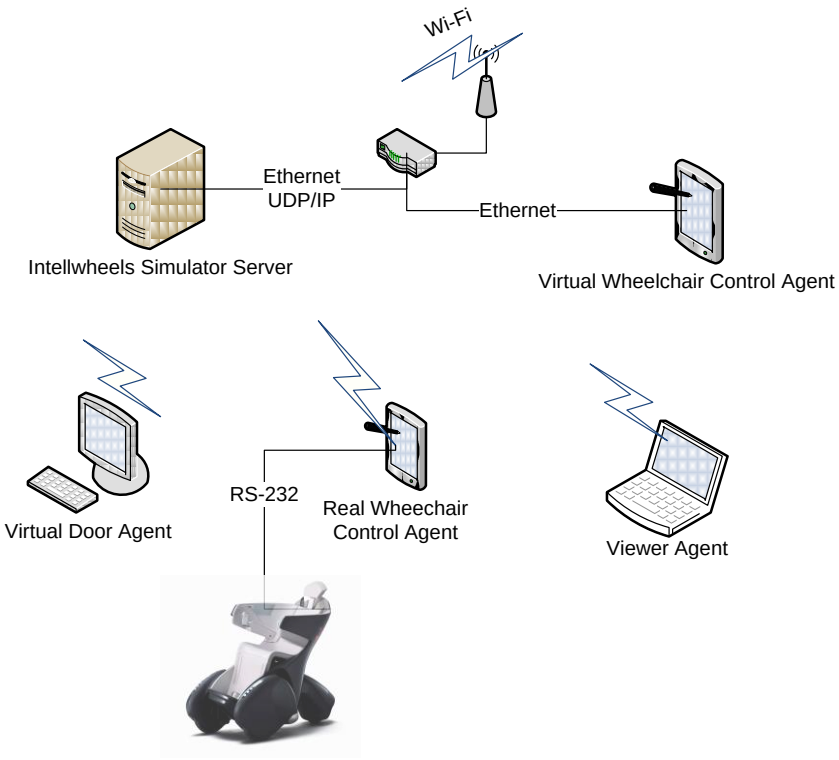


Figure 3.5 - Technology implemented for Intellwheels simulation [28]

The core of the system is a central computer that runs the simulator server, to which every agent application connects to. The information exchange is made through XML messaging which ensure human and machine-readable content [66].

The system is composed by a simulator server, which can be a Linux OS or a Windows OS (although, during this project, it was only compiled a Windows version of Intellwheels Simulator). It sets a UDP listen port, to which it will await agents' registration requests. Through specifically ports, attributed individually to each agent, it sends information of their concern: sensor perception (in case of robotic agents) and map, collisions and positioning information (in case of viewer agents). The simulator is also capable of accepting incoming messages to these ports to update the simulation: robot action orders and simulation commands from the viewers.

Now we have an overview of the Intellwheels conceptual architecture and the technology it used for its development and implementation. It presented the concepts of the multi-agent system and the support for external application connection. Intellwheels provides a new mode of Intellwheels simulation where it is possible to connect not only agents for virtual robots but, at the same time, real Intellwheels controllers which can, themselves, work on augmented reality mode. The simulator, on the other hand, will be under an augmented virtual environment, receiving information of real wheelchairs and calculating their interaction result with the virtual objects modeled.

Every Agent is modeled with a rectangle shaped body, with configurable center of movement, height and width. Additional physical characteristics, such as the acceleration curve and the maximum speed it can achieve were also modeled. The proximity sensors can be defined by their opening cone of sight and their orientation, in degrees, and can be placed through X and Y coordinates relatively to the robot's center.

Now, it's easy to understand why this simulator is the choice for the role as the robotic simulator that is the center of our solution design. There's great potential considering its great abilities to work with mixed realities and its great overall flexibility with communications and modeling. So, to conclude this section, the functionalities that extend to it in our solution design are as follows:

It does the detailed simulation of the autonomous vehicles that are to be simulated. It directly tests for collisions, calculates sensor values, receives information from the Agents' actuators and uses it in its calculations, validates V2V/V2I communication between the Agents connected to it, and the agents that are connected to other connecting simulators (like the MAS-Ter Labs Traffic Simulator).

It receives information about relevant Agents connected to the other simulators, and takes them into account for its own simulation's calculations (for example, receiving information about nearby cars that are in the traffic, being controlled by the MAS-Ter Labs Traffic Simulator, and taking them into account when calculating collisions, Agent's sensor values, etc.).

It can be modified to simulate detailed physics and mechanics details such as the environment being able to affect Agents' movements and sensors, affecting the random sensor noise, and also affecting wireless V2V and V2I communications.

It can also be modified to simulate failures, for example, not sending information to an Agent regarding a sensor value, or sending corrupt data, or ignoring actuator commands, etc. This would help test Agents that are programmed with critical systems, to see how they'd react to such failures: Invaluable if our lives are to be trusted to autonomous vehicles.

Also, it could receive and load simulation maps from other simulators.

3.2.1.3 – Agents

The Agents connect to the simulator and send all the information it takes for the simulation to know the necessary characteristics about the Agent. This includes component positioning, and type of Agent. It can be a Driver Agent, in which case it will have a car assigned to it. It can also be an Infrastructure Agent, like traffic lights, intelligent street signs, communications access point, obstacles, malfunctioned vehicles, etc. Some of these agents are an integral part of the road network, and will connect to the MAS-Ter Labs Traffic Simulator instead of connecting to the Intellwheels simulator, but those Agents communicate

with the Intellwheels Simulator's Agents and with the Viewer (essentially another Intellwheels Agent, explained further in 3.2.1.4) through V2V and V2I communication.

During the development of the Intellwheels Simulator, some simple and generic controlling agents were created to overcome the initial difficulties in the Ciber-Rato study, to test UDP [65] communications with Ciber-Rato as well as to test XML messaging [66]. The main purpose of this application was to have the ability to connect to the original Ciber-Rato as a robotic agent and as a simulation viewer agent [28]. This game type Agent, is a simple Agent that doesn't have decision-making abilities. It has an interface to the user so that he can directly control a vehicle in the simulator. This will come in hand for testing and debugging the simulator. The Agent is able to send its characteristics (from size, center of movement, acceleration to top speed and sensor definition) and basic commands for movement (forward, back, turn left and right), as well as receiving information (for various types of sensors) which allows full test of autonomous vehicles.

As is mentioned in the previous section, real vehicles (and other agents that have a physical manifestation in the real world) can connect to the simulator thanks to the implementation of the "Real" type of robot in the Intellwheels Simulator. And send and receive information regarding the mixed realities of both the virtual world and the real world. This allows for simulations where a real vehicle is able to avoid virtual obstacles.

3.2.1.4 – Viewer

Visualization is a very important aspect in simulation. It is the means to easily understand the large quantity of information that results from a simulation, which would otherwise be too great or complex for most people to fully grasp in a reasonable amount of time. Graphical representation is now taken for granted and it would be unconceivable to develop a simulator without some sort of visuals. Humans construct and comprehend the world in a graphical way for we have an innate ability to process graphic information in a preconscious, involuntary fashion, similar to breathing [67]. Visualization is, therefore, the foundation for our understanding of the results of a simulation.

This section will explain the architecture and workings of the Intellwheels Viewer, and is heavily based on chapter 6 of [28]. The viewer is a very important part of the simulation, and must be credible if the simulation is to be useful. There must be enough interactivity during the simulation, in order to easily display the information the user intends to see. The 3D camera, or perspective, must be easy to manipulate, and the animation and movements should be free of glitches so that it's easier to take results and conclusions from the simulation. It should be possible to view the entire simulation as if one was sitting on the real vehicle.

Taking these concepts into consideration, it became clear that the original viewer for Ciber-Rato [68] would not fit the needs of the Intellwheels Simulator. So, the decision to develop a new viewer from scratch was made.

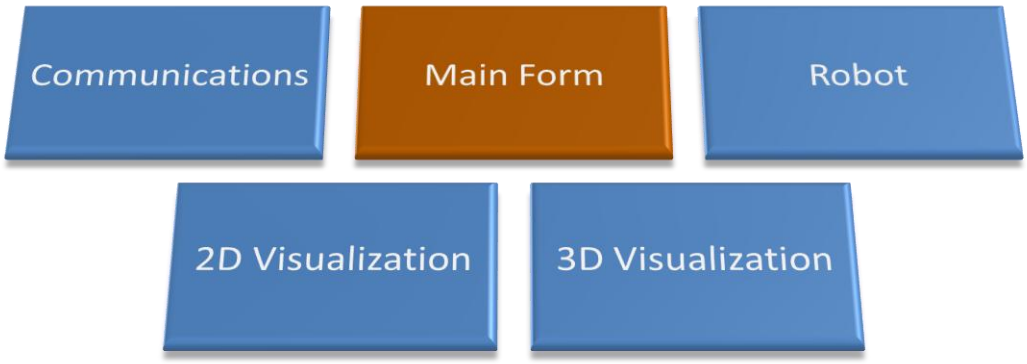


Figure 3.6 – Intellwheels Viewer architecture [28]

Conceptually, the viewer developed contains 5 main software modules (Figure 3.6):

- The main module (Main Form) is where everything comes together. It allows the communication configuration activation, robot selection (for selected information display) and visualization selections. This main module is also responsible for storing the information concerning the map's characteristics and wall definitions;
- The Communications module contains the IP/UDP configurations and handling and XML message parsing. This module ensures that the messages sent by the simulator are correctly received and transformed into system variables for use by the other modules;
- The Robot module is where all the robots' information is stored. Their physical characteristics, status, position and orientation are stored here. It is through this module that the rest of the application will access updated and ordered information on the robots, either for show purposes or for calculations;
- The 2D and 3D Visualization modules have similar functioning modes. They access the map and robot's information and reproduce them graphically. The only special characteristic of the 3D module is that it calls and uses external OpenGL libraries [61][62]. Since the simulator only provides 2D information, the 3D viewer will generate the Z axis coordinates in such a way that will make it easier to perceive the simulation.

The developed viewer successfully creates credible graphical representation of the simulation and contributes to the entire simulation project with an evolved scenario. The Intellwheels Viewer implements drawing algorithms and 3D model loading functions that

produce a fluid and realistic visual representation of the simulation. Moreover, the application itself is flexible enough so that it can be easily expanded for increased performance or modified for different purposes.

To further improve our solution design, it should be able to send information of rendered images back to the Simulator for optical sensors like cameras (to test vision-based algorithms), including infrared cameras. Those rendered images, or video streams, are in turn sent to the relevant Agents so that they will analyze the images and make decisions based on those, with their own Artificial Intelligence algorithms. This will improve the simulation of autonomous vehicles a lot, especially if the 3D Viewer graphic engine is updated for very realistic video output. Other sensor information should be used by the viewer to display, for example, the ranges of the sensors and the range of wireless communications. And finally, it should be able to render the visualizations differently according to the environment conditions like weather effects, luminosity, daylight or nighttime, fog or clear sky. All these conditions, if rendered correctly, will greatly increase the accuracy and realism of the tests done in term of vision-based systems and their abilities to recognize objects and signs in conditions far from the ideal.

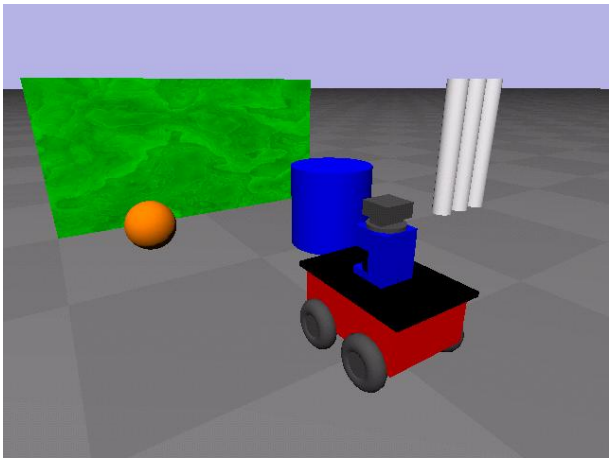


Figure 3.7 – Screenshot of Gazebo. A Robotic Simulator with a sample Graphic Engine[4]

[4] Image source: http://playerstage.sourceforge.net/index.php?src=gazebo

Figure 3.8 – Screenshot of the Wheelman game[5]

As you can imagine from the above pictures (Figure 3.7 and Figure 3.8), object recognition and identification in each of the two screenshots above, are totally different matters. A vision-based system that is successful in a simulation using the graphic engine depicted in Figure 3.8 is much more likely to perform well in the real world, than a system that is only successful in a simulation that is running a graphic engine similar to the one in Figure 3.7.

3.2.2 – Prototype Scope

The solution design is very complex and comprehensive. The idea now is to develop a prototype with some of the basic functionalities in order to test the basics of the proposed architecture. The complementary functionalities will be part of further development and future work.

There are two main reasons for this:

- The implementation of a solution design like this would hardly fit in a Master's thesis;
- Testing should be done as new functionalities are added, while developing a complex application.

[5] Image source: japamd, "Wheelman". 6 Apr 2009. Accessed at: http://www.adrenaline.com.br/forum/pc/249430-wheelman-topico-oficial-3.html

The following table presents some relevant specific objectives and functionalities of the solution design, sorted by groups, and then by relevance to the project. The relevance is based on how related the specific objective is to our main goals. It is also a bit based in the novelty of the given feature, as well as if any other relevant features depend on that one.

There's also a very rough estimation of how easy they are to implement, but that's very subjective.

Table 3.1 – Summary chart with the specific objectives

	Relevance	Easiness	Specific objective description
Simulator	10	7	Collision detection (done)
	8	3	Map reformatting
	4	4	V2V/V2I communication controlled by the simulator
	2	3	Environment affecting physics and/or sensors
	3	4	Failure simulation
	2	4	Sensor noise
	2	4	Communications degradation
Agents	10	10	Agents connecting to the simulator (done)
	9	9	Agents sending actuator values (done)
	9	9	Agents receiving sensor values (done)
	9	8	Game type Agent (done)
	9	8	Agents send their traits (done)
	1	3	Real vehicle connecting as an agent
	1	3	Real vehicle reacting to virtual sensor responses
	7	5	Infrastructure Agents
	4	4	Obstacles Agent
3D Viewer	10	5	Implementation of a 3D Simulation Viewer (done)
	10	9	3D simulation viewer connects to the simulator (done)
	5	6	Range indicators for Sensors and Communications
	5	3	3D simulation viewer streaming images back to the optic sensors
	3	5	Environment affecting visualization
Traffic Simulators	10	8	Connects to the MAS-Ter Lab Traffic Simulator
	8	6	Loads a road network map from MAS-Ter Lab traffic simulator
	8	5	Traffic information received from the MAS-Ter Labs simulator
	8	4	Agent information sent to MAS-Ter Labs traffic simulator
	4	7	Connects to pedestrian simulator
	2	3	Receives information about pedestrian traffic

Relevance: Highest relevance from 10, to the lower relevance, 1.

Easiness: Highest easiness to implement from 10, to the highest difficulty, 1.

Some of these functionalities happen to be implemented already, in the latest versions of the simulators that we're using. Those that made it to the prototype are marked with “(done)” after the description field in the above table (Table 3.1).

But for the implementation of the prototype done during this thesis, another set of functionalities was done, ones less relevant for the simulation functionalities, but more relevant to testing the concept during the time-frame of this dissertation.

The following functionalities are those that we considered to be fundamental for these first tests:

- Simple XML maps to use during the prototype's simulators' tests (the same map, in different formats, for each simulator);
- 3D Models and Textures for better immersion in the tests;
- Functional playable type agent to demonstrate collision tests, physics and sensors;
- Modification of Intellwheels simulator to adapt to simulation of autonomous vehicles;
- Implement vehicle's sensor interaction with the vehicles whose positions are being communicated by the MAS-Ter Labs traffic simulator.

The changes that were implemented during this thesis are explained in the next chapter (Chapter 4), and the tests that were done are described in Chapter 5, along with the importance of those tests.

3.3 – Summary

In this chapter, the concept of the solution design was developed, as well as the ambitions of the global project. Recently, there's been a lot of simulators being developed here in LIACC, and it was about time we tried to connect them all together to increase their capabilities. The steps taken during the dissertation were also discussed, as well as the tools that were used.

The scope of this whole study is mainly about the inter-connectivity between different functional simulators and their modules. This wide theoretical scope required a very comprehensive study and we can't try to do everything at once when it comes to implementing something as complex as this, because testing needs to be done as the functionalities are steadily incremented. To balance out the wide scope that was set for the concept, a narrow one needed to be defined for the prototype implementation so that the efforts would not go astray for the tests. After all, the point was to develop a prototype with some of the basic functionalities, so that the concept could be tested.

Chapter 4

Prototypical Development

In this chapter, the development of the prototype is discussed. The functionality, the architecture, and the basic workings of each of the simulators software used are explained. Also discussed are the changes and implementations made to the existing software in order to implement the prototype of the solution design exposed in the previous chapter. Also explained here is the creation and usage of exterior elements like the XML road network maps, XML Intellwheels maps, 3D models and textures.

Also suggested at the end of each of the following sections are hypothetical implementations to the software to achieve the solution design described in the previous chapter. These could serve as a guide should this work be continued and developed in the direction of implementing the full solution design.

4.1 – MAS-Ter Labs

Here, the prototype implementation of the MAS-Ter Labs Traffic Simulator will be briefly described. Note that the implementation is far simpler than the solution design of that same simulator. Only the relevant differences between the implementation and the design will be introduced. For more information about the implementation of the MAS-Ter Labs Traffic Simulator prototype, see Chapter 4 of [2]. The next part of this section will be heavily based on that chapter.

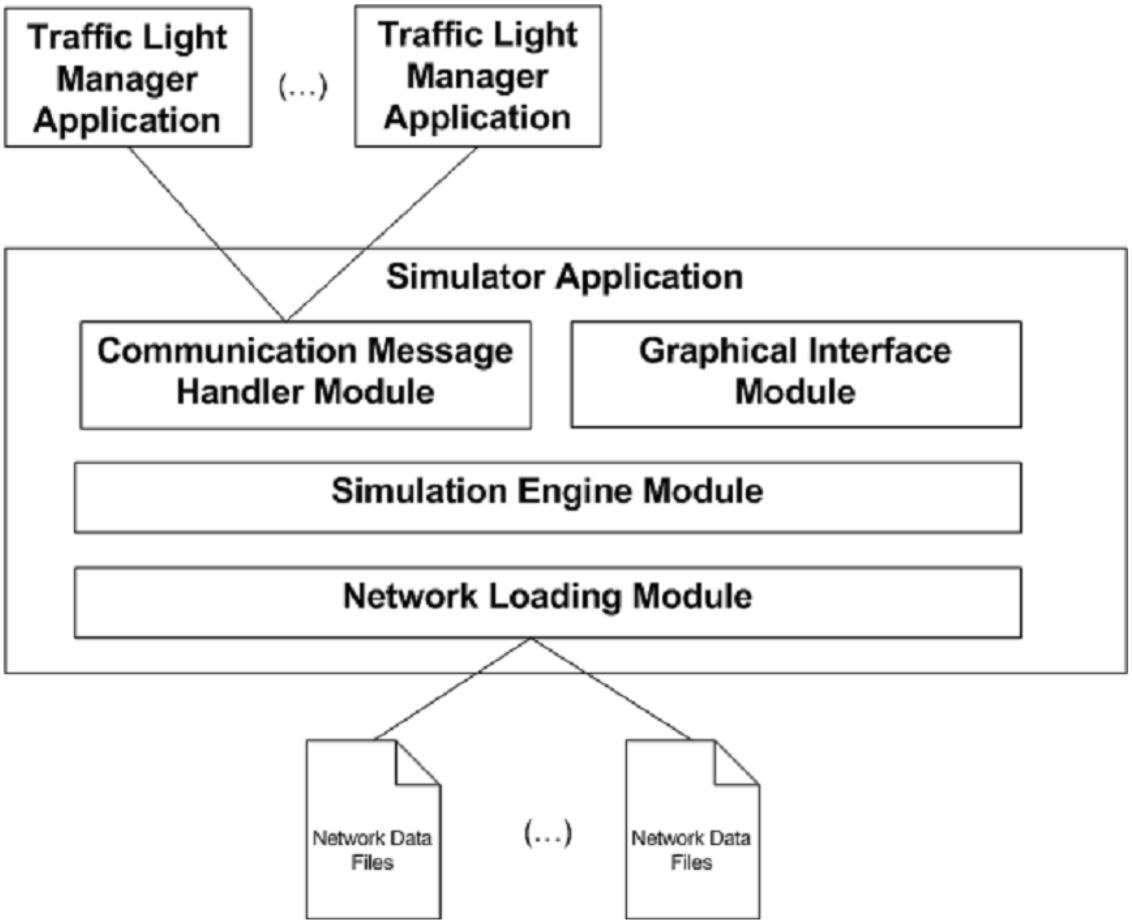


Figure 4.1 - MAS-Ter Labs Traffic Simulator Prototype Architecture Overview [2]

There are four main modules, as illustrated by Figure 4.1, in the simulator application:

- **Network Loading Module** – The network loading module contains all classes responsible for loading networks from files or geo-referenced databases into the simulator. The developed prototype allows only networks to be loaded from an XML file. Nevertheless, it's constructed in a very modular way to allow an easy implementation of loaders for other type of files or databases;
- **Graphical Interface Module** – The graphical interface module contains all classes responsible for the graphical user interface. The visualization of the simulation is made in 3D by using the Qt framework inner module that provides access to OpenGL library features. This module has been logically separated from the Simulator Engine Module so that more graphical interfaces can be implemented and easily included in the current developed prototype. The objective of this separation is to allow the physical separation of this module into autonomous applications in the future, which can display different graphical representations

of the information output provided by the simulator engine module. This will allow having several graphical interfaces illustrating one single simulation;

- **Simulator Engine Module** – The simulator engine module contains all classes responsible for the simulation execution and information retrieval. The current section details the network model used in this module and the execution cycle of a simulation time step;
- **Communication Message Handler Module** – The communication message handler contains all classes that are responsible for handling outgoing/incoming messages from or to other applications. The prototype handles only messages for traffic light intersection control; nonetheless, these features were constructed in a very modular way to allow an easy integration of other type of messages that can be developed in the future;
- **Traffic Light Manager Application** – The Traffic Light Manager Applications are allowed to connect to the simulator, request a list of traffic light intersection and register as a traffic controller of a specific traffic light intersection. After registering the simulator sends this type of applications information about the ongoing and outgoing traffic flow of the controlled intersection. These applications may also change the traffic light plan of a controlled intersection.

This implementation of the prototype works rather well for what we want, considering that it wasn't originally designed for it. See [2] for the test results and conclusions drawn from it.

Unfortunately, the way it was implemented makes it harder to test it for the desired purposes when it comes to testing the global design solution studied here. The way that the Moveable Agents are processed stops us from being able to directly affect them with inbound messages. In order to accomplish that, heavy modification of the main module of the prototype is required. And that isn't the scope of this study. But it's still possible to extract information from its Moveable Agents using outbound messages. And fortunately, the Intellwheels Simulator has the ability to update its "Real" type of Agent with inbound messages, and so, the functionality of loading traffic cars into the Intellwheels Simulator could be tested if the connection between the two simulators had been successfully implemented.

One modification that occurred to this prototype, prior to this dissertation, was part of another study, about V2V and V2I communications [9]. Among the modifications during that study, was one that consisted of communicating the positions of all the Moveable Agents in the traffic simulation through a TCP stream socket connection through port 8000.

Unfortunately, the successful communication between the modified MAS-Ter Labs Traffic Simulator and the Intellwheels Simulator was not successfully accomplished by the time of

this study's conclusion, and this hindered the amount of tests done to our prototype, as well as the results achieved.

The difficulty lied in a few facts, the main one being that while the output message was implemented as a single string being sent through a stream socket via TCP, the input format of the Intellwheels Simulator is XML messaging sent via UDP. A module to convert one format into the other would have to be implemented in one of the sides, and that was not accomplished on time.

Further implementations to get us closer to the solution design would be:

- Changing the main module of the MAS-Ter Labs Traffic Simulator to allow its Moveable Agents to be moved through an outside source, to enable the communication of the positions of Intellwheels Simulator Agents to the MAS-Ter Labs Traffic Simulator, and have those be perceived by the rest of the Agents being controlled by the MAS-Ter Labs Traffic Simulator;
- Changing the Communication Message Handler Module so that it could send road networks imported from XML files to other simulators connected to it. Those could include traffic light information, so that the Intellwheels simulator could import and use that information to load a map from it, for its own simulation;
- Exchange of V2V and V2I messages between the Moveable Agents inside the MAS-Ter Labs Traffic Simulator and the Agents connected to the Intellwheels Simulator.

4.2 – Intellwheels Simulator

To explain the modifications needed in this simulator, some of its aspects must be further analyzed, namely, its Real type Agents, and its XML communications.

An important part of the simulator is its capability of admitting the connection of different robotic agents. Specifically, it is possible to distinguish an agent that controls a virtual robot from an agent that controls a real robot, meaning that the simulator can register two types of robotic agents: "Real" and "Simulated".

If a "Simulated" type robotic agent connects to the server, it will treat it as a controller for a purely virtual robot. The simulation will then provide it with the world perception, through the modeled virtual sensors. It will also accept incoming XML messages containing actions that set the desired input power to be given to the motors which, consequently, will be a parameter that the simulation engine itself will use to calculate the robot's following position. It is a completely virtual environment.

But in a case where the robot's type is "Real", the simulator will regard this agent as an application controlling a real robot, in a mixed reality mode. It is expected that the agent

provides the simulation with the robot's X and Y coordinates (in meters) as well as the angle (in degrees). This allows the virtual world modeled in the simulator to expand with information concerning the real wheelchair. On the other hand, knowing the real wheelchair's position, direction and physical characteristics, the simulator can virtually insert sensors on to it and calculate their values. As an example, the simulator could detect the proximity of the real wheelchair to any other object in the simulation, being virtual or real, like another robot. In sending this new data to the real wheelchair, the simulator is augmenting its reality perception, now acknowledging more information than it could by itself.

This kind of scenario confers the simulator a mixed reality support characteristic that greatly increases the testing capabilities of the software. The robot prototype numbers and costs are no longer obstacles in cooperative and complex experimentations. And these capabilities that the Intellwheels Simulator has are a great potential for our solution design. These "Real" type Robots can be used to simulate our traffic cars. Since their characteristics and status are updated from external applications, changes are implemented so that this information comes from the MAS-Ter Labs Traffic Simulator. With the module described near the end of section 4.1, to filter the information between the two simulators, this will allow the solution design to be able to deal with immense amounts of traffic, since only the relevant traffic vehicles are present in the detailed simulation's calculations.

The Extensible Markup Language (XML) is now a widely used standard, mainly due to its characteristic of facilitating communications across different systems [66]. Specifically in the Intellwheels Simulator's environment, it is expected that different applications, developed in different platforms exchange data in an easy human understandable way. Ciber-Rato originally made usage of this with proven success, so, the concept was kept for the Intellwheels Simulator. XML tags were defined for every kind information exchange between the simulator and the agents to register their physical characteristics, sensors (there is a limit of 8 sensors in the current implementation of the Intellwheels Simulator), to move the robot, etc. But the way the robot is moved depends on its type.

For a "Simulated" robot type, the simulator will be responsible for new position calculation. It will also handle the modeling of the motors and thus the translation of the input power to robot's speed. As such, to move the robot, its controlling agent must send a XML message, sending the power values to give to each motor.

In case the connecting agent is "Real" robot type (in augmented reality mode) the simulator relinquishes the task of position determination to the agent itself. Conceptually, the simulator is working in augmented virtual mode and so, the agent must inform it, at all times, it's X and Y coordinates (in meters) and orientation (in degrees). Through this action it is possible to allow interaction between the real and virtual world, particularly updates on virtual sensor value calculation and collision detection.

Not only these functionalities enable us to move traffic cars around while simulating a complex autonomous car, but it also allows us to merge realities by having a real car, equipped with sensors and actuators, participating in the simulation. The Intellwheels Simulator takes care of merging the information of the virtual sensors into the information coming from the real sensors so that the real car can perceive virtual objects as well as real objects. The simulator can also test for collisions between the real car and the virtual objects to detect Agent failures.

Unfortunately, as stated in the end of section 4.1, the implementation of the module that would allow the communication between the Intellwheels Simulator and the MAS-Ter Labs Traffic Simulator was not complete, and so, we could not test the ability of moving traffic cars in the Intellwheels Simulator.

Fortunately, some aspects of the solution design can still be tested. If it is shown that the "Simulated" vehicles (with the simulated sensors) detect the "Real" vehicles (whose information would be updated with the position of traffic cars), then it is proven that the "Simulated" autonomous vehicle detects traffic vehicles and can react accordingly once the connection between both the simulators is made, and it is also proven that collision tests are also performed on the vehicles. For further information about the tests, refer to the next chapter.

Another implementation that could be done to further achieve the solution design's goals would be:

- Altering the physical and mechanical calculations of the simulation so that the movements and behaviors of the robots would be closer to a car's, rather than a wheelchair's;
- Implementing new sensors, to adequately test autonomous vehicles, such as: Luminosity, Optical camera, Infra-red camera (the viewer has the ability to render images and videos and stream them directly to these sensors), Laser scanner (LIDAR), Ultrasound, Inertial Measurement, Radar, and speed measurement, to complement the ones already implemented in the Intellwheels Software;
- Creating a XML parser that would read the XML files that are loaded to the MAS-Ter Labs Traffic Simulator and transferred to the Intellwheels Simulator. This parser would read the roads from those XML files and calculate the positions of the obstacles, or walls (the spaces in the map that are not the roads) to create a map in the Intellwheels Simulator format. By defining different parameters, it could create different obstacles automatically by creating the sidewalks by the road, and, only further away, blocks of buildings.

- Receiving information from the traffic lights, and other infrastructure agents that are not connected to the Intellwheels Simulator, but are instead connected to the MAS-Ter Labs Traffic Simulator, and being able to both use that information in its calculations, and send it to the Intellwheels Viewer;
- Achieving V2V/V2I communication between agents connected in the different simulators;
- And all the other features that were suggested in Section 3.2.1.2. regarding environment variables, weather variables, sensor noise and failure, and sending all that information to the Intellwheels Viewer.

4.3 – Simulation Agents

The information it takes for the simulation to know the necessary characteristics about the Agent is sent to the simulator through the connection between them. This includes component positioning and type of Agent. During the development of the Intellwheels Simulator, simple and generic controlling agents were created to overcome the initial difficulties in the Ciber-Rato study, to test UDP [65] communications with it as well as to test XML messaging [66]. These are simple Agents that don't have decision-making abilities. There's an interface to the user so that he can directly control a vehicle in the simulator. The Agent is able to send its characteristics, and basic commands for movement, as well as receiving information which allows full test of autonomous vehicles.

The simulator, by default, is listening to every communication sent to port 6000. Once it receives a message, it is analyzed and checked for a Robot or Viewer agent XML registering message. If the message does not match with any of these, the message will be ignored. If it is a robot connecting and if the Id is specified, it must not already be in use in the simulation, otherwise the registry will be denied. In a successful registration, the simulator will send a XML message confirming it. In this first message, the UDP datagram sent will specify a new port, to which every robot action (or viewer command, depending on which agent type is connected) must be sent. The simulator binds the robotic agent's sending IP and port to this new port. No communication can be done to any other port, from this point forward. The main reason for this behavior is to ensure that port 6000 remains free for new robot registration [28]. The application also allows custom message sending. This permits that any message can be sent to the simulator, testing its response.

The only changes made regarding tests was that the Agent's characteristics like size and speed were changed to approach those of an autonomous vehicle instead of an electric wheelchair. And these changes don't require programming as this characteristics can be directly inserted in the dialog of the Agent interface window.

Things that could be implemented to further approach the prototype to the solution design would be:

- Developing this interface to behave more like a car's handling, instead that of a wheelchair;
- Improving the agent interface so that it could connect to a real vehicle reporting as an agent;
- Adding the functionality of connecting other types of robotic Agents, for example, intelligent traffic signs, obstacles like broken vehicle and emergency traffic signs, road infrastructures, etc.

4.4 – Intellwheels Viewer

All the major changes to the code of this application were in the 3D Visualization Module. It is the module responsible for accessing the simulation's information and reproducing it graphically. It calls and uses external OpenGL libraries [61][62] to render a 3D imagery output of the simulation. Since the simulator only provides 2D information, the 3D viewer will generate the third dimension's coordinates in such a way that will make it easier to perceive the simulation.

Taking Matthew Rohrer's conclusions [67] on the preconscious image processing capabilities of human beings, the visualization for this simulation is more effective if we have the option to see the simulation from inside the vehicle.

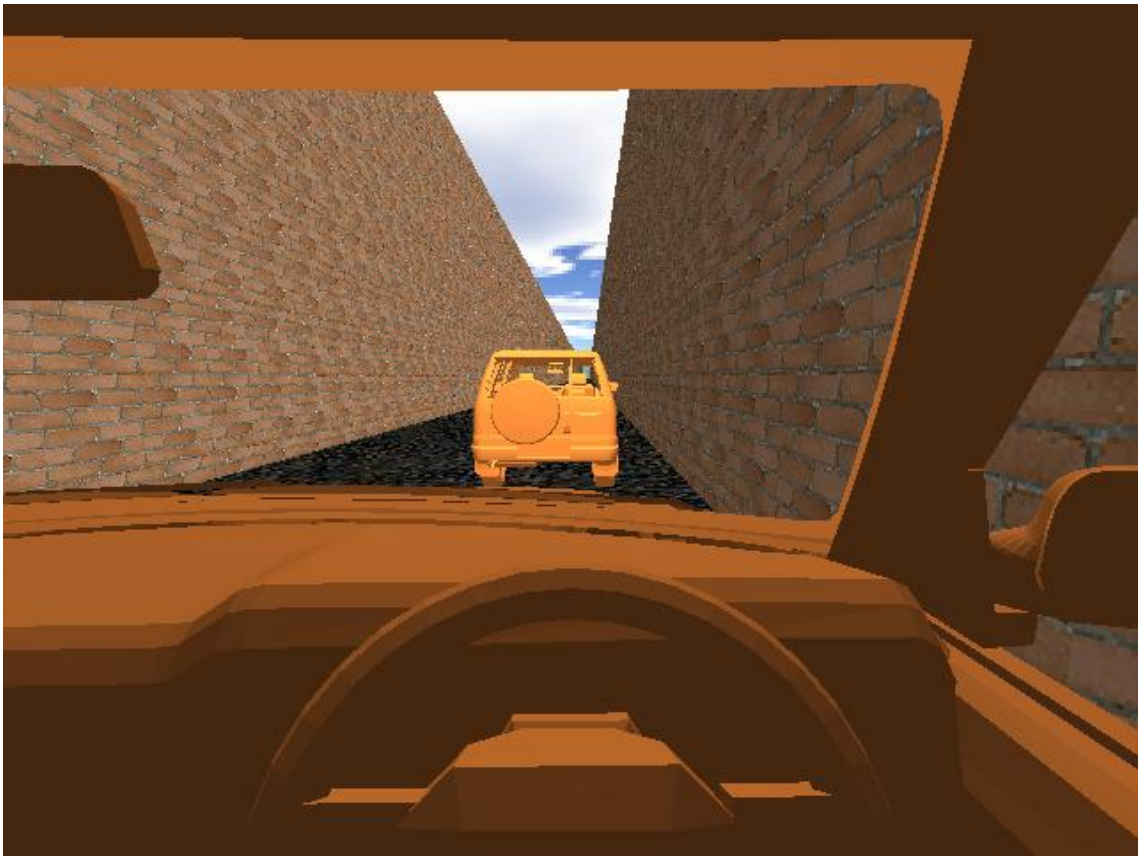


Figure 4.2 – Modified Intellwheels Viewer 3D, 1st person view

Through OpenGL libraries, it became possible to draw the objects by defining their surfaces, and vertexes, relatively to a given center. As an example, drawing a cube is done by indicating the corner coordinates of each of the six faces. When the camera viewing point is set, the OpenGL motor itself automatically handles the complete redrawing of the shown image. It continues to do so automatically, once the camera view position changes. The cycle is repeated many times per second so that the animation seems fluid, like watching a movie or playing a video-game.

As for the 3D models, the simulator models the vehicles as simple 2D horizontal rectangles with no height, and as long as they occupy the same space in X and Y coordinates on the 3D viewer, there is no restriction on how the object itself is drawn. In fact, if one actually sees the car, instead of a mere cube, it makes the visualization (and consequently the simulation itself) much more credible. More information regarding the 3D model of the vehicle used in the simulator is in the next section of this document (Section 4.5).

With the map and the 3D objects loaded, the drawing of the simulation is done by resizing and translating the objects according to the information stored for each robot (position, dimensions, orientation, etc.).

Various modifications were done to the Intellwheels viewer in order to improve its functionality:

- The function that draws each agent was changed for the car to appear with the correct orientation, since the wheelchair's model had different axis information;
- The function that draws the floor, the sky, and walls was changed as well, to adapt those to the big difference in the simulation area required to simulate a road network, since the Intellwheels Viewer was initially prepared to draw a simulation that ran inside a room;
- The function that renders the 3D world of the simulation had to be modified in order for it to be able to draw objects that were further away from a room's length, it now should be able to draw objects that are 10 kilometers away from the camera, as opposed to the previous 50 meters;
- Changed the functions related to keyboard input and to the camera controls in order to increase the speed at which one can move the perspective being rendered, since that the old camera speeds were adapted to navigating around a room, it would take literally hours to move the camera to the other side of a small town.

The following changes should be done in order to further improve the viewer and to get it closer to the functionalities proposed in the solution design. Specifically, changes in the way that the Main Form module and the Communications module are implemented, in order to make it possible to pass more information from the Intellwheels Simulator to the 3D Viewer, and also changes in the 3D Visualization module that would enable it to render aspects of the simulation based on that information, for example:

- Range of robot sensors and communications, in order to draw a representation of those as shapes around the vehicle, so that the user can quickly grasp when obstacles are or not in the range of the sensors or communication devices;
- Rendering the environment, weather effects, daylight, fog, nighttime, etc.

Other functionalities could be implemented by allowing the 3D Visualization module to send information back to the Intellwheels simulator, through the Communications Module. That would enable it to render more than one 3D imagery output, from different perspectives (an optical camera, for example) allowing the Viewer to send a video stream back to the simulator, destined for a specific Agent, so that it analyzes the video stream and makes decisions based on those, with its own Artificial Intelligence algorithms. This, allied with a major graphic engine upgrade, would make the testing of vision-based algorithms much better.

Figure 4.3 – Screenshot of the Need for Speed game[6]

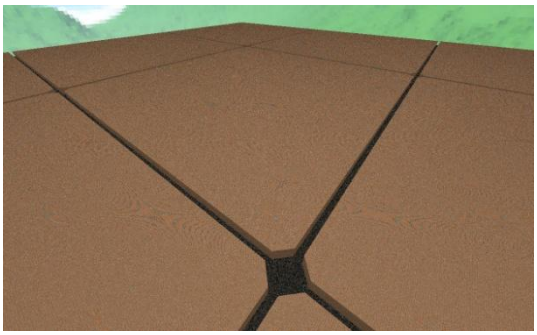
4.5 – XML Maps, Textures, and 3D Models

Let's introduce the five XML files that describe a road network for the MAS-Ter Labs traffic simulator, since most of them were modified for the prototype to work properly. The usage of XML files allows an easy processing by the programs, it is concise, formal and human-legible [66].

- **Network Main XML file** – This is the main XML file of the network. It contains the name and paths to the other four XML files that describe the road network. It also contains information about other simulation parameters like the value of the time step, time multiplier value, and the priority rule that drivers should follow in a prioritized intersection.
- **Road XML file** – This file contains all the necessary information about the entire roads network. For each road, it contains information about the two intersections that are connected by the road, the road segments that belong to it, as well as information about how they are connected to each other, the number, positions, length, width and direction of lanes in each road segment, as well as information about which are adjacent to the two intersections.

[6] Image source: http://recensioni-videogiochi.dvd.it/images/Need_For_Speed_Pro_Street/need-for-speed-pro-street-04-l.jpg

- **Intersections XML file** – This file contains all the necessary information about the entire intersections network. For each intersection, it defines the position coordinates, the area that it occupies, the source and drain roads, as well as the directions allowed. Based on such directions and the intersection area, the intersections inner road segments are dynamically built by the MAS-Ter Labs traffic simulator. In case of being a source node, it also specifies the traffic flow that it should generate, percentage of cars and trucks among the vehicles, and the trip assignments set, or routes set, that it should assign to the vehicles spawning there. If the intersection is controlled by a traffic light, then the id of the traffic lights' plan is specified and read from the traffic light XML file.
- **Traffic Light XML file** – This file contains all the information of the traffic light plans that will be used in all of the intersections controlled by traffic lights, like all the information about position, direction, and internal timers of those traffic lights.
- **Trip Assignment XML file** – This file contains the information about all the possible trip assignment vector sets (also known as predetermined routes) for all the source intersections (network entry points, where the vehicles will be generated). A percentage is associated to each trip assignment vector, representing the probability for a driver to choose one of the trip assignment vectors as its desired path, ending in a drain type of node. This decision happens when a vehicle enters the network in one of the source intersections.



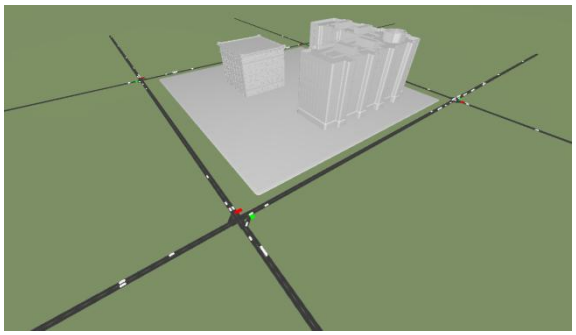


Figure 4.4 – The XML map adapted for the Intellwheels Simulator

Figure 4.5 – MAS-Ter Labs road network map used in the prototype

All the existing road network maps had their origin coordinates somewhere near the center of the map. But the Intellwheels simulator doesn't deal well with negative coordinates. So, an existing map was modified to ensure compatibility with the Intellwheels simulator. In short, the whole network was moved to the first quadrant (+,+) so that only positive coordinates existed in the map. The Network Main XML file remained unchanged

since the simulation parameters didn't change, but the coordinates of the roads were changed in the Road XML file to reflect their new positions. The same was done with the intersections and with the traffic lights, in the Intersections XML file and Traffic Light XML file, respectively. The Trip Assignment XML file remained unchanged as well, because it has no references to coordinates whatsoever.

The Intellwheels simulator uses a different kind of XML file to create a map. The main difference between the previous map and this one, is that while the MAS-Ter Labs traffic simulator creates the map from a series of roads and intersections, the Intellwheels simulator creates the map from a series of walls and obstacles. The map outer limits are a rectangle defined by its height and width. Inside the limits there can be walls which are defined by the coordinates of their corners and by its height. For competition purposes, a wall can have different heights, which affect the compass sensor of the robot: if the wall is high the beacon will not be in the robot's line of sight, thus disabling compass sensor readings. An ordered sequence of consecutive corner coordinates (minimum of three corners) defines a wall and one map can contain any amount of walls.

```
[…]
<Wall Height="1.00">
  <Corner X="16.50" Y="10.00" />
  <Corner X="16.50" Y="5.50" />
  <Corner X="21.00" Y="5.50" />
  <Corner X="21.00" Y="6.50" />
  <Corner X="17.50" Y="6.50" />
  <Corner X="17.50" Y="10.00" />
</Wall>
[…]
```

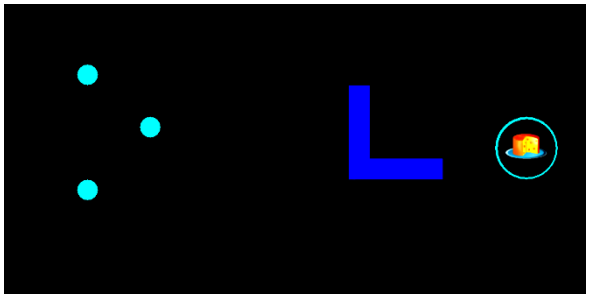


Figure 4.6 – Ciber-Rato Viewer's design of a XML modeled wall [28]

Since there was no existing map that resembled a road network, one was made from scratch. It is coinciding with the map loaded in the MAS-Ter Labs traffic simulator, since the agents in both simulators move in both of the maps at the same time. Care was taken with the sizes of the intersections, and the length and width of the roads, in order to create a map that is basically the walls that are limiting the roads. Be them buildings, or simply off-road terrain.

As for the textures, they were simple bitmaps already existing in the Intellwheels simulator. Their size was modified, and the resolution decreased, so that their pattern appears larger thus using less resources from the graphics card of the machine running the 3D Viewer. The main reason for this change was to make the textures look more natural when looked at from greater distances, since the dimensions of this map are much bigger than the typical Intellwheels map.

Finally, a model of a big car, already included in a modification of the MAS-Ter Labs traffic simulator [9], was imported into the Intellwheels simulator in order to replace the wheelchair. This kind of modeling is too complex to be made “by hand” through low level programming. It is saved in a stereolithography file type (STL) [69][70][71]. This kind of file stores the coordinates of the 3D object’s vertexes (X, Y and Z coordinates), as well as the orientations of the surfaces.

Figure 4.7 – Modified Intellwheels Viewer 3D, free view of the car model

4.6 – Summary

In short, from all the features needed to test various things about this concept, many were already done, a few of them were implemented, one didn't quite make it to the prototype, and there were also a few of them that didn't fall into the scope of the thesis.

While verifying the different functionalities of the simulators we worked with, the point was to modify or add to them if we required them for the tests. One example of the important functionalities that were already implemented before the thesis began was collision detection being modified for rectangular shapes, to replace the shapes of the circular Cyber-Rato robots. But of course, some of the programs didn't have the required features for the testing, and the most serious case was that of the 3D Viewer. Some of the implementations done to it, to make testing possible, were alterations to make it capable of dealing with the big distances and numbers involved in the testing of autonomous vehicles. Such modifications included the rendering routine for the camera for an increased draw distance, alterations to the drawing routines of the map, agents, etc. Other details were created and adapted, like the maps, while others were simply imported from other simulators of the same project, like the car model.

And then, there's the case of the functionalities that could be implemented next to improve the prototype and to approach it to the solution design, like implementing further connections between the different components of the architecture, as well as improvements to each individual software that is part of the solution design.

The prototype is ready for some tests, and those are described in the next chapter.

Chapter 5

Preliminary Results and Analysis

This chapter contains the tests and results done during the study. It also contains the methodology used for the tests, information about the testbed and environment and the expected results, and analysis of the results.

Two kinds of tests were done: A performance test, and a functionality test.

5.1 – Simulation Performance

The performance test basically consists of finding out how the Intellwheels Simulator, Viewer, and Agents perform as the amount of information they have to deal with increases. The main goal of this test was to find out how many Traffic cars ("Real" type Agents) the Simulator can deal with in a Simulation with one "Simulated" type of Agent acting as the autonomous vehicle being simulated. Note that Cyber-Rato was originally built for just 3 Agents, and later adapted to 5. It's not optimized to deal with many more Agents than that.

The methodology consists of the following: The Intellwheels Simulator is started, and the Intellwheels Viewer is connected to it. Then we start the simulation, and slowly connect more and more Agents to it, up to 100. The Intellwheels Simulator is run in a Desktop Computer, while the Intellwheels Viewers and Intellwheels Agents are run in a Laptop Computer, both with average characteristics considering the computer technologies to date. They are connected through the FEUP network. Response times are measured with a timer (starting the timer when connecting a new agent, and stopping the timer when the simulator's confirmation is received back) five times and the average time is noted down, while the computer load, memory usage and network usage are monitored using Windows Task Manager's performance monitor.

Desktop computer:

- Intel Pentium Core2 6400 @ 2,13GHz;
- 2 Gb of RAM,;

- Nvidia Geforce 7300 GS;
- Running Windows Vista Ultimate Service Pack 1 (32bits version).
- Network Interface: 100Mbps

Laptop computer:

- Intel Pentium Dual-Core T4200 @ 2,00GHz;
- 4 Gb of RAM,;
- Nvidia Geforce G 105 M;
- Running Windows 7 Ultimate (64bits version).
- Network Interface: 54Mbps (wireless)

The simulator calculates the sensor values for the 100 Agents and communicates those values to them (400 sensors). In practice, we wouldn't need any sensor in those traffic cars, because their decisions will be made by the MAS-Ter Labs Traffic Simulator, using other information. We only need the sensors in the simulated autonomous vehicles. So, the expected result is higher response latency after adding enough Agents to the simulation, much higher than it would be if the implementation was optimized for the solution design. The logic applies to the CPU loads and memory.

The results of the test (Test1) were the following: Through the whole test, up to 100 Agents connected, the frames per second of the Intellwheels 3D Viewer were fluid, above 20 frames per second in the worst case. That's good considering the 3D Viewer still has a long way to go when it comes to optimizing the code for better performance. Also the Simulator didn't use many computer resources: The CPU load was fixed at 50% throughout the whole test, and the memory used by it was at most, 13 MB. The only thing worth mentioning is that the network bandwidth used was, at most (when the 100 Agents were connected), less than 2% of the capacity in the desktop machine and a bit over 3% in the laptop machine. That corresponds to just over 200 KB/s considering the total bandwidth of the machines' network interfaces. Comfortable in a local network, but not so good if the simulation was distributed by different machines over the internet, considering today's technologies.

With just 1 "Simulated" Agent connected, the response times of the agent and sensors were instant. When we started adding "Real" type Agents to the simulation, the response times increased. By the time we had 100 agents added to the simulator, trading sensors information with it, the latency had reached an average of 400 ms. Considering the speed at which a vehicle can travel, more than 0.4 seconds is a lot of time for a reaction, and is unacceptable. For details on these results check Figure 5.1.

The test was repeated again (Test2), the only difference being that the Intellwheels Simulator, the Intellwheels Viewer, and the Intellwheels Agents are all run in the Laptop computer. The results: The CPU was close to full load roughly at 17 Agents connected, since the Intellwheels Simulator, Intellwheels Viewer, and the Intellwheels Agents were all fighting for CPU resources (See Figure 5.2 for the CPU Load on both tests). Interestingly enough, even at 100% CPU load, with the 100 Agents connected, the simulation had no glitches or crashes. And the Viewer frame rate was very fluid, over 20 frames per second. The latency measured was better than the previous test too: The range of the values was roughly the same, but the measured values were less chaotic on the second test, since the first test depended in external interference from the network traffic. Response times were almost instant with 10 Agents: less than 20ms in both cases. (More results in Figure 5.1 and 5.2) The simulation time-steps appeared to be suffering slowdowns by the time we had 100 agents connected.

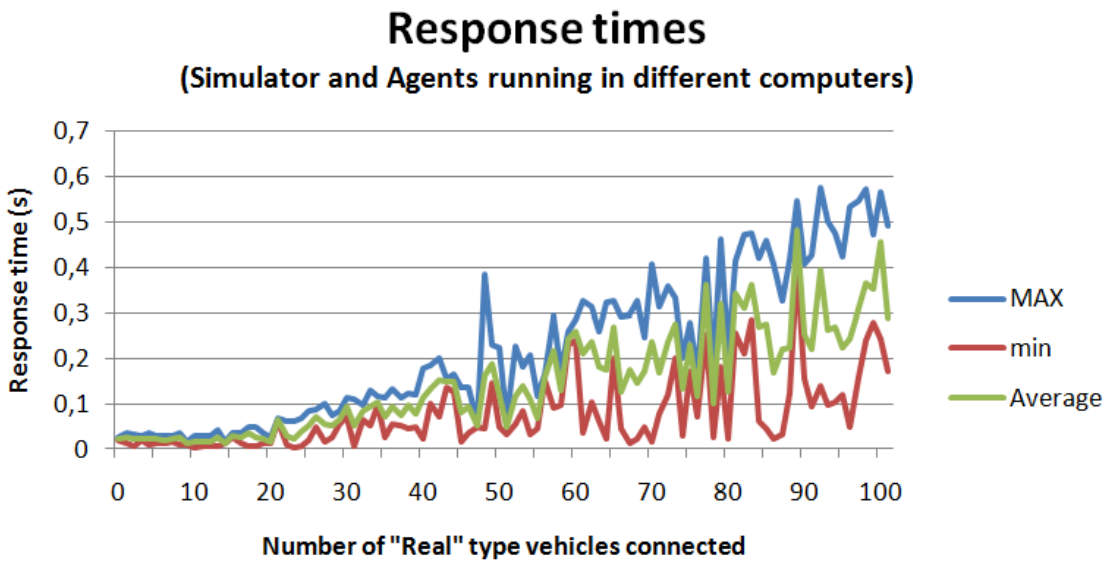


Figure 5.1 – Chart with the Response Times (ms) as a function of the Number of Agents (Test1)

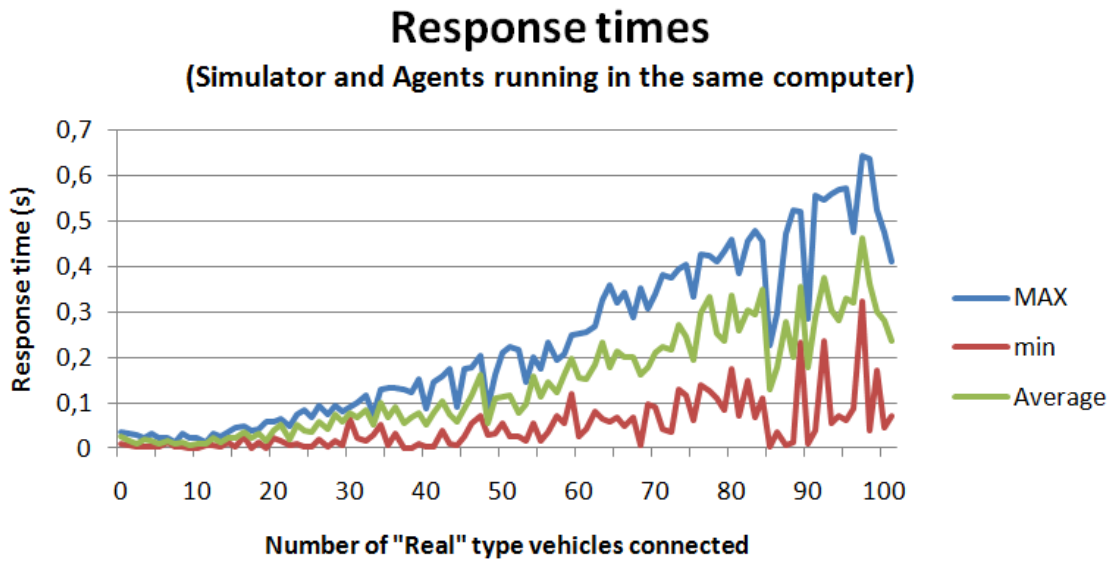


Figure 5.2 – Chart with the Response Times (ms) as a function of the Number of Agents (Test2)

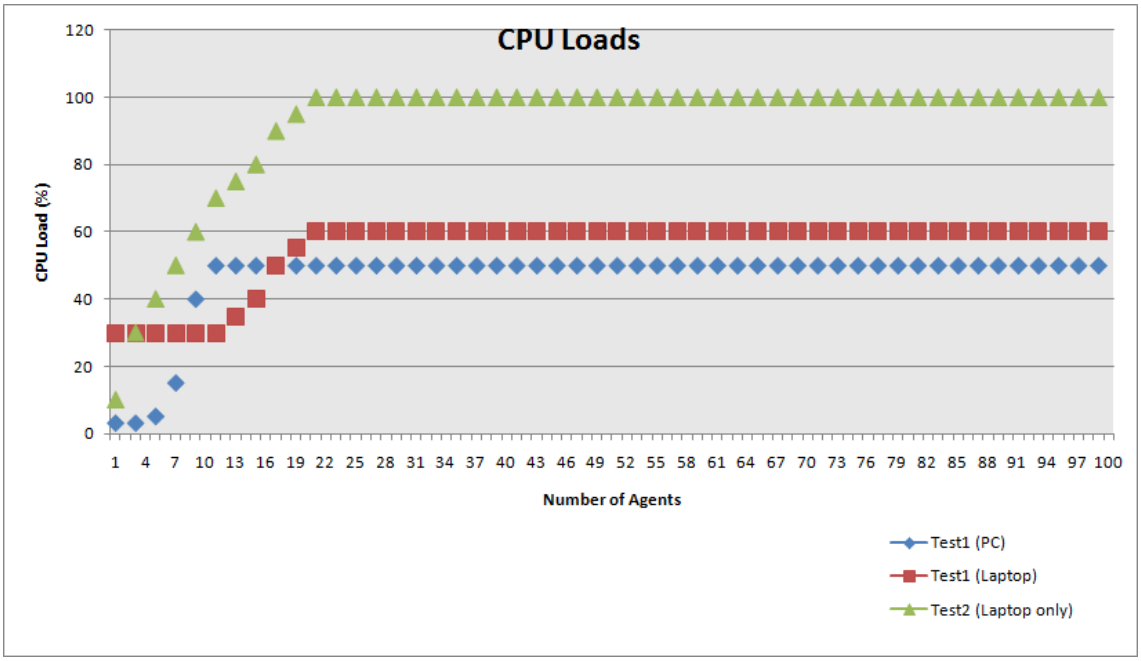


Figure 5.3 – Chart with the CPU Loads (%) as a function of the Number of Agents

Analyzing the results of both tests, it is apparent that although the Simulator is not optimized for large amounts of Agents, it can deal easily with many Agents. Having 100 Agents connected to it, each of them requesting sensor information the whole time, is like simulating the 100 detailed vehicles. The goal of the previously explained solution design is to test a few vehicles in detail, in the middle of traffic vehicles, which means that, ideally, those 100 Agents that were tested would involve no communications at all when it comes to

sensor information, that would improve the latency a lot when it comes to the messages between the Intellwheels Simulator, Intellwheels Viewer and the Intellwheels Agents.

A suggested implementation to optimize this simulator for our solution design is, registering no sensors at all when registering an Agent of the type “Real”, if the Agent is meant to represent a vehicle from the MAS-Ter Labs Traffic Simulator. That would cut down the communications bandwidth and latency a lot, and make it possible to simulate one autonomous vehicle in the middle of hundreds of traffic vehicles without performance issues.

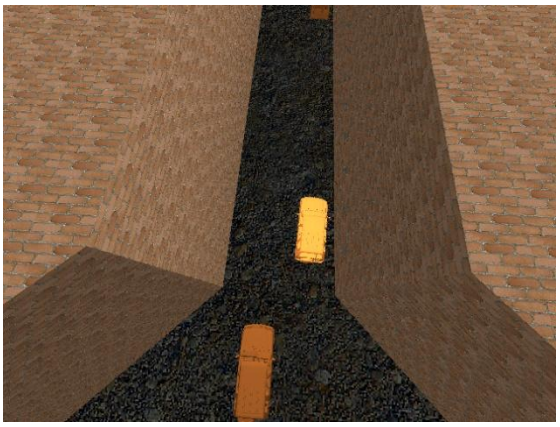
5.2 – Simulation Functionality

The functionality test basically consists of finding out if an Agent of the type “Simulated”, performing as an autonomous vehicle in the Intellwheels Simulator, can detect an Agent of the type “Real”, performing as a traffic vehicle (coming from the MAS-Ter Labs Traffic Simulator). The main goal of this test was to find out if the current implementation of the simulator, regarding sensors, can deal with simulating a car that reacts to the type of agents whose positions are updated by an external application. In reality, the module that would allow the MAS-Ter Labs Traffic Simulator to update the “Real” Agent’s position in the Intellwheels Simulator was not implemented, but the test is still valid when it comes to testing the sensors detecting cars whose positions are externally controlled. The only real downside to the module not being implemented is that the traffic car will be stopped.

In addition, we’ll have the Agent detect another “Simulated” type Agent, since our design solution predicts that more than one autonomous car can be present in the same simulation.

The methodology consists of the following: The Intellwheels Simulator is started, and the Intellwheels Viewer is connected to it. Then we start the simulation, and connect the three Agents needed for the test: two “Simulated” ones, and one “Real” one. The “Real” one is positioned in front of one of the “Simulated” ones, and the third agent, type “Simulated”, will be overtaking the two previously mentioned vehicles. And we’ll record the sensor value of the proximity sensor of the right side of the vehicle to see if it detects the vehicles being overtaken.

If all goes well, the Agent that will be overtaking the other two vehicles will detect first nothing to his right side, then the first vehicle being overtaken (one “Simulated” type Agent), and then nothing again, and then the second vehicle being overtaken (the “Real” type Agent). This will be translating in the value for the right-side sensor increasing as the vehicle is overtaking the others, since the higher the value in the proximity sensor, the closer the distance from the sensor to the obstacle is.



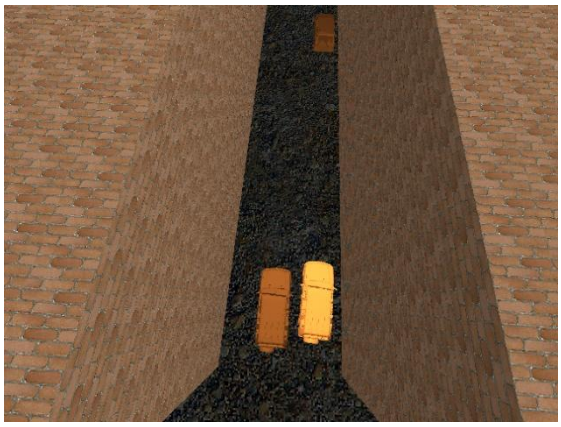
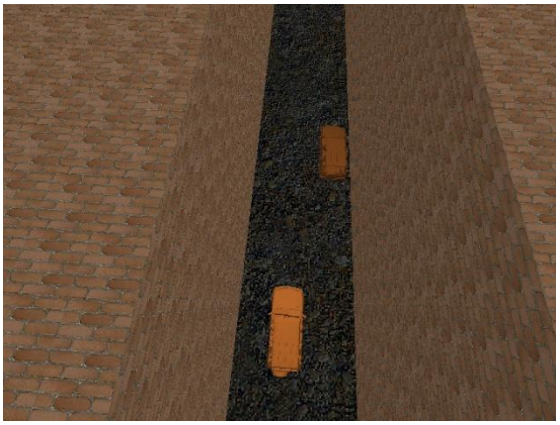


Figure 5.4 – Overtaking test (step 1): “Simulated” vehicle with no vehicle by its side

Figure 5.5 – Overtaking test (step 2): “Simulated” vehicle overtaking other “Simulated”



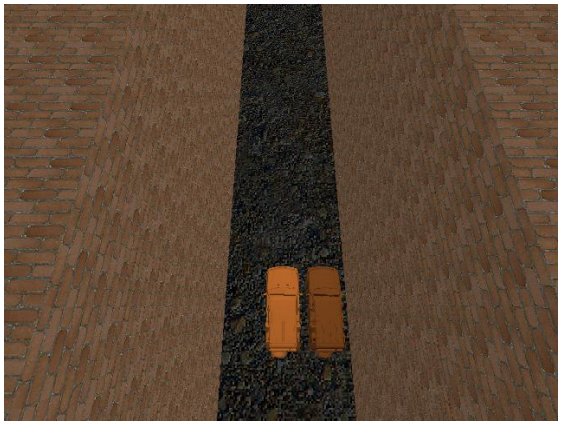


Figure 5.6 – Overtaking test (step 3): “Simulated” vehicle with no vehicle by its side

Figure 5.7 – Overtaking test (step 4): “Simulated” vehicle overtaking “Real”

The right-side sensor indicated the following values:

- Step 1 (Figure 5.1) – Right-side sensor value: 0.170409
- Step 2 (Figure 5.2) – Right-side sensor value: 1.29558
- Step 3 (Figure 5.3) – Right-side sensor value: 0.43369
- Step 4 (Figure 5.4) – Right-side sensor value: 1.52454

Analyzing the results, they do make perfect sense. The lowest number, during the step 1 (Figure 5.1), is due to the fact that the road is wider in that zone, and so there is a lot of room to the right of the vehicle. The value rises over one during step 2 (Figure 5.2), due to the close proximity to the car being overtaken. Note that the vehicle being overtaken here is

an Agent of the type “Simulated”. During step 3 (Figure 5.3), the vehicle is not overtaking anyone, and to his right side is only a building wall. But the street is still narrower than in step 1 (Figure 5.1). That explains why the sensor value decreased to 0.43369, instead of decreasing a low as it did in the first step. Finally, our autonomous vehicle overtakes another one. This time the vehicle that is an Agent of the type “Real” (the one performing the role of a traffic vehicle being controlled from an external simulator). The sensor value increases to 1.52454, meaning that it is detecting an object that is very close.

These are very good news. These results tell us that the functionality test was successful. An Agent of the type “Simulated”, performing as an autonomous vehicle in the Intellwheels Simulator, successfully detected an Agent of the type “Real”, performing as a traffic vehicle (whose position would be updated by the MAS-Ter Labs Traffic Simulator). In addition, we verified that the agent detected another agent of the type “Simulated”, meaning that an autonomous vehicle detects others like it, as was expected. All the goals of this test were achieved.

5.3 – Summary

After the performance tests were done, we can conclude that although the Intellwheels Simulator is not optimized for large amounts of Agents, it can deal easily with many of them before it starts getting slow. Results would easily improve with simple implementations like making the "Real" type Agents connect with no sensors at all, instead of the default 4 sensors each. With that improvement alone, it's very reasonable that the autonomous vehicle can be simulated along hundreds of traffic cars with little to no slow-downs, since the bottleneck of the simulations seemed to be the communications between the numerous agents and the simulator.

And with the functionality tests, we learned that vehicle sensors work fine detecting the traffic vehicles that would be controlled by the MAS-Ter Labs Traffic Simulator, as well as other autonomous vehicles.

We can conclude from both types of tests that further development of this solution design is feasible and worthwhile. The next step would be improving the prototype to approach the solution design, and run further tests to the different modifications to see if they're feasible.

Chapter 6

Conclusion

This chapter concludes the entire project with a few general remarks, followed by the most relevant test results and the achieved objectives. Further developments are also discussed, followed by potential future works, approaching possible paths of additional research and development from this point on, as well as suggesting how the simulation concepts presented in this study can be applied in totally different fields of study.

6.1 – General Remarks

Only recently did the concern arise about the effectiveness of the testing and simulating of autonomous vehicle projects. The simulators used are far from being ideal: The more features they tend to have, and the more realistic they are, the more resources they need to do their calculations in a reasonable amount of time. But simulation won't help much if it's oversimplified.

A balance between the realism of the simulation, and the simplicity of its calculations, is needed. When deciding the type of simulation that we need to run, we need to decide what aspects are important to simulate, and what aspects really don’t matter much: We don’t have an ideal simulator, but a nice thing to have is a connection between different types of simulators, that would each simulate different aspects and areas of a scenario to complement each other’s strengths. This is what this study aimed for.

Many simulators of different natures have been developed lately at LIACC, and we tried to create a solution design where we could put them all together connected to work cooperatively for a scenario that would otherwise be hard to simulate. So, we came up with this solution design.

The wide theoretical scope required for this study forced us to balance it out with a narrow focus for the modifications implemented, because it required a very comprehensive study as it is. After all, the point was to develop a prototype with some of the basic

functionalities, so that the concept could be tested. It is always best to do some testing as the functionalities are increased, before developing further.

While verifying the different functionalities of the simulators we worked with, the point was to modify or add to them if we required new functionalities for the tests. Some important ones were already implemented, while others had to be implemented during this thesis. The functionalities that could be implemented next to improve the prototype and to approach it to the solution design, were also discussed.

Having tested the performance and functionality of the prototype, it was concluded that it can deal with a reasonable number of agents without any optimizations, but that it still requires a few simple modification to make it able to deal with hundreds of traffic cars without slowdowns. We also concluded that vehicle sensors work fine detecting the traffic vehicles that would be controlled by the MAS-Ter Labs Traffic Simulator, as well as other autonomous vehicles.

We can conclude from both types of tests that further development of this solution design is feasible and worthwhile. The next step would be improving the prototype to approach the solution design, and run further tests to the different modifications to see if they're feasible.

6.2 – Main Results

A study was done about the state of art of the simulation of autonomous vehicles and semi-assisted driving. And having confirmed the strengths and weaknesses of the existing simulators, and having also confirmed what was needed to complement today's traffic and robotic simulators, the concept study for the integration of a traffic simulator with a detailed autonomous vehicle simulator was developed. After the programs and tools that would be used to build it were chosen, maps, 3d models and textures were created and adapted to aid with the testing of the prototype. The concept was tested by implementing a prototype with some basic functions, and performing tests to it with a user-controlled agent to demonstrate the vehicle's sensors' interactions with the vehicles whose positions would be updated by the MAS-Ter Labs traffic simulator. Those tests proved successful: The prototype showed us that with some further development, it can be used to simulate autonomous vehicles in intense traffic scenarios. Additionally, performance tests were made with the prototype to ensure that an average computer is able to run the simulation even thought the prototype is not optimized to perform well with the extra functionalities.

6.3 – Further Developments

The next step in implementing the prototype would no doubt be implementing some of the functionalities of the communication filtering module mentioned in section 3.2.1.1. Specifically, the one that would send information from the MAS-Ter Labs Traffic Simulator to the IntellWheels Simulator.

That step, allied with making those traffic cars connecting with zero sensors on them, would greatly increase the functionality of the prototype. With that done, we'd be able to run a very big and complex scenario with this solution design, and do further testing with it. The MAS-Ter Labs Traffic Simulator is very scalable, and that module would keep information about the Agents connected to the Intellwheels Simulator, filtering the information from the MAS-Ter Labs Traffic Simulator, only letting information pass to the Simulator if it’s about Agents that are somehow in the range of the Agents connected to the Simulator. So the simulations would be much bigger, but taking less computer and network resources. We'd still need the whole module implemented as well as having many changes made in the MAS-Ter Labs Traffic Simulator if we want to test traffic cars avoiding the Intellwheels Agents. But that's something for Future Works.

6.4 – Future Work

The natural evolution steps from here are pretty obvious. Incrementally implementing more and more functionalities from the solution design described in Chapter 3, and test them appropriately, until the prototype simulators have the same functionalities as the solution design. And from there, optimize it for better performance in every way.

This includes implementing the solution designs of the other simulators, since they'd be more functional than their respective prototypes. At some point, this would be necessary to achieve this study's solution design.

The kind of approach studied here has great potential in other fields. With simulators that simulate huge amounts of simple entities, and simulators that simulate few identities with complex details, coming together for a rich simulation of a big scenario by working cooperatively. This is a great way to solve problems and test theories where there are great amounts of information on different scales. This concept is not necessarily related to the simulation of autonomous vehicles.

An example of a completely different situation that could benefit tremendously from this concept is the simulation of complex astrophysical events. In such events, there's a big amount of bodies that can be approximated in groups when it comes to their influence (gravitational pulls or other forces). Such can be the case of solar systems, clusters of solar

systems, entire galaxies, and even clusters of galaxies. These are big and relatively easy to predict and model. Big bodies like these can influence small but detailed bodies or events, like the formation of moons, planets, and solar systems. Today, the planet-forming theories are always changing as new observations and simulations reveal new aspects of our reality. A set of simulators connected in a configuration like the one studied in this thesis would help a lot with solving such a problem, because while one simulator could take care of the many, but far bodies and send their influential information to another simulator, that other simulator could take that information to simulate in detail an unpredictable, and complex event. Using the outside information to calculate gravitational pulls and tidal forces, sources of radiation, end everything else that could possibly affect that complex event.

References

[1] L. C. Davis, "Effect of adaptive cruise control systems on traffic flow," *Physical Review*, vol. 69, no. 6, 2004.

[2] P. Ferreira, "Specification and Implementation of an Artificial Transport System," Thesis for Master in Informatics and Computing Engineering, Informatics Engineering, FEUP, Porto, 2008.

[3] R. J. F. Rossetti, E. C. Oliveira, and A. L. C. Bazzan, "Towards a specification of a framework for sustainable transportation analysis," in *Workshop on Artificial Intelligence Applied to Sustainable Transportation Systems, 13th Portuguese Conference on Artificial Intelligence*, Guimarães, 2007, pp. 3-4.

[4] P. A. F. Ferreira, E. F. Esteves, R. J. F. Rossetti, and E. C. Oliveira, "A Cooperative Simulation Framework for Traffic and Transportation Engineering," in *5th International Conference on Cooperative Design, Visualization and Engineering*, Mallorca, 2008, pp. 89-97.

[5] N. Lau, A. Pereira, A. Melo, A. Neves, and J. Figueiredo, "Ciber-Rato: Um Ambiente de Simulação de Robots Móveis e Autónomos," *Revista do DETUA*, vol. 3, no. 7, pp. 647-650, Sep. 2002.

[6] N. Lau, A. Pereira, A. Melo, J. Neves, and J. Figueiredo, "Ciber-Rato: Uma Competição Robótica num Ambiente Virtual," in *Workshop Entretenimento Digital e Jogos Interactivos, Games-2004*, vol. 3, Lisboa, Sep. 2004.

[7] Stanford Racing Team. (2007, Nov.) DARPA Challenge invites Stanford Racing team to build a robotic car. [Online]. http://cs.stanford.edu/group/roadrunner/

[8] Team Victor Tango, Virginia Tech. (2009, Feb.) Victor Tango Urban Challenge. [Online]. http://www.me.vt.edu/urbanchallenge/

[9] J. F. B. Gonçalves, "Arquitectura Baseada em Serviços para Redes Veículo-a-Veículo," Thesis for Master of Electrotechnical and Computer Engineering, FEUP, Porto, 2009.

[10] E. B. Lieberman and B. Andrews, "The Role of Interactive Graphics When Applying Traffic Simulation Models," in *Proceedings of the 22nd conference on Winter simulation*, New Orleans, 1990, pp. 753-758.

[11] S. L. Jones, A. J. Sullivan, N. Cheekoti, M. D. Anderson, and D. Malave, "Traffic Simulation Software Comparison Study," Civil & Environmental Engineering, University of Alabama, Birmingham, Technical Report, 2004.

[12] L. Bloomberg and J. Dale, "Comparison of VISSIM and CORSIM traffic simulation models on a congested network," in *The 79th annual meeting of the Transportation Research Board*, Washington, 2000, pp. 52-60.

[13] F. Choa, R. T. Milam, and D. Stanek, "CORSIM, PARAMICS and VISSIM: What the manuals never told you," in *ITE Conference*, Philadelphia, 2002.

[14] L. Bloomberg, M. Swenson, and B. Haldors, "Comparison of Simulation Models and the HCM," in *Transportation Research Board, 82nd Annual Meeting*, Washington, 2003.

[15] E. Barrios, M. Ridgway, and F. Choa, "The Best Simulation Tool For Bus Operations.

Improving Transportation System Safety and Performance," in *ITE Spring Conference and Exhibit*, 2001.

[16] M. Trueblood, "CORSIM...SimTraffic: What's the Difference?," *PC-TRANS*, 2001.

[17] Z. Z. Tian and N. Wu, "Probabilistic Model for Signalized Intersection Capacity with a Short Right-Turn Lane," *Journal of Transportation Engineering*, vol. 132, no. 3, pp. 205-212, Mar. 2006.

[18] S. A. Boxill and L. Yu, "An Evaluation of Traffic Simulation Models for Supporting ITS Developments," Southwest Region University Transportation Center, Report, 2000.

[19] M. D. Middleton and S. A. Cooner, "Evaluation of Simulation Models for Congested Dallas Freeways," Texas Transportation Institute, Austin, Technical Report, 1999.

[20] D. P. Watling, "Urban traffic network models and dynamic driver information systems," *Transport Reviews*, vol. 14, no. 3, pp. 219-246, Jul. 1994.

[21] G. E. Cantarella and E. Cascetta, "Dynamic process and equilibrium in transportation networks: towards a unifying theory," *Transportation Science*, vol. 29, no. 4, pp. 305-309, 1995.

[22] R. Liu, D. Van Vilet, and D. P. Watling, "DRACULA: dynamic route assignment combining user," in *Proceedings of the 23rd European Transport Forum*, vol. E, Warwick, 1995, pp. 143-152.

[23] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI architecture," in *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, 1991, pp. 473-484.

[24] M. Treiber, A. Hennecke, and D. Helbing, "Congested Traffic States in Empirical Observations and Microscopic Simulations," *Physical Review E*, vol. 62, no. 2, pp. 1805-1824, Aug. 2000.

[25] M. Treiber and D. Helbing. (2007, Feb.) The Lane-change Model MOBIL. [Online]. http://www.vwi.tu-dresden.de/~treiber/MicroApplet/MOBIL.html

[26] T. Alberi, "A Proposed Standardized Testing Procedure for Autonomous Ground Vehicles," Thesis for Master of Science, Mechanical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2008.

[27] A. L. Kornhauser, et al. (2007, Jun.) DARPA Urban Challenge Princeton University Technical Paper. [Online]. http://www.darpa.mil/grandchallenge/TechPapers/Princeton_University.pdf

[28] P. Malheiro, "Intelligent Wheelchair Simulation," Thesis for Master of Electrotechnical and Computer Engineering, Electrical Engineering and Computers, FEUP, Porto, 2008.

[29] Departamento de Electrónica, Telecomunicações e Informática. (2008, Mar.) CiberMouse at DCOSS08. [Online]. http://www.ieeta.pt/lse/ciberdcoss08/docs/ciberDCOSS08_Rules.pdf

[30] C. Reinholtz, et al. (2007, Apr.) DARPA Urban Challenge Technical Paper. [Online]. http://www.darpa.mil/GRANDCHALLENGE/TechPapers/Victor_Tango.pdf

[31] B. Stroustrup, *The C++ programming language*, 2nd ed.. USA: Addison-Wesley Longman Publishing Co., Inc, 1991.

[32] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *IEEE International Conference onRobotics and Automation (ICRA)*, 2007, pp. 1400-1405.

[33] B. Browning and E. Tryzelaar, "ÜberSim: A Multi-Robot Simulator for Robot Soccer," Carnegie Mellon University, 2003.

[34] T. Bräunl, "The EyeSim Mobile Robot Simulator," University of Auckland, CITR at Tamaki Campus, 2000.

[35] C. Basarke, C. Berger, and B. Rumpe, "Software & Systems Engineering Process and Tools for the Development of Autonomous Driving Intelligence," *Journal of Aerospace Computing, Information, and Communication*, vol. 4, no. 12, pp. 1158-1174, Oct. 2007.

[36] Carnegie Mellon Tartan Racing. (2009, Feb.) Boss at a Glance. [Online]. http://www.tartanracing.org/press/boss-glance.pdf

[37] Stanford Racing Team. (2007, Oct.) All about Junior. [Online].

http://cs.stanford.edu/group/roadrunner/pdfs/final_factsheet_junior.pdf

[38] Carnegie Mellon Tartan Racing. (2009, Feb.) Tartan Racing @ Carnegie Mellon. [Online]. http://www.tartanracing.org/

[39] Ben Franklin Racing Team. (2009, Feb.) Building Fast Autonomous and Safe Robotic Vehicles for Urban Environments. [Online]. http://www.benfranklinracingteam.org/

[40] Cornell University DARPA Urban Challenge Team. (2009, Feb.) Cornell DARPA Urban Challenge. [Online]. http://www.cornellracing.com/

[41] MIT DARPA Grand Challenge Team. (2008, Aug.) MIT DARPA Urban Challenge. [Online]. http://grandchallenge.mit.edu/

[42] I. Miller and M. Campbell, "Team Cornell's Skynet: Robust Perception and Planning in an Urban Environment," *Field Robotics*, vol. 25, no. 8, p. 493–527, Accessed in ??/???/????, at http://www3.interscience.wiley.com/cgi-bin/fulltext/120846938/PDFSTART.

[43] Microsoft. (2009, Feb.) Microsoft Robotics. [Online]. http://msdn.microsoft.com/en-us/robotics/default.aspx

[44] R. Azuna, et al., "Recent Advances in Augmented Reality," *IEEE Computer Graphics and Applications*, vol. 21, no. 6, pp. 34-37, Nov. 2001.

[45] P. Milgram and F. Kishino, "A Taxonomy of Mixed Reality Visual Displays," in *IEICE Transactions on Information Systems*, vol. E77-D, 1994, pp. 1321-1329.

[46] Universidade de Aveiro. (2009, Feb.) Concurso Micro-Rato. [Online]. http://microrato.ua.pt

[47] L. P. Reis, "Ciber-Feup - Ensino de Robótica e Inteligência Artificial através da Participação em Competições Robóticas," *Electrónica e Comunicações*, vol. 7, no. 3, Sep. 2002.

[48] L. Almeida, P. Fonseca, L. J. Azevedo, and B. Cunha, "The Micro-Rato Contest: Mobile Robotics for All," in *CONTROLO 2000, The Portguese Control Conference*, Guimarães, Portugal, 2000.

[49] RTSS - Real-Time Systems Symposium. (2008, Jun.) RTSS - Real-Time Systems Symposium. [Online]. http://www.rtss.org/

[50] Universidade de Aveiro. (2007, Nov.) CiberMouse at DCOSS08. [Online]. http://www.ieeta.pt/lse/ciberdcoss08/

[51] D. Barteneva, N. Lau, and L. P. Reis, "Implementation of Emotional Behaviors in Multi-Agent System using Fuzzy Logic and Temperamental Decision Mechanism," in *Proceedings of EUMAS 2006*, Lisbon, Portugal, 2006, pp. 5-15.

[52] D. Barteneva, N. Lau, and L. P. Reis, "Bylayer Agent-Based Model of Social Behavior: How Temperament Influences on Team Performance," in *21st European Conference on Modelling and Simulation – ECMS 2007*, Prague, Czech Republic, 2007, pp. 181-187.

[53] L. Lemos, F. Cruz, and L. P. Reis, "Sistema de Resgate e Salvamento Coordenado Utilizando o Simulador Ciber-Rato," in *CISTI 2007 - 2ª Conferência Ibérica de Sistemas e Tecnologias de Informação, Novas Perspectivas em Sistemas e Tecnologias de Informação*, Porto, Portugal, 2007.

[54] R. A. M. Braga, M. Petry, E. Oliveira, and L. P. Reis, "Multi-Level Control Of An Intelligent Wheelchair In a Hospital Environment Using A Cyber-Mouse Simulation System," in *5th International Conference on Informatics in Control, Automation and Robotics*, Funchal, Madeira, Portugal, 2008, pp. 179-182.

[55] R. A. M. Braga, M. R. Petry, A. P. Moreira, and L. P. Reis, "Platform for intelligent wheelchairs using multi-level control and probabilistic motion model," in *8th Portuguese Conference on Automatic Control, Controlo 2008*, Vila Real, Portugal, 2008.

[56] R. A. M. Braga, M. Petry, A. P. Moreira, and L. P. Reis, "INTELLWHEELS - A Development Platform for Intelligent Wheelchairs for Disabled People," in *5th International Conference on Informatics in Control, Automation and Robotics*, vol. I, Funchal, Madeira, Portugal, 2008, pp. 115-121.

[57] M. K. Dalheimer, *Programming with Qt*, 2nd ed.. O'Reilly, 2002.

[58] Qt Software. (2009, Mar.) Qt - A cross-platform application and UI framework. [Online]. http://www.qtsoftware.com/

[59] Borland Software Company. Borland Software Company. [Online]. http://www.borland.com

[60] M. Cantù, *Mastering Delphi 7*, 1st ed.. Sybex, 2003.

[61] OpenGL. (2008, May) The Industry's Foundation for High Performance Graphics. [Online]. http://www.opengl.org/

[62] M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide, Third Edition: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley, 1999.

[63] E. F. Esteves, "Utilização de agentes autónomos na simulação pedonal em interfaces multi-modais," Thesis for Master in Informatics and Computing Engineering, Informatics Engineering, FEUP, Porto, 2009.

[64] P. Ferreira, E. Esteves, R. Rossetti, and E. Oliveira, "Extending microscopic traffic modelling with the concept of situated agents," in *Proceedings of the 5th Workshop in Agent in Traffic and Transportation, AAMAS'08*, Estoril, Cascais, 2008, pp. 87-93.

[65] D. P. Reed. (1980, Aug.) Internet Engeneering Task Force. [Online]. http://tools.ietf.org/html/rfc768

[66] J. C. Lopes and C. Ribeiro. (2008, Feb.) João Correia Lopes | Homepage. [Online]. http://paginas.fe.up.pt/~jlopes/teach/2007-08/LAPD/lectures/01-XML-intro.pdf

[67] M. R. Rohrer, "Seeing is Believing: The Importance Of Visualization in Manufacturing Simulation," in *Winter Simulation Conference*, 2000, pp. 1211-1216.

[68] A. Neves, J. Figueiredo, N. Lau, A. Pereira, and A. Melo, "O Visualizador do Ambiente de Simulação Ciber-Rato," *Revista do DETUA*, vol. 3, no. 7, pp. 651-654, Sep. 2002.

[69] E. Béchet, J. C. Cuilliere, and F. Trochu, "Generation of a finite element MESH from stereolithography (STL) files," *Computer-Aided Design*, vol. 34, no. 1, pp. 1-17, Jan. 2002.

[70] M. Burns, "The StL Format," in *Automated Fabrication*. Prentice Hall, 1989, ch. Section 6.5.

[71] M. Burns, "The STL File Format," in *Automated Fabrication - Improving Productivity in Manufacturing*. Prentice Hall, 1993, ch. Section 6.5.