Simulator-Based Design in Practice

Mario García
Alejandro López

Master Thesis
Department of Management and Engineering, division of Industrial Ergonomics at Linköping Institute of Technology.
LIU-IEI-TEK-A-08/00435--SE
Linköping University, June 2008.
Preface

This master thesis has been conducted under the supervision of Torbjörn Alm, associate professor at the department of Management and Engineering, division of Industrial Ergonomics at Linköping Institute of Technology, Linköping University, Sweden.

We have worked in parallel with the project “Driver Support Optimization” carried out by Oskar Pettersson and Erik Svensson. This project has also been supervised by Torbjörn Alm and it has been done in collaboration with Saab. Both projects have been performed in the Virtual Reality Laboratory at Linköping University.

Linköping, June 2008
Mario García & Alejandro López
Acknowledgements

First of all we want to express our gratitude to Torbjörn Alm and Kjell Ohlsson for giving us the opportunity of working in this project. A special mention for Torbjörn Alm, the person who made possible this thesis, thanks for guiding us through the whole process and for giving us his support and his advice with his expertise and long experience in the simulator-based design area.

During the project we have received support from many people who had made possible to realize this work. We want to thank Vitalij Savin for his support in the Virtual Reality Laboratory. Calle Issakson from ACE Simulation for his help in everything related to the ASim software. Magnus Grundberg for his introduction course to ActionScript 3.0 and all the doubts he solved for us in the mails we exchanged, thanks for sharing his knowledge within the programming field. Thanks to Anne Johansson and Henrik Bergström for the guided tour at Trollhättan at the Saab headquarters and the discussion and ideas about the rear view mirror solution. A special mention goes for Oskar Pettersson and Erik Svensson who have been working in parallel with us, thank you for all the help and for being such great partners.
Abstract

The automotive field is becoming more and more complex and cars are no longer just pure mechanical artifacts. Today much more than 50% of the functionality of a car is computerized, so, a modern car system is obviously based on mixed technologies which emphasize the need for new approaches to the design process compared to the processes of yesterday. A corresponding technology shift has been experienced in the aerospace industry starting in the late sixties and today aircraft could not fly without its computers and the pilots’ environment has turned to a so called glass cockpit with no iron-made instrumentation left. A very similar change is still going on in the automotive area.

Simulator-Based Design (SBD) refers to design, development and testing new products, systems and applications which include an operator in their operation. Simulator-Based Design has been used for decades in the aviation industry. It has been a common process in this field. SBD may be considered as a more specific application of simulation-based design, where the specific feature is a platform, the simulator itself. The simulator could consist of a generic computer environment in combination with dedicated hardware components, for instance a cockpit. This solution gives us the possibility of including the human operator in the simulation.

The name of the project is Simulator-Based Design in Practice. The purpose of this master thesis is to get a complete practice in how to use a human-in-the-loop simulator as a tool in design activities focusing on the automotive area. This application area may be seen as an example of systems where an operator is included in the operation and thus experience from the car application could be transferred to other areas like aviation or control rooms in the process industry.

During the performance of the project we have gone through the main parts of the SBD process. There are many steps to complete the whole cycle and many of them have iterative loops that connect these steps with the previous one. This process starts with a concept (product/system) and continues with a virtual prototyping stage followed by implementation, test design, human-in-the-loop simulation, data analysis, design synthesis and in the end a product/system decision. An iterative process approach makes the cycle flexible and goal oriented.

We have learnt how to use the simulator and how to perform the whole cycle of SBD. We first started getting familiar with the simulator and the ASim software and then we were trying to reduce the number of computers in the simulator and changing the network in order to find good optimization of the computer power. The second step has been to implement a new application to the simulator. This new application is the rear mirror view and consists of a new LCD monitor and the rear view vision that must be seen in the new monitor. Finally we updated the cockpit to the new language program Action Script 3.0.

The information gathering consisted of the course Human-System interaction in the University, the introduction course to ASim software and the course of Action Script 3.0.
# Table of content

1. Introduction .................................................................................................................. 2
   1.1. Background ........................................................................................................... 2
   1.2. Simulators ............................................................................................................. 3
   1.3. In-Vehicle Systems ............................................................................................... 6
   1.4. Conclusion ............................................................................................................ 7
2. Purpose and Project Agenda ......................................................................................... 10
3. Theoretical frame of reference ....................................................................................... 12
   3.1 Human-machine interaction (HMI) ....................................................................... 12
   3.2 Human Machine Design ....................................................................................... 15
   3.4 SBD ....................................................................................................................... 24
   3.5 Virtual prototyping and iterative design ............................................................... 25
   3.6 System Theory ....................................................................................................... 27
4. The simulator facility at LIU ......................................................................................... 30
   4.1 Simulator equipment and hardware ....................................................................... 30
      4.1.1 The simulator cockpit ................................................................................... 31
      4.1.2 Simulator platform and control room ......................................................... 33
   4.2 Simulator Software ............................................................................................... 34
      4.2.1 ASim .............................................................................................................. 34
5. Methodology ................................................................................................................. 38
   5.1 Human-system interaction course ......................................................................... 38
   5.2 ASim course .......................................................................................................... 38
   5.3 Flash Introduction course ..................................................................................... 38
   5.4 Visit to the Saab simulator in Trollhättan ............................................................. 39
6. Implementation ............................................................................................................. 42
   6.1 Computer optimization in the Simulator ............................................................... 42
      6.1.1 The computer network in the simulator ....................................................... 42
      6.1.2 The new configuration ............................................................................... 42
      6.1.3 Sharing the computers .............................................................................. 44
      6.1.4 The graphic card ......................................................................................... 45
      6.1.5 Conclusion ................................................................................................... 46
   6.2 Rear view augmentation ......................................................................................... 47
      6.2.1 Mirrors ......................................................................................................... 47
      6.2.2 Supplementing alternatives ......................................................................... 47
      6.2.3 Introduction and LCD monitor ................................................................. 48
      6.2.4 The new configuration of the simulator software ...................................... 48
      6.2.5 Position of the rear camera ....................................................................... 48
      6.2.6 The solution at Saab .................................................................................. 49
   6.3 Cockpit ................................................................................................................... 50
      6.3.1 Cockpit file .................................................................................................. 51
      6.3.2 Cockpit and events ..................................................................................... 52
      6.3.4 Cockpit final design ................................................................................... 55
      6.3.4 The four modes ........................................................................................ 57
      6.3.5 The seven categories ................................................................................. 64
7. Results ......................................................................................................................... 72
8. Conclusions .................................................................................................................. 74
   8.1 Conclusions .......................................................................................................... 74
   8.2 Recommendations and future possible developments ......................................... 75
9. References .................................................................................................................... 78
<table>
<thead>
<tr>
<th>Appendix</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix</td>
<td>84</td>
</tr>
<tr>
<td>Appendix A</td>
<td>84</td>
</tr>
<tr>
<td>Appendix B</td>
<td>109</td>
</tr>
<tr>
<td>Appendix C</td>
<td>113</td>
</tr>
<tr>
<td>Appendix D</td>
<td>116</td>
</tr>
</tbody>
</table>
## Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flight Simulator [1]</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Evolution of flight simulator usage and technology [2]</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Driving Simulator [3]</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Conceptual model sketch</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>Scheme of Seven Stages of Action [4]</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>Responsible parts of security [5]</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>System classification</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>SBD design loop [6]</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>Concept of iterative design in software development [2]</td>
<td>26</td>
</tr>
<tr>
<td>10</td>
<td>Model for computer-based in-vehicle systems</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>View of the simulator cockpit at LiU[7]</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>Replacement of the ordinary steering system [7]</td>
<td>31</td>
</tr>
<tr>
<td>13</td>
<td>PC cluster arrangement [7]</td>
<td>33</td>
</tr>
<tr>
<td>14</td>
<td>Control room [7]</td>
<td>33</td>
</tr>
<tr>
<td>15</td>
<td>Schematic view of ASim [8]</td>
<td>34</td>
</tr>
<tr>
<td>16</td>
<td>ASim interface user</td>
<td>36</td>
</tr>
<tr>
<td>17</td>
<td>Simulator facility at Saab [9]</td>
<td>39</td>
</tr>
<tr>
<td>18</td>
<td>Projector panel distribution</td>
<td>42</td>
</tr>
<tr>
<td>19</td>
<td>Sketch of the first configuration</td>
<td>43</td>
</tr>
<tr>
<td>20</td>
<td>Sketch of the final configuration</td>
<td>45</td>
</tr>
<tr>
<td>21</td>
<td>Camera configuration</td>
<td>48</td>
</tr>
<tr>
<td>22</td>
<td>PDC file</td>
<td>53</td>
</tr>
<tr>
<td>23</td>
<td>Wheel and buttons</td>
<td>54</td>
</tr>
<tr>
<td>24</td>
<td>Buttons names</td>
<td>55</td>
</tr>
<tr>
<td>25</td>
<td>Program flow chart</td>
<td>56</td>
</tr>
<tr>
<td>26</td>
<td>Default mode</td>
<td>57</td>
</tr>
<tr>
<td>27</td>
<td>Night mode</td>
<td>60</td>
</tr>
<tr>
<td>28</td>
<td>Sport mode</td>
<td>63</td>
</tr>
<tr>
<td>29</td>
<td>Economy mode</td>
<td>63</td>
</tr>
<tr>
<td>30</td>
<td>CCR category</td>
<td>64</td>
</tr>
<tr>
<td>31</td>
<td>Source category</td>
<td>65</td>
</tr>
<tr>
<td>32</td>
<td>Media category</td>
<td>66</td>
</tr>
<tr>
<td>33</td>
<td>Climate category</td>
<td>66</td>
</tr>
<tr>
<td>34</td>
<td>Navigation category</td>
<td>67</td>
</tr>
<tr>
<td>35</td>
<td>Phone category</td>
<td>68</td>
</tr>
<tr>
<td>36</td>
<td>Mode category</td>
<td>69</td>
</tr>
<tr>
<td>37</td>
<td>VP-222 datasheet [10]</td>
<td>113</td>
</tr>
<tr>
<td>39</td>
<td>Aspect DVI-VGA [12]</td>
<td>114</td>
</tr>
<tr>
<td>40</td>
<td>LCD monitor datasheet [13]</td>
<td>115</td>
</tr>
</tbody>
</table>
Chapter One

This first chapter explains to the reader the context in which this thesis has been carried out. The Simulator-Based Design concept is introduced and explained. A brief description of simulator functionality is also given. We hope that reader enjoys reading this thesis, and find it useful to get basic knowledge about what a simulator means and how it could be used in the product design process.
1. Introduction

1.1. Background

The main focus of this master thesis project was to practice Simulator-Based Design (SBD) in the automotive area and to give further insight in this field of application.

SBD may be seen as a more specific application of simulation-based design, where a specific platform, the simulator itself is used. It could also run on a generic computer environment but it would include more dedicated hardware components, for instance, an aircraft or ground vehicle cockpit and a visualization system for environment presentation. This solution gives us the possibility of including the human operator in the simulation.

We might say that research in the area of Simulator-Based Design focuses on integrating advanced information technologies and techniques for enhancing design and engineering activities in areas where the human operator is an essential part of the complete system (Alm, 2007). In the following we will present a short review of human-in-the-loop simulators. This means, the ones that involve human-system interaction. The best way to obtain valuable insight into the impact of new automation and control tools (input/output functions) is to use real time human-in-the-loop simulations. People involved in the experiments will interact with realistic models of the tools and will perform as they would be handling corresponding activities in real life. In our case we wanted to explore how the entire human-machine system would perform in different environments when facing different traffic situations. To introduce a short review of human-in-the-loop simulators we will have to mention flight simulators, the pioneers in this area. Ground vehicles simulators are relatively new compared to flight simulators and from some perspectives not so developed yet. Simulators are also used in many other fields such as aerospace programs, medicine, process control, etc. Without simulators we could not have reached many of the technological advances we find today.

One main feature of simulators is that it is possible to do things that would be extremely dangerous in a real world situation, or maybe unexpected situations that occur very seldom in real life. In a simulator making a mistake will not mean a risk of lives or a huge loss of money, it will not have any other consequences than that the person who made the mistake will learn from the complicated situation. Of course it is also important to have the possibility to repeat such situations as many times as we wish for training or other purposes in a completely controlled way. (www.vtt.fi)

In concluding this background section, it is necessary to present some global technology trends from the automotive area. There is a fast ongoing transition from the pure mechanical car to a mixed technology product, where today more than 50 % of all functions are computer-based. In the modern cockpit we could see a trend towards screen-based interfaces. In aviation they coined the expression glass cockpit in the eighties as a notion for this interface approach. This development was also a basic prerequisite for the application of SBD, a programmable end product with a glass
cockpit and a corresponding simulator resource which opened for an iterative design process with no physical prototypes.

1.2. Simulators

Simulators could be found in many areas today, as mentioned above, but the pioneer in using these tools was the aviation community. The first known flight simulator was constructed in 1929 by Edwin Link. It was a completely mechanical flight simulator, where people could experience and train the fundamentals of flying. In the sixties the simulators became more advanced due to introduction of computers which made the simulators much more real and it was now possible to create much more elaborated scenarios and more advanced aircraft instrumentation. This opened for possibilities to train tactical missions and emergency situations.

![Figure 1. Flight Simulator](image)

In the seventies design activities appeared as a new issue in the flight simulation area. During the eighties and nineties, simulation took further steps and also the use of simulators for marketing purposes became important. Now it was often more useful to letting customers meet a new aircraft by flying the simulator than having real flights. Since then another important issue was included, verification of new functions and systems in simulators and thus reducing the number of test flights. This simulator development is further summarized by the graph in Figure 2, *(Alm, 2007)*
Moving the focus from flight to car simulators we found that the main interest so far has been in training and behavioral research. This finding was established by scanning the agendas of simulator sites presented as links at the home page of the French Research Institute INRETS (Institut National de Recherche sur les Transports et leur Sécurité), (http://www.inrets.fr/ur/sara/Pg_simus_e.html). Our finding indicates that the overall usage profile could be compared with the usage profile of flight simulators in the sixties (Figure 2). Obviously, there are steps to climb for the automotive area. We can also see that most of the simulators belong to research organizations and universities including Linköping University while only a few simulators in the industry are presented. However, there is a growing industrial interest to use simulators as tools for design. Through our coupling in this project to Saab Automobile and GM, we know that such efforts are ongoing both in Trollhättan, Sweden and in Detroit, USA.

There are two basic prerequisites for using driving simulators as tools in the car design process. The first is to have a software system that allows for implementing user-made applications which could work seamlessly together with the complete car system. On the software side this is of crucial importance for an effective design process. If you need to go back to the simulator software supplier for modifying the software every time you want to add new car functions, this will be a too high threshold.

The second prerequisite is to have a glass cockpit. This means that the driver interface should be software-based and the hardware for interaction should be based on screen solutions and other facilities necessary to offer a multi-modal interface.
This means that recourses should be available not only for visual interaction but also tactile and auditory interaction. If this goal is achieved it is possible to implement driver interface solutions very freely with no physical changes. (Alm, Alfredson, and Ohlsson, 2007)

Figure 3. Driving Simulator [3].

The above mentioned prerequisites do no cover every aspect on the simulator resource for an effective application of SBD. Some further insight could be found in chapters 3.4 and 3.5 where the SBD process and the important virtual prototyping procedure is described.

A final issue in this section will be to briefly comment fidelity and validity, which are often brought to discussion, not the least in comparing test car driving and simulator studies. Fidelity in this area is referred to how well a simulator represents reality. Thus, high fidelity simulators have sophisticated movement platforms, good environment visualization systems, and realistic system performance. An example in this category is the National Advanced Driving Simulator (NADS) in Iowa, USA. However, this simulator does not fulfill the above mentioned basic demands of SBD. This means that there could be validity problems in a SBD evaluation in NADS. On the other hand and interpreted from outside, NADS was not designed to be a tool for such studies.

Referring to this discussion, validity is nothing which comes automatically in a high fidelity simulator even if driving in such a resource feels very close to real driving. Validity, which means measuring what was intended to measure and the possibility to make conclusions possible to generalize to real world performance, must be considered for every study separately. The specifics of each study will determine the resource level needed to achieve the goals. In SBD studies the notion of relative validity is in focus. In other terms, the system or function in question will always be compared with a baseline alternative, often together with another product approach. All alternatives will be investigated during exactly the same conditions, traffic events, obstacle appearances, etcetera. This is something that is almost impossible to achieve in real driving as well as exposing the driver to dangerous situations.
1.3. In-Vehicle Systems

Since more than a decade the automotive industry have had a system approach to its products. Today this view could be taken further step where the vehicle could be regarded as a system of systems and where this second level could be described as In-Vehicle Systems (IVS). Here we could find an increasing number of systems which involve the driver such as different warning systems, satellite navigation systems, media applications, mobile phones, DVD players, and e-mail systems.

Thus, we have experienced a big explosion of new computer-based functionality. In many ways this has been a copy of what already happened before in the aircraft area. As in aviation, many of these new systems are related to safety issues and to enhancing situation awareness. However, we have to consider that every time a new system is added to the vehicle we increase the information content and too often also the driver workload which should be kept within the driver’s capability. The way to achieve this goal is by system integration.

Thus, one of the main issues when referring to In-Vehicles Systems is system integration. This is especially important since many of these systems are developed and delivered by suppliers. The integration could be purely technical but also covers the Human-Machine Interactions (HMI) perspective which is the focus of SBD. If we look around on what is offered on the market today we could find numerous examples of badly or not at all integrated solutions. For example, very often safety related systems have its own sensors, computer-based functionality, and its own presentation resource in the cockpit. Such approaches are bad from all perspectives not the least the economic and ergonomic perspectives. Sometimes the complete solution is even counterproductive. The purpose of the system was meant to improve safety but the resulting situation with no integration could mean decreased safety. The simulator is a great tool for integration work and in the SBD approach good integration is possible to achieve long before introduction to real driving. (Alm et al., 2007)

In the simulator environment it is necessary to treat different parts of the in-vehicle system differently. We will come back to this in section 3.5, Virtual Prototyping.
1.4. Conclusion

The car technology is changing very fast and the cars are no longer just pure mechanical artifacts. They have more and more electronic components including new technologies and a raising number of software-based systems. Thus, cars are becoming more complex which put higher demands on the driver. As we have seen along the chapter there are a great number of in-vehicle systems which are included in the cars nowadays. SBD is a great tool for the integration of all these new systems. In the past physical ergonomics was good enough in the automotive industry. Nowadays cognitive aspects must be included in order to satisfy the demands on modern approaches to Human-Machine Interaction.
Chapter Two

This chapter summarizes the fundamental ideas which have been focused on this thesis. The followed agenda is also mentioned as well as different courses and literature reviews which have been gone through in order to understand the background to SBD and the software and hardware environment in the driving simulator.
2. Purpose and Project Agenda

The first project agenda was set at the beginning of the project in January of 2008. During the project some aspects have been modified since this was a preliminary project agenda. The main purpose and all the results we have achieved are explained during this chapter.

Project agenda

At this initial stage the agenda was not yet completely decided but this was a preliminary list of activities:

- **Education**
  - Ongoing course in Human-Systems Interaction
  - Introduction course on the simulator hardware and software
  - ActionScript 3.0 (the tool used for application design in the simulator)

- **Design and implementation**
  - Implementation of a LCD rear view mirror for the simulator cockpit
  - Porting existing Flash ActionScript 2 applications to ActionScript 3
  - Design and implementation of a navigation module
  - Computer optimization in the simulator

- **Simulator testing**
  - Analysis of test data.

- **Report production**
  The master theses report will be produced on a more or less continuous basis during the whole project.

Purpose

The name of the project is Simulator-Based Design in Practice. The purpose of the project is to get insight and hands-on practice in how to use a human-in-the-loop simulator as a tool in design activities within the automotive area. This application area may be seen as an example of systems where an operator is included in the operation and thus experience from the car application could be transferred to other areas like aviation or control rooms in the process industry.

The main activities that we have performed have been the cockpit application in the new language ActionScript 3.0, the investigation about computer optimization by reducing the number of computers in the simulator and finally implementing the rear mirror view application.

We have to mention that the only part that has been excluded in the complete process of SBD has been the part related to the evaluation of data, what it is called data analysis in the SBD loop. Thus, we have exercised the complete process of SBD but with this final point excluded and gone through the following steps: virtual prototyping, implementation, both hardware and software, test design and finally human-in-the-loop simulation. In this the collaboration with a parallel project on driver support optimization was an important task.
Chapter Three

This chapter contains the theoretical frame of reference. Several human factors are analysed. We emphasize how important the designer task is. Some system theories are also mentioned. Important concepts as interface and interaction are introduced. We will go deeper in the importance of computer in the industry nowadays.
3. Theoretical frame of reference.

In this chapter different terms and concepts are analysed in order to understand in which context this thesis has been developed. A brief definition of main ideas and concepts will be given. Furthermore, general knowledge about Human Factors is described, because it is easier to use concepts derived from the Human Factors area in order to know about any system functionality from a human point of view. Moreover, several concepts related to systems such as machine, computer or vehicle will be described. This chapter will focus on human vehicle interaction. For this purpose, first of all, a top level description is gone through and, subsequently, it will focus on vehicle issues.

3.1 Human-machine interaction (HMI)

Human-machine interaction is the study of interaction between humans and machines. It is normally referred to as the intersection of computer science, cognitive science, behavioural sciences, design and several other fields of study. Interaction between humans and machines occurs at the user interface, which includes both software and hardware, for instance, general computer peripherals and large mechanical systems, such as aircraft and automotive industry.

System: it is a set of components, which are related to each other in order to produce a certain operation. This is a very general and wide definition in the sense that lot of items can be considered as elements. There is an infinite amount of systems. They are commonly used to operate with objects and situations every second in the daily life. For this reason it is a very important issue to manage how to handle them. System handling has to be a concern of system designers, who accomplishes the interface design. A large part of the researchers within the HMI field are psychologist, that is, they focus on the human component and they do not usually address the machine and interaction issues in a profound way. However, within the branch of Engineering Psychology, the main emphasis is usually on the interaction parts in the HMI. We would like to stress the pertinence of the machine aspects in the human-vehicle interaction. Designer labour is also very important in order to obtain an efficient and effective system in a holistic meaning.

Let us define a couple of new concepts introduced: interface and interaction. Technically, interface is the hardware and software that allows the user to control and communicate with a system. Widely, interface is all type of gadgets, which allows the communication between operator and system. Somehow, it can be viewed as the main communicator between the user and the system. For the sake of obtaining easy interfaces between human and system several Human Factors issues must be analysed. (Norman, 2002)
Interfaces are designed to produce interaction between two or more objects, in our case, between the operator (driver) and the different systems, which constitute a car. Interaction in this case is the interplay (including communication and feedback) between the driver and the machine with its systems in order to reach a common goal, a safety journey.

System will be considered as a machine, computer or vehicle, all kind of objects, which needs the human brain to guarantee a correct operation. From now on, we will focus on the car as a system. The man is considered as the operator. Car and driver must work together. At the beginning, car industry was a concern of mechanical engineers. In those days, there were no computers used as an integrated system in a car. The car in its most pure concept was a totally mechanical vehicle. From this idea the concept of HMI comes up. HMI people work, to a large extent, with real-time applications in the automotive industry.

The breakthrough and the posterior developing of computers with respect to both software and hardware produced an important advancement in car industry and many other industries. Computer science and electronics are closely related to each other. They contributed together to perform new systems, which were installed in cars. Nowadays, new electronic appliances and computer systems keep coming up. Let us begin with what a computer means. (Happian-Smith, 2001)

Computer: an electronic machine, which is used for multiple tasks such as organizing, storing and finding words, numbers and pictures. Other possible uses are for doing calculations, but the most important task from our point of view, concerning this project, is for controlling other machines. In this last case, computers are known as embedded computers and this is the most widespread use nowadays. (www.techterms.com)

The development of different industrial applications based on a widespread usage of computers led to a new concept: HCI, Human Computer Interaction. Computers are based on integrated circuits and processors. Embedded systems are placed in all kind of machines from vehicles to industrial robots, mobile phones, toys, electrical appliances, from nuclear reactors to animation software and so on. Some of them are real-time applications whereas most of them are more focused on non real-time applications like office systems, gaming etcetera.

The matter is that a poor design of both computer and interface software can cause a wide variety of unwanted events and negative consequences. The Human-Computer Interaction must be fluent and must accomplish in a deadline time in critical situations. This condition is very important in real time systems. The time is a main factor in order to not produce a catastrophic situation. A fault may be life-threatening. For this reason, all the aspects related with the system must be studied and designed properly. According to how a conventional computer is made, the interface to interact with this is, in most cases, a software interface. Actually, technology progress is growing in a very high speed. Every day new technology appears on the market. Somehow, the pace of events is being dramatic and fast. There are different investigation lines opened in which the time response is vital. The reader should notice that we are viewing computers as a part of real time system installed on cars. (Cooling, 2003)
One important point is that real time system must be capable of recovering from a mistake and keep going on with its task, otherwise, a disaster could occur. Anyway, a real time system should never fail ideally. To explain how important this idea is, a real time system is explained below.

Computer systems dealing with night vision in vehicles may be a good example of a time critical driver support system. The aim is to design smart cars to make the driving easier and more safe. The traffic volume decreases more than 60% during the night hours. Even though this reduction, almost 42% of mortal accidents occur during the night according to CEA (Comisionado Europeo del Automovil, www.cea.es). This fact is due to the loss of vision sharpness and the reduction of the visual field (e.g. gaze field), this reduction is because the car light cannot illuminate everywhere. The project DRIVSCO is being developed by the University of Granada (Spain) cooperating with other European researchers. (www.pspc.dibe/)

This project consists of installing a chip in the car, which allows the detection of external events during the night, that is, the system warns the driver of possible obstacles on the road or around it. This process occurs before the obstacles achieve the visual field of the driver. This is really important because the cross light reaches up to 56 meters and the break distance is approximately of 88 meter when the speed is 100 km/h. Autoliv AB has tested their night vision system in the car simulator used in the current study. In addition to this, it should be mentioned that Linköping University on behalf of Saab Automobile AB has developed a prototype of a head-up display that has been installed in the car simulator. Linköping University has also developed a prototype of a head-up-display of higher fidelity for installation in forensic machines in cooperation with Skogforsk in Uppsala and Komatsu AB in Umeå. They former is only designed and prototyped to be able to simulate this security system in the simulator, whereas the latter is designed with an explicit purpose to be installed in future forestry machines. Let us go back at the DRIVSCO project.

The system is based on two infrared cameras placed in the car. The chip takes information of these cameras using data as depth, movement and so on. The chip is capable of processing where objects are moving in real time.

Inside this project there are also other universities such as the University of Münster (Germany). They are working in the study of the driver eyes. They try to figure out a model to find out where the driver is going to look at. The system is known as eye-tracker. This project is the continuation of the project ECOVISION, which also aimed at ADAS (Advanced Driving Assistance Systems). (www.pspc.dibe/)
3.2 Human Machine Design

Human Factors

One important task of a designer consists of knowing perfectly in which context his/her system design is going to be used, what the purpose is and who will use it. The designer must learn and analyse the psychology of possible customers or users. The designer must carry out a study of user psychology in order to create an easy interface-system.

Even though a commercial purpose, a system must satisfy user needs: easy to understand, easy to handle, good operation, low complexity etcetera (Nielsen and Holland, 1996). Designers have to provide the way to obtain a fluent communication between customer and systems. For the sake of this fluent communication, it is important to point out some Human Factors that should be taken into consideration by designers: education, work experience, work physiology (moreover, human physiology). Each factor is explained in more detail below.

Education. Social environment in which people are moving in, determines very often the education of them. The society is split into social-economic layers. Each layer has its own characteristic. It makes, currently, that people receive different sorts of education at almost each social level. In one hand, there are systems, which can be considered very easy for certain set of people. In the other hand, these systems are able to produce troubles in others groups. Designers should analyse these education level and figure out how to focus their system on the user. Another concern to take into consideration is that not all people have the same level of abstracting ideas and concepts.

Experience. Work experience is another factor to take into consideration. It is not the same to develop an advanced system for users who are familiarized with similar environment than users who have not seen something similar in their life. For this reason, the designers should develop interfaces adjusted to user experience: beginners, intermediate or advanced drivers.

Human physiology. As it was mentioned before, the nature of the purpose is also really important. Another concept related to this one is human physiology. Depending on human physical limitations and the nature of the purpose that user want to manage, systems can be developed in distinct ways. For example, a man trying to cross the street walking over a pedestrian crossing does not apparently need sounds signs, which are very important to blind pedestrians. (de Waard, Brookhuis, van Egmond & Boersema, 2005)

Moreover, a deeper classification can be done in function of the nature of human components. These human components are visibility, sensation, perception, communication cognition and decision, motor control, muscular strength, other biological factors and so on.

Perception marks individual differences. Designers should try to find out how users are able to perceive his/her design and the corresponding interface. Communication, cognition and decision are areas and skills, which can be improved
following a training period. Actually, this training is a continuous process along the entire life. The most important human component which allows people to handle situations is the motor control. Stress influences the motor control. This is not a task of a designer to help user to control himself, but designer can help user trying to create an interface, which does not produce unnecessary stress in order to provide a nice and quite environment. (Wickens, Lee, Liu, & Gordon Becker, 2004)

Model of human behaviour

The control system needs manipulation of a large amount of data at very high rate efficiently from the human operator. It is also very important that cognition processes of a human are taken into consideration. Rasmussen tries to explain a way to understand the human cognitive behaviour. “The main purpose is, on one hand, to study the cognitive aspect of human cognitive behaviour and, on the other hand, to investigate artificial intelligent technologies for modelling the human decision making.” (Qureshi, 2002)

Rasmussen split the human behaviour into three levels: skill-based behaviour, rule-based behaviour and knowledge-based behaviour. The skill-based behaviour is acquired by a training process. It allows people to respond briefly at external stimuli. Rules-based behaviour means to manipulate a specific plan. Knowledge-based behaviour treats to handle an unusual or unexpected situation, for which there are no plans to solve the situation. (Rasmussen, 1983)

Conceptual model

The meaning of a conceptual model is different depending on the kind of topic we are dealing with. It has different meanings. The common basic idea is that the conceptual model is a representation of something. It is not the same as the conceptual model that a biologist has of a natural system (for example, how it is a protein formed.) than that one which a doctor has (for example, an organ) or an architect has about how a building is built. Evidently, all of them have a different vision of how is a protein, an organ and the building like. A normal cognitive process follows the next steps: “people receive information, process this information and respond accordingly many times each day.” (www.serc.carleton.edu).

This sort of processing of information is essentially a conceptual model (mental model) of how things in our surrounding environment work. People create conceptual models of device and try to imagine mentally how a system is used for and how to manage it. The conceptual model is usually based on device properties and attributes.
Figure 4 shows a theoretical scheme of human system interaction. It is possible to distinguish three different types of conceptual model: design model, user’s model and system image.

Design model: This is the designer’s conceptual model. In the same way, user’s model means the user’s conceptual model. It has been produced due to interaction between user and system. Finally, system image results as a consequence of building the physical structure for the system. The designer’s goal is to get the user’s model identical with the design model. Ideally, the system image approach perfection when user’s model and the design model are quite similar. This goal is really difficult to reach, since the quality of a system is given by the grade of similarity between the mental concepts that user and designer have for the same system.

Important features of the system image which are able to help the user to manage the system are: Its affordance, mapping, visibility and feedback. The designer must keep these features in mind in order to offer a good product.

Affordance: In a psychological context this term refers to the inherently perceived and actual properties of the system. Fundamental features which determine the device appearance and how it could be used. If extra pictures or explanations are needed, then, the design has failed from and ideal point of view. Clearly, the perfection is not reachable and, furthermore, there are many different users. It is utopia to find a universal system image, which all users are able to manage.
Visibility: This is one of the most important principles of design. The correct parts must be visible and they must convey the correct message.

Feedback: It is a term referred to as the information that the system gives the user back corresponding to an action recently accomplished by the user, even the status of the system or whichever data that can be useful for the user. Capabilities of humans are very appreciated talking about feedback. These capabilities are sight, hearing, touch, smell or taste, the basic senses. Anyway, in vehicle systems, touch, sight and hearing are the main senses used to give feedback information to the driver.

Mapping: It means what user expects to occur when he goes through with a certain action. Mapping takes place in accordance with a mental model of the situation.

As a general rule, all system in need of additional information to be used, means that it has not been very well designed. (Norman, 2002)

Design theories.

Upon reaching this point, theories and models for software and interface design should be explained. We will focus on one theory called Seven Stages of Action and one model of user performance for design: GOMS. To go through with a good design, the designer should get into the user’s mind. For this reason, both theory and model are explained from both points of view: user’s point of view and the one according to the designer.

Seven Stages of Action. This theory was developed by Donald A. Norman (1986). It is formed by two “corridors” and seven steps. These two corridors are named as execution corridor and evaluation corridor. They are used to communicate goals of the user and the real word (physical system).

Now, it is time to describe this theory. First of all, the designer needs to get into user’s thought in order to understand and obtain the needs and goals of the user. It is the first step. Once it is achieved, the designer knows what he/she has to reach. As it is drawn in the diagram below, the theory is going over a single and return way (mentioned before as corridors). However, these corridors have different steps. The single way (first corridor) is named execution and the return way (second corridor) evaluation.

Execution. Intention to act, sequence of actions and execution of action sequence are the set of steps, which form this corridor. Goals must be converted into intentions (intention to act), and the designer must decide how to accomplish (sequence of actions or action specification) its intentions. The designer should define the sequence of physical actions, which are going to be used. Once the actions to be done are defined, the user should go through with them. User executes actions (execute to action or interface mechanism). Following all these steps, we have transformed a goal into a physical action (real world). Four steps have been described so far. Let us go to explain the second corridor.
Evaluation. Perceiving the state of the world (interface display), interpreting the perception and evaluation of perceptions are the last three steps of this theory. In this corridor, user observes which events emerge in the world. Later on, user has to interpret changes and events happened and, finally, an evaluation process is accomplished. This evaluation is based on comparing what the user wanted to achieve and what actually has been produced.

![Diagram of Seven Stages of Action](image)

**Figure 5. Scheme of Seven Stages of Action [4].**

**GOMS.** It stands for goals, operators, methods and selection rules. This model was developed by Card, Moran and Newell (1983) and expanded by Kieras (1988). The GOMS model contributes to a detailed description of the user, the tasks and, sometimes, tries to find out user behaviour before using a certain system. GOMS is a series of steps taken to acquire knowledge which are perceptual, cognitive or motor operators.

The fundamental use of this model is to make an interface description and comment software functionality in a straight way. Designers have to identify which goals are looked for by users; they should be capable of providing all possible alternatives to reach each goal or sub goal. Finally, the designer has to specify all the rules used to apply different prior analysing alternatives.
GOMS can be seen as a tool which defines a language used for modelling interactive software creating interfaces and permitting communication between user and system. “Seven stages of action” theory is (as whichever theory) a way to describe the communication process between user and the real world. Although GOMS and seven stages of action are very closely related to each other, the main difference is that GOMS try to apply the theory to develop an interface to manage the physical world. Another difference that is also an advantage with the GOMS model, deals with making predictions. “Seven stages of action” does not predict events. The evaluation process is accomplished when the physical world responds to actions.

To develop a good interface, the GOMS model has to take into consideration these concepts defined before: affordance, feedback, visibility and mapping. Designers ought to develop the next group of items: basic screen design, dialog styles, menus, fill-in forms, questions-answers, command languages, function keys, direct manipulation and a natural language. (Wickens, Lee, Liu, & Gordon Becker, 2004)

It is worthy to achieve the ideal goal that an interface is perfectly handled without any external and additional help. Obviously, this is almost never reached and, for this reason, extra information is enclosed the delivered product. This extra information, that supports created mechanisms, is presented in manuals and as online help.

Usability

Usability is defined as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”. A good rule in marketing consists of defining the necessity of the customer, who our customers are and what they expect of the product. On this point, we will not provide marketing knowledge, but we will define who our system is dealing with, the wanted goal of them (what this is used for) and the context of use. All in all, it provides us with facilities to explain effectiveness, efficiency and satisfaction regarding our system. By explaining these three terms, it will be possible to analyse usability. (www.modulobus.org)

Effectiveness is the accuracy and completeness with which specified users can achieve specified goals in a particular environment. (Davidson, Dove, Weltz, 1999)

Efficiency refers to the resources expended in relation to the accuracy and completeness of goal achieved. (Davidson, Dove, Weltz, 1999)

Satisfaction is the comfort and acceptability of the work system to its users and other people affected by it use. (Davidson, Dove, Weltz, 1999)
HMI car design conclusion

First of all, an automobile must be viewed as a machine, which can cause accident and hurting people even the loss of human life. Technology progress supports the designer to conduct better safety measurements in order to avoid attention loss among drivers. Do not forget that the automobile is a machine and humans are the final users of it.

Car safety is a concern both with regard to installed safety measurements by engineers and to responsibility of the driving humans. Safety used to be a concern of people and constitutes a factor that can influence drivers in almost every situation. Human traffic behaviour is subjected to societal educational efforts along the different stages of life. Nevertheless, the automobile can be much improved by human research and development. On one hand, safety measurement must be developed in order to obtain a car with better safety and, on the other hand, less complexity and efficient interfaces must be developed in order to achieve an easy handling. The latest point is very important, because it allows drivers not to lose attention.

![Diagram of responsible parts of security](image)

**Figure 6. Responsible parts of security [5].**

Other contextual factors car safety as well. Taking a look at different period of the history, it is possible to assert that weather conditions, road state, speed, safety development together with Human Factors contribute to the entire traffic safety. At the beginning of the car history, cars were not able to reach high speeds, so the risk of suffering an accident was low even though cars were not much developed or there were no roads as we have today. The big deal of designers of today is primarily to get a balance between car speed and safety. This is also a governmental concern. All factors should be improved together to achieve the highest safety ratio as possible.
The car manufacturers’ main goal was in former days to create very fast cars without taking any other factor into consideration. Nowadays, statistics show us the number of death per year produced by car accidents. It has made that the society: the government, companies and people are worried for these avoidable deaths. In this sense companies are investing more time, efforts and money to obtain the most modern safety system. Governments have to support safety in automobile, by passing laws, which improve the driving conditions. The status of the road is also a possible cause of accidents. To improve road design and putting the correct warning signal at visible places are important tasks for governments.

First, safety measurement was a concern of mechanical engineers, who dealt with ergonomic issues. Electronic progress and the advancement of information technology converts mechanical safety measurement into electrical, electronic and computing engineers matters. In this section several safety systems will be presented, both mechanical measurement and various sorts of safety applications related to human system interfaces.

Embedded systems are commonly used for controlling the automobile. There are some embedded systems (automated system), where the driver usually is unaware of their existence. But, there are some other embedded systems (semi-automated systems), which the driver must be capable of understanding, managing and handling in order to get a perfect knowledge and control of the automobile. The most important issue is how to maintain safety at the same time that people handle systems.

Designers (for now on, designers can be renamed as constructors or engineers) must develop interfaces bearing in mind how the driver keep his/her attention during driving. Interfaces must not contribute to increased distraction. On the contrary, interfaces have to minimize human errors.

A good interface is the one that is easy to understand physically (high affordance). Actions that driver can accomplish are perfectly defined and buttons used for carrying out these actions are very visible (good visibility). Actions responses of the system are the wanted responses (mapping); there are no possibilities for unwanted events and, eventually, the interface provides information (feedback) about system status and other data, which can be of uttermost importance for the driver in order to control the automobile in the best manner.

As a general rule, a satisfactory interface design for people with special needs will in general be satisfactory for anyone else, specifically, the average people (Ohlsson, Persson, & Östlin, 2005). The variability of human behaviour, potentials and limitations is huge, therefore it might not be possible to create universal interfaces. However, the potential for universal design is still great in the automotive industry. Each person has different physical characteristic and today people must often adapt to standard design solutions. Standard interface design means a design accepted by the highest possible amount of drivers. The final goal is to receive the greatest possible safety during driving or even when the car is voluntarily or accidentally stopped.
Figure 7. System classification.
3.4 SBD

SBD stands for Simulator-Based Design and it refers to design, development and testing of new products, systems and applications. Now we will focus on the automotive perspective. In order to present the complete concept we will present a cycle that goes from concept to final product. In the Figure 8 an overview of the main steps are presented.

![Figure 8: SBD design loop](image)

As we can see in Figure 8 there are many steps to complete the whole cycle and many of them have iterative loops that connect these steps with the previous one. This is related to the last point where we explained it. The iterative design makes the cycle flexible and goal oriented. We may also observe how some of the initial steps are replaced by the virtual prototyping. Thus we can avoid the more costly physical prototyping process. We also mentioned and discussed this concept in the previous section. Using SBD for implementing, testing and evaluating new applications and products is greatly facilitated in the simulator environment that is developed at the Virtual Reality and Simulation Lab that was used in the current study. (Spengel, & Strömberg, 2007)

The iterations are part of the normal procedure of the SBD, and it leads the first virtual prototype towards the final version. One basic point in SBD is to start these iterations as close as possible to the early concept and not so much to the final product. In some cases a prototype could go all the way until the design synthesis stage, where we will obtain an answer such as yes or no. If the answer is yes the additional information obtained during the SBD procedure will redefine the next prototyping step.
It might look like this long procedure could end up being a waste of time and resources. But, the truth is that when we repeat the different steps with a more elaborated prototype we will be able to re-use or even copy the previous work we have done. We will add new details every time we start a new round in the circle, but it does not necessarily involve creation or invention of something completely new.

This conceptual model is our reference in the laboratory. The SBD describes a process closely related to the software design, the one mentioned in the previous section. We could compare the 4 iterations of the SBD model with the 4 different steps on the concept of iterative design in software development. (www.acusim.com)

3.5 Virtual prototyping and iterative design

One of the main features of SBD is the iterative design process. This concept is closely related to the software design framework. SBD implies an iterative process, which is also considered as one of the keys of software design. However in the mechanical engineering field, it has not been frequently used, since physical prototypes are too expensive. There is no way of having changes in an iterative process due to economic reasons. The iterations imply the existence of prototypes. The first basic design could be considered as the first prototype and then we will make changes in this model by creating new prototypes. Our simulator and SBD in general have one of its main features in the iterative approach. It also addresses the other requirements such as end-user and the use of component-based architecture, which will facilitate the system extension, a better understanding and the possibility of reusing development steps during the complete life cycle. This is an important part of the SBD methodology. This concept has been applied to our simulator resource at the University. (Alm, 2007)

Virtual prototyping replaces some of the initial stages of physical prototyping process, which is very expensive when developing a new product. The most important idea behind virtual prototyping is the use of digital product models. In the first steps of the product development the idea is to use it before any physical prototype has been produced. With virtual prototyping we save time and money. We also save a lot of effort and it offers a faster and easier testing and evaluation process. For instances, if we want to design a new main instrument, before any physical model is carried out in the vehicle, we can create a prototype and evaluate it and test it in advance. The virtual prototypes constitute preliminary steps before we obtain any physical product. In the software field a prototype is just a software program, since there is no physical goal. When referring to SBD we have to mention that the goal can be both a mixture of hardware and software implementation. During our project we have placed both new hardware and we have implemented software. (Alm, 2007)
In the SBD approach it make no sense adopting the sequential tradition from the mechanical engineering field. The virtual prototype concept is a software-based tool. In Figure 9 we can observe the way we go through the different design steps. It is a sequence of steps in a sequential order including some control loop.

![Figure 9. Concept of iterative design in software development [2].](image)

In Figure 10 different parts of the IVS is described and the different approaches we need to implement for the simulation. This conceptual model has three layers.

![Figure 10. Model for computer-based in-vehicle systems.](image)
The first layer is related to peripheral parts of the system, focused on collecting data and realizing actions. Here we can find the sensors, all the devices that can collect data. This data can be collected from the external information or even from the driver. In the SBD all these data comes from the data simulator, apart from that we can collect from the driver. Some sensors would not make any sense in the simulator, since we can have all the information directly from the simulator. We do not need a real GPS, since the simulator is not moving and we would not need any cameras, since the information appears on the displays.

We can also find the actuators, that is, the sensors in the simulated world. They have no physical meaning and they are replaced by software, for instance, the speed pedal or the brake. The information is transferred to the simulator and the simulator will influence the vehicle in the simulated world. All these procedures have to be carried out in real-time.

The second layer is the reasoning part of the system. This is the core of the model. This layer will receive all the data from the cockpit, the driver and all the information from the virtual sensors. All this data will be processed and then the simulator will influence the environment and the vehicle. The content of this level is completely based on software and it is built in a programming language. In terms of SBD this level would be like a function in a real system and in the best case these computer functions could be transferred to the final product platform.

The third one is responsible for bringing the operator into the loop. We also call this layer the interface level. It can be considered as the beginning or the end of the system. The end refers to the system output, and the beginning to the input. In terms of Simulator-Based design it is crucial to support design activities by having programmable solutions at this level. (Alm, 2007)

3.6 System Theory

Every system must be viewed in a holistic perspective. Most of the systems should be formed by adding small system components, but people must see it in a general view. Fredich Hegel (1770-1831) wrote the following sentences related to the nature of the systems. The important is the whole, and “the whole is more than the sum of the parts” (Skyttner, 1996). Anyway, the parts must be understood by itself, without studying the whole and the parts are related to each other and cooperate together in order to obtain the final purpose. The whole defines the scope and the nature of the parts.

The nature of a system is defined by its properties. Ludvig von Bertalanfly (1955), Joseph Litterer (1969) and other people joined their conclusion to create a property list depicting an elaborated system view:

- Interrelationship and interdependence of objects and their attributes, holism, goal seeking, transformation processes, inputs and outputs, entropy, regulation, hierarchy, differentiation, equifinality and multifinality.
A system can be viewed as a black box, which contains a set of input and outputs. The inputs are data coming from the environment and the device controlled. The output is the result of the different processes occurred inside the black box. Sometimes, it is important to record the outputs and feedback them to go on with the process.

Many system theories to explain several mechanisms observed in nature haven been figured out. Some theories are “General Living Systems Theory”, “The Geopolitic Systems Model”, “The Control Theory”, “Quantal System”, “Categories of Nature”, “Viable System Model” and so on.

We will focus on theories and processes related to systems that we can find in an aircraft or other vehicles. Developing this idea, engineers have created more or less autonomous systems by means of artificial intelligence known as expert systems. They can obtain a result but, very often, they must be fed by the knowledge of human specialists. The weakness of this system is that they are not capable of analysing all the Human Factors so it can produce an undesired event. To finish this section we will explain how an expert system is like. It is usually made of the following parts.

- “A knowledge database containing domain facts, the heuristics associated with the problem and the understanding and solving of the same”.

- “An interference procedure, or control structure, for the choice of relevant knowledge when using the knowledge database in the solution of the problem”.

- “A working memory or global database, keeping track of the problem status, the input data for the actual problem and a history of what has been done”.

- “An interface, managing the interaction between man and machine and working in a natural and user-friendly way, preferably with natural speech and images. The interface must permit new data and rules to be implemented without change of the internal structure of the system” (Skyttner, 1996).
Chapter Four

In this chapter we describe the facilities at the Virtual Reality Laboratory at Linköping University. The simulator equipment and hardware is described. We also focus on the software used in the simulator, mainly ASim provided by ACE simulation.
4. The simulator facility at LIU

In 1996 a Virtual Reality & Simulation laboratory (VRS) was founded at Linköping University (LiU). The department responsible for this new lab was the Division of Industrial Ergonomics. During the first years the purpose was to explore the potential of Virtual Reality (VR) in different areas but since 2001 the lab has focused on developing and evaluating in-vehicle systems from the HMI point of view and following the SBD procedure. To make this approach possible a car simulator facility was built, which is further described in the following.

![Figure 11. View of the simulator cockpit at LiU[7].](image)

4.1 Simulator equipment and hardware

The VR lab at LiU consists of three major parts; simulator hall, control room, and a virtual prototyping area. While the two first resources are specific for the simulation activities, the prototyping area is generic. This means that also other virtual prototyping activities than those for car applications are carried out here. This virtual prototyping workshop consists of five PC workstations. Here students and researchers have access to advanced tools for design and prototyping activities related to HMI design. The idea is to use these computers for this purpose and let the computers in the simulator only working for the simulator itself, in activities such as implementation or experimental sessions.

The simulator hall contains what we call the driver environment. It consists of the physical cockpit with the steering-wheel and pedals of a rebuilt Saab 9-3, video projectors with belonging screens for environmental presentation, and loudspeakers. The cockpit is built according to the glass cockpit concept. Actually, there are two
cockpits available, in addition to the personal car cockpit there also is a truck cockpit, built on the same principles. However, since the Saab cockpit was used in this project we based the following description on this application.

4.1.1 The simulator cockpit

It is far from trivial to convert an ordinary car to a simulator cockpit for design studies. Traditionally, most simulator sites just bring in a serial car, which is possible to couple to the simulator software by some interface. In our case all car functionality has to be programmable, which means something more complicated. We cannot go into all details, but will mention the major parts. At first, we will present the steering system.

All mechanical parts of the steering system, except for the steering-wheel and the steering column is excluded and a programmable torque motor is connected to the column to give tactile feedback to the steering-wheel, relevant for the specific vehicle dynamics, in this case of the Saab 9-3. This also means that it is possible to add other tactile feedback overlaid the ordinary signal, for instance, to simulate tactile warning information. This arrangement could be seen in Figure 12.

Figure 12. Replacement of the ordinary steering system [7].
The picture of what is under the hood was taken relatively early in the cockpit building project, which was carried out some years ago. Since then, two major components have been added. First a completely new electrical interface, which among other features makes it possible to easily change the functionality of button-based functions in the cockpit. The second component is a projector in upward position, which produces the head-up display (HUD) image. There is a corresponding hole in the hood, which allows for displaying the image on a horizontal screen attached to the ceiling of the simulator hall. This image is then reflected to a mirror on the cockpit dashboard, which finally sends the image to the windshield where the driver could see the HUD information. This somewhat complicated arrangement is made to obtain beam distances from the projector to the windshield that result in a HUD image “at infinity”, which in practice means well away from the car front. The implication of this is to let the driver read the HUD information with no need to refocus the eyes from looking at the environmental information. In a real car HUD application the same goal is reached by other (and more expensive) means.

As could be seen in Figure 11 the main instruments and the center console functions are screen-based which makes it possible to vary the design solutions completely freely. As indicated in the picture the center console solution consists of a touch screen. As described above also a HUD is included as well as a “tactile driver’s seat”. In this seat 16 vibrators (tactors) are implemented in a 4 by 4 matrix. These tactors could be initiated in a number of different patterns depending on the purpose of the signal. This programmable tactile display is controlled by a graphical interface on a screen outside the cockpit in a solution where photo diodes react on each position and turn on or off the tactors in the driver’s seat. Again, this is a typical simulator solution and for a real car other techniques must be used.

Finally, the ordinary loudspeakers in the cockpit could be used together with the surrounding loudspeakers in the simulator hall controlled by a mixer board in the control room. During our project the sound system was used to produce normal engine sound from the simulated own car, but it is also possible to use the sound system for other purposes, like warning information.
4.1.2 Simulator platform and control room

The simulator is running on a PC cluster. It is built on 10 computers. We have to mention that one of them was not operative due to technical reasons, so during our thesis project we worked with 9 computers. Apart from this cluster there is a main server that controls everything during the simulations. Since the basic software principle is modularization, the number of PC computers could change depending on the number of in-vehicle system prototypes or specific measurement systems.

Figure 13. PC cluster arrangement [7].

The control room is where we have the complete control of all displays in the driver environment. We also have all the information related to driving scenarios, including the traffic situation. Here we control everything even beyond what is seen on the environmental screens and the cockpit instruments. We can also communicate with the driver from this room and it is possible to “drive” from the main computer, just in case we need to do something in the simulator and we do not have a driver. Some key measures can also be followed on-line. These measures might be different log data from the simulator software or other measurements like for example eye tracking data.

Figure 14. Control room [7].
4.2 Simulator Software

This section has the purpose to present major software tools used for virtual prototyping and simulation activities in the VR lab at LiU. The output from using these tools are prototypes of new in-vehicle systems but also traffic scenarios to drive through in order to investigate the benefits of the system in question. What is not covered here is the modeling work in the production of virtual environments like cities or highways. These activities are excluded since neither our nor the parallel project included such activities since we had the possibility to reuse already produced models. The concept of reusing already developed resources is a cornerstone in the simulator-based design approach.

4.2.1 ASim

The heart of the simulator resource at LiU is the simulator software ASim. ASim was developed by ACE Simulation, a spin-off firm from the VR lab. Today the name of the company is HiQ Ace and has about 30 employees. Since early 2008 the firm is a subsidiary within the HiQ group which is quoted on the stock exchange in Sweden. Also Saab Automobile uses the ASim software in their corresponding simulator resource which facilitates the collaboration between the labs. The main characteristics of ASim is its modularized architecture. The different modules communicate with each other through the included communication network functions and every module has access to all information in the simulation and can deliver its own data the same way. The simulator kernel is the unit that runs all the communication processes which means that every module is synchronized and can be logged with the rest of the simulation data. This approach also opens for implementation of user-made modules which could work seamlessly together with all other modules. This is the basic prerequisite for Simulator-Based Design. Without this architecture the only possibility for the “site owner” to implement new functions, often a new in-vehicle system is to go back to the developer and to make the necessary functional changes.

![Figure 15. Schematic view of ASim [8].](image-url)
Thus, the goal of the ASim software is to support two main areas that are crucial in the product development process of vehicle systems. These aspects are fundamental within the accelerated approach to product development regarding simulator-based design. These two areas are:

1. Concept evaluation: It refers to the specific purpose of the system and system requirement validation. This concept is supported by virtual prototyping. Product developers create a software based virtual prototype of the target in-vehicle function. This conceptual virtual prototype is evaluated in the specific environment for fast system design, product requirement feedback and user acceptance.

2. Solution evaluation: It refers to the system solution verification and validation of the “final” prototype. This concept is supported by virtual testing. ASim software and environment is meant to generate input to and measure output from the different experiments that is carried out for the system evaluation. This could be done with total control of the test situation. And of course we have to mention that these tests will be completely safe from the driving perspective even if hazardous events will be included.

We also want to mention the features of ASim software that fully support simulator-based design:

Dynamic and open architecture: All the modules can communicate with each other through the RT communication functions provided by the simulator kernel. The user (the organization which is responsible for the development of the specific in-vehicle system) may add new modules if needed. All the modules have access to all the information available and they are automatically synchronized. The different modules and their processes can run on different computers in the network. We also have the possibility to distribute the processes as we wish in the different workstations.

Software based modular functionality: The simulator consists of physical hardware such as data screens, speakers, steering wheel, cockpit, pedals, etc. The prototyping environment is a set of software based modules that run under the supervision of the simulator kernel. The physical hardware that we mentioned before has direct mapping to the modules in the prototyping facility.

Easy scenario and environment production: There are different functions that will let us set up a scenario easily. These functions are mainly triggers and actions that are provided for control of numerous simulation functions. For instance they may control different situations like traffic behavior, brake failures, automatic mode switching, warning initiation, etc.

Integrated and synchronized data logging: The users are not limited to the set of simulation data given by the standard modules. New user modules can add new data logging sets if necessary. The simulator kernel data log function provides this kind of simulation data.

The environment for the simulation consists of different applications and objects. These applications and objects will have their own responsibilities and
purposes. There is a control device called “daemon” which collects the input data from
the user through the different input devices. In our case these devices can be the mouse,
the keyboard, etc. This data is then moved to the physical properties. The physical
properties refer to the devices that the driver can use in the simulation. The steering
wheel, the gas pedal, the brakes and more functions in the cockpit. Based on the input
data and the data from the environment the simulator will make all the calculation to
establish the positions of vehicles. All the data generated such as the position will be
broadcasted onto the network.

There are different objects in order to represent the driver’s vehicle in the
environment. The cameras represent the driver’s view, so we will have to arrange them
in order to create a realistic view on the screens. The vehicles and other objects are
represented with graphical objects. All these objects have their own positions in the
world relative to the position of the dynamics model. *(ASim Immersive 5.0 tutorial)*

![Figure 16. ASim interface user.](image-url)
Chapter Five

In this chapter we describe the methodology used during the realization of this master thesis including information gathering.
5. Methodology

No specific methodology has been used outside the SBD approach. We have followed the SBD concept to get insight the whole process. Information gathering has been quite important during the project and it is described later on in this chapter.

During the time that we have been working in our Master Thesis we have done different activities in order to get more information and education in the fields related to our project. These activities are described in the following points.

5.1 Human-system interaction course

We have taken the course Human-System Interaction (TNK047) at the Department of Science and Technology (ITN), Linköping University. The teacher responsible for the course was Ivan Rankin. During this course we have read the book: *The Design of Everyday Things, Donald A. Norman* printed out by MIT Press in 1998.

The aim of this course was to provide knowledge on factors that are relevant when planning, implementing and evaluating technical systems that are considered human-in-the-loop systems. This means that humans play a central role in the system. In the course we have attended to seminars and classes where we have discussed about concepts such as: human cognitive processes, usability criteria, interaction among humans and interaction between humans and systems. We have also seen the different interaction stiles, design principles and evaluation methods. The content of this course is more thoroughly reviewed in chapter 3.

5.2 ASim course

In order to learn how the simulator works we had an introductory course with Calle Isaksson from ACE Simulation. During two sessions Calle guided us through the whole process of the simulator. The ASim software is described in the previous chapter 4. After the introductory course and our gained experience from the months spent in the simulator site, we now are able to work with this software. This means that we know how to implement user-made modules and how to create different scenarios and thereby simulate different driving situations.

5.3 Flash Introduction course

One of our main goals in the project was to use the new Flash ActionScript 3.0 tool for virtual prototyping. Magnus Grundberg from Pixcode has been responsible for the introduction course and training sessions. He is a programmer and graphics designer at the company Pixcode. This company also works with Saab and more specifically Saab’s simulator.
5.4 Visit to the Saab simulator in Trollhättan

In March we have visited the headquarters of Saab in Trollhättan. The visit consisted of two main parts. We did a guided tour in the company facilities where we observed all the manufacturing process of the vehicles. The guided tour showed the visitors the whole process of car manufacturing. Since the very beginning when the first pieces are assembled until the end when the vehicles go through the final tests. The other part was to visit the driving simulator, since simulation was the most important issue in our thesis. The visit was conducted by Anne Johansson from Saab and Henrik Bergström from Pixcode. They guided us in their simulator facilities and explained their solutions. We compared the different solutions and applications in their simulator with those in the LiU simulator and discussed the background for their choices. In Figure 17 we can see a picture of the simulator site at Saab and as could be noticed they use a complete vehicle. The other difference is that they also have a circular screen, instead of different panels like we have in our lab. They also use 18 computers. Their budget is quite bigger than ours, so we could say that their fidelity is quite good. In terms of validity the results are quite similar.

Figure 17. Simulator facility at Saab [9].
Chapter Six

This chapter contains all technical aspects which have been developed to reach all goals proposed at the beginning. Computer optimization, rear mirror and Flash Action Script 3 application’s (like the cockpit) are the primary topics covered in this chapter from an implementation and assembly point of view.
6. Implementation

In this point we will present the three different tasks that we have performed during our thesis. There are three different fields where we have been working: Computer optimization, rear mirror view and migration of the main instrument code from ActionScript 2.0 to ActionScript 3.0.

6.1 Computer optimization in the Simulator

The computer network of the simulator consists of ten computers and one server. During this thesis we have investigated the possibility to decrease the number of computers.

6.1.1 The computer network in the simulator

Reducing the number of computers can very useful because budget for the laboratory is limited. It is not suitable to follow the generic principle to add an additional computer every time a new major function is to be implemented is really expensive. In our project we should implement a new application, the rear mirror and at the same time investigate the possibility to reduce the number of computers. So, in our project we have tried to share computers for more than one application. It is also a good idea to leave computers for next applications to come, or just to save the need of a new computer when implementing a new application. During our thesis we have managed to reduce the number of computers in the simulator. However, the results are not always as good as we expected, in this point we will explain the procedure that we followed to reduce the number of computers and our conclusion.

6.1.2 The new configuration

The first step was to decide which computer we would like to share and which applications could be candidates for this. Our idea was to use the same computer for the right forward and for the right side environment screens, as you can see in the figure. Thus, we will leave one free computer to use for the rear view mirror.

![Figure 18. Projector panel distribution.](image)
Consequently, we set up the graphic card to control two camera display applications in the same central processing unit (CPU). We will go deeper into the graphic card later on. The point is that we have two monitors (TFT) for one CPU so we figured out which connections and settings are necessary in order to split the images and the use of the CPU. For this purpose, different hardware is necessary. The elements which we have added to the existing configuration are two switcher (VK-222) and DVI-VGA connector. The VK-222 switcher can be viewed as splitter also. The splitter VK-222 allows for obtaining the same video signal in two different outputs. Actually, it has two inputs and two outputs. Pressing one button or not makes possible to select one input and presents this input signal in both outputs. This is necessary in order to send the signal to both monitor and projector. The DVI-VGA connector is related to the graphic card output. Currently, the output of graphic card used to be VGA (Video Graphics Array). Since some time ago, the DVI (Digital Visual Interface) port has begun to be used in graphic cards because its better features and performance. Definitely, the DVI port has replaced the VGA port on the market. VGA connector only can transmit analogical signal and DVI can transmit both digital and analogical signals. DVI is better because a TFT monitor plugged into a DVI port produce the maximum quality possible pixel to pixel. There is a necessity of transforming the digital output signal to an analogical signal because the monitor used in the simulator has VGA input. The new configuration is explained in the next figure:

**Initial proposal configuration**

![Diagram of the initial proposal configuration](image-url)
6.1.3 Sharing the computers

In order to get the best performance from the shared computer we wanted to distribute the processes among the computers. This distribution is carried out from the computer server: ASim Server IP 101. For accomplishing this task, it was necessary to run the application of the simulator. When the scenario is running, the user has to move on all the required processes to the different computers looking for a trade-off in the frame rate.

There are some process belonging to windows and other applications that can not be moved. Possible processes that could be moved are related to the simulator application. The frame rate is the most important parameter that allows us to evaluate the capacity of our solution. We should consult and observe the frame rate of every computer in order to get a diagnostic of the situation. Another interesting parameter to be observed in the CPU is the percentage of CPU in use. This percentage is given by the adding of all the percentages of CPU in use relative to each process running in each CPU. Looking at this table, it is possible to decide which process could be sent to other computers with a low frame rate and a low percentage of CPU in use. This table can be shown selecting the option process from the task manager. The task manager can be selected by clicking the next keys in a row: Ctrl-Alt-Supr. Once the task manager is opened, we only have to press the process window and observe the CPU column.

Let us go back to the frame rate. It can be viewed pressing down the keys Ctrl-S. Frame rate measures the frequency of updating for the screen. Usually, a good frame rate for a display application is around 60 Hz. This rate is enough to be viewed as a continuity for the human eyes which means without flickering. Here, with two applications running the frame rate decreased to 15-20 Hz when the two environment screen applications shared CPU. This was considered not good enough for the drivers in the simulator. For this reason, moving processes was one alternative, probably supplementary solution was tested. It was not enough so, the next step was to reduce the resolution of the screen. The frame rate increased to 30 Hz but, on the other hand, image quality got worse. This could be acceptable for not detailed environments like in an open landscape, but for city driving it was not good enough.
Here it is important to point out that all environment models are based on polygon structures. These polygon structures influence the percentage of CPU in use and the frame rate directly and the more details in the environment the more polygons must be used and as a consequence the frame rate will decrease. To deceive the human eye the frame rate must be at least 35-40 Hz and that was not possible to achieve for all surroundings. After all tests of this sharing concept the decision was to keep the existing approach with one camera view for each computer. The final solution is presented in Figure 19 below:

![Image of a dual display solution from just one computer](image)

**Figure 20. Sketch of the final configuration.**

### 6.1.4 The graphic card

The graphic card used in the computer is the Nvidia Gforce 3 6800 GTO. Since we wanted to have the rear view application shared with the right side view we should have two different views running on just one computer to produce the right side environment view and the rear view on the LCD monitor. Also in the control room we needed to have the two views on two different LCDs. For this reason we created a dual display solution from just one computer. In order to get the dual display functionality it was necessary to access the properties of the graphic card and also a new configuration of the cables for the PC cluster and the control room.
6.1.5 Conclusion

After the new configuration was set up we analyzed the results we had achieved. The simulator was working properly but the frame rate of the computers was not as good as it was before, when each computer had only one application running.

The frame rate depends on the specific graphics to be shown and will not be the same for an open landscape as for a scenario in the city. The buildings and the graphics in the city scenarios are much heavier in terms of CPU requirements.

The maximum frame rate that we have achieved is 30 Hz in the city for the poorest view. For the others screens the frame rate is 60 Hz, so there is a big difference between them. When we were driving we could really tell the difference and it was not really nice for the vision.

Our final conclusion is that with the equipment we have in the simulator during this project it is not possible to release any computers in order to get advantages for future applications. However, future computer upgrades may solve the problem.
6.2 Rear view augmentation

6.2.1 Mirrors

The standard mirror equipment for cars is to have one mirror at each side of the car and one inside the cabin. Heavy trucks may have more mirrors to compensate for the height of such vehicles. Here we will concentrate on the side mirror of a personal car since this is in focus for this project.

A wing mirror (or side mirror) is the one located on motor vehicles for the intention of making easier the driver see areas behind and to the both sides of the vehicle, behind the driver's peripheral vision. A side mirror is located (usually) on a wing or equivalent part. Sometimes a wing mirror is also named a side-view mirror because it is developed or mounted on the side of the automobile. Car manufacturers allow mirrors to be either adjusted by hand or electrically to fit the size and position of the driver.

A mirror a surface that is able to reflect sufficient no scattered light to materialize a representation of an object in front of it. Mirrors are not only simples ornament or decoration objects, they are needed for safety reasons. Mirrors loyally reflect or present a true picture of something and can lead the drivers in their plans of changing lane or performing an overtaking. Drivers can mount to their cars unlike sort of mirrors with different objective. These auto mirrors contain towing mirrors, automatic dimming rearview mirror, side mirrors, wide-angle mirror, power mirror and many more.

6.2.2 Supplementing alternatives

Recently, rear-view video cameras have been assembled into a lot of novel types of automobile, such as the Mazda Hakaze Concept. This was as a response to the rear-view mirrors' incapacity to display the area straightly behind the car since the rear trunk is blocking out a distance as much as 3–5 meters (10–15 feet). As a tragic consequence of this weakness, about fifty small children per year are killed by SUVs in America. This fact occurs due to driver is not able to perceive them in their ordinary rear-view mirrors. These camera systems are typically mounted to the rear fender or other lower parts of the car. Video cameras are also used for covering the so called blind spot, which is the area between the angle covered by the ordinary side mirror and the normal angle of sight of the driver.

There are other solutions available today for the same or similar purposes based on other sensor technologies like ultra-sound. However, most of these new systems suffer from a poor HMI design. This fact is one reason for implementing a side mirror solution for the LiU simulator as a part of an oncoming complete solution.
6.2.3 Introduction and LCD monitor

One of the purposes of our project was to place a LCD monitor in the left rear mirror position of the simulator cockpit. Our first duty was to find an LCD that was appropriate for our needs. The complete technical specification of the chosen LCD could be seen in Appendix C. It may be mentioned that another alternative to this LCD solution would be to keep the ordinary mirror and add one or two more screens for environment presentation. However, this solution would become about 20 times more expensive and also put demands for expanding the space of the simulator hall. So, this was no serious alternative. The size of the LCD was chosen as close to the ordinary mirror size as possible. Another consideration was the wish for avoiding buttons at the front frame of the LCD. This was mainly for practical reasons since the plan was to build in the LCD in a cover where the buttons could become out of reach.

6.2.4 The new configuration of the simulator software

The rear view mirror application was decided to run on a separate computer. The entire configuration of the distributed simulator system and the computers was explained in section 6.1. This new configuration affords to have one computer free for the new LCD mirror.

6.2.5 Position of the rear camera

First step is to get the point of reference in which all cameras are situated before modifying by software means their position in the simulated world. Once this reference point is located coupled to the own car position, it is possible to measure where the rear mirror of the car is allocated related to this point. For this purpose, a regular meter has been used. Obviously, this measurement has been carried out over the car which is in the simulator.

The real mirror must be placed one meter over the ground but the ground is not the point of reference in the simulator. After looking at the rest of the camera installed on the simulator the configuration chosen for the rear camera is the one shown next:

![Camera configuration](image)

Figure 21. Camera configuration.
The application developed is based on C++ software programming. An outline of this application is attached in the appendix B because this code is protected by Saab, the company which has the rights. This part of the code is attached in the case other students want to keep developing new performance and applications following this investigation line. Here again it is worth reminding of one key element in the SBD approach – reuse of already existing resources. Thus it is necessary to have this demand in mind in every project, including our.

6.2.6 The solution at Saab

As mentioned above, there are many different solutions for solving the problem of the rear mirror. Saab in their laboratories, have decided to use another solution. They have 2 projectors that show the image in a big screen behind the car, and they have kept the original mirrors of the cars. This solution requires more investment because more hardware is needed. SAAB simulator in Trollhätan Central factory has around twenty computers in the network and by contrast Linköping University has only 10 computers. The budget invested at SAAB is much higher than the one invested at the university. For this reason, the solution proposed in this project is simple and economically more suitable.
6.3 Cockpit

Cockpit is a term defined as: “the small enclosed space where the pilot sits in an aircraft, or where the driver sits in a racing car”. (2005; Cambridge Advanced Learner’s Dictionary; second edition; The Edinburgh building, Cambridge).

This is the basic idea of what a cockpit means. The cockpit that is going to be analysed in this area is a car cockpit. We will not go deeper in all parts which form a cockpit, but we will go slowly into those interfaces between driver and the vehicle. Any physic parts of the cockpit, which are not related with interfaces we are dealing with, will not be explained or described.

Most applications installed in the simulator are not the typical for what we can see today in real car applications. The reason behind this statement is that in our simulator we have a glass cockpit while this concept still is not implemented in real cars. We can see some steps in that direction but most of the car instrumentation is still hardware-based. The applications we find in the LiU simulator have been developed by the efforts of many student and research projects. Speedometer, air conditioning control, level of gas, temperature, blinkers signal, radio, and many warning systems for different purposes. Most of these systems were or still are in real applications. As it was explained in previous chapters, electronic and computer progress including display technology has had and still will have a great impact on the development of in-vehicle systems.

So, in conclusion, the simulator at LiU is more or less crowded with innovative cockpit solutions where some already could be seen in real cars while other concepts may appear in the future. The overarching goal for all projects in the LiU simulator has been to contribute to better situation awareness for the driver, higher degree of flexibility in the interface (do not show everything all the time) by mode-based approaches, and having solutions which keep the driver in the loop in an intuitive way.

In previous projects the cockpit interface resources were developed using Flash action script 2.0. From now on all applications must be performed based on Flash action script 3.0 because it is going to be the new software graphic environment used. For this reason, there is the necessity of transforming all useful designs from previous applications to the new standard. In our work it was really beneficial to understand what was programmed in action script 2.0 since all these applications had been developed and tested from a user point-of-view. Applying the theory of conceptual models described previously, it is useful to obtain the user model and create the designer model later on. The code programming (action script 2.0) was thoroughly analysed to figure out some ideas and algorithms which could be reused in the new programming environment.

The goal was not only to realize applications in Flash action script 3.0, but also to modularize these applications into minor function blocks as much as possible. The purpose of this kind of software design is that future projects could use all modules separately. It makes easier to export and place different applications and it also allows future projects to implement new ideas without changing the entire application. Additionally, dividing applications in several modules makes the programming structure and the code used more easy to understand. From a didactic point of view, it is a good programmer habit to divide applications in code packages in order to clarify the
content. Users who read this code can learn easier and faster how to use a new programming language.

6.3.1 Cockpit file

As it was mentioned before, Flash action script 3.0 is the tool used to generate the cockpit by software. In flash, there are two important files: *.as and *.fla. In one hand, the former one (*.as) is the file which contains all the flash code. It is a typical object oriented programming language which takes similarity with JAVA or C++. In the other hand, the file *.fla is the one which is used to create the graphic representation. Both files are associated each other in order to generate the file *.swf after the compiling process. This is the executable file and must be set up at the computer to manipulate the cockpit. This setting up process is explained immediately.

Two computers are taking part in this process: ASimServer IP 101 and AsimCockpit IP 110. The swf file generated must be placed in the ASimCockpit IP 110 computer. Moreover, it must be allocated in the next folder: c:\ProgramFiles\pxc\FlashWin2. In this folder we will find two files which are used in this setting up process also. They are setup.xml and pxFlashWindow2.exe.

1. SWF file must be generated in the ASimCockpit IP 110.
2. SWF moved into c:\ProgramFiles\pxc\FlashWin2.
3. Change the path in the setup.xml. (Explained later on).
4. Time to run the simulator software in ASimServer IP 101.
5. Execute the file: pxFlashWindow2.exe in the ASimCockpit 110.

From now on the cockpit application is running in the computer. Changes must be done in the properties setting of the windows screen in order to make the application visible on the cockpit display.

6. Display properties =>
   Settings =>
   Resolution:
   Change 1024 x 768 pixels to 800 x 600 pixels.

Setup.xml

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<SETUP>
  <SWFFILE NAME="default_mode.swf" />
  <WINDOW FULLSCREEN="0" SCALE="0" ONTOP="1" HEIGHT="600"
    WIDTH="800" BORDER="1" />
  <SCREENPOS CENTER="0" POSX="0" POSY="0" />
  <LOGFILE LEVEL="1" />
  <TRANSPARANCE ON="1" RED="0" GREEN="255" BLUE="0" />
  <WINCOLOR RED="0" GREEN="0" BLUE="0" />
</SETUP>
```
Implementation

Setup.xml file is used by pxFlashWindow2.exe executable file to make visible the flash application. The user must rename the SWFFILE NAME and write his owned name properly. There are many other properties which could be modified in this file.

6.3.2 Cockpit and events

The cockpit interface designed in this thesis allows driver to set up several options. For this purpose, interaction between user and cockpit has to be feasible. On one hand, the simulator should send to the computers all information and data related to ongoing events, for example, car settings, road status, weather conditions, etcetera. On the other hand, the user must be able to manipulate all the interfaces which are used to control different parts of the car such as the cockpit functions.

For the sake of the programmer or users, it is interesting to know how and which data are sent to computers. These data must be handled by programming software. When the simulator runs on the ASimServer IP 101, one application named PDC.exe is executed on ASimRearleft IP 102. This application shows data sent by the simulator. It is placed in c:\Program Files\AsimSimulator\bin\DPC.exe.

The information is differentiated into events groups and each group is formed by several events. Each event should be read as a function of the type of data which is being sent. We will see further what is happening by an example later on. The information represents different aspect of the car and its environment as: audio display, steering data, camera, collision, position, graphical object, engine, sound, video display, etcetera. Users should understand this file in order to obtain the relevant data.

To clarify the file mentioned before, a practical example is explained next following the figure 26. First of all, all the event groups are displayed on the left part of the file. For this explanation, the event group DINAMICS_DATA has been chosen. Each group has a specific number. Moreover, DINAMICS_DATA group is numbered by 1096. This information is allocated at the information part of the file. This group is composed by four events. They are also numbered from 0 to 3. In this example, the event ENGINE_DATA is chosen. It is referenced by the number 3 which is possible to see at the information part of the file. Last thing to analyse consists of reading the information stored in the variables. The reader must know that the simulator sent the information to the computers in a row. For this reason, all information has to be read but only use the required information. In this case, the gear information is pointed out. This data is an integer value. The variable must be used as it is shown in the code later on:

```javascript
if (tGroup==1096 && tEvent==3)
{
    var name:String = pxFlashWin.readString(ev._byteArray);
    var type:Number = ev._byteArray.readByte();
    var rev:Number = ev._byteArray.readDouble();
    var tor:Number = ev._byteArray.readDouble();
    var gear:Number = ev._byteArray.readByte();
    ....
}
```
Users must notice that there are different kinds of variables: int8 (byte), int32 (double), float, string, etcetera. Each one should be read in a different manner. To find out more about this we refer to appendix A where more variables are used and commented in the code. Furthermore, investigate more playing with the PDC.exe file.

**PDC file capture.**

In this file all the data coming from the simulator can be found. The different data coming from the car is presented in different data type (float, integer, vector…). These data must be treated in order to obtain the functionality required. The next figure is a capture of the pdc file.

![PDC file capture](image)

**Figure 22. PDC file.**
Later on, a general and basic scheme of how a programmer must begin to develop an application based on Flash code is shown. Users have to create their own code for the specific application in mind. New user code should be added to the main program.

In the cockpit application, buttons placed on the steering wheel must be named to control the cockpit. These are the buttons:

Figure 23. Wheel and buttons.
The string (kind of data in language programming) names written on the picture before mean the names sent by simulator to the computer when each key is pressed. For the sake of making a difference between a pressed key (button down) and a un-pressed key (button up), the simulator sends two data: the string name and a number indicating the state, `1` when the states is button down.

![Diagram of buttons names]

**Figure 24. Buttons names.**

The five values: “CLR”, “NXT”, “SRC”, “VolDwn” and “VolUp” are used to change the cockpit Flash application manually, that is changing mode. By using these keys it is possible to understand how the cockpit works. The blinkers are not shown on the figure above but they have also names associated, “RIGHT” and “LEFT”.

### 6.3.4 Cockpit final design

This part of the thesis is based on “*Interface Design in an Automobile Glass Cockpit Environment*”, a master thesis accomplished by Michael Spendel and Markus Strömberg in June 2007, Linköping. This thesis project was carried out using Flash action script 2. Most of the objects created in their thesis have been possible to reuse in our project. All the graphics they developed were created from scratch in Adobe Photoshop CS2.

**Overview**

First of all, we would like to point out one important difference that has been carried out in this project. The multi-button on the left side of the steering wheel is not used anymore to move throw the cockpit in order to select the different categories implemented. It is more intuitive and easier for the driver to manipulate the cockpit interface with only one hand, the right. From now on, the multi-button on the right side
of the steering wheel is used to scroll through all the cockpit options. The cockpit interface is used to present information of interest for the driver and a platform for driver input functions. The functionality of the interface is divided into four modes: night, sport, economy and default. At a lower level there are seven categories: climate, media, navigation, source, ccr, phone and mode. This application is formed by one file .fla “default_mode.fla” and several *.as files where the code is allocated: main, climate, navigation, source, phone, ccr, modes, media and src. The purpose of creating different files is to modularize, as much as possible. Obviously, there is a fundamental file which controls the rest of the files. This file is the main.as. There is no possibility of developing different files without one which control the rest of them. The others files content information related to each own module implementing functions. When the program flow requires any actions or function from a specific category, the function needed will be called by the main file and brought from the others files. It is not possible to use these files with absolutely independence. The only way to build seven different interfaces (one for each category) consists of programming seven main files totally independently each other. For this reason, our blocks are oriented to clarify and simplify the code.

![Program flow chart](image-url)

**Figure 25. Program flow chart.**
6.3.4 The four modes

This design allows driver to choose the possibility to change its appearance depending on external circumstances or depending on what the driver desires. The modes are: sport, night, default and economy.

Modes are sharing the basic key information. It is a task of the programmer to make a software program which is able to use these keys for each mode. Let us begin with the default mode.

Default Mode

This is the mode designed to appear every time the application start. From this mode, the user can move through the flash application and choose different options. The cockpit sends visual and audio feedback to the user.

![Default mode](image)

**Figure 26. Default mode.**

The screen is formed by several movie clips (typical objects created in Flash Action Script). Movie clips are graphical objects composed by a few pictures with format *.png. The purpose of this pictures combination is to obtain the vision that these objects are digital displays. The different pictures overlap each other in function of how the code program has been developed. In the default mode we can observe the most vital features of the system.

A fuel indicator is on the left of the screen. Actually, the purpose of this design was to providing a more naturalistic view of the fuel situation than what we usually find in real cars. This indicator is not a movie clip. Let us explain all the movie clips allocated in this screen.
The blinkers, the speed needle, default buttons, speed counter, revolution per minute (allocated on the right part) and the menu buttons which allow drivers to choose the different categories are also movie clips. All these buttons are controlled by the multi-button function situated on the right part of the steering wheel as was seen in figure 2 and 3. As mentioned in previous section, these buttons are handled by the buttons named “NXT” and “SRC” instead of “Up” and “Dwn”. The improvement of this measure is that driver only should be taken care of the right hand. It allows user to get more concentration in other thoughts and actions. The feedback of the cockpit is based on the vision capability of the driver. It is used a visual channel.

The blinkers are two movie clips instanced with the names: “blink_right” and “blink_left”. The sound is also another movie clip named “blink_sound”. When any blink is activated just moving up or down the handle and event occurs and the information is sent to the computer to be analysed by the flash application. The code used to manipulate is coming next:

```javascript
If (tGroup==1099 && tEvent==4)
{
    var id:Number  = ev._byteArray.readByte();
    var desc:String  = pxFlashWin.readNullString(ev._byteArray);
    var tName:String  = pxFlashWin.readNullString(ev._byteArray); // Device Name
    var tState:Number  = ev._byteArray.readInt(); // State 0 = ButtonUp, 1 = ButtonDown

    selection_mode(tName,tState);
    blinkers_function(tName,tState);
}
```

The procedure for this event must be written on the part related to the event that has just started. In this case and for all the buttons placed on the steering wheel, the event group is the 1099 and the event is the number 4. Looking at glance, it is possible to observe that the information passed by reference to two functions, one for the blinkers and the other for the mode.

The “blinkers_function” is going to control the sound and movement of the blinkers. Inside this function the movie clips objects are manipulated using the input parameter “tName” and “tState”.

The “selection_mode” is almost the most important function created in order to control the program. This function decides which category or mode is going to play. It manipulates the information provided by the multi button function.

The “CLR” multi-button needs an extra code than, whichever other button. This is because the button is used such as to select as to exit from the different modes and categories.

The needle is a movie clip which is treated in a special manner. The movement of the needle is indicated by the function “pxSmoothRotation” which has been defined and implemented in a “PixCode” library.
public class main extends MovieClip
{
    ...
    var _speedNeedle:pxSmoothRotation;

    public function main()
    {
        try
        {
            ...
            _speedNeedle = new pxSmoothRotation(this.needle, -60, 119, 180);
        }
        catch (error:*)
        {
            pxFlashWin.setDebugText("main() catch() : " + error);
        }
    }
    ...

    private function netDataMessCallback( ev:pxNetDataMess )
    {
        if(ev._dataProvider == "asim")
        {
            var tGroup:int = ev._orgObject.GROUP;
            var tEvent:int = ev._orgObject.EVENT;
            if(tGroup==1096 && tEvent==0)
            {
                ...
                var speed:Number = ev._byteArray.readDouble();
                _speedNeedle.setValue(speed);
                ...
            }
        }
    }
}
Night Mode

The visibility is a feature that changes obviously from day to night. For this reason there is an option to change the brightness and the colour of the important elements of the screen. The user can select this option. The main parts of the night mode are the needle and the mileage counter which change their colours in comparison with, for example, the default mode. Also the amount of information is reduced in the night mode.

![Night mode](image)

**Figure 27. Night mode.**

The colour of the needle changes using the property “gotoAndStop”. The needle is a movie clip object defined by four similar pictures, where each picture has its own colour. Thus, the needle has four colours, depending on the mode selected and also the speed limitation. The economy, sport and default modes use the same kind of colour for the needle: white under the speed limitation and red if the speed limitation is exceeded. The night mode has green colour for under limitation and yellow colour in case of exceeding. The speedometer will be further explained later on.

The date is an object of “DATE” class, it is defined in action script 3 tools. It is very easy to manipulate. First of all, it is necessary to declare a variable of the object “DATE”. Next step consists of calling at the constructor of the class in order to create a possible value in the variable.

This statement must be written inside the “try area” of the main program. It produces that the function “Update” is called every time.

```javascript
this.addEventListener(Event.ENTER_FRAME, Update);
```

The function “Update” modifies, when is called, the object “DATE” which has been created inside the function.
private function Update(ev:Event) {
  var tDate:Date = new Date();
  this.txt_Time.text = tDate.getHours().toString() +":"+ tDate.getMinutes().toString()+":"+ tDate.getSeconds().toString();
  if (varmode==1) {
    txt_Time.textColor=0x00ff7e;
    temptext.textColor=0x00ff7e;
  } else {
    txt_Time.textColor=0xffffff;
    temptext.textColor=0xffffff;
  }
}

The variable “varmode” is an integer number used to distinguish between the night mode and the rest of the modes.

Another feature of this mode is that the default buttons, which indicate the possible function of the different modes, disappear. The purpose of this is to create a very basic, unique and easy night mode in order to not distract the driver since during night-time driving it is especially important to be concentrated on the road and traffic situation. However, if a category is selected even during the night mode, the buttons appear to to give the driver feedback on the mode manipulation.

The property “visible” allows and object to be hidden or not. This property in fundamental in the development of all modes and categories since it makes it possible to distinguish between the conditions.

*.visible=true; the object is visible.
*.visible=false; the object is hidden.

Sport Mode

The main differences between the sport mode and the default mode are two: the default buttons have been changed by a movie clip. This new movie clip indicates the state of the gear. The other change is the gas deposit picture. In this mode, the gas icon has been replaced by the revolution per minute object. This new movie clip is similar to the one allocated on the right part of the display.

In the same way it is possible to read the speed in variable, both gear and revolutions can be achieved in two variables: “rev” and “gear”.
if(tGroup==1096 && tEvent==3)
{
    var dat1:String  = pxFlashWin.readNullString(ev._byteArray);
    var dat2:Number  = ev._byteArray.readByte();
    var rev:Number  = ev._byteArray.readDouble();
    var dat3:Number  =  ev._byteArray.readDouble();
    var gear:Number  = ev._byteArray.readByte();
    rpm_function(rev);
    //this is for the turbo in the sport mode
    if (gear != prevgear)
    {
        turbo_temp.nalanim.gotoAndPlay(1);
    }
    prevgear=gear;
    ...
}

public function rpm_function (rev:Number)
{
    if (rev == 825) rpm.gotoAndStop(1);
    if (rev >= 1000) rpm.gotoAndStop(2);
    if (rev >= 1500) rpm.gotoAndStop(3);
    if (rev >= 2000) rpm.gotoAndStop(4);
    if (rev >= 2500) rpm.gotoAndStop(5);
    if (rev >= 3000) rpm.gotoAndStop(6);
    if (rev >= 3500) rpm.gotoAndStop(7);
    if (rev >= 4000) rpm.gotoAndStop(8);
    if (rev >= 4500) rpm.gotoAndStop(9);
    if (rev >= 5000) rpm.gotoAndStop(10);
    if (rev >= 5500) rpm.gotoAndStop(11);
    if (rev >= 6000) rpm.gotoAndStop(12);
    if (rev >= 6500) rpm.gotoAndStop(13);
    if (rev >= 7000) rpm.gotoAndStop(14);
}

From this part of the code, the function “rpm_function” is called to execute correctly the revolution per minute movie clip.

Looking at the picture above, it is possible to denote how the digital display appears only when the speed limit is over taken. This option is common to all the modes and categories. There is a function which checks and sets up all the time the speed limit which can vary due to some trigger signal allocated in our scenario. In the real world this function could be based on GPS data and a digital road map including speed limitations. The car speed is also sent to this function, once that the speed is higher than the speed limitation the background to the digital speedometer change to red.

If someone wants to know more about how the digital speedometer works, he should take a look at the following functions in the appendix A:

legal_speed_function(round_speed:int, legal_speed:int);
visible_digital_speed();
unvisible_digital_speed();
The objective of the economy mode is to reduce the gas consumption for environmental and economic reasons. A way to achieve such results is to make the driver aware of how his driving performance affects the gas consumption. So, this is the means applied here by the economy mode leaving out other techniques for more active consumption reduction.

This mode has the same structure than the default mode. The only one difference is that an “eco-indicator” has been created. This eco indicator is formed by two elements: one small needle and the circular scale indicating the level of consumption. When the user presses the accelerator “too much”, the indicator changes its colour from green to a yellow warning.
6.3.5 The seven categories

As mentioned above, seven functionality categories have been implemented in the simulator. From an implementation perspective this number could easily be increased following the same steps as for the now existing categories. If this is wise or not is an HMI question, but technically there are no obvious limitations.

CCR

This system stands for Computer Command Ride. It is the updated version of Speed Dependent Damping Control, also known as SD²C. It was an automatic damper system installed in the 1980s and the early 1990s in Cadillac automobiles. The system was updated as Computer Command Ride in 1991. This new system included acceleration, braking rates, and lateral acceleration to the existing vehicle speed metric.

Figure 30. CCR category.
Source

When people are travelling both long and short distances, they like listening music and other things. It can be a method to get relaxed as well as having an exciting company. So, this category facilitates the choice of CD, MP3 or radio as a music or news source.

![Image of source category]

**Figure 31. Source category.**

The radio, CD, MP3 and the volume are included in the same movie clip. As it was mentioned in previous section, the multi-button functions placed on the right part of the steering wheel is shown in the central part of this screen. Five buttons are represented, ‘-’ and ‘+’ are used to increase or decrease the volume and the other two icons are, obviously, used to select the type of source. The central symbol ‘i’ is related to the button CLR and allows driver to exit this category.

It is an important advancement that the radio can be controlled by the driver from the steering wheel and at the same time have a direct feedback on the displays. Some accidents occur when users try to switch the channel or turn the music on/off on buttons located somewhere head-down. This risk is minimized with this solution.
Media

In the previous category the source was selected while this category is used to manipulate the selected source. As can be seen in Figure 36, the basic functions related to the four different modes are still available. The object in which the radio channel is selected or the volume is turn up or down is one movie clip.

![Figure 32. Media category.](image)

Climate

This category allows driver to control from the steering wheel the temperature and also the fan inside the car. Perhaps, climate functions are not so often changed as other categories but, for simplicity, its functions has also been implemented to be controlled from the steering wheel.

In this category the multi-function button allows the driver to manipulate the fan and the temperature. There is one movie clip formed by two movie clips: one of them indicating the fan speed and the other representing the temperature digitally.

![Figure 33. Climate category.](image)
The picture above has been chosen as it appears in the night mode to illustrate how the categories appear depending on the current mode.

**Navigation**

GPS (Global Position System) devices have been more and more common as separated devices or as embedded system in cars. In our simulator a navigation system is implemented simulating a GPS. GPS is controlled by satellite and its accuracy is almost within one meter. GPS are replacing the traditional paper maps and offers several options as the possibility of being used in different languages or setting up for pedestrian, car or biker modes.

The code associated to this category is, probably, the one most difficult. This category starts as the other categories, pressing the CLR key when the satellite is selected. Once than the user has selected the category, the navigation system begins to work. There are a few elements which deserve special attention.

**Compass object.** The compass indicates where the driver is going to, that is, indicates the heading: north, west, east or south.

The simulator sends the heading (hx) to the computer in radians value. There is the necessity of treating the data mathematically and change the data from radians to grades. This process is done using the next line: 

\[ \text{"heading} = \left(\frac{(\text{hx} \times 180)}{3.14}\right) + 180 \].

The variable “heading” must be declared as a number variable at the beginning of the program in where all variables are declared and defined.

\[
\text{compass.compassstext.rotation}=\text{heading};
\]

The compass object is used for more purposes. For example, the selection of an area and the possibility of zoom in or zoom out. At the beginning, when the user has selected this category, there are three buttons which can not be pressed. These buttons are “zoom in”, “zoom out” and “map off”. They will appear as activated buttons when

---

Figure 34. Navigation category.

The code associated to this category is, probably, the one most difficult. This category starts as the other categories, pressing the CLR key when the satellite is selected. Once than the user has selected the category, the navigation system begins to work. There are a few elements which deserve special attention.

**Compass object.** The compass indicates where the driver is going to, that is, indicates the heading: north, west, east or south.

The simulator sends the heading (hx) to the computer in radians value. There is the necessity of treating the data mathematically and change the data from radians to grades. This process is done using the next line: 

\[ \text{"heading} = \left(\frac{(\text{hx} \times 180)}{3.14}\right) + 180 \].

The variable “heading” must be declared as a number variable at the beginning of the program in where all variables are declared and defined.

\[
\text{compass.compassstext.rotation}=\text{heading};
\]

The compass object is used for more purposes. For example, the selection of an area and the possibility of zoom in or zoom out. At the beginning, when the user has selected this category, there are three buttons which can not be pressed. These buttons are “zoom in”, “zoom out” and “map off”. They will appear as activated buttons when

---

67
the “map on” button is pressed. It presents a hierarchical structure with different levels. To return to this level, the “map off” key must be pressed and, like last step, “CLR button” has to be pressed to exit this category. The code used for this purpose is coming next:

```javascript
if(tGroup==1096 && tEvent==0)
{
  var sender:String  = pxFlashWin.readNullString(ev._byteArray);
  var type:Number  = ev._byteArray.readByte();
  var px:Number  = ev._byteArray.readDouble();
  var py:Number  = ev._byteArray.readDouble();
  var pz:Number  = ev._byteArray.readDouble();
  var hx:Number  = ev._byteArray.readDouble();
  var hy:Number  = ev._byteArray.readDouble();
  var hz:Number  = ev._byteArray.readDouble();
  var pitch:Number  = ev._byteArray.readDouble();
  var roll:Number  = ev._byteArray.readDouble();
  var speed:Number  = ev._byteArray.readDouble();
  //Compass rotation
  heading = (((hx*180)/3.14)+180);
  compass.compassText.rotation=heading;
}
```

**Phone**

Since the mobile phone was invented and begun to be used by drivers, there has been an increasing number of accidents with this background. The mobile is a disturbing element, especially when it is not integrated in the car. Nowadays, the authorities have noticed the problem and some laws have been passed in order to restrict the use of mobiles. However, the legislation situation differs a lot from country to country. For example, in Sweden there are no such laws.

The purpose in the LiU simulator in the introduction of a phone category was to see if it was possible by design means to minimize the negative effects of using mobile phones in driving. This investigation remains to be done, but a first prototype is implemented as could be seen here.

![Figure 35. Phone category.](image)
This category includes the most basic and typical element of a mobile phone: one agenda or phone list, and the possibility of picking up and hanging up. The user can choose which people he/she wants to have on the list. The list is predefined in a movie clip object and can not be modified by the driver. This is of course a restriction coupled to the simulator and is not meant as a solution in a real car application.

**Mode**

This is one of the most important categories because allows the driver to choose one of the four different modes described in previous sections. Its functionality is very important but it was easily built. Every mode has its own appearance and special symbol so it is very easy to distinguish between them.

![Mode category](image)

*Figure 36. Mode category.*
Chapter Seven

In this chapter the evaluation of the different applications performed are shown. The testing process has been accomplished using the LiU simulator environment.
Results

7. Results

A test scenario was created in the simulator environment to test the interface solutions. Several triggers were allocated along the virtual test road to check the different functions implemented in the cockpit.

Warnings of velocity and its corresponding prohibition were set up and associated to triggers in order to test the digital speedometer as well as the traditional one. Triggers were initiating speed limitations of 30, 50, 70, 90 and 110 km per hour. The test was satisfactory and speedometers worked perfectly in all of the different modes and categories.

Attending to our own experience, the results of the interview added as appendix C and the opinion of the guys who has been working together with, we can appreciate the interface:

- The speed limit system is clear and efficient. This information system is effective. It shows us with accuracy the excess of velocity allowing the speeding control.

- The different modes and categories implemented are merely informative.

- The symbols and pictures used to build the interface are very representative. A majority of these interface components were inherited from the project “Interface Design in an Automobile Glass Cockpit Environment” carried out by Michael Spendel and Markus Strömberg.

- The simulator and the SBD methodology is a very powerful and helpful tool for design purposes. Also the possibility to collect data full of information is really helpful in order to make the correct design decisions.

- The manipulation of the cockpit interface is very intuitive, that is, the interface is very well-designed and the interaction works properly.

- The design improves the security in the car. Although the design is made in order to create more significant experiment and more significant safety systems and measures, the interface presents safety for itself. The fact that many systems are concentrated in just one point and its handling is only with one hand increases safety.
Chapter Eight

In this chapter conclusions drawn are listed from the results and previous discussions. These ideas and points can be full of interest for future students and can be used as brainstorm to define the purpose of their thesis or projects.
8. Conclusions

8.1 Conclusions

This project has been useful for us to get knowledge about simulation. This was the first time that we had the opportunity to face a car simulation environment and, moreover, an environment so real. The project has been very pleasant because all the developed designs could be tested briefly and it produces a great sensation. The interface design developed is also nice because it is a graphic interface full of pictures, colours, object and so on, allowing interaction between the driver and the car. Sometimes, the project became tough but we could always drive for a moment to get out the stuck. Definitely we have enjoyed working with our mates and performing our duties in the simulator. We have mixed some concepts full of knowledge typically from Mechanical Engineering and other concepts came from Computer Science, Telecommunication Engineering, and Industry.

These are the primary conclusions drawn from this project. The conception of a car and the car industry has changed quite a lot since the car industry is not pure mechanical. The electronic progress and the computer and communication advancements really have influenced the car technology. This trend must also influence the car industry itself and the way to design future cars. The technology is making a breakthrough continuously. These breakthroughs applied to different industry, items, articles and so on, mark the development of the society.

Cars have more and more electronic components, they are becoming more complex. They include new technologies and a raising number of software-based systems. These last systems are very often in the daily life. One important distinction is that software-based systems or computer-based systems use to be real-time systems installed in cars due to safety reasons. All new developed systems are oriented to provide safety and comfort. However, if integration is not carried out properly the result of implementing a new system could be the opposite.

Nowadays cognitive aspects must be included in order to satisfy the demands on modern approaches to Human-Machine interaction. Human factors must be studied to create good interfaces and good interaction design. It must be emphasized that a nice looking interface to a poorly designed functionality is not acceptable and the inverse of this statement gives the same poor result.

It has been really interesting to see how the students work in parallel to develop and improve the simulator. The simulator is used as a tool to coordinate the work between various students so it is useful to increase the skill of working in groups. We think this point is very important because the interaction and communication between employers and people is fundamental especially when students apply for a job and get it in a company.

Technically, we think that the simulator is a little bit obsolete in the sense that the processors of the computers are not the ones used for graphic design and video games in most of the companies dealing with this topics and this fact lead not to share one computer for two different camera views. Analysing this data we back to the idea of
investing money. The simulator is very well done for the technical limitation it has. The budget is limited and students must accomplish a trade-off between quality and technology. Anyway, to invest more money does not mean to obtain better results but it could be a way to reach them.

Computers placed over the room are going very often so slow, it would be better for future projects review computers which are not working properly. It will save effort to the students and it would help students to develop their tasks and projects in a better way.

8.2 Recommendations and future possible developments

Future projects could be focused on the migration of all remaining applications made in Flash Action Script 2 to the most recent version Flash Action Script 3. For example, the project named Blind spot which was made before is very interesting. Its migration to Flash Action Script 3 could be very useful in order to be able to work with these applications in different studies. Students who include some knowledge about software programming in their background will develop this migration faster and more efficiently. Basic knowledge in C/C++ and JAVA would be interesting.
9. References

Printed sources

- Alm, T. (2007) Simulator-Based Design – methodology and vehicle display applications, Division of Industrial Ergonomics, Linköping University, Linköping, Sweden


- Asim information brochure.

- Asim Immersive 5.0 tutorial.


- Mary Jo Davidson, Laura Dove, Julie Weltz (1999) Mental Models and Usability. Depaul University, Cognitive Psychology 404


- Spendel, Michael; Strömberg, Markus (2007). Interface Design In an Automobile Glass Cockpit Environment. Department of Management and Engineering, Linköping University, Sweden
Internet Sources


  Mark S. Shephard. Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY


- http://www.pspc.dibe.unige.it/drvsco/


- http://www.modulobus.org/agilesdu/understanding_spatial_data_usability_definitions.htm

- http://www.CEA.es

- http://www.techterms.com/definition/computer

- http://www.pixcode.se

- http://www.acesimulation.com

- http://www.kramerelectronics.com

- http://www.partstrain.com/ShopByDepartment/Mirror

- http://en.wikipedia.org/wiki/Wing_mirror


References

Figure sources


[3] Colorado State University, Department of Psychology
http://www.colostate.edu/Depts/Psychology/cognitive/labs.shtml


[8] www.acesimulation.se


Chapter Ten
10. Appendix

Appendix A

Introduction to ActionScript 3.0

In the field of computer science everything is moving really fast and we have to adapt to the new realities and we will have to update ourselves many times. Until now we have been used in the simulator ActionScript 2.0 but now it is time to move forward to ActionScript 3.0. The main reason is that Saab will use this new language and since we work right now in some projects together it is a good idea to use the same software so we will be able to use some of their application, and of course our application will be used by them.

The programming language that we have used in the project is ActionScript. It is the programming language for the Adobe Flash Player run-time environment. This language performs interactivity, data handling, and much more in Flash content and applications. It is based on Java so it is easy to work with it if you know some java programming. It is an object-oriented language so we have all the advantages that we have in languages such as Java or C++. This language offers a robust programming model that will be familiar to developers with a basic knowledge of object-oriented programming. ActionScript is executed by the ActionScript Virtual Machine, which is part of Flash Player. The byte code is embedded in SWF files. These are the final files that we will export or we will be working with. They are executed by the Flash Player, the run-time environment. Flash applications are getting more and more popular, specially, on the web. One of the main reasons is that this SWF files are quite small and we don’t need a lot of memory to store them.

We have collected information in the tutorial: Programming ActionScript™ 3.0, 2007 Adobe Systems Incorporated about the new features in this language. The main key features of ActionScript 3.0 are the following:

- A new ActionScript Virtual Machine that uses a new byte code instruction set and provides significant performance improvements.
- A more modern compiler code base that performs deeper optimizations than previous versions of the compiler.
- An expanded and improved application programming interface. This is especially helpful for the programmers.
- A core language based on the upcoming ECMAScript (ECMA-262) edition 4 draft language specification.
- An XML API based on the ECMAScript for XML (E4X) specification (ECMA-357 edition 2). E4X is a language extension to ECMAScript that adds XML as a native data type of the language.

This new version is designed to facilitate the creation of highly complex applications with large data sets and object-oriented, reusable code bases. Another great improvement is the new virtual machine AVM2 that can execute up to ten times faster.
than legacy ActionScript code. However, the older versions of the language can run in the new Flash Player 9.0.

ActionScript 3.0 still keeps many classes and features that will be familiar to ActionScript programmers, but it is architecturally and conceptually different from previous versions.

The new features in ActionScript 3.0 also include a renewed core language and an improved Flash Player API that provides increased and easier control of low-level objects. The core language defines the basic building blocks of the programming language, such as statements, expressions, conditions, loops, and types. This new version of ActionScript contains many new features that speed up the development process. (http://www.adobe.com/support/documentation/en/flash)

**Code in ActionScript 3.0**

We have implemented the main instrument in ActionScript 3.0 based on the last version in ActionScript 2.0. We based our new cockpit on the last project and we have used their graphics. As it was said in previous chapters the main differences between them is that we have done different modules in order to have a better organization of the code and files. It will be easier for futures reviews of the code and it will be easy to work in the different modules and improve one of the modules separately. We have created seven different classes for the different modules. The communication with the simulator is included in the main file and it is also completely new since we have used the new libraries. We have used the library pxFlashLib from PIXCODE to communicate with the simulator. This library includes two folders. One it is called “Utils” and it contains the file pxSmoothRotation.as. We use this file to create the rotation of the needle. The other folder it is called “Network”. It has the files: asimDP.as, Base64.as, inetDP.as, pxFlashWin.as, pxNetDataMess.as, pxNetEventMess.as, udpSockDP.as. We will not include this code since it is property of PIXCODE. We have not used any code, apart from “stops” in the frames so it is easier to understand and review the code. The project consists of eight files and the two folders that we have mentioned.

These are the different modules:

- Main.as
- Navigation.as
- Climate.as
- Src.as
- Media.as
- Mode.as
- Ccr.as
- Phone.as
Main.as

package {

////////////////////////////////////////////////////////////////////
// LIBRARIES
////////////////////////////////////////////////////////////////////
//We import all the libraries needed
import flash.system.Security;
import flash.system.SecurityPanel;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.external.ExternalInterface;
import flash.display.MovieClip;
import flash.text.TextField;
import flash.events.MouseEvent and Event;
import flash.utils.Base64String and ByteArray.
//This is the library written by Pixcode.
//It is needed for the communication with the simulator.
import pxFlashLib.Network.*;
//This library is used for the rotation of the different needles
import pxFlashLib.Utils.*;

public class main extends MovieClip {

////////////////////////////////////////////////////////////////////
// VARIABLES
////////////////////////////////////////////////////////////////////
//Here we declare all the variables used later on in our code
//we need to create a varibale of this type
var _flashWin:pxFlashWin;
//the speed
var round_speed:Number;
//previous gear
var prevgear:Number=new Number(1);
//rpm
var rotrev:Number=new Number();
//rpm
var mellanrev:Number=new Number();
//rpm
var evenrev:Number=new Number();
//variable for the different modes
var varmode:int=new int(0);
//var for the indicators
var var_indicator:int=new int(0);
//var for the left button
var inhibit_var:int=new int(0);
//rotation of the speed needle
var _speedNeedle:pxSmoothRotation;
//legal speed
var legal_speed: int;
//It tells if we are at the begining
var cuenta_inicio:int=new int(0);
//Counter
var conta:int=new int(0);
//variables to control the buttons
var exit_mode:int=new int(0);
//we control in mode we are any moment
var frame_mode:int = new int(0);
//for the navigation
var mapvisible:int = new int(0);
//The data we need to pass to the compass
var heading:Number = new Number();

////////////////////////////////////////////////////////////////////
// OBJECTS
////////////////////////////////////////////////////////////////////
//object for the climate mode
var _climate:climate=new climate(MovieClip(this));
//object for the src mode
var _src:src=new src(MovieClip(this));
//object for the media mode
var _media:media=new media(MovieClip(this));
//object for the phone mode
var _phone:phone=new phone(MovieClip(this));
//Object for the ccr mode
var _ccr:ccr=new ccr(MovieClip(this));
//Object for the mode modes
}
Appendix

var _modes:modes=new modes(MovieClip(this));
//Object for the navigation mode
var _navigation:navigation=new navigation(MovieClip(this));

///////////////////////////
// MAIN FUNCTION
///////////////////////////
public function main()
{
    try
    {
        // This code is due to security reasons
        Security.allowDomain("*");
        stage.showDefaultContextMenu = false;
        stage.scaleMode = StageScaleMode.EXACT_FIT;
        stage.align = StageAlign.TOP_LEFT;
        //here we stablish the communication with the simulator
        _flashWin = new pxFlashWin();
        _flashWin.addEventListener("netDataMess", netDataMessCallback);
        setupNetwork();
        this.addEventListener(Event.ENTER_FRAME, Update);
        //This variable is for the speed needle
        _speedNeedle = new pxSmoothRotation(this.needle, -60, 119, 180);
        if(cuenta_inicio==0)
        {
            //Initial Status for the different modes
            _climate.visibility_climate();
            _src.visibility_src();
            _media.visibility_media();
            _phone.visibility_phone();
            _ccr.visibility_ccr();
            _navigation.visibility_navigation();
            _modes.visibility_modes();
            //Some movie clips that have to be visible in the default mode
            default_buttons.visible=true;
            fuel.visible=true;
            numbers.visible=true;
            speedcolor.visible=true;
            nightnumbers.visible=false;
            cuenta_inicio=1;
        }
        catch (error:*)
        {
            pxFlashWin.setDebugText("main() catch() : " + error);
        }
    }
}

// FUNCTIONS: ALL FUNCTIONS MUST BE DECLARED AND DEFINED FROM HERE

private function Update(ev:Event)
{
    var tDate:Date = new Date();
    this.txt_Time.text = tDate.getHours().toString() + ":" + tDate.getMinutes().toString()+ "\" + tDate.getSeconds().toString();
    //If we are in the night mode, we change the color of the numbers
    if(varmode==1)
    {
        txt_Time.textColor=0x00ff7e;
        temptext.textColor=0x00ff7e;
    }
    else
    {
        txt_Time.textColor=0xffffff;
        temptext.textColor=0xffffff;
    }
}

//Function to control the clock

public function visible_digital_speed()
{
    //If we are in the night mode, it is different
    if(varmode==1)
    {
        //We control the visibility of the digital speed.
    }
}
needle.gotoAndStop(4);
digitalspeed.gotoAndStop(4);
}
else
{
    needle.gotoAndStop(2);
digitalspeed.gotoAndStop(2);
}
digitalspeed.visible=true;
}

//We control the visibility of the digital speed.
public function unvisible_digital_speed()
{
    if(varmode==1)
    {
        needle.gotoAndStop(3);
digitalspeed.gotoAndStop(3);
    }
    else
    {
        needle.gotoAndStop(1);
digitalspeed.gotoAndStop(1);
    }
    if(frame_mode==0)
    {
        digitalspeed.visible=false;
    }
}

//This functions shows or hides the digital speed, depending the speed
//limit that we have. It uses the last two functions above.
public function legal_speed_function(round_speed:int, legal_speed:int)
{
    switch (legal_speed)
    {
        case 1:
            if(round_speed>30)
            {
                visible_digital_speed();
            }
            else
            {
                unvisible_digital_speed();
            }
            break;
        case 2:
            if(round_speed>50)
            {
                visible_digital_speed();
            }
            else
            {
                unvisible_digital_speed();
            }
            break;
        case 3:
            if(round_speed>70)
            {
                visible_digital_speed();
            }
            else
            {
                unvisible_digital_speed();
            }
            break;
        case 4:
            if(round_speed>90)
            {
                visible_digital_speed();
            }
            else
            {
                unvisible_digital_speed();
            }
            break;
    }
}
break;
case 5:
if(round_speed>110)
{
    visible_digital_speed();
}
else
{
    unvisible_digital_speed();
}
break;
}

//Function for the blinkers.
public function blinkers_function(tName:String,tState:Number)
{
    switch (tName)
    {
    case "RIGHT":
        if (tState==1)
        {
            blinkright.gotoAndPlay(2);
            blinker_sound.gotoAndPlay(2);
        }
        if(tState==0)
        {
            blinkright.gotoAndStop(1);
            blinker_sound.gotoAndStop(1);
        }
        break;
    case "LEFT":
        if (tState==1)
        {
            blinkleft.gotoAndPlay(2);
            blinker_sound.gotoAndPlay(2);
        }
        if(tState==0)
        {
            blinkleft.gotoAndStop(1);
            blinker_sound.gotoAndStop(1);
        }
        break;
    }
}

//Function to change the color of the speed limit with the triggers
public function speedcolor_function (trigg: String)
{
    if(trigg== "speed_30_1" || trigg=="speed_30_2" || trigg=="speed_30_3")
    {
        if (varmode==1)
        {
            nightnumbers.gotoAndStop(1);
        }
        else
        {
            speedcolor.gotoAndStop(1);
        }
    }
    if(trigg== "speed_50_1" || trigg=="speed_50_2" || trigg=="speed_50_3")
    {
        if (varmode==1)
        {
            nightnumbers.gotoAndStop(1);
        }
        else
        {
            speedcolor.gotoAndStop(2);
        }
    }
    if(trigg== "speed_70_1" || trigg=="speed_70_2" || trigg=="speed_70_3")
    {
if (varmode==1)
{
    nightnumbers.gotoAndStop(2);
}
else
{
    speedcolor.gotoAndStop(3);
}

if(trigg == "speed_90_1" || trigg=="speed_90_2" || trigg=="speed_90_3")
{
    if (varmode==1)
    {
        nightnumbers.gotoAndStop(3);
    }
    else
    {
        speedcolor.gotoAndStop(4);
    }
}

if(trigg == "speed_110_1" || trigg=="speed_110_2" || trigg=="speed_110_3")
{
    if (varmode==1)
    {
        nightnumbers.gotoAndStop(3);
    }
    else
    {
        speedcolor.gotoAndStop(5);
    }
}

////////////////////////////////////////////////////////////////////
//Control the movie clip with the RPM
public function rpm_function (rev:Number)
{
    if (rev == 825) rpm.gotoAndStop(1);
    if (rev >= 1000) rpm.gotoAndStop(2);
    if (rev >= 1500) rpm.gotoAndStop(3);
    if (rev >= 2000) rpm.gotoAndStop(4);
    if (rev >= 2500) rpm.gotoAndStop(5);
    if (rev >= 3000) rpm.gotoAndStop(6);
    if (rev >= 3500) rpm.gotoAndStop(7);
    if (rev >= 4000) rpm.gotoAndStop(8);
    if (rev >= 4500) rpm.gotoAndStop(9);
    if (rev >= 5000) rpm.gotoAndStop(10);
    if (rev >= 5500) rpm.gotoAndStop(11);
    if (rev >= 6000) rpm.gotoAndStop(12);
    if (rev >= 6500) rpm.gotoAndStop(13);
    if (rev >= 7000) rpm.gotoAndStop(14);
}

////////////////////////////////////////////////////////////////////
//In this function we control the button wheel that consists of the button
//nxleft button),src(right button),VolUp(Up button),VolDwn(Down button),
//and CLR(central button).
public function selection_mode(button:String,numero:Number)
{
    switch (button)
    {
    case "NXT":
        //move to the right
        if ((numero==1)&&(frame_mode==0))
        {
            move_right();
        }
        //when we are in one of the functions
        //CLIMATE
        if ((numero==1)&&(frame_mode==1))
        {
            _climate.morefan();
        }
        //SRC
        if ((numero==1)&&(frame_mode==2))
        {
            _src.srcright();
        }
Appendix

//MEDIA
if ((numero==1)&&(frame_mode==3)&&(srctext.currentFrame==3))
{
    _media.channelright();
}

//CD
if ((numero==1)&&(frame_mode==3)&&(srctext.currentFrame==1))
{
    mode_buttons.gotoAndPlay(10);
}

//MP3
if ((numero==1)&&(frame_mode==3)&&(srctext.currentFrame==2))
{
    mode_buttons.gotoAndPlay(10);
}

//PHONE
if ((numero==1)&&(frame_mode==4))
{
    _phone.put_down();
}

//CCR
if ((numero==1)&&(frame_mode==5))
{
    _ccr.ccrright();
}

//NAVIGATION
if ((numero==1)&&(frame_mode==6)&&(mapavisible==1))
{
    _navigation.morezoom();
}

//MODES
if ((numero==1)&&(frame_mode==7))
{
    _modes.moderight();
}

break;

case "SRC":
//move to the left
if ((numero==1)&&(frame_mode==0))
{
    move_left();
}

//when we are in one of the functions
//CLIMATE
if ((numero==1)&&(frame_mode==1))
{
    _climate.lessfan();
}

//SRC
if ((numero==1)&&(frame_mode==2))
{
    _src.srcleft();
}

//MEDIA
if ((numero==1)&&(frame_mode==3)&&(srctext.currentFrame==3))
{
    _media.channellleft();
}

//CD
if ((numero==1)&&(frame_mode==3)&&(srctext.currentFrame==1))
{
    mode_buttons.gotoAndPlay(2);
}

//MP3
if ((numero==1)&&(frame_mode==3)&&(srctext.currentFrame==2))
{
    mode_buttons.gotoAndPlay(2);
}

//PHONE
if ((numero==1)&&(frame_mode==4))
{
    _phone.calling();
}

//CCR
if ((numero==1)&&(frame_mode==5))
```c
{ _ccr.ccrleft();
} //NAVIGATION
if (((numero==1)&&(frame_mode==6)&&(mapavisible==1))
{ _navigation.lesszoom();
} //MODES
if (((numero==1)&&(frame_mode==7))
{ _modes.modedleft();
} break;
```

```c
case "VolUp":
if (numero==1)
{
    switch (frame_mode)
    {
    //CLIMATE
    case 1: _climate.moretemperature();
    break;
    //SRC
    case 2: _src.morevolume();
    break;
    //MEDIA
    case 3: _media.moretvolume();
    break;
    //MEDIA
    case 4: _phone.morelist();
    break;
    //CCR
    case 5: _ccr.ccrup();
    break;
    //NAVIGATION
    case 6: _navigation.mapon();
    mapavisible=1;
    break;
    //MODES
    case 7: _modes.mode_on();
    //night
    if (mode_mode.modes.currentFrame==2)
    {
        varmode = 1;
    }
    //economy
    else if (mode_mode.modes.currentFrame==1)
    {
        varmode = 2;
    }
    //sport
    else if (mode_mode.modes.currentFrame==4)
    {
        varmode = 3;
    }
    else
    //default
    {
        varmode = 0;
    }
    break;
    }
}
break;
case "VolDwn":
if (numero==1)
{
    switch (frame_mode)
    {
    //CLIMATE
    case 1: _climate.morettemperature();
    break;
    //SRC
    case 2: _src.morevolume();
    break;
    //MEDIA
    case 3: _media.moretvolume();
    break;
    //MEDIA
    case 4: _phone.morelist();
    break;
    //CCR
    case 5: _ccr.ccrup();
    break;
    //NAVIGATION
    case 6: _navigation.mapon();
    mapavisible=1;
    break;
    //MODES
    case 7: _modes.mode_on();
    //night
    if (mode_mode.modes.currentFrame==2)
    {
        varmode = 1;
    }
    //economy
    else if (mode_mode.modes.currentFrame==1)
    {
        varmode = 2;
    }
    //sport
    else if (mode_mode.modes.currentFrame==4)
    {
        varmode = 3;
    }
    else
    //default
    {
        varmode = 0;
    }
    break;
    }
```
case 1:  
  _climate.lesstemperature();  
  break;
//SOURCE
case 2:  
  _src.lessvolume();  
  break;
//MEDIA
case 3:  
  _media.lessvolume();  
  break;
//PHONE
case 4:  
  _phone.lesslist();  
  break;
//CCR
case 5:  
  _ccr.ccrdown();  
  break;
//NAVIGATION
case 6:  
  if (mapavisible==1)  
    {  
      _navigation.mapoff();  
      mapavisible=0;  
    }  
  break;
}
break;

case "CLR":
  mode_buttons.gotoAndPlay(18);
  if ((modesymbols.currentFrame==3)&&(numero==1))
  {
    if(exit_mode==0)
    {
      //if we are in the sport mode or economy
      if (varmode==2)
      {
        economy.visible=false;
      }
      if (varmode==3)
      {
        turbo_temp.visible=false;
      }
      frame_mode=1;
      exit_mode=1;
      _climate.init_climate();
    }
    else
    {
      //if we are in the sport mode or economy
      if (varmode==2)
      {
        economy.visible=true;
      }
      if (varmode==3)
      {
        turbo_temp.visible=true;
      }
      _climate.exit_climate();
      exit_mode=0;
      frame_mode=0;
    }
  }
//mode src frame_mode=2
  if ((modesymbols.currentFrame==2)&&(numero==1))
  {
    if(exit_mode==0)
    {
      //if we are in the sport mode or economy
      if (varmode==2)
      {  
        economy.visible=true;
      }
      if (varmode==3)
      {  
        turbo_temp.visible=true;
      }
      _climate.exit_climate();
      exit_mode=0;
      frame_mode=0;
    }
  }

//CLIMATE
if (varmode==2)
{
    economy.visible=false;
}
if (varmode==3)
{
    turbo_temp.visible=false;
}
frame_mode=2;
exit_mode=1;
_src.init_src();
}
else
{
    //if we are in the sport mode or economy
    if (varmode==2)
    {
        economy.visible=true;
    }
    if (varmode==3)
    {
        turbo_temp.visible=true;
    }
    _src.exit_src();
    exit_mode=0;
    frame_mode=0;
}
}
//mode media frame_mode 3
if ((modesymbols.currentFrame==1)&&(numero==1))
{
    if(exit_mode==0)
    {
        //if we are in the sport mode or economy
        if (varmode==2)
        {
            economy.visible=false;
        }
        if (varmode==3)
        {
            turbo_temp.visible=false;
        }
        frame_mode=3;
        exit_mode=1;
        _media.init_media();
    }
    else
    {
        //if we are in the sport mode or economy
        if (varmode==2)
        {
            economy.visible=true;
        }
        if (varmode==3)
        {
            turbo_temp.visible=true;
        }
        _media.exit_media();
        exit_mode=0;
        frame_mode=0;
    }
}
//phone media frame_mode 4
if ((modesymbols.currentFrame==4)&&(numero==1))
{
    if(exit_mode==0)
    {
        //if we are in the sport mode or economy
        if (varmode==2)
        {
            economy.visible=false;
        }
        if (varmode==3)
        {
            turbo_temp.visible=false;
        }
    }
frame_mode=4;
exit_mode=1;
_phone.init_phone();
}
else
{
  //if we are in the sport mode or economy
  if (varmode==2)
  {
    economy.visible=true;
  }
  if (varmode==3)
  {
    turbo_temp.visible=true;
  }
  _phone.exit_phone();
  exit_mode=0;
  frame_mode=0;
}
//CCR frame_mode 5
if ((modesymbols.currentFrame==7)&&(numero==1))
{
  if(exit_mode==0)
  {
    //if we are in the sport mode or economy
    if (varmode==2)
    {
      economy.visible=false;
    }
    if (varmode==3)
    {
      turbo_temp.visible=false;
    }
    frame_mode=5;
    exit_mode=1;
    _ccr.init_ccr();
  }
  else
  {
    //if we are in the sport mode or economy
    if (varmode==2)
    {
      economy.visible=true;
    }
    if (varmode==3)
    {
      turbo_temp.visible=true;
    }
    _ccr.exit_ccr();
    exit_mode=0;
    frame_mode=0;
  }
}
//navigation frame_mode 6
if ((modesymbols.currentFrame==6)&&(numero==1))
{
  if(exit_mode==0)
  {
    //if we are in the sport mode or economy
    if (varmode==2)
    {
      economy.visible=false;
    }
    if (varmode==3)
    {
      turbo_temp.visible=false;
    }
    frame_mode=6;
    exit_mode=1;
    _navigation.init_navigation();
  }
  else
  {
    //if we are in the sport mode or economy
    if (varmode==2)
{  
edconomy.visible=true;
}
if (varmode==3)
{
  
turbo_temp.visible=true;
}
_navigation.exit_navigation();
exit_mode=0;
frame_mode=0;
}

//modes frame_mode 7
if ((modesymbols.currentFrame==5)&&(numero==1))
{
  if(exit_mode==0)
  {
    /if we are in the sport mode or economy
    if (varmode==2)
    {
      economy.visible=false;
    }
    if (varmode==3)
    {
      turbo_temp.visible=false;
    }
    frame_mode=7;
    exit_mode=1;
    _modes.init_modes();
  }
  else
  {
    /if we are in the sport mode or economy
    if (varmode==2)
    {
      economy.visible=true;
    }
    if (varmode==3)
    {
      turbo_temp.visible=true;
    }
    _modes.exit_modes();
    exit_mode=0;
    frame_mode=0;
  }
}
break;
case "Dwn":
if (((numero==1)&&(inhibit_var==0))
{
  switch (var_indicator)
  {
    case 0:
      indicator.gotoAndPlay(2);
      var_indicator=1;
      break;
    case 1:
      indicator.gotoAndPlay(73);
      var_indicator=2;
      break;
    case 2:
      indicator.gotoAndPlay(160);
      var_indicator=0;
      inhibit_var=1;
      break;
  }
}
break;
case "Up":
if (((numero==1)&&(inhibit_var==1))
{
  indicator.gotoAndPlay(1);
  inhibit_var=0;
}
break;
private function move_right()
{
    if (modesymbols.currentFrame == 7)
    {
        modesymbols.gotoAndPlay(1);
        green_cover.gotoAndPlay(2);
        default_buttons.gotoAndPlay(7);
    }
    else
    {
        modesymbols.gotoAndPlay(modesymbols.currentFrame +1);
        green_cover.gotoAndPlay(2);
        default_buttons.gotoAndPlay(default_buttons.currentFrame+1);
    }
}

private function move_left()
{
    if (modesymbols.currentFrame == 1)
    {
        modesymbols.gotoAndPlay(7);
        green_cover.gotoAndPlay(2);
        default_buttons.gotoAndPlay(7);
    }
    else
    {
        modesymbols.gotoAndPlay(modesymbols.currentFrame - 1);
        green_cover.gotoAndPlay(2);
        default_buttons.gotoAndPlay(default_buttons.currentFrame-1);
    }
}

private function setupNetwork()
{
    if(_flashWin.dataProviderExist("asim") )
    {
        asimDP.init( "NAME" );    // Name of Application
        asimDP.registerEventGroup( 1096 );  // ASim SPATIAL DATA.
        asimDP.registerEventGroup( 1099 );  // ASim MutiFunction Event 4.
        asimDP.registerEventGroup( 1107 );  // ASim TriggerBox Event.
        // These Codes are specified in the ACE application DPC.
        // Date Package Constructor i think its called.
        // Basiclly you have to know what you want and where to get it.
        //All Message events are Listed in that application.
    }
    else
    //In case we don’t have the communication with the simulator
    pxFlashWin.setDebugText( "!Flash - asimDP not existing" );
}

private function netDataMessCallback( ev:pxNetDataMess )
{
    if(ev._dataProvider == "asim")
    {
        var tGroup:int = ev._orgObject.GROUP;
        var tEvent:int = ev._orgObject.EVENT;
        // All Messages are dispatched here so we have to figure out which one it is.
        // ASim SPATIAL DATA. (DPC Shows wrong Header info. Header is 3 double not one)
        if(tGroup==1096 & & tEvent==)
        {
            var sender:String  = pxFlashWin.readNullString(ev._byteArray);
            var type:Number  = ev._byteArray.readByte();
        }
    }
Appendix

```javascript
var px:Number = ev._byteArray.readDouble();
var py:Number = ev._byteArray.readDouble();
var pz:Number = ev._byteArray.readDouble();
var hx:Number = ev._byteArray.readDouble(); // < - Header is 3 Double
var hy:Number = ev._byteArray.readDouble(); // < - Header_2
var hz:Number = ev._byteArray.readDouble(); // < - Header_3
var pitch:Number = ev._byteArray.readDouble();
var roll:Number = ev._byteArray.readDouble();
var speed:Number = ev._byteArray.readDouble();
// We create the object for the Needle rotation
_speedNeedle.setvalue(speed);
// Compass rotation
heading = ((hx*180)/3.14)+180;
compass.compassx.rotation=heading;
// This is for the digital speed
var speed_digit :Number = new Number(Math.round(speed));
digitalspeed.hasx.text = speed_digit;
// We need to round the speed so we can work with it
round_speed= new Number(Math.round(speed));
// Needle for the turbo
if (speed==0)
{
    turbo_temp.nalanim.gotoAndStop(10);
}
// For the speed limit
legal_speed = (speedcolor.currentFrame);
legal_speed_function(int(round_speed), legal_speed);

// Asim MultiFunction Controll
if(tGroup==1099 && & tEvent==4)
{
    var id:Number = ev._byteArray.readByte();
    var desc:String = pxFlashWin.readNullString(ev._byteArray);
    var tName:String = pxFlashWin.readNullString(ev._byteArray); // Device Name
    var tState:Number = ev._byteArray.readInt(); // State 0 = ButtonUp, 1 = ButtonDown
    Trace("DeviceName: " + tName + ", State: " + tState);
    selection_mode(tName,tState);
    blinkers_function(tName,tState);
}

// Asim TriggerBox
if(tGroup==1107 & & tEvent==14)
{
    var tTrigger:String = pxFlashWin.readNullString(ev._byteArray);
    trace ("TriggerName: " + tTrigger);
    speedcolor_function(tTrigger);
}

// ASim RPM
if(tGroup==1096 & & tEvent==3)
{
    var dat1:String = pxFlashWin.readNullString(ev._byteArray);
    var dat2:Number = ev._byteArray.readByte();
    var rev:Number = ev._byteArray.readDouble();
    var dat3:Number = ev._byteArray.readDouble();
    var gear:Number = ev._byteArray.readByte();
    rpm_function(rev);

    // this is for the turbo in the sport mode
    if (gear != prevgear)
    {
        turbo_temp.nalanim.gotoAndPlay(1);
    }
    prevgear=gear;
    // this is for the econal in the economy mode
    // ECONOMY
    evenrev=Math.round(rev);
    mellanrev=Math.round(evenrev/10)
    rotrev=mellanrev-95;
    if (rotrev<=-17 && rotrev<=195) economy.econal.rotation=rotrev;
    if (rev==825) varvtal_eco.gotoAndStop(1);
```

if (rev>=1000) varvtal_eco.gotoAndStop(2);
if (rev>=1500) varvtal_eco.gotoAndStop(3);
if (rev>=2000) varvtal_eco.gotoAndStop(4);
if (rev>=2500) varvtal_eco.gotoAndStop(5);
if (rev>=3000) varvtal_eco.gotoAndStop(6);
if (rev>=3500) varvtal_eco.gotoAndStop(7);
if (rev>=4000) varvtal_eco.gotoAndStop(8);
if (rev>=4500) varvtal_eco.gotoAndStop(9);
if (rev>=5000) varvtal_eco.gotoAndStop(10);
if (rev>=5500) varvtal_eco.gotoAndStop(11);
if (rev>=6000) varvtal_eco.gotoAndStop(12);
if (rev>=6500) varvtal_eco.gotoAndStop(13);
if (rev>=7000) varvtal_eco.gotoAndStop(14);
if (rev>=2000) economy.ecolampa.gotoAndStop(2);
if (rev<=2000) economy.ecolampa.gotoAndStop(1);

Climate.as

package {

    import flash.display.MovieClip;
    import flash.utils.getDefinitionByName;
    import flash.events.Event;
    import flash.events.TimerEvent;
    import flash.utils.Timer;
    import flash.events.Event;

    public class climate extends MovieClip {
        private var _stage:MovieClip;
        public function climate(Stage:MovieClip) {
            _stage = tStage;
        }

        //////////////////////////////////////////////////////////////////////////
        //FUNCTIONS
        //////////////////////////////////////////////////////////////////////////
        //We make non-visible the movie clips refered to this mode
        public function visibility_climate() {
            _stage.digitalspeed.visible=false;
            _stage.grades_temp.fan.gotoAndPlay(5);
            _stage.mode_buttons.visible=false;
            _stage.grades_temp.visible=false;
            _stage.default_buttons.visible=false;
        }

        //When we enter the module we make the movie clips visible
        public function init_climate() {
            _stage.digitalspeed.visible=true;
            _stage.module_screen.gotoAndPlay(2);
            _stage.mode_buttons.visible=true;
            _stage.grades_temp.visible=true;
            _stage.default_buttons.visible=true;
        }

        //When we exit the mode we hide the movie clips of the module
        public function exit_climate() {
            _stage.digitalspeed.visible=false;
            _stage.module_screen.gotoAndPlay(9);
            _stage.mode_buttons.visible=false;
            _stage.grades_temp.visible=false;

            if (_stage.mode_mode.modes.currentFrame==2) {
                _stage.default_buttons.visible=false;
            }
        }

        //More power to the fun
        public function morefan() {
            if (_stage.grades_temp.fan.currentFrame==9) {
                _stage.grades_temp.fan.gotoAndPlay(8);
            }
            _stage.grades_temp.fan.gotoAndPlay(_stage.grades_temp.fan.currentFrame + 1);
            _stage.mode_buttons.gotoAndPlay(10);
        }
    }
}
//Less power to the fun
public function lessfan() {
    if (_stage.grades_temp.fan.currentFrame==1) {
        _stage.grades_temp.fan.gotoAndPlay(2);
    }
    _stage.grades_temp.fan.gotoAndPlay(_stage.grades_temp.fan.currentFrame - 1);
    _stage.mode_buttons.gotoAndPlay(2);
}

//More temperature
public function moretemperature() {
    if (_stage.grades_temp.num.currentFrame==9) {
        _stage.grades_temp.num.gotoAndPlay(8);
        _stage.grades_temp.climate_gradient.y = 80;
    }
    _stage.grades_temp.climate_gradient.y = _stage.grades_temp.climate_gradient.y + 20;
    _stage.grades_temp.num.gotoAndPlay(_stage.grades_temp.num.currentFrame + 1);
    _stage.mode_buttons.gotoAndPlay(6);
}

//Less temperature
public function lesstemperature() {
    if (_stage.grades_temp.num.currentFrame==1) {
        _stage.grades_temp.num.gotoAndPlay(2);
        _stage.grades_temp.climate_gradient.y = -60;
    }
    _stage.grades_temp.climate_gradient.y = _stage.grades_temp.climate_gradient.y - 20;
    _stage.grades_temp.num.gotoAndPlay(_stage.grades_temp.num.currentFrame - 1);
    _stage.mode_buttons.gotoAndPlay(14);
}

Media.as

package {
    import flash.display.MovieClip;
    import flash.utils.getDefinitionByName;
    import flash.events.Event;
    import flash.events.TimerEvent;
    import flash.utils.Timer;
    import flash.events.Event;

    public class media extends MovieClip {
        private var _stage:MovieClip;

        public function media(tStage:MovieClip) {
            _stage = tStage;
        }

        //We make non-visible the movie clips refered to this mode
        public function visibility_media() {
            _stage.digitalspeed.visible=false;
            _stage.volumen.visible=false;
            _stage.volumen.gotoAndPlay(9);
            _stage.mode_buttons.visible=false;
            _stage.src_mode.visible=false;
            _stage.song.visible=false;
            _stage.radio.visible=false;
            _stage.default_buttons.visible=false;
        }

        //When we enter the module we make the movie clips visible
        public function init_media() {
            _stage.digitalspeed.visible=true;
            _stage.module_screen.gotoAndPlay(2);
            _stage.volumen.visible=true;
            _stage.mode_buttons.visible=true;
            _stage.src_mode.visible=true;
            _stage.default_buttons.visible=true;
        }
    }
}
if (_stage.src_text.currentFrame == 3) {
    _stage.radio.visible = true;
} else {
    _stage.song.visible = true;
}

// When we exit the mode we hide the movie clips of the module
public function exit_media()
{
    _stage.digitalspeed.visible = false;
    _stage.module_screen.gotoAndPlay(9);
    _stage.volumen.visible = false;
    _stage.mode_buttons.visible = false;
    _stage.src_mode.visible = false;
    _stage.song.visible = false;
    _stage.radio.visible = false;
    if (_stage.mode_mode.modes.currentFrame == 2) {
        _stage.default_buttons.visible = false;
    }
}

// More volume
public function more_volume()
{
    if (_stage.volumen.currentFrame == 17) {
        _stage.volumen.gotoAndPlay(16);
    }
    _stage.volumen.gotoAndPlay(_stage.volumen.currentFrame + 1);
    _stage.mode_buttons.gotoAndPlay(6);
}

// Less volume
public function less_volume()
{
    if (_stage.volumen.currentFrame == 1) {
        _stage.volumen.gotoAndPlay(2);
    }
    _stage.volumen.gotoAndPlay(_stage.volumen.currentFrame - 1);
    _stage.mode_buttons.gotoAndPlay(14);
}

// We change the radio station (to the right)
public function channel_right()
{
    if (_stage.radio.currentFrame == 4) {
        _stage.radio.gotoAndPlay(1);
    } else if (_stage.radio.currentFrame == 3) {
        _stage.radio.gotoAndPlay(4);
    } else if (_stage.radio.currentFrame == 2) {
        _stage.radio.gotoAndPlay(3);
    } else if (_stage.radio.currentFrame == 1) {
        _stage.radio.gotoAndPlay(2);
    }
    _stage.mode_buttons.gotoAndPlay(10);
}

// We change the radio station (to the left)
public function channel_left()
{
    if (_stage.radio.currentFrame == 1) {
        _stage.radio.gotoAndPlay(4);
    } else if (_stage.radio.currentFrame == 4) {
        _stage.radio.gotoAndPlay(3);
    } else if (_stage.radio.currentFrame == 3) {
        _stage.radio.gotoAndPlay(2);
    } else if (_stage.radio.currentFrame == 2) {
        _stage.radio.gotoAndPlay(1);
    }
    _stage.mode_buttons.gotoAndPlay(2);
}

Src.as

package {

    importflash.display.MovieClip;
}

// we import the libraries
import flash.display.MovieClip;
import flash.mxml.importDefinitionByName;
import flash.events.Event;
import flash.events.TimerEvent;
import flash.utils.Timer;
import flash.events.Event;

public class src extends MovieClip {
    //we can control the stage with _stage
    private var _stage:MovieClip;

    public function src(tStage:MovieClip) {
        _stage = tStage;
    }

    //FUNCTIONS
    //We make non-visible the movie clips refereed to this mode
    public function visibility_src() {
        _stage.digitalspeed.visible=false;
        _stage.volumen.visible=false;
        _stage.volumen.gotoAndPlay(9);
        _stage.mode_buttons.visible=false;
        _stage.src_mode.visible=false;
        _stage.srctext.visible=false;
        _stage.default_buttons.visible=false;
    }

    //When we enter the module we make the movie clips visible
    public function init_src() {
        _stage.digitalspeed.visible=true;
        _stage.module_screen.gotoAndPlay(2);
        _stage.volumen.visible=true;
        _stage.mode_buttons.visible=true;
        _stage.src_mode.visible=true;
        _stage.srctext.visible=true;
        _stage.default_buttons.visible=true;
    }

    //When we exit the mode we hide the movie clips of the module
    public function exit_src() {
        _stage.digitalspeed.visible=false;
        _stage.module_screen.gotoAndPlay(9);
        _stage.volumen.visible=false;
        _stage.mode_buttons.visible=false;
        _stage.src_mode.visible=false;
        _stage.srctext.visible=false;
        if (_stage.mode_mode.modes.currentFrame==2) {
            _stage.default_buttons.visible=false;
        }
    }

    //More volume
    public function morevolume() {
        if (_stage.volumen.currentFrame==17) {
            _stage.volumen.gotoAndPlay(16);
        }
        _stage.volumen.gotoAndPlay(_stage.volumen.currentFrame + 1);
        _stage.mode_buttons.gotoAndPlay(6);
    }

    //Less volume
    public function lessvolume() {
        if (_stage.volumen.currentFrame==1) {
            _stage.volumen.gotoAndPlay(2);
        }
        _stage.volumen.gotoAndPlay(_stage.volumen.currentFrame - 1);
        _stage.mode_buttons.gotoAndPlay(14);
    }

    //We move the source to the right, radio, cd, mp3
    public function srcright() {
        if (_stage.srctext.currentFrame==3) {
            _stage.srctext.gotoAndPlay(2);
        } else if (_stage.srctext.currentFrame==2) {
            _stage.srctext.gotoAndPlay(1);
        } else if (_stage.srctext.currentFrame==1) {
            _stage.srctext.gotoAndPlay(3);
        }
        _stage.mode_buttons.gotoAndPlay(10);
    }
}
public function srcleft() {
    if (_stage.srctext.currentFrame==3) {
        _stage.srctext.gotoAndPlay(1);
    } else if (_stage.srctext.currentFrame==2) {
        _stage.srctext.gotoAndPlay(3);
    } else if (_stage.srctext.currentFrame==1) {
        _stage.srctext.gotoAndPlay(2);
    }
    _stage.mode_buttons.gotoAndPlay(2);
}

Phone.as

package {

    //We import the libraries
    import flash.display.MovieClip;
    import flash.utils.getDefinitionByName;
    import flash.events.Event;
    import flash.events.TimerEvent;
    import flash.utils.Timer;
    import flash.events.Event;

    public class phone extends MovieClip {
        private var _stage:MovieClip;
        //Variables
        private var cont:int=new int(1);
        private var cont2:int=new int(1);
        public function phone(tStage:MovieClip) {
            _stage = tStage;
        }

        //FUNCTIONS
        //We make non-visible the movie clips refered to this mode
        public function visibility_phone() {
            _stage.digitalspeed.visible=false;
            _stage.mode_buttons.visible=false;
            _stage.telephone.visible=false;
            _stage.default_buttons.visible=false;
        }
        //When we enter the module we make the movie clips visible
        public function init_phone() {
            _stage.digitalspeed.visible=true;
            _stage.mode_buttons.visible=true;
            _stage.module_screen.gotoAndPlay(2);
            _stage.telephone.visible=true;
            _stage.telephone.calling.visible=false;
            _stage.default_buttons.visible=true;
        }
        //When we exit the mode we hide the movie clips of the module
        public function exit_phone() {
            _stage.digitalspeed.visible=false;
            _stage.module_screen.gotoAndPlay(9);
            _stage.mode_buttons.visible=false;
            _stage.telephone.visible=false;
            if (_stage.mode_mode.modes.currentFrame==2) {
                _stage.default_buttons.visible=false;
            }
        }
        //Move the phone list up
        public function morelist() {
            _stage.telephone.gotoAndPlay(_stage.telephone.currentFrame +1);
            _stage.mode_buttons.gotoAndPlay(6);
        }
        //Move the phone list down
        public function lesslist() {
            if (cont==1) {
                cont=6;
            }
        }
    }
}
//Hang up the phone call
public function put_down() {
    _stage.telephone.calling.visible=false;
    _stage.mode_buttons.gotoAndPlay(10);
}

//Call the number chosen in the phone list
public function calling() {
    _stage.telephone.calling.visible=true;
    switch (_stage.telephone.currentFrame) {
        case 1 :
            _stage.telephone.calling.gotoAndStop(1);
            break;
        case 2 :
            _stage.telephone.calling.gotoAndStop(2);
            break;
        case 3 :
            _stage.telephone.calling.gotoAndStop(3);
            break;
        case 4 :
            _stage.telephone.calling.gotoAndStop(4);
            break;
        case 5 :
            _stage.telephone.calling.gotoAndStop(5);
            break;
    }
    _stage.mode_buttons.gotoAndPlay(2);
}

Navigation.as

package {
import flash.display.MovieClip;
import flash.utils.getDefinitionByName;
import flash.events.Event;
import flash.events.TimerEvent;
import flash.utils.Timer;
import flash.events.Event;

public class navigation extends MovieClip {
    private var _stage:MovieClip;

    public function navigation(tStage:MovieClip) {
        _stage = tStage;
    }

    //FUNCTIONS
    //We make non-visible the movie clips referred to this mode
    public function visibility_navigation() {
        _stage.digitalspeed.visible=false;
        _stage.mode_buttons.visible=false;
        _stage.navii.visible=false;
        _stage.compass.visible=false;
        _stage.default_buttons.visible=false;
    }

    //When we enter the module we make the movie clips visible
    public function init_navigation() {
        _stage.digitalspeed.visible=true;
        _stage.module_screen.gotoAndPlay(2);
    }
}
__stage.mode_buttons.visible=true;
__stage.navi.visible=true;
__stage.compass.visible=true;
__stage.default_buttons.visible=true;

} //When we exit the mode we hide the movie clips of the module
public function exit_navigation() {
  __stage.digitalspeed.visible=false;
  __stage.module_screen.gotoAndPlay(9);
  __stage.mode_buttons.visible=false;
  __stage.navi.visible=false;
  __stage.compass.visible=false;
  if (__stage.mode_mode.modes.currentFrame==2) {
    __stage.default_buttons.visible=false;
  }
}

} //We display the map
public function mapon() {
  __stage.compass.visible=false;
  __stage.navi.gotoAndPlay(2);
  __stage.mode_buttons.gotoAndPlay(6);
}

} //We hide the map
public function mapoff() {
  __stage.compass.visible=true;
  __stage.navi.gotoAndPlay(1);
  __stage.mode_buttons.gotoAndPlay(14);
}

} //We get closer to the map
public function morezoom() {
  if (__stage.navi.map_zoom.currentFrame==2) {
    __stage.navi.map_zoom.gotoAndPlay(3);
  } else if (__stage.navi.map_zoom.currentFrame==1) {
    __stage.navi.map_zoom.gotoAndPlay(2);
  }
  __stage.mode_buttons.gotoAndPlay(10);
}

} //We make a zoom out in the map
public function lesszoom() {
  if (__stage.navi.map_zoom.currentFrame==2) {
    __stage.navi.map_zoom.gotoAndPlay(1);
  } else if (__stage.navi.map_zoom.currentFrame==3) {
    __stage.navi.map_zoom.gotoAndPlay(2);
  }
  __stage.mode_buttons.gotoAndPlay(2);
}

}

Modes.as
package {
import flash.display.MovieClip;
import flash.utils.getDefinitionByName;
import flash.events.Event;
import flash.events.TimerEvent;
import flash.utils.Timer;
import flash.events.Event;

public class modes extends MovieClip {
  private var _stage:MovieClip;

  public function modes(tStage:MovieClip) {
    _stage = tStage;

    } //FUNCTIONS

public function visibility_modes() {

_stage.digitalspeed.visible=false;
_stage.mode_buttons.visible=false;
_stage.mode_mode.visible=false;
_stage.turbo_temp.visible=false;
_stage.economy.visible=false;
_stage.varvtal_eco.visible=false;
_stage.default_buttons.visible=false;
}

//When we enter the module we make the movie clips visible
public function init_modes() {
    _stage.digitalspeed.visible=true;
    _stage.module_screen.gotoAndPlay(2);
    _stage.mode_mode.visible=true;
    _stage.mode_buttons.visible=true;
    _stage.default_buttons.visible=true;
}

//When we exit the mode we hide the movie clips of the module
public function exit_modes() {
    _stage.module_screen.gotoAndPlay(9);
    _stage.mode_buttons.visible=false;
    _stage.mode_mode.visible=false;
    if (_stage.mode_mode.modes.currentFrame==2) {
        _stage.default_buttons.visible=false;
    }
}

//We chose one of the modes(economy, night, default, sport)
public function mode_on() {
    _stage.mode_buttons.gotoAndPlay(6);
    switch (_stage.mode_mode.modes.currentFrame) {
    case 1://economy
        _stage.turbo_temp.visible=false;
        _stage.economy.visible=true;
        _stage.rpm.visible=false;
        _stage.varvtal_eco.visible=true;
        _stage.fuel.visible=true;
        _stage.numbers.visible=true;
        _stage.speedcolor.visible=true;
        _stage.nightnumbers.visible=false;
        _stage.mode_buttons.visible=true;
        _stage.default_buttons.visible=true;
        break;
    case 2://night
        _stage.turbo_temp.visible=false;
        _stage.economy.visible=false;
        _stage.varvtal_eco.visible=false;
        _stage.rpm.visible=false;
        _stage.mode_buttons.visible=false;
        _stage.fuel.visible=false;
        _stage.numbers.visible=false;
        _stage.speedcolor.visible=false;
        _stage.nightnumbers.visible=true;
        _stage.default_buttons.visible=false;
        break;
    case 3://default
        _stage.turbo_temp.visible=false;
        _stage.economy.visible=false;
        _stage.varvtal_eco.visible=false;
        _stage.rpm.visible=true;
        _stage.digitalspeed.visible=false;
        _stage.fuel.visible=true;
        _stage.numbers.visible=true;
        _stage.speedcolor.visible=true;
        _stage.nightnumbers.visible=false;
        _stage.mode_buttons.visible=true;
        _stage.default_buttons.visible=true;
        break;
    case 4://sport
        _stage.turbo_temp.visible=true;
        _stage.economy.visible=false;
        _stage.varvtal_eco.visible=false;

Ccr.as

package {
import flash.display.MovieClip;
import flash.utils.getDefinitionByName;
import flash.events.Event;
import flash.events.TimerEvent;
import flash.events.Timer;
import flash.events.Event;

public class ccr extends MovieClip {
  //Variables
  private var _stage:MovieClip;
  private var onoff:int = new int(0);

  public function ccr(tStage:MovieClip) {
    _stage = tStage;
  }

  //FUNCTIONS

  //We make non-visible the movie clips refered to this mode
  public function visibility_ccr() {
    _stage.rpm.visible=true;
    _stage.fuel.visible=false;
    _stage.numbers.visible=true;
    _stage.speedcolor.visible=true;
    _stage.nightnumbers.visible=false;
    _stage.mode_buttons.visible=true;
    _stage.default_buttons.visible=true;
    break;
  }

  //We turn off the mode
  public function mode_off() {
    _stage.mode_buttons.gotoAndPlay(14);
  }

  //We move to the next mode(to the right)
  public function moderight() {
    if (_stage.mode_mode.modes.currentFrame==3) {
      _stage.mode_mode.modes.gotoAndPlay(4);
    } else if (_stage.mode_mode.modes.currentFrame==2) {
      _stage.mode_mode.modes.gotoAndPlay(3);
    } else if (_stage.mode_mode.modes.currentFrame==1) {
      _stage.mode_mode.modes.gotoAndPlay(2);
    }
    _stage.mode_buttons.gotoAndPlay(10);
  }

  //We move to the next mode(to the left)
  public function modeleft() {
    if (_stage.mode_mode.modes.currentFrame==4) {
      _stage.mode_mode.modes.gotoAndPlay(3);
    } else if (_stage.mode_mode.modes.currentFrame==3) {
      _stage.mode_mode.modes.gotoAndPlay(2);
    } else if (_stage.mode_mode.modes.currentFrame==2) {
      _stage.mode_mode.modes.gotoAndPlay(1);
    }
    _stage.mode_buttons.gotoAndPlay(2);
  }
}
}
/**
 * When we exit the mode we hide the movie clips of the module
 * public function exit_ccr() {
 *     _stage.digitalspeed.visible=false;
 *     _stage.module_screen.gotoAndPlay(9);
 *     _stage.mode_buttons.visible=false;
 *     _stage.ccr_mode.visible=false;
 *     if (_stage.mode_mode.modes.currentFrame==2) {
 *         _stage.default_buttons.visible=false;
 *     }
 * }
 *
 * // We don’t need this button
 * public function ccrup() {
 *     _stage.mode_buttons.gotoAndPlay(6);
 * }
 *
 * // We don’t need this button
 * public function ccrdown() {
 *     _stage.mode_buttons.gotoAndPlay(14);
 * }
 *
 * // We don’t need this button
 * public function ccrright() {
 *     _stage.mode_buttons.gotoAndPlay(10);
 * }
 *
 * // We activate the ccr mode or we turn it off
 * public function ccrlft() {
 *     if (onoff==0) {
 *         _stage.ccr_symbol.visible=true;
 *         _stage.ccr_mode.gotoAndPlay(2);
 *         onoff=1;
 *     } else {
 *         _stage.ccr_symbol.visible=false;
 *         _stage.ccr_mode.gotoAndPlay(1);
 *         onoff=0;
 *     }
 *     _stage.mode_buttons.gotoAndPlay(2);
 * }
 */
Appendix B

#include <osgDB/ReadFile>
#include <osgDB/WriteFile>
#include <osgViewer/Viewer>
#include <osgViewer/ViewerEventHandlers>
#include <osgGA/TrackballManipulator>
#include <osgGA/FlightManipulator>
#include <osgGA/AnimationPathManipulator>
#include <iostream>

class ModelHandler : public osgGA::GUIEventAdapter
{
public:
    ModelHandler():
        _position(0) {}

    typedef std::vector<std::string> Filenames;
    Filenames _filenames;
    unsigned int _position;

    void add(const std::string& filename)
    {
        _filenames.push_back(filename);
    }

    bool handle(const osgGA::GUIEventAdapter& ea, osgGA::GUIActionAdapter& aa)
    {
        osgViewer::Viewer* viewer = dynamic_cast<osgViewer::Viewer*>(&aa);

        if (!viewer) return false;
        if (_filenames.empty()) return false;
        switch(ea.getEventType())
        {
            case(osgGA::GUIEventAdapter::KEYUP):
            {
                if (ea.getKey() == 'l')
                {
                    osg::ref_ptr<osg::Node> model = osgDB::readNodeFile(_filenames[_position]);
                    ++_position;
                    if (_position >= _filenames.size()) _position = 0;
                    if (model.valid())
                    {
                        viewer->setSceneData(model.get());
                    }
                }
            }
            break;
            default: break;
        }
        return true;
    }

    bool _done;
};

void singleWindowMultipleCameras(osgViewer::Viewer& viewer)
{
    osg::GraphicsContext::WindowingSystemInterface* wsi = osg::GraphicsContext::getWindowingSystemInterface();
    if (!wsi)
    {
        osg::notify(osg::NOTICE) << "Error, cannot create windows." << std::endl;
        return;
    }
    unsigned int width, height;
    wsi->getScreenResolution(osg::GraphicsContext::ScreenIdentifier(0), width, height);
    osg::ref_ptr<osg::GraphicsContext::Traits> traits = new osg::GraphicsContext::Traits;
    traits->x = 0;
    traits->y = 0;
    traits->width = width;
}
traits->height = height;
traits->windowDecoration = true;
traits->doubleBuffer = true;
traits->sharedContext;
osg::ref_ptr<osg::GraphicsContext> gc = osg::GraphicsContext::createGraphicsContext(traits.get());
if (gc.valid()) {
    osg::notify(osg::INFO) << " GraphicsWindow has been created successfully." << std::endl;
ge->setClearColor(osg::Vec4f(0.2f, 0.2f, 0.6f, 1.0f));
ge->setClearMask(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
} else {
    osg::notify(osg::NOTICE) << " GraphicsWindow has not been created successfully." << std::endl;
}
unsigned int numCameras = 2;
double aspectRatioScale = 1.0;
(double)numCameras;
for(unsigned int i = 0; i < numCameras; ++i) {
    osg::ref_ptr<osg::Camera> camera = new osg::Camera;
camera->setGraphicsContext(gc.get());
camera->setViewport(new osg::Viewport((i * width) / numCameras, (i * height) / numCameras, width / numCameras, height / numCameras));
GLenum buffer = traits->doubleBuffer ? GL_BACK : GL_FRONT;
camera->setDrawBuffer(buffer);
camera->setReadBuffer(buffer);
viewer.addSlave(camera.get(), osg::Matrixd(), osg::Matrixd::scale(aspectRatioScale, 1.0, 1.0));
}
void multipleWindowMultipleCameras(osgViewer::Viewer & viewer, bool multipleScreens) {
    osg::GraphicsContext::WindowingSystemInterface* wsi = osg::GraphicsContext::getWindowingSystemInterface();
    if (!wsi) {
        osg::notify(osg::NOTICE) << "Error, cannot create windows." << std::endl;
        return;
    }
    unsigned int width, height;
    wsi->getScreenResolution(osg::GraphicsContext::ScreenIdentifier(0), width, height);
    unsigned int numCameras = 6;
double aspectRatioScale = (double)numCameras;
double translate_x = double(numCameras) - 1;
    for(unsigned int i = 0; i < numCameras; ++i, translate_x = 2.0) {
        osg::ref_ptr<osg::GraphicsContext::Traits> traits = new osg::GraphicsContext::Traits;
        traits->screenNum = multipleScreens ? i / 3 : 0;
        traits->x = (i * width) / numCameras;
        traits->y = 0;
        traits->width = width / numCameras - 1;
        traits->height = height;
        traits->windowDecoration = true;
        traits->doubleBuffer = true;
        traits->sharedContext = 0;
        osg::ref_ptr<osg::GraphicsContext> gc = osg::GraphicsContext::createGraphicsContext(traits.get());
        if (gc.valid()) {
            osg::notify(osg::INFO) << " GraphicsWindow has been created successfully." << std::endl;
        } else {
            osg::notify(osg::NOTICE) << " GraphicsWindow has not been created successfully." << std::endl;
        }
        osg::ref_ptr<osg::Camera> camera = new osg::Camera;
camera->setGraphicsContext(gc.get());
camera->setViewport(new osg::Viewport(0.0, width / numCameras, height));
GLenum buffer = traits->doubleBuffer ? GL_BACK : GL_FRONT;
camera->setDrawBuffer(buffer);
camera->setReadBuffer(buffer);
viewer.addSlave(camera.get(), osg::Matrix::scale(aspectRatioScale, 1.0, 1.0) * osg::Matrix::translate(translate_x, 0.0, 0.0),
        osg::Matrix());
    }
int main(int argc, char **argv) {
Appendix

```cpp
{ osg::ArgumentParser arguments(&argc, argv);

  if (argc<2)
    {
    std::cout << argv[0] << ": requires filename argument." << std::endl;
    return 1;
  }

  std::string pathfile;
  osg::ref_ptr<osgGA::AnimationPathManipulator> apm = 0;
  while (arguments.read("-p", pathfile))
  {
    apm = new osgGA::AnimationPathManipulator(pathfile);
    if (!apm || !apm->valid())
    {
      apm = 0;
    }
  }

  osgViewer::Viewer viewer(arguments);
  while (arguments.read("-s"))
  {
    viewer.setThreadingModel(osgViewer::Viewer::SingleThreaded);
  }
  while (arguments.read("-g"))
  {
    viewer.setThreadingModel(osgViewer::Viewer::CullDrawThreadPerContext);
  }
  while (arguments.read("-d"))
  {
    viewer.setThreadingModel(osgViewer::Viewer::DrawThreadPerContext);
  }
  while (arguments.read("-c"))
  {
    viewer.setThreadingModel(osgViewer::Viewer::CullThreadPerCameraDrawThreadPerContext);
  }

  bool limitNumberOfFrames = false;
  unsigned int maxFrames = 10;
  while (arguments.read("--run-till-frame-number", maxFrames))
  {
    limitNumberOfFrames = true;
  }

  while (arguments.read("-1"))
  {
    singleWindowMultipleCameras(viewer);
  }
  while (arguments.read("-2"))
  {
    multipleWindowMultipleCameras(viewer, false);
  }
  while (arguments.read("-3"))
  {
    multipleWindowMultipleCameras(viewer, true);
  }
  if (apm && apm->valid())
  {
    viewer.setCameraManipulator(apm.get());
  }

  viewer.addEventHandler(new osgViewer::StatsHandler);
  viewer.addEventHandler(new osgViewer::ThreadingHandler);
  std::string configfile;
  while (arguments.read("--config", configfile))
  {
    osg::notify(osg::NOTICE)<<"Trying to read config file "<<configfile<<std::endl;
    osg::ref_ptr<osg::Object> object = osgDB::readObjectFile(configfile);
    osgViewer::View* view = dynamic_cast<osgViewer::View*>(object.get());
    if (view)
    {
      osg::notify(osg::NOTICE)<<"Read config file succesfully"<<std::endl;
    }
    else
    {
      osg::notify(osg::NOTICE)<<"Failed to read config file : "<<configfile<<std::endl;
      return 1;
    }
  }
}
```
while (arguments.read("--write-config", configfile))
{
  osgDB::writeObjectFile(viewer, configfile);
}
if (arguments.read("-m"))
{
  ModelHandler* modelHandler = new ModelHandler;
  for(int i=1; i<arguments.argc();++i)
  {
    modelHandler->add(arguments[i]);
  }
  viewer.addEventHandler(modelHandler);
}
else
{
  ref_ptr<osg::Node> loadedModel = osgDB::readNodeFiles(arguments);
  if (!loadedModel) loadedModel = osgDB::readNodeFile("cow.osg");
  if (!loadedModel)
  {
    std::cout << argv[0] << ": No data loaded." << std::endl;
    return 1;
  }
  viewer.setSceneData(loadedModel.get());
}
viewer.realize();
unsigned int numFrames = 0;
while(!viewer.done() && !(limitNumberOfFrames && numFrames>=maxFrames))
{
  viewer.frame();
  ++numFrames;
}
return 0;
#else
#include <osgViewer/Viewer>
#include <osgDB/ReadFile>
#include <osgDB/WriteFile>
int main( int, char **)
{
  ref_ptr<osg::Node> model = osgDB::readNodeFile("cow.osg");
  for(unsigned int i=0; i<5; ++i)
  {
    osg::notify(osg::NOTICE)<<"New frame *********************************************"<<std::endl;
    osgViewer::Viewer viewer;
    viewer.setSceneData(model.get());
    viewer.run();
    osg::notify(osg::NOTICE)<<std::endl<<std::endl;
  }
  return 0;
#else


Appendix C

In this appendix we attach the figures and data sheets of the different elements that we have used for the performance of the tasks in the implementation.

Datasheet VP-222

![VP-222 Datasheet](image_url)

**Figure 37. VP-222 datasheet [10].**
Appendix

VP-222 picture.

![VP-222 Picture](image1)

Figure 38. Aspect VP-222 [11].

DVI-VGA connector

![DVI-VGA Connector](image2)

Figure 39. Aspect DVI-VGA [12].
Figure 40. LCD monitor datasheet [13].
Appendix

Appendix D

User examination

Some additional and random users (concretely two) have driven the car simulator in order to obtain their impression and evaluate the design accomplished. This is the manner to test the cockpit design from a user point of view. First of all, users drove for five minutes over the environment created testing all the options which form the digital cockpit. They explained their impression answering a questionnaire. This questionnaire consists of eleven questions:

Questionnaire

1. ¿Do you have driving license?
2. ¿How long have you been driving for?
3. ¿Do you drive regularly?
4. ¿Do you have any kind of physical disability?
5. ¿ Do you think the digital cockpit designed resembles to a real car cockpit?
6. ¿ Do you miss any option?
7. ¿ What is your opinion about the feedback provided by the cockpit and moreover by all the car?
8. ¿ Can all the information presented in the cockpit distract your attention when you are driving?

For the next question, firstly, users had to test the cockpit designed by Michael Spendel and Markus Strömberg in the project named “Interface Design in an Automobile Glass Cockpit Environment”. The purpose of this event was to analyze the buttons on the wheel. Project just mentioned used both right and left buttons to choose the different options implemented in the glass cockpit environment. This thesis only uses the right buttons placed on the right side of the wheel (see figure 27 and figure 28).

9. ¿ What do you think is better to use only right buttons or both of them? ¿Why?
10. ¿ Are your sensations very unlike from driving a real car than driving the simulator?
11. ¿ Do you have any additional observation?

Participant one

Information given about this person is that he is male and is twenty-three years old.
1. Yes, i have.
2. I got it for the last five years.
3. Not now, due to my condition of exchange student but I used to drive every day to go at my home university. I drove sometimes in the weekends.
4. No. I have not.
5. Well, actually my car is so old, I think my parents bought it 22 years ago and it has not most of the functions that the simulator do has.
6. No, I do not miss any option. Actually, they are options for me!
7. The car gives much information back. The seat vibrates when you are close to any obstacles. The cockpit information is quite good.
Appendix

8. The information is useful although it is clear that more options you have more possibilities of distraction exist.
9. I do not use to hand buttons in the wheel but I found easier to hand only the right buttons.
10. Totally different, although the appearance is very real, the driving is absolutely different.
11. I have finished very dizzy!

Participant two

This participant is a twenty five years old female.
1. Yes, I have.
2. Since 2001, it was the first thing I did when I came of age.
3. Yeah, I have to, I live in a small town and I need the car for almost every thing.
4. No really, I wear glasses but this is not a physical disability, is it?
5. It looks real even this cockpit collects more options that do not form part of the car of my cockpit. The GPS or Navigation mode as you name it, I have it in an extern and separate module.
6. No, I do not.
7. The feedback is much better than the one I receive from my car.
8. It is quite nice to concentrate the information in one screen instead looking at the radio or the GPS device installed in different modules.
9. I think it is better to use only the right buttons and the right hand because using both hands can distract you very much. It is also very intuitive with one hand only.
10. The velocity sensation is not the same. Although the speedometer marks very high speeds, I feel that I am driving slowly. The gas and the break are not working as well as a real car.
11. The information of the head up displays seems strange for me. In general, it is a very real simulator.