FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

An Integrated Architecture for Autonomous Vehicles Simulation

José Luis Ferrás Pereira

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Supervisor: Rosaldo J. F. Rossetti (PhD)

June 2011

Abstract

Research on autonomous vehicles has come a long way since first findings, and its software tools are increasingly acclaimed by the research community. Particularly with robotics simulators, autonomous vehicles were provided with a suitable test-bed for experimentation of new methodologies such as long-term navigation algorithms, map building and intelligent reasoning. However, when it concerns the deployment and validation of such vehicles in a larger urban traffic scenario, robotics simulators do not seem to provide the required functionality for road traffic analysis, or inter-vehicular communication infrastructure as they seem present in today's traffic simulators. The improvement of such features is the key for the successful practical deployment of such a critical system.

The main objective of this dissertation is the integration of two types of simulators, namely a robotics and a traffic simulator. This integration will enable autonomous vehicles to be deployed in a rather realistic traffic flow as an agent entity (on the traffic simulator), at the same time it simulates all its sensors and actuators (on the robotics counterpart). Also, the statistical tools available in the traffic simulator will allow practitioners to infer what kind of advantages such a novel technology will bring to our everyday's lives. Furthermore, the current features and issues on current robotics and traffic simulators are presented and a taxonomy for selecting these simulators is proposed. An architecture for the integration of the aforementioned simulators is proposed and implemented in the light of the most desired features of such software environments.

To assess the usefulness of the platform architecture towards the expected realistic simulation facility, a comprehensive system evaluation is also performed and critically reviewed, leveraging the feasibility of the integration. Further developments and future perspectives are pinpointed up in the end.

Resumo

A investigação e o desenvolvimento na área de veículos autónomos têm vindo a dar cartas desde as primeiras descobertas, e as suas ferramentas de software são cada vez mais aclamadas pela comunidade científica. Fazendo uso de simuladores robóticos, os veículos autónomos foram dotados de ferramentas adequadas para a experimentação de novas metodologias, tal como algoritmos de navegação, contrução de mapas ou inteligência artificial. No entanto, no que concerne ao desenvolvimento e à validação destes veículos em cenários de redes de tráfego em maior escala, estes simuladores não fornecem as ferramentas de análise ou simulação de redes que são comummente utilizadas nos simuladores de tráfego. A introdução destas ferramentas na área de veículos autónomos representa a verdadeira chave para o desenvolvimento de um sistema desta envergadura.

O principal objectivo desta dissertação é a integração de dois tipos de simuladores: um da área dos transportes e outro da área da robótica. Esta integração permitirá o desenvolvimento de novas metodologias para o controlo de veículos autónomos, que se vão inserir na rede de tráfego simulada de uma forma transparente ao mesmo tempo que simula todos os seus sensores e actuadores no simulador robótico. Sendo assim, as ferramentas de estatística disponíveis nos simuladores de tráfego permitirão que os utilizadores da plataforma infiram dados sobre as vantagens ou consequências que esta nova abordagem poderá trazer para as nossas sociedades.

Inicialmente, o estado da arte é apresentado tendo em conta os tópicos base do projecto, e é explicitada uma taxonomia para avaliar a efectividade que um simulator de tráfego ou robótico tem nesta integração. Seguidamente, é proposta e implementada uma arquitectura para a integração dos simuladores supracitados tendo em conta os requisitos mais importantes.

Para se avaliar a utilidade da plataforma proposta, alguns testes de desempenho e validação são efectuados e criticamente discutidos, tendo-se em conta os requisitos inicialmente regulados. Trabalhos e perspectivas futuras são apontadas no final do documento.

Acknowledgments

I gratefully thank all my friends, family and colleagues who have contributed directly or indirectly to such a wonderful achievement which is to conclude a Master's Dissertation, and to make it become a reality.

I would like to thank LIACC/FEUP for the opportunity I had to do research in the laboratory, which enriched my personal character and scientific abilities as a young researcher; I particularly thank Dr. Rosaldo Rossetti, who had believed in my potential from the first day, for his professionalism and friendship.

Many thanks to José Xavier, for his assistance and companionship all over the period of preparation of this work.

I could not forget all my friends out there, namely Emanuel, Pedro, Hugo, Nuno, Marco, the LIACC members, and so forth, for your friendship and availability to advise me whenever I needed.

To my mother, father, brothers and family, who have accompanied me in the good and the bad moments, supporting me with the best they could give, my tender and eternal gratitude.

To Duarte and Ricardo, for being the best roommates and for their support and enthusiasm over these years, a great thank you.

Last but not least, the most special thanks go to my beloved girlfriend Neuza, who have encouraged me over the last six years with her gracefulness. Thank you for all your concerns and advices that have helped me to end up with this Dissertation in your so special way.

I hereby dedicate this dissertation to all of you!

José Pereira

“Great art presupposes the alert mind of the educated listener”

Arnold Schoenberg

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Symbols and Abbreviations

Hz	Hertz
ABS	Anti-lock Breaking Systems
AGV	Automated Guided Vehicle
AIBO	Artificial Intelligence Robot
ALV	Autonomous Land Vehicle
ANN	Artificial Neural Networks
ASIMO	Advanced Step in Innovative Mobility
AUV	Autonomous Underwater Vehicles
CPU	Central Processing Unit
DARPA	Defence Advanced Research Projects Agency
DUA	Dynamic User Assignment
DLL	Dynamic-link Library
ELROB	European Land Robot Trial
ESC	Electronical Stability Control
FEUP	Faculdade de Engenharia da Universidade do Porto
FUT	Future Urban Transport
GCDC	Grand Cooperative Driving Challenge
GIS	Geographical Information Systems
GPS	Global Positioning Systems
GUI	Graphical User Interface
ITS	Intelligence Transportation Systemss
IVC	Inter-vehicular Communications
LIACC	Laboratório de Inteligência Artificial e de Ciência de Computadores
MAS-T2er Lab	Laboratory for MAS-based Traffic and Transportation Engineering
MOAST	Mobility Open Architecture Simulation and Tools
MRDS	Microsoft Robotics Developer Studio
OSM	OpenStreet Maps
R&D	Research and Development
RCS	Real-time Control Systems
SUMO	Simulation of Urban MObility
TraCi	Traffic Control Interface
UAV	Unmanned Aerial Vehicles
UDK	Unreal Development Kit
UE	Unreal Engine
UDP	User Datagram Protocol
USARSim	Urban Search and Rescue Simulation
UTA	Urban Traffic Assistant
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle

Chapter 1

Introduction

This chapter consists of a motivation section, where a brief overview and a contextualization of the problems facing current traffic systems is discussed. An objectives section, containing a description of the goals that guide the project associated with this document, a planning section, containing all major milestones in the project's scope, and a document structure section consisting of an exposition of how this document is arranged.

1.1 Problem Statement

Urban traffic represents one of the most problematic products of our contemporary society. Traffic accidents caused by drivers' mistakes are commonly associated with alcohol and drugs consumption, stress and, more recently, distraction due to cell phone use. Traffic congestions are found very often on road networks, causing high levels of pollution as well as increased delays and money spent in a journey. Ultimately, society's quality of life is substantially degraded with these effects. A typical traffic congestion in a highly populated city is depicted in Figure 1.1.

With the introduction of Intelligent Transportation Systems (ITS) and accounting for Future Urban Transport Systems (FUT) requirements, traffic flow in overpopulated areas was vastly improved along with overall road safety and security. This effort has been gaining a wider focus with research of autonomous urban vehicles, as recent advances in technology processes supply the needed software and hardware requirements for such critical and complex systems.

There are already present in the literature various approaches to this concept since the 70s, albeit only recently the industry market started investing on the appropriate technology. Also, some autonomous vehicles competitions were held in the past millennium, such as the DARPA (Defense Advanced Research Projects Agency) Grand Challenge, to encourage researchers onto developing driverless cars [2].

Some authors [3, 4, 5] present the autonomous vehicle paradigm as a software problem. Not surprisingly, the most difficult challenges about autonomous systems are on the algorithmic side. Particularly on autonomous vehicles research, the highly non-linear behavior of traffic networks' heterogeneous interacting entities is a rather complex task to handle.

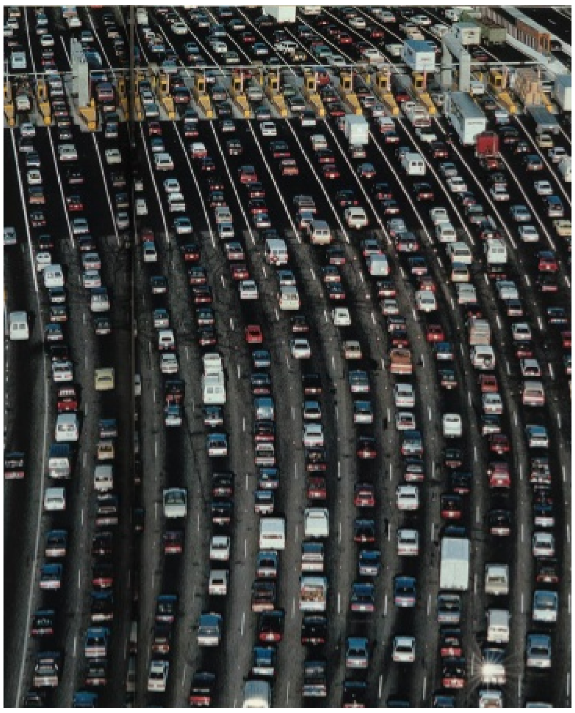


Figure 1.1: Urban traffic is becoming more and more saturated. New ITS solutions are emerging to optimize traffic without changing the network topology[1].

Robotics simulators have been the standard test-bed to experiment with new vehicles methodologies of autonomous navigation, obstacle avoidance and human-machine interaction. However, when developing some novel approaches such as cooperative driving behavior in realistic multimodal traffic scenarios, these simulators are still very unfeasible.

Facing this emerging interest regarding the application of robotics on regular-sized vehicles, both traffic and robotics simulators must be integrated with each other to allow engineers and researchers to evaluate and deploy new methodologies towards FUT, accounting for the futuristic perspective of driverless cars populating our cities as well.

1.2 Aim and Goals

This thesis aims to bridge the gap between traffic network analysis and testing autonomous vehicles in urban scenarios. This must be accomplished using a distributed environment. The ultimate goal of the project is the successful simulation of a population of vehicles together, as autonomous agent entities, following a seamless interaction. The selected approach envisions the integration of two simulators, one from the robotics field and another from the transportation field. Moreover, a communication layer between them will be built on top of a compatible metadata.

The application should be distributed, as one computer will be simulating a large traffic environment whereas other computers simulate single autonomous vehicles to be integrated in the

[1]Courtesy from `http://streetwise.kittelson.com/`

former simulation. The latter will not only simulate its robotic sensor information, but also the surroundings of the autonomous vehicle, which would be handled by a control agent managing its navigation.

Ultimately, the software bundle will have a simple and clear configuration, maintaining the integrity of the simulators, so their documentation is still valid.

1.3 The Methodological Approach

To accomplish all aforementioned goals, several robotic and traffic platforms are initially reviewed. The problem statement is going to be addressed by analyzing both selected simulators, fixing each issue that inhibit their integration. Furthermore, a high-level architecture of the system is modeled. On the implementation phase, good software practices should be applied, such as unit testing, code style definition or design patterns, along with a comprehensive documentation.

A sample agent will be devised to validate and produce consistent results of the platform, which controls a autonomous vehicle following a predefined route using sensor data. Finally the encountered issues towards the initial goals are critically analyzed.

As a form of synthesis, the tasks expected to be carried out throughout this dissertation are the following:

- Review and assess most of the well-known traffic and robotics simulators in the light of the project's requirements;
- Devise a flexible integration architecture to couple both simulators;
- Pinpoint each simulator issue which does not comprehend to the project goals and suggest a fix;
- Implement the possible suggested fixes and network layer for the proposed integration;
- Establish coherent performance evaluation metrics to measure the platform efficiency.

1.4 Document Outline

This report is structured as follows:

- In this chapter, an introduction to the subject of work and most important goals to achieve are presented. Also, a methodological approach is established;
- In Chapter 2, a literature review introducing the main concepts and current state of the art is presented;
- In Chapter 3, the design solution is proposed along with its fundamental issues to be addressed;

- Chapter 4 introduces the SUMO microscopic traffic simulator and depicts its implemented fixes to allow proper application of the design solution;
- USARSim robotics simulator is presented in Chapter 5 and its implemented fixes are also described;
- Chapter 6 proposes a solution for the encountered problems on the aforementioned simulators integration.
- In Chapter 7, some functionality and performance tests to the integration implementation are performed, and its results are discussed.
- Chapter 8 concludes the document depicting the main contributions, final remarks and future work.

Chapter 2

Literature Review

In order to frame the thesis' subject of study, this section provides a brief historical description, explains the main concepts, and describes and contextualizes all related topics. The first approached topic is Game and Simulation Engines and their application in the research arena. Afterwards, a detailed history of Autonomous Vehicles and its generic high level architecture is presented. This section is followed by a brief clarification of the Robotics field, ending with an overview on Traffic Simulation.

2.1 Game and Simulation Engines

A game engine is a software framework suitable for video games development. When first commercial video consoles appeared, games were developed from scratch since their relative code extension and algorithm complexity was low [6] . However, with the increasing competition in the game industry and constant evolution of technology, game makers needed more time to develop a product making use of such a high computing power.

To mitigate such issues, several software frameworks have been introduced as game engines. These software packages have in account video games' requirements thus implementing efficient and high-level modules, for instance rendering, physics, audio, logic, networking, Artificial Intelligence and input abstractions. This methodology would allow game developers to be more focused in the game plot itself also benefiting from state-of-the-art graphics and multimedia to compete with the more than ever demanding industry. On the other hand, a single game engine can be used within several game titles, making it a rentable business. In fact, there are companies nowadays devoted to game engine development only, as a commercial license can float from ten thousand dollars to one million dollars. Some of the most known game engines in the history are Doom [7] and Unreal Engine [8]. Figure 2.1 illustrates a state-of-the-art driving simulator game.

2.1.1 Game Engines in Research

Nowadays, there is a major interest from the research community in using game engines as simulator frameworks [9]. Similarly to game development, in that regarding simulation of realistic

Figure 2.1: General aspect of a driving simulator video game from 2010, Grand Turismo 5. Its graphical aspect resembles an highly realistic environment[1].

world-sized environments, the use of such technology would produce cost-effective immersive serious games, or in another way, simulation deployment, such as a robotics simulator, or in education purposes, a driving simulator [10]. As one of the subjects of this project is supported on the robotics field, a brief analysis on the adequacy of this technology in robotics simulation is performed below.

Some of the key features game engines provide, which are favorable to robotics researchers, are listed below:

- Distributed architecture:
  - Support for multiple processor cores is included in earlier frameworks for maximum computational resources exploitation;
  - It is possible to simulate multiple entities in multiple networked computers, distributing the processing power over all nodes.
- Cutting-edge graphics:
  - Albeit the use of game engines will increase significantly the level of detail and realism of the environment, relating to camera sensor simulation, higher resemblance from virtual to real world can be achieved.
- Realistic Simulation:
  - Suitable for virtual-reality environments, such as a driving simulator;
  - Most recent game engines feature both rigid and soft body dynamics, some of them even use a new dedicated hardware named Physics Processing Unit (PPU). Also cutting-edge lightning effects, polygon rendering and realistic destructible environments are present/considered.

[1]Courtesy from http://games.gearlive.com/playfeed/article/gran-turismo-5-realism-q111

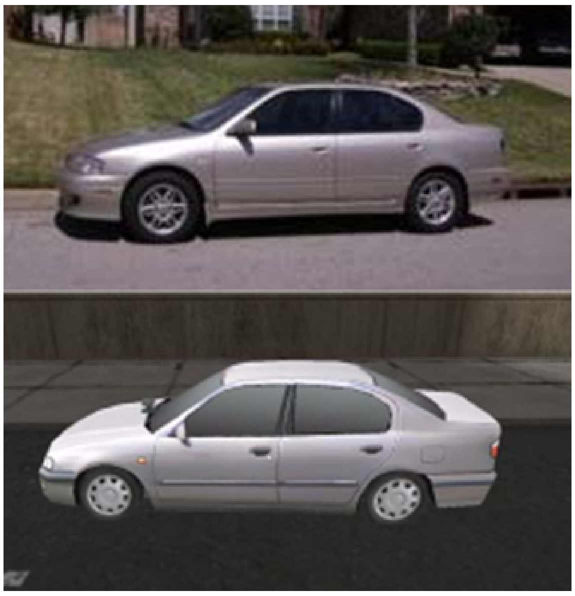


Figure 2.2: Real Sedan above, USARSim simulated sedan below.

- Scriptable environment:
  - Featuring simple but powerful scripting languages, game engines can be rapidly extended to support a new type of sensor, or an optimized statistics module.

As a matter of fact, there are already some robotics simulators following this approach, such as the Urban Search and Rescue Simulation (USARSim, for short) [11] which uses the Unreal Engine (UE), and Player/Stage/Gazebo [12], using the open-source rendering engine Ogre3D. Figure 2.2 illustrates the visual resemblance of a real vehicle and a simulated vehicle in the USARSim robotics simulator.

2.1.2 Current Issues

When researching robotics simulation as game engines, only behaviour-based dynamics should be tested. Hence, when high-fidelity simulation is required, dedicated software containing mathematical representation of the subsystems should be used in order to achieve realistic calculations. These software is often validated with *hardware-in-the-loop* techniques, largely used in the evaluation of computer-based testing equipments. High-level algorithms for trajectory planning, vision-based processing, and multi-agent systems interactions are examples of suitable fields to use with robotics simulators based on game engines [10].

2.2 Autonomous Urban Vehicles

From the Ancient Greek, autonomous means auto-"self" and nomos-"law". The word autonomy represents the one who gives oneself their own law, that means, an entity capable of making an informed, un-coerced decision. On the other hand, a vehicle (from the Latin *vehiculum*) is a device qualified to transport people or cargo, over land or in space.

One should not confuse being autonomous with being intelligent. However, when it comes to vehicles, regarding the complex interactions of such systems and their environment, an autonomous vehicle is commonly considered an intelligent vehicle.

Looking from a today's perspective, there are already some autonomy features present in production vehicles, such as ABS or ESC (Anti-lock breaking system & Electronic Stability Control), and there are some other in development as Automated Parking or Vehicle Platooning. However, despite these methods are considered to be driver-assistance components as well, the ultimate effort is the deployment of such a vehicle as actually being capable to perform a full point-to-point journey without human intervention, maintaining rather complex aspects in consideration such as security, efficiency and scalability, and respecting the time and space constraints human travelers are used to. Arguably, autonomous vehicles research is a sub-field of robotics, thus a brief history and contextualization of it is presented in Section 2.3. The former description of an autonomous vehicle is not in the scope of this thesis, furthermore, the content of the following sections are referred only to the latter definition.

2.2.1 Brief History

The vision for cars that drive themselves while their drivers relax is not new, as it was presented in 1939 at the World Fair in New York, in the General Motors Pavilion [13]. However, further Research and Development (R&D) began from early 70's, when the space race competition between Soviet Union and United States towards the supremacy of outer space exploration was in run. There were already plans for space missions on Mars, and giving the distance from it to Earth, it was simply not feasible to control a robot by an earth-based operator viewing a live TV picture. Moreover, the vehicle needed to be controlled by an on-board computer [14]. Arguably, one of the most obvious reasons this unmanned autonomous roving vehicles emerged was from massive funding from these governments. From then on, many attempts to build a fully automated consumer vehicle were then made.

In the 80's, DARPA funded a project for an Autonomous Land Vehicle (ALV) which successfully completed the first road demonstration using laser technology [15] (Figure 2.3).

The first activities in Germany were held in 1980, at the Universität der Bundeswehr München, and autonomous driving pioneer Ernst Dickmanns, implemented a vision-guided Mercedes van achieving a maximum speed of 25m/s on a separated highway. The vehicle named VaMoR (*Versuchsfahrzeug zur autonomen Mobilität und Rechnersehen*, Vehicle for Autonomous Mobility and Computer Vision) was a success and an inspiration to the forthcoming projects [16].

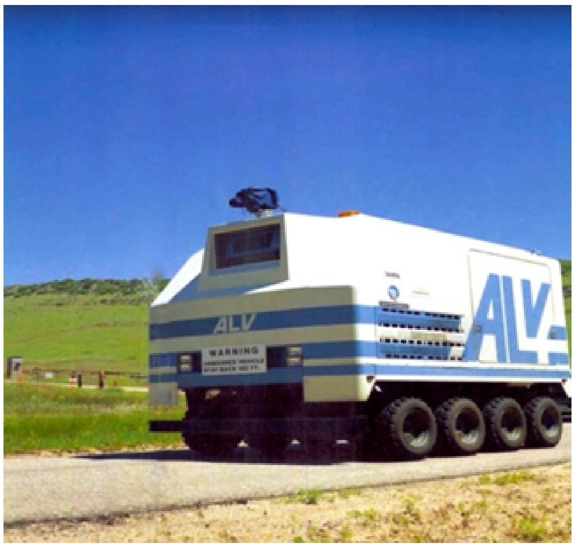


Figure 2.3: Autonomous Land Vehicle from DARPA (1980)[1]

The Eureka PROMETHEUS Project (Program for European Traffic of Highest Efficiency and Unprecedented Safety) was launched in 1987, by Daimler-Benz AG, being the biggest R&D project ever in driverless cars, with a today's equivalent of one billion dollars in funding from the European Commission [17]. From 1987 to 1994, the PROMETHEUS Project has developed an autonomous road vehicle named "Vision Technology Application" (VITA) which featured lane orientation, obstacle and intersection detection and lane changing initiated by the supervisor [18, 5]. Another vehicle resulting from it was the "VaMoRs PKW", VaMoR's automobile (VaMP). This vehicle drove more than 1000km on a Paris three-lane highway with an average speed of 36m/s [3].

Paralelly, "Rapidly Adapting Lateral Position Handler" (RALPH) vehicle by Carnegie Mellon University, drove from Pittsburgh, PA through eight states to San Diego, CA. Using a vision-based approach, its lateral control operated autonomously whereas the speed was controlled by human supervision. It reached an average speed of about 25m/s [19, 20, 21].

In 1994, Daimler-Benz demonstrated the “Optically Steered Car” (OSCAR) using a vision-based adaptive intelligent cruise control approach [22]. It was further enhanced within a vehicle called "Urban Traffic Assistant" (UTA), combining the autonomy of following a lead vehicle in urban environments and providing assistance in inner-city areas [23]. A curious introduction was the pedestrian detection, traffic sign and traffic light recognition [24, 25, 26].

Much effort has been made on the R&D of autonomous vehicles, more recently from Italy in 1996-2001 with the ARGO project [27], and from Germany in 1998 with the KLAUS project by Volkswagen [28].

In 2004, DARPA began to fund a multi-million dollar international competition for R&D in driverless vehicles. This alternative approach had a great adherence and contributed for many

[1]Image retrieved from http://www.fastcompany.com/pics/here-come-autonomous-cars?slide=2.

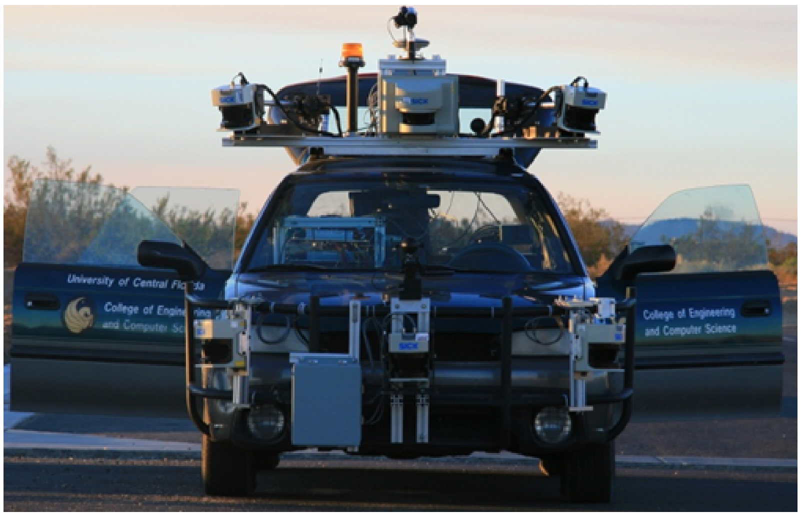

University of Central Florida
College of Engineering and Computer Science
College of Engineering and Computer Science

Figure 2.4: Knight Rider from TeamUCF in Darpa Challenge 2007.[1]

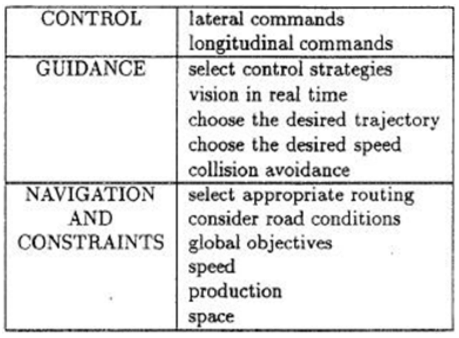
advances of correlated areas. The first competition called "Barstow to Primm" consisted of a course with a 142 miles length. From the 15 final competitors, none actually made it to the end and the farthest distance traveled for the winning team (from Carnegie Mellon University's team) was 7.4 miles [29]. Later in 2005, DARPA repeated the challenged and named it "Desert Classic" with a modification on the qualification process. From the 23 teams which made it to the final event, only four made it to the finish line in the ten-hour limit. "Stanley" from Stanford University won the first place $2 million prize [30]. In 2007, the last competition to date took place, aiming to study the behavior of autonomous vehicles in populated urban traffic networks, accomplishing a number of missions within a given time period [31]. The Team Tartan Racing from the Carnegie Mellon University won the challenge and the $2 million prize [32]. Figure 2.4 illustrates a typical sensor-equipped vehicle on the DARPA Challenges.

There was a similar program in 2006, held by German Federal Armed Forces. However, it was not a competition but a demonstration of capabilities for current robotic research. It is denominated European Land Robot Trial (ELROB). The following trials where held in 2009 and 2011 [33].

An international challenge on cooperative driving took place in the Netherlands. The goal consisted of interacting with other traffic participants to optimize the overall traffic flow [34]. Another curious company to enter silently in autonomous vehicles technology is Google, which gathered some of the best engineers from winning teams who won the DARPA challenges. Their modified Toyota Prius already logged over 140.000 miles, and uses Google's data centers and previous manually data gathered to map its position from the car sensors [35].

Commercially autonomous vehicles are increasingly appearing, however these still rely on a pre-built infrastructure making it unsuitable for personal use. There are already many achievements to the autonomous vehicles area. However, we still have many years ahead for this technology to be a part of our automobiles. The expectations are to replace 90% of today's vehicles in 2030 by fully autonomous vehicles. But what are the true advantages of this kind of technology

[1]Image retrieved from `http://knightrider.eecs.ucf.edu/node/306`



CONTROL	lateral commands
	longitudinal commands
GUIDANCE	select control strategies
	vision in real time
	choose the desired trajectory
	choose the desired speed
	collision avoidance
NAVIGATION AND CONSTRAINTS	select appropriate routing
	consider road conditions
	global objectives
	speed
	production
	space

Figure 2.5: The classic driver model [36]

facing today's traffic situation? Actually, there are a few. Giving the coldness of non-rational responses of robots, these can be very effective on avoiding many of the accidents caused by human-behavior, such as stress, drinking, high-speed and others. Also, the optimal energy efficiency for traveling and raise of its user's productivity can be the major reasons for this technology to become part of our lives. Apart from the technical issues, user acceptance, privacy and equity are some of the social aspects which still need to be tackled meanwhile.

2.2.2 System Architecture

Towards the simulation requirements over an autonomous vehicle system, a general architecture overview of the vehicle is briefly presented. This is crucial as the simulator should fulfill all timing and space requirements of the several control layers.

The architecture of an autonomous vehicle is based on the general driver behavior, depicted in Figure 2.5. This classic approach (namely human, automatic, electronic or mechanical) models the tasks to successfully guide its vehicle to accomplish a global goal, such as going from a point A to a point B or delivering a mail in a office [36].

The **control level** is the lowest level, i.e. the physical control of the vehicle, i.e. the sensors and actuators of the driver's model. It must have the highest sampling rate of all three levels (in the milliseconds order).

The **guidance level** comprehends the middle level supervision from simple tasks such as follow a line or a path and speed control to more complex tasks such as adjusting speed anticipating a curve or collision avoidance. It performs input processing such as object or lane detection based on sensor availability. Its time frame may vary from one second to a few seconds.

The **navigation level** performs higher level tasks related to driving such as controlling the global objectives, trajectory planning, efficiency and commodity, taking into account the driving conditions. The time frame is from a few seconds to several hours.

The classic driver model represents the overall concept of a vehicle driver in its general sense. However, being a sensor-actuator based robot, an autonomous vehicle also follows a sensor- and

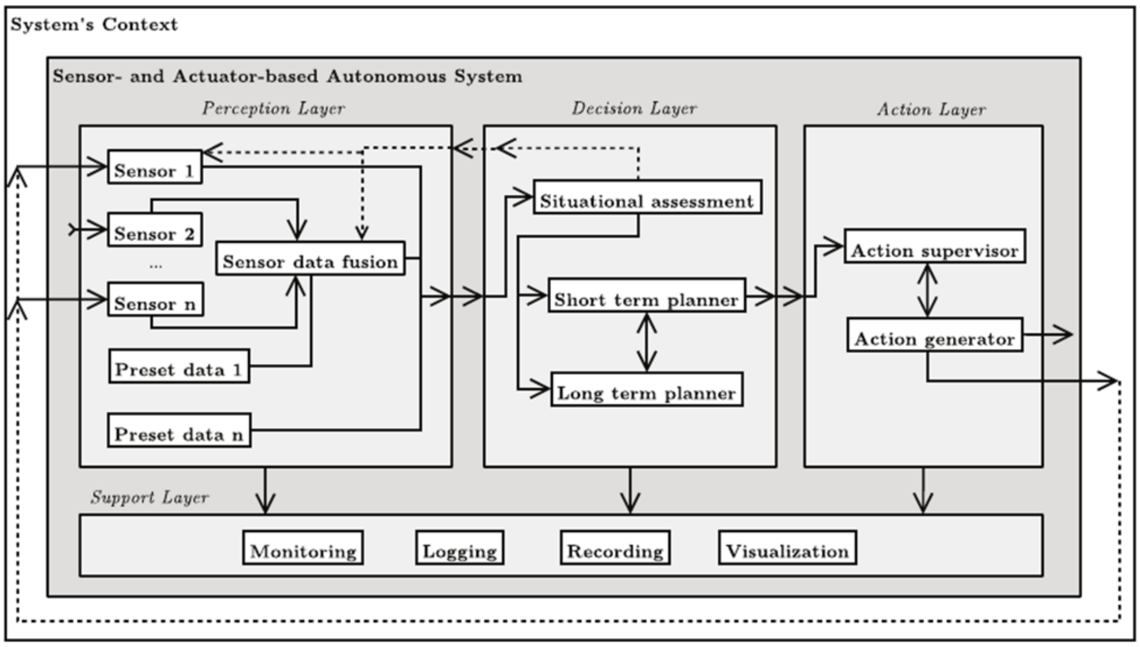


Figure 2.6: A generic sensor- and actuator-based autonomous system architecture [37]

actuator-based autonomous system architecture, consisting on a Perception Layer, a Decision Layer, and an Action Layer [37] (Figure 2.6). These layers represent the data flow between the data acquisition through the system for an action to be carried out. The Support Layer is attached below for supporting the three major parts.

The **Perception Layer** is responsible for the acquisition of all data, such as from a vision-based sensor, or radar-based sensor. Prior to the acquisition the data is merged into an unique map, having in account the confidence on the sensors' data. This is called sensor data fusion [38].

Perhaps the most complex layer is the **Decision Layer**. It is fed by the Perception Layer providing feedback data to further optimize the data acquisition and interprets all incoming data from it to generate a reasonable output to the Action Layer. The Situational Assessment provides the input evaluation for short and long term planners; they should influence each other to avoid short term decisions which do not accomplish the overall goal. Artificial Intelligence algorithms are commonly used in the Decision Layer mainly due to the highly non-linear behavior of real environment such as Neural Networks, Machine Learning, Fuzzy Logic or Genetic Algorithms [39].

The last layer, known as **Action Layer**, receives commands through the Decision Layer into the action supervisor, which sets up the abstract decision into set points to be fed by the actuators' controllers. The action generator denotes the system controllers and performs the low-level actions in the actuators, also monitoring the feedback variables to further process the new actuating variables.

The use of such an architecture for sensor/actuator based autonomous vehicles is common in many AGV applications. Furthermore, regarding the simulation of such robotic systems, one must take into account all the aspects of physical implementation to further simplify the transition from virtual- to real-world scenarios.

The Agent paradigm has been introduced as well, as a way to address some of the current issues in autonomous-based driver behavior regarding its distribution capabilities, computing efficiency and scalability [36, 40].

2.2.3 Autonomous Vehicles Simulation

Robotics simulation is already in use for several years now, and have reached a fairly stable state. Moreover, and regarding the autonomous vehicle as a robot, robotics simulators are typically used in autonomous vehicles simulation as well. In 2.3.5 a simulator comparison pointing out some important aspects is presented, with focus on the autonomous vehicles field. In the next section, a brief contextualization in the robotics history, current applications, issues and simulation is introduced.

2.3 Robotics

The word robot comes from the Czech "robota", which means "hard work" and was introduced in the science-fiction play R.U.R (Rossum's Universal Robots) in 1920 by Czech writer Karel Capek.

In this section we present a brief history of robotics, from its concept to nowadays' current development'. Current uses of robots, major issues regarding the research field, most used sensor types and a study on robotics simulators are briefly discussed.

2.3.1 A Brief History

Research on the robotics field has began some decades ago, but the underlying concepts are much older. In the Greek mythology, *Pygmalion* was a sculptor who fell in love with a statue he had carved which came into life. Around 1495, Leonardo Da Vinci sketched plans for a humanoid robot probably based on anatomical research recorded in his Vitruvian Man. In 1738, Jacques de Vaucanson designed and created a mechanical flute player, a pipe player and a duck. Later on, in 1818, the book Frankenstein written by Mary Shelley resembles the theme of robots replacing their human creators. Similar conceptions are found in recent sci-fi movies such as Blade Runner (1982) and The Terminator (1984). However more positive images of robots were also seen in A.I. (2001), I,Robot (2004) and Wall-E (2008). Science fiction writer Isaac Asimov, created the Three Laws of Robotics which were written in the 1942's' short story Runaround [41]. The laws can be summarized as below:

1. A robot may not harm a human being, or, through inaction, allow a human being to come to harm;

2. A robot must obey the orders given to it by the human beings, except where such orders would conflict with the First Law;

3. A robot must protect its own existence, as long as such protection does not conflict the First or Second Law.

In early days, robots of all kinds of shapes are being inserted on our society's way of life. In the description below, some achieved milestones which approximated the robotics field from industry to consumer electronics is presented (partially adapted from [42]), as chronologically listed below:

1956 The first robot company called Unimate was created by Joseph Engelberger together with George Deroe;

1961 The first commercial robot is shipped by Unimate;

1968 Shakey was developed by Stanford Research Institute, being the first robot able to reason about his own actions [43];

1970 ASEA company formed an industrial robotics division (ABB Robotics nowadays) with first production robot in 1974;

1977 Using a stereo vision in order to percept the environment, The Stanford Cart moved about four meters per hour;

1984 The Waseda University presented the Waseda biped robot that walked many kilometres, and the Wasebot piano playing humanoid;

1986 Honda develops E0, a humanoid robot prototype that inspired a series of twelve following models;

1990 Carnegie Mellon demonstrated autonomous highway driving at high speed with vehicle Navlab 5;

1994 In Germany, Dickmann's group from Munich demonstrated autonomous driving in Paris on heavy traffic;

1996 Huskvarna presented a robotic lawn mower;

1997 The Sojourner rover operated on the Mars surface;

1998 HONDA annonces the ASIMO robot series, acronym for "Advanced Step in Innovative MObility" [44] (Figure 2.7);

1999 AIBO pet was introduced, a well-known and popular robot from Sony;

2000 The first vacuum cleaner robot was shipped by Electrolux, named Trilobite.

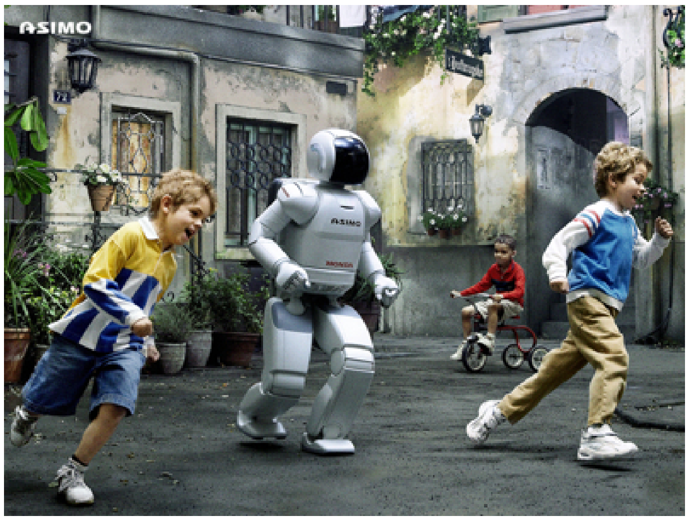


Figure 2.7: ASIMO robot from Honda. In the future, robots will be involved in common social scenarios.

Nowadays, robots are present on a variety of contexts, namely Industrial Manufacturing, Mining, Unmanned aerial vehicles (UAV), Autonomous Underwater Vehicles (AUV), Autonomous Ground Vehicles, Service Robots and more. The focus of this thesis is regarded on autonomous vehicles simulation which is a subfield for autonomous robots. Therefore, the following sections are referred to devices which should not require human intervention.

2.3.2 Characteristics of Robotics

The use of autonomous robot technology is highly acclaimed due to its peculiar characteristics. Here are some of research areas that have challenged the scientific community as well as the industry:

Mobility Robots always have moving parts. Moreover, they can be stationary (e.g. manipulator) or movable, notably an autonomous vehicle. New locomotion approaches are being developed as well, many of them inspired on animal behavior, for instance;

Sensing To perceive the surrounding environment, the robot uses sensors, e.g. GPS, camera, laser devices. This characteristic is explored in detail in Section 2.3.4;

Energy To power itself, the robot needs energy. Commonly it relies on battery packs, optionally with solar panels. The excessive price of a reliable energy source for mobile robots is one primary issue for robot investigation;

Versatility There are a lot of specialized robots nowadays, particularly in industry, e.g. painter robots and welding robots, or in consumer electronics, e.g. robotic pets or vacuum cleaners. Nevertheless, specialists have higher goals regarding to systems capable of carrying out wider range of tasks. One example of it is the construction of anthropomorphic robots

(robots with human resemblance) which should in practice be able to carry out random human-like activities;

Artificial Intelligence Apart from roboticists not agreeing they are working within the field of artificial intelligence, as the fields are decoupled nowadays, several techniques from AI are fairly used such as artificial neural networks (ANN), fuzzy logic or methods for learning and navigation planning. The use of such practices is a key ingredient to obtain a more robust response from the robot albeit its use is not imperative.

2.3.3 Current Trends and Issues

Regarding the characteristics described above, there are yet many limitations on robotic development, not only software but also hardware related. Robotics is considered a multi-disciplinary field, as it can involve electronics, mechanics and mechatronics in a hardware level, and control-theory, computer science, programming and mathematics at the software level.

Some years ago, R&D on robotics was very expensive, as researchers had to build their own hardware. Today robots can be purchased on-package, i.e. with a hardware and software bundle, which minimizes building costs, arguably giving more time to tackle software paradigms. Also, the increasing computational power and data storage at low costs give opportunity to anyone willing to enter in the area. Sensing has improved a lot as well, with e.g. optical cameras, laser range-finders becoming smaller and cheaper. One must note however that state-of-the-art sensors are obviously very expensive, and some of them are being introduced, e.g. in autonomous vehicles. Further discussion in sensor types is presented in the following section. As referred already, battery hardware and device consumption is also one of the major problems. Electrical batteries are still very heavy and have low energy density when compared to petrol.

In order to have a full autonomous robotic system, commonly regarded as a complex system, a whole software architecture must be carefully modeled. Some of the key modules which must be present on such device are locomotion, navigation, data fusion, localization and planning, interaction with moving objects (e.g. humans) and obstacle avoidance [45]. Each of these competences requires a specific domain knowledge, meaning that in the process of developing a full-featured robot, many field experts are needed. Also, a well-defined methodology and software framework is mandatory, to provide scalability and cost-effective production. To mitigate these problems, using methods such as software engineering, software re-use [46] or simulation is critical to minimize development times maintaining code uniformization. In the today's reality', besides the use of a common software bases for easy exchange of algorithms across laboratories and technology transfer, researchers use their own tools to develop the entire robotic system, sometimes using their own simulation tools as well [47].

2.3.4 Robotic Sensors

One of the objectives of this thesis is the simulation of autonomous vehicles in a urban traffic environment. Thus, some important aspects to take into consideration while selecting a robotics simulation framework should be the types of sensors supported and their resemblance to reality, e.g. the virtual and real sensor model (range laser), or camera output when compared to reality.

The following list contains the most used sensor types in autonomous robots, with a particular focus on ground vehicles. A comprehensive review was also made on technical papers from the DARPA's Urban Challenge 2007 teams.

GPS Provides absolute location and velocity using orbiting satellites. However, it only works on open spaces. High quality GPS systems can have a precision on the centimeter scale when in optimal conditions; [48].

Optical Camera Gives vision to the robot. On controlled conditions, vision-based algorithms recognize anything in the surroundings;

Infra-red Camera Similarly to the former optical camera, it gives vision to robots in the infra-red spectrum. It is widely used along with an infra-red light to provide night vision;

Laser scanner Also known as Light Detection And Ranging (LIDAR) sensor (Figure 2.8). Laser scanners are one of the most popular and expensive sensors used on autonomous vehicles. It returns information of surroundings as a 2D or 3D point cloud, typically at a 12.5Hz sampling rate, i.e. more than a million points per second (on a 3D model);

Ultrasound Measures short distances using the sound spectrum;

Radar Similarly to ultrasound, but in the electromagnetic spectrum. It can achieve even more precise results at a long distance;

Odometry Estimates changes in velocity, acceleration and position from moving sensors, e.g. legged joints or wheels;

Inertial Measurement This sensor type returns a measure from the relative movement of the robot, in a linear or angular context, using an accelerometer or a gyroscope;

Compass Determines the robot direction relative to the Earth's magnetic poles.

Actuator types are not going to be thoroughly evaluated as the sensors were, as the only actuators present in autonomous vehicles are the locomotion and steering systems which are widely supported in any up-to-date simulator. These elements should as well be extensively tested with *hardware-in-the-loop* techniques as commented in Section 2.1.2.

2.3.5 Robotics Simulators

Nowadays, robotics simulators are indispensable for reliable intelligent robot development, concerning overall system complexity and expenses. Moreover, as autonomous vehicles research is a branch of robotics, typical solutions seem to adopt these as a basis for autonomous vehicles

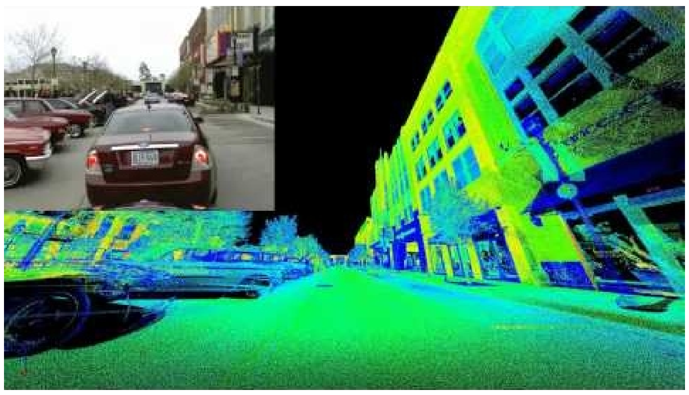


Figure 2.8: The aspect of a point cloud data retrieved from a LIDAR sensor. The visualization uses a distance-color encoding.

simulation. Also, when using a Hardware Abstraction Layer (HAL) between the hardware level (actuators/sensors) and the vehicle control algorithms, more flexibility would be achieved in the two parts as if they were virtually decoupled.

Bearing this in mind, a taxonomy for comparing autonomous vehicles simulators is proposed. We assume that these simulators perform both reliable 3D simulation and visualization of the scenario, and also are widely used by the research community. At least one robotics HAL framework must be supported.

The following criteria depict the most important aspects to take into account while selecting a robotics platform in an autonomous vehicles approach:

3D rendering The visual robustness of the simulation;

License Whether the simulator is open-source, free of charge, or commercially licensed;

External Agent Support In order to control a vehicle using an agent-based methodology the simulator should feature a distributed architecture at the control level;

Parallelism/Distribution In order to distribute processing power over processor cores or networks;

Level of Maturity If the simulator is already widely used and validated;

Fault-tolerance When a hardware module fails, higher level modules should rapidly make decisions whether to stop or adapt the control system. When developing a final product such behavior should be strictly tested;

Realistic Scenario Simulation The level of realism to simulate difficult context scenarios e.g. snow, day and night, and wind, not only interactively but also physically, i.e. affecting the sensorial input.

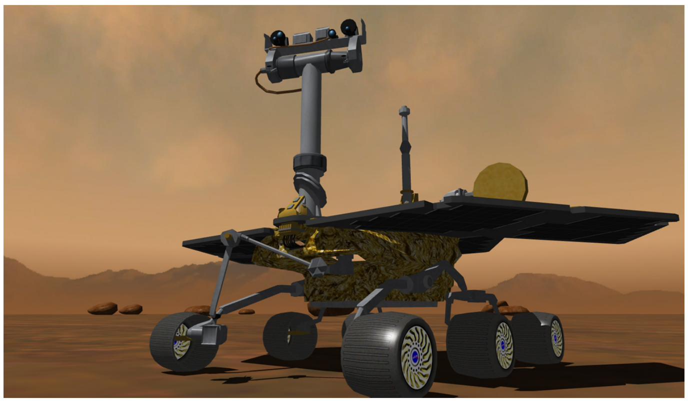


Figure 2.9: A Mars rover simulation in MRDS[1].

2.3.6 Simulators Analysis

After careful research on the related papers, this analysis was limited to the review of two open-source and two closed-source simulators: MRDS (Microsoft Robotics Developer Studio) [49], Player/Stage/Gazebo [12], USARSim [11] and Webots [50].

MRDS is a robotics platform from Microsoft Company using .NET-based technology. It features visual programming, web- and windows-based interfaces, 3D simulation with advanced physics, as well as easy access to robot's sensors and actuators from a number of languages. Figure 2.9 illustrates a typical simulation on MRDS.

The Player/Stage project is an open-source distributed robotics platform. It features a distributed POSIX-compatible HAL on a minimal design. Stage and Gazebo are its 2D and 3D simulators, respectively. Gazebo is build on top of the Ogre3D rendering engine (also open-source) to provide more realistic environments.

USARSim is a high-fidelity open-source simulator based on the Unreal game engine. It is the official simulator platform of the Robocup virtual robot competition [51] supporting a wide range of sensors with noise input. The USARSim simulator is fully compatible with the Mobility Open Architecture Simulation and Tools (MOAST) [52] and Player frameworks. One curious characteristic of MOAST is that it implements the Real-time Control System (RCS) reference model architecture [53]. This model was developed by National Institute of Standards and Technology (NIST), and is suitable for many software-intensive, real-time control problem domains. It is commonly applied in several types of robot control such as autonomous vehicles control.

Webots is a commercial robot simulator developed by Cyberbotics and it is being used by more than 800 universities and research centers worldwide. It has reached a fairly stable state and supports a wide range of hardware. It is also supported by the Universal Robot Body Interface (URBI) [54], a client/server based framework targeted for humanoid devices.

Using [47] and [55] as a reference for this study, the following table summarizes these simulators" capabilities on the basis of the earlier proposed criteria.

[1]Courtesy from `http://www.wikipedia.org`

Table 2.1: Features comparison of robotics simulators for agent-based autonomous vehicle simulation.

Simulator	License	External Agent Support	Parallelism/Distribution	Level of Maturity	Fault-Tolerance	Realistic Scenario Simulation
Ms Robotics	Free	No	Yes^3	Medium	No	High
Gazebo	GPL^1	Yes	Yes^3	Medium	No	Medium
USARSim	GPL	Yes	Yes^2	Medium	No	High
Webots	Commercial	Yes	Yes^1	High	No	Medium

Support for autonomous vehicles in robotics simulators lacks some key characteristics in order to realistically simulate from the sensor to environment level. However, its state already allows practitioners to perform fairly complex testing within its development life-cycle.

2.4 Traffic Simulation

The use of simulation methodologies in the field of Transport Systems is widely acclaimed for decades. If we look for current transportation state in urban scenarios, high traffic saturation levels due to the increasing demand and unoptimized transportation planning is evident [56].

Traffic simulation tools intend not only to cope with undesired events as mentioned above, but also to generate scenarios, optimize control, and predict network behavior at the operational level. This allows a specialist to virtually modify the network topology or the traffic control strategies in order to validate the reliability of new models without any disruption to traffic in a real network. With the emergence of ITS and FUT [57], traffic simulators are as well being constantly upgraded to support new features such as inter-vehicular communications (IVC), multi-modal simulation, or environment and emissions reporting [58]. Figure 2.10 depicts Paramics, a state-of-the-art Traffic Simulator from a British Company.

Traffic models are generally classified according to the following criteria[59]:

- Scale of the independent variables (continuous, discrete, semi-discrete);
- Representation of the processes (deterministic, stochastic);

[1] A license for Unreal Engine must be purchased. Future versions will use Unreal Development Kit, which is free of charge.

[2] Supported on robot domain. Latest development version also features distributed simulation processing, i.e. sensors can be simulated on different machines.

[3] Only supported on robot domain.

Figure 2.10: Paramics Traffic Simulator offers a high degree of realism[1]

- Scale of application (networks, links, and intersections);
- Level of detail (submicroscopic, microscopic, mesoscopic, macroscopic).

Since a simulation model usually describes a dynamical system, its time-scale should be considered. A discrete model represents state changes discontinuously after some regular time interval or event has passed. Contrarily, a continuous model describes the traffic changes continuously over time. Albeit the time-scale is characterized by these models, other variables can also be either discrete or continuous (e.g. position, velocity) [60].

Regarding the representation of the processes, these can be deterministic or stochastic. The deterministic models are predictable, i.e. if two simulations are run with the same input parameters, the output of each other would be the same. However, as stochastic models are derived from probabilistic theory the results of the two simulations should differ, as random and unpredictable behavior is present. One example would be the estimation of a lane flow, by a constant value or a random variable for deterministic and stochastic models, respectively [60].

With respect to the scale of application, a model should represent the dynamics of single entities (e.g. a lane, a corridor, a whole network), according to the area of application of it.

As for traffic simulation models, there are four distinct types depending on their different granularity scales, i.e. in terms of their *level of detail* classification. These are the macro-, meso-, micro- and submicroscopic modeling approaches.

The macroscopic simulators are based on mathematical models describing the vehicles' flow through the network, an approach similar to the fluid dynamics. More robust models have various types of vehicles, but they all follow as well the above principle [61].

[1]Courtesy from http://www.paramics-online.com/paramics-media.php

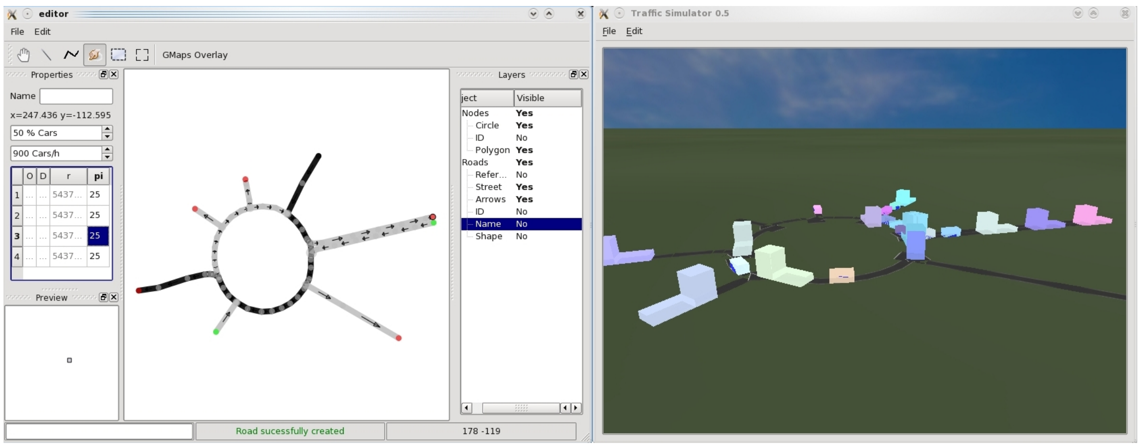


Figure 2.11: A microscopic traffic modeler (of the left) and simulator (on the right) developed on LIACC/FEUP.

The microscopic traffic simulation consists of a set of models representing the individual vehicle behavior in a traffic road that should be calibrated to follow the macroscopic traffic flow patterns. Despite its complex configuration, once a good calibration is set up, the model follows the macroscopic traffic flow patterns allowing for a wider analysis on the vehicle behavior, proving itself suitable for individual intersection optimization, e.g. traffic light planning. A microscopic traffic modeler and simulator are illustrated in Figure 2.11.

The mesoscopic simulation manages to get the advantages of both macro- and microscopic simulators, combining the high level detail of entities, but describing their interactions and behaviors in a lower level, for instance in probabilistic terms. These descriptions can take different approaches, such as with vehicle grouping as a single entity with its speed calculated for each link using a speed-density based function or with individual vehicles grouped into cells that manage their behaviors being the cell responsible for determining the speed of individual vehicles [62].

In addition to describing the time-space behavior of the individual entities in the traffic network system, submicroscopic simulation also known as nanoscopic simulation describes the functioning of specific parts and processes of vehicles and driving tasks, i.e. apart from a detailed description of the driving behavior, the vehicle control behavior (e.g. tire deformations, changing gears, inertia) is also modeled in detail in correspondence to prevailing surrounding conditions [59].

If one takes into account all described *level of detail* models as described above and the requirements needed to the reliable development of autonomous vehicles simulator in an urban environment, the use of microscopic or even submicroscopic levels should be considered, given the fact that individual driving behavior algorithms and sensing/actuating features are to be tested.

2.4.1 Microscopic Traffic Simulation

The general approach when simulating traffic in a microscopic *level of detail* is to treat the driver and vehicle as one single unit. Hence, there are several dynamic rules also called behavioral models, each one supporting a specific interaction. The most important behavioral model is one that permits handling the longitudinal interaction between two preceding vehicles, and is commonly known by the car-following model. However, there are other sub-models used depending on the simulators' capabilities and type of road to be simulated (e.g. lane-changing, gap-acceptance, overtaking, ramp metering, speed adaptation) [63].

The aforementioned rules should be reliable to simulate a realistic traffic environment. However, they still suffer from the following drawbacks:

- They are usually decoupled from each other (i.e. on a lane changing, no longitudinal acceleration is affected);
- The vehicles lateral position is discrete (per-lane);
- There is no cooperation capabilities among the various entities of the road.

Other recent improvements on microscopic traffic simulation are the integration of the agent computing paradigm into many aspects of transportation systems, such as in modeling and simulation, dynamic routing and congestion management and intelligent traffic control [64]. Also, the use of game engines technology in simulators is proposed in [65] with promising potential.

2.4.2 Evaluating Traffic Simulators for Autonomous Vehicles Simulation

Bearing this thesis project's perspective in mind, some microscopic traffic simulators are briefly analyzed in order to choose the most suitable one to meet its requirements. One must note that only fully validated and widely used simulators are described. We assume these simulators perform the minimal aspects of a reliable microscopic traffic simulation to simplify the study. In [66], a comparison taxonomy for microscopic simulators applied to FUT is proposed, thereby offering a good starting point into the criteria selection for the integration with autonomous vehicle simulators. Furthermore, the selected criteria are described below:

Extensibility When using a closed-source software, its extensibility should be analyzed in order to study if it suits the integration of other tools, i.e. the level of accessibility of the simulation core;

Software License Open-source simulators are generally inferior in features as compared to commercial ones. Nonetheless, when well documented they tend to be more flexible and rapidly extended due to community support;

External Agent Support The ability to use the agent technology, not only in driver behavior modeling, but in simulation initiation, control or deployment;

Parallelism/Distribution To support a large traffic scenario, simulators must feature distributed processing over several cores or a computer network;

Inter-vehicular communications (IVC) Virtual communication infrastructure support for V2V or V2I and physical restrictions simulation must be present as well;

Interactivity What features are controllable from simulation in run-time, and general graphical aspect;

Level of Maturity Whether the simulator is widely used and validated by the scientific community.

A taxonomy to compare microscopic traffic simulators and their applications to multi-agent autonomous vehicles simulation was proposed. Finally, some results are described in the following section.

2.4.3 Traffic Simulators Analysis

Following the comprehensive analysis in [66], a general overview of the referred simulators against the selected criteria is presented, namely VISSIM [67], PARAMICS [68], AIMSUN [69], MITSIM [70], SUMO [71] and MAS-T2er Lab [72].

Table 2.2: Feature comparison of microscopic traffic simulators for agent-based autonomous vehicle simulation.

Simulator	License	Extensibility	Agent Oriented	Parallelism/Distribution	IVC	Interactivity	Maturity Level
VISSIM	Commercial	Yes	No	Yes[3]	No	High	High
PARAMICS	Commercial	Yes	No	Yes[34]	No	High	High
AIMSUN	Commercial	Yes	No	Yes[3]	No	High	High
MITSIM	Both	Yes[2]	No[2]	Yes[3]	No	Low	Low
SUMO	GPL	Yes[2]	No[2]	Yes[3]	No[5]	Medium	High
MAS-T2erLab	Free	Yes[1]	Yes[1]	Yes[3]	No	Medium	Low

Regarding the extensibility of these simulators, all of them offer some type of modularization. MAS-T2er Lab simulator only provides an UDP connection to control semaphoric intersections and statistical data. SUMO simulator provides an extension named TraCi, which provides statistical data and direct access to some core elements, however it is in a very embrionary level. Nonetheless, it has been extended by the TrasMAPI project to support the implementation of agents in Java [73]. All commercial simulators seem to fulfill this requirement, but a tougher analysis should be needed to evaluate the possibility of integrating external vehicles (e.g. from an autonomous vehicle simulator) in real-time into the simulations.

VISSIM, PARAMICS and AIMSUN are full closed-source packages, MITSIM have both the close- and the open-source variants and SUMO and MAS-T2er Lab are open-source, being SUMO the most featured and referenced open-source project with over hundred papers. The importance of simulators license is high, as in the case of modification needed on the core, only open-source ones can fully allow it.

Only MAS-T2er Lab seems to support agent-based driver behavior simulations, with multi-connection and local information.

PARAMICS is the only *off-the-shelf* simulator supporting distributed computing over a network. The remaining simulators support parallel processing over all CPUs.

Inter-vehicular communications are not supported by any of the commercial variants, as they do not seem to be targeted at research purposes. Only SUMO was already modified to support this kind of simulation [74].

Regarding the interactivity criterion, the most 3D realistic simulators are the commercial ones, followed by MAS-T2er Lab. Only SUMO and MITSIM do not have 3D visualization. In-simulation parameter modification is only widely supported by commercial applications and SUMO. With regard to the last criterion, commercial simulators have an expected high maturity, and on the open-source side, only SUMO is being actively developed.

There is an obvious disparity when looking at microscopic traffic simulators. Although commercial packages are very expensive, they provide the most feasible results. However, their source-code is not accessible, meaning that if an important core modification was needed, a dead end would be reached.

On the other hand, open-source simulators are in an inferior maturity level, albeit they provide full custom control of the application which seems to be most suitable when in a research context. A brief summary of the aforementioned observations is presented in table 2.2.

2.5 Summary

This chapter demonstrates a brief overview of the current state of the art on game engine technology, autonomous urban vehicles, microscopic traffic simulation and robotics simulation domains.

The importance of game engines in today's game industry was described, as well as its increasing potential as a simulation framework support for scientific purposes. Furthermore, the autonomous urban vehicle history was briefly presented, as well as its generic system architecture along with its relation to the robotics simulation field. With it, some simulation tools were analyzed having in mind the requirements for a successful simulation of autonomous urban vehicles. Finally, the traffic simulation topic was approached regarding its models classification, and some

[1]Only for external traffic light control.
[2]Being an open-source software, it can be extended through source code modification.
[3]Supports Parallel simulation processing.
[4]Supports Distributed simulation processing.
[5]Successful implementation reported in [74].

simulators were analyzed having in mind the same requirements as mentioned already. In the next chapter, the overall solution for simulators integration is introduced, along with the key issues to be addressed while developing the framework.

Chapter 3

Solution Design

In the former chapter, a comprehensive review on current state-of-the-art topics for this thesis document were presented, with a particular focus on the analysis of the most well-known traffic and robotics simulation frameworks to date.

This chapter describes a proposed high level architecture towards integration of both a traffic and a robotics simulator.

First, current issues for integration of the aforementioned simulators are delineated along with practical solutions aiming to address them. Finally, the selected simulators are comprehensively introduced and justified, and the integration framework is represented throughout a high level system architecture.

3.1 Integrating Traffic and Autonomous Vehicle Simulations

One of the key criterion for simulator selection is its distributivity. In other words, each simulator must allow a networked access to its core level, in order to provide high-speed data interconnection between them.

To integrate the two simulators in time and space, a bidirectional communication should take place, with the autonomous vehicles providing kinematic variables to the traffic simulator. The traffic simulator then calculates its surroundings and return its data back to the autonomous vehicle simulator. All of this transactions should occur in the same time step.

The proposed architecture for integration must allow the simulation to be performed in real-time, i.e. we will consider 30 Hz as the minimum frame rate for this real-time simulation.

To implement such an infrastructure between the two simulators, a set of parameters should be properly handled on the data exchange methodology. Furthermore, some critical issues must be properly tackled to achieve the desired goal:

Network topology consistency between simulators Both the traffic and the robotics simulator require specific scenario information. While the traffic simulator may need consistent data about road characteristics and positions, such as number of lanes, road segments or traffic lights status, the autonomous vehicle simulator demands for a highly realistic and calibrated

3D scenario with accurate road geometry, usually not present in most traffic simulators. In order to frame this issue, each file format for the two simulators should inherit from the same network topology description file (e.g. a shape file) towards internal consistency.

Synchronization of Simulators Although traffic simulators are not implemented with hard real-time constraints, the processing power of today's computers allow us to consider this is quite acceptable. As most traffic simulators support more than a thousand individual vehicles in real-time, a 30 Hz frame rate may be achievable on an optimized data exchanging between simulators. If we take into account this simulation approach should support more than one autonomous vehicle, a large data flow should be expected between the two simulators. Moreover, all step calculations need to be inferior to the overall frame rate of the simulation for a correct user experience.

From the former synchronization critical issue, an obvious choice for solving such problem is the minimization of data flow between traffic and autonomous vehicle simulators. Below, three methods are presented aiming to minimize them:

Simulation of the surrounding elements only in the autonomous vehicle simulator When simulating an autonomous vehicle entity, only its surrounding elements (vehicles, traffic lights, etc.) information is needed to process the next sensors' state. Moreover, to save network resources between simulators, a 2D envelop is defined around the autonomous vehicle which will be synchronized with the traffic simulator. Using this approach, the internal congruence can be maintained between the simulators without having a large amount of redundant data among them.

Serialized binary data Former distributed approaches make use of the XML file format for data exchange. However, newer formats such as protocol buffers [75] reports 20 to 100 times more efficiency in data transactions. This technology approach is well acclaimed in today's large-scale distributed systems (e.g. Google data centers).

Asynchronous data exchange To minimize data flow and to account for its content, instead of a specific number of transactions between simulators be exchanged in each time step, it should be reduced to when a change in the variable state happens. For instance, considering an autonomous vehicle X is in a state S_τ in step τ, only when $S_\tau \neq S_{\tau+1}$ the simulator will report the traffic simulator about its changes.

One must note that, although the cited issues are the most obvious problems for integration of a traffic simulator with an autonomous vehicle simulator, implementation issues may also arise depending on the chosen software and the architecture of the whole system. Furthermore, an architecture towards the integration of the simulators is proposed in the following section.

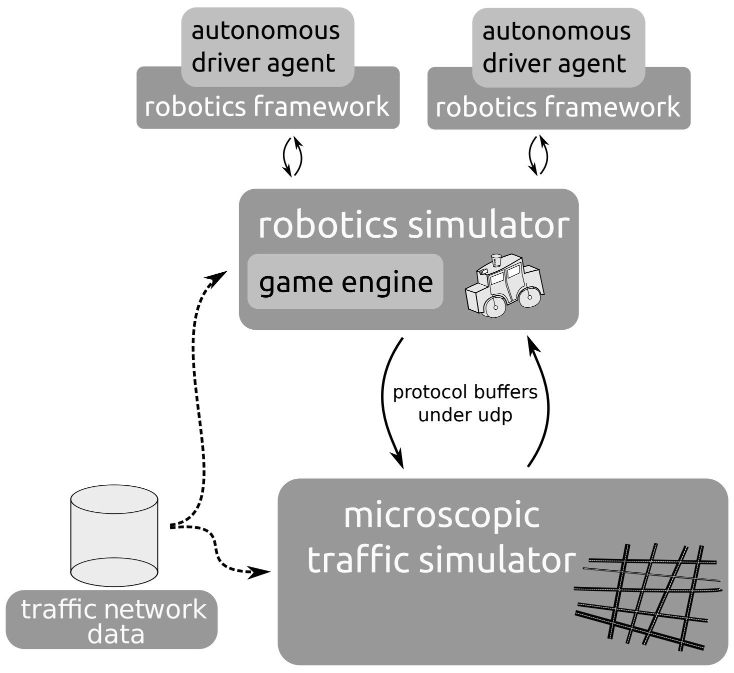


Figure 3.1: The proposed architecture for autonomous vehicle simulation in a traffic environment

3.2 The Proposed Architecture

This section aims to provide a technical design and solution to the integration of a traffic and an autonomous vehicle simulation. All issues referred earlier on were taken into consideration, and a practical solution is proposed below. First, the two simulators are chosen according to the requirements discussed already, followed by the high level architecture of the system, depicted in Figure 3.1.

3.2.1 Simulator Selection

As pointed out earlier, the use of two types of simulators may be a feasible approach when simulating autonomous vehicles on an urban environment. Such an idea was already introduced in [47], although the selected software architecture and frameworks did not provide satisfactory results. In the former paper, the traffic and robotics simulation softwares chosen did not have the sufficient maturity level, which led to various implementation difficulties and, consequently, to an inefficient platform. However, its findings have provided some lights on the usefulness of the integration and difficulties to be overcome.

The software frameworks selected to be used on the implementation of this project are the SUMO and USARSim simulators. Although their maturity level is inferior to their commercial counterparts, they have a strong support from the open-source community and will allow for full core access which should be favorable for low level optimizations.

SUMO is a highly portable, microscopic road traffic simulation package designed to handle large road networks and has a strong commitment with the academia and research community. It is mainly developed at Institute of Transportation Systems at the German Aerospace Center and it is written is C++.

USARSim is a high-fidelity robotics simulator based on the Unreal Tournament game engine. Due to Unreal license restrictions, USARSim is written in Unreal Script, the official scripting language which offers interface with the engine core. This particular issue can be crucial as there is no direct access to all the features of the engine. However, USARSim's current state and high quality sensor simulation and physics rendering make it the best choice to the project. One current major drawback of USARSim is its dependence on Windows Platform, as Unreal Engine 3 only supports it. However, as there are no other dependencies to the remaining softwares, all the produced software will be multi-platform.

3.2.2 The Integrated Simulator Architecture

A software architecture for the autonomous vehicle simulation in a traffic environment is proposed, as depicted in Figure 3.1. It consists of four major modules, briefly described below:

Microscopic Traffic Simulator Simulates almost all vehicles with a resemblance to the macroscopic simulation of real traffic streams. It also maintains all infrastructure systems, such as induction loops or traffic light plans. Given the extensibility of the simulator, a higher level statistical framework may be coupled in order to study traffic behavior patterns of individual and cooperative strategies for autonomous vehicles;

Robotics Simulator Performs the simulation of all autonomous vehicles in the environment, along with all its sensors and actuators, and mirrors every surrounding object. It also features a game engine for immersive 3D animation through both powerfull physics and visualization modules;

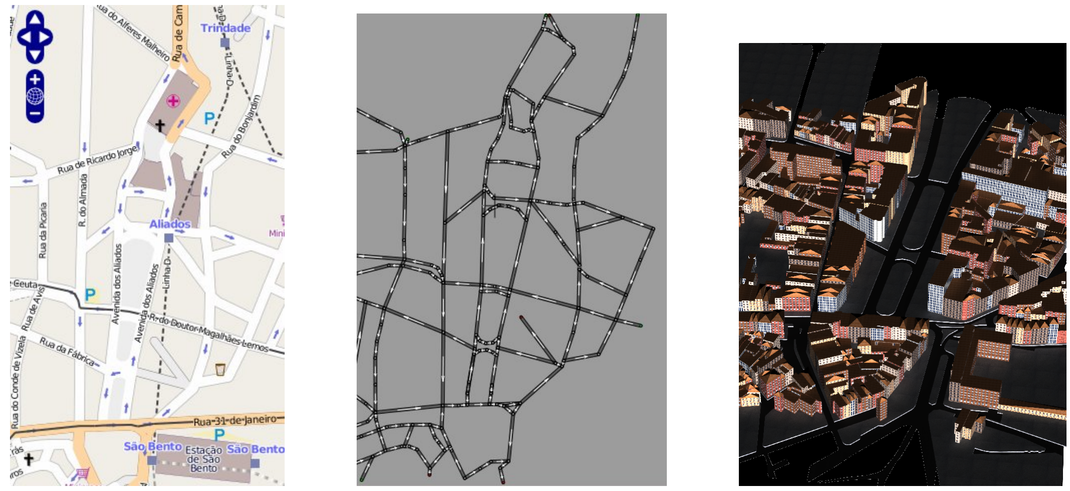
Coherent Network Data Represents the traffic network topology model, as well as its realistic 3D environment data;

Autonomous vehicle interface and control Manages the brain of the vehicle. Typically an external software driver can be deployed to perform the autonomous vehicle high-level tasks using an agent-based methodology. A Hardware Abstraction Layer (HAL) is underneath for transparent real/virtual world development, and sensor/actuator permutation.

3.3 The Prototype Development

A prototype describing the proposed framework was developed and comprehensively documented in the next chapters. Its methodological approach is many-fold:

Modify SUMO simulator SUMO is a vehicle- and edge-based microscopic simulator. Therefore, it is not prepared to allow lane-independent vehicles to circulate. Also, a spawning



(a) Aliados in OpenStreetMaps (b) Aliados in netEditor (c) Aliados in a 3D model

Figure 3.2: Aliados in several representations.

mechanism must be implemented to carefully place and identify an autonomous vehicle in the network. Finally, a communication class must be implemented for it to efficiently communicate with USARSim. These modifications can be consulted in Chapter 4;

Modify USARSim simulator USARSim is also not fully prepared to accommodate a urban autonomous vehicle. The latest version (UE3 based) does not provide LIDAR range scanner, and there are no 3D models of a four wheeled conventional vehicle and realistic terrain. Similarly to the previous step, a communication class needs to be implemented. The reader is referred to Chapter 5 for further informations on selected modifications;

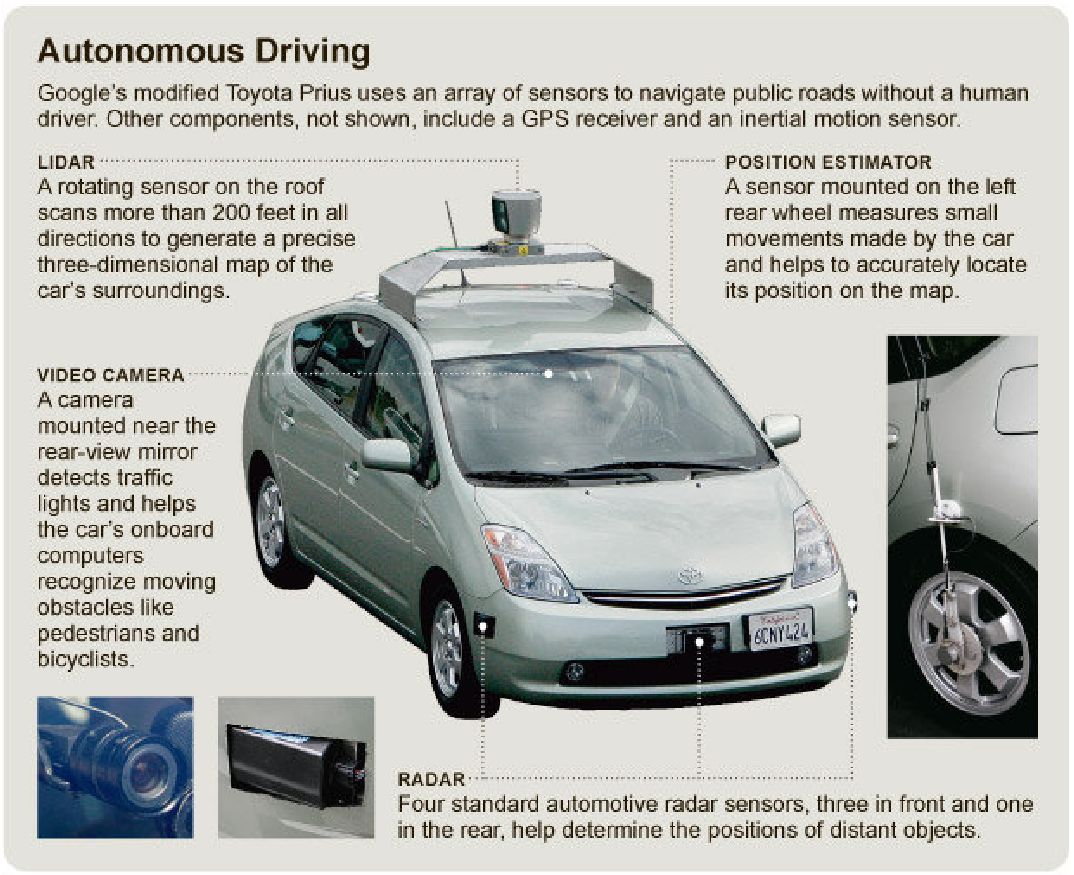
Integrate simulators Apply the required modifications to integrate both simulators (detailed in Chapter 6);

Implement a control agent To successfully validate the proposed architecture and prototype in its essence, a simple agent needs to be coded, with a simple dashboard showing real-time sensor data from the autonomous vehicle. The control algorithm will make the vehicle move through a white line, obeying the car-following rule, and possibly deviating from random objects. In Chapter 7 all implementation details are discussed.

3.3.1 Reference Network

The reference network to which the autonomous vehicles are going to be simulated on is the the zone of Aliados, Porto, Portugal. Its model has to be represented in two forms, in order to be applied in the project. Each one will fit the requirements of the two SUMO and USARSim simulators, namely a topology model and a 3D model.

In the SUMO case, the netEditor [76], a traffic network editor developed in FEUP/LIACC, is going to be used to import the required network from OpenStreetMaps (OSM), implement an O-D


Autonomous Driving

Google's modified Toyota Prius uses an array of sensors to navigate public roads without a human driver. Other components, not shown, include a GPS receiver and an inertial motion sensor.

LIDAR
A rotating sensor on the roof scans more than 200 feet in all directions to generate a precise three-dimensional map of the car's surroundings.

POSITION ESTIMATOR
A sensor mounted on the left rear wheel measures small movements made by the car and helps to accurately locate its position on the map.

VIDEO CAMERA
A camera mounted near the rear-view mirror detects traffic lights and helps the car's onboard computers recognize moving obstacles like pedestrians and bicyclists.

RADAR
Four standard automotive radar sensors, three in front and one in the rear, help determine the positions of distant objects.

Source: Google

THE NEW YORK TIMES; PHOTOGRAPHS BY RAMIN RAHIMIAN FOR THE NEW YORK TIMES

Figure 3.3: The autonomous vehicle by Google Inc. This vehicle is the reference for the prototype development[1].

Matrix for starting nodes, and select all allowed directions on each road (refer to Section 4.5 for more details). Figure 3.2a represents the OSM Aliados topology, and Figure 3.2b the imported network from OSM.

Regarding the USARSim simulator, the 3D model from Aliados was also gathered from FEUP, using a software designed and developed in [77]. This software uses Procedural Modeling and geographically referenced information to render a realistic scenario to be used on computer games. This is crucial as this work intends to make the simulation framework as realistic as possible, and a rapid prototyping of 3D scenarios is imperative. Figure 3.2c depicts Aliados 3D model rendered in the Autodesk 3ds Max modeling tool (refer to Section 5.2.1 for more details) [78].

3.3.2 Reference Vehicle

In this section, the reference vehicle which this prototype is going to use is the Google Autonomous Vehicle (not an official name). This modified Toyota Prius vehicle uses the most common sensors for automatic environment recognition and position estimation. Its brief presentation and aspect is depicted in Figure 3.3.

[1]Courtesy from `www.nytimes.com`

A brief description of sensor types was already presented in Section 2.3.4, as they are very common in robotics applications.

The 3D model to be rendered with USARSim is not going to be the same as in Figure 3.3, as its 3D model is not freely available in public repositories. Refer to Section 5.2.3 for a more detailed description of the vehicle implementation.

3.4 Summary

A practical solution for the integration of a microscopic traffic simulator with a robotics simulator was presented. Although there are still many gaps in the aforementioned simulators to be bridged for a *quasi*-realistic framework, this proposed point of view would allow specialized practitioners to handle each simulator as if they were virtually decoupled from each other, and therefore, their original documentation can still be used. This is a key feature of the architecture, as there is almost no documentation needed to setup the integration. In the following chapters a more technical overview of the prototype model will be detailed.

Chapter 4

The SUMO Microscopic Traffic Simulator

Simulation of Urban MObility (SUMO) is a well-known 2D microscopic traffic simulator project almost in its 1.0 version. Although it serves its purpose effectively, SUMO does not provide proper mechanisms to integrate an autonomous vehicle in its own network model.

In this chapter, a brief contextualization to the SUMO software architecture is described, followed by all changes to the simulator required for the integration of autonomous vehicles in a traffic network. Also, a communication framework model to provide connection facilities to the robotics simulator is specified. Afterwards, a more comprehensive discussion on how the reference network from Aliados was imported into SUMO is pinpointed, summing up with the implementation of a simple client to test the aforementioned changes. A comprehensive description on the SUMO network file descriptors can be found in appendix A.

4.1 Simulation of Urban MObility - a brief description

SUMO simulator is perhaps the most scrutinized microscopic traffic simulator in the research community, with hundreds of scientific papers referring to it. This project started in the year 2000 with a need of an open-source tool into which several algorithms can be implemented and evaluated, such as road networks, demand and traffic controls. Figure 4.1 illustrates a typical simulation scenario on SUMO. Its main announced features (adapted from [79]) are:

1. Implemented in standard C++;
2. Cross-platform;
3. Includes all applications needed to prepare and perform a traffic simulation (network and routes import, dynamic user assignment (DUA), simulation);
4. Simulation:
    - Space-continuous and time-discrete vehicle movement;
    - Manages different vehicle types;

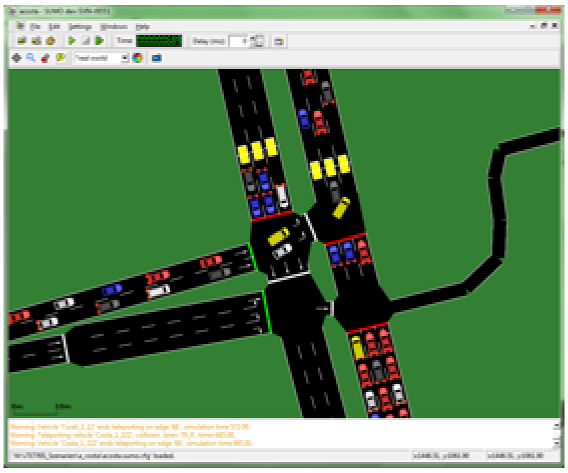


Figure 4.1: A screenshot of the open-source Simulation of Urban Mobility (SUMO) traffic simulator

- Multi-lane streets with lane changing;
- Different right-of-way rules, traffic lights;
- Fast openGL graphical user interface;
- Manages networks with several 10.000 edges (streets);
- Fast execution speed (e.g. up to 100.000 vehicle updates/s on a 1GHz machine);
- Interoperability with other application at run-time (using TraCI interface [1]);
- Network-wide, edge-based, vehicle-based, and detector-based outputs;
- GUI and command-line based simulation;

5. Network Import:
   - Imports VISUM, Vissim, Shapefiles, OpenStreetMaps, RoboCup, and XML-Descriptions;
   - Missing values are determined via heuristics;
6. Routing:
   - Microscopic routes - each vehicle has its own;
   - Different DUA algorithms;
7. High portability
   - Only standard C++ and portable libraries are used;
   - Availiable packages for Windows and Linux distributions;
8. High interoperability through usage of XML-data only;
9. Open source (GPL);

SUMO is a complex project with several contributors, and consists of more than 200.000 lines of code, distributed in around 450 classes. Given the limited scope of the project, a brief

[1]TraCI is an acronym for "Traffic Control Interface". It consists on a network protocol which gives external applications access to a traffic simulation running on SUMO

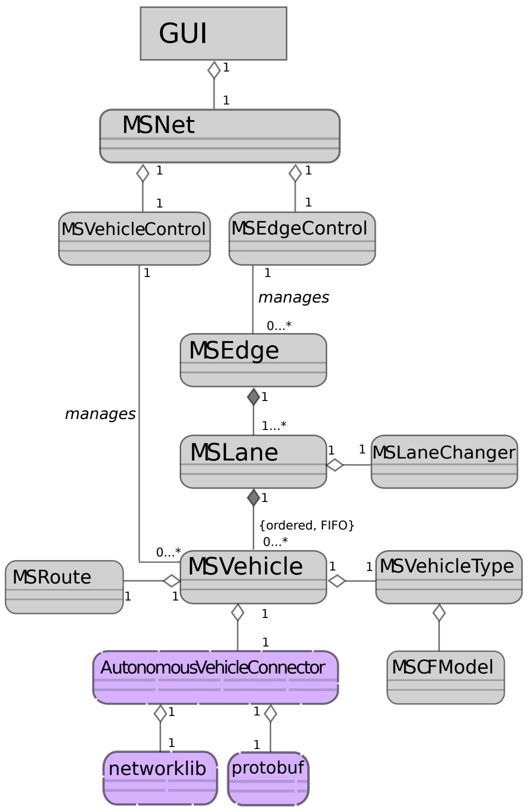


Figure 4.2: A general overview of SUMO's microscopic simulation module implementation[1]

description on SUMO simulation files and microsimulation architecture is presented in the two following sections respectively, to familiarize the reader with the platform.

4.2 SUMO Microsimulation Architecture

SUMO comprehends an efficient and flexible microsimulation core, described by gray modules on the simplified diagram in Figure 4.2. As observed, the microsimulation architecture is formed by the following classes:

GUI Consists of several classes comprehending all the Graphical User Interface model, which controls the microsimulation parameters and deployment;

MSNet The simulated network and simulation performer. MSNet also contains all microsimulation related objects;

MSEdgeControl Stores and manages edges and lanes, and performs movement of vehicles;

MSVehicleControl The class responsible for building and deleting vehicles from simulation;

[1]Note: This is a simplified overview of SUMO's microsimulation classes, and therefore, not all classes are depicted in the diagram.

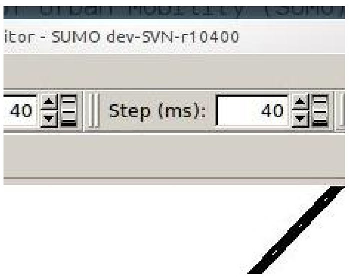


Figure 4.3: SUMO dynamic step spin-box

MSEdge A road/street connecting two junctions, consisting in one or more lanes;

MSLane Representation of a lane in the micro simulation;

MSLaneChanger Performs lane changing of vehicles (it is an abstract class);

MSVehicle Representation of a vehicle in the micro simulation, such as vehicle type, current speed, angle, lane, and so on;

MSVehicleType Comprehends vehicle parameters that can be generally associated with a real vehicle type (e.g. buses, trucks, cars), containing several parameters such as vehicle shape, emission class, car-following model, maximum speed, etc;

MSCFModel An abstract class to be implemented by a car-following model;

MSRoute A vehicle route description.

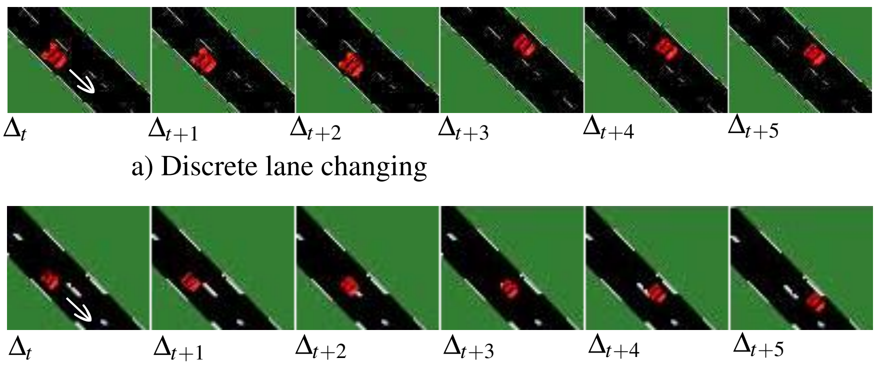
Having gained acquaintance with SUMO's general ideas, how it is implemented and its microsimulation structure, we are now ready to foster the required patches to accommodate a free vehicle within the traffic scene. The following section will introduce the problems and demonstrate its implemented solutions.

4.3 Selected Modifications

Some of the requirements SUMO platform lacks were issues already mentioned in Section 2.4.1. The most difficult step was to unbound the lane dependency of the vehicle, as SUMO represents each vehicle position relative to its current lane. In the next sections, a comprehensive description of such added features is presented.

4.3.1 Simulation Step

SUMO implements a space-continuous time-discrete simulation on its traffic flow models. Furthermore, two global variables in the simulation loop are implemented to provide flexibility in time step calculations:



a) Discrete lane changing

b) Continuous lane changing

Figure 4.4: SUMO vehicle lane-changing behavior before and after patching

DELTA_T Represents the delay between simulated time steps (formerly known as Δ);

mySimDelay Contains the delay *DELTA_T* should take in real time.

By default, SUMO only allows to change *mySimDelay* dynamically with a spin-box on the main window, and *DELTA_T* is hard-coded to 1000ms. This brings up an issue on the autonomous vehicle framework, as it does not respect the established minimum of 30Hz updating frame-rate. In order to perform a real-time simulation on sumo, both *DELTA_T* and *mySimDelay* must have the same value which cannot exceed a certain value. This means that *DELTA_T* must respect a maximum value of $1000/30 \approx 33ms$ time step.

To overcome this drawback a new spin-box was added to the main interface, to allow a dynamic control over the value of *DELTA_T*. This way, user can select whether he wants a real-time simulation ($DELTA\_T = mySimDelay$), accelerated ($DELTA\_T > mySimDelay$) or slowed down ($DELTA\_T < mySimDelay$) simulation on whichever frame-rate. Figure 4.3 depicts the main window modification to control *DELTA_T* value.

4.3.2 Continuous Lane Changing

SUMO implements a discrete lane-changing model as illustrated in Figure 4.4a. When a driver/vehicle intends to change its lane, e.g. to overtake another car or when approaching an intersection, it instantaneously disappears from the former lane to appear on the desired lane. However, this brings an undesired effect with same consequences of Section 4.3.1, visually perceived as a lack of smoothness.

A continuous lane-changing approach was implemented on SUMO to overcome this undesirable effect. Furthermore, a class named *laneChangeAnimation* was implemented as a nested class[1] over *MSVehicle* (which manages vehicle microsimulation parameters in SUMO), to "animate" the translation between lane changes. This translation movement is linear and performed

[1] A nested class is a class declared inside the scope of another class.

in one second (simulation time). Although not entirely realistic, the implementation is very efficient thus not adding any overhead onto SUMO simulator. Figure 4.4a depicts the former discrete vehicle movement whereas 4.4b illustrates the new vehicle movement aspect after the patch was implemented.

4.3.3 Vehicle-Edge Decoupling

Microscopic Traffic Simulation presupposes a correspondence between each vehicle and its lane containing it. SUMO Vehicles also follows this rule, and therefore, in order to correctly simulate an autonomous vehicle, it must be detached from any lane, and be able to freely select its movements, i.e. should be able to navigate through traffic's scene cartesian space.

To address this issue, a new variable named *outsideRoadPosition* was created in *MSVehicle* representing the vehicle's absolute position in the scene. Also, the vehicle angle was completely decoupled from its current lane angle. Using this lane-independent values, we can now render the autonomous vehicle on the screen.

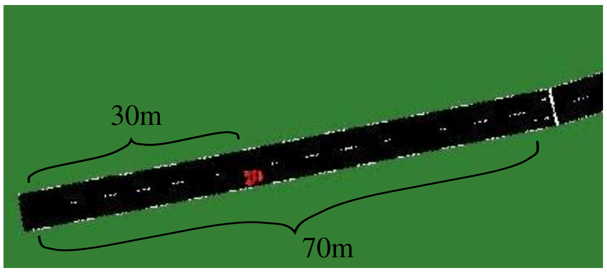
To maintain the remaining SUMO vehicles aware of this new vehicle type, in each time step, the autonomous vehicle entity verifies if it entered or exited a lane using its lane-vehicle shapes intersection. Despite its complexity, this operation is fairly efficient, as the OpenGL graphics rendering framework underneath SUMO maintains an indexed list of objects on the scene to be picked up. Meanwhile, the queue containing all vehicles in the lane is updated and sorted, so that all vehicles in it are aware of the autonomous vehicle.

The following Algorithm 1 states the autonomous vehicle-lane detection mechanism:

Algorithm 1 The autonomous vehicle-lane detection mechanism

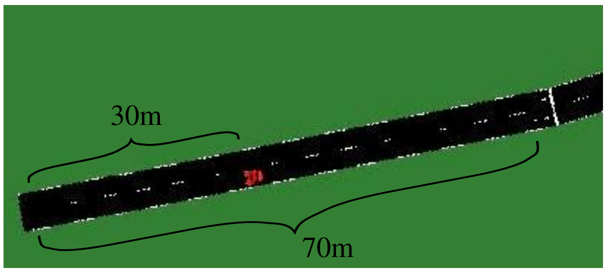
```
$detectedLane \leftarrow detectLaneOnPos(vehicle.myPos)$
if $(vehicle.myLane \neq detectedLane)$ then
  if $(vehicle.myLane \neq null)$ then
    {vehicle exited lane}
    $vehicle.myLane.removeVehicle(vehicle)$
    $vehicle.myLane \leftarrow null$
  else
    {vehicle not in a lane}
  end if
  if $(detectedLane \neq null)$ then
    {vehicle entered a lane}
    $vehicle.myLane \leftarrow detectedLane$
    $detectedLane.insertVehicle(vehicle)$
    $detectedLane.sortVehicleList()$
  else
    {vehicle outside lanes}
  end if
end if
```

a)



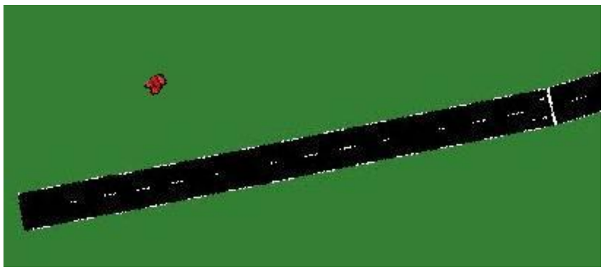
$$\begin{cases} vehicle.myPos & = & 30m \\ vehicle.myAngle & = & lane.myAngle \\ vehicle.myLane & = & lane \end{cases}$$

b)



$$\begin{cases} vehicle.myPos & = & 30m \\ vehicle.outsideRoadPosition & = & (x_1, y_1) \\ vehicle.myAngle & = & \alpha \\ vehicle.myLane & = & vehicle \end{cases}$$

c)



$$\begin{cases} vehicle.myPos & = & \text{undefined} \\ vehicle.outsideRoadPosition & = & (x_2, y_2) \\ vehicle.myAngle & = & \beta \\ vehicle.myLane & = & null \end{cases}$$

Figure 4.5: SUMO coupling/decoupling variable states before (a) and after (b,c) patching

Figure 4.5 depicts hypothetical situations of SUMO vehicle states before (a) and after (b,c) the patch has been applied.

4.3.4 Calculate Surrounding Vehicles

To determine the surrounding vehicles over a studied area (around the autonomous vehicle position in our case), an efficient method was critical as it had to respect the simulation timing requirements. Moreover, as SUMO uses OpenGL to render its traffic scene, the OpenGL picking[1] feature allows to render the scene (thus not displaying it), to further pick the objects that were drawn in the rendered area. This method is already used in SUMO to allow the user to select an item in the scene with the right-click of a mouse. Furthermore, a new method to pick all vehicles in a certain area was implemented, and 50x50m was determined to be satisfactory to our results, accounting for the field of view of sensors in the autonomous vehicle.

4.3.5 Vehicle Parameterization, Spawning and Simulation

After meeting all requirements to simulate an autonomous vehicle within a classic microscopic traffic simulator, SUMO in this case, a mechanism to parameterize and spawn these vehicles must be created for a practitioner to have flexibility in selecting and positioning a certain number of the aforementioned entities into the traffic scene. Afterwards, these vehicles must be simulated within cartesian space, despite former vehicles move on a lane-to-lane basis.

SUMO simulator uses a XML file descriptor with extension *.rou.xml (detailed in Appendix A), which manages the traffic demand from one source node to end node with two major traffic demand models, namely the distribution-based demand, and individual vehicle based.

It is evident the individual demand is essential to customly spawn an autonomous vehicle into the traffic scene, as we need to know explicitly every autonomous vehicle instance created. A sample line from *example.rou.xml* that spawns a vehicle is defined in Listing 4.1 below.

Listing 4.1: A sample description for a vehicle instance on example.rou.xml.

```xml
<vehicle id="veh1" route="route01" type="CarA" color="1,0,0" depart="0010"/>
```

To account for the autonomous vehicle integration with an external client, some new attributes are needed to complement this new vehicle type:

type	Associate "autonomous" to this new vehicle type.
listenport	Port which the autonomous vehicle is listening to.
externalid	External client vehicle ID.

Furthermore, the following line would represent a definition of an autonomous vehicle for SUMO in *example.rou.xml*. This vehicle is listening to port 7890 at depart time 0010, with color red, using "autonomous1_usar" id in its external client (in our case, USARSim simulator client). Listing 4.2 illustrated the aforementioned modification.

[1]Picking is the task of determining which screen-rendered object a user has clicked on.

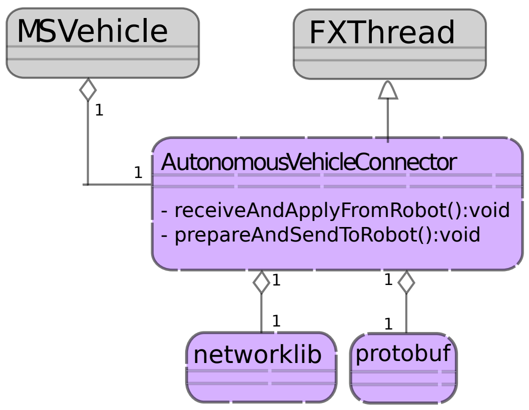

MSVehicle
FXThread
AutonomousVehicleConnector
- receiveAndApplyFromRobot():void
- prepareAndSendToRobot():void
networklib
protobuf
1
1
1
1
1
1

Figure 4.6: The implemented *AutonomousVehicleConnector* has networklib and protocol buffers as its dependencies.

Listing 4.2: A vehicle instance accounting for an external client.

```
<vehicle id="av1" listenport="7890" externalid="autonomous1_usar"
                    type="autonomous" color="1,0,0" depart="0010"/>
```

On every autonomous vehicle spawned by SUMO a warning message with its listening port is displayed in the main window. One must note that once the vehicle is spawned it becomes part of SUMO's lane infrastructure, therefore its surrounding cars movements are determined by it.

We have our vehicle ready to be rendered on SUMO traffic scene. The final step is to write the equations of motion (4.1 and 4.2), accounting for the vehicles" dependencies on its current angle and speed. This equation is processed once in each simulation step.

$$myPos.x = myPos.x + mySpeed \times \frac{1}{DELTA\_T} \times \cos(myAngle) \tag{4.1}$$

$$myPos.y = myPos.y + mySpeed \times \frac{1}{DELTA\_T} \times \sin(myAngle) \tag{4.2}$$

One must note that vehicle dynamics is strictly dependent on the external client behavior, and an abrupt modification of vehicles speed or angle would result on undesired behavior.

4.4 Communication Infrastructure

Bearing in mind the discussion in section above, when an autonomous vehicle is placed in the SUMO simulation environment, a communication channel must be opened with an external client, that will control it. Thus, a communication class named *AutonomousVehicleConnector* was implemented on SUMO to provide such feature (illustrated in Figure 4.6).

When an autonomous vehicle is created, *AutonomousVehicleConnector* is also instantiated, binding a socket and starting to warn the user referring the port the connection is listening to.

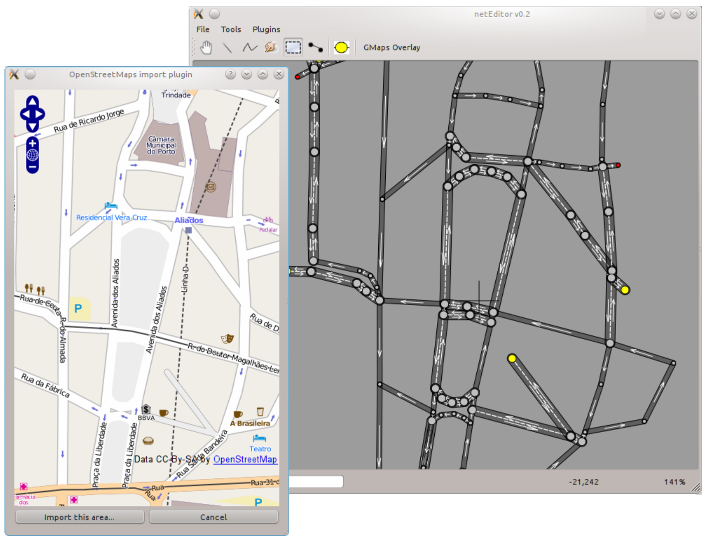


Figure 4.7: netEditor after importing the Aliados network with the OpenStreetMaps plug-in.

AutonomousVehicleConnector is a child of *FXThread* to inherit the threaded ability, integrating itself parallelly to SUMO without disrupting any existing functionality.

Initially, this class was using Protocol Buffers as its serialized transport protocol towards a more efficient and low-latency channel. However, further research on the USARSim platform demonstrated the impossibility to implement such feature (refer to Section 5.3.2 for further details). Furthermore, a communication protocol was devised to couple both SUMO and USARSim simulators, using *ASCII* characters. In Section 5.3.2, this protocol is analyzed more in more detail.

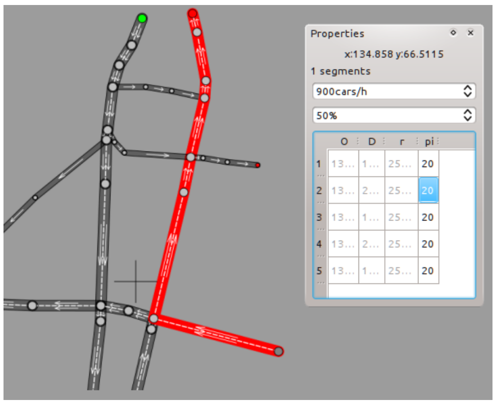
Besides the unavailability of using Protocol Buffers in the USARSim platform, in the following section a simple SUMO external client is presented, using protocol buffers as its serialization layer in order to account for the simplicity and the potential of such a tool. Diagram 4.6 depicts the *AutonomousVehicleConnector* class and its dependencies.

4.5 Importing the Reference Network

To import our Aliados reference network (already presented in Section 3.3.1) into SUMO, the netEditor [76] traffic editor was used to model it.

netEditor is an open-source traffic network modeling tool devised on the concept of plug-ins to support the addition of import/export interfaces in a collaborative environment. It was modeled and implemented by the author, when on an FCT BII Undergraduate Scholarship. Moreover, it comprehends import plug-ins from other Geographical Information System (GIS) sources, such as from OpenStreetMaps, or shape files, and features exporting to the SUMO simulator as well. Therefore, some steps are needed to successfully import Aliados into SUMO:

Import Aliados topology to netEditor To import Aliados, the OpenStreetMaps plugin is opened and the network is selected by panning into the intended region. Figure 4.7 depicts the


Properties

x:134.858 y:66.5115

1 segments

900cars/h

50%

	O	D	r	pi
1	13…	1…	25…	20
2	13…	2…	25…	20
3	13…	1…	25…	20
4	13…	2…	25…	20
5	13…	1…	25…	20

Figure 4.8: netEditor traffic demand modeling.

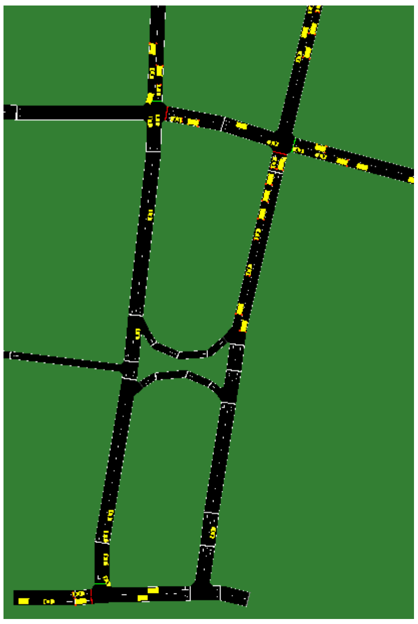
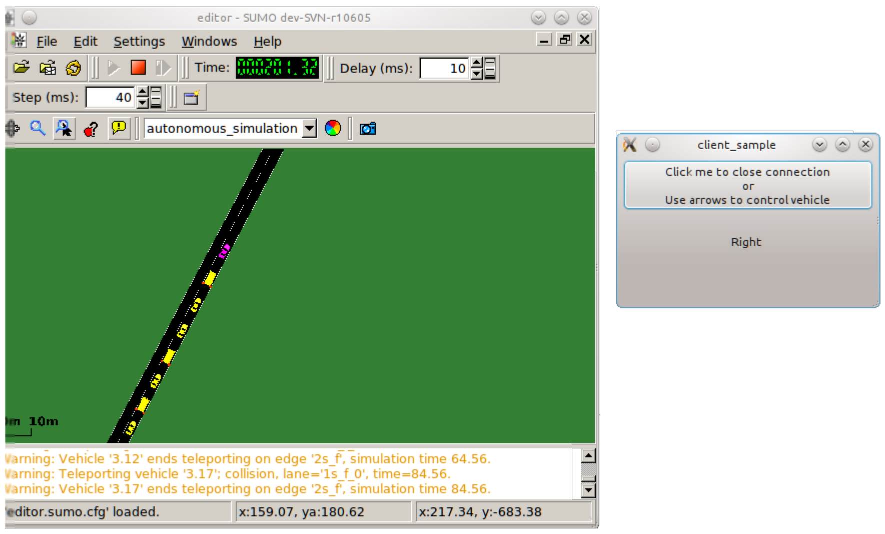


Figure 4.9: Aliados simulation close-up on SUMO being modeled in netEditor.


editor - SUMO dev-SVN-r10605
File Edit Settings Windows Help
Time:
Delay (ms): 10
Step (ms): 40
autonomous_simulation
10m
client_sample
Click me to close connection
or
Use arrows to control vehicle
Right

```
Warning: Vehicle '3.12' ends teleporting on edge '2s_f', simulation time 64.56.
Warning: Teleporting vehicle '3.17'; collision, lane='1s_f_0', time=84.56.
Warning: Vehicle '3.17' ends teleporting on edge '2s_f', simulation time 84.56.
```

```
'editor.sumo.cfg' loaded.    x:159.07, ya:180.62    x:217.34, y:-683.38
```

Figure 4.10: SUMO autonomous vehicle (in Purple color) connected to a simple client.

network on netEditor after importing;

Fix bad topology OpenStreetMaps does not have coherent network information about the number and direction of some lanes in the Aliados area. netEditor provides the appropriate tools to manually change the lane information, and using the real information from satellite photos (e.g. from Google Maps) this changes can be easily applied;

Setup vehicle demand on start nodes The netEditor modeler automatically generates traffic demand flows from each start node[1], although its information is useless when a real topology is applied. However, as no information about traffic flows was available, demand was generated empirically, on the basis of practitioners' experience and previous studies. Moreover, a higher flow was given to the two main Aliados roads (the two-way vertical roads). The netEditor comprehends a widget allowing to select routing-based information, such as the probability a vehicle will follow a specific route. Figure 4.8 depicts the demand modeling phase of the Aliados network.

Export the network topology and demand Having the network prepared as well as its trip demand, we are ready to start a simulation on SUMO. The netEditor will generate the appropriate files to perform it successfully (refer to Appendix A for information on SUMO network files).

Figure 4.9 depicts the simulation appearance of our exported network.

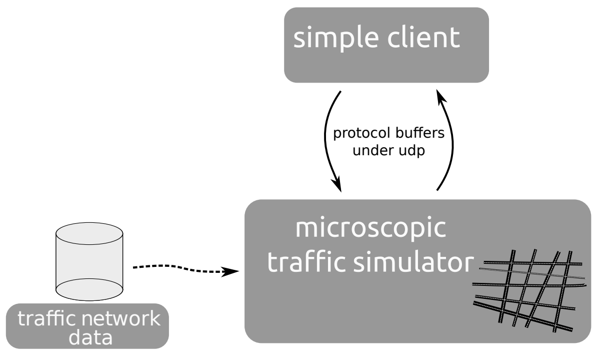


Figure 4.11: The implemented SUMO $<-->$ Simple Client test software architecture.

4.6 A Simple Client

When researching and adapting SUMO to support autonomous vehicles in its traffic scene, a simple client to connect the vehicle was mandatory. This simple application provides control over a connected vehicle through arrow keys from the computer keyboard. It has a simple push button, and a label stating the key pressed at the moment. The application was coded in C++ along with the Nokia Qt4 framework [80] to design its GUI and Networking, and Google Protocol Buffers as the basis for data serialization. Figure 4.10 illustrates the simple client aspect whereas Figure 4.11 pinpoints its high-level architecture.

Protocol buffers uses a simple **.proto* file extension with the specification of the serializable data (in the form of messages), using an object-oriented approach. After defining the messages, *protoc* command-line application provides both .h and .cpp files that manage the serialization process, which are included on our application. Figure 4.12 depicts the needed steps to integrate a protocol buffer specification in our project. Listing 4.3 presents an example *proto* file, illustrating the simple configuration of the protocol.

Listing 4.3: Example *test.proto* file descriptor

```
message Vehicle {
        required string id=1;
        optional float speed=2;
        optional float angle=3;
        optional float x=4;
        optional float y=5;
```

[1]A start node is a node where vehicles are spawned whereas an end node removes them from the simulation.

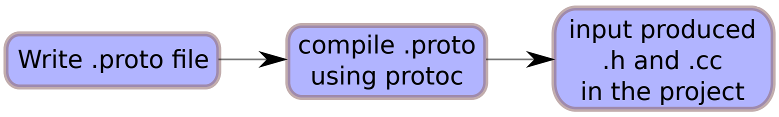

Write .proto file
compile .proto using protoc
input produced .h and .cc in the project

Figure 4.12: Simple steps to produce a protocol buffer specification in the project.

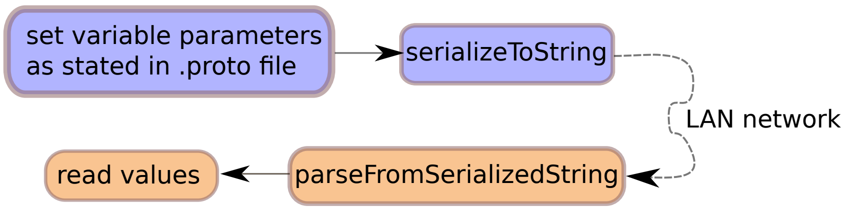

set variable parameters as stated in .proto file
serializeToString
LAN network
read values
parseFromSerializedString

Figure 4.13: The protocol buffers straight-forward serialization process.

```
}

message RobotToTraffic {
    optional int32 timeStep = 1;
    optional Vehicle autonomous = 2;
}

message TrafficToRobot {
    optional int32 timeStep = 1;
    repeated Vehicle neighbourCars = 2;
}
```

The **optional** and **required** parameters states whether a variable is mandatory to be assigned to a value. If not, the transport protocol will decrease the packet size for each parameter omitted. The **required** parameter states that variable as an array.

After SUMO and the simple client connect to each other, SUMO gathers information about the vehicles' surroundings including other vehicles, inserts each one in a *Vehicle* message within the array *neighbourCars*, serializes the global message, and finally sends it through the networklib[1].

The simple client receives SUMO's values and represent them on the console (as no vehicle rendering is implemented in the prototype). After that, the angle and speed gathered from the arrow keys is inserted in a message, serialized and sent back to SUMO. This transactions should occur in a single simulated time step.

Note that to minimize bandwidth throughout the network, the (x_v, y_v) values are only sent at a 100 time steps rate (as a calibration measure), whereas the speed and angle values are only transmitted if its value has changed.

Given the processing power of today's computers, the major bottleneck to connect several clients to SUMO is the connection bandwidth. However, as we take into account the threading and Protocol Buffers mechanism, hundreds to thousands of autonomous vehicles might be able to be controlled externally in SUMO.

Figure 4.13 depicts the serialization process from the simple client to SUMO.

[1] networklib is a designation to SUMO's bundled network library

The simple client and SUMO modifications were presented to SUMO developers,who have kindly blog posted the project[1]. Another video of a created network from netEditor to SUMO can be watched on Youtube[2] as well.

4.7 Summary

This chapter proposed a modification of SUMO to support it with a general purpose robotics simulator.

The potential concerning the integration of a SUMO vehicle to an external control client is evident. Although SUMO is already in an high matury level regarding its feature-set and code organization and documentation, a lot of changes were made to allow it to support an autonomous vehicle.

In the next chapter, a brief introduction to the USARSim robotics simulator is presented, as well as its modifications prior to its integration with a traffic simulation, the SUMO simulator, in the case of this project.

[1] http://www.youtube.com/watch?v=KgPSREMmA_0

[2] http://sourceforge.net/apps/wordpress/sumo/2011/05/19/autonomous-vehicle-by-jose-pereira/

Chapter 5

The USARSim Robotics Simulator

Following the proposed selection presented in Chapter 3, this chapter introduces the USARSim robotics simulator and exposes the intended modifications which should fulfill the requirements towards its coupling with a microscopic traffic simulator, SUMO in this project's context. USARSim was the robotics simulator selected earlier on Section 3.2.1, which seams the best suitable platform to be used in the integration platform. Moreover, a initial description of the USARSim software architecture and technical aspects are initially reviewed, followed by a discussions on the implemented extensions.

5.1 The USARSim Platform - a brief description

USARSim, an acronym for Unified System for Automation and Robot Simulation, is a 3D simulation environment research tool developed by the National Institute of Standards and Technology to be a test bed for a standard Test Course for Urban Search and Rescue Robots [81]. USARSim is used by the research community all over the world, and it has been extended to support a variety of robots, such as humanoids, submarines, helicopters, ackermann-based, and so forth. It is considered a low cost high fidelity simulator as it is built on top of a commercial platform, which provides high quality visual rendering and physics modeling. USARSim simulates a handful of sensor types such as Range Scanners, IR, Camera, Inertial Measurements, with noise generation, making it flexible for many robot applications.

USARSim uses the Unreal Engine 3 (UE3) as its underlying game engine in its current version, although its stable version is based on Unreal Engine 2. These two software engines are not compatible with each other, meaning that a careful step must be taken when selecting the USARSim version, having into account the requirements of this project.

From now on, lets consider two versions of the USARSim simulator, namely the UE2 and UE3 versions for this document, i.e. the stable and the unstable releases. USARSim source consists of a set of script files (written in Unreal Script), configuration files and provides various sample configuration robots, such as ackermann-based, humanoid-based, underwater-based and others.

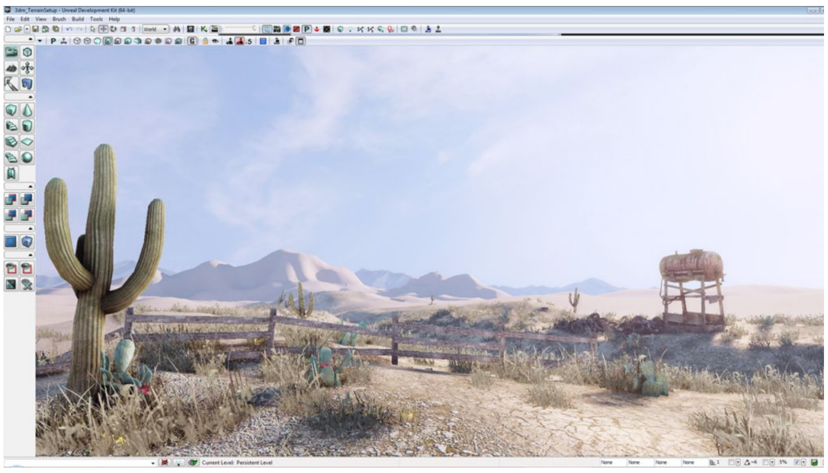


Figure 5.1: Unreal Development Kit scenario represents a state-of-the-art rendering environment, with advanced shadow and lighting techniques

In the following sections, a brief introduction of USARSim and Unreal is described, with a particular focus on Ackermann[1] steered vehicles. The reader is also encouraged to read the USARSim reference documentation for a comprehensive and detailed on the USARSim system architecture [82].

5.1.1 The Unreal Engine^{TM}

The Unreal Engine is a game engine developed by Epic Games, already in its third edition. It is written in C++, although to use it with a free non-business license, developers are restricted to the proprietary Unreal Script programming language. The Unreal Engine is a state-of-the-art game engine using a client-server based networking model, featuring all kinds of modules to facilitate game development, such as Rendering, Physics, Audio, Logic, Networking, Artificial Intelligence and Input Abstraction (as mentioned in section 2.1).

Bundled within several games, Unreal Engine had its first version in 1998, being one of today's most successful game engines of the industry. More recently, the Unreal Development Kit was launched as a gaming SDK allowing all kinds of business to use it without paying surreal amounts of money. Instead, Epic charges its product using game-consumer based fee rates, i.e. for non-consumer products the Unreal Engine can be used free of charge, whereas its price will vary according to the effective number of game purchases.

In order to use USARSim UE2 and UE3 editions, one must purchase an Unreal Tournament 2004 and Unreal Tournament 3 licenses, respectively. However, a development version of USAR-Sim on top of UDK is currently being developed, which will allow researchers to use it free of charge.

[1]An Ackermann-steered vehicle is a four-wheeled vehicle which solves the difference of angles between the two front steering wheels during vehicle turning.

Figure 5.1 depicts the UDK editor, a full-featured Unreal game editor to create and modify scenes, characters and model their interactions.

5.1.2 The Unreal Script

Unreal Script is an object-oriented scripting language used in Unreal Engine game code. It follows the principals of Java, having a garbage collector, automatic support for meta-data and profiling, and compiles to byte-code to be run in Unreal Virtual Machine.

The Unreal Script supports backwards-compatibility with its older versions, although as each Unreal Engine uses a different physics engine (Karma engine in UE2, Physx in UE3), some modifications are crucial when porting an older script.

The Unreal Script was created as a way for game developers to modify their games easier and faster, which was a crucial aspect of the Unreal Engine longevity.

5.1.3 The Unreal Editor

Unreal Editor is a tool bundled with the Unreal Engine , which is used for managing Players and World configurations, such as import models from other 3D modeling tools, and place items (such as objects, walls, crates, sky domes) or scene options (lighting, player starts, etc). In this project, we need to deal with environment and vehicle modeling, thus Unreal Editor will be a crucial tool to its accomplishment. The Unreal Editor appearance can be visualized in Figure 5.1.

5.1.4 USARSim *.ini files

There are several configuration files for Unreal Engine, with the extension *.ini. This files allow to input default values for the Unreal Scripts without having to recompile them, being useful to retain specific sensor parameters, such as the input noise of an inertial sensor or which sensors has a robot. The most important file to have in consideration in this project is the *USARBot.ini*, as it manages the robots configuration, and allows to instantiate the implemented sensors (and its parameters) for each robot.

5.1.5 3D Models in Unreal Engine

Unreal manages different types of models according to its purpose, which will be rendered in the simulation environment. These models can be static (e.g. walls, buildings, crates), dynamic (humans, vehicles) or with multiple definitions, such as a weapon, with a first-person view, world view and opponent view. As for this project, we need to establish a model for a world map (see Section 3.3.1, our reference network is in Aliados zone, in Porto, Portugal), and a vehicle to model the autonomous vehicle (also defined in Section 3.3.2).

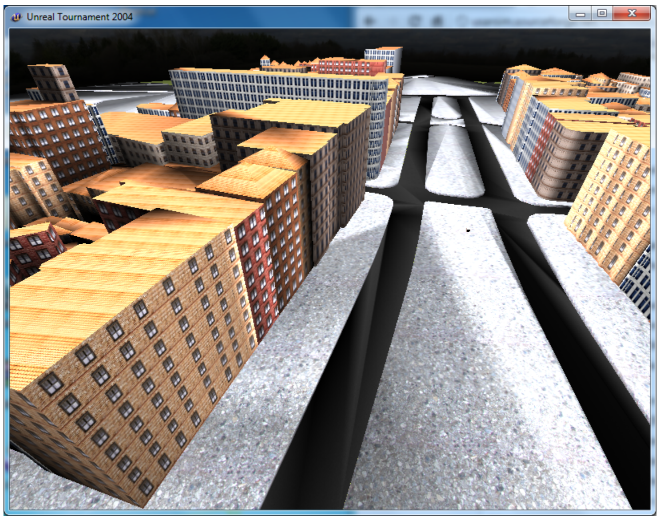


Figure 5.2: Aliados zone rendered in USARSim UE2

5.2 Selected Modifications

This section is focused on the implementation of the Aliados Zone and an autonomous vehicle robot in the USARSim simulator. The Aliados model consists in modifying the COLLADA file description to be imported by the Unreal Editor. The autonomous vehicle was based on a pre-existing model, and was modified to meet the requirements of this project. Finally, the communication infrastructure developed to be connected by an external traffic simulator is documented.

5.2.1 Environment Modeling

One of the most important questions while brainstorming this project and conceiving its aim and goals, was to assess the practicalness of building a realistic traffic network to perform a simulation of this kind. Moreover, having met an in-house software developed by PhD student Pedro Silva [77], which uses procedural modeling[1] techniques for creating virtual urban environments, it seemed feasible to experiment it to model our Aliados Zone. This project uses a GIS database containing topology information of Porto city, and models a realistic 3D scene to be used, e.g. on video games technology. The file format used to export the model is the COLLADA[2] [83] open format.

After exporting the Aliados 3D model from the aforementioned application, it needed to be converted into a supported file format of the Unreal Editor 2 as COLLADA was not supported. Therefore, the 3ds Max modeler application from Autodesk [78] provided the appropriate tools to export this model to the ASCII Scene Exporter (ASE) file format, supported by Unreal Editor. Along with the 3D model, six textures were used to enhance the walls, roads and roofs, of the 3D

[1]Procedural Modeling is the designation for a set of techniques applied in computer graphics to generate rule-based 3D models and textures.

[2]COLLADA (acronym for COLLAborative Design Activity) is an open standard XML file format for interactive 3D applications.

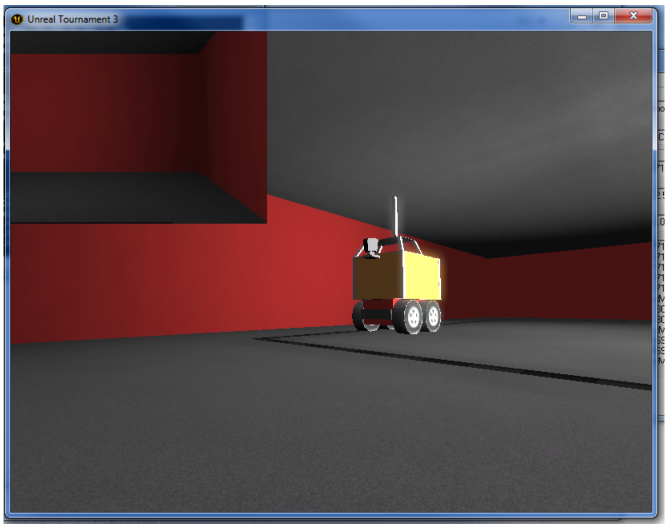


Figure 5.3: A test robot scenario in USARSim UE3

model and properly assigned to each mesh. They were also resized to a power of 2 (e.g. 256x256) and converted to the bitmap image format (*.bmp) to be compatible with the Unreal Editor.

In game engine technology, object models are also common to be provided with a collision model, i.e. a simplified mesh for handling collisions between objects therefore simplifying the global amount of calculations. Nevertheless, the COLLADA file did not contain this mesh, which was properly created in the 3ds Max. In this way, the autonomous vehicles' sensors would be properly aware of the surroundings.

Having the Aliados model in Unreal Editor, the environmental lightings were set up to keep a daily aspect in the environment, and a skybox was also placed around the model. Figure 5.2 depicts the final model of Aliados, rendered in USARSim UE2.

5.2.2 Vehicle Modeling

To maintain a realistic aspect of the simulation, a 3D model of a ground vehicle was essential. However, as of 2011 the USARSim UE3 version did not contain a modeled example of a consumer vehicle and the workload to create one from scratch was far beyond this thesis scope. Nevertheless, this was one of the main reasons to adopt the USARSim UE2 version on this project. However, the USARSim UE3 was also an object of study, as the Unreal Engine 3 features novel rendering techniques such as high dynamic range rendering, per-pixel lighting, dynamic shadows and more. Figure 5.3 depicts the aspect of a custom scenario built to test the implemented control agent on USARSim UE3.

As already stated in Section 3.3.2, the reference vehicle to be simulated on the framework resembles the Google Autonomous Vehicle, a modified Toyota Prius with a set of commonly used

[1]In videogame technology, a skybox is a method of creating textured backgrounds to make the environment look larger than it really is.

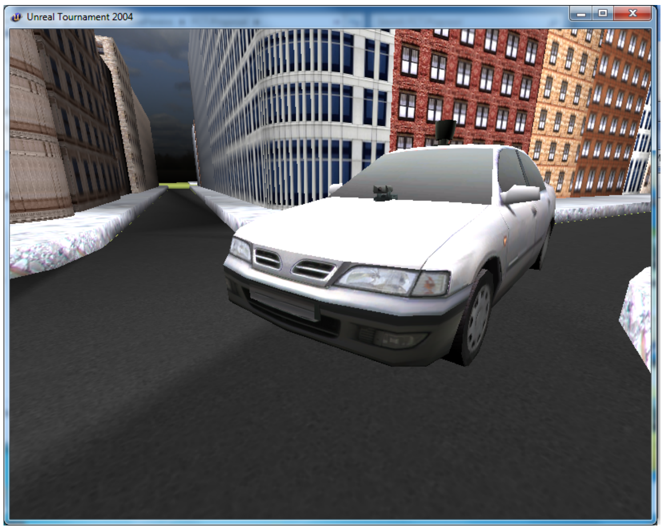


Figure 5.4: Sedan Vehicle rendered in USARSim UE2

sensors. Bearing in mind there was already an Ackermann-based ground vehicle implemented on USARSim UE2, it was a good starting point to add its sensors to the vehicle. The result aspect of the vehicle is illustrated in Figure 5.4. In this section, a comprehensive and detailed discussion on the vehicle setup is presented.

5.2.3 Vehicle Sensoring

One of the most important aspects to reflect about is vehicle sensoring. Similar to the five senses of the human kind, robots also need a set of receptors to be able to know where they are, and what is its surrounding environment. To be able to accomplish this rather complex task, Simultaneous Localization and Mapping (SLAM)[84] techniques are applied to the sensing data in order to probabilistically locate and situate the robot in its environment.

To devise our Ackermann-steered robot in USARSim, the *USARBot.ini* file was modified so as to account for the vehicles sensor topology (refer to Section 2.3.4 for each sensor description). As a form of example, Listing 5.1 presents a Lidar Range Scanner instantiation.

Listing 5.1: Example sensor instantiation in *USARSim.ini*

```
[USARBot.Sedan]
(...)
Sensors=(ItemClass=class'USARModels.Lidar',ItemName="Scanner2",
                                Position=(Y=0.0,X=0.06,Z=−1.3),Direction=(x=0,y=0,z=0))
(...)
```

From the above frame we can conclude it is rather simple to insert a robotic sensor, by configuring the sensor type, name, position and direction vector. A brief description of the applied sensors and their implementation details are listed below:

LIDAR Comprehends full 360^o horizontal field of view, and $\approx 20^o$ vertical field of view retrieving a 3D point cloud range data. A 3D model based on Velodyne LIDAR was designed in 3ds Max, and consequently exported to the Unreal Editor (it can be seen in vehicles rooftop);

Odometer The simulated odometer uses the vehicles's front left and right wheel encoders to estimate the robot's position. It returns the estimated position in the Unreal scene coordinates;

Inertial Motion Sensor Measures the current vehicle linear acceleration and angular velocity to estimate the current vehicle position and orientation;

GPS receiver As only a limited part of terrain can be simulated, a reference GPS position must be set on the scene. It can be set up either in *USARBot.ini* configuration file or placing a *ReferenceGPSCoordinate* item in the scene through the Unreal Editor. The GPS receiver will return a simulated Latitude and Longitude (converted from meters in the scene), a fixed value, whether it acquired or not its position, and current tracked satellites;

Video Camera Given the limited access to the core of Unreal Engine, the camera framebuffer cannot be accessed from its scripting language. Furthermore, a Dynamic-link Library (DLL) denominated Hook.dll is provided with USARSim which hooks itself to the Unreal executable and transmits camera frames using a client/server architecture. The DLL starts to grab the OpenGL/DirectX framebuffer pixels and sends them to its clients, either in raw or jpeg compressed formats.

A more comprehensive detail on the sensor information protocol used to communicate with the control agent is presented in the following section. The interested reader is also encouraged to read the USARSim documentation for more details [82].

5.3 Communication Infrastructure

Bearing in mind the Unreal Engine limited core access, there is no default protocol that allow characters in the game to be controlled via network sockets connected to other programs. Therefore, the Gamebots project [85] started at the University of Southern California's Information Sciences Institute aims to address this issue providing a custom protocol allowing modification of the robots, sensors, as well as world characterization states.

As represented on the projects integration architecture (Section 3.2.2), the USARSim communication infrastructure is two-fold: one connection is made with the control agents and the other with the SUMO traffic simulator, which are approached in the following sections.

5.3.1 Control Agent Communication

The connection to the agent controller is held using the common network infrastructure for robot control provided with USARSim, which can be consulted in its reference documentation [82]. It consists in sending high level messages to USARSim to spawn and control the robot, and receiving consistent data from its implemented sensors through Gamebots.

5.3.2 Traffic Simulator Communication

According to the projects design overview (as presented in Chapter 3), USARSim simulates all the autonomous vehicles in the scene whilst SUMO manages and simulates the remaining traffic vehicles. Moreover, SUMO needs to know the autonomous vehicle position and rotation in each step to replicate it on the traffic scene.

To connect USARSim to the traffic simulator, a custom modification of Gamebots was necessary to allow SUMO ask for a vehicle position in USARSim. The following message depicts an example command from SUMO to USARSim:

```
GETLOCATION {Name Autonomous1}
```

With *Name* meaning the autonomous vehicle id in USARSim, the same as *external_id* in SUMO network file description. USARSim should answer as follows:

{Location x,y,z} {Rotation α}

With *Location* meaning the vehicles absolute position and *Rotation* its α angle related to the Z-axis of USARSim's coordinate system. Note that this transaction is always referred to as the current step. No network delay is assumed, i.e. it is considered that the network connection is fast enough to deliver the information in time.

We must bear in mind that to increment the efficiency of the operation, the network protocol should be serialized and asynchronous as stated for the initial solution design. However, the limited access to the Unreal core has constrained the possibilities of such implementation in this project.

On the other hand, USARSim needs to mirror SUMO's surrounding vehicles. The USARSim implementation provides a class named *WordController* [86] aiming to create,move, animate and delete objects dynamically in the simulation, which proves feasible to tackle the aforementioned problem. Initially, this object must be instantiated issuing the following command using the Gamebots protocol:

INIT {ClassName USARBot.WorldController} {Name WC} {Location x,y,z}

Where x,y,z is any location in the map, typically near the ceiling. The *WorldController* does not interfere with the simulation, although it must be put on the scene and it is typically identified by a little cube mesh. Three methods are then used to control and position a surrounding vehicle:

```
CONTROL {Type Create} {ClassName USARModels.ComVeh} {Name VehX}
```

This creates a vehicle *VehX* with the mesh *USARModels.ComVeh*. Note that as we have flexibility to select any mesh from the Unreal database, a more realistic simulation would be achieved with a large vehicle list.

```
CONTROL {Type AbsMove} {Name VehX} {Location x,y,z} {Rotation 0,0,α}
```

The command above sets the *VehX* vehicle in the absolute position x,y,z with an angle α.

```
CONTROL {Type Kill} {Name VehX}
```

This removes the *VehX* vehicle from the simulation scene.

5.4 Summary

A practical solution to adapt USARSim for autonomous vehicles was presented, albeit some intended modifications were not as expected on the initial proposed design. Using serialized data to establish a communication between the two simulators would decrease the required bandwidth by a notorious amount. However, that was proven to be unfeasible giving the semi-closed nature of the Unreal Engine working as a basis for USARSim. However, the modeling of both the Aliados network and the autonomous vehicle was accomplished. In the following chapter, the integration of the SUMO and USARSim simulators will be discussed in detail.

Chapter 6

SUMO & USARSim Integration

This chapter comprehends the final road towards the coupling of both a traffic simulator and a robotics simulator, SUMO and USARSim respectively. Having addressed all particular issues in each simulator, a comprehensive analysis to their integration is performed, with focus on the practical usability of the platform and encountered difficulties.

A simple reactive agent was also modeled and implemented to validate the platform. It consists of a simple GUI showing information relative to vehicle sensors, and an automatic control. The simple agent is described later on in this chapter.

6.1 Surrounding Vehicles Integration

We are ready to manage all surrounding vehicle information from SUMO to USARSim. SUMO already calculates the surrounding vehicles (see Section 4.3.4) around an autonomous vehicle. Therefore, a control code to handle creation, moving and deletion of the former is presented above. It uses an hash-table comprehending vehicles already created on the scene, an calculates their next movements. Its pseudo-code is represented in Algorithm 2. *AbsMove(veh)*, *Create(veh)* and *Kill(veh)* correspond to the *WorldController* USARSim command definitions already discussed in Section 5.3.2.

6.2 Network Coherence Between Simulators

Having successfully imported the Aliados road network into SUMO using the netEditor (see Section 4.5) and to USARSim using the procedural modeling application (see Section 5.2.1), we are ready to implement the traffic network coherence and calibration method between simulators.

As both networks for the two simulators rely on different sources, albeit they both resemble the same physical zone of Aliados, in Porto city, their coordinate references are not the same. Moreover, to solve this issue a coordinate transformation method is used and presented in the following section, which should be feasible to transform each point on SUMO system coordinate to USARSim and vice-versa.

Algorithm 2 Surrounding vehicles control algorithm

```
$surroundingVehicles = myVehicle.getSurroundingVehicles()$
**for** $curVeh = surroundingVehicles.first() \rightarrow surroundingVehicles.last()$ **do**
    **if** $vehHashMap.exists(curVeh)$ **then**
        $AbsMove(curVeh)$
    **else**
        $Create(curVeh)$
        $AbsMove(curVeh)$
        $vehHashMap.insert(curVeh)$
    **end if**
    $curVeh \leftarrow curVeh + 1$
**end for**
**for** $curVeh = lastSurroundingVehicles.first() \rightarrow lastSurroundingVehicles.last()$ **do**
    **if** $!vehHashMap.exists(curVeh)$ **then**
        $Kill(curVeh)$
    **end if**
    $curVeh \leftarrow curVeh + 1$
**end for**
$lastSurroundingVehicles \leftarrow surroundingVehicles$
```

6.2.1 Coordinate System Transformation

Giving the fact that SUMO and USARSim use rather different coordinate systems, a coordinate system transformation method is used to map a point from one coordinate system to the other. Also, to calculate the transformation parameters (A matrix and b_1, b_2), the practitioner only will need to know the location of two points in both simulators coordinate system.

We must note that despite the 3D nature of the USARSim simulator, as the 3D model contains plain roads, the z value will always remain constant, therefore this transformation is $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$.

The result of applying a Euclidean transformation to a point $p(x,y)$ to obtain $p'(x',y')$ is given by the formula:

$$(x', y') = (x, y)A + b \tag{6.1}$$

where A is a 2X2 matrix and b is a pair of numbers that depend on the transformation, that is,

$$\begin{aligned} x' &= xA_{11} + yA_{21} + b_1 \\ y' &= xA_{12} + yA_{22} + b_2 \end{aligned} \tag{6.2}$$

If we modularize A and b_1, b_2 to scale, rotation and translation movements, we can infer that

$$A = S \times R \tag{6.3}$$

where S is the scaling matrix (diagonal), R a rotation matrix and (b_1, b_2) are the translation offsets in (x, y). Therefore we have:

$$S = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{6.4}$$

The scaling factor from p to p' coordinate system (s) will be given by:

$$s = \frac{||p'_2 - p'_1||}{||p_2 - p_1||} = \frac{\sqrt{(x'_2 - x'_1)^2 + (y'_2 - y'_1)^2}}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \tag{6.5}$$

and θ is the difference between the points' angles:

$$\theta = \alpha - \alpha'$$

$$\theta = \arctan\left(\frac{x_2 - x_1}{y_2 - y_1}\right) - \arctan\left(\frac{x'_2 - x'_1}{y'_2 - y'_1}\right) \tag{6.6}$$

To calculate (b_1, b_2), we grab a point p and its mapped p', calculate the A matrix and evaluate the following equations:

$$(\bar{x}, \bar{y}) = A(x, y) \tag{6.7}$$

$$\begin{aligned} b_1 &= \bar{x} - x' \\ b_2 &= \bar{y} - y' \end{aligned} \tag{6.8}$$

We can apply this method to both USARSim to SUMO coordinate transformation and vice-versa. We consider p_1^U and p_2^U reference points in the USARSim coordinate system, and p_1^S and p_2^S the same points in the SUMO coordinate system, and apply the aforementioned transformation to map from one simulator to another.

This coordinate transformations were implemented on SUMO simulator given its C++ native implementation and a standard library containing the trigonometric functions.

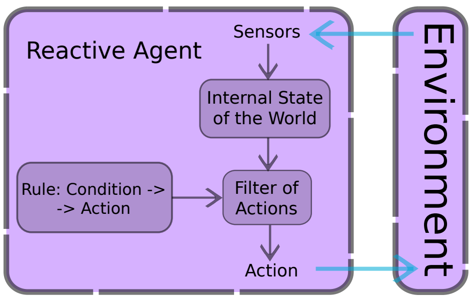


Figure 6.1: Simple reactive agent diagram (adapted from [1]).

6.3 Network Similarity

Having the coordinate mapping between the two simulators ready, the vehicles from SUMO were correctly translated to USARSim and the autonomous vehicle was also shown on SUMO. However, this improvements have shown a network disparity between the two models, as the road lengths and positions from each other were not exactly the same thus mirroring the vehicles in a slight different position from where they should have been drawn. To address this issue, the Aliados SUMO network description was edited in netEditor to approximate the two network positions, however the lack of a proper tool that could render both networks at the same time prevented us to carry out this task on the most approximate way.

6.4 Multi-agent communication

The USARSim platform implements a *Wireless Communications Server* to act as a middle man between robots, simulating message and connection dropping when its distance is not realistically feasible which can be used for Multi-Agent Systems based coordination methodologies. In the following section an example implementation of a reactive agent to control an autonomous vehicle in USARSim is detailed, as a mean to introduce a practitioner to the development of agent-based vehicle control in the proposed platform. However, for the sake of time this agent do not feature any type of communication abilities with other vehicles. Refer to the USARSim documentation for further elucidation on the *Wireless Communications Server* protocol [82].

6.5 Simple Reactive Agent

To validate the proposed framework and its implementation as already discussed in the last chapters, a simple autonomous agent was developed to spawn a vehicle in the simulation, receive its sensors information and calculate a trajectory based on it. It follows a reactive agent methodology, with the model as depicted in Figure 6.1.

This application was coded in C++ with the Qt4 framework from Nokia, which provides cross-platform Graphical User Interface (GUI) and networking capabilities.

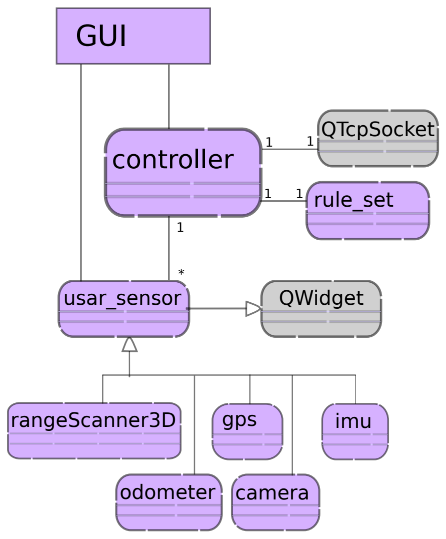


Figure 6.2: Simple reactive agent architecture.

6.5.1 Agent Architecture

The simple reactive agent software architecture is depicted in Figure 6.2. It consists of a central class *controller* which sets up the communication *QTcpSocket* with USARSim and instantiates an *usarsim_sensor* for each sensor in the vehicle. It uses an hash-table that maps each sensor id to its correspondent object, while redirecting the received sensor message for parsing. Each sensor also inherits a *QWidget* class, so a widget containing sensor information in a visual format can be specific to the sensor type. For example, a LIDAR (*rangeScanner3D*) sensor provides a 3D point cloud visualization in OpenGL as illustrated in Figure 6.4.

The reactive agent rules set is represented by the *rule_set* class, to which the *controller* feeds with received sensor data, and retrieves the proper driving speed an steer angle to apply onto the robot vehicle. In this reactive agent example, the camera is used to process the lane position and tries to maintain its aim at it, whereas the LIDAR sensor prevents the vehicle to collide with close objects. Figure 6.3 depicts a sequence diagram with the most important transactions between the simulators and the reactive agent.

6.5.2 Navigation

This section provides more detailed description on the *Drive* and *Steer* commands sent to USARSim, using LIDAR and camera sensors respectively, as an example of a simple autonomous control for an urban vehicle. The reader must note that these methods are not validated to be used on robust autonomous vehicle control, but only to model a very simple navigation algorithm for the sake of demonstration.

To drive the vehicle, only the front side point data from the LIDAR sensor is considered to make it aware of its front vehicles. Furthermore, we select a square window with horizontal and

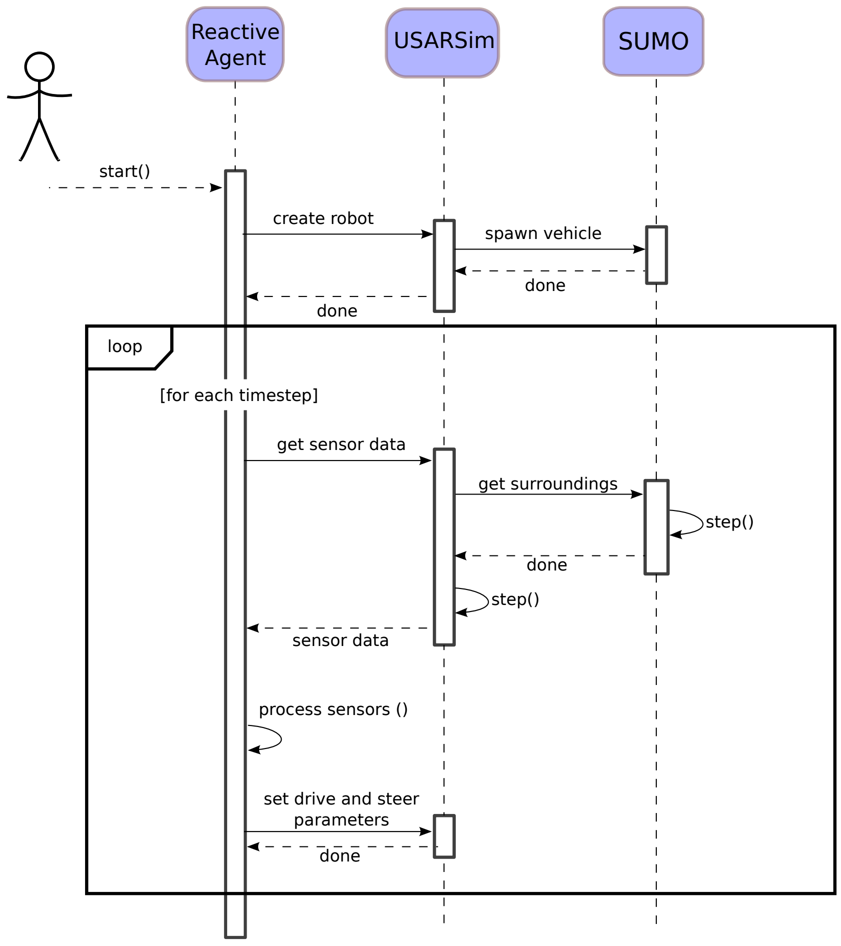


Figure 6.3: Sequence diagram stating the interaction between simulators and the reactive agent.

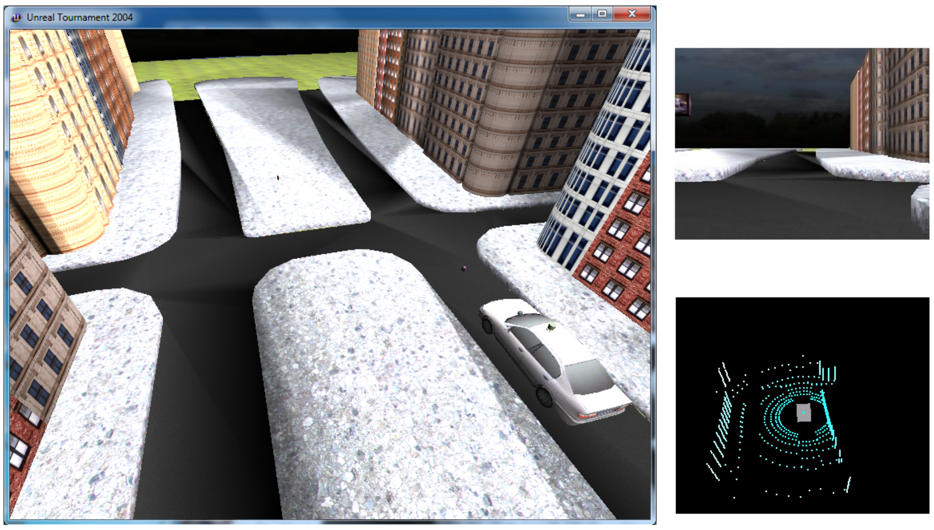


Figure 6.4: Optical Camera and LIDAR sensor visualization interface on a preliminary simulation in USARSim.

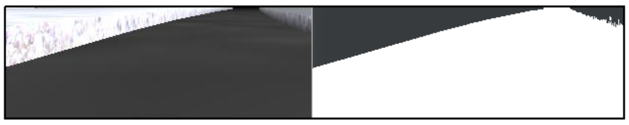


Figure 6.5: The binarization process of the vehicle front camera image.

vertical angles (typically 5 to 10 degrees), and apply the mean to it. Afterwards, the *Drive* speed is calculated using the following formula (Equation 6.9):

$$drive\_speed = max\_speed \cdot (1 - e^{-m/10}) \cdot (1 - e^{-\alpha/8}) \tag{6.9}$$

The expressions in parenthesis attenuate the speed depending on the measured mean m to make the vehicle slow down in other vehicles' presence, and on the steering angle α, to prevent it to steer in high speeds.

The vision-based method to support vehicle steer decisions is processed in the following steps:

Image binarization with a predefined threshold to filter lanes only (in white);

Calculate white pixel density of both left (ρ_{left}) and right (ρ_{right}) side of the image;

Implement a proportional control with the density difference, given by Equation 6.10.

$$\alpha = K \cdot (\rho_{right} - \rho_{left}) \tag{6.10}$$

Therefore if the left pixel density is higher, the steering angle α will be negative and the vehicle will move towards left. If the right pixel density is higher, α will be positive and the vehicle will move towards right, as illustrated in Figure 6.5. Figure 6.6 illustrates the aspect of the implemented reactive agent GUI.

6.6 Summary

This chapter has provided an integrated evaluation of the most important features aiming to pinpoint the required improvements to be implemented in both simulators. A simple reactive agent was devised to push forward the simplicity of an implementation control for autonomous vehicles.

In the next chapter, a coherent performance evaluation metrics is established to assess the framework's practicability regarding its implemented features and the main goals of this dissertation.

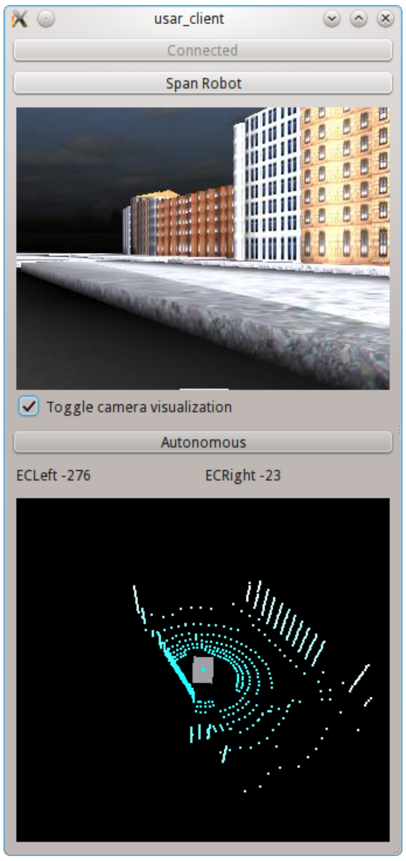


Figure 6.6: A screenshot of the simple reactive agent GUI.

Chapter 7

Preliminary Results and Discussion

To properly validate the integration architecture and implementation devised on this project, several metrics should be analyzed to assess its effectiveness and therefore give a more consistent critic about each part of the system. This way, we can assess the selected choices of the traffic and the robotics simulators, SUMO and USARSim respectively, which seemed the most appropriate to be used in this project. Moreover, this chapter will discuss on the metrics used to evaluate the system, followed by the obtained results and their interpretation.

7.1 Selected Metrics

As the most important piece of work was done on the optimization of the networking and vehicle managing in the two simulators, the following metrics will apply:

- Functionality tests:
  - Move an autonomous vehicle towards a SUMO lane to assess its visibility by SUMO vehicles;
  - Detect a SUMO vehicle in USARSim though the reactive agent application.
- Performance tests:
  - Central Processing Unit (CPU) usage;
  - Network bandwidth.

This evaluation tests were performed on two relatively recent desktop computers available in LIACC/FEUP denominated Shon and Wag. Table 7.1 illustrates the computers specifications, which should be considered in the critical assessment of the result values. Also, Figure 7.1 shows which application will be run on which computer.

The following sections will describe the tests performed, whereby it started by functionality tests, followed by performance tests. The last section summarizes the obtained results.

Table 7.1: Two computer set-ups used to evaluate the integrated platform.

PC	Shon	Wag
CPU	Intel Core 2 Duo E8500	Intel Core 2 Duo E6550
Clock speed	3.16GHz	2.33GHz
RAM size	3.71GB	3.00GB
Graphics Card	Intel Q45 Xpress	Nvidia GeForce 8600GT
Operating System	Windows 7	Ubuntu Linux 11.04

7.2 Functionality Tests

To evaluate the first functionality test, the SUMO client example implemented in Section 4.6 is the best candidate to serve it as the user can control the autonomous vehicle simply by using the keyboard arrows to control it. Furthermore, the following steps were taken to validate the first test:

1. Start SUMO loaded with a sample network (see Appendix A);
2. Start the client example, and connect to SUMO;
3. Move the autonomous vehicle to a vacant lane, using keyboard arrows;
4. Verify if the approaching vehicles decrease velocity as they reach the autonomous one.

This test was performed on Wag only, and as already illustrated in Figure 4.10, it was successfully completed. As the remaining vehicles approached the autonomous vehicle, they have stopped right behind it. When the autonomous vehicle accelerated, the surrounding vehicles would as well accelerate, as they respected the car-following model (see 4.2). One must note that when the autonomous vehicle approaches an intersection in SUMO, the former validation test can also be applied, as each intersection is comprehended by a set of internal lanes working in a similar way than the former. Figure 7.2 depicts the internal lanes as they are shaped in SUMO.

The next validation test is intended to detect a SUMO vehicle in USARSim through the reactive agent application. Furthermore, all the integration platform applications must be loaded, i.e. SUMO, USARSim and the reactive agent. The sensors to be used in this validation test are the front vehicle camera and the LIDAR. The following enumeration identifies the steps to validate it:

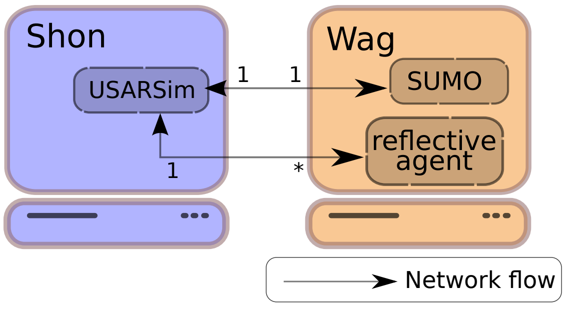


Figure 7.1: Computer-application arrangement to perform evaluation testing.

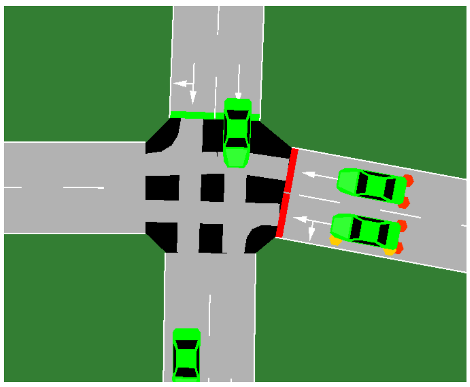


Figure 7.2: SUMO close-up of intersection implementation using internal lanes.

1. Start SUMO loaded with Aliados network in Wag,
2. Start USARSim with Aliados network in Shon,
3. Start the reactive agent application in Wag,
4. Manually move the autonomous vehicle to a lane,
5. Capture a scene when a surrounding vehicle is within range of the autonomous vehicles' sensors.

From Figure 7.3 depicted below with the captured scene, we can state the successfulness of the test. On the first row we can visualize the autonomous vehicle as the yellow car, which is surrounded by three other cars; one at front, one at the back and another on the left side. From the USARSim view we can see the same environment in a 3D model which confirms the synchronization of the two simulators. A careful look at the network topology yields a slight offset amongst the vehicles position in each simulator, as a consequence of the calibration issues already discussed in Section 6.3. The last row on the figure depicts the camera image, which is seeing the front vehicle on the top, whereas a LIDAR point cloud data clearly detecting the road is depicted below. Given the limited resolution of the range sensor it is not evident the presence of a car just using its data.

7.3 Performance Tests

To evaluate the platform performance to be eventually scrutinized against other frameworks the CPU usage of both simulators will be assessed, as well as the network bandwidth, with several test runs.

These test runs use the bundled operating system profiling tools to measure the aforementioned metrics. Thus, on Windows 7 (the Shon machine) the metrics are accessed using the "Performance" tool on Administrative Tools > Performance. In Ubuntu Linux, the *top* and *iftop* commands provide the CPU and network usage respectively.

The first approach is to simulate a SUMO network connecting several autonomous vehicle entities and evaluating the CPU usage and network bandwidth used by them. As one of the main

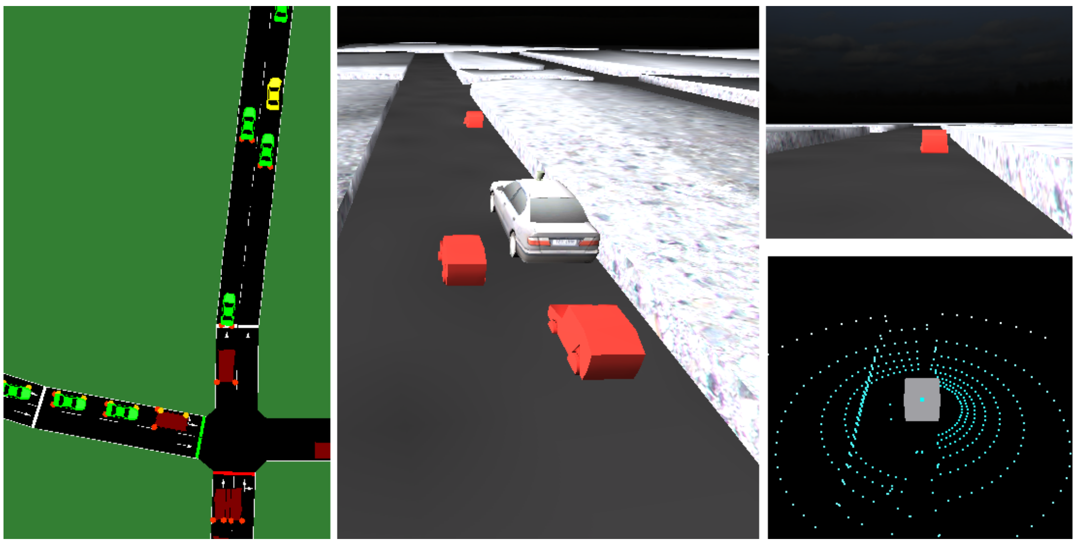


Figure 7.3: Validation of second functionality test. On the first row, SUMO view is presented followed by USARSim. The camera and LIDAR sensor from the reflective agent is depicted in last.

goals of the framework is to retain a real-time refresh-rate, i.e. a minimum 30Hz, when the network latency is superior to 33ms the test-run is stopped. The steps to be taken into account in this SUMO performance test are:

1. Start SUMO loaded with Aliados network in Wag;
2. Start the number of instances of the client sample as required in Wag;
3. Wait until the simulation reaches 10s simulation time;
4. Record both CPU usage and network bandwidth on Wag.

Figure 7.4 depicts the achieved results from the test run. It really looks promising as we can simulate over 1000 autonomous vehicles without significant loss. However, we have to note that when the simulation grows with more vehicles, the consequent number of vehicles surrounding the autonomous vehicles will increase, and so will the network flow and CPU usage.

The next test run depicts the CPU usage and network bandwidth of USARSim connected from one to several autonomous vehicle agents, with all the sensors from the reference vehicle presented in Section 3.3.2. Therefore, the steps to reproduce the test run are:

1. Start USARSim loaded with Aliados network in Shon;
2. Start the number of instances of the reflective agent as required in Wag;
3. Wait until the simulation reaches 10s simulation time;
4. Record both CPU usage and network bandwidth on Shon.

Figure 7.5 plots the results acquired during this second test run. Despite being run on Shon, the fastest machine, USARSim consumes a lot of CPU power given its realistical simulation complexity. Furthermore, the lack of a dedicated Physics Processing Unit (PPU) leaves all physics

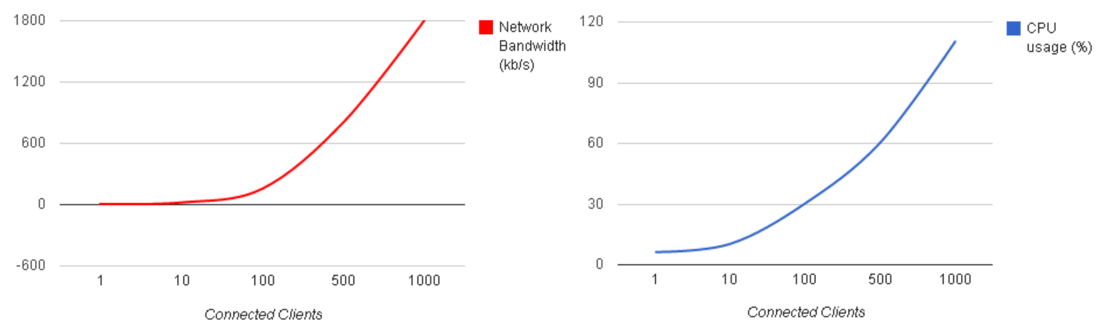


Figure 7.4: Performance Test Run 1 - SUMO simulator performance with several connected clients.

calculations to the CPU. The required network bandwidth is also significantly higher than in its SUMO counterpart, as sensor data is quite large (5 sensors) and the network protocol is not serialized thus wasting higher bandwidth (typically it needs 50 to 100 bytes per each message whereas serialized buffer only require 5 to 20 or even less).

From the practical results we can infer that the required network bandwidth for the framework application can represent a bottleneck if certain conditions are met. Therefore, researchers who intend to adapt this architecture are encouraged to comprehensively study the message passing efficacy when selecting the simulation tools.

7.4 Summary

This chapter discussed some simple functionality test-runs aiming to evaluate the practicability and usefulness of the implemented integration platform. On the SUMO side and from the first functionality test, we can acknowledge the successful implementation of all patches, which allowed it to connect a vehicle with an external controller, thus providing some distributed capabilities to the SUMO platform. On the other hand, the USARSim simulator was not as heavily modified as

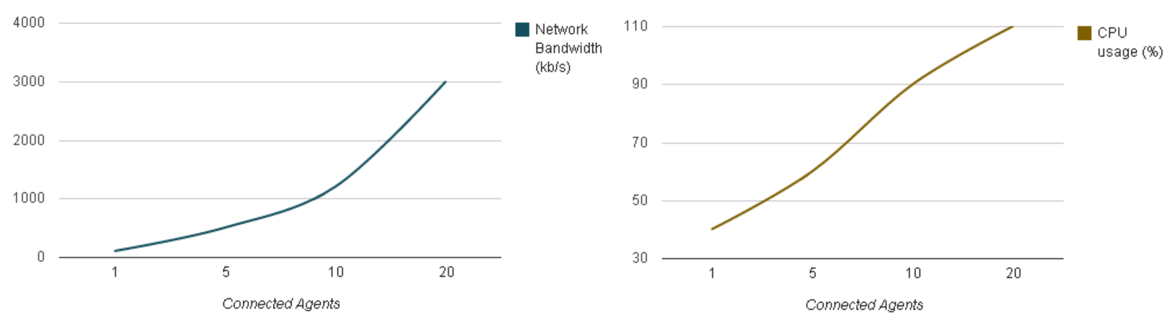


Figure 7.5: Performance Test Run 2 - USARSim simulator performance with several reactive agents.

SUMO, and giving its semi-closed license presented before, the serialized protocol-based transport network was impossible to implement, which provided a network performance below what we had expected. Also, the performance tests demonstrated that the high amount of sensors in the vehicles can be a major drawback in the processing requirements for the autonomous vehicles to be simulated in USARSim, particularly the LIDAR sensor, as its simulation requires an algorithm to search for hundreds to thousands of points in each time step, and since it is implemented in a scripted language, it is not strictly optimized as should be.

The next chapter will conclude this work with a general overview of every chapters discussed, the main achieved results and a future perspective onto further developments of the platform.

Chapter 8

Conclusions

Research in autonomous vehicles has been proving that it is quite possible to integrate fully operator-independent robotic systems within an urban traffic environment. Indeed, in June 2011 Nevada state have announced they will be the first in the world to allow self-driving cars in its urban roads [87].

Facing the current traffic situation in most developed countries it is now imperative to foster new transportation methods using state-of-the-art technologies towards Future Urban Transport (FUT). With the attention on sustainability, autonomous vehicles will also regain a particular focus as its paradigm envisions an equal and sustainable ground transport which should ensure its users productivity and mobility.

Traffic simulation already comprises modern tools for advanced data retrieval from several sources such as emissions reporting, traffic flow, and much more. However, these classic approaches are centered on microsimulation models thus preventing the deployment of an arbitrarily positioned vehicle externally controlled.

The following sections will foster a critical analysis of the designed and implemented solution against the initial goals of of this dissertation, along with its main results. Finally, some short and long term developments to improve the platform and its usability are suggested as well.

8.1 Final Remarks

The main objective of this project was to devise a solution for the coupling of two simulators, a traffic simulator and a robotics simulator to increase the reliability of simulation with several self-driving vehicles, seamlessly integrated within a common traffic environment.

In the literature review chapter, an introduction to the most important topics and current research is overviewed, as well as a taxonomy to evaluate both traffic and robotics simulators for autonomous vehicles development is presented. Furthermore, an integration architecture was devised pinpointing the main aspects towards an efficient communication among simulators, which have been addressed to some extend. This flexible approach was not bound to any specific simulation software architecture thus opening up new opportunities to the development of other platforms.

A prototype to demonstrate the integration was also devised and effectively implemented with both SUMO and USARSim, which were accordingly patched to meet the integration requirements. To couple these two simulators, a method for the management of seamless vehicle mirroring was implemented using efficient data types and networking. The robotics simulator provided its position to the traffic one, whereas the latter calculates all microsimulation vehicles, feeding back their position to the robotics simulator in the same time step.

The effectiveness of the framework has culminated in the need for modeling and developing the reflexive agent, which comprehends a simple navigation and control algorithm as a means to exemplify and measure the framework's promising potentiality and efficiency .

8.2 Main Contributions

Considering the framework implementation outcome and its reported performance measures, we can acknowledge the project was concluded quite successfully.

The proposed architecture envisioned for a flexible approach to the integration of two simulators, one from the transportation area, and the other from robotics area. The initially selected SUMO and USARSim simulators were extensively studied to be modified accordingly to the integration requirements.

On the SUMO side, some modifications were made to decouple the microsimulation vehicle from a bounded lane, and to open up a communication channel so it could be externally controlled. Also, a network file description was built using netEditor, importing the Aliados topology and modeling the routing demand with some resemblance to the actual traffic flow in the area.

USARSim already comprised most of the required parts to be integrated on our simulation framework, albeit its strict bound to the proprietary Unreal Engine retained a major bottleneck as our initial proposed serialized data communication was impossible to implement. To model Aliados, the network was imported from a procedural modeling tool, which has provided a fairly realistic environment to be rendered on USARSim. The autonomous vehicle was based on an already modeled USARSim robot named Sedan whereby it was modified to resemble the Google Autonomous Vehicle sensor architecture.

The integration phase of the project consisted on adapting the simulators to be seamlessly connected. To do so, some efficient approaches were analyzed, such as the network communication, which uses modern distributed computing techniques, or optimized algorithms to calculate the surrounding vehicles and manage its visualization in both simulators in real-time. Finally, the implemented reactive agent demonstrated the ease of use of the platform, as it still respects the documentation provided by both simulators. Therefore, a practitioner already familiar with such tools will not notice any disparity among them.

8.3 Further Developments

Giving the complexity of the simulation frameworks used in this project, the next step towards its solidification would be to fix some occasional bugs that might still persist, particularly on SUMO as it was subjected to several modifications.

Also in SUMO, the collisions between vehicles could be carefully thought of as currently one in which the collided vehicle is teleported (usually removed from the scene) to prevent further simulation discrepancies. To increase the realism throughout the 3D model some objects could be added as well, such as trees, garden seats, or even pedestrians. Also, the integration communication could exchange more variables such as traffic light states, surrounding vehicles presence lights (brakes, turn). The latter would involve a modification on the vehicle replication management mechanism *WorldController*, as at the moment it just supports static meshes.

The protocol buffers should be throughly compared to the former used *ASCII* method which is demanded by Unreal Engine, to assess its major issues when a larger scale network is simulated.

Ultimately, some coordinate mapping techniques could be assessed to study the possibility of using three dimensional topologies of traffic roads in the robotic simulator side, while using the same plain roads in the traffic simulator. The network model calibration method should be evaluated as the result of importing from two different data sources.

In summary, the developed platform should be allowed to grow along with its simulators new releases, for it to become more mature and implement newer features. The possibilities are vast and uncountable.

8.4 Future Perspectives

As for future perspectives, the platform should be tested in a real case study, which will thereby dictate the real practicalness of this project. One suggested work is to study new coordination methods for autonomous vehicles within common urban traffic scenarios.

Another long-term goal would be to foster the integration of this platform with even one more simulator, namely a pedestrian simulator, whereby it can push further the real implementation of an even more realistic Artificial Transportation System.

Appendix A

SUMO Network Files Structure

In this section, a basic understanding of the essential descriptor files to perform a traffic simulation in SUMO is presented.

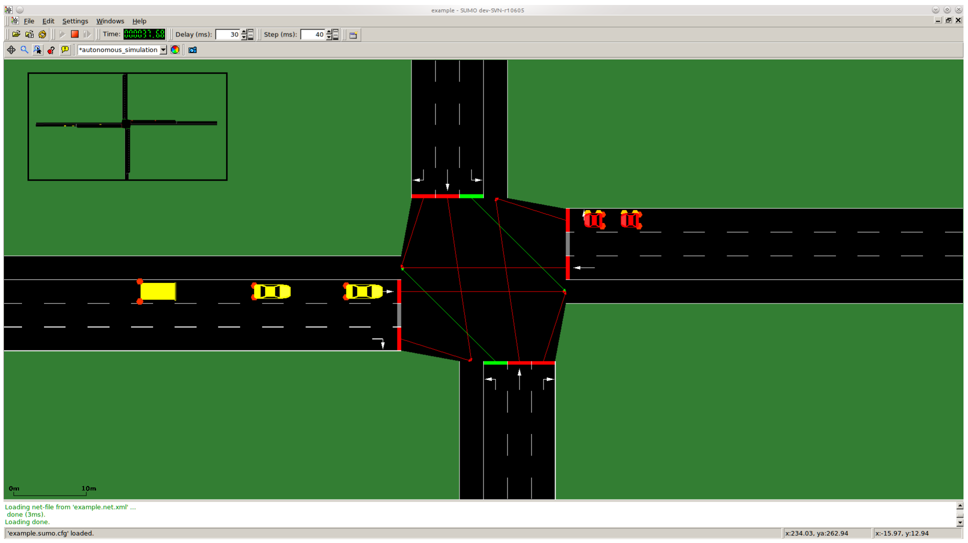
SUMO requires a set of four XML file descriptors to describe a traffic network, namely a **.edg.xml*, **.nod.xml*, **.typ.xml* and **.con.xml* file (in some situations **.typ.xml* can be omitted). Below an example network is comprehensively described:

example.nod.xml Nodes are points in the network for edges to connect to. An intersection will be defined if more than two edges are connected to a node. Each node comprehends an id_n and its (x_n, y_n) position. Also an optional *type* parameter defines the node type. This file descriptor can be analyzed in Listing A.1.

Listing A.1: example.nod.xml file descriptor.

```
<nodes> <!-- The opening tag -->
    <node id="0" x="0.0" y="0.0" type="traffic_light"/> <!-- def. of node "0" -->
    <node id="1" x="-250.0" y="0.0"/> <!-- def. of node "1" -->
    <node id="2" x="+250.0" y="0.0"/> <!-- def. of node "2" -->
    <node id="3" x="0.0" y="-250.0"/> <!-- def. of node "3" -->
    <node id="4" x="0.0" y="+250.0"/> <!-- def. of node "4" -->
    <node id="m1" x="-125.0" y="0.0"/> <!-- def. of node "m1" -->
    <node id="m2" x="+125.0" y="0.0"/> <!-- def. of node "m2" -->
    <node id="m3" x="0.0" y="-125.0"/> <!-- def. of node "m3" -->
    <node id="m4" x="0.0" y="+125.0"/> <!-- def. of node "m4" -->
</nodes> <!-- The closing tag -->
```

example.typ.xml Defines an edge type (e.g. a common road, ranch road, highway, etc), containing all parameters associated with it. In the example bellow *priority* states the edge priority, *nolanes* the number of lanes on it, and *speed* the maximum speed allowed (in m/s). Other optional parameters are available such as edge *length*, *shape*, etc.



```
Loading net-file from 'example.net.xml' ...
 done (3ms).
Loading done.
'example.sumo.cfg' loaded.
```

Figure A.1: SUMO example network from presented file descriptors. Arrows in the intersections represent connections.

This descriptor can be omitted if each edge is individually parameterized in *.*edg.xml* file. An example of the descriptor can be seen in Listing A.2.

Listing A.2: example.typ.xml file descriptor.

```
<types>
    <type id="a" priority="3" nolanes="3" speed="13.889"/>
    <type id="b" priority="2" nolanes="2" speed="11.111"/>
    <type id="c" priority="1" nolanes="1" speed="11.111"/>
</types>
```

example.edg.xml Contains information about the network edges. Its characteristics inherit from a specified edge *type* or individually. *from* and *to* variables represent both the start and end nodes *id* whose vehicles direction will follow. Listing A.3 illustrates an example description.

Listing A.3: example.edg.xml file descriptor.

```
<edges>
    <edge id="1fi" fromnode="1" tonode="m1" type="b"/>
    <edge id="1si" fromnode="m1" tonode="0" type="a"/>
    <edge id="1o" fromnode="0" tonode="1" type="c"/>
    <edge id="2fi" fromnode="2" tonode="m2" type="b"/>
    <edge id="2si" fromnode="m2" tonode="0" type="a"/>
    <edge id="2o" fromnode="0" tonode="2" type="c"/>
    <edge id="3fi" fromnode="3" tonode="m3" type="b"/>
    <edge id="3si" fromnode="m3" tonode="0" type="a"/>
    <edge id="3o" fromnode="0" tonode="3" type="c"/>
    <edge id="4fi" fromnode="4" tonode="m4" type="b"/>
    <edge id="4si" fromnode="m4" tonode="0" type="a"/>
    <edge id="4o" fromnode="0" tonode="4" type="c"/>
</edges>
```

example.con.xml Represent the allowed movements from a source to a destination edge. $from$ and *to* items represent the edge ids. SUMO also provides a mechanism to automatically predefine such connections.

Figure A.1 illustrates the assigned connections of the descriptor below, which automatically generated connections to edges not refered in it. Listing A.4 presents an example file descrition.

Listing A.4: example.con.xml file descriptor.

```
<connections>
    <connection from="1si" to="3o"/>
    <connection from="1si" to="2o"/>
    <connection from="2si" to="4o"/>
    <connection from="2si" to="1o"/>
</connections>
```

After describing the traffic network to be simulated, a traffic demand descriptor with extension **.rou.xml* must be created to place vehicles in the lanes. Finally, a **.sumo.cfg* file containing the global configurations for the simulation should be created containing all aforementioned XML descriptors.

example.rou.xml Two types of vehicle routing methods are demonstrated on the example descriptor. The first one represents distribution based modeling, whereas the user can define a vehicle type *vtype* and its characteristics and defining its probability to be spawned. The same procedure can be done as for vehicles' routes, using *route* definition.

Let V be the event "vehicle *type1* was spawned" and R the event "vehicle followed *route1*". Using the discrete conditional probability, we obtain:

$$P(R|V) = \frac{P(V \cap R)}{P(V)} \Leftrightarrow 0.4 = \frac{P(V \cap R)}{0.5} \Leftrightarrow P(V \cap R) = 0.2$$

Arround 20% of spawned cars will be of *type1* following *route1*.

Finally a *flow* element joins this parameters, along with the vehicles per hour information.

SUMO allows also individual modeling of vehicles. In the example of Figure A.1 two red vehicles are individually spawned at depart time 0001 and 0010 as specified in the example descriptor on Listing A.5. The remaining yellow cars represent the former distribution-based approach.

Listing A.5: example.rou.xml file descriptor.

```
<?xml version="1.0" encoding="UTF−8"?>
<routes>
 <!−− Traffic flow based demand modeling −−>
 <vtypeDistribution id="typedist0">
 <vtype id="type1" accel="0.8" lenght="7.5" maxspeed="34" probability="0.5"
   color="0,1,0" guiShape="passenger"/> <!−− color="r,g,b" −−>
 <vtype id="type2" accel="1.8" lenght="18.75" maxspeed="25" probability="0.5"
    vclass="transport" guiWidth="2.6" guiShape="bus" color="0.5,0,0"/>
 </vtypeDistribution>
 <routeDistribution id="routedist0">
   <route id="route1" edges="1fi 1si 2o" probability="0.4"/>
   <route id="route2" edges="1fi 1si 3o" probability="0.6"/>
 </routeDistribution>
 <flow id="0" type="typedist0" route="routedist0" vehsPerHour="500" />

 <!−− Individual demand modeling −−>
 <vtype id="CarA" accel="3.0" decel="6.0" length="5.0" maxspeed="50.0" sigma="0.5" />
 <route id="route01" edges="2fi 2si 4o"/>
 <vehicle depart="0001" id="veh0" route="route01" type="CarA" color="1,0,0" />
 <vehicle depart="0010" id="veh1" route="route01" type="CarA" color="1,0,0" />
</routes>
```

Despite SUMO only provides two vehicle demand methods, some bundled applications manage to extend it with higher levels of decision [88]:

Using flow definitions and turning ratios Uses turning ration at junctions intead of the destination edges for flows.

Using OD-matrices Have to be converted to trips first, then from trips to routes;

Using random routes This is a fast way to fill the simulation with life, but nothing that has something to do with reality;

Describing the population in the network This method is called activity-based demand modeling.

example.sumo.cfg Defines SUMO configuration files location for a simulation. It can contain other parameters such as the start and end times, or TraCi listening ports. An example description is presented in Listing A.6.

Listing A.6: example.sumo.cfg file descriptor.

```
<?xml version="1.0" encoding="UTF−8"?>
<configuration>
```

```
    <input>
        <net−file value="example.net.xml"/>
        <route−files value="example.rou.xml"/>
    </input>
</configuration>
```

After having all the descriptors defined, a command-line tool named "netconvert" must be executed to convert the first four files in a network file with an **.net.xml* extension:

```
netconvert -n example.nod.xml -e example.edg.xml -c example.con.xml -t example.typ.xml -o example.net.xml
```

Finally we can simulate the generated network and demand issuing the following command:

```
sumo-gui -c example.sumo.cfg
```

References

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[2] Team Leader, Charles Reinholtz, Randolph Hall, Mail Code, Virginia Tech, Thomas Alberi, Jesse Farmer, Scott Frash, Chris Terwelp, and Al Wicks. DARPA Urban Challenge Technical Paper. *Defense*, 2007.

[3] E.D. Dickmanns, R. Behringer, D. Dickmanns, T. Hildebrandt, M. Maurer, F. Thomanek, and J. Schiehlen. The seeing passenger car 'vamors-p'. In *Intelligent Vehicles '94 Symposium, Proceedings of the*, pages 68 – 73, oct. 1994.

[4] D. Pomerleau and T. Jochem. Rapidly adapting machine vision for automated vehicle steering. *IEEE Expert*, 11(2):19 –27, apr 1996.

[5] B. Ulmer. Vita-an autonomous road vehicle (arv) for collision avoidance in traffic. In *Intelligent Vehicles '92 Symposium., Proceedings of the*, pages 36 –41, jun-1 jul 1992.

[6] L. Bishop, D. Eberly, and T. Whitted. Designing a PC game engine. *Computer Graphics*, 18(1):46–53, 2002.

[7] id Software. "DOOM 3" game engine by : `http://www.idsoftware.com/`, 2011.

[8] Inc. Epic Games. "Unreal" game engine: `http://www.unrealtechnology.com/`, 2011.

[9] Eike Falk Anderson, Steffen Engel, Peter Comninos, and Leigh McLoughlin. The case for research in game engine architecture. *Proceedings of the 2008 Conference on Future Play Research, Play, Share - Future Play '08*, page 228, 2008.

[10] V Giasolli. Serious Gaming . *Microscopy and Microanalysis*, 12(S02):1698, August 2006.

[11] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. USARSim: a robot simulator for research and education. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1400–1405, April 2007.

[12] B. Gerkey, R.T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, number Icar, pages 317–323. Citeseer, 2003.

[13] R.E. Fenton. Ivhs/ahs: driving into the future. *Control Systems, IEEE*, 14(6):13 –20, dec 1994.

[14] D.E. Kirk. A path-finding algorithm for an unmanned roving vehicle. *Technical Report 32-1369*, 1969.

[15] AW&ST. Researchers Channel, AI Activities toward Real-World Applications. Aviation Week and Space Technology, 1986.

[16] A Zapp. Automatische Straßenfahrzeugführung durch Rechnersehen. PhD thesis, Universität der Bundeswehr München., 1988.

[17] H. Braess and G. Reichart. PROMETHEUS: Vision des "intelligenten Automobils" auf der "intelligenten Strasse"- Versuch einer kritischen Würdigung. Automobiltechnische Zeitschrift, 6:330–343. 1997.

[18] B. Ulmer. Vita ii-active collision avoidance in real traffic. In *Intelligent Vehicles '94 Symposium, Proceedings of the*, pages 1 – 6, oct. 1994.

[19] D. Pomerleau. RALPH: rapidly adapting lateral position handler. *Proceedings of the Intelligent Vehicles '95. Symposium*, pages 506–511, 1995.

[20] D. Pomerleau. Visibility estimation from a moving vehicle using the RALPH vision system. *Proceedings of Conference on Intelligent Transportation Systems*, pages 906–911, 1998.

[21] D. Pomerleau and T. Jochem. Rapidly adapting machine vision for automated vehicle steering. *IEEE expert*, 11(2):19–27, 1996.

[22] Uwe Franke, Dariu Gavrila, Steffen Gorzig, Frank Lindner, F. Puetzold, and Christian Wohler. Autonomous driving goes downtown. *Intelligent Systems and Their Applications, IEEE*, 13(6):40–48, 2002.

[23] DM Gavrila, Uwe Franke, C. Wohler, and S. Gorzig. Real time vision for intelligent vehicles. *Instrumentation & Measurement Magazine, IEEE*, 4(2):22–27, 2002.

[24] U. Franke, S. Gorzig, F. Lindner, D. Mehren, and F. Paetzold. Steps towards an intelligent vision system for driver assistance in urban traffic. *Proceedings of Conference on Intelligent Transportation Systems*, pages 601–606, 1998.

[25] U. Franke and S. Heinrich. Fast obstacle detection for urban traffic situations. *IEEE Transactions on Intelligent Transportation Systems*, 3(3):173–181, September 2002.

[26] U. Franke and a. Joos. Real-time stereo vision for urban traffic scene understanding. *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*, (Mi):273–278, 2000.

[27] Alberto Broggi, Massimo Bertozzi, Alessandra Fascioli, C.G.L. Bianco, and Aurelio Piazzi. The ARGO autonomous vehicle's vision and control systems. *International Journal of Intelligent Control and Systems*, 3(4):409–441, 1999.

[28] R. Schmidt, H. Weisser, P. Schulenberg, and H. Goellinger. Autonomous driving on vehicle test tracks: overview, implementation and results. *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*, (Mi):152–155, 2000.

[29] D. Shipley. DARPA Plans Grand Challenge for Robotic Ground Vehicles. DARPA, 2003.

[30] T G Goodwin. A huge leap forward for Robotics R&D. DARPA, 2005.

[31] DARPA. News Release, 2006.

[32] D. Shipley and J. Walker. Tartan Racing wins 2 million prize for Darpa Urban Challenge., 2007.

[33] Elrob Website. http://www.elrob.org/, 2011.

[34] High Tech Automotive Systems. Grand Cooperative Driving Challenge. http://www.gcdc.net, February 2011.

[35] Google Company. http://googleblog.blogspot.com/2010/10/what-were-driving-at.html, 2011 February.

[36] C.-a. Brunet, R. Gonzalez-Rubio, and M. Tetreault. A multi-agent architecture for a driver model for autonomous road vehicles. In *Electrical and Computer Engineering, 1995. Canadian Conference on*, volume 2, pages 772–775. IEEE, 2002.

[37] Christian Berger. *Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles*. PhD thesis, 2010.

[38] D.L. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.

[39] Lyle N. Long, Scott D. Hanford, Oranuj Janrathitikarn, Greg L. Sinsley, and Jodi a. Miller. A Review of Intelligent Systems Software for Autonomous Vehicles. *2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications*, (Cisda):69–76, April 2007.

[40] Peter Stone, Park Ave, and Florham Park. Multiagent Systems : A Survey from a Machine Learning Perspective. *Robotics*, 2000.

[41] Isaac Asimov. *I, Robot, Spectra*. 1950.

[42] Anders Orebäck. A Component Framework for Autonomous Mobile Robots. *Simulation*, 2004.

[43] L.S. Coles, C.A. Rosen, B. Raphael, T.D. Garvey, R.O. Duda, and CA. *Application of Intelligent Automata to Reconnaissance*. Defense Technical Information Center, 1969.

[44] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent ASIMO: system overview and integration.

[45] R. Aylett and G. Pegman. A generic robot architecture. In *First International Conference on Intelligent Systems Engineering*, pages 1 –6, 1992.

[46] James S. Albus. An introduction to intelligent and autonomous control. chapter A reference model architecture for intelligent systems design, pages 27–56. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[47] Miguel Cordeiro Figueiredo. An Approach to Simulation of Autonomous Vehicles in Intense Traffic Scenarios. *MSc. Thesis*, (June), 2009.

[48] D. Hohman, T. Murdock, E. Westerfield, T. Hattox, and T. Kusterer. GPS roadside integrated precision positioning system. *IEEE 2000. Position Location and Navigation Symposium (Cat. No.00CH37062)*, pages 221–230, 2000.

[49] B Y Jared Jackson. Standardizing Robotic Coordination and Control. *Communication*, (December):82–87, 2007.

[50] Olivier Michel. Webots: Symbiosis between virtual and real mobile robots. In *Virtual Worlds*, pages 254–263. Springer, 1998.

[51] S. Tadokoro, H. Kitano, T. Takahashi, I. Noda, H. Matsubara, A. Shinjoh, T. Koto, I. Takeuchi, H. Takahashi, F. Matsuno, and Others. The robocup-rescue project: A robotic approach to the disaster mitigation problem. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 4, pages 4089–4094. IEEE, 2002.

[52] Christopher Scrapper. MOAST and USARSim: a combined framework for the development and testing of autonomous systems. *Proceedings of SPIE*, pages 62301T–62301T–12, 2006.

[53] J.S. Albus. A Reference Model Architecture for Intelligent Systems Design., 1994.

[54] J.-C. Baillie. URBI: towards a universal robotic body interface. *4th IEEE/RAS International Conference on Humanoid Robots, 2004.*, pages 33–51, 2004.

[55] M. Petry, A. Moreira, L. Reis, and R. Rossetti. Intelligent Wheelchair Simulation: Requirements and Architectural Issues. In *11th International Conference on Mobile Robots and Competitions, TU Lisbon, Portugal*, pages 102–108. Springer, 2011.

[56] Yali Yang, Hao Chen, and Lihua Chen. Evaluation of public transportation system in shanghai, china. In *Computer and Communication Technologies in Agriculture Engineering (CCTAE), 2010 International Conference On*, volume 2, pages 197 –199, june 2010.

[57] Lúcio Sanchez Passos and Rosaldo J F Rossetti. 2010.

[58] Andrew J Sullivan, Dillip Malave, and Naveen Cheekoti. Traffic simulation software comparison study. *Security*, (June 2004).

[59] S P Hoogendoorn and P H L Bovy. State-of-the-art of vehicular traffic flow modelling. *Proceedings of the I MECH E Part I Journal of Systems & Control in Engineer*, 215(4):283–303, August 2001.

[60] Alexis Champion, René Mandiau, and Christophe Kolski. Traffic generation with the SCANeR II simulator : towards a multi-agent architecture. *Main*, (2).

[61] P.a.M. Ehlert and L.J.M. Rothkrantz. Microscopic traffic simulation with reactive driving agents. *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.01TH8585)*, pages 860–865, 2001.

[62] Wilco Burghout. Hybrid Microscopic-Mesoscopic Traffic Simulation Modelling. *PhD thesis*, 2004.

[63] Johan Janson Olstam. A model for simulation and generation of surrounding vehicles in driving simulators. *Science And Technology*, (1203), 2005.

[64] B Chen. A review of the applications of agent technology in traffic and transportation systems. *Intelligent Transportation Systems, IEEE*, 11(2):485–497, June 2010.

[65] Qinghai Miao, Fenghua Zhu, Yisheng Lv, Changjian Cheng, Cheng Chen, and Xiaogang Qiu. A Game-Engine-Based Platform for Modeling and Computing Artificial Transportation Systems. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–11, 2011.

[66] Zafeiris Kokkinogenis, Lúcio Sanchez Passos, Rosaldo Rossetti, and Joaquim Gabriel. Towards the next-generation traffic simulation tools : a first evaluation Future Urban Transport.

[67] Urban Traffic. VISSIM: A microscopic Simulation Tool to Evaluate Actuated Signal Control including Bus Priority 1. *Simulation*, pages 1–9, 1994.

[68] R Bertini and R Lindgren. Application of PARAMICS Simulation At a Diamond Interchange. *Report: PSU-CE-TRG-02-02*, 2002.

[69] J Barceló, J Ferrer, and D Garcia. Microscopic traffic simulation for ATT systems analysis. *A parallel computing version. Contribution to*, pages 1–16, 1998.

[70] Moshe Ben-akiva and Angus Davol. MITSIMLab : Enhancements and Applications for Urban Networks.

[71] C. Sommer, Zheng Yao, R. German, and F. Dressler. Simulating the influence of ivc on road traffic using bidirectionally coupled simulators. In *INFOCOM Workshops 2008, IEEE*, pages 1 –6, april 2008.

[72] Paulo Ferreira. Specification and Implementation of an Artificial Transport System. *Engineering*, 2008.

[73] I.J.P.M. Timoteo, M.R. Araujo, R.J.F. Rossetti, and E.C. Oliveira. Trasmapi: An api oriented towards multi-agent systems real-time interaction with multiple traffic simulators. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 1183 –1188, sept. 2010.

[74] Christoph Sommer, Isabel Dietrich, and Falko Dressler. Realistic Simulation of Network Protocols in VANET Scenarios. *2007 Mobile Networking for Vehicular Environments*, pages 139–143, 2007.

[75] Google Protocol Buffers. http://code.google.com/p/protobuf/, 2011.

[76] José L. F. Pereira, Rosaldo J. F. Rossetti, and Eugénio C. Oliveira. Towards a cooperative traffic network editor. In *Proceedings of the 6th international conference on Cooperative design, visualization, and engineering*, CDVE'09, pages 236–239, Berlin, Heidelberg, 2009. Springer-Verlag.

[77] R. Rodrigues, A. Coelho, and L.P. Reis. Data model for procedural modelling from textual descriptions. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1 –8, july 2010.

[78] Autodesk Autocad 2011. http://www.autodesk.com/, 2011.

[79] SUMO at a glance. http://sourceforge.net/apps/mediawiki/sumo/index.php, 2011.

[80] Qt4 cross-platform SDK. http://qt.nokia.com, 2011.

[81] Adam Jacoff, Elena Messina, and John Evans. A standard test course for urban search and rescue robots. In *In Proceedings of the Performance Metrics for Intelligent Systems Workshop*, pages 499–503, 2000.

[82] USARSim reference documentation. http://iweb.dl.sourceforge.net/sourceforge/usarsim/USARSim-manual_3.1.3.pdf, 2011.

[83] Collada file format. `http://www.collada.org/`, 2011.

[84] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, pages 1442 –1447 vol.3, nov 1991.

[85] The Gamebots project. `http://gamebots.sourceforge.net/`, 2011.

[86] USARSim World Controller. `http://digilander.libero.it/windflow/eng/Usage/worldcontroller.htm`.

[87] Nevada becomes world first state regulating autonomous vehicles. `http://sanfrancisco.ibtimes.com/"`, 2011.

[88] SUMO demand modeling. `http://sourceforge.net/apps/mediawiki/sumo/index.php?title=IntroductionDemand`, 2011.