DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Improvements in an Interactive Traffic and Driving Simulator for Organised Truck Convoys

Monografia submetida à Universidade Federal de Santa Catarina

como requisito para a aprovação da disciplina:

DAS 5511: Projeto de Fim de Curso

Rodolfo Gondim Lóssio

Florianópolis, abril de 2007

Improvements in an Interactive Traffic and Driving Simulator for Organised Truck Convoys

Rodolfo Gondim Lóssio

Esta monografia foi julgada no contexto da disciplina
DAS 5511: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação Industrial

Banca Examinadora:

Dr.-Ing. Tom Tiltmann
Orientador Empresa

Prof. Dr. Werner Kraus Junior
Orientador do Curso

Prof. Dr. Augusto Humberto Bruciapaglia
Responsável pela disciplina

Prof. xxxxxxx, Avaliador

aluno1, Debatedor

aluno2, Debatedor

Appreciativeness

I would like to thank the international organization of students AIESEC for accepting me as a member. The AIESECers (people who work in AIESEC) from Florianópolis Office have provided me with all the support to search for internships and the ones from the German Office have helped me in my first weeks in a foreign country and with bureaucratic papers.

My parents were another source of sustainment, who since the first moment have supported me to have this international experience. One year far away from them was just possible due to our weekly callings.

In the institute where I worked, I would like to give special thanks to my team leaders that I have had during my one year of internship: Andreas Friedrichs, Phillip Meisen and Uschi Rick. Another important person was Martin Henne, my adviser and consulter for technical questions, who explained the system in a brilliant way with his natural teaching skills, besides being a great cooker!

I can not forget to thank the person who gave me the opportunity to do this internship, my supervisor Mr. Tom Tiltmann, who coordinated my work, supported this report and allowed me taking German class during some hours from my work schedule. Moreover he lent me a bicycle that saved me a lot of time and invited me for a ride around the city.

And for last, to my co-workers Sebastian, Alem and Gregor, the modelers of the project and the closest co-workers. To Martin Selbach, Verena Jänen, Leony Petry and Walter Spanjersberg. I still have to thank the responsible professor of the institute, Prof. Klaus Henning.

I would like to thank also Prof. Werner, who was my supervisor from UFSC in this internship and Prof. Augusto, who always helped me with matters of internships during my studies. Some colleagues from my university course, with whom I have exchanged information about life and work: Vitor Bazzo, Rodolfo Flesch, Marcelo De Lellis and Adriano Bess.

Abstract

In the future an intelligent vehicle convoy is a possible form of organization for goods traffic in motorways, which only the leading vehicle is manually driven while all following vehicles are automatically controlled. Those vehicles will be equipped with special electronic devices that allow the communication between them and sensors that define their situation in motorways.

Besides the technical construction of the experimental vehicles, a driving simulator is required to examine the driver's work load and acceptance. This simulator allows also predicting the impact of this convoy system in motorways, since the direct implementation of this system in the real world could create terrible results.

This report presents a description about an interactive driving simulator that integrates this concept of convoys for truck vehicles, including a real cockpit of a truck with the same electronic devices that the real one will be equipped. In addition, this system combines a driving simulation and a traffic simulation, which considers the effects of driving towards the surrounding traffic and vice versa in real-time. The reaction of following traffic is visible and also included into the simulation due to integrated rear view mirrors.

In addition to provide a virtual simulation for the convoy systems, the same interactive driving simulator can be applied for the following fields - testing of new vehicle technologies, analysis of critical traffic conditions, reconstruction of accidents and valid classification of responsibilities, optimisation of human-machine systems, schooling and training for driving safety.

Resumo Estendido

O presente relatório refere-se ao projeto do acadêmico Rodolfo Gondim Lóssio para a conclusão do curso de Engenharia de Controle e Automação Industrial pela Universidade Federal de Santa Catarina. A tradução livre do título desse relatório, *Melhoramentos em um Simulator de Direção Interativo para Comboios de Caminhões Organizados*, baseia-se nas atividades realizadas pelo acadêmico durante Abril de 2006 e Março de 2007 no Centro de Aprendizado e Gerenciamento de Conhecimento (ZLW-IMA, *Zentrum für Lern- und Wissensmanagement*) pela Universidade Técnica de Aachen (RWTH Aachen). Porém, o cronograma de trabalho desse relatório se baseia nos últimos 6 meses de atividades. O acadêmico foi supervisionado pela equipe de Engenharia de Produto (Produkt-Engineering).

Devido ao aumento do tráfego de veículos nas auto-estradas, e falta de recursos para sua manutenção e expansão (quando possível), novas idéais surgiram para harmonizar o fluxo de veículos. Uma delas é a implementação de comboios inteligentes nas auto-estradas, no qual somente o primeiro veículo é controlado manualmente, enquanto os demais veículos o são automaticamente. Para tanto, os veículos integrantes desses comboios devem ser equipados com dispositivos eletrônicos que permitem a comunicação entre eles e sensores para determinar sua posição nas auto-estradas, incluindo câmeras e sistema de GPS.

Além da parte técnica para construção desses veículos experimentais, um simulador de direção é necessário para investigar o comportamento do motorista. Esse simulador permitiria também prever os impactos de sistema de comboio nas auto-estradas, uma vez que a implementação direta no mundo real poderia acarretar vários problemas.

O presente relatório objetiva a descrição de um simulador de direção interativo que integra esse conceito de comboio inteligentes para caminhões. Tal simulador é dotado por uma cabine original de um caminhão, além dos mesmos dispositivos extras especificados para efetuar o comboio. No geral, o sistema combina simulação de tráfego e direção, o que garante em tempo real a interação entre um motorista humano e o tráfego de carros simulados, que são visualizados por animação computacional, através de projetores para a visão frontal e monitores LCD para os retrovisores.

Para garantir essa realidade virtual, um programa computacional é necessário,

a fim de simular o ambiente real e permitir que o motorista interaja com o sistema. Tal ambiente é dotado por objetos tridimensionais, que devem ser previamente projetados por específicos *software* de modelamento 3D e 2D. Outra parte importante é a simulação do tráfego de veículos e a leitura das ações realizadas pelo motorista, que irão influenciar na física do movimento.

Considerando somente o simulador de direção, o mesmo pode ser aplicado em outras áreas, como por exemplo, no teste de novas tecnologias no setor automobilístico, análise de condições críticas de tráfego, reconstrução de acidentes e seu impacto, otimização de sistemas homem-máquina e treinamento de direção.

Index

List of Figures

List of Tables

Table of Symbols

2D	Two dimensional
3D	Three dimensional
API	Application Programming Interface
CAD	Computer Aided Design
CAN	Controller Area Network
CPU	Central Processing Unit
CS	Crystal Space
CVS	Concurrent Versions System
EPROM	Erasable Programmable Read-Only Memory
FIS	Fahrersysteminformation - Driver Information System
FLTK	Fast Light Toolkit
FPS	Frames per second
GPCL	General Polygon Clipping Library
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
InDriveS	Interactive Driving Simulator
LGPL	GNU Lesser General Public License
NIC	NIOBE Command
OS	Operation System
PNG	Portable Network Graphics
SDK	Software Development Kit
RAM	Random Access Memory
RWTH Aachen	Rheinisch-Westfälische Technische Hochschule Aachen - Aachen University of Technology
TCP/IP	Transmission Control Protocol / Internet Protocol address
WLAN	Wireless local area network
ZLW-IMA	Zentrum für Lern- und Wissensmanagement - The Center of Learning and Knowledge Management

Chapter 1: Introduction

The road infrastructure in several countries has almost reached its highest capacity, due to the constant increase of traffic of vehicles and the lack of investment to improve and expand it, when that is possible. In consequence, there is a continuous traffic jam along motorways that can cause delays and car accidents. Solutions for this problem are an absolute necessity to relieve the current traffic system without changing the existing road infrastructure.

Former studies in this research area recommend the introduction of an intelligent and organized convoy systems in order to optimize traffic by harmonizing its flow. Therefore, a project called KONVOI was created, that aims at the development and evaluation of truck convoys on motorways, to be implemented in commercial use by freight forwarding companies[1].

This project integrates recent research coming from different projects. One of them is concerned with an electronic technique for driving safely and comfortably, which is responsible for the development of the required technology of truck convoys (sensors, actuators and driving assistance system). Such technology is essential for establishing truck convoys, since they provide an operational and a safe way to drive without considering the direct human interference.

Due to the complexity of the whole system and its implementation in real motorways, one segment of the KONVOI project is the creation of a simulator that provides the same conditions found in real motorways. Besides that, the same technology embedded on the trucks will be integrated in a static real prototype. That includes the communication with other virtual trucks using a driving system assistant module, which will integrate the convoy.

Nowadays, simulation is used in many contexts, including training, testing, education and safety engineering. Normally, the simulators are based on computer, and it has become a useful part of modeling many real systems, for example: cities, driving and flight simulators, or traffic of vehicles and industrial processing.

The computer-simulated environment provides for the user a virtual reality that is essentially a visual, acoustic and tactile information feedback. The visual module can use computer graphics to perform the virtual world, which includes some requisites for

the visualization. The first one is called modeling, that describes the shape of a real object into a virtual world. In complement, the animation (motion) and the rendering (image) of this object are also necessary.

For a driving simulator, which is an object of work in this report, the tactile information is composed by the main elements of a vehicle, as the wheels, pedals, etc. Besides that, a traffic simulation is required as well, therefore, the driver can interact with a scenario that contains virtual vehicle drivers.

1.1: Objectives

The building of this driving simulator aims to provide a testable environment for the project KONVOI. The reason to build this simulation is to identify failures in the system, errors of protocols, lack of information oriented to the trucks and check the robustness of the convoy. The validation of the results is essential for the implementation in real motorways. Therefore, the simulated environment must present the same characteristics found in real life, as physics events (gravity, inertia), meteorological factors (fog, night, sunlight) and acoustic.

The frame of this end-course project corresponds to develop, improve and support a driving simulator for the project KONVOI. These activities are related specially with the visual module and its tools to guarantee a realistic environment for the driver. Moreover, some tools must be developed to monitor and assist the simulation.

It is also intended to build a demonstrator that contains the main functionalities of the project KONVOI, but with a simple, flexible and portable hardware, that can be assembled and disassembled easily and quickly.

1.2: Course context and correlations

Simulators based on computer environments use software that works with graphic libraries and specific hardware as interface for users. In this project context, a driving simulator that includes a real vehicle requires a special network communication with a computer. In terms of software, programming for the user interface and modeling of the virtual world are required as well.

The disciplines from the Control and Automation Engineering course related to

this project are:

- Analytic Geometry: transform of system of coordinates, build of geometry;
- Industrial Computing I: use of computer language;
- Industrial Computing II: real time application and multi-threads application;
- Software Engineering: use object-oriented programming;
- Feedback Control Systems: application of controlling theory to solve dynamic and communication problems;
- Microprocessors: support the choice of a microcontroller for the project;
- CAD Systems in Engineering: modeling of 3D objects;
- Computer Networks for Industrial Automation: use of network libraries and CAN bus protocol used in real vehicles.

1.3: Methodology

In virtual simulation, the activities can be separated in two groups: modeling of objects and computer programming. The first group consists in building 3D models of real objects included in the virtual environment for simulation. The second group aims at performing the dynamic of these objects and the communication with the user or operator of the simulator.

All tools used by the academic for activities concerned with the real driving simulator are open software[1]. For the demonstrator project, proprietary tools are used. The 3D objects are modeled with a program called Blender, that supports animation and rendering of 3D models. The objects are built based on real models and they have to be similar with their real dimensions and textures.

The computer program responsible for loading the 3D objects in the simulation and managing the animation and the communication between the system and the user/operator is called NIOBE. This software is developed with C++ computer language

[1]software with source code available to the general public with either relaxed or non-existent intellectual property restrictions.

and uses a Software Development Kit (SDK) for 2D or 3D games called Crystal Space. This SDK is based on OpenGL[2] and supports physical of movement and acoustic.

Another tool used for developing 2D graphic application is called Fast Light Toolkit. In this case, its usage is related to assistant programs for the simulator, since the building of programs based on window with this tool is fast. The main tool created with this kit is called Roadcraft, which is an editor of the virtual world.The operation system used to develop those tools is GNU/Linux, though it is possible to compile them in Microsoft Windows as well, since the utilized tools are multi-platform.

Another important software in the project is called PELOPS, a private package developed by another institute. This software is responsible to simulate the traffic of vehicles and interpret the information of the real driver, therefore it is necessary to study how to use this tool and configure it.

The studying of the usage of each tool and the concepts of computer simulation are essential for performing the activities, as the building of assisting programs to send and receive information to/from the simulation, the optimization of algorithms and elaboration of new functionalities.

The tasks delegated to the academic are in agreement with the follow hypothesis:

- Identification of errors, bugs and limitation in the system;
- Necessity of the modelers to create the virtual world, for example, the lack of functionality in tools like Roadcraft;
- The conclusions by battery of testing;
- Elaboration of new tools for the simulator;

The project of a demonstrator for KONVOI is defined as a parallel task of the current activities on the main simulator. The activities are concerning with:

1. Represent possible solutions for the project;
2. Analyze each possibility;
3. Make a decision;
4. Start the implementation.

[2]library for developing of graphic applications in 2D and 3D

1.4: Schedule

The activities were executed between September/2006 and Feb/2007. During this project, the following tasks were scheduled:

1. Studying of specific software, tools for developing and API libraries:
   (a) CVS, make files, VIM (text editor);
   (b) Crystal Space, Fast Light Toolkit;
   (c) Blender, GIMP (image editor);
   (d) PELOPS, NIOBE, Roadcraft;
2. Implementation of new functionalities in;
   (a) Assistant tools;
   (b) NIOBE;
   (c) Roadcraft;
3. Optimisation and testing of NIOBE/Roadcraft;
4. Demonstrator Project;
5. Project documentation;
   (a) End-course's report;
   (b) Software documentation;

The timetable in the figure 1.1 shows when the activities were done.

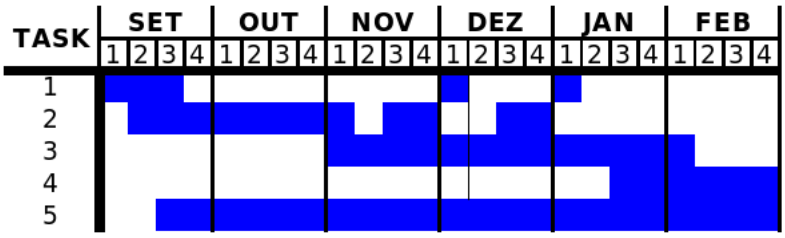


Figure 1.1: Activities during the end-course project

1.5: Report organization

Following this chapter, there is information about the institute where the end-course project was managed, as well as the relevant contents about the project KONVOI and the driving simulator. In the same chapter, the main programs used in the simulator are described in high level, including the steps of modeling and simulation.

The third chapter is concerned with the theory and tools necessary to develop the activities. That includes computer libraries, as Crystal Space and Fast Light Toolkit, and software for editing and modeling 3D objects, as Blender.

In the fourth chapter the driving simulator project is explained. Besides that, some specific implementation in the modules are described and the creation of new tools to assist the simulation.

The fifth chapter is concerned with the demonstrator project, which describes its objective and how it is specified. There is technical information about the hardware used in the project as well.

The sixth chapter contains information about the tests and results of the implementation commented in the fourth chapter. Some graphics of the results are shown in the annex.

Finally, the last chapter contains the conclusions and perspectives for this project and further ones.

Chapter 2: Institute and System Description

The Centre of Learning and Knowledge Management of the Department of Computer Science in Mechanical Engineering (ZLW-IMA) from the RWTH-Aachen University is located in Aachen, Germany. One characteristic of the institute is the interdisciplinary composition of all project teams. The current project - in which the academic has taken part - has engineers, a sociologist, a psychologist, technical and computer scientists.

The business units from the institute reach the following areas - Communication and Organisational Development, Software Engineering, Product Engineering, Knowledge Management and Personal & Controlling. The department of Product Engineering is managed by Dr.-Ing. Tom Tiltmann, who is responsible to orient the activities to the academic.

This department is responsible to provide a service that aims at the optimization of technical production and development processes for enterprises. In particular, this unit provides simulation environments that are specially designed for the automotive technology and networked software. The following sections are discuss the projects from this department in which the academic has participated.

2.1: Electronically Coupled Truck Convoy Project

The Electronically Coupled Truck Convoy Project (KONVOI) was started in May 1st, 2005 and it is planned to finish in July 31st, 2008. The project aims at developing and investigating the behavior of a truck convoy on motorways and an automatic control system for driving. It is planned to have virtual and practical driving tests by using experimental vehicles and a truck driver simulator[1]. The virtual driving environment is demanded to analyze the impact of the system on traffic, before implementation in real life, and drivers behavior as well.

The initial tests are planned to use 4 trucks for the convoys. The real trucks will be equipped with actuators (steering and break), vehicle-vehicle devices for communication (GPS, GSM and WLAN), sensors (object registration in close and far range, recognition of lane), a control unit and a operational assistant unit[1]. The combination

of this work packages allows the driver to contact other drivers from the convoy and set the vehicle as automatic-pilot for special situation.

Another important component of the project is the evaluation of the truck convoy on motorways. In a regular convoy, the trucks can maneuver in several ways, for example coupling into a convoy, change the lane in the motorway or decoupling from the convoy. Therefore, several conditions are defined to guarantee the evaluation of the convoy and the stability. The vehicle behavior is managed by the control unit. For the maneuvers, it is created an electronic coupling of trucks into a convoy by the operational assistant unit on board of the vehicles, called FIS (see section 2.2.1). This application is based on a touch screen monitor running a user friendly software for drivers, which they can easily operate while driving.

The module responsible to acquire information from the testbed truck or from the real truck is loaded in the control unit. This module uses the original CAN Bus system of a truck and can recognize the most important actions from the driver - pedals movement (acceleration, break), wheel movement and indicators buttons.

An initial idea of the whole system is drawn in figure 2.1, that shows how the system will work in a real motorway. In this situation, there is a convoy with three trucks. There is WLAN communication between the vehicles and with GPS satellite. The vehicles are also equipped with an automatic guidance device. In detail, on the top left, it is shown the scrap of the driver information system, which is installed on-board of each vehicle.

Considering now a microview of the system, the block diagram is built and shown in figure 2.2. This diagram is a micro representation of the main elements of the KONVOI project, focused on the driver. Those elements are the base to build the modules for the driving simulation.

The expected results from the KONVOI project are listed below (some of them will not be described in this document, since they are based on parameters out of context from this report):

- Up to 14% increase of traffic flow rate,
- Up to 10% decrease of fuel consumption,
- Reduction of work load of the drivers,
- High acceptance of other traffic participants,

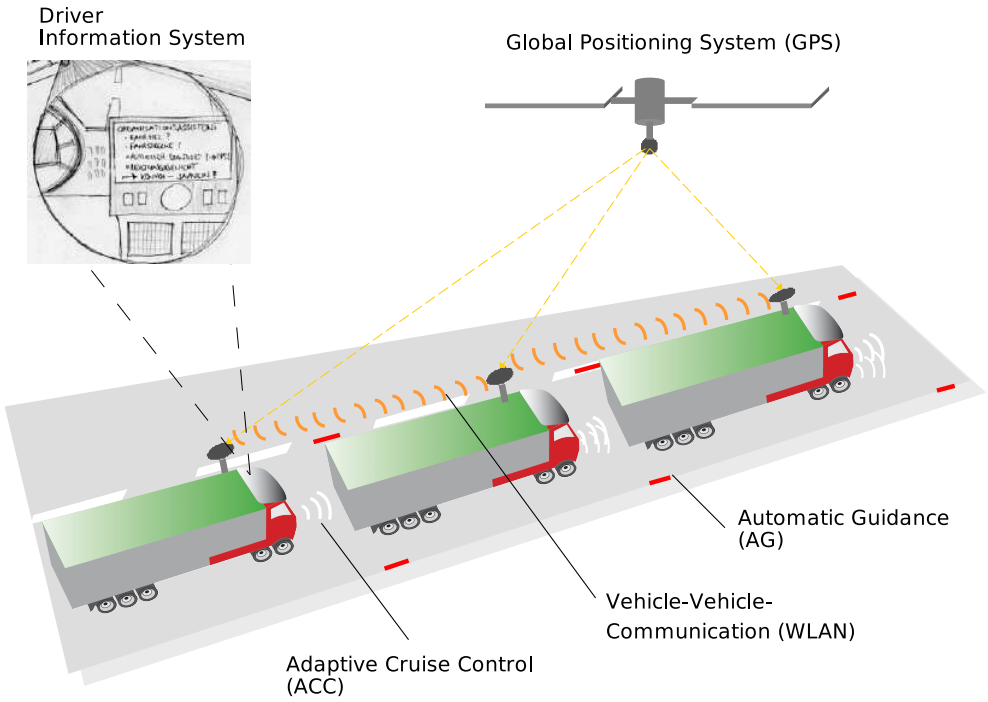


Figure 2.1: Macroview of the KONVOI system

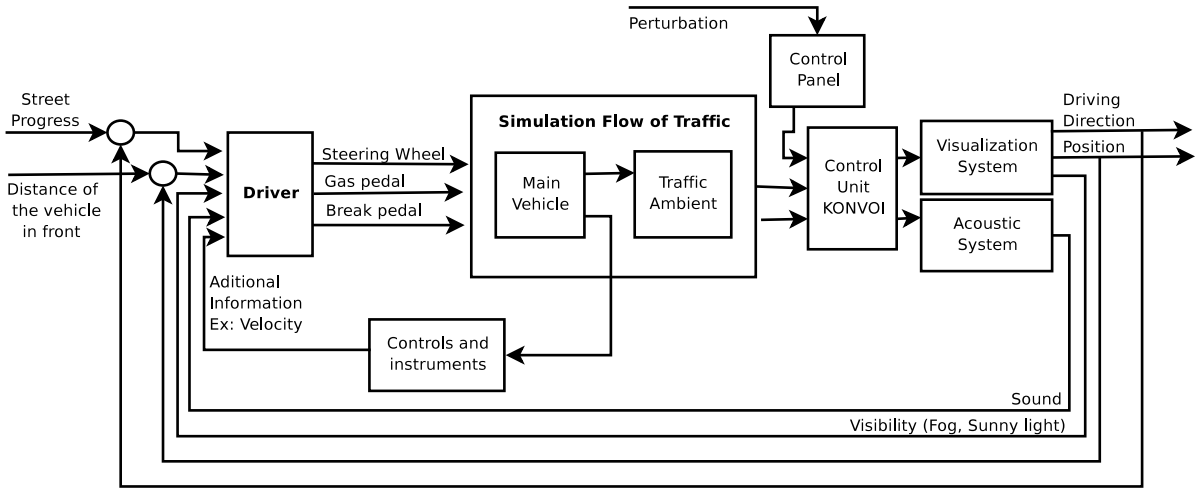


Figure 2.2: Control Diagram of the KONVOI system

- Lowered risk of accidents.

2.2: The driving simulator

There is a large quantity of research concerned with how to describe the real traffic simulation. In the last decades several mathematical or physical models were developed that achieved a great accuracy in simulating the traffic. However, the results provided to the users are mostly given in form of diagrams and tables or only an

insufficient graphical version.

In order to combine the advantages of traffic and driving simulation, two institutes from the RWTH-Aachen University, ZLW-IMA and FKA developed the Interactive Driving Simulator (InDriveS). This simulator provides the effects of driving towards the surrounding traffic and vice versa in real-time. The reaction of following traffic is visible and also included into the simulation view.

The traffic simulation, including the convoy system, is generated by a computer where runs an application called PELOPS (see section 2.2.4). This machine is connected to the truck cabin using CAN bus network. This kind of network is the same utilized in the real truck. The results of the simulation are sent using Ethernet with a broadcast address[1]. The application is responsible to interpret the signals coming from the driver, as the wheel movement, the indicators lights and the pedals.

The other computers have a NIOBE application running. This is responsible for the visualization, therefore, it is connected with the graphic hardware. Those are two video projectors that project the frontal scenario from the driver's point of view, and two LCD monitors, that represent the outside mirrors. For each view there is a dedicated computer to run NIOBE application. For the front view, the computer machine has two video boards, one for each video projector. The information about traffic and the convoy system that NIOBE receives is delivered by Ethernet network. The whole system can be visualized in the figure 2.3.

In the truck cabin exists another system, called *Fahrersysteminformation* (FIS), in English that means Driver Information System. This system is an interface communication with the driver, and it is also connected to the PELOPS machine using CAN bus network. The next section describes FIS in detail.

2.2.1: Driver Information System (FIS)

The Driver Information System provides for the driver in the cockpit an interface to interact with the convoy system. The FIS is based on a touch screen device connected to the truck cockpit and provides a simple communication with a friendly interface for doing maneuvers, for example, coupling and decoupling the convoy.

Besides that, it was implemented a virtual FIS, which makes it possible to apply

[1] a broadcast address is an IP address that allows information to be sent to all machines on a given subnet rather than a specific machine

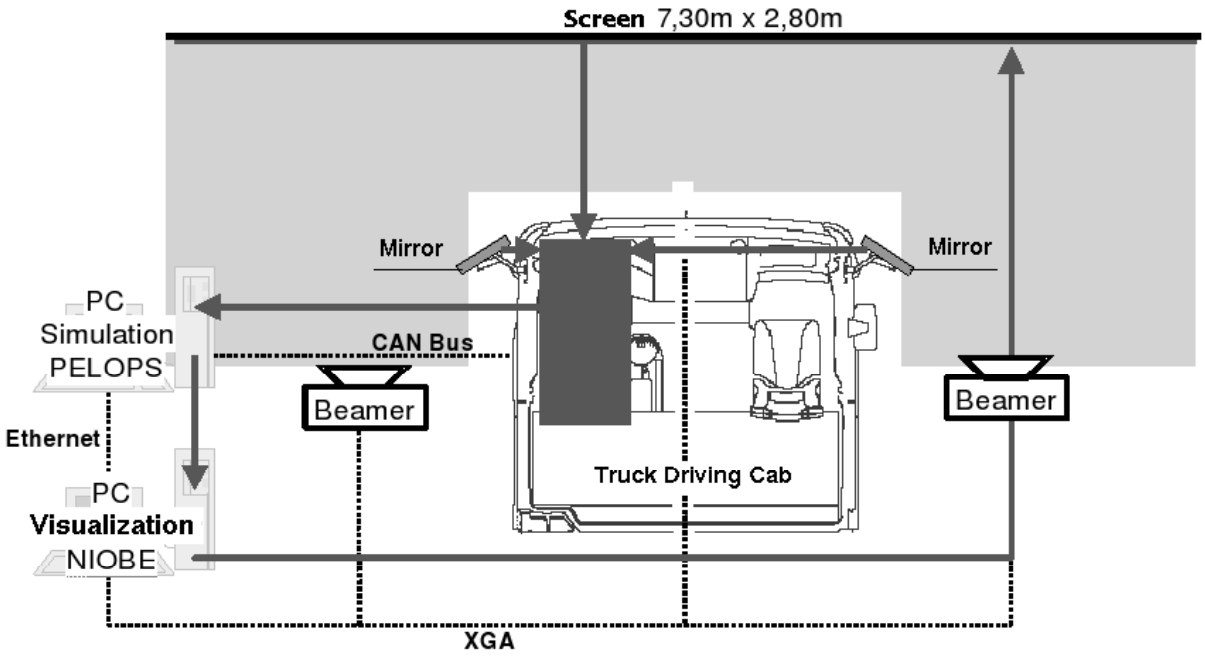


Figure 2.3: Composition of the IndriveS

the same operations using a local application instead of the real device seen in the figure 2.4.

Figure 2.4: FIS module in detail running with the whole system

With both applications the operator/driver can apply different convoy maneuvers and visualize some important parameters of the system, for example, the distance between the front car and the current one.

2.2.2: Visualization Software - NIOBE

NIOBE is part of the driver simulator responsible for the 3D visualization and the audio environment. This application is written in C++ computer language and the source code can be compiled for both operation systems - Windows and GNU/Linux - since the libraries used in it are provided for both OS. The main library used is a 3D-Game-Engine called Crystal Space. This library is explained in another chapter of this document.

The 3D objects viewed in NIOBE are stored in a database that contains geometry and texture information of each object. The position and rotation of the objects are stored in a file generated by Roadcraft program (see section 2.2.3). That includes the motorway, buildings, trees, signs and the others static elements. The dynamic elements, as all vehicles, are defined in a config file. This file contains the initial position, type (truck, car or motorcycle) and an identity number for each vehicle. The figure 2.5 shows the NIOBE's dependencies.

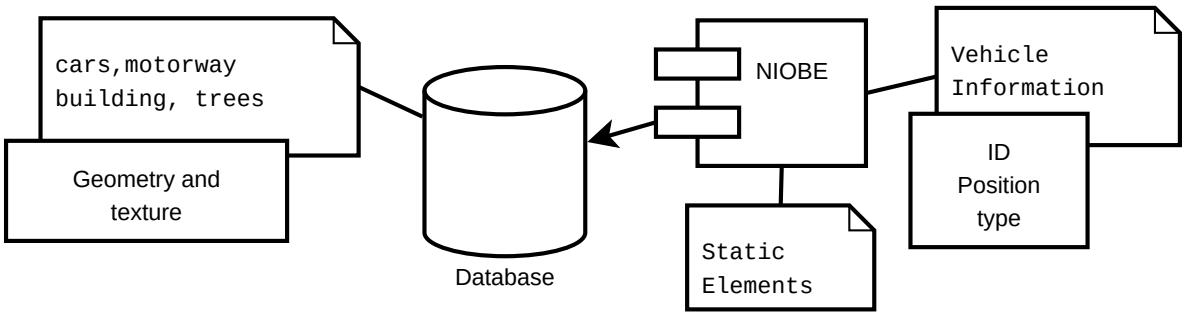


Figure 2.5: NIOBE application and dependencies

Console

Besides the view, NIOBE provides a console where the user can run a list of commands, which characterizes its own language called NIC (NIOBE Command). This language is really useful to debug and change some parameters in offline and especially in runtime mode. Every time that NIOBE is executed, a autoexec file is loaded which contains several commands defined by the programmer. Runtime mode is understood when NIOBE is receiving information about the vehicle positions from an external source. In this situation, it is possible to visualize some action on screen. The list of commands is located in the Annex A.

The console shows in the prompt this label "`[niobe2] >`" for a receipt of a command. There is a thread that reads every line written by the user and interprets in

agreement with NIC. Moreover, the console is used for showing messages from the system, like error and warning messages. It is possible to request some information of the application by console command, for example the position of a specific object or the number of objects loaded in the memory.

Mouse and Keyboard events

NIOBE supports also events from the mouse and keyboard, what means that specific buttons from the keyboard can be pressed to change some parameters during running. Some of these shortcuts are already defined in source code, however it is possible to create new ones in runtime. The mouse can also be used to get some information from the screen and change some conditions of view as well.

A useful example using those events is when the user has to move through the virtual world. It is possible using the arrows and other buttons from the keyboard to move the camera position and use the mouse to change the direction of the movement.

Source code structure

The source code is structured in agreement with the Crystal Space engine. The main thread is responsible for loading the graphics modules, processing the frames and checking the events from the mouse and keyboard. The graphics loading is executed in the beginning and it includes the plugins from the 3D engine and the virtual objects (cars, street, buildings...). After that, the process enters a loop that process every frame. Besides that, it is checked if the user had pressed any button from the keyboard or mouse.

Another thread running in this process is responsible for interpreting the NIC written in the console and show messages as well. There is also another thread responsible for receiving messages from the network. This thread changes the current information of every vehicle in the real world (the dynamic objects) in agreement with those network messages. There is a protected region in the memory where this thread can change the values and the main thread that processes the frames can read this information and update the position of each vehicle in the virtual world.

The console and the view window are showed in the figure 2.6.

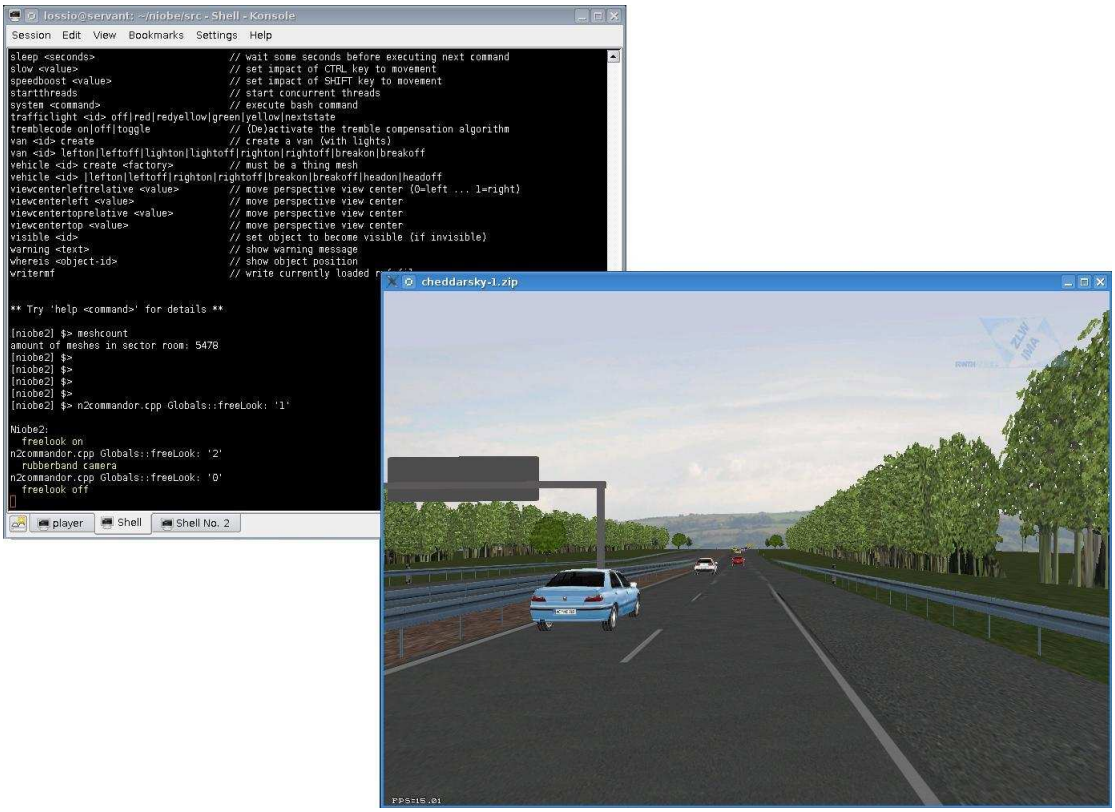


Figure 2.6: NIOBE Application (Console and Graphic Window)

2.2.3: Scenario Editor - Roadcraft

Roadcraft is an editor specially built to create scenarios for NIOBE application, moreover the scenarios are compatible with Crystal Space Engine. With this application, it is possible to easily create a long track representing a motorway and place objects aside the street, such as traffic signs, trees and buildings in a 2D view.

The objects loaded by Roadcraft are from the same database that NIOBE uses. These objects are modelled by a tool called Blender, which is commented in another chapter of this document. Each object can be visualized with a real representation of its top view contour, since Roadcraft depicts a plane view of the objects fixed in the world.

The streets can be built from GPS[2] recorded data track. This is really important, since the scenarios that will be run in NIOBE must be very realistic. The GPS data is obtained by a vehicle driving in the motorway with a GPS device on board that registers the position of the car periodically. In this way, Roadcraft can build a precise representation of the street. However, the objects have to be allocated for each part of

[2]Global Position System

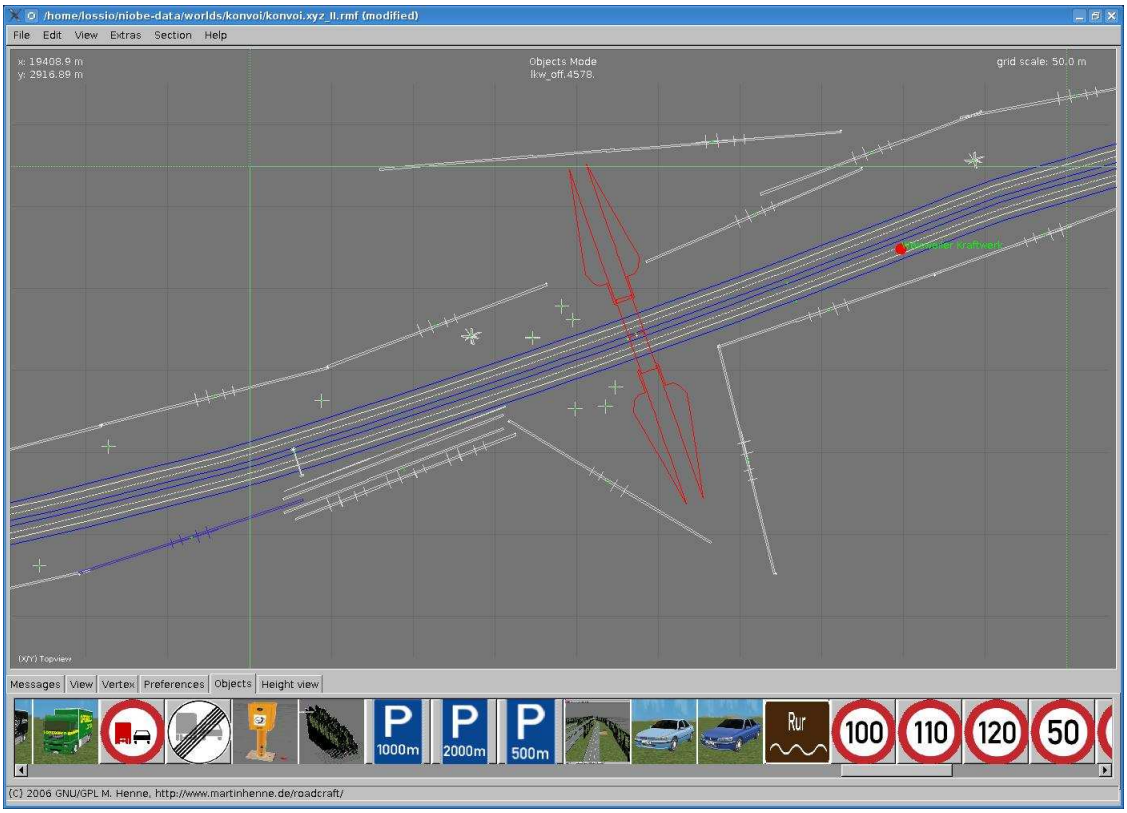


Figure 2.7: Roadcraft Editor with objects in closed

the street. Therefore, the application allows the users to add, move, rotate, delete and copy the objects. The figure 2.2.3 shows detailed objects with their outline and a part of the street.

Roadcraft is able to generate the necessary information for NIOBE to load the scenarios. Basically, it demands two files, one of them containing the data from the whole street, and the other one from the objects. This software must also provide information about the street for the PELOPS application, however the result files differ from one another. NIOBE requires a file based on Crystal Space Engine, on the other hand PELOPS requires the representation of the street as three files that contain the height and angle of the street, position of street segments and number of lanes with their width, respectively.

2.2.4: Traffic Simulation - PELOPS

Program for the Development of Longitudinal Traffic Process in System Relevant Environment (PELOPS) consists in analyzing and simulating the interchanges between vehicle, driver and an environment. This simulation program was developed by another institute called FKA from Aachen. The core of the program is formed by three essential

elements: traffic systems stretch/environment, driver and vehicle [3].

PELOPS is used for the evaluation of the traffic for the driving simulator. The characterization of the traffic element ”driver” is divided into a decision- and a handling model. In the decision model the parameter of the local driving strategy such as speed and lane selection are determined. The handling model converts the characteristics of the local driving strategy into specific vehicle controls, for example, acceleration pedal position, gear lever etc[3].

In synthesis, PELOPS is responsible for the artificial intelligence of each car involved in the traffic simulation and for the collection of data from the driver. Every information concerning the position of the vehicles must be sent to the system by Ethernet, so that the other applications, as NIOBE and FIS, can access and use this information. The periodicity of transmission must be as fast as possible, therefore was it defined a constant period equal to 10 ms. This period has to be less than the period between two consecutive frames processed by NIOBE, because the movement viewed in NIOBE will be smoother and more realistic.

PELOPS allows to set several parameters for the traffic simulation. Following, the possibilities of configuration for the vehicles, street and the traffic are listed.

- Vehicle configuration:
  - Vehicle geometry - width, height and length;
  - Kind of vehicle - motorcycle, truck, car;
  - Maximum velocity and acceleration;
- External parameters for the vehicle and driver:
  - Aggressiveness of the driver;
  - Initial velocity, acceleration and position;
- Traffic environment:
  - Number of vehicles;
  - Size of the bubble of vehicles around the driver;
- Driving environment:
  - Road curvature and position;

- Road inclination;
- Number of lanes;
- Weather conditions;
- Road sign;
- Obstacles;

This chapter described how the driving simulator is related to the engineering project called KONVOI. The main modules of this simulator are objects of work for the academic. On the next chapter, the tools used for building this simulation in software layer are commented.

Chapter 3: Computer-simulated Environment and Tools

A computer-simulated environment provide for the user an environment that tries to describe as high as possible a detailed simulation of a world through the use of visual, acoustic and tactile effects with support of computer technologies. The simulated environment can be similar to the real world and may be used, for example, in simulations for driver training (propose for the project of driving simulator). However, there are several technical limitations to create a good quality virtual reality environment, such as image resolution, objects processing, processing power and communication bandwidth.

One segment of a computer-simulated environment is the 3D computer graphics, whose works of graphic art is created with specialized 3D software. These programs are used to create 3D objects, generally, with certain similarities to real objects.

Due to the popularity of 3D graphics, specialized APIs (Application Programming Interfaces) and SDKs (Software Development Kits) have been created to facilitate the processes in all stages of computer graphics generation. These tools have a really low level communication with the computer graphics hardware manufactures, since they provide access to the hardware in high-level layer for programmers. One of those SDKs is describe on the next section, which was used in this project.

3.1: Framework for developing 3D Applications - Crystal Space Engine

Crystal Space (CS) is a portable modular 3D Software Development Kit, including components for building various applications and games. It is free and falls under the GNU copyleft for libraries LGPL. In short, the LGPL allows you to use Crystal Space as a library in commercial products, but modifications to the library or derivative works incorporating parts of the library must be made freely available to everyone, under the LGPL's terms[4].

CS is programmed in object oriented C++ and is written to run under a wide variety of hardware and software platforms, as Windows, Unix and Apple. The main features of CS are listed below, which some of them are organized in libraries[4]:

- Geometry utility library with handy classes such as 2D and 3D vectors, matrices, transforms, quaternions and oriented bounding box routines.
- General utility library with template arrays, smart pointers, hash maps, object registry, plugin manager, strings and command-line parsing.
- Higher level tool library containing things like procedural textures, collider support and texture generational tools.
- Virtual file system and transparent support for ZIP files.
- Flexible and extensible event system.
- Various types of mesh objects: polygonal lightmapped objects, triangle meshes.
- Animated 3D models.
- Supports various common image formats.
- Lighting static, pseudo-dynamic, dynamic and shadows.
- Powerful sequence manager to control movement, animation and other features in a world.

3.1.1: Components of a World

This section describes the objects used to build a world. Every virtual world based on Crystal Space has the following components[4]:

- **Sectors:** An area in the virtual world that contains several geometrical objects.
- **Mesh Factories:** This component is used to create mesh objects that inherit characteristics from their mesh factory. Basically a mesh objects factory is like a blue-print for creating a mesh object. Usually, the factory defines the actual geometry of an object. It is possible to create multiple mesh objects from those factories.
- **Mesh Objects:** these objects represent the geometry in the sectors. Everything visualized in the virtual world is represented by some kind of mesh object. This is a very important concept in the Crystal Space engine.

- **Textures:** Basically a texture represents an actual image that can be used in materials. Textures are used to give the mesh objects a real appearance. Textures are never used alone but always in a material.
- **Materials:** Basically a material represents a surface appearance for a polygon or triangle. A material is most typically made out of a single texture.
- **Render loops:** This component is an engine structure that tells the engine how to render the objects in a given sector. Basically it tells the engine the steps required to do the rending of the mesh objects and also the steps required to do the lighting.
- **Lights:** A light is needed to illuminate the world. There are three different ways how lighting is applied to objects. Static, pseudo-dynamic and dynamic lights.

3.1.2: Source code structure

An application based on Crystal Space must inherit some basic classes of the 3D library. The most important classes of CS to build an application are *csApplicationFramework* and *csBaseEventHandler*. The first one is responsible for providing an object-oriented wrapper around the CS initialization and start-up functions. The second class is used to manage the event mechanism, like a mouse click or a keyboard press.

A default application usually has the following features, as NIOBE has:

- Basic setup of Crystal Space application using *csApplicationFramework* class;
- Setting up the application event handler using *csBaseEventHandler*;
- Using events to draw 3D frame;
- Loading of the basic plugins needed to get the engine;
- Creating of the 3D window;
- Loading of a texture and material of the world;
- Creation of a simple section and add few lights so it is possible to view everything;
- Creation of a view and a camera;
- Basic movement using the keyboard, mouse or an external source.

3.2: 3D Software - Blender

Blender is an open source software for 3D modeling, animation, rendering, post-production, interactive creation and playback. Available for all major operating systems under the GNU General Public License[6].

The basic features and stages that is found in this software are described on the following sections.

3.2.1: Modeling

This stage describes how the shape of the 3D objects is built. There are several techniques of modeling, for example:

- Constructive solid geometry: with this technique the modeler creates a complex surface or object using Boolean operators to combine them. The most simple shapes that is possible to create are cuboids, cylinders, prisms, pyramids, spheres and cones.
- Polygonal modeling: in this case the modeler use polygons for representing the surface of objects.
- Subdivision surfaces: with this technique is known as to smooth a surface.

Modeling processes may also include editing object surface or material properties, adding textures and other features. In the figure 3.1, it is presented two 3D objects modeled.

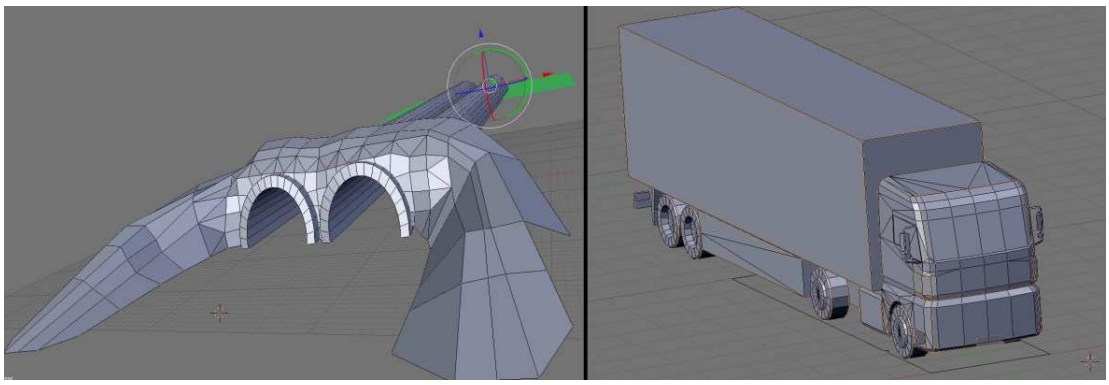


Figure 3.1: Model of a tunnel and a truck in Blender

3.2.2: Rendering

Rendering is the final process of creating the actual 2D image or animation from the prepared scene. This can be compared to taking a photo or filming the scene after the setup is finished in real life. It would contain geometry, viewpoint, texture and lighting information.

3.2.3: Texturing

Polygon surfaces can contain data corresponding to not only a color but, in other cases (as in this project), can be a virtual canvas for a picture, or other scanned image.

A part of the 3D modeling process that is used in almost every object created for the driving simulator is called UV mapping. This technique allows making a 2D map for representing a 3D model. This map is associated with an image known as a texture. In contrast to X, Y and Z, which are the coordinates for the original 3D object in the modeling space, U and V are the coordinates of this transformed map of the surface. Then the image is wrapped onto the surface of the 3D objects.

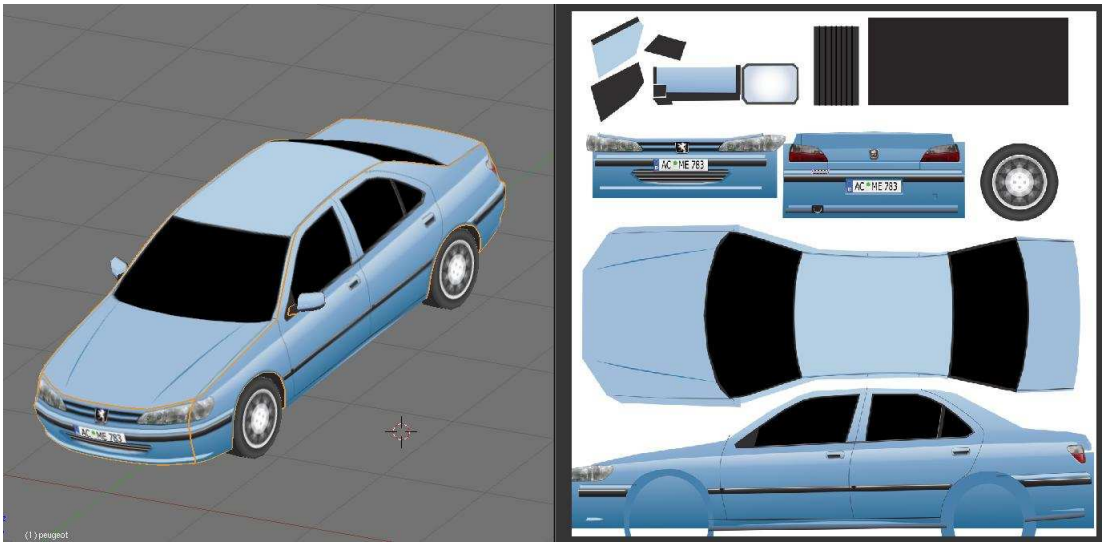


Figure 3.2: Mapping a Peugeot using UV technique

3.2.4: Crystal Space Plugin

Blender has a plugin that allows generating a compatible content for the Crystal Space Engine. In Crystal Space as in Blender there is a clear distinction between an

object and the mesh data it references (the factory). Thus, several objects can share a factory and save space and processing time[8].

Basically a factory is a model for creating objects instances of a specific type. The objects or meshes generated from a factory will inherit some of the characteristics from this factory, as the geometry and textures.

Object types

Crystal Space supports several kinds of Mesh Object types. The most used are listed here:

- Genmesh Object: The genmesh object is one of the most useful mesh objects. It can be used for static data (walls, building, floor ...) or for dynamic data (moving objects). It supports multiple materials, vertex lighting, and even animation.
- Thing Mesh Object: The thing object is often used for static geometry (walls and buildings), usually for complex objects.
- Terrain Mesh Object: The terrain object is very useful if it is wanted landscapes.
- SpriteCald3D Mesh Object: This object is one of the options for animating skeletal models. It is not used in this project.

Basically, in the driving simulator it is used just the Genmesh and Thingmesh objects.

Export Section

This plugin in Blender supports several tools. It is possible to export the element built in Blender as a library or a world file. When a library is exported, that means the file created will be loaded in CS like a factory. If a world file is exported, CS will load it directly in the engine memory and it will be possible to visualize the elements.

Another tool is a wrapper to a demo provided by the CS library, which makes it possible to run the world created in Blender in a CS environment that supports keyboards events and console commands. This tool is used to check if the world or object modeled in Blender would run normally in the CS engine.

3.3: FLKT - Fast Light Toolkit

The Fast Light Tool Kit is a cross-platform C++ GUI that provides modern GUI functionality without complexity. The interface is based on windows and it can be built using a provided user interface builder called FLUID. This program provides the usual widgets to develop common applications and new widgets can be created easily through C++ subclassing.

In addition to building the graphic interface, the program FLUID also generates the source code to be linked to a normal C++ application. All FLTK applications (and most GUI applications in general) are based on a simple event processing model. User actions such as mouse movement, button clicks, and keyboard activity generate events that are sent to the application, which may then ignore the events or respond to the user[10].

In conclusion, the software tools used in the simulation are computer libraries or programs for modeling (3D objects or GUI). On the next chapter it will be described how these tools are used to build the modules commented in the previous chapter, besides explaining some technical details about the simulator.

Chapter 4: InDriveS Project - Workpackages

The project InDriveS combines the several modules described in the chapter 2. The complexity of each module and their interdependency create a system that requires a package of auxiliary tools for testing, analyzing and generating relevant information. Besides that, some implementations for the modules are demanded as a requirement to optimize the system or create new functionalities.

The previous state of the whole driving simulator presented a lot of problems and missing tools. The lack of tools for testing the modules is the main problem, since to test the system before, every module should be used. To contour that, it was proposed to create a package of tools that can be used to replace some of the modules.

Another problem found in the previous state of the simulator was the existence of bugs in the source code and incompatibility of information treated. For example, wrong or insufficient information registered in files, problems of real time tasking and lack of management of memory. In this chapter, the solutions to these issues that were implemented by the author are discussed.

4.1: The Communication among the modules

The main modules used during a simulation of the system are PELOPS, NIOBE, FIS and the auxiliary tools. Roadcraft is an offline tool that is not demanded during the simulation. Except for the communication between PELOPS and FIS, the other tools are connected through an Ethernet network. Therefore, it was defined a protocol of communication based on exchange of network messages. This protocol is defined in a header file that contains several structures for each relevant information. On the table 4.1, it is listed the type of messages exchanged between the modules, indicating the source and destination. Some of them are not defined, however it is predicted to be used in the future.

The messages are sent using a package written in C language, which contains declarations of routines and structures to send and receive TCP/IP messages (point to point or broadcast). The driving simulator system just uses messages in broadcast mode. Therefore, some requisites for communication concerned with IPs and ports

Type Message	Description	Source	Destination
Road element	information about the street	not defined	not defined
Lane	information about a lane	not defined	not defined
Sign	information about a traffic sight	PELOPS	NIOBE
VehicleData	information about a vehicle	not defined	not defined
VehiclePosition	indicates position of the vehicle	PELOPS	NIOBE
EndOfSimulation	indicates the end of the simulation	not defined	not defined
GraphicEngineReady	indicates NIOBE is ready	NIOBE	not defined
Ready	message of acknowledge	not defined	not defined
Success	message of acknowledge	not defined	not defined
Error	indicates an error in the system	not defined	not defined
TraffLightState	indicates the state of a traffic light	not defined	n. d.
InternalNiobeEvent	indicates an event to be run	NIOBE	NIOBE
Accident	indicates an accident	PELOPS	not defined
PelopsTextMessage	a command using NIC language	PELOPS	NIOBE
PlaySingleBeep	command the sound	not defined	NIOBE
Fis2Pelops	FIS commands to PELOPS	FIS	PELOPS
Pelops2Fis	PELOPS information for FIS	PELOPS	FIS
Forces2Pelops	specific commands for PELOPS	Force2Pelops	PELOPS

Table 4.1: Table with the network messages and their description

were defined.

Basically, there are two broadcast addresses, both using the port number **10006**. Every message sent to PELOPS must use the IP address **226.0.0.2** and the other messages (to NIOBE, FIS or other tools) must be addressed to IP address **226.0.0.1**.

In the annex B the variables of the structures used in the implementations of this chapter are shown.

4.2: The human elements

The intern members of the institute define a list of human elements that integrates this project. The members that compose the team are listed below:

- Driver: this is the elemental component of the driving simulator, usually this person is a real truck driver that can give a feedback about how realist is the simulation;
- Support: the support team is a group of people that assists the driver during the simulation. It is composed of sociologist and psychologist. They prepare the driver before the simulation and analyse his behavior;

- Operator: this is the person responsible to run and supervise the simulation. This person has technical knowledge about the system.
- Developers and Modelers: these people are responsible to create the scenario and develop the programs for simulation.

In figure 4.1 it is shown how they take place in the driving simulator environment. The simulation hall is the place where the equipments of the driving simulator are located.

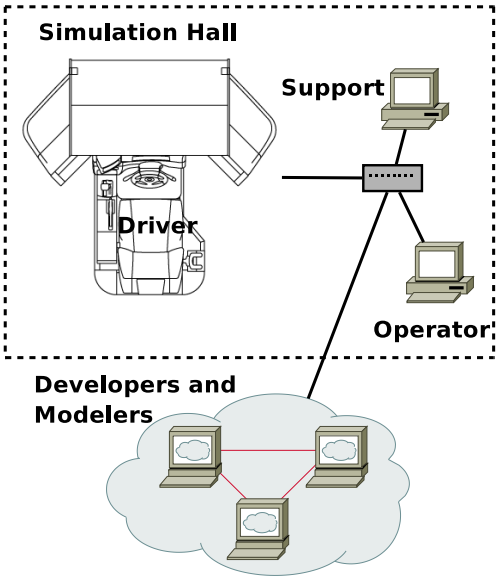


Figure 4.1: Integrants of the driving simulator project

4.3: Niobe implementations

The previous state of NIOBE was functional and the main functionalities were already implemented. During the internship several packages of task were required to improve NIOBE's performance, add new tools and fix several bugs. On the next sections is described the tasks developed.

4.3.1: Map module

This module aims to show in the screen a 2D map of the world loaded in NIOBE and to show specific objects as pixels on that. This tools is just used for testing purposes, since the map is shown in transparent mode on the screen where occurs the

simulation. The basic idea for showing this map is to know where exactly the objects and vehicles are.

As NIOBE supports its own language, a group of commands was create to work with this module. Every command from this group has to start with the word `map`. The first command that should be invoked concerns loading the picture that represents the map, after that it is possible to define which objects in the world must appear like pixels in the map. The module still supports defining the position and the size of the map. On next, the whole commands are listed:

- `map load <filename> <world size XY> <origin XY>` : this command requires the local of the map picture and it must be a PNG, the size of the real map (defined by two scalar values) and which position the corner up left represents in the real world;
- `map setpos <position XY>` : it is possible to change the position of the map on the screen using this command;
- `map setdir <true,false>` : this command rotates the map, sometimes becomes necessary because the picture from the world was rotated;
- `map (un)track <object ID> <color>` : with this command it is possible to show or hide objects in the world on the map by means of a pixel with specific color;
- `map on,off,toggle` : commands to show and hide the whole map.

In the figure 4.2 it is shown how the map appears in NIOBE. The steps for each operation are written down:

1. map load map.png 1095.0 1447.8 686.0 1371.5
2. map dir PI/2
3. map setpos 300 0 360 360
4. map track vehicle
5. map on

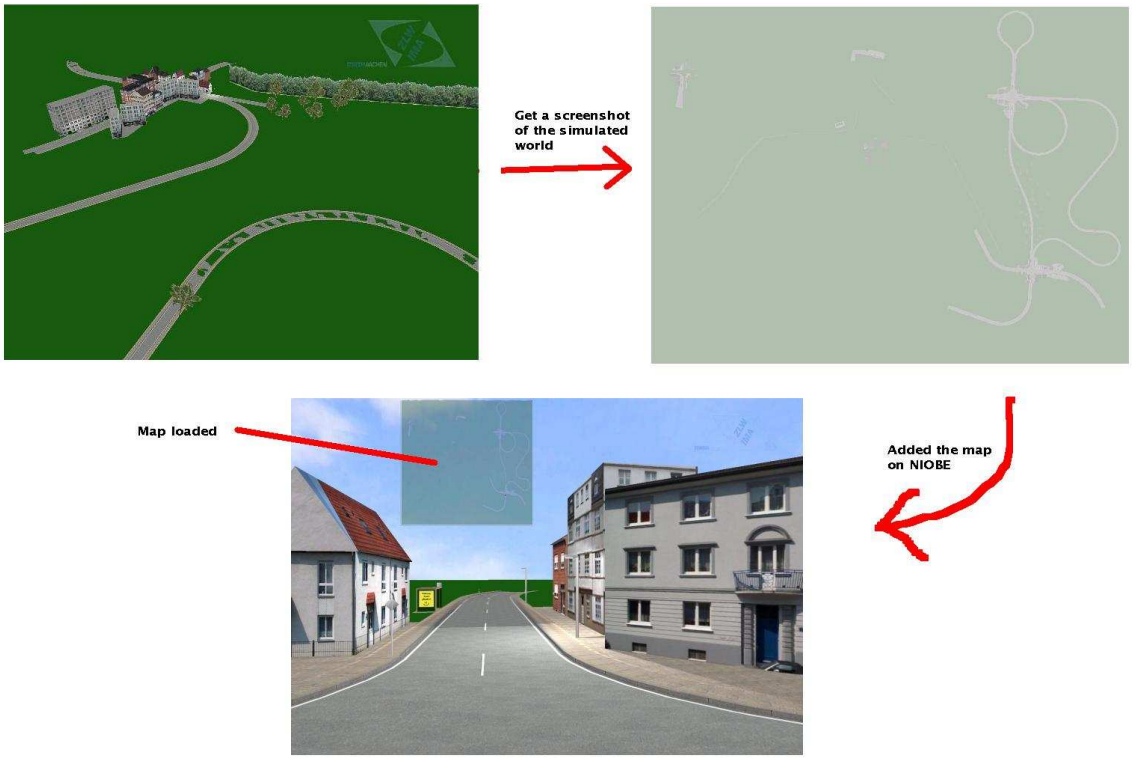


Figure 4.2: Steps to load a map in NIOBE

4.3.2: Trembling problem

Previous state

In the driving simulator system, the PELOPS program has to send periodically messages with vehicle information through the network and, on the other side, the NIOBE program has to read those messages and update the vehicle states on screen. This way, in order to guarantee a smooth movement of the vehicles, these messages should be sent between a short interval of time and the vehicles position should be updated in NIOBE exactly in the time that it is defined (this time is set in one of the attributes of the structure sent by PELOPS).

However, the previous state of this communication was done in such way directly, every message received by the NIOBE program was read and the vehicle position updated without any mathematical treatment. The result of this implementation is a visible trembling of the vehicles in the simulation. The source of this trembling could be in several causes:

- The timing system is not the same between NIOBE and PELOPS machines;
- Delays and loss of information from network communication;

- The operation system scheduling and processing limitation;

Another concern is the Crystal Space Library, who is responsible to calculate and process each frame shown on screen. The library provides some functions that return the interval time between two frames, but as NIOBE receives the vehicle state from the network during the frame processing and the time of a frame processing depends of how many objects are in that frame, it is not possible to define exactly the position of each vehicle that must be set, since prior to processing a frame, it is necessary to foresee where the vehicles will be.

Solution

To contour this inconvenient, it was figured out a way to set this interval of time between two frames as constant, therefore it would be possible to know where exactly the vehicle must be located. As it was written, NIOBE program use Crystal Space to process the frames. For applying a constant period frame, the first thing that should be done is to know how much time is necessary to process each frame. That can be measured using time functions from the proper Crystal Space. So, before finishing the processing of a frame, the system must wait for some time, and this time is defined in agreement with a constant period frame previously defined. This time is calculated in this way:

$$tt = cpf - pfm$$

Where `tt` means the time to wait, `cpf` is the constant period frame defined and `pfm` the period frame measured.

Algorithm

The main idea of this algorithm is to create a base timer for the NIOBE program that is incremented for each frame processed. *A priori*, this increment must be equal to `cpf`. From the network the structure called `VehiclePosition_t` (see annex B) is received. One of the attributes of this structure is the the time stamp (ms) that defines exactly when the vehicle should be for that position. This time stamp must be compared with the base timer managed by NIOBE, and usually, there is a difference between both timers, so it is necessary to make a mathematical treatment to predict where the vehicles must be located. The difference between the timers is named `delay` for implementation reasons. There are four possibilities to treat the position of the vehicles:

- Delay equals ZERO: In this case NIOBE and PELOPS are synchronized, so it is not necessary to apply any calculation, the positions are updated as they are received;

- Delay more than ZERO: In this case the vehicle position received is defined in the future, so it is necessary to compensate the difference multiplying for a factor, that is calculated like this:

$$FACTOR(f) = \frac{cpf}{DELAY - cpf}$$

  This formula is obtained to define exactly the position of the vehicle for a different time (in agreement with the base timer). The biggest is the `DELAY`, the smallest is the `FACTOR`. Since it was received an information from the future, the distance would be bigger, so this factor is smaller.

- Delay less than ZERO: In this case, the distance is also multiplied by a factor, but because it was not received the position required for the estimation, the last positions are stored in a vector, and based on them it is calculated an estimation of the vehicle's position.

- Delay is very BIG: In this case something is going wrong (for example, bad PELOPS or NIOBE performance), therefore it is just updated the current information received from the network.

Besides this algorithm it is also implemented a controlling system. When the delay is more than zero, the base timer is incremented. When the delay is less than zero, the base timer is decremented. In the perfect situation, the base timer is added to `cpf`. The controller adopted to manage this system is a proportional, since the presence of delay in permanent regiment is totally acceptable, because the real system of a vehicle has also such characteristic, for example, when the driver breaks, the vehicle will not react instantly.

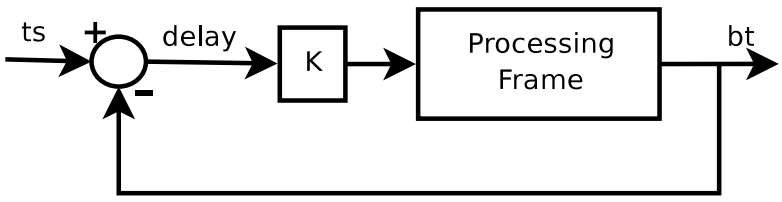


Figure 4.3: Diagram of blocks to control the base timer

In the figure 4.3 is shown a block diagram of the system. The difference between the base timer (bt) and the time stamp (ts) defines the delay.

4.3.3: Loading problem

The normal scenario to load in NIOBE for the KONVOI simulation is a long street with about 100 km. Moreover, buildings, trees and other objects are located around this street to make the environment look more realistic for the driver. Considering every element of this scenario, the virtual world is really huge and requires a large space in the memory. Another problem is to manage the big number of objects in the memory that requires CPU processing and decreases the framerate of the application. For these reasons, it was proposed some solution to improve the performance and save memory of NIOBE.

Solution using thread for selecting

One of the components of the world in Crystal Space is called sector, which contains the geometric objects. In the previous state of NIOBE, every object from the world was stored in just one sector. The idea of this solution is just to store in the current sector the objects near the camera, therefore just a small group of objects will be in that sector. However this does not mean that the application will consume less memory, since the function used to calculate the objects near the camera is implemented in the CS library, therefore, every object from the world must be loaded in the CS engine, but not in the sector.

This way, the solution considers just an improvement in the CPU processing, therefore the activation of this algorithm will be just used in small worlds without so many objects and a long street. The basic idea of this implementation is to add a thread in the process that calculates the objects close to the camera and add them to an alternative sector. After a specific interval of time (enough to calculate the objects around the camera), the CS engine switches the current sector to this alternative sector.

The explanation for a better performance of CPU is based on the render loop for a small sector, since the CS engine will just manage few objects stored in the memory.

Solution using sections

Another solution proposed considers using a new concept called section. This word was defined by the developers of this project and is not related with Crystal Space.

A section is a group of objects that are organized normally in agreement with their position in the world. The implementation of this solution involves Roadcraft and NIOBE.

On the NIOBE side, some functions were implemented to load a file generated by Roadcraft that contains information about the sections and the objects. The basic idea of this solution is to add in the current sector of the CS engine only the objects that belong to a section where the camera is located and its neighbours.

However, to perform each object in the engine appropriately, it is necessary to manage how the objects will be added in the CS engine. Every time the camera changes of section it is necessary to add new objects to the engine. If this action would be done at once, NIOBE would get a serious problem of performance on that moment, therefore it is created an intermediate buffer that allows just one (or a couple of) object to be added to the engine per frame. In this way, the objects from the current section and its neighbor will take some time to be added. For example, if the framerate of the application is constant and equal to 30 frames per second, in one second 30 objects will be added to the engine (considering 1 object added per frame). This rate is enough to have a dynamic driving world when the sections have an area of 500x500 meters and the velocity of the camera (or the vehicle) is less than 150 km/h, since the number of objects for each section is in average 100 objects for this project.

4.3.4: Mouse events

During the driving simulation the mouse events are not required, however these events are indispensable for the team responsible to build and edit the virtual world. Roadcraft tool was developed to build *a priori* the scenario, however, when it is loaded in NIOBE, sometimes it is necessary to change little parameters, such as the angle of rotation, the height or just to check the scenario in a 3D environment.

In the previous version of NIOBE, only the movement of the mouse was supported to assist the camera dislocation. The next step concerning mouse events is to provide mouse button events and tools combined with that to edit, move and identify objects in the world. The list of events added to NIOBE goes in table 4.2.

The first event in the table is really useful to identify the objects in the world, therefore it is possible to change some parameters of the object, such as the rotation and position. When it is known the ID of the object, it is possible to change it also in Roadcraft. The implementation of this functionality is based on the location of the

Mouse Event	Result
LEFT_BTN	Show in the console the ID of the object clicked
LEFT_BTN+KEY_SHIFT	Select the object clicked to move
LEFT_BTN+KEY_SHIFT+KEY_ALT	Select the object clicked to move using rubberband
RIGHT_BTN	Deselect the object

Table 4.2: List of mouse events in NIOBE

objects in the world. After rendered a frame, it is possible to view the objects in a picture or in 2D plane. When the user clicks on the screen, it is possible to obtain the position in the coordinate X and Y, however not in the coordinate Z. In this case, it is common to have more then one object in the same XY plane, but with different distance from the camera. For example, in the figure 4.4, the user clicks on the screen where the target is, in this case, for that coordinate XY, where it is located the truck, the barrier and the trees on background. To differ which object was clicked, it is calculated which object is closer from the camera, so the truck is the object clicked.

Figure 4.4: Screenshot of NIOBE to analyze a mouse event

The second event from the table is a great tool to edit the position and rotation of the objects. Sometimes, only the visualization from Roadcraft is not enough to position the object. To proceed correctly, the user must click on the left button from the mouse

in one object and keep the key SHIFT from the keyboard pressed. After that, the object can be moved and rotated using the keyboard. The third event has the same idea, the unique difference between them is that the camera stays static (for the second event) and just the object selected is moved. For the third event the camera follows the object like a rubberband.

The last event is just to return to the normal behavior of the system, which provides the movement for the camera using the keyboard.

4.3.5: Log register

This tool is important to get some information about NIOBE's performance. It is responsible to register in a file the framerate and the position of the camera for each frame. Created this file, it is possible to apply a mathematical treatment and generate graphics for analysis. Besides that, the position of the camera indicates the relation between the framerate and a specific part of the virtual world, since the framerate depends on the number of objects for that frame and how complex they are.

To activate/deactivate this tool some functions are created using the NIC language. The base function is called `registerfps`, and it requires a word action:

- registerfps on: to start the register;
- registerfps off: to stop the register;
- registerfps clear: to clear the information registered.

A script using OCTAVE program is implemented to work with the data stored. This script can plot graphics with information about the framerate and calculate some statistic values, like the average, minimum and maximum framerate.

4.3.6: New street type

Due to problems of performance, a new way to build the motorway was proposed to increase the framerate of the NIOBE application. The current algorithm uses several textures to build the street. The new idea is to create a unique texture and apply the UV mapping technique to define the final street. Another difference between the implementation is that the current one builds polygon per polygon the street, in a way

that the algorithm creates a huge file that contains all information about every polygon of the street and its relate texture. The new idea is to create a default model of 20 meters of the street using Blender, and based on this model (factory), change some characteristics to connect the peaces and shape the motorway.

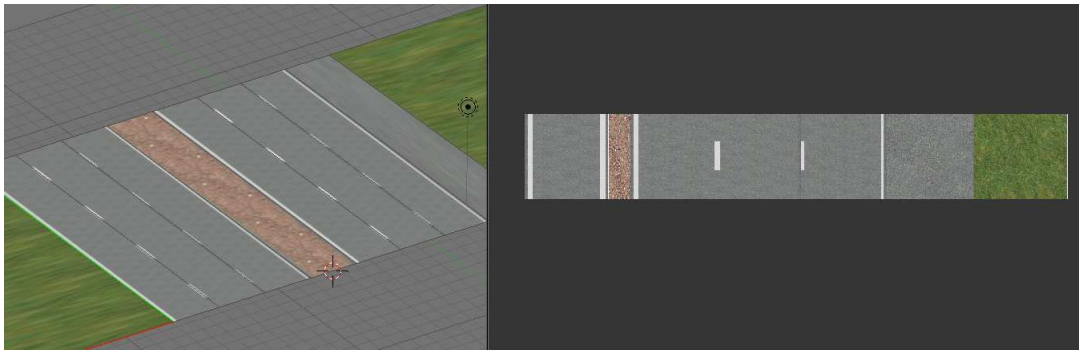


Figure 4.5: Factory from the street in Blender

This new factory is showed in the figure 4.5. Note that there is just an texture that contains every part of the street. It is known that Crystal Space manages better a object that contains just one texture. Those kind of objects are called *GenMesh*. The CS engine works faster when they work with this kind of objects instead of objects with multiple textures, called *ThingMesh*.

4.4: Roadcraft implementations

4.4.1: Section division

Roadcraft is responsible to define which objects will belong to each section. The sections are built considering a grade of rectangular cells with specific number of rows and columns. Each cell represents a section, and the objects that are inside it belong to the respective section. In figure 4.6 it is presented Roadcraft calculating the sections, which are represented by the green squares. In this situation it is possible to visualize the name of the sections and the objects.

Once the objects are ordered and the sections are calculated, Roadcraft must generate a file that contains all information, so that NIOBE can load it.

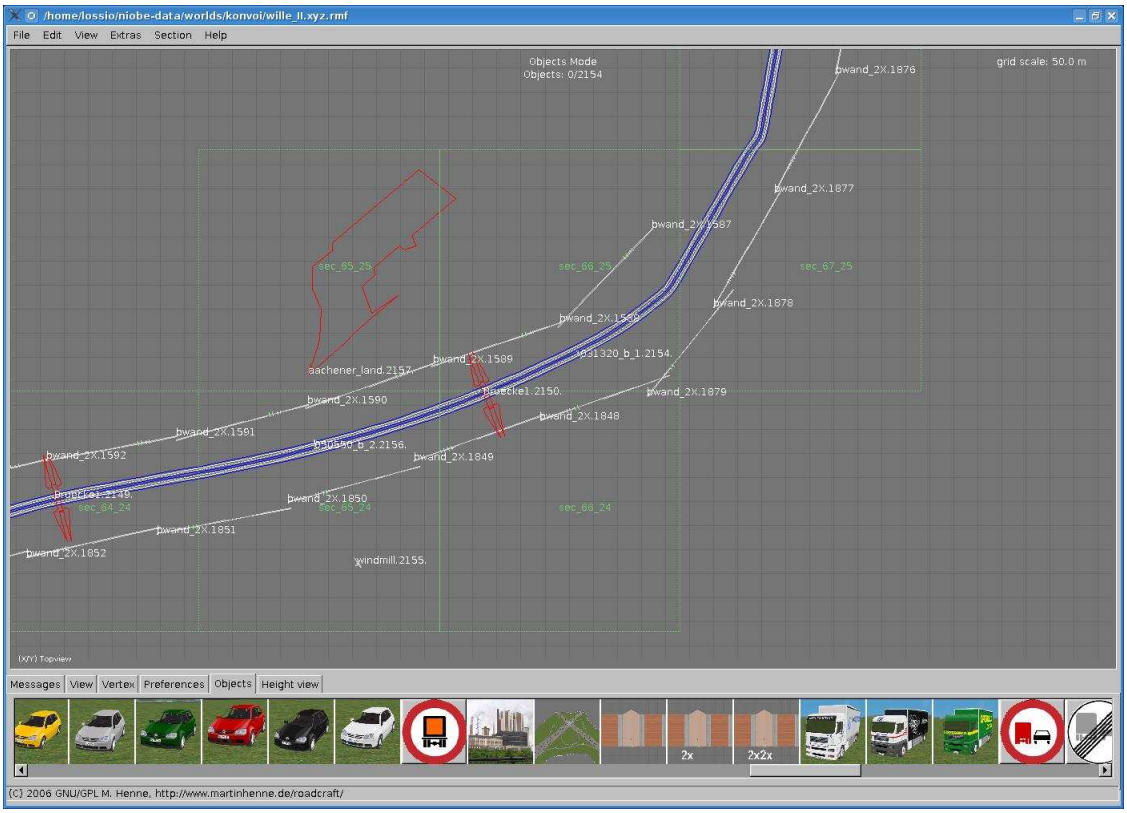


Figure 4.6: Section cells are represented in Roadcraft

4.4.2: Polygon representation

Every 3D object modeled in Blender has an outline. This word means a polygon representation of a 3D object seen by the top, but considering only its border. In the previous state of the system, the modeler should create its outline for every 3D model (factory). However, some of them were not compatible with the 3D model or they did not represent exactly how they are. That implies a wrong representation in Roadcraft of the objects in the world, since the program used these outlines to show how they are located.

Therefore, it was proposed to create a real outline for every object based on the real geometric modeled in Blender. In this way, the objects loaded in Roadcraft are represented with a great precision. The idea of this representation is to clip all polygons of a factory and get just the outline. To proceed with that, it is used a library called General Polygon Clipping Library (GPCL). This library is written in C language, thus the first task of programming is to wrap this library in a C++ class, since Roadcraft is written in C++ language.

When Roadcraft is executed, the program checks in a specific database the

factory objects and calculates their outline. However, this calculation requires time when the object has a large number of polygons. This problem is contoured when it is stored the outline of the object in that database, so that Roadcraft checks if there is already an outline created for that object. In positive case, the program will load it and will not perform any heavy calculation.

4.5: VirtualFIS

This program has to send and receive messages to interact with the driving simulation. The messages sent are responsible to reproduce the same effect that the real FIS makes in the convoy. The received messages aim to give the user some basic information about the trucks that belong to the convoy and about other vehicles.

The user of VirtualFIS can send the following messages to the PELOPS, which is responsible to change the state of the convoy:

- Coupling: when the driver wants to couple in the convoy;
- Decoupling front: when the first driver wants to decouple;
- Decoupling rear: when the driver wants to decouple with gap behind the vehicle;
- Confirm coupling: when a driver from the convoy allows a truck to couple in the convoy;
- Reject coupling: when a driver from the convoy rejects a request from a truck to couple to the convoy;
- Lane change: when a driver wants to change the lane in the motorway;
- Right/Left indicator: when a driver triggers the right or left indicator;

To provide important information for the driver or the operator, FIS and VirtualFIS show on the screen the distance of the truck in relation to other trucks of the convoy and how the current situation of the convoy is. Since VirtualFIS is supposed to run in a normal computer or a laptop, instead of a device on board of the cockpit, some functions are added only to VirtualFIS to assist the operator. One of these tools is a module map, similar to the one described on section 4.3.1. However, this map is dynamic, that means it is possible to zoom and move the map visualization.

Besides that, on the map it is possible to define which vehicles are shown on the screen. For the first four trucks, there are special buttons to show them, for the other cars, there is a list of checking box to select which vehicles are shown on screen. In figure 4.7 it is viewed the map module and its functionalities. The checking box called *Follow* indicates whether the map updater must follow the vehicle that belongs to this VirtualFIS or not.

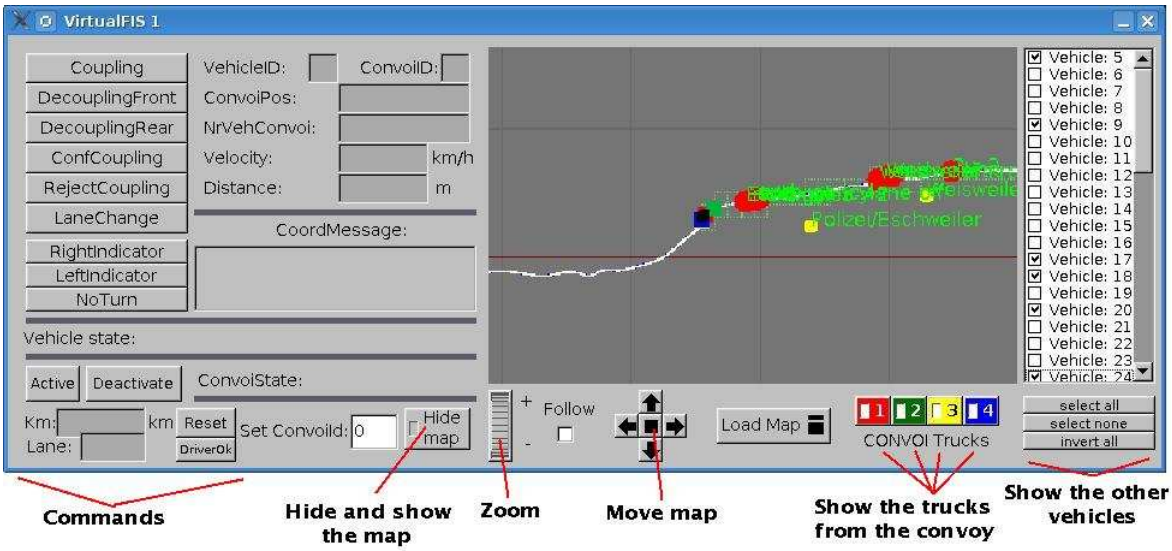


Figure 4.7: Screenshot of VirtualFIS with the map module opened

4.6: Forces to Pelops

This tool is created to influence the behavior of any vehicle during a simulation. These possibilities provided for the user concern with the velocity, position and the lane on which the vehicle stands. Basically, the user has to configure the parameters available and push a button to send the information to PELOPS. The same effect can be sent for more then one vehicle, therefore a checking box with a list of the vehicle IDs is created, so that the user can choose which vehicles will be forced. In figure 4.8 the interface components of this program are described.

Another useful tool for the simulation is an automatic jump of the vehicles. One of the testing defined by the psychologist team is to force the driver to drive the same part of the street several times. So it is possible to configure the position where the vehicles must jump and to where they have to go.

There is also an easy way to force a car to change the lane and break. This functionality is used when the psychologist team wants to observe the behavior of the

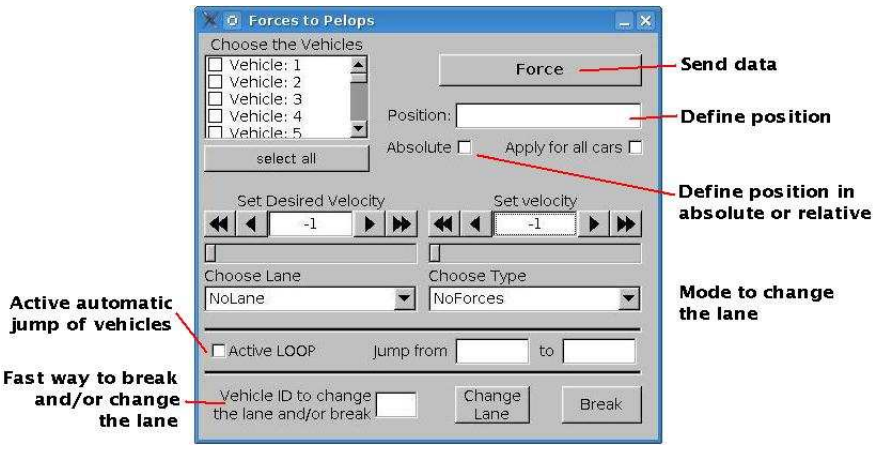


Figure 4.8: Interface of Force to PELOPS program

real truck driver in the situation when a vehicle breaks in his front.

4.7: Ampex Mark IV

Ampex Mark IV is a tool that works like a player and a recorder of information sent or received from the modules in the driving simulator. The player mode is used to emulate the PELOPS communication. The idea is to load a file that contains information of the simulation and send to the NIOBE program or/and to other tools. A screenshot is presented in figure 4.9. The interface is very similar to a normal CD-player, for example, the eject button is to open a data file, the slider provides an easy way to run forth and back the time and the three common buttons that every player demands - RUN, PAUSE and STOP. It is possible also to configure the IP and the port to be used.

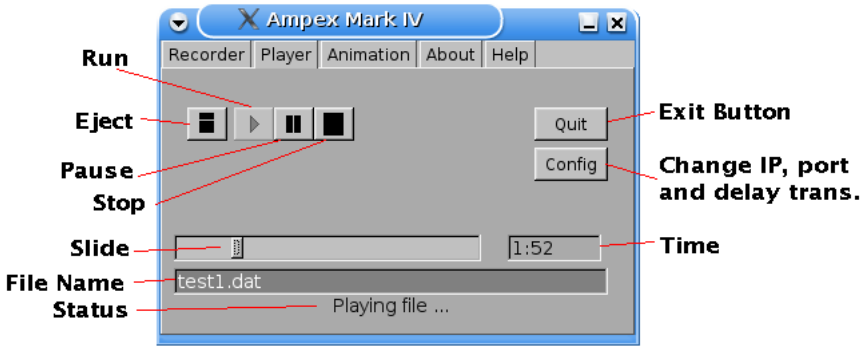


Figure 4.9: Ampex Mark IV setting as a player

The recorder mode is used to record the information broadcasted by the PELOPS application or even by the Ampex Mark IV player. The data is stored in a

temporary file during the recording. At the end of the simulation, the user can create a file that contains the whole data listened from the network. In figure 4.10 the recorder options are shown.

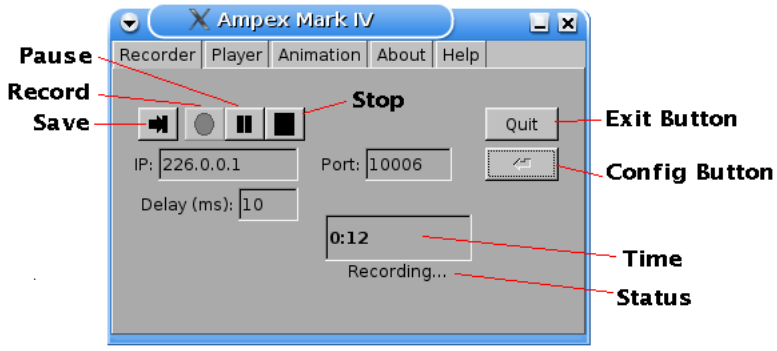


Figure 4.10: Ampex Mark IV running as a recorder

In addition to the recorder and the player, this software provides an animation for observing the velocity and position of the vehicles. This tool does not influence in recording nor playing. In figure 4.11 it is presented a screenshot.

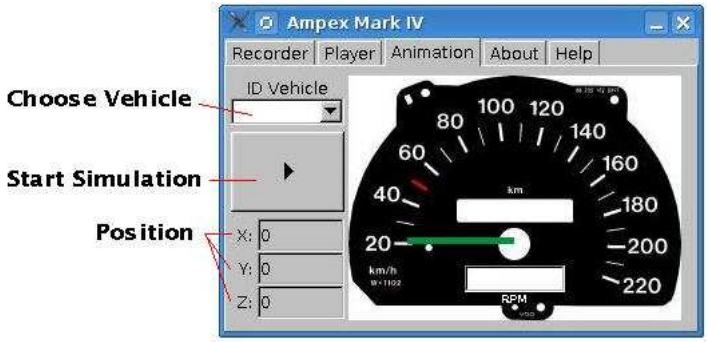


Figure 4.11: Ampex Mark IV running the animation mode

For the player module, an important functionality is implemented for testing. When the application is started, it is possible to provide as argument a file that contains data information to be sent, a desired time for simulating and an integer to decide how many times the simulation must be repeated. The flowchart from the figure 4.12 represents the idea of this new tool:

This chapter described the contribution by the academic for the project. Some programs were entirely built, as Ampex IV and Forces to PELOPS. Other implementations were improvements and creation of new functionalities in programs like Roadcraft, NIOBE and VirtualFIS. The next chapter discusses about another project involving the driving simulator are commented.

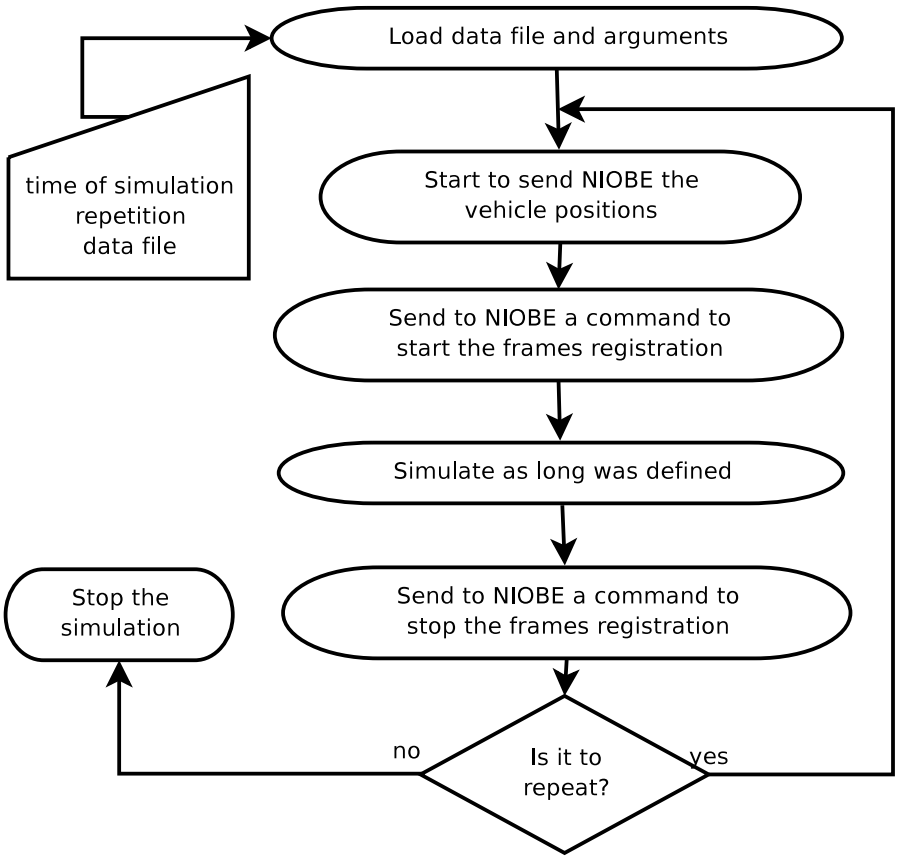


Figure 4.12: Flowchart for the test simulation

Chapter 5: Demonstrator Project

The current hardware of the driving simulator for the KONVOI Project requires a real cabin of a truck and demands two beamers and two LCD monitors to represent the rear views. This combination of hardware is not portable to show the functionality of the project for special events, as conferences and public demonstrations, due to the dimension, complex structure and lack of assembly strategies, since the transportation and assembly of the simulator is demanded.

This chapter describes the work done by the author to create a demonstrator of the KONVOI system that allows the demonstration of the basic functionalities provided by the current version of the simulator. The idea is to use a simple, flexible and portable hardware setup that can be assembled and disassembled quickly.

The first activity is to define possibilities of hardware and software for this demonstrator. After that, the options of configuration are analysed. Each option is judged by a list of parameters that considers costing, dimension, timing, available hardware/software and human resource. Once defined the configuration, the activities of developing are planned, which are identified as a group of problems to be solved, for example, an electronic, algorithm and communication problem. The demonstrator uses the final modules already implemented in the current simulator, like NIOBE and PELOPS applications. However, some possible modification of compatibility with the new hardware must be done.

The final demonstrator provides a perfect platform to show the KONVOI system for any event outside from the institute for people interested on, specially in conferences and open public events outside from the institute.

5.1: The modules

The modules of the demonstrator can be divided into four sections in agreement with the hardware used and the software already developed. They and their elements and tasks are listed as follows:

- PELOPS

  - Traffic simulation;
  - Process driving data;

- Modules on Network
  - Visualization (NIOBE)
  - Driver Interface (FIS)
  - Operator

- Driver Station
  - Steering Wheel;
  - Pedals (brake and gas);

- Data Converter

Figure 5.1 shows how they are connected. The elements of the Network module are already implemented, so it is possible to reuse them. The PELOPS module can be also reused, however the other two modules - the driver station and the data converter - must be implemented as well as the communication between them.

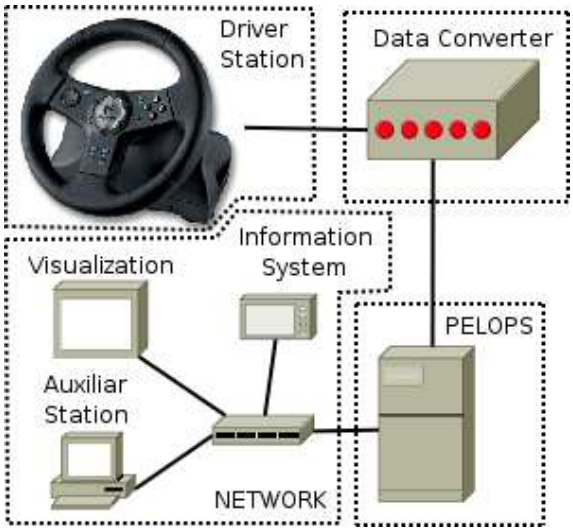


Figure 5.1: Module diagram of the Demonstrator Project

5.1.1: Driver Station

This module is responsible to make the communication between the driver and the system. It is constituted by a steering wheel with force-feedback and pedals, that

Figure 5.2: Example of the driver station

includes gas and brake. In the driver station it is possible to find some buttons as well, where the driver can use as indicators left and right. Figure 5.2 shows an example of a driver station that includes a steering wheel and pedals. This kit of driving is provided for computers or video-games.

5.1.2: Data Converter

This module is responsible to interpret the signals from the driver station and convert them to a treatable data for PELOPS. Basically, this module can be a microcontroller or a computer program to process and convert the signal to CAN bus, which is the protocol used by PELOPS. Three possibilities to implement it are presented, as follows:

- The first one considers to have a microcontroller that supports USB (typical connection of the driver station) and CAN Bus (for PELOPS);
- The second possibility is to have a microcontroller that supports CAN Bus as well, nonetheless the connection between it and the driver station would be done by the developers. This means that electronic circuit inside the driver station will be redone;
- The third possibility is to create a library to be linked to PELOPS that interprets the signal from the driver station directly connected by USB to the computer which PELOPS runs.

5.1.3: Analysis of possibilities

Three possibilities are established according to the possible kind of data converters defined in the previous section. Table 5.1 presents the analysis considering the timing, costing, complexity and external difficulties. The items vary from 0 (the lowest) to 5 (the highest).

Alternatives	Costing	Timing	Complexity	External Difficulties	**Total**
First	5	3	3	0	11
Second	5	4	4	2	15
Third	2	2	3	3	**10**

Table 5.1: Grade of analysis for the Demonstrator

The timing considers time of studying and developing. Another item, the costing, considers how much will be spent to buy the new hardware. The complexity involves the human resources available and their knowledge. And the last item considers a dependency of working from other people that do not belong to the institute.

Table 5.1 says that the third possibility presents the best criterions, however these do not have the same weight of analysis. For example, the third option requires an important modification in PELOPS that depends on the developer team from another institute. Other negative point of this solution is that the team leaders from the ZLW-IMA institute want to have know-how of microcontrollers used in the automobile area. As the most common microcontrollers in the automobile area do not support USB, the option chosen by the team is the number two.

5.2: Technical details

Considering the solution mentioned in the previous section that includes a microcontroller to process the signal from the driver station to the PELOPS machine, on the next section, technical information about this device is described.

5.2.1: Microcontroller

The chosen microcontroller to be used in the data converter is the C167CR by Infineon. This company is a leading innovator in the international semiconductor industry. They design, develop, manufacture and market a broad range of semiconductors

and complete system solutions targeted at selected industries. Some of their products serve applications in the automotive area.

The C167CR are high-end members of the Infineon full featured single-chip 16-bit microcontrollers. High CPU performance is combined with peripheral functionality and enhanced I/O-capabilities. A wide variety of on-chip features such as large on-chip ROM, multi-functional standard peripherals, and application-specific peripherals (e.g. optional CAN) is available. The C167CR features an on-chip CAN module which has been designed to fulfill the requirements of automotive and industrial control applications[18].

5.2.2: Development kit

For facilitating the development using the microcrontoller mentioned in the previous section a development kit was ordered. This kit contains a single board computer module (that includes the microcontroller) mounted on a applicable carrier board that features all hardware needed for immediate start-up of the module[17].

This development kit is supplied by PHYTEC company. The main features from the single board computer are listed below:

- Credit card-sized (85x55 mm) SBC;
- Infineon C167CR controller on-chip Full CAN 2.0B;
- 256 KB (to 2 MB) external SRAM;
- 256 KB (to 2 MB) external Flash;
- Flash supports on-board programming via RS-232 interface;
- Up to 1 MB optional EPROM;
- All controller ports & signals extend to standard-width (2.54 mm) pins aligning three edges of the board;
- 16-channel A/D-converter with 10-bit resolution;
- RS-232 transceiver supports two serial interfaces;
- Full 2.0B CAN interface.

And the carrier board that receives this single board computer has as main features:

- Simple jumper configuration allows use of the Carrier Board with all 5V PHYTEC micro- and miniMODUL Single Board Computers;
- Pin header receptacles accommodating both micro- and miniMODULs;
- Wire wrap field (60 mm x 65 mm) supports development of user-designed circuitry;
- DB9-socket for RS-232 interface;
- Second DB9-plug which can be configured as a CAN or RS-485 interface according to user needs and the underlying controller;
- Reset switch;
- Boot switch;
- VG96-connector;
- Single power source via a low-voltage socket.

Chapter 6: Tests and Results

In this chapter the results and analysis of several tests are commented. The sections are divided in agreement with NIOBE performance (framerate) and memory management. Some results for Roadcraft are commented as well.

6.1: Framerate testing

This testing is concerned with the performance of NIOBE, which is an important parameter to add new effects to the virtual world, like shadows, lights and animation. The realism of the environment depends on the performance, that is related to the framerate. Therefore, this parameter is the object of analysis as well as the investigation of the components that influence it.

6.1.1: Tools for testing

The suite of tools used in these testings consists in simulating the system several times with different configurations for the scenario loaded in NIOBE. However, the first task for the testings is to record the action of a real driver during an interval of time defined in 10 minutes. This recording is the base for analysing, and the scenarios will be always tested with it. The tools used for testing simulation are:

- The player to substitute PELOPS and to have always the same information sent;
- The log register in NIOBE to store the data for analysing;
- Octave scripts to analyse the data;
- Different configurations of the KONVOI scenario.

6.1.2: Different configurations

In each scenario it is possible to change several conditions before starting the simulation. As the objective of this testing is to identify the components that influence the framerate, a specific configuration is changed or removed to notice its influence. The different tested configurations are the street, the objects and the traffic.

Each testing is done more than once for verifying the robustness of the results. The algorithm that limits the frames per second is set as a high value, so the framerate can oscillate freely. In addition to the graphics, some statistical results are computed in the table 6.1.

Configuration	Maximum framerate	Minimum framerate	Average
Normal	83.33	6.37	47.62
	83.33	13.70	47.62
	83.33	9.01	45.45
No street loaded	111.11	34.48	71.43
	100.00	22.22	76.92
Small street loaded	100.00	10.20	52.63
	100.00	12.35	55.56
No objects loaded	100.00	15.87	58.82
	100.00	18.87	58.82
No sky loaded	83.33	4.25	45.45
	83.33	10.87	45.45
No lights on the car	90.91	5.21	47.62

Table 6.1: Table with different configurations

Comparing the results to the normal configuration, the street and the objects are the main components that influence the framerate.

6.2: Visualization Results

It is known that humans can not distinguish framerates over 30 frames per second[15]. Based on this affirmation, the framerate in NIOBE is limited to 30 fps. That means any significant variation of framerate in the application would be perceived by the driver. Therefore, the graphics shown in the annex C must have the minimum value set to more than 30 frames per second, so that the movement during the simulation of the dynamic objects would be smooth.

Another issue concerns the trembling algorithm explained in the section 4.3.2. The results achieved with its implementation can be noticed in the simulation. As the feeling of this performance is more like a matter of visualization, it was not created a tool to calculate the degree of improvement of this algorithm.

6.3: Solution using a selector thread

The implementation described in section 4.3.3 to solve the loading problem in NIOBE presented a significant improvement in the simulation. However, as the scenario was being incremented with new objects, the memory became a problem and even the performance. In this way, this solution could be only used for small scenarios with a short street with few objects. However, another problem was found that discarded the use of this implementation. The thread that calculates the objects near the camera is not safe, that means the application can crash if the main thread and the selector thread access at the same time the same space in the memory.

6.4: Section Management

The implementation of this algorithm saves a huge space in the memory, since every object in the world does not have to be loaded. The performance of NIOBE improved as well. Just considering the objects, it was possible to save more then 39% of RAM memory used by NIOBE when it is started.

Analyzing the KONVOI scenario, the following parameters are determined in table 6.2.

	No objects	**With Section Management**	**Without Section Management**
Memory used by NIOBE (MBytes)	65	110	180

Table 6.2: Table with memory used of NIOBE

There are no graphic results that show the gain of performance between a version without and with section management for the objects. However, after the objects are subscribed in the section management, the street, which was before loaded at once in the memory, follows the same strategy. In this case, some testings were done to show the difference concerning the framerate for the street on the section management. In the annex D, it is shown this difference. For these testings, the framerate was registered for each processed frame, while the tests from the annex C were done by each 10 frames. The performance is also better due to the compilation of CS, because before it was compiled in debug mode, that dicreased the framerate of NIOBE. In the new testings it is used CS in optimize mode, therefore causing a big difference among

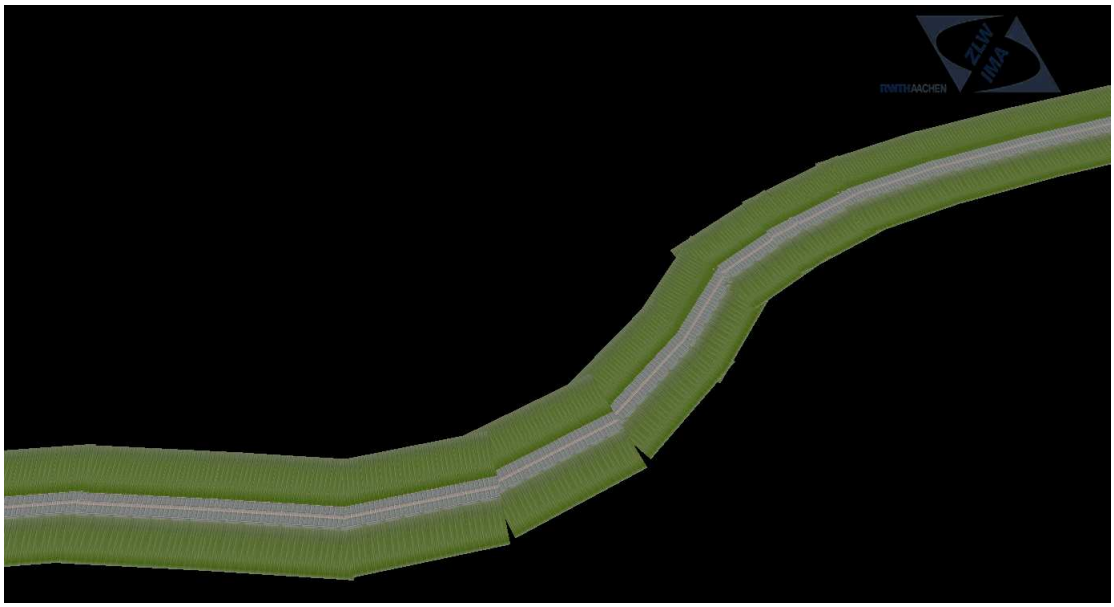


Figure 6.1: New motorway using a common factory

the values obtained. Another modification is concerned with the graphic generated by the script that shows only a point instead of connecting lines.

6.5: New street

To compare the current street and the new street, which uses an unique texture instead of many, a simple motorway that does not contain the safety fence was built. As the safety fences are presented during the whole motorway, they influence a lot in the framerate, therefore the results are very different from the normal scenario.

It is important to point out that the new streets used in this test are not the final ones. They are built always with the same factory, in a way that the streets are not connected. In figure 6.1 there is picture of the street. The idea for this test is to analyse how much the performance can be improved using an unique texture.

The results are shown in annex E. The new solution caused an enhancement of 16% in the framerate average. However, the minimum framerate from this new street is smaller than that of the current street and even in the graphic, the distribution of the framerate is more uniform.

6.6: New Outline

The new representation of the outline in Roadcraft brings a new dimension to build the world with a great precision, since the modelers can fix the objects faster and exactly in the world. However, this new implementation requires more CPU processing power, and the performance of Roadcraft is not good when there is a big concentration of objects in a small area. This density is not high for the traditional scenarios that are developed for the current simulation, so Roadcraft runs well. Anyway, this problem can be contoured by deactivating the outline representation, so that the objects in Roadcraft would be shown like a simple polygon. In figure 6.2 the new and old outline are showed.

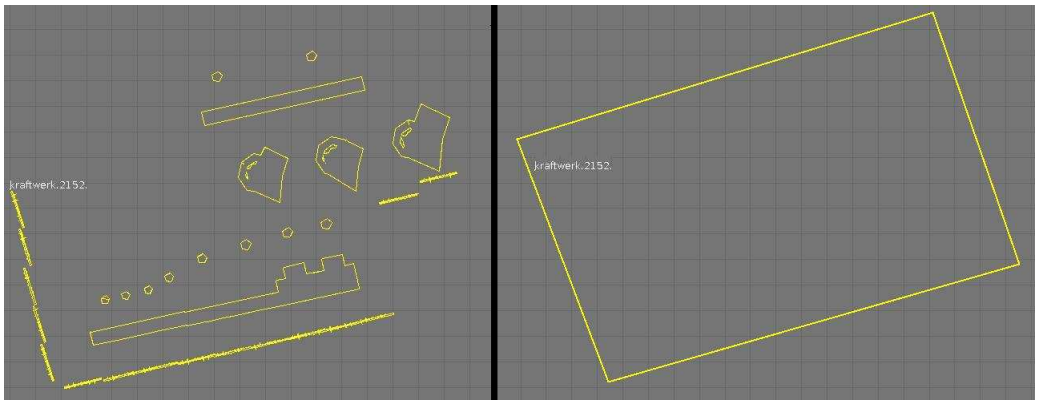


Figure 6.2: The new outline and the old one

The real object modeled in Blender is shown in figure 6.3. Every border of the this object' construction can be visualized in the new outline.

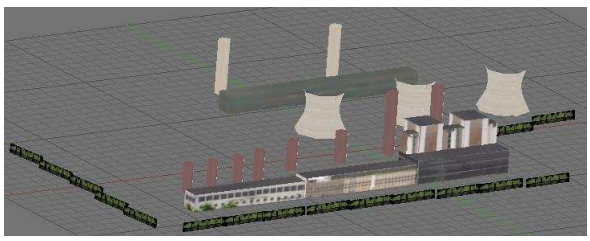


Figure 6.3: Real object modeled in Blender

6.7: New package tools for simulation

During the simulation, it is normal to have problems when performing the maneuvers between the trucks. For example, the desired truck for coupling is faster than the real driver or there is a car between two trucks that disables a coupling.

The combination of some new tools created can be applied to solve these kind of problems during the simulation. For example, the tool *Force to Pelops* (section 4.6) is used to influence the vehicles. For such, it is necessary to know the ID of the vehicle that would be influenced. To obtain its ID, the operator can run NIOBE in a local machine (the same where *Force to Pelops* runs), and use the mouse button click to identify which vehicle is on the screen. Another way to obtain the ID of the vehicles is by running VirtualFIS with the map module opened.

With these new tool packages, the testers can quickly reach the desirable scenario for the driver of the simulator. Before, they would have to restart the system, or wait a long time of simulation to contour those problems.

Chapter 7: Conclusions and Perspectives

The intelligent convoy system of trucks in motorways is a complex structure with several technical and non-technical variables. The integration of various segments of research in several areas is just made possible through a multidisciplinary environment, that integrates engineers, sociologists, technicians and psychologists.

The construction of a simulator of the whole system is a crucial task to guarantee the robustness of the project. The results of straight testings in real motorways could be catastrophic without analysing and interpreting the behavior of the driver and the electronic system embedded in the truck. Until the end of the conclusion of this end-course project, the team of testers is still using the driving simulator to study the consequences of this system in the real traffic, as well as how the truck driver would interact with the system.

To ensure the realism of the driving simulator, several tests and investigation were done to improve the performance of the NIOBE application, since it is the main visualization component for the driver. Therefore, a creation of a framework for testing was necessary to be implemented, making it possible to identify clearly the elements that influence in the simulation and to measure the gain of performance.

This gain of performance is related to the memory and CPU processing. One of the significant improvements of performance concerning with memory was to develop a dynamically loading process of the objects in the world. By means of implementing the idea to create sections in the virtual world, which involves a group of objects organized by their location, it was possible to reduce 39% of the memory used in NIOBE application for the current scenario of testing. That same idea of sections was implemented for the street, which reached a better average of framerate, an improvement of 12%. Besides that, the minimum framerate increased as well, which is another important factor.

Another solution to solve the problem of performance, the one by using threads instead of sections, was rejected, since the application can crash due to the non-thread-safe implementation. Despite getting a significant visual improvement during the simulation, this solution would not work well for the final scenario for testing, that uses a large space of memory. Therefore, trying to become the application thread-safe was discarded as well.

An attempt to improve the performance of NIOBE was to substitute the current street - that uses several textures - by a new street that contains just one texture. Analysing the results, one should notice a significant improvement in the framerate average. Nevertheless, an important factor is the minimum framerate as well, and the new street proved to be worse than the current street when considering the distribution of minimum framerates during the simulation. Hence, this solution was rejected and the current street was kept.

One of the main problems during the initial stage of the driving simulation was the way that NIOBE updates the dynamic objects (vehicles). The responsible element for the physical events of the simulation is the application PELOPS, which sends to NIOBE messages through the network that contains information of where the vehicles must be located. So far, NIOBE just updated the vehicles position using the raw values stored in these messages. The result was a terrible trembling in the vehicle movements. After a mathematical treatment that considers the time clock of NIOBE machine and the time stamp of each position received, the vehicles' movements are now smooth and realistic.

Prior to simulation, there is an important phase that aims to build the scenarios. A scenario is constituted by a street and objects surrounding it. To build the street and include the objects in the virtual world, the modelers, who are responsible to create the scenarios, must use the application Roadcraft. This stage of the project demands a long time to be concluded, since the scenarios for testing contains motorways with more than 50 kilometers. For this reason, several functionalities were implemented to accelerate and facilitate their work.

One of those functionalities that speeded up the construction of the scenarios is the implementation of the real outline for the objects viewed in Roadcraft. As the old outlines usually were a rectangle that involves the object, consequently, it was hard to locate correctly the objects in the world. Besides that, the modelers should use NIOBE as editor, and move the objects in a 3D environment. With the real outline of the objects, the modelers rarely need NIOBE to adjust them, only when the object requires to be rotated on the angle that is not possible using Roadcraft, since it allows just one rotation direction.

Considering now the simulation running, several tools are used for monitoring, controlling, recording and playing. The VirtualFIS showed an important tool to evaluate the state of the convoy, as well as a monitoring tool due to the new map module imple-

mented on it. Another important controlling tool is *Force2Pelops* that allows influencing the behavior of the virtual drivers. This new package used during the simulation provides a powerful mechanism to analyse how the real driver reacts for different situations during the driving.

The whole package of implementation done by the academic contributed to improve the interactive driving simulator, which is an important segment of the project KONVOI, which provides a testing platform. Parallel to that, a new project was started, which aims to develop a small and flexible driving simulator called demonstrator. The main goal of this project is to provide a system to demonstrate in conferences the main modules of the project. Another motive is to learn how to use a typical microcontroller common in the automobile area.

Basically, the main technical task implemented in the demonstrator was to interpret the signals coming from the driver station and convert it to the CAN bus protocol. As the microcontroller already supports CAN bus communication, the complexity was reduced drastically and the other modules responsible for simulation are the same of the real simulator.

As perspective for the future, the performance of NIOBE is still an object of study and investigation. The improvement and optimization of this application are essential, therefore new functionalities in the visualization could be possible to implement and the realism of the simulation would be increased. To complement this investigation, an upgrade of the tools for testing must be requested, providing more variables for analysing and generating conclusions.

References

[1] Tiltmann, T.; Friedrichs, A. *Automated truck-trains on motorways - vision or reality?*, Aachen, 2005. Article from ZLW/IMA, RWTH Aachen University.

[2] Preuschoff, E.; Friedrichs, A. *Kombinierte Fahr- und Verkehrsfluss-Simulation*, Aachen, 2004. Article from ZLW/IMA, RWTH Aachen University.

[3] Forschungsgesellschaft Kraftfahrwesen Aachen. *PELOPS Project* ,at `http://www.pelops.de`. Access in November, 2006.

[4] Manual On-line. *Crystal Space Project*, at `http://www.crystalspace3d.org`. Access in September, 2006.

[5] Henne, M. *Indrives Handbuch*, Aachen, 2006. ZLW/IMA, RWTH Aachen University.

[6] *Blender Documentation*, at `http://www.blender.org`. Access in September, 2006.

[7] *Gimp Documentation*, at `http://www.gimp.org`. Access in September, 2006.

[8] *Blender to Crystal Space Plugin*, at `http://b2cs.delcorp.org`. Access in September, 2006.

[9] *Octave Documentation*, at `www.octave.org/`. Access in October, 2006.

[10] *FLTK Documentation*, at `htpp://www.fltk.org`. Access in August, 2006.

[11] *OpenGL Project*, at `htpp://www.opengl.org`. Access in December, 2006.

[12] *CVS Documentation*, at `http://www.nongnu.org/cvs/`. Access in August, 2006.

[13] *A General Polygon Clipping Library*, at `http://www.cs.man.ac.uk/~toby/alan/software/gpc.html`, Access in August, 2006.

[14] *POSIX Threads Programming*, at `http://www.llnl.gov/computing/tutorials/pthreads/`. Access in August, 2006.

[15] *The Facts about Games and their Frames Per Second*, 2006. Article from Tech-Connect Magazine.

[16] *CAN Protocol*, at `http://www.kvaser.com/can/protocol/`. Access in February, 2007.

[17] PHYTEC Technology Holding Company. *miniMODUL-167, Hardware Manual*, edition August 2002.

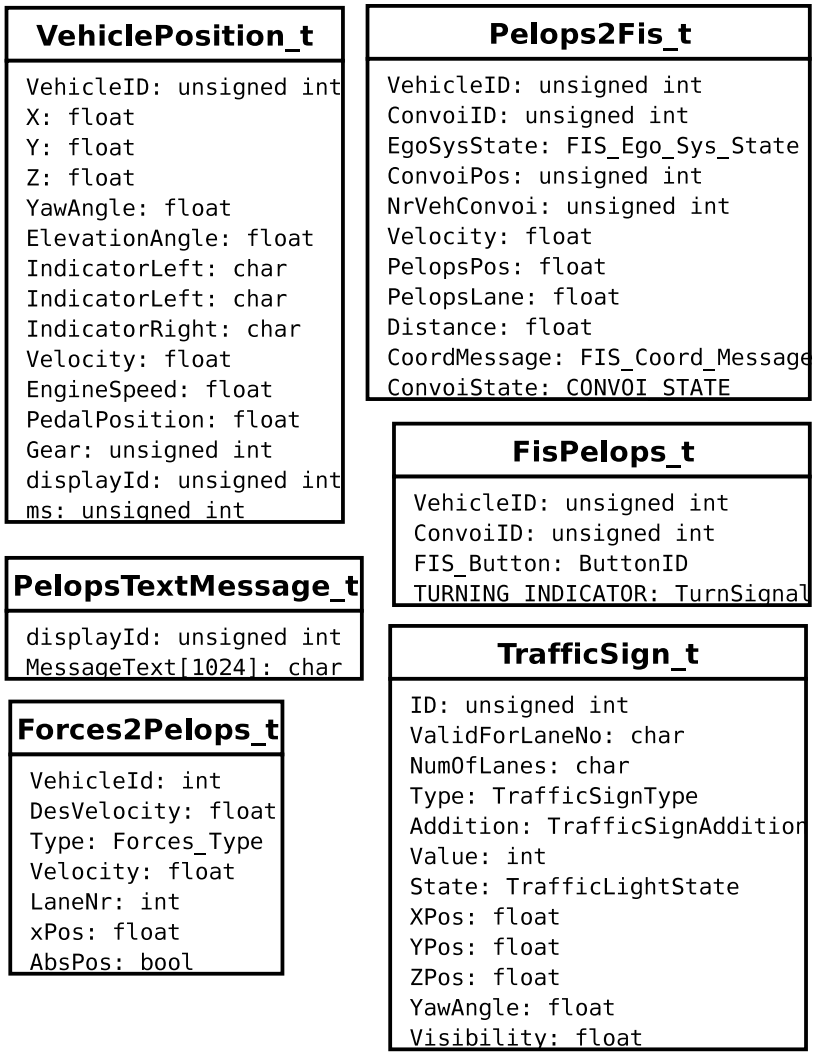
[18] Infineon Technologies. *User's Manual, V3-2, C167CR Deviratives*, edition May 2003.

Annex A : List of Commands from NIOBE

```
actionsperframe <int, default: 2>      // set number of actions to be taken per frame
actiontest [<action>] [<args>]         // test action (developers only)
addsection <name> <x> <y> <z>          // create section
ambientlight <red> <green> <blue>      // set ambient light
bind <key> <command>                   // bind key to command
camid <id>                             // id of the car, the camera is tied to
camposrel <x> <y> <z>                  // move campos relative
camposrel+ <x> <y> <z>                 // add xyz to relative campos
campos <x> <y> <z>                     // set absolute campos
camrotrel <x> <y> <z>                  // rotate camera relative
camrotrel+ <x> <y> <z>                 // add to camera rotation angle
camrot <x> <y> <z>                     // set absolute camera rotation
cat <filename>                         // like bashcommand 'cat'
cinthread                              // start console input thread
clearmessage                           // clear textmessage on screen
countobjectssof <sectionid>            // count objects in section
createmesh <factoryname> <meshid> [pos="0 0 0"] [sector="room"]
                                       // create mesh object
culling d|dyvanis|f|frutsvis           // set the culling as dyvanis or frustvis
day                                    // activate day settings
delete <object-id>                     // remove object from engine
drawmessages on|off                    // switch for messages
enginefreqbase <float>                 // default: 0.7 (frequency factor=base+rpm/divider)
enginefreqdivider <float>              // default: 18000 (frequency factor=base+rpm/divider)
enginesound on|off|toggle              // switch engine sound on and off
enginevolumedown                       // tune engine sound volume
enginevolumeup                         // tune engine sound volume
enginevolume <value> # 0.0 - 1.0       // tune engine sound volume
eoc                                    // end of input console
error <text>                           // error message
etime                                  // get elapsed time in ticks
exec <commandfile>                     // exec file with niobe commands
execif <id> <cmd>                      // execute cmd if message receiver id is 'id'
factorycount                           // count factories in sector
factorylist                            // list factories in sector
farplane <value>                       // set farplane for renderer
fog color <red> <green> <blue>         // adjust fog
fog density <value>                    // adjust fog
fog on|off|toggle                      // fog on or off
fovrelative <value>                    // field of view in degrees, relative
fov <value>                            // field of view in degrees, absolute
freelook on|off|toggle                 // set camera freelook mode
getcampos                              // where is the camera?
getniobeid, setniobeid <id>            // what ID am I running at
getsections                            // get amount of existing sections
globalsection [on|off]                 // activate/deactivate objects in global section
headlight on|off|toggle                // headlights (deprecated, may crash)
help [command]                         // help on particular command
invertMouse                            // invert mouse y-axis
invisible <id>                         // set object to be invisible
keeppelopsthreadrunning|kpr 1|0        // pelops listener on and off
keymoved <id>                          // set object to get moved by keyboard
keys                                   // show key bindings
leadercam [on|off]                     // if on, camera id follows leading convoi truck
lightradius <id> <radius> [setup]      // radius of headlights
lightmeshnames <id> <head> <break> <left> <right> // set id's of extra light meshes
limitfps <maxfps or 0>                 // limit frames per second or not (0)
listobjectsof <sectionid>              // show a list of this sections objects
listsections                           // list existing sections and its contents
loadlib <filename>                     // load xml cs library file
loadpackedlib <filename>               // load packed library file
loadrmfobjects <filename.rmf>          // load objects from rmf file
loadrmfsections <filename.rmf>         // load sections from rmf file
loadworld <filename> [dir]             // load xml cs world file
logo on|off|toggle                     // toggle zlw/ima logo on/off
look north|south|east|west|up|down     // make camera look to direction (if in freelook mode)
looknorth                              // use 'look north'
map load <filename> <sizeX> <sizeY> <OriginX> <OriginY> <Dir>
map on|off|toggle                      // overlay map
map track <id> <color>                 // activate object as a spot on overlay map
map untrack <id>                       // deactivate object on overlay map
meshcount                              // count meshes in sector
messageid <id>                         // get/set message receiver id
mirror on/off                          // mirror view (actually 'off' does not work correctly)
mirrorrotangle <roll angle>            // rotate mirror
mousecapture on|off|toggle             // capture mouse
move <id> <x> <y> <z>                  // move object to absolute position
movekeys on|off|toggle                 //
movespeed <value>                      // set speed for keyboard movement
msgact <sec> <text>                    // test action queue with a message action
night                                  // activate night settings
niobeid <id> <command>                 // execute command on niobe with id 'id'
showpointscs <file> <factory=genpoint> // show points from ascii file in cs coords
showpointsrc <file> <factory=genpoint> // show points from ascii file in roadcraft coords
send2niobe <id> <command>              // send command to niobe with message receiver id
ob2sec <obj-id> <sec-name>   UNFINISHED          // tie object to section
ob2sec all                             // assign all objects to sections that have been assigned by roadcraft
```

```
parentofchild <parent> <child>         // set parent <-> child relationship
play <soundfile> | HORN_SOUND          // play soundfile, must be loaded
precache                               // precache all objects, handle with care
prepare                                // handle with care
quit                                   // leave the simulation
readrmf <filename>                     // read roadcraft meta file
registerfps on|off|clean               // register fps in a file fps.log
relight world|sector                   // relight sector or world, handle with care
rereadcampos                           // re-read denso camera positions (outdated)
roadcraftlib <object name> <directory> // load library in roadcraft directory structure
rotate <id> <x> <y> <z>                // rotate object
r | repeat | again                     // repeat last niobe command
rubberband elasticity|stiffer|softer   // adjust rubberband camera
rubberband set <x> <y> <z>  | show     // adjust rubberband camera
rubberband up|down|left|right|forward|back
sectionlist                            // list existing sections
sectionstatus                          // debugging info about the section management
send GraphicEngineReady | Success | Ready // send net message (netmsg.cc)
send2niobe <id> <cmd>                  // send niobe command to be executed by niobe with message receiver id
sequence <sequence name>               // run cs xml sequence (must be loaded)
setasvehicle <id>                      //
setmessage <seconds> <text>            // set a message to be shown on screen
setperiod <period>                     // set minimum period between two frames (to limit fps)
setrpm <float>                         // engine rounds per minute - affects engine sound
showactivesections                     // show a list of currently active sections
showmousecmd                           //
showneighborsof <sectionid>            // show a list of this sections neighbors
showneighbors                          // show a list of all sections neighbors
sleep <seconds>                        // wait some seconds before executing next command
slow <value>                           // set impact of CTRL key to movement
speedboost <value>                     // set impact of SHIFT key to movement
startthreads                           // start concurrent threads
system <command>                       // execute bash command
trafficlight <id> off|red|redyellow|green|yellow|nextstate
tremblecode on|off|toggle              // (De)activate the tremble compensation algorithm
van <id> create                        // create a van (with lights)
van <id> lefton|leftoff|lighton|lightoff|righton|rightoff|breakon|breakoff
vehicle <id> create <factory>          // must be a thing mesh
vehicle <id> |lefton|leftoff|righton|rightoff|breakon|breakoff|headon|headoff
viewcenterleftrelative <value>         // move perspective view center (0=left ... 1=right)
viewcenterleft <value>                 // move perspective view center
viewcentertoprelative <value>          // move perspective view center
viewcentertop <value>                  // move perspective view center
visible <id>                           // set object to become visible (if invisible)
warning <text>                         // show warning message
whereis <object-id>                    // show object position
writermf                               // write currently loaded rmf file
```

Annex B : Structures of the Network Messages



Annex C: Framerate analysis

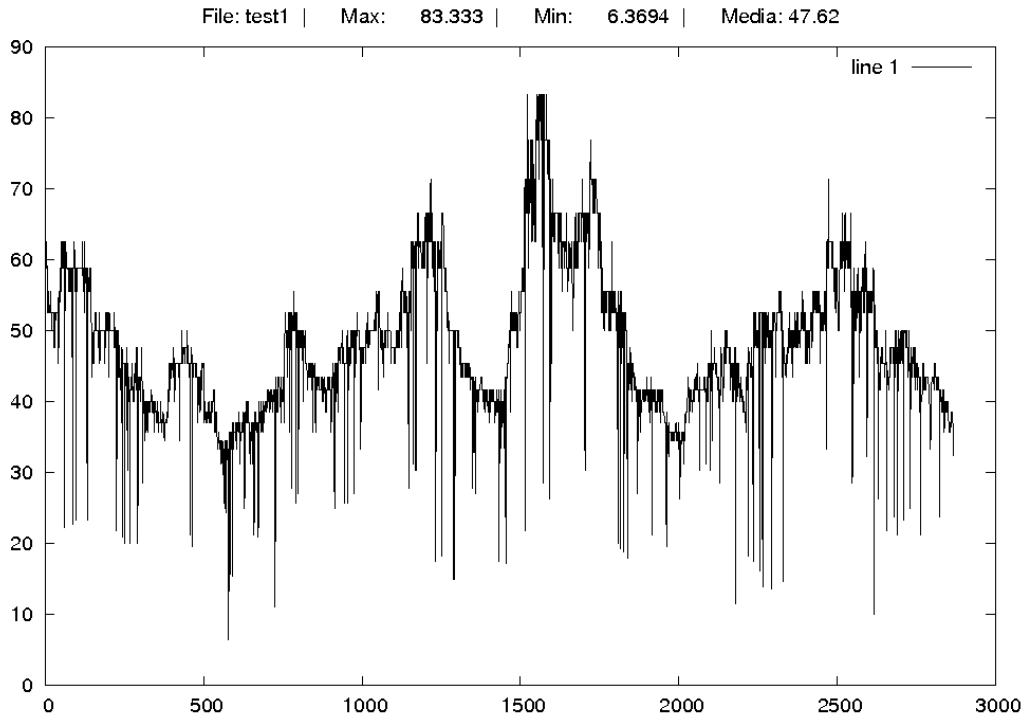


Figure C.1: Normal configuration of the KONVOI scenario

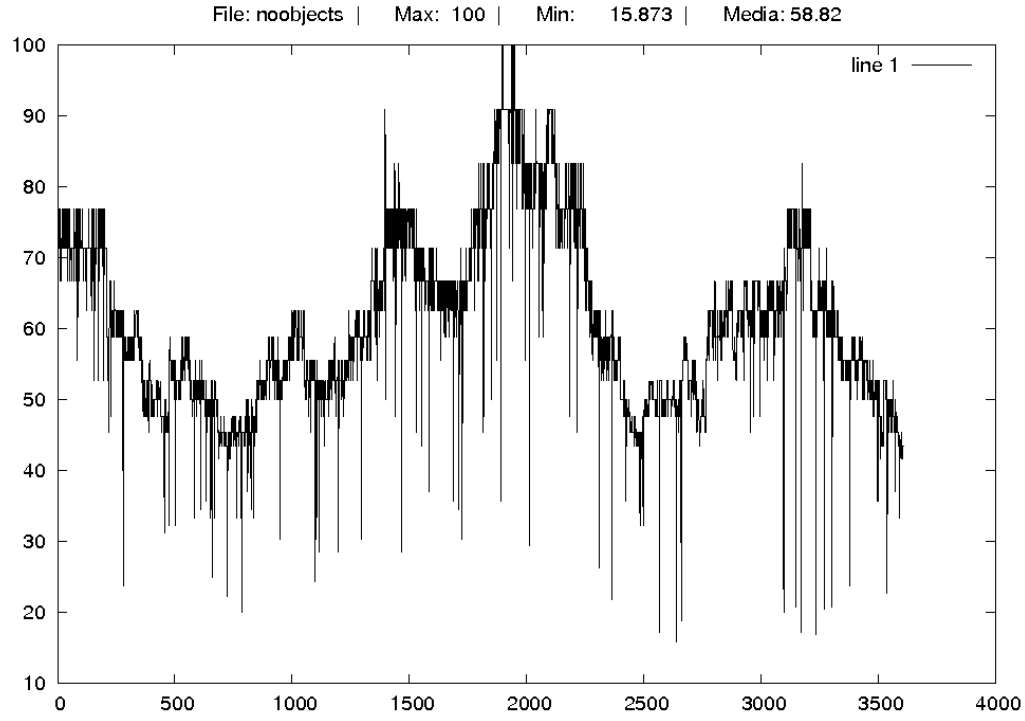


Figure C.2: Configuration without objects in the world

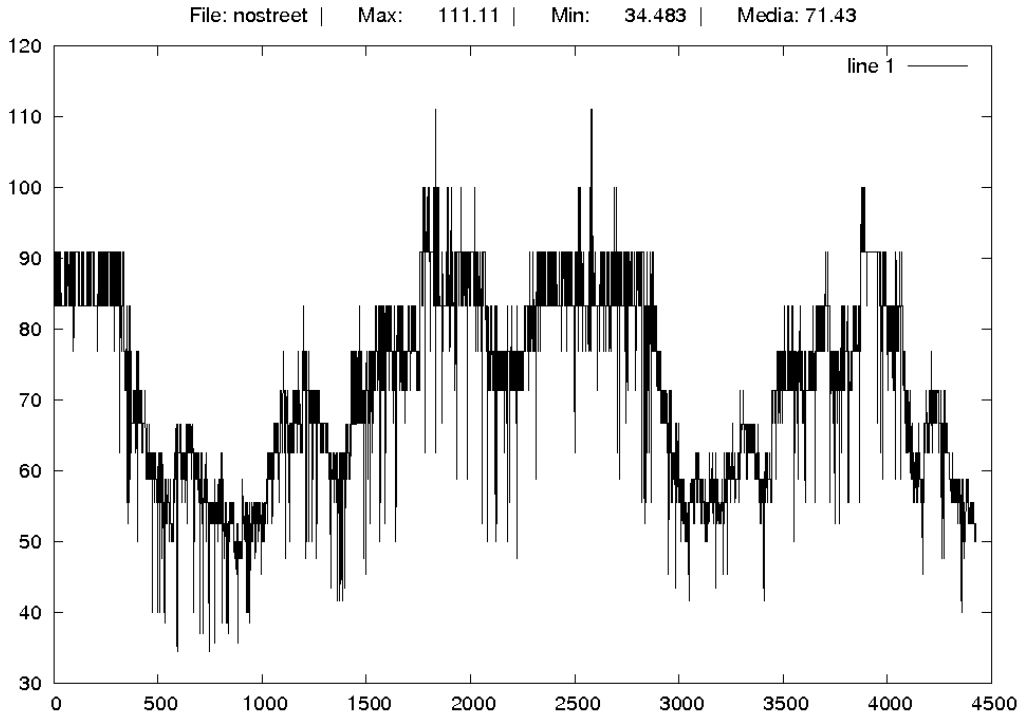


Figure C.3: Configuration without street

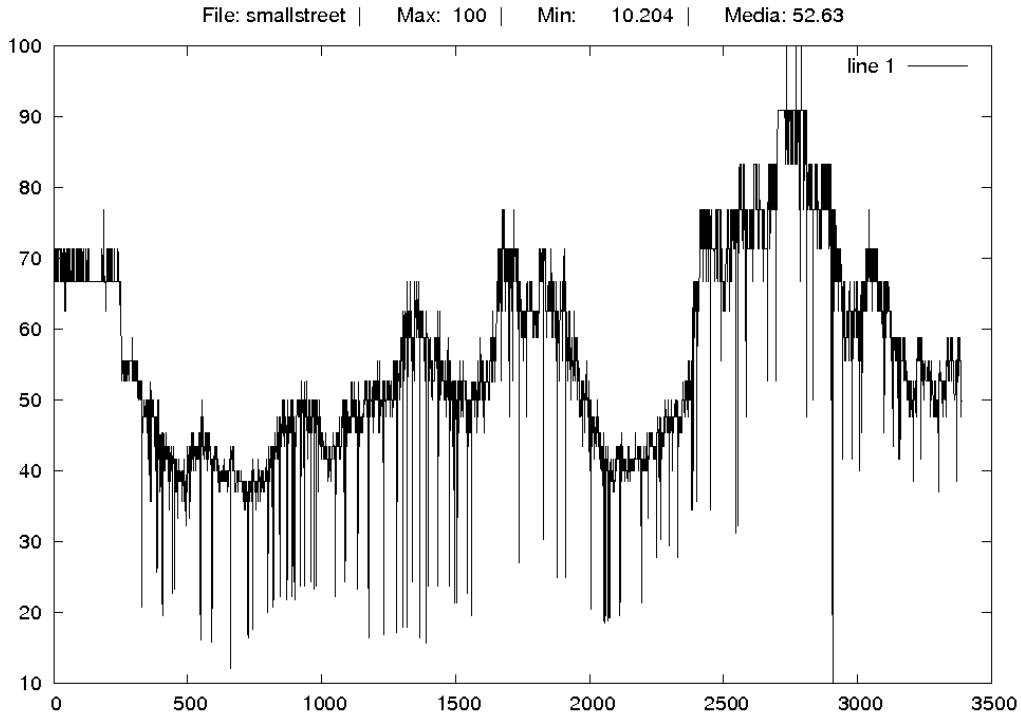


Figure C.4: Configuration using a small part of the street

Annex D : Results of the Section Management for the Street

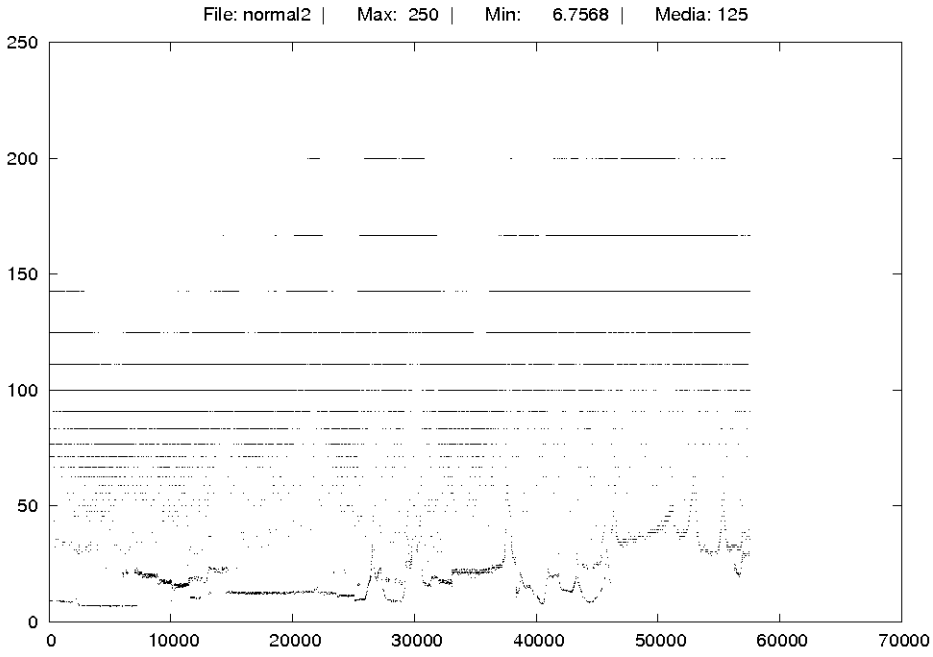


Figure D.1: Configuration without street on the section management

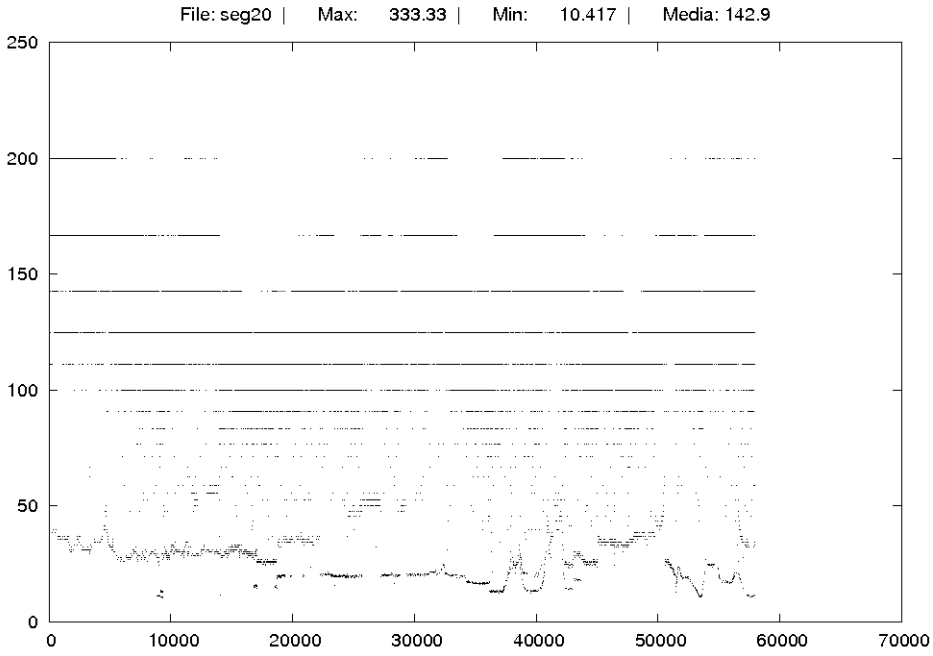


Figure D.2: Configuration with street on the section management

Annex E: Comparison between the new street and the current street

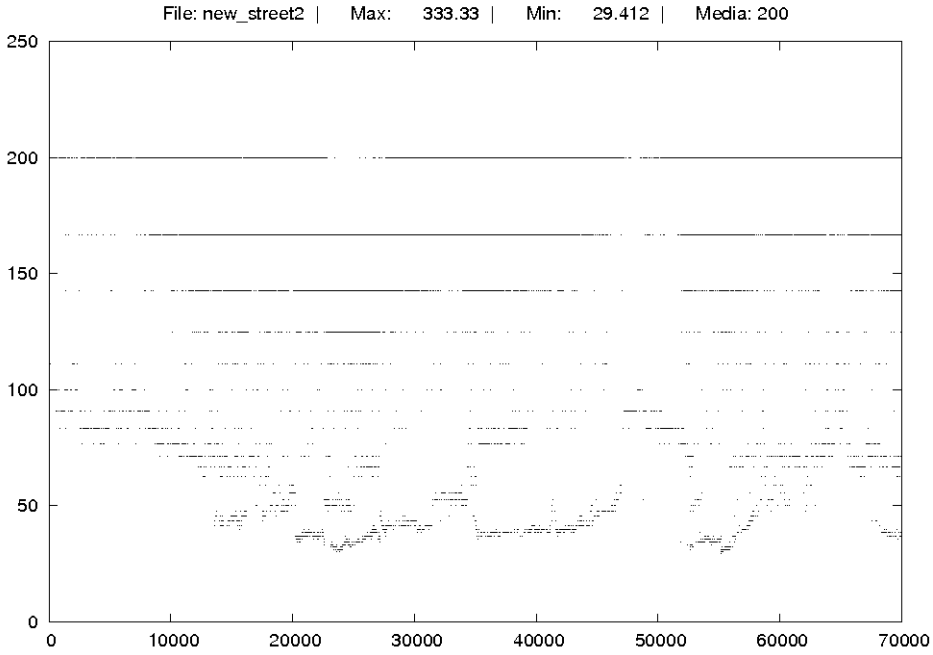


Figure E.1: Scenario with street using one texture without safety fences

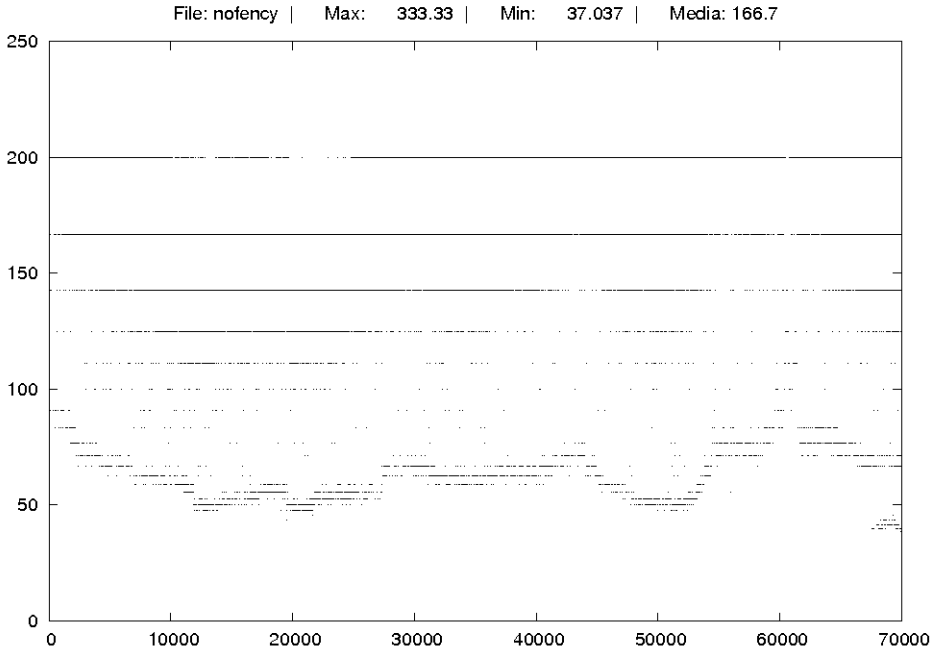


Figure E.2: Scenario with normal street without safety fences