Multimodal Focus Attention Detection in an Augmented Driver Simulator

Alexandre Benoit, Laurent Bonnaud, Alice Caplier, Phillipe Ngo
Laboratoire des Images et des Signaux, Grenoble, France
Lionel Lawson, Daniela G. Trevisan
Communications and Remote Sensing Laboratory, Université catholique de Louvain, Belgium
Vjekoslav Levacic
Faculty of Electrical Engineering and Computing at University of Zagreb, Croatia
Céline Mancas
Falculté Polytechnique de Mons, Belgium
Guillaume Chanel
Université of Genève, Switzerland

***Abstract*— This project proposes to develop a driver simulator, which takes into account information about the user state of mind (level of attention, fatigue state, stress state). The user's state of mind analysis is based on video data and physiological signals. Facial movements such as eyes blinking, yawning, head rotations… are detected on video data: they are used in order to evaluate the fatigue and attention level of the driver. The user's electrocardiogram and galvanic skin response are recorded and analyzed in order to evaluate the stress level of the driver. A driver simulator software is modified in order to be able to appropriately react to these critical situations of fatigue and stress: some visual messages are sent to the driver, wheel vibrations are generated and the driver is supposed to react to the alertness messages. A flexible and efficient multi threaded server architecture is proposed to support multi messages sent by different modalities. Strategies for data fusion and fission are also provided. Some of these components are integrated within the first prototype of OpenInterface (the Multimodal Similar platform).**

***Index Terms*— driver simulator, facial movements analysis, physiological signals, stress, attention level, data fusion, fission, OpenInterface.**

I. Introduction

The main goal of this project is to use multimodal signals processing to provide an augmented user's interface for driving. The term augmented here can be understood as an attentive interface supporting the user interaction. So far at most basic level, the system should contain at least four components:

1. sensors for determining user state of mind;
2. an inference engine feature extractor to evaluate incoming sensor information
3. an adaptive user interface based on the results of step 2
4. an underlying computational architecture to integrate these components.

In fact a fully functioning system would have many more components, but the previous components are the most critical for inclusion in an augmented cognition system and they are covered in the project implementation.

Basically to provide such multimodal application, we address the following issues: which driver simulator to use? How to characterize a user's state of fatigue or stress? Which biological and/or physiological signals to take into account? What kind of alarm to send to the user? How to integrate all these pieces – data fusion and fission mechanism? Which software architecture is more appropriate to support such kind of integration?

A software architecture supporting real time processing is the first requirement of the project because the system has to be interactive. A distributed approach supporting multi thread server can address such needs.

The choice of the driver simulator has to take into account some features such as: open source software, "First person view": (i.e. cockpit view with wheel) and dashboard, source code easy to modify and possible use of a vibration feedback wheel.

We consider two user's states to be detected: stress and fatigue. The detection of these states is based on video information and/or on biological information. From video data we extract relevant information to detect fatigue state while the biological signals provide data for stress detection. Physiological signals could be associated to video data in order to detect fatigue but in the context of the driver simulator used in this project, a real fatigue state is very difficult to obtain. It is possible to simulate fatigue on video data (by closing the eyes or by yawning for example). On the contrary, such kind of simulation is not possible on physiological signals.

This report, as well as the source code for the software developed during the project, is available online from the eNTERFACE'05 web site: www.enterface.net.

The third step is to define what kind of alarms to provide to the user. Textual messages and force feedback are considered to alert the user.

An other challenge of this project is to provide a concrete example in order to test OpenInterface, the multimodal Similar platform.

The rest of the paper is organized as follow: section II present the global architecture of the demonstrator, section III describes how we detect driver's hypo-vigilance states by the analysis of video data, section IV presents how to detect driver's stress states by the analysis of some biological signal, sections V and VI describe the data fusion and fission strategies and section VII gives details about the demonstrator implementation.

II. CONCEPTUAL ARCHITECTURE

The diagram of Figure 1 presents the conceptual architecture of our attentive driver simulator. We propose a distributed approach to integrate our components. On one PC under Linux we have integrated all video data based detection and analysis as well as the fusion and fission components. An other PC under Windows is used to run the driver simulator and a third PC is used for biological signals acquisition and analysis. Communication between all the PCs is done exchanging XML messages. For that the Dialog Controller included in the driver Simulator software should be able to receive multi messages (i.e. from biological signals station and from video based station). In this case a multi thread server approach is developed and included in the driver simulator.

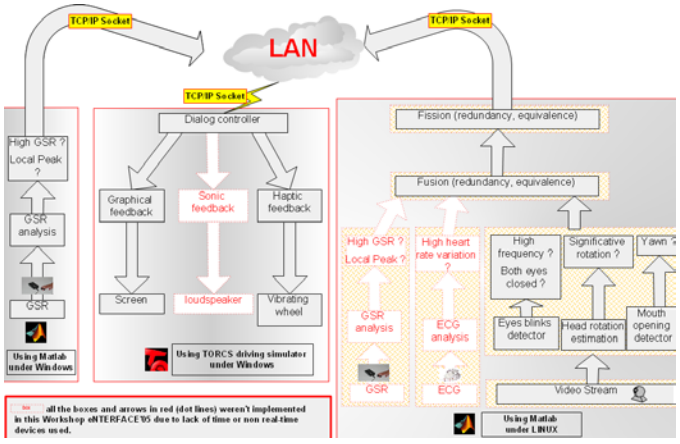

Figure 1: Overview of the system architecture

III. HYPO-VIGILANCE DETECTION BASED ON VIDEO DATA

The state of hypo-vigilance (either related to fatigue or inattention) is detected by the analysis of video data. The required sensor is a camera facing the driver. In this project, three indices are considered as hypo-vigilance signs: yawning, head rotations and eyes closing for more than 1s.

A. Face detection

Face detection is the first and maybe the most crucial step of the image processing phase. The face detector should be robust (no error in face localization) and should work in real time. The chosen face detector is the free toolbox MPT [5]. This face detector extracts a square-bounding box around each face in the processed image. Face detection is done for each image of the sequence without any face tracking. The advantage is that the head is not lost because of tracking error propagation. The main drawback is the decrease of the frame rate even though MPT works nearly in real time for pictures of size (320x200 pixels), which is not the case of other face detectors such as OpenCV [13] for example.

Whichever face detector you use, the extracted face bounding box is not exactly the same from frame to frame so that we use a temporal median filter with temporal adaptive position mean to make the spatial localization of the face temporally stable (note that the size of the bounding box of the face is supposed to be constant during an experiment: in a car the driver face distance w.r.t the camera is stable if the driver stays on his seat).

B. Head motion analysis

Once a bounding box around the driver face has been detected, head motion such as head rotations, eyes closing and yawning are detected by using an algorithm working in a way close to the human visual system. In a first step, a filter coming from the modeling of the human retina is applied. This filter enhances moving contours and cancel static ones. In a second step, the FFT of the filtered image is computed in the log polar domain as a modeling of the primary visual cortex. Figure 2 gives a general overview of the algorithm.

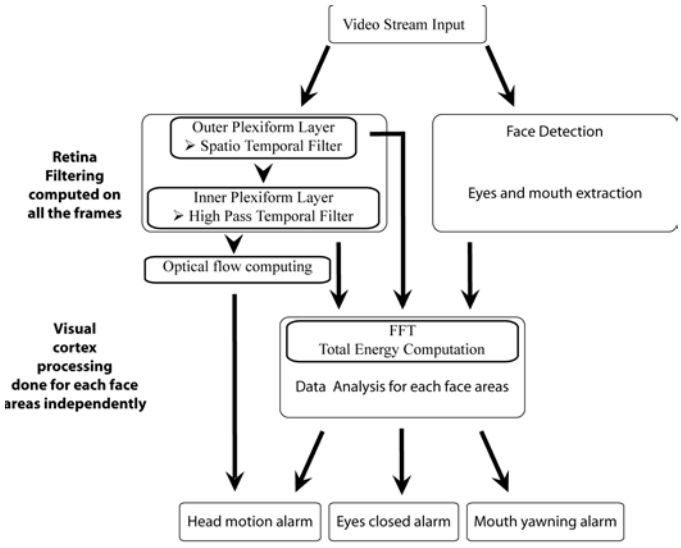

Figure 2: Algorithm for hypo-vigilance features extraction from video data

The first step consists in an efficient prefiltering [1]: the retina OPL (Outer Plexiform Layer) that enhances all contours by attenuating spatio-temporal noise, correcting luminance and

whitening the spectrum (see Figure 3) . The IPL filter (Inner Plexiform Layer) [1] removes the static contours and extracts moving ones.

The second step consists in a frequency analysis of the spectrum of the OPL and IPL filters outputs in each region of interest of the face: global head, eyes and mouth (see section C for the description of eyes and mouth region of interest extraction).

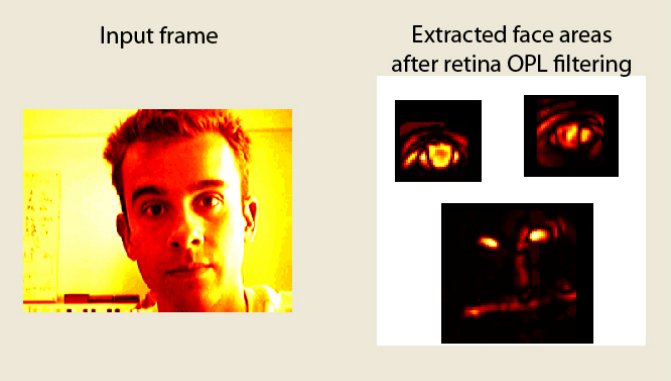

Figure 3 : OPL filtering results

In order to estimate the rigid head rotations [3], the proposed method analyses the spectrum of the IPL filter output in the log polar domain. It detects head motion events and is able to extract its orientation. Cortical optical flow filters [2] are oriented filters that compute the optical flow of the global head to extract the motion direction. Optical flow is computed only when motion is detected.

For the detection of yawning or eyes closing, three identical processes are done independently [4]. On each region of interest (each eye and the mouth), a spectrum analysis of the OPL and IPL filters output is done for motion event detection: we are looking for vertical motion related to eyes closing or to yawning.

C. Eyes and mouth detection

The mouth can be easily extracted in the lower half of the detected bounding box of the head. The detection algorithm will work even if the mouth is not perfectly centered in the area because we analyze the spectrum energy instead of spatial features, which is more robust. Moreover, there are no disturbing contours in that area that could generate false detections.

Concerning the eyes, the spectrum analysis in the region of interest is accurate only if each eye is correctly localized. Indeed around the eyes, several vertical or horizontal contours can generate false detection (hair boundary for example).

The MPT toolbox proposes an eye detector but it requires too much computing time so that it is not compliant with real time constraint. We use another solution: eye region is supposed to be the area in which there is the most energized contours. To do so, assuming that the eyes are localized in the 2 upper quarters of the detected face, we use the retina output. The retina output gives the contours in these areas and due to the fact that the eye region (containing iris and eyelid) is the only area in which there are horizontal and vertical contours, the eye detection can be achieved easily. We use two oriented low pass filters: one horizontal low pass filter and a vertical low pass filter and we multiply their response. The maximum of the result is obtained in the area in which there are the most horizontal and vertical contours that is an eye region. To make the eye areas temporally stable, their position is smoothed from frame to frame using adaptive mean positions. This eye detection takes about 6 operations per pixel for each search area (i.e. each upper quarter of the face bounding box).

Figure 4 gives an example of the input picture of the eye detector; bright areas are the most important contours. Figure 5 shows the output of the horizontal vertical filters.

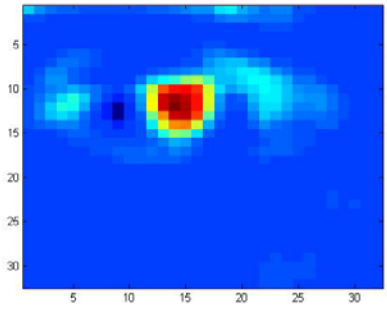

Figure 4: Input picture for eye detection : one of the 2 upper quarters of the face-bounding box

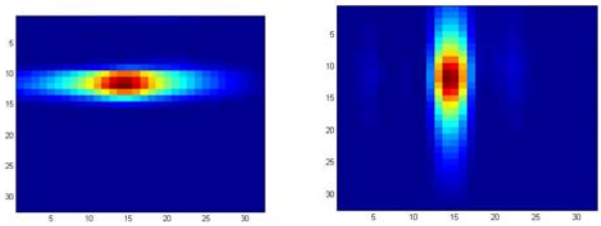

Figure 5: Output of the vertical and horizontal low pass filters, both filters report maximum amplitude on the eye center.

D. Hypo-vigilance alarms generation

- We generate an alarm when both eyes are closed longer than a specific time period (1 second for example).
- We detect mouth yawning: when a yawn occurs, the mouth is wide open, then, this generates a very high-energy increase on the spectrum that can be easily extracted.
- The global head motion events are detected with the global head spectrum analysis. We only extract the fact that a head motion has occurred. The proposed algorithms are able to extract the motion direction with the cortical optical flow algorithm, but it is not yet integrated in the fusion system.

E. Fusion strategy

After the video analysis, Boolean information about yawning or not, about eyes closing or not and about head moving or not are available. A very simple and easy to compute fusion strategy based on the three index is proposed:

if head motion is detected
send an alarm to the user
hypo-vigilance value=100
else
if both eyes are closed during 1s
send an alarm to the user
hypo-vigilance value = 50
if the driver is yawning
send an alarm to the user
hypo-vigilance value = 50+hypovigilence value
end

The variable hypo-vigilance associated to each index is set to 50 or 100. The highest the value, the highest the hypo-vigilance.

Note that in this very simple fusion strategy, information about head motion kind of rotation is not taken into account. A more sophisticated fusion strategy has been tested and is described in section V.

IV. Stress Detection Based on Biological Signals Analysis

Physiological signals are used in order to detect stress situation. ECG (Electrocardiogram) and GSR (Galvanic Skin Response) are announced by literature as very promising to detect driver stress in real situations [11, 12]. In a stressful time, the GSR signal and the heart rate signal (extracted from the ECG) are supposed to increase. Two different experiments have been considered; they aim at detecting either driver stress over a long time period or punctual driver stress.

In this experiment, we use the Biopac system MP30B-CE for ECG and GSR acquisition.

Figure 6: On the left, ECG devices and on the right, GSR device

The main drawback of the data acquisition system is that for the moment, on line analysis is not possible. For that reason, the study on biological signal for stress detection has not been implemented in the final demonstrator.

A. ECG signals analysis

1) Prefiltering

Since we are analysing the stress state of a driver, the ECG can be disturbed by several muscle artefacts that generally come from hands or arms movements (see Figure 7). This is why it is necessary to pre-filter signals.

Figure 7: Driver with electrodes on the wrists for ECG measure

The pre-filtering is based on the characteristics of the ECG peaks: we observed that these peaks contain energy in the frequency band 10-35 Hz. As we do not need other components of the signal we choose to band-pass the signal using this interval and a Butterworth IIR filter with 8 coefficients. Figure 8 shows the original signal and the results after filtering.

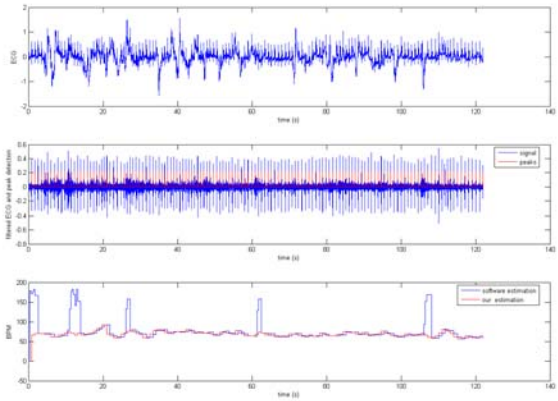

Figure 8: ECG before (first line) and after pre-filtering (second line); computed heart rate from ECG (third line, red signal)

2) Heart rate computation

The first thing to do, before computing heart rate, is to identify peaks in the filtered signal. For this, we use a reference record (ECG when the subject is supposed to be relaxed) as a baseline to identify the general height of the peaks depending on the subject. We define the general height of peaks as one third (chosen empirically) of the maximum value. In order to improve the peak detection, we also use a priori information: we consider that the heart rate cannot exceed 180 BPM (Beats Per Minute). If two peaks are too close, so that they do not validate this assumption, we keep only the one with the maximum value.

Finally, the heart rate is computed by evaluating the number n of samples between two peaks and by using this simple formula:

$$HR = 60/ (n*(1/f_e))$$

Where f_e is the sampling rate.

3) Stress level assessments

In case of unexpected or stressful events, the heart rate does not increase as generally assumed, but one can observe a raise in its variation. In order to determine this variation, we use the absolute value of the first derivative of the signal. After smoothing the result by a Gaussian filter we obtain what we call the stress level (see Figure 9).

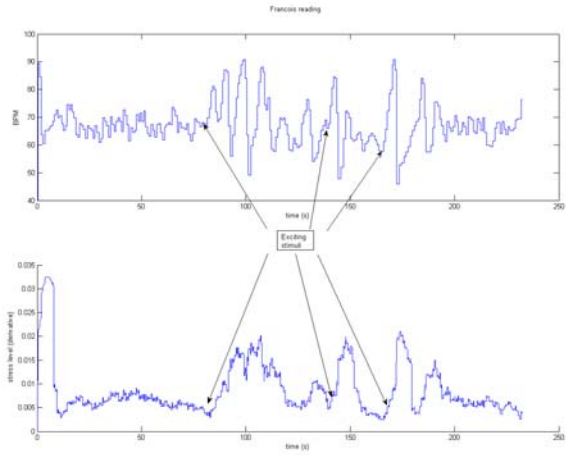

Figure 9 : stress level (bottom) computed from heart rate (top) for exciting stimuli

B. GSR signal analysis

Due to the chosen sampling rate (200 samples per second) and the apparatus, some artefacts occur in the initial signals and a filtering is also required. After trying several smoothing filters and because of the large variability in the conductivity of each user, we opt for a multiscale median filtering. Four successive filters are used with a decreasing window size (100, 50, 30 and 20 samples).

1) Global stress detection

We can easily measure the minimum and maximum of the user's GSR level. By normalizing the signal to analyze with these values as usual:

$$\frac{GSR\_level\_to\_analyse - \min(GSR\_rest)}{\max(GSR\_rest) - \min(GSR\_rest)}$$

where *GSR_rest* correspond to the values of the GSR when the user is supposed to be relaxed.

We can then define a score to know the global stress level. Global stress can occur after a very difficult day of work or when the traffic jam is increasing for example. Global stress is related to slow but constant increase of the GSR. The global stress level is used to know the initial state of the driver or with a sliding window, to know the global state of the driver in a certain amount of time.

2) Local stress detection

Local stress is supposed to be related to punctual and unforeseen events as it could occur on roads such as a pedestrian crossing and so on. Local stress detection can be modelled as high peaks in GSR signals. GSR signals have the property to react quite quickly to an event but to have a decreasing response to go back to a calm situation very slowly.

We used this particular property as a priori information in our algorithm.

First of all, we detect local maxima using the watershed algorithm. Local maxima correspond to watershed pixels.

Once maxima are detected, we keep only those, which have a difference of 1 unit with the previous maximum. This threshold of 1 unit is based on the correlation of punctual events, precisely recorded, and GSR signals. Then, we remove all maxima, which are too close to each other by keeping only the highest one. This rule is based on the assumption that if two maxima are too close to each other (inferior to 5 seconds), they belong to the same event. Figure 10 presents an example of GSR local increasing detection: each peak on the bottom curve corresponds to a stress alert.

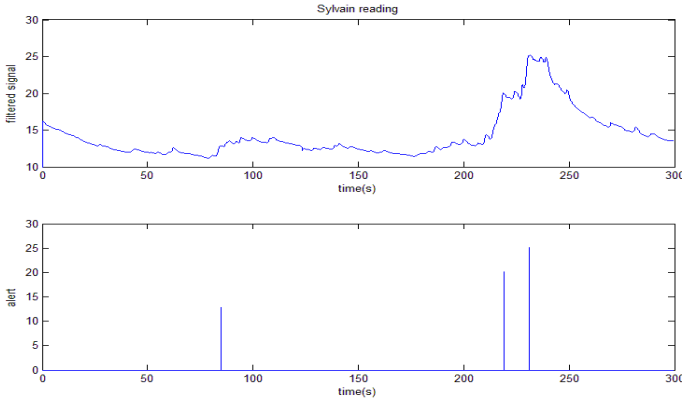

Figure 10: GSR record and punctual stress event detection

V. Fusion Strategy

In this section, we describe and test a data fusion based on Bayesian Network. It is used for the purpose of hypo-vigilance detection but it also represents a global fusion method for the integration of additional information in the detection process. Note that this fusion process is not integrated in the final demonstrator for the moment due to the lack of significant data. Both fusion strategies are implemented in the demonstrator but for the moment, only the simplest one described in III.D. is used by default for computational cost reduction.

Human fatigue generation is a very complicated process. Several uncertainties may be present in this process. First, fatigue is not observable and it can only be inferred from the available information. In fact, fatigue can be regarded as the result of many contextual variables such as working environments, health and sleep history. Also, it is the cause of many symptoms, e.g. the visual cues, such as irregular eyelid movements, yawning and frequent head tilts. Second, human's visual characteristics vary significantly with age, height, health and shape of face. To effectively monitor fatigue, a system that integrates evidences from multiple sources into

one representative format is needed. Naturally, a Bayesian Networks (BN) model is a good option to deal with such an issue.

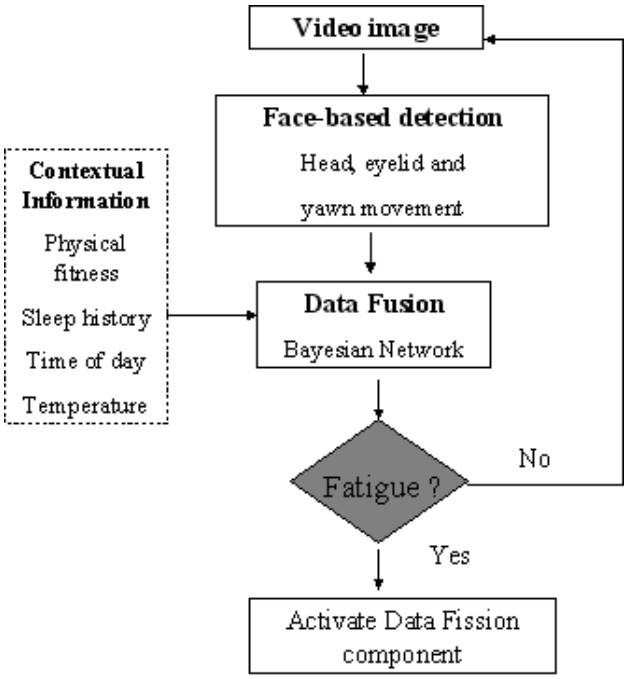

Figure 11: Fusion strategy based on a Bayesian Network

A BN provides a mechanism for graphical representation of uncertain knowledge and for inferring high-level activities from the observed data. Specifically, a BN consists of nodes and arcs connected together forming a directed acyclic graph. Each node can be viewed as a domain variable that can take a set of discrete values or a continuous value. An arc represents a probabilistic dependency between the parent node and the child node.

Some contextual information such as temperature, time of day, sleep history, etc can be used to build a prior probability for the fatigue node. For that we use the parameters proposed in [7]. For the face data fusion we have considered a very preliminary version where the network evidences change when: eyes closed more than 1 sec; yawning occurs; down head motion are detected simultaneously or not. As result we got the level of fatigue, which is sent to the data fission component.

VI. Fission strategy

Data fission duty is to collect the data from data fusion and to generate an alert XML message that is sent to the driver simulator. Data fission function is called at the rate the driver state detection is progressing. Generated messages are in XML format. We decided for XML because it is extendable and messages are sent only when the driver state changes. Driver state may be defined by a fatigue value (either coming from the Bayesian Network result or from the simple fusion process) that is an output variable of data fusion. For example, we can set the range of values for fatigue level that determine the driver state. For those range of values we can define different screen messages and wheel shaking power. Table 1 and Table 2 present the fusion strategy for the simple method and for the Bayesian network based method respectively.

Fatigue range	50	50	100
Message	Open the eyes	Yawning: be careful	Stop moving the head
Shaking power	'100'	'100'	'100'

Table 1: fission strategy with the simple fusion process

Fatigue range	[0,33]	[33,66]	[66,100]
Message	''	'Tired'	'Asleep'
Message color	''	'Green'	'Red'
Shaking power	'0'	'0'	'100'

Table 2: fission strategy with the BN based method

Data fission only creates the message if the driver state has changed and is different than the previous driver state. If the user state is the same as in previous call, data fission generates 'NOT_CHANGED' message. In that way the XML message does not need to be sent to the driver simulator after each call of the data fission function.

Once the alert message has been sent, the driver is supposed to acknowledge to the system that the message has been understood. For example, in the case of the simple fusion process, each time an alert is detected, wheel vibrations are triggered. The driver has to stop these vibrations by pushing a button. The reaction time is also recorded, this time being correlated with the hypo-vigilance or fatigue user state.

VII. Demonstrator

A. Overview of the global system

The developed demonstrator is made of

- 2 PCs: one under Windows for the driver simulator and one under Linux for hypo-vigilance states detection
- 1 SONY digital camera
- 1 LOGITECH force feed back wheel
- 1 projection screen
- 1 video-projector
- 2 loudspeakers

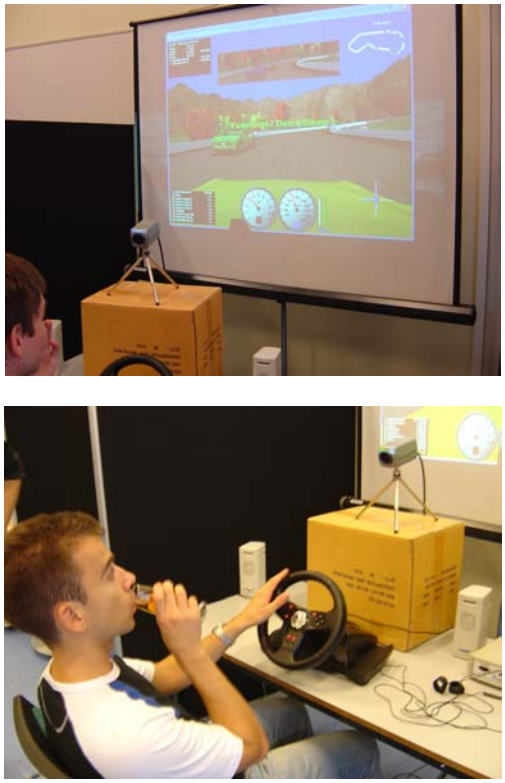

Figure 12: Global views of the demonstrator

On the used computer (Pentium 4 2.4Ghz), the frame rate is about 5 frames per second but it could be increased up to 8 frames per second thanks to some MPT optimization.

B. *Driver Simulator*

Around ten driver simulators have been studied. The chosen driver simulator is TORCS [9] because it is a well architectured GPL program with well structured source code and a well designed user interface.

This simulator is working under Linux and windows platforms. The main sources are written in C++ with the OpenGL library. The graphics quality of the simulator is correct and it has a first person view. Figure 13 presents an illustration of TORCS simulator.

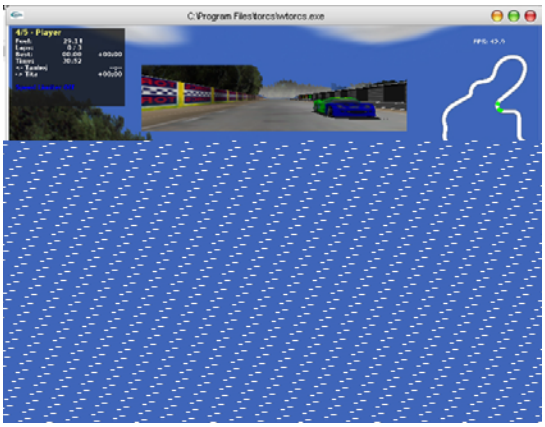

Figure 13: Torcs driver simulator illustration

We integrate an interaction from the Data Analysis Kernel to our driving Simulator.

The main work consisted in

- Allowing a Text Message to be displayed within the game graphical interface.
- Creating a multi-threaded Server within the application whose purpose is accepting different clients connexions.
- Integrating a force Feedback wheel in order to warn the user with an other modality than the visual one.
- Allowing the user to make a feedback on the message displayed by stopping it.
- Parsing XML messages from the multimodal analysis of the driver. Indeed, it is possible to change the color, the string of the sent message and the feedback power.

C. *Implementation of hypo-vigilance detection*

Due to the fact that ECG and GSR signals cannot be processed on line with the data acquisition station we used, the detection of stress state has not been implemented in real time. Only the detection of hypo-vigilance state based on video data is available at the moment.

1) *Face detection algorithm modifications*

For face detection, we use the Matlab implementation of *mpiSearch* function belonging to the MPT library, which receives a RGB or Gray level frame as input. Outputs of the function are the bounding box coordinates of the detected face.

Due to the relative slowness of the *mismatch* function developed under Matlab a fine study of the algorithm has been done in order to increase the computational rate. We managed to figure out how the algorithm behaves in dynamic environment when the video is acquired with the help of DirectInput library. While streaming, *mpiSearch* uses a special object that caches the detected face-bounding box.

The trick is to modify the *mpiSearch* Mex function and to put this object as a global DLL variable. Global DLL variables are preserved in Matlab memory space after the DLL is first accessed by Matlab. In that way after each Matlab call of the *mpisearch* function caching state is preserved.

With using the Microsoft VS.NET 2003, we additionally increase the performances by compiling the code for new Pentium 4 generation of processors.

With all these modifications, we achieve about 2.7 times speed increase.

In order to increase the frame rate again, some of time-consuming parts of the *mpisearch* code may be written in assembler language. This is beyond of scope of the project.

D. *Alert message generation*

1) *The Display Changes*

Three different messages depending on the index of hypo-vigilance can be displayed on the first view of the driver simulator (see Table 1 for the different considered messages and Figure 14 for an example of message incrustation during the game). In order to show the message, we need to change all the graphical classes within the source code. The communication between the main program and all the libraries is created by using some global variables and also by creating new links between several libraries.

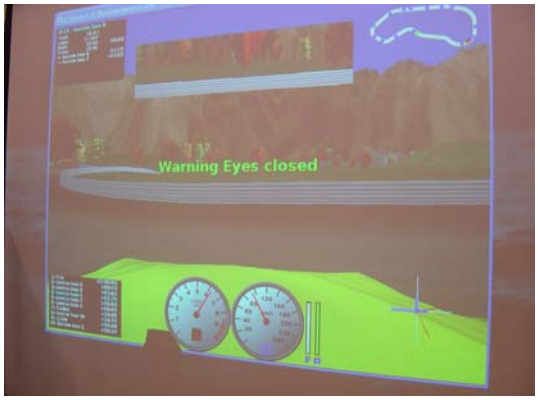

Figure 14: alert message in case of long eyes closing

2) Force Feedback implementation

Force Feedback is used to make the wheel shaking when hypo-vigilance is detected. Shaking power is defined by XML message of data fission. Shaking becomes stronger and gradually reaches its maximum value. Force Feedback uses the DirectInput library, a part of the Microsoft DirectX SDK. The library is based on "The Force Feedback Direct Input Library (DIL)" made by Bryan Warren and Alex Koch. This library can be loaded at [8]. In our project, the library has been altered from functions to class. In class we can set the time period that shaking needs to reach the maximum vibration time, vibration activation threshold and to modify the shaking of the wheel dynamically. We also use the class to check whether the button is pressed, and if pressed stop the wheel shaking.

3) Message parsing and controlling the input devices of driver simulator

After the XML message arrives through the socket connection it is parsed. We use Microsoft XML parser to parse the message. You can download the MSXML parser from Microsoft web site. After the parsing, controller class activates the screen display message or starts the wheel shaking.

4) The Server Side:

We choose to implement a Server side for the interaction between the multimodal devices and the user. The network protocol used is TCP/IP. We implement this socket by using threads. Those threads access global variables under mutual exclusion. We use a "GPL" library called Openthreads for this implementation.

E. Openinterface integration

1) OpenInterface: a short presentation

OpenInterface is the Similar Software Platform that includes software components dedicated to multimodal interaction and multimodal data fusion. OpenInterface integrates results from the Human-Computer Interaction (HCI) community as well as from the Signal Processing community.

Each component is registered into OpenInterface Platform using the Component Interface Description Language (CIDL described in XML). The registered components properties are retrieved by the Graphic Editor (Java). Using the editor the user can edit the components properties and compose the execution pipeline of the multimodal application. This execution pipeline is sent to the OpenInterface Kernel (C/C++) to run the application. The OpenInterface Tutorial can be found on the Similar web site [10].

2) OpenInterface implementation of the demonstrator

Currently, OpenInterface is in its early development stage and this driver simulator project is used as a test bed for the working prototype. The latter provides several services and also has limitations.

The services provided by this first OpenInterface prototype are:

- Interface Description Language for the specification of reusable software code.
- Seamless integration of reusable heterogeneous software (C/C++, Java, Matlab).

Some limitations are:

- The description of a software interface is currently not automated and has to be written by hand. This can be cumbersome as the language syntax is very strict and the validity of a description is not heavily enforced besides DTD checking. This lack of robustness might lead to an inappropriate binding and therefore to unexplained application crash.
- At the beginning of the project, it was impossible to perform two ways communication with any Matlab script. The latter could only be called by OpenInterface component.

Several issues have been encountered during the integration phase:

- First of all as the current prototype is only running under Linux, we had to reduce the set of software to those that could be run under Linux. It has been decided that the communication with non-compliant Linux software will be done through network communication. Thus, only the following software code have been integrated into OpenInterface: firewire camera driver, face detection, head motion analysis, fusion for hypo-vigilance detection, fission strategy for alerting the user and a Java GUI to start the application. Figure 15 shows how these components are connected into OpenInterface.
- After deciding which software to use as component into OpenInterface, the main problem we had to face has been to make those components reusable. Indeed, in their first version, the software was designed for a particular goal and could not be reused by a third-part (e.g. OpenInterface) as tools. Thus, we redesigned the components interaction interfaces. To avoid this redesigning step in the future, modular programming habit should be enforced from the beginning among components programmer.
- From this step we have pulled out that a two-way communication with Matlab script is mandatory to

prevent heavy code rewriting. Indeed the camera driver has to be called by the face detection component, not the contrary. Therefore, due to the Matlab Engine API limitations, the only way to allow Matlab script to interact with component already running into OpenInterface is to perform communication through UNIX socket. Of course this is transparent to the user.

- The third integration step was the description, in CIDL, of all components interface. This step was straightforward and we did not find any difficulties in expressing the interfaces in the integration platform's CIDL.
- Finally we started the integration of the components. It is performed in a gradual way so that we were able to test each component inside OpenInterface and point out the incompatibility. Some problems arose when we put two Matlab components in the same execution pipeline. It turned out that due to another undocumented Matlab Engine API limitation, we could not call another m-script while another is running. The lack of time directed us to fix the problem by having one Matlab engine instance per Matlab script taking part in a pipeline. One would think that this solution would drastically slow down the computer but that did not happen. A Matlab process actually uses resource proportional to the computation done by the running script.

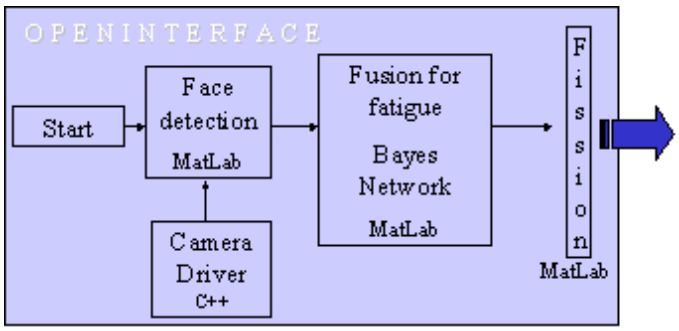

Figure 15: Openinterface components coming from the project

As a conclusion of this integration phase using OpenInterface platform, we can say that thanks to this first project integration a lot of feedback has been collected to improve the current prototype of the OpenInterface tool in order to provide another improved version as soon as possible. The future engineering work on OpenInterface will be narrowed to improve the integration of Matlab components and also to provide a user with a friendly integration interface. This would allow distributing the platform and letting people test it on their own.

VIII. Future Works and Conclusion

During the project, we have developed an augmented driver simulator based on video analysis for driver's attention controlling. First promising studies about physiological data have to be improved and integrated in the global system. This will induce the development of an appropriate data fusion method in order to control both the driver's attention level and the driver's stress.

Once the driver has been alerted, it will be necessary to perform some specific tests in order to control that driver's stress or fatigue has actually decreased.

For the moment, the global system is running almost 10 frames per second. It will be necessary to optimize video data analysis algorithms in order to speed up the frame rate.

Acknowledgment

This work was supported by the Similar network of excellence [10]

References

[1] Beaudot W., "The neural information processing in the vertebrate retina: A melting pot of ideas for artifficial vision", PhD Thesis in Computer Science, INPG (France) december 1994.

[2] Torralba A. B., Herault J. (1999). "An efficient neuromorphic analog network for motion estimation." IEEE Transactions on Circuits and Systems-I: Special Issue on Bio-Inspired Processors and CNNs for Vision. Vol 46, No. 2, February 1999.

[3] Benoit A. , Caplier A. "Head nods analysis : interpretation of non verbal communication gestures " IEEE, ICIP 2005, Genova, Italy

[4] Benoit A. , Caplier A. "Hypovigilence Analysis: Open or Closed Eye or Mouth ? Blinking or Yawning Frequency ?" IEEE, AVSS 2005, Como, Italy

[5] Machine Perception Toolbox (MPT) http://mplab.ucsd.edu/grants/project1/free-software/MPTWebSite/API/].

[6] Bayes Net Toolbox for MatLab [http://www.cs.ubc.ca/~murphyk/Software/BNT/bnt.html].

[7] [Qiang Ji, Zhiwei Zhu and Peilin Lan, Real-Time Nonintrusive Monitoring and Prediction of Driver Fatigue, IEEE Transactions on Vehicular Technology, Vol. 53, No. 4, July, 2004, p1052-1068].

[8] Force Feedback Direct Input Library http://courses.washington.edu/css450/Fall2003/web_contents/direct_input_lib/DirectInput.html

[9] TORCS Driver Simulator: http://torcs.sourceforge.net/

[10] Similar Network of Excellence: www.similar.cc

[11] S. K. Lal, A. Craig, "Driver fatigue: electroencephalography and psychological assessment", Psychophysiology, 39, 3, May 2002, 313-21.

[12] J. Healey, J. Seger, R. Picard, "Quantifying driver stress: developing a system for collecting and processing bio-metric signals in natural situation", MIT technical report n°483.

[13] OpenCV : www.intel.com/technology/computing/opencv

Alexandre Benoit was born in 1980 in France. He graduated from Institut Nationnal Polytechnique de Grenoble (INPG). His PhD subject concerns head motion analysis. His work is based on the human visual perception system. He teaches signal processing to engineering students. He prepares his PhD from the INPG, his thesis started in october 2003 at the Laboratoire des Images et des Signaux (LIS) in Grenoble.

Laurent Bonnaud was born in 1970. He graduated from the École Centrale de Paris (ECP) in 1993. He obtained his PhD from IRISA and the Université de Rennes-1 in 1998. Since 1999 he is teaching at the Université Pierre-Mendès-France (UPMF) in Grenoble and is a permanent researcher at the Laboratoire des Images et des Signaux (LIS) in Grenoble. His research interests include segmentation and tracking, human motion and gestures analysis and interpretation.

Alice Caplier was born in 1968. She graduated from the École Nationale Supérieure des Ingénieurs Électriciens de Grenoble (ENSIEG) of the Institut National Polytechnique de Grenoble (INPG), France, in 1991. She obtained

her Master's degree in Signal, Image, Speech Processing and Telecommunications from the INPG in 1992 and her PhD from the INPG in 1995. Since 1997, she is teaching at the École Nationale Supérieure d'Électronique et de Radioélectricité de Grenoble (ENSERG) of the INPG and is a permanent researcher at the Laboratoire des Images et des Signaux (LIS) in Grenoble. Her interest is on human motion analysis and interpretation. More precisely, she is working on the recognition of facial gestures (facial expressions and head motion) and the recognition of human postures.

Guillaume Chanel was born in 1978 in Switzerland. He obtains both his engineering diploma in computing and robotics and his master degree in automatics during the 2002 year. He is currently a PhD student at the Computer Vision and Multimedia Laboratory (CVML) of the University of Geneva. His research interest is to detect emotional states from recordings of EEGs and other physiological signals in order to improve human computer interactions.

Lionel Lawson was born in 1982 in Bénin. He graduated from the Engineering School of Université Catholique de Louvain (UCL) and obtained his Master degree in Computer Science and Engineering in 2004. He is currently working at the Communication and Remote Sensing Laboratory (TELE) on the development of OpenInterface, an open source component-oriented integration platform.

Vjekoslav Levacic was born in 1981 in a small but pleasent city at north of Croatia called Cakovec. He ended two high schools, gymnasium and classical music high school. In 2000 he become an student in Faculty of Electrical Engineering and Computing in University of Zagreb. He has worked on various projects including buiding the enterprise systems and web applications.

Céline Mancas-Thillou holds two Master degrees, in Audiovisual Systems and Networks Engineering (ESIGETEL, 2002) and in Applied Sciences (FPMS, 2004). She is working for the TCTS lab since January 2003 and is pursuing a PhD in Applied Sciences since March 2004. Her research deals with text extraction, segmentation and degraded character recognition in SYPOLE project. She has been a visiting PhD student at the University of Bristol for 3 months in 2005 to work on Super Resolution Text for an embedded application.

Phillipe Ngo was born in 1980 in France, He is still undergraduated from the UTBM (university of technology of Belfort Montbéliard (France)).
His major is computer science with specialization in Picture, interaction and virtual reality. He is currently working for the LIS (INPG Grenoble France) for its last internship of computer science. His work is focused on human and computer interactions within a virtual reality environment.

Daniela G. Trevisan was born in Santa Maria, Brazil, on 1974. She graduated in Informatic at the University Federal of Santa Maria, Brazil (UFSM) in 1997. She obtained Master degree in Computer Science from University Federal of Rio Grande do Sul, Brazil (UFRGS) in 2000. Currently she is PhD student at the Université Catholique de Louvain (UCL) at the Communication and Remote Sensing Laboratory (TELE) and she is also member of the Belgium Computer-Human Interaction Laboratory (BCHI). Her research topics are focused on human-computer interaction (HCI) field such as modelling multimodal interfaces, model based-approach, augmented and mixed reality and multimodal interfaces for image-guided surgery.