MSc. MKE

Real Time Eye Blink Detection using a Configurable Processor

Presented at the Faculty of EWI

of the Delft University of Technology (Netherlands) by

Remco Geert Jan Janssen

in Oktober 2010

Graduation Committee:

Prof. Drs. Dr. L.J.M. Rothkrantz

Dr. Ir. P. Wiggers

Ir. H.J.A.M. Geers

Ir. L. Dricot

Preface

AW Europe▶[1] is a company specialized in Automatic Transmission and Navigation Systems. It is a world wide company, with two locations in Belgium. One, located in Mons (Belgium) with primary goal to create and build the Navigation systems. While the other department is located in Braine L'Alleud, under the smoke of Brussels (Belgium). First part of my Master project is performed at the Braine l'Alleud site at the Advanced Development Department under the supervision of Dr. Ir. Stéphane Petti. At the start of the internship a new perception processor was developed by a French company. The main question of AW is what this processor is and what it could do. In order to find answers to this question, a series of tests where performed to gain understanding of the processor.

This question was the start of the project. Besides finding the possibilities of the processor, some constraints where put on the project. Initially a Software Development Kit (SDK, Gloss. 2) was available that simulated the processor. Goal of the SDK is to get familiarized with the processors methods and objects. The SDK is implemented in C++ and provides tutorials on how to get simple examples working. Given the way the processor works, it takes some time to get familiar with the structure and approach (more on this in: 3.4 Theory & Concepts) but after some reading in the tutorials it is quite easy to develop simple running prototypes. Using the SDK, it was possible to perform some early tests with the processor. Results looked promising and next step was to continue with the perception processor itself. Those results and experiments led to this Thesis.

[1]http://www.awtce.be

Real Time Eye Blink Detection using a Configurable Processor

Remco Janssen
Student No. 1274996
October 2010

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Man-Machine Interaction Group

Abstract

This thesis describes the approach to create a real time eye blink detector using a new perception processor. Tried is to describe how the processor works and to find out possible fields of application. Supposed strength of the processor is it's speed combined with the possibility of changing the configuration at run time. Since the current version is a prototype, the perception processor is still undergoing some improvements and updates. By performing extensive tests not only to the domain of eye blink detection, advice is given about promising directions for further development of the processor. Ultimately, the demonstrative application created for this thesis is robust enough to be used as a real time blink detector that can be used in different domains of computer vision. In order to create the blink detector, a step by step approach is taken. First, the face is located and tracked. Next, the eye region is to be found, which should lead to real time locating and tracking of the pupils. Finally the blinks are detected to give an indication of a persons blink rate. Goal is to find out the limitations and the field of application for the perception processor.

Acknowledgments

I would like to thank AW Europe for offering me the opportunity to participate in their advanced development team. Seeing and experiencing what it is to work in a growing company and how to deal with the issues evolving from such work. The ADEV team really helped me finding my way in the company and spending quality time in the Belgian pubs. Special thanks go to my supervisors, Stéphane Petti, Alexandra Degeest and Lionel Dricot for stimulating me and for all the help and tips they gave me. On a personal level, I would like to thank Blandine for her hospitality and good company during my Internship. Thanks to Professor Rothkrantz as well, for the discussions and for giving me the opportunity to realize my own project. His experience helped me through some of the hard times throughout my project. Also, I like to thank the TU Delft for letting me use all the equipment needed to perform my tests.

Furthermore, I would like to give my gratitude to my cousin, Solenn, for all that he has done and for helping me to arrange this trainee ship. Tante Mieke, for her delicious bread and support. I can't forget my roommates Frank Beers and Jeroen Kwast, for their enthusiasm and for participating during the experiments. Off course, I will not forget my parents, for their efforts and financial support. Without them, this project would not have been possible. Last but certainly not least, I would give special gratitude to my girlfriend, Esther, for her patience and for being there.

Table of contents

1 Introduction

Since the beginning of the computer age, new technologies emerge in the blink of an eye. People get accustomed to new devices and software to make their everyday life easier. People demand new technologies to perform an increasing amount of tasks. For navigation systems, this implies that giving directions alone no longer fulfills the users' needs. Companies are constantly looking for ways to improve their navigation systems as to close the gap between device capabilities and users expectations. Be it in simplifying the ways of interaction, or by expanding the tasks that can be performed.

1.1 Problem definition

One of those companies is AW Europe, located in Braine L'Alleud under the smoke of Brussels. Through the acquisition of a new prototype processor, which enables to find the location of the eyes, tried is to find ways to expand the possibilities of a new generation of navigation systems. This processor, designed to mimic task in the same fashion as the human brain, offers a whole lot of new possibilities. What tasks can be accomplished by the processor and how well do they perform compared to well known or established techniques? Given the prototype stadium of the processor, it's an illusion to think the processor can outperform the existing techniques, but the flexible nature could result in a solution applicable to a wide range of problems.

More specific, the domain of blink detection is chosen as a test case to test the performance and reliability of the processor, since blink detection is a known problem and is known for it's complexity given the fact that blinks typically occur very fast and appear in a small area of the human face. How well can the processor perform in this area and what other possibilities are present to which the processor can be a useful tool.

1.2 Research questions

Goal of the project is to determine how well the perception processor can perform in the domain of blink detection inside a car environment by using low cost hardware without disturbing the driver? To find the answer to this question a number of sub problems are posed:

- What is the perception processor, how does it work and what problems can be solved by it?
- In what domains can the perception processor be an added value?
- Design and Implement an easy to use interface for interacting with the perception processor.
- Create an easy to use tool to monitor the eye blink rate while the perception processor is running.
- Can the perception processor detect and track, in real time, the human face within a sequence of images?
- Is it possible to extract the location of the pupils in real time using the perception processor?
- What is the optimal way to find and track in real time the face, eye locations and pupils using the perception processor within sequence of images?
- Do the extracted features provide sufficient and reproducible data to confidently determine the blink rate?

The sub problems will lead to clues as how to answer the overall problem of this thesis, namely, how will the perception processor behave in a real life environment using low cost hardware to solve the blink detection problem.

Problem Definition:

"How well can the perception processor perform in the domain of blink detection inside a car environment by using low cost hardware without disturbing the driver?"

1.3 Societal relevance

Several studies have shown that driver inattention or micro sleeps behind the steering wheel is one of the main causes for accidents on roads [1][2][3]. By enhancing the capabilities of the new generation of navigation systems, the amount of distraction for the drivers is increased as well. Other research has shown as well, that a persons blink rate directly relates to a persons alertness level [4]. By using new technologies available to manufacturers it might be possible to reverse the trend of increased inattention into a tool to prevent distraction. Not only for drivers on the road, but all systems depending on real time detecting and finding objects, the processor could prove to be an added value.

1.4 Methodology

To optimally create and test the application running on the processor, the following steps have been executed:

Table 1: Actions performed for this thesis.

Duration	Action	Goal
2 months	Literature Study	Gain knowledge about Face and Blink Detection methods
1 month	Learn SDK Simulator	Learn to use the concepts involved in working with the processor
1 month	Perform SDK tests	Test performance and techniques to create working applications
4 months	Design & Implementation	Create an easy to use interface for adjusting the processor settings and monitoring the output
1 month	Data Acquisition	Acquire data for measuring the processor performance
4 months	Analyze Performance	Perform the analysis to give a final judgment about the processor performance.

1.5 Project challenges

The main challenge within this thesis is to create a well performing prototype that is capable of real time blink extraction under different lighting conditions and on different people. Since the proposed solution should be a relatively low cost solution, no high speed or infra red cameras will be used. Furthermore, a solution without any wearables is preferred since we don't want to trouble the driver with such distractions. By using those kinds of hardware, the overall performance can probably be improved.

1.6 Outline

First of all, a summary of related work in the field of Face and Blink detection is depicted in Chapter 2 Related work. After a brief review, a description of the processor itself along with terminology involved when working with the processor is given in Chapter 3 Perception processor. Chapter 4 Model & Approach, gives a description about the approach taken to tackle and solve the problem stated before. Chapter 5 Implementation will explain all the details of the chosen implementation for the Blink Detector. The tests performed during this Thesis are explained in Chapter 6 Experiments and the results of those tests will be presented in Chapter 7 Results. Finally, a conclusion to the posed questions is given in Chapter 8 Conclusion, followed by a series of recommendations for future work in 9 Future work.

2 Related work

Summary. *A brief description of related work done in the fields of Blink- and Drowsiness Detection.*

Much work has been done trying to create fully automated, real-time systems that are able to classify faces, and emotions. Within this thesis, focus lies on detecting blinks as to get an indication for driver distraction. The problem of blink detection can be tackled in many different ways, but nearly all of them sub divide the problem into face detection, eye/pupil detection and finally blink detection.

2.1 Face detection

Approaches to detect faces within images or sequences of images are not a novel idea. Ever since the arrival of computers, people have been trying to detect and track faces on digital images. One of the best known solutions to face detection is the Viola Jones algorithm [5]. This techniques uses a tree hierarchy to quickly scan an image for specifically designed features. Implemented on a conventional desktop face detection proceeds at 15 frames per second. Many improvements have been proposed and implemented for the algorithm, resulting in techniques with even higher frame rates. The Viola Jones algorithm is so wide spread that it even made it's way into Intel's OpenCV library [6] for Computer Vision as a standard implementation for face detection. Figure 1 shows the result of the standard openCV solution for eye detection. Although this method provides good results, the Face Detection problem is far from solved. Phenomena like occlusion, head posture, position and facial hair can still be difficult to overcome.

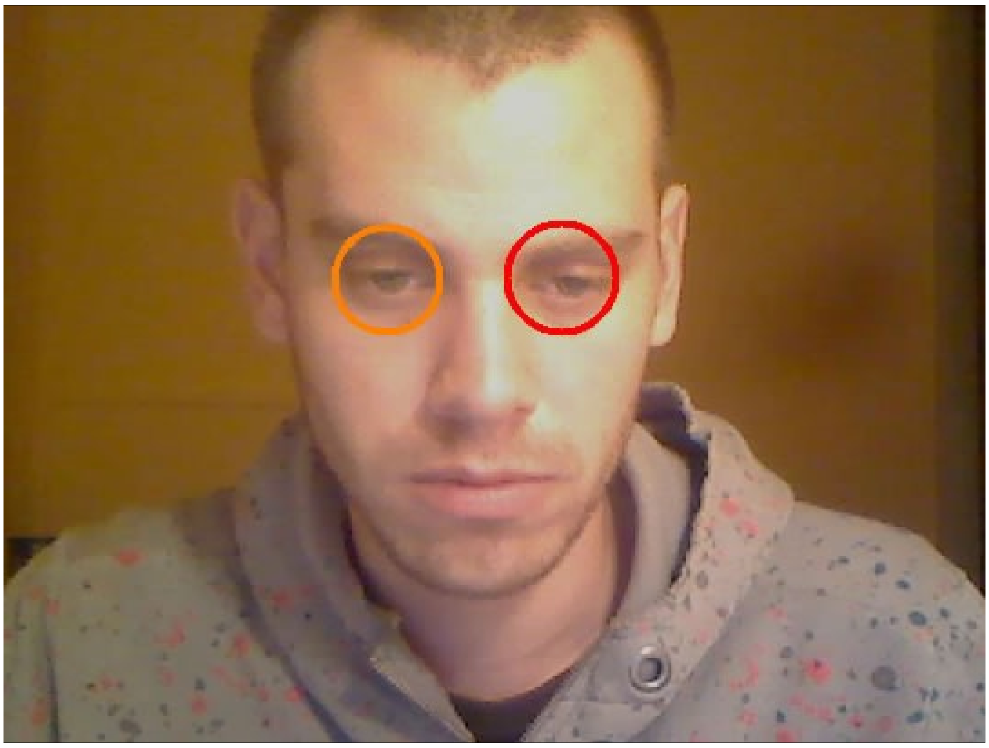


Figure 1: Result of using the standard implementation of the openCV library using haar cascades for detecting the eyes.

Still, other solutions exist. Feraud et al, 2001 [7] use Neural Networks and Rajagopalan et al. 1998 [8] use Higher order statistics. So far, the Viola Jones detector seems to be most popular. Some critics argue that when used on a processor, such a detector would eliminate the need for the perception processor. But, again, stressed is that such a processor would be limited towards face recognition versus the flexibility of the perception processor. Those kinds of processors can be found in modern photo camera's.

2.2 Eye detection

Some research for eye detection has been devoted towards using special lighting conditions to extract the pupils. Most efforts use multiple light sources [9], near infra red lighting conditions [10], or actual infrared illumination techniques [11] making use of the unique intensity distribution or shape of the eye. With the constraint of using a regular camera, these options are not suitable for this thesis. Other research focuses on traditional image based methods for eye detection. These methods can roughly be classified into three categories:

- Template Based
- Appearance Based
- Feature Based.

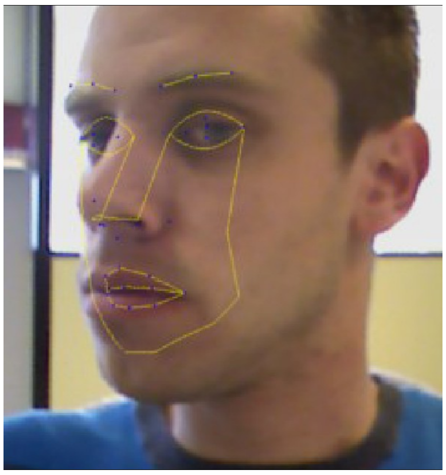


Figure 2a: Results of Face API.

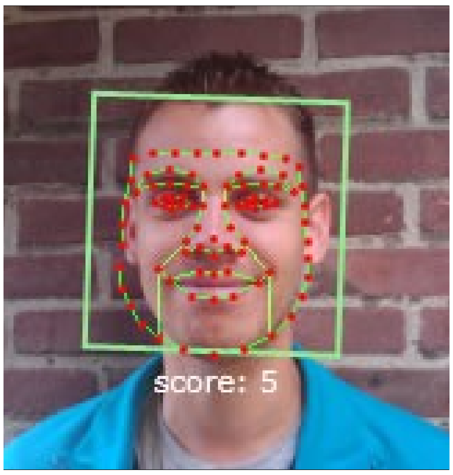


Figure 2b: Results of Beta Face.

Template based methods usually make use of a pre-defined model based on the spatial characteristics of the eyes [12][13]. Template matching is then used to determine the location of the eyes. A common addition is the use of Hough transforms, as proposed by Nixon [14], to enhance the fitting of the model onto the eyes. Appearance based methods try to detect the eyes by looking at the visual appearance of the eyes rather than the spatial characteristics [15][16]. These techniques usually require large training sets of data to train neural networks, vector machines or other learning algorithms to perform well. Extensions to this approach include the use of EigenFaces [15], wavelets and sophisticated classifiers such as the RBF NN classifier [16]. The Feature based approach basically looks at anything that is discriminative for a specific feature. This includes edge orientation, intensity and color distributions [17][18][19][20]. Kwatao et al. do not use the eyes themselves. Instead, features between the two eyes are used to derive the eyes location from [21]. Tian et al. [22], proposed a new technique by using a modified version of the well known Lucas-Kanade algorithm [23]. Once the eyes have successfully been detected, a common approach is to track the eyes using a Kalman filter, which essentially uses physical laws, dynamic models and sensor measurement to form an estimate of the system's varying quantities.

2.3 Fatigue & Blinks

One of the interesting things that can be achieved by using the before mentioned techniques is to apply the techniques for usage within practical applications. One particular interesting topic for a company creating navigation systems, is to create a drowsiness detection system. A lot of effort has gone into creating fully automatic drowsiness detectors. The main problem in classifying drowsiness is how to measure the entity. EEG scanners can monitor brain waves which can give an indication of the level of fatigue. Different researchers have different hypothesis as to what is the indicator for drowsiness. Some argue that the disappearance of alpha activity in favor of Delta activity is viewed as crucial [24][25]. While others mark the increase of power in the Alpha-Theta band as important [26][27]. While trying to find the exact discriminator for fatigue, others have moved on using other techniques, while EEG scans are not very practical to apply in field situations. Those other efforts mainly focus on oculomotoric parameters.

2.3.1 Blinks

Blink frequency and duration remain the best examined oculomotoric indicators for current alertness state and and drivers ability to react to environmental stimuli [28][29]. It is well known that fatigue is associated with increased blink frequency. Blink an increased blink rate can also be an indicator for increased emotional activity. Thus, a lot of research is devoted to extract the meaning of increased blinks, blink duration and saccades. Hargutt [28], concluded that a state of light fatigue was indicated by an increased blink rate alone whereas the transition from severe sleepiness was accompanied by increased blink duration. Some argue, that blink frequency and blink duration have to be considered as two independent factors in blink behavior. A lid closure that lasts for more than 500 milliseconds and covers the pupil for that period is usually described as a micro sleep. However, lid closure alone is insufficient to detect severe sleepiness for all people.

2.4 Commercial solutions

Face API is a tool designed by SeeingMachines [30]. It can be used to track faces in real time. It uses the feature points of a 3d model, to find and track faces in real time.

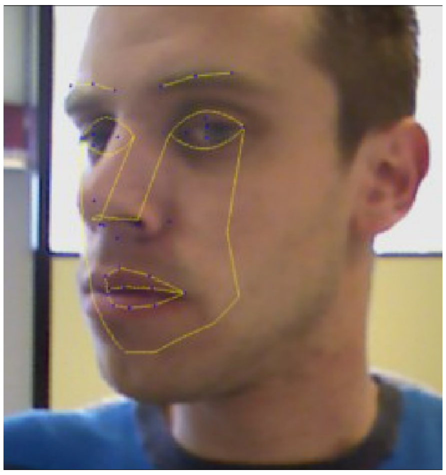


Figure 3a: Results of Face API.

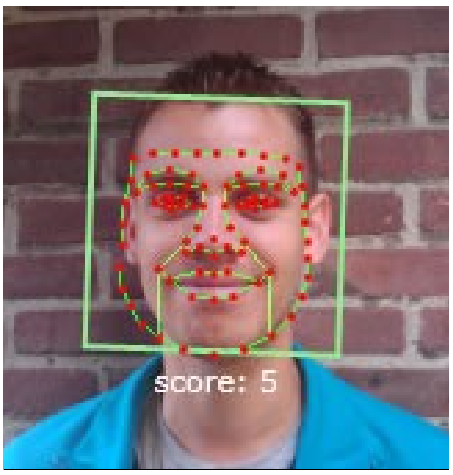


Figure 2b: Results of Beta Face.

FaceAPI provides a robust solution and can even deal with rotation and occlusion to some extend. Beta Face [31] is another commercial product that can be used for finding and tracking faces. Although both products provide real good solutions, they only offer a solution to a specific problem, namely Face Detection. Trying to perform other tasks, such as blink detection would be possible, but requires a completely new approach. Resulting in yet another round of optimizations and tweaking of parameters for optimal performance. The potential strength of the processor approach is that the processor is configurable and can be used for a wide variety of tasks. Not all as good and robust as the specialized solutions, but it hopefully good enough to be used to tackle real life problems.

2.5 Literature Study

The perception processor is designed to focus on perception rather than image processing. The idea is that, the more one perceives, more the understanding is easy. A literature study has been done to learn all the ideas and concepts behind the perception processor, as well as related techniques and other solutions. The results are presented within the literature study [32]. Upon finding the best approach for blink detection, several preliminary tests have been performed (see 4 Model & Approach). Goal is to find a robust method for detecting eye blinks in real time. Once a robust method has been developed, the prototypes can be used to ultimately create a drowsiness indicator for drivers within a car environment. This final step however, is not part of this thesis. Once more details are known between the relation between blink frequency and duration and saccadic speed, an first step towards a drowsiness detector can be created.

Other areas of research go even further. Some research is focussed on Emotion detection, in which, face and eye detection are a sub problem as well. An automated recognition system for facial expressions has been created by Pantic et al [33], but the eyes have to be identified manually. Datcu et al [34], have improved this technique further and combined the expert system with automatic localization of the eyes but has a frame rate of 5fps, indicating that it quit hard to do eye detection fully automatic and high frame rates.

2.6 Summary

A lot of work has been done in the different fields prior to blink detection as well as to blink detection itself. However, most solutions that are fast enough to be used for real time blink detection are optimized for specific tasks. Even when providing good results under different conditions, the proposed solutions can be used for one thing only. The biggest advantage of the processor would be that it is highly configurable and can be configured to solve problems in the areas of Face Detection, Blink Detection as well as Factory chain monitoring, Traffic monitoring and Robotics. This wide range of applicable situations is inherent to the flexible nature of the processor since it does not require extensive training. Off course, the processor can be used in conjunction with Artificial Intelligence techniques to provide more stable results, but the basic idea is that it should be able to perceive anything independent of color, size, shape or orientation.

3 Perception processor

Summary. *Concepts that originate from biology have been the inspiration for the perception processor. This chapter describes the underlying ideas, the schematics of the processor and finally describes how to use the processor. Both hardware and software configurations are discussed.*

The processor (Figure 4) used for this project is called: "Perception Processor". This name comes from the way the processor is designed. Tried is to mimic some important characteristics of the human visual system.

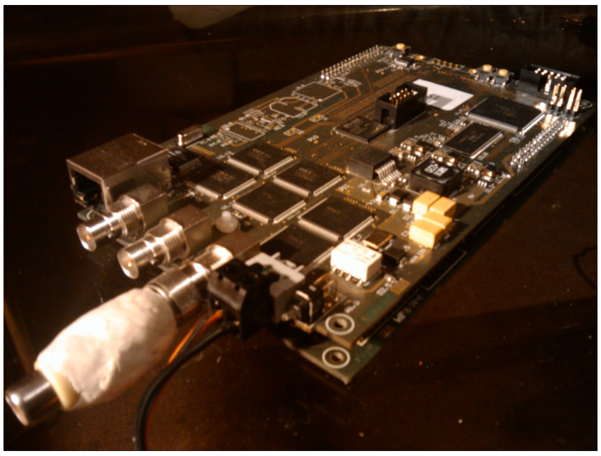


Figure 4: The Perception Processor is capable of delivering 100 frames per second.

The circuits try to capture the existing properties of the human visual system. This includes adapting to background lighting, tracking relative movements of objects, anticipating their movements and learning to detect certain objects. The circuits are inspired upon processing principles observed in populations of retinal and cortical neurons. Aim is not to get a complete neuronal implementation, which is very costly in electronic circuits, but rather capture as simple as possible the adaptive properties of the neural processes.

3.1 Hardware

To capture these adaptive properties, the processor consists of 16 individual cells. These cells can be configured to perform different tasks. For example, a single cell could be configured to select only pixels moving faster than a certain threshold. All pixels of the image that have moved within the set criteria are than selected. Through majority voting, the biggest group of those pixels are considered a group, and can be tracked over time. These groups are called Regions Of Interest or ROI. To track a ROI, a cell must:

- Select the pixels that meet the desired criteria
- Store the location of the ROI for later retrieval.

How this is done in the electronic circuits is described in the next section: 3.1.1 Schemes.

3.1.1 Schemes

Figure 5, shows the rough schematics of the processor. The sensor's output is split into sequences of images. One single image is considered to be part of the Spatial Domain. In the spatial domain, the processor can follow and track the locations Regions of Interest (ROI). The output of the spatial domain is send to a Spatial STN. At the same time, the sequencer stores images and cross references them with new images, to form the temporal domain. Within the temporal domain, parts of the image like colour, movements and orientation of edges can be extracted. The temporal output is send to a temporal STN.

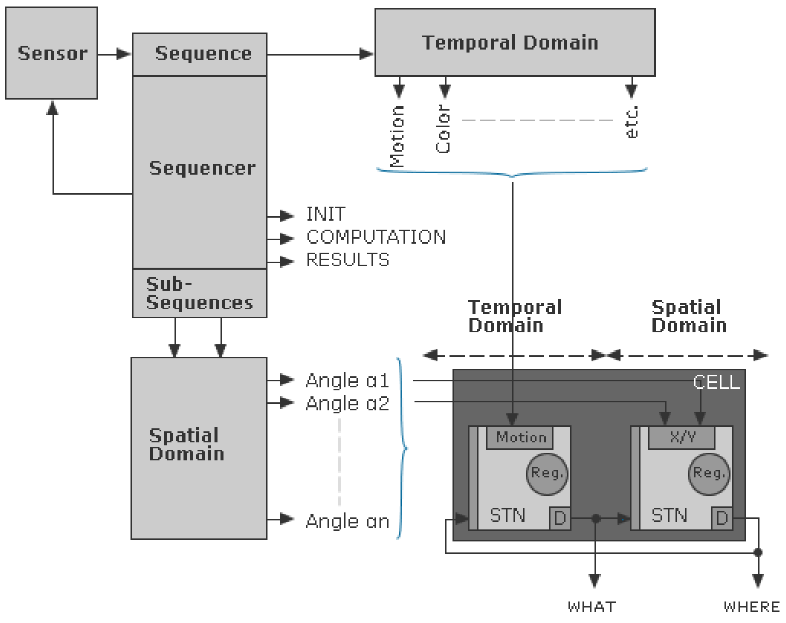


Figure 5: Sensor output is split into spatial and temporal cues, which are send to a cell. The cell itself consists of two separate STN's.

This structure allows to search within image sequences for all kinds of interesting parts and extract the location of those interesting parts. Details of a single STN can be seen in Figure 6.

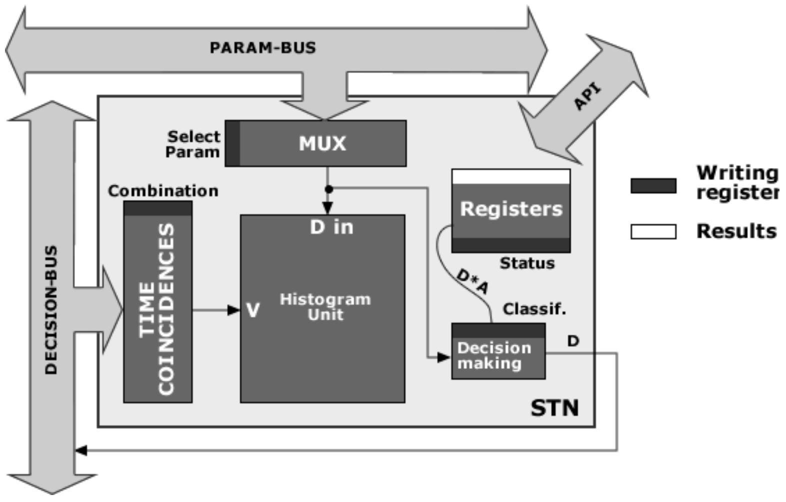


Figure 6: Schemes of a single STN.

The writing registers must be configured through the API, before the processor will produce some usable results. The results can be retrieved directly by the API as well.

Like mentioned before, the prototype used for the project consist of 16 of such cells. Each cell capable of performing statistical analysis in the temporal as well as the spatial domain.

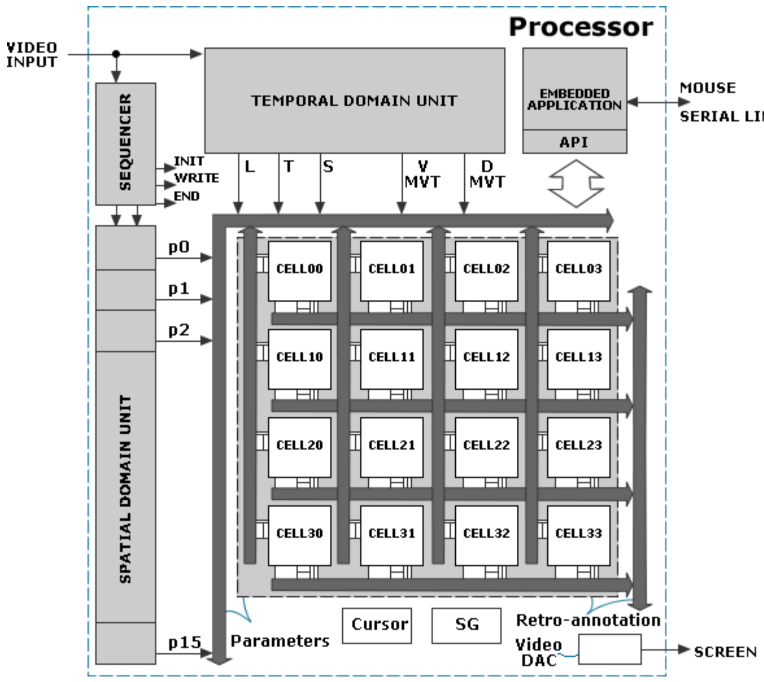


Figure 7: Schemes of the prototype processor, consisting of 16 individual cells than can be interconnected.

The complete processor layout is shown in Figure 7. As an input, a camera has to be connected to the input, and a UTP cable is required to write to and read from the cell registers. More on how to use the processor is described in the section: 3.1.2 Setup.

3.1.2 Setup

The input of the processor can be a camera, DVD player or even a web cam, as long as it is fitted with an analogue output. To read and write from the cell registers, a UTP cable is required, resulting in the configuration shown in Figure 8. Furthermore, a power supply is needed to power the processor. Throughout the project, different power supplies have been used.

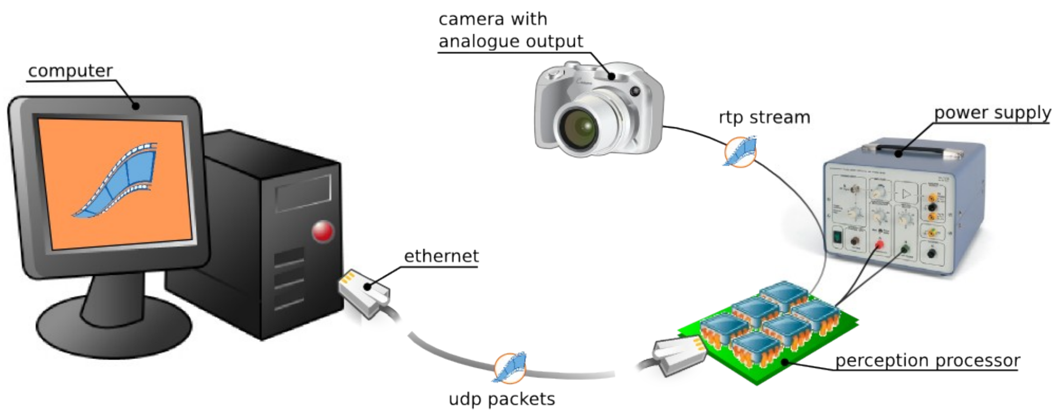


Figure 8: Required configuration to work with the perception processor.

3.2 Software

Now the hardware has been discussed, this section will focus on the software that is needed to get the processor to work. The first way to work with the processor is by using a simulator to get to know the concepts and the methodologies. The processor comes with an Software Development Kit (SDK) that can be used to gain understanding in the concepts used to work with the processor. Using the simulator is described in detail in: 3.2.1 Simulator & SDK .The simulator does not require to actually connect to the processor, while it is a pure software implementation. To really get started with the processor, one has to connect to the processor and send hexadecimal packets to configure the processor. Everything needed to do this is explained in: 3.2.2 Connect to the processor. Either way, all software is implemented for usage on Linux machines. Throughout the project, Ubuntu 8.10 Intrepid Ibex has been used.

3.2.1 Simulator & SDK

An SDK has been developed to simulate working with the processor. The SDK runs on Linux and is implemented in C++. The simulator can be used with the Eclipse IDE (Res. 1) in combination with the CDT plug-in. When running the simulator, typically a web cam is used as the input for the simulator. Additionally, the openCV (Res. 4) library is used for different functions in the simulator.

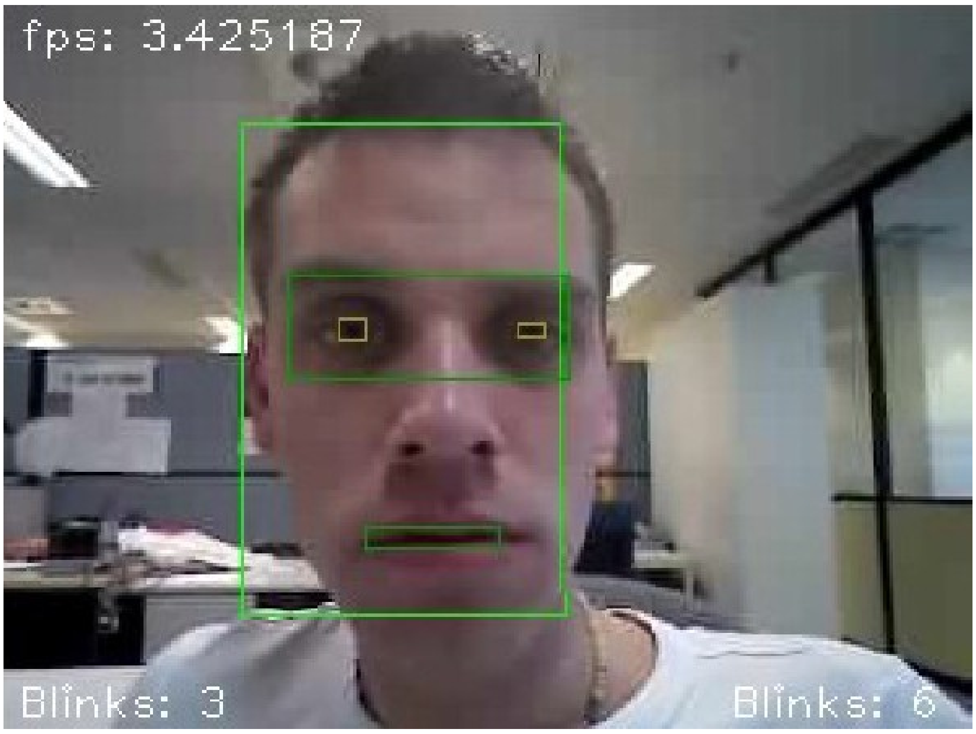


Figure 9: Early Test using the Simulator only.

This screen shot shows a simple blink detector running on the simulator. First the face is detected, within the face a new cell is searching for dark horizontal areas. Yet another cell is looking into the horizontal areas for small dark features. Based upon the height of the squares it is possible to extract blinks. This example shows a possible result of using the SDK simulator, but as can be seen the frame rate is around 3.5 fps, which is too slow to use in real time systems within a car environment.

An screen shot of one of the first applications created using the SDK is shown in Figure 9. It is an early blink detector using a single cell to detect skin color. Inside the skin rectangle is looked for big dark regions. This result in one area containing both eyes, and another region containing the mouth. Finally, inside the biggest of the two, yet two more cells look for the darkest areas. This configuration worked surprisingly well. However, since it runs on the SDK simulator, e.g. without the actual processor, the frame rate is very low (around 3.5 fps). As can be seen in the top of the figure.

3.2.2 Connect to the processor

After some preliminary tests with the simulator, it's time to really connect to the processor. This is completely different from working with the simulator in the sense that there is no need for an IDE. The only way to work with the processor is by sending hexadecimal packets to the processor, and catching the returned packets send from the processor (more on the communication in: 3.3 Communication). To accomplish this, NetCat (Res. 3) can be used on Linux systems. Besides sending and catching packets, the images send from the processor have to be shown using Mplayer (Res. 2). Mplayer is used to catch images from an Realtime Transport Protocol and display them on a computer screen. The images captured include the results of the processor. E.g. if a face classifier is used, the rectangular boundaries are included in the images.

3.3 Communication

The main advantage of the Perception Processor is that it allows for very fast computations. Many processors have been devoted towards executing a single predefined task. Whereas the perception processor is suitable for a much broader collection of applications. It is highly customizable since it can be configured at run time. Communication takes place through the use of User Datagram Protocol (UDP) packets. The UDP protocol has the advantage of causing less overhead compared to for example TCP protocol. One advantage of this protocol is that there is no need for hand shaking. This allows for less overhead, but can also be a disadvantage, since there is no guarantee that the packets will arrive at the destination.

3.3.1 UDP Packets

Communication with the perception processor occurs through the sending of UDP packets. UDP packets are very small in size and are often used when speed is of the essence. Since the perception processor has to respond in real time, UDP seems like the right protocol for the job. The protocol does not guarantee that the packets arrive at the destination. UDP packets are strings of hexadecimal numbers which represent data. All packets consist of a header, with some information about the packet, followed by the real data in the packet. Depending on the header, one can set criteria for classification, connect flows to cells, or connect multiple cells together. AW Wiki contains extensive documentation on which hexadecimal codes are related to what actions. Figure 10 shows roughly what the UDP protocol looks like. The top part in the figure is the UDP standard header. The lower part is a custom header for the processor itself and that is the part that has to be send to the board. A packet can vary in size from 50 bytes up to around 60 kilobytes, but on average packets do not exceed 512 bytes.

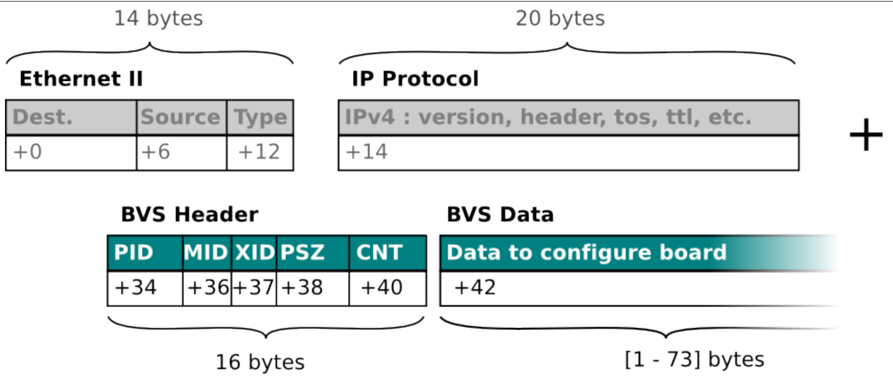


Figure 10: UDP Protocol structure to work with the perception processor.

Creating all packets needed to configure the perception processor for a specific task requires patience and alertness for the inexperienced programmer. To make life easier, a tool was developed to create, edit and send the packets to the perception processor. This allows to create packets without knowing the exact structure of the implemented protocol. More on this tool is to be found in chapter 5: Implementation.

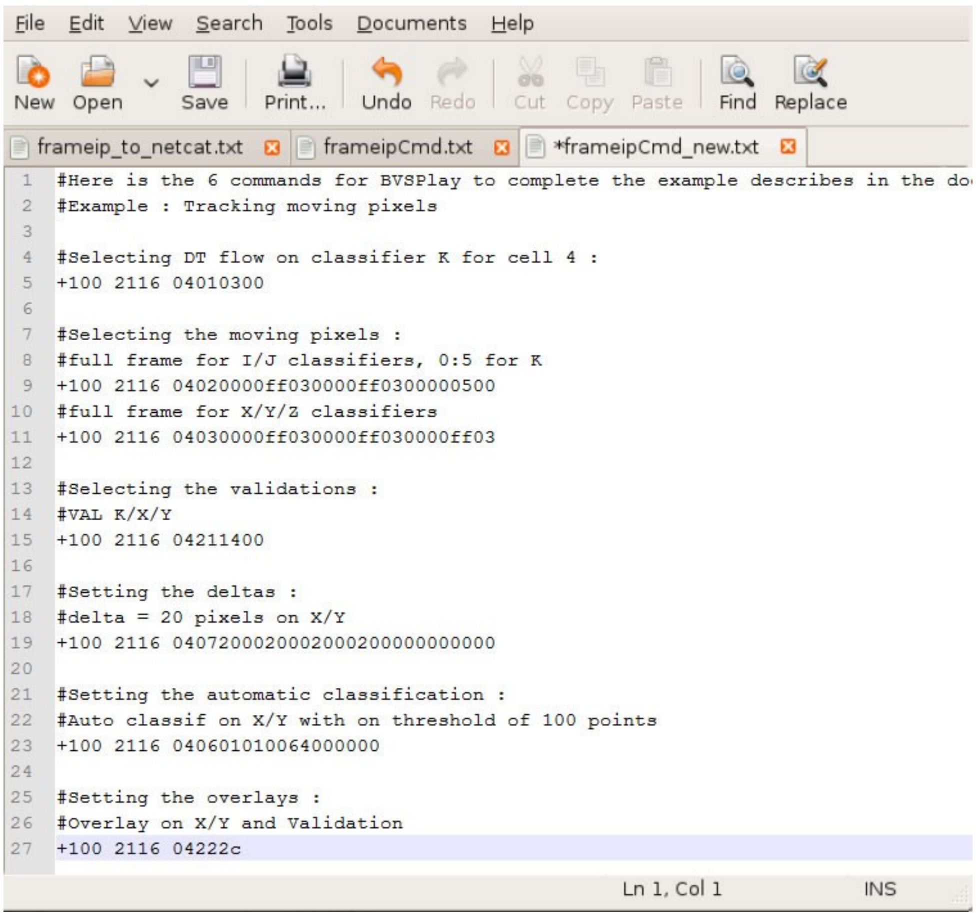
3.3.2 Speed

Using UDP packets to communicate with the perception processor results in very high frame rates. Frames are being send to the computer through an RTP stream and the frames are being send 100 times per second. However, the camera used for recordings operates at a speed of 25 fps. This means that all frames recorded by the camera are evaluated 4 times. Note, that this speed is the optimal performance and such speeds cannot be achieved during more complex operations (more details on this can be found in 3.6 Implications of interaction).

3.3.3 Sending configurations

To send the a specific configuration to the perception processor, initially a command line tool was available. Using a simple texteditor it is possible to write the packets and save them in a file. Using the command line, these packets can than be send to the perception processor. Figure 11, shows such a text file. This specific example enables the perception processor to find and track the biggest moving object in scene.

As can be seen in the example, the file consists of a couple of instructions. The commands consist of a general code `+100`, followed by a reference to a specific register `2166`, and finally the hexadecimal code for the value to write to the register, in the first case `04010300.` This file can than be send to the processor by the following command in the command line: `BVSplay <scriptfile> <destinationIP> <destinationPort>`.

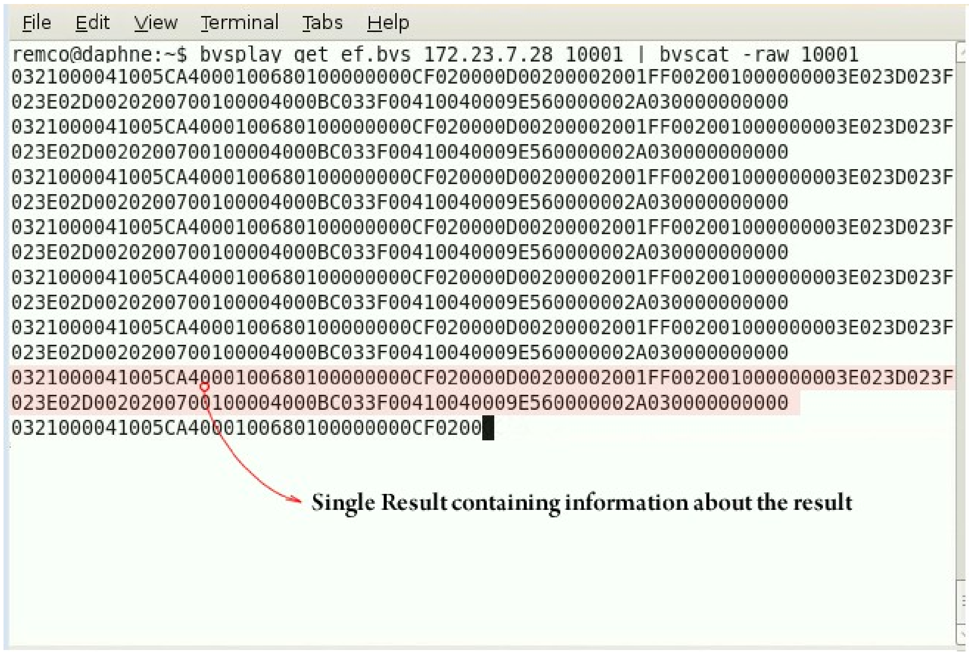

File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
frameip_to_netcat.txt
frameipCmd.txt
*frameipCmd_new.txt
Ln 1, Col 1
INS

```
#Here is the 6 commands for BVSPlay to complete the example describes in the do
#Example : Tracking moving pixels

#Selecting DT flow on classifier K for cell 4 :
+100 2116 04010300

#Selecting the moving pixels :
#full frame for I/J classifiers, 0:5 for K
+100 2116 04020000ff030000ff0300000500
#full frame for X/Y/Z classifiers
+100 2116 04030000ff030000ff030000ff03

#Selecting the validations :
#VAL K/X/Y
+100 2116 04211400

#Setting the deltas :
#delta = 20 pixels on X/Y
+100 2116 040720002000200020000000000

#Setting the automatic classification :
#Auto classif on X/Y with on threshold of 100 points
+100 2116 040601010064000000

#Setting the overlays :
#Overlay on X/Y and Validation
+100 2116 04222c
```

Figure 11: An example of a configuration to track the biggest object within a scene.

3.3.4 Catching results

After configuring the processor, results can be retrieved from the perception processor by sending a specific result packet. Once this configuration packet is send to the processor, the processor sends hexadecimal results back to the computer. An example of such a result is shown in Figure 12. This code includes information about what cell it belongs to, what configuration is used for that cell, as well as some statistical data about the results. E.g. searching for a face results in a packet that contains detailed information about the histograms.


File Edit View Terminal Tabs Help
Single Result containing information about the result

```
remco@daphne:~$ bvsplay get ef.bvs 172.23.7.28 10001 | bvscat -raw 10001
0321000041005CA40001006801000000000CF020000D00200002001FF002001000000003E023D023F
023E02D0020200700100004000BC033F00410040009E560000002A030000000000
0321000041005CA40001006801000000000CF020000D00200002001FF002001000000003E023D023F
023E02D0020200700100004000BC033F00410040009E560000002A030000000000
0321000041005CA40001006801000000000CF020000D00200002001FF002001000000003E023D023F
023E02D0020200700100004000BC033F00410040009E560000002A030000000000
0321000041005CA40001006801000000000CF020000D00200002001FF002001000000003E023D023F
023E02D0020200700100004000BC033F00410040009E560000002A030000000000
0321000041005CA40001006801000000000CF020000D00200002001FF002001000000003E023D023F
023E02D0020200700100004000BC033F00410040009E560000002A030000000000
0321000041005CA40001006801000000000CF020000D00200002001FF002001000000003E023D023F
023E02D0020200700100004000BC033F00410040009E560000002A030000000000
0321000041005CA40001006801000000000CF020000D00200002001FF002001000000003E023D023F
023E02D0020200700100004000BC033F00410040009E560000002A030000000000
0321000041005CA40001006801000000000CF0200
```

Figure 12: An example of the results being send from the board to the computer.

3.4 Theory & Concepts

Knowing the internal structure of the processor is one part. This section will focus extensively on the Theory & Concepts behind the processor. How it works as well as the limitations of the processor. One of the main concepts of the processor is Flow. Flows are transformations of an image, that contain a certain aspect of data. These flows can easily be obtained by performing different matrix transformations of the input image, transforming them into a specific domain.

3.4.1 Flows

Flows can, like images, be considered as a two dimensional matrix. A flow is the transformation of an image into another image (or matrix). Let *I* be an image whose dimensions are $m \times n$. *I* is constructed of RGB colour values :

$$I = \{I_r, I_g, I_b\} , \qquad (1)$$

where I_r, I_g and I_b are the red, green and blue components of the pixels of *I*. To convert *I* to the more useful Hue, Saturation, Value representation (Figure 13: Colour spaces. , shows the HSV colour space). Then *H*, *S* and *V* are defined as:

$$V(m,n)=max\{I_r(m,n),I_g(m,n),I_b(m,n)\} \tag{2}$$

$$S(m,n)=\begin{matrix}(V(m,n)-C(m,n))*255/V(m,n) & if\,V(m,n)!=0\\ 0 & else\end{matrix} \tag{3}$$

, where

$$C(m,n)=min\{I_r(m,n),I_g(m,n),I_b(m,n)\}$$

and,

$$H(m,n)=\begin{matrix}0+(I_g(m,n)-I_b(m,n))*60/S(m,n) & if\,V(m,n)=I_r(m,n)\\ 120+(I_b(m,n)-I_r(m,n))*60/S(m,n) & if\,V(m,n)=I_g(m,n)\\ 240+(I_r(m,n)-I_g(m,n))*60/S(m,n) & if\,V=I_b(m,n)\end{matrix} \tag{4}$$

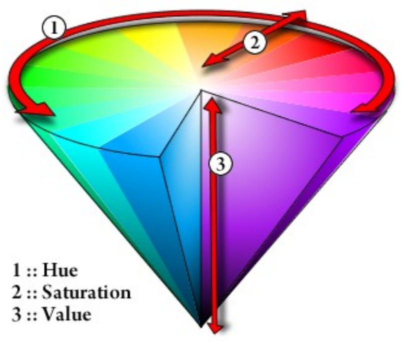
Then, any image *I*, can be transformed into their respective Hue or Saturation or Value representation by multiplying the original image by one of the defined transformations.

$$I_{hue}=I\cdot H \text{ , and } I_{saturation}=I\cdot S \text{ , and } I_{value}=I\cdot V \tag{5}$$

These transformations can be useful, because it allows to observe or perceive an image from a different perspective. Figure 14 shows an input image and the corresponding transformations to the HSV domains. The transformations (2),(3) and (4) operate on the colour domain. They represent the RGB channels of an image in a different way. Currently only transformations to the HSV colour space (Figure 13: Colour spaces.) are possible. While other colour spaces exist, the HSV space has been proven to be very useful in the field of face detection. Not only colour contains important information about the objects within an image. The structure of the objects, edges and contrast contain viable information as well. Other transformations exist to be able to extract different information about the objects within an image. All the different kind of transformations yield image representations suitable for different approaches. Below is a list of the four different categories of transformations currently available:

- Colour transformations
- Gradient transformations
- Movement transformations
- Spatial transformations.

Colour, gradient and movement transformations are known as Temporal Flows as opposed to Spatial Flows, that only deal with x and y locations of pixels.



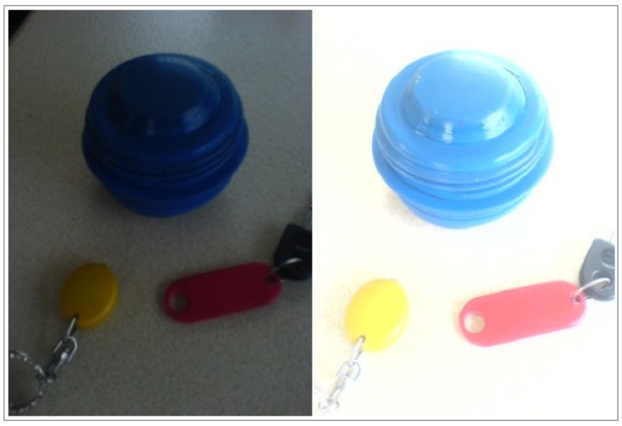


Figure 13: Colour spaces.

(Left) Hue is the angle between 0 and 360°, saturation is the radius and Value is the height of the cone. Note that when the value is zero, all colours are black and the Hue and Saturation values have no effect.

(Right) Lighting conditions affect the RGB colours of an image, the Hue and Saturation are maintained however. The difference in light can be seen in a change of the “Value” component. These two pictures show why it is easier to work with Hue and Saturation in face detection, since they are more or less fixed with changing lighting conditions.

3.4.2 Colour transformations (temporal)

Colour transformations can transform an image to HSV colour space. This transformation can be used to look for specific colours in HSV space. The “*Value*” is actually the Luminance representation. Figure 14 shows how a source image is transformed into the respective flows[2]. The numbers behind the explanation indicate the range of the values obtained after the transformation.

[2]Different colours can have different values. E.g. the colours in the Oriented edge indicate the orientation of an edge [0, 2555], while the colours for speed represent the values between [0,10].

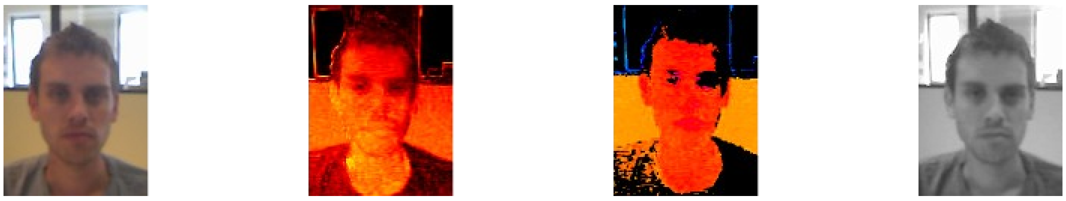


Figure 14: From left to right, The original input image and the corresponding transformations to Saturation [0,100], Hue [0,255] and Luminance [0,255] respectively. Different transformations reveal different information about the objects within the image.

3.4.3 Gradient transformations (temporal)

Gradient transformations allow to perceive texture. They can transform images into edge representation, oriented edge representations or curvature representations. This set of transformations allow for edge detection, and detection of specific shapes. Figure 15 shows the differences between those representations.

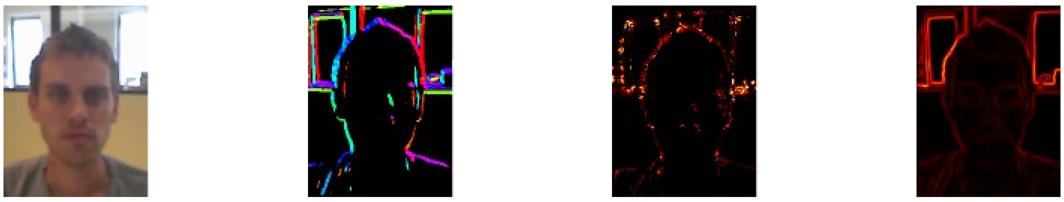


Figure 15: Source, Oriented Edge [0, 255], Curvature [0,255], Edge[0,255].

3.4.4 Movement Transformations (temporal)

Last but not least, a set of movement transformations are defined. The movement transformations take into account time, and they give information about the direction of movement as well as the speed of movement of all pixels. Additionally a

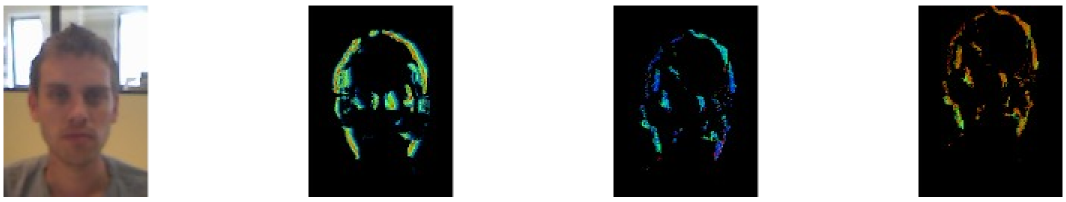


Figure 16: Source, variability of Luminance [0,7], Direction [0,8], Speed [0,10].

flow to measure temporal variability of luminance is included. Figure 16 shows how those representations look when applied to a source.

3.4.5 Spatial transformations

Additionally to the flows mentioned before, spatial flows are implemented. The spatial flows, as opposed to temporal flows, define transformations in the X and Y plane. In the sense that they are used to calculate the distance from the left and bottom of the image respectively. They are especially handy, when objects have been isolated from an image, and the location of the isolated object needs to be computed. The next section will discuss how to isolate objects using flows.

3.4.6 Classification

Classification is the process of setting criteria for pixels. And the goal is to separate the good from the bad. The result of classification is a set of pixels that meet the criteria given. Suppose we have want to classify our input image *I*, for hue values smaller than 200, then the resulting image $I_{hue<200}$ for pixel located at (*m,n*) is defined as:

$$I_{hue<200}(m,n) = \begin{matrix} 1 & , if\ H(m,n)<200 \\ 0 & , else \end{matrix} . \quad (6)$$

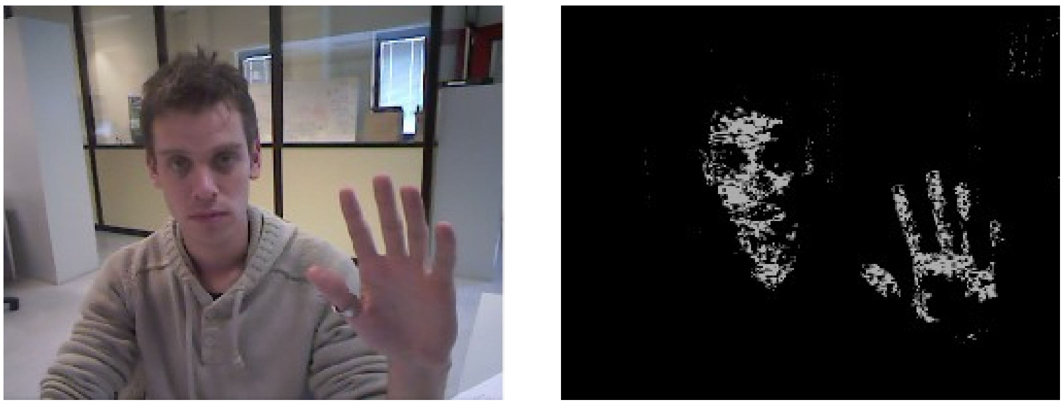


Figure 17: Classify the input image (left) on Hue values < 200. Result is on the right.

where H is the transformation to Hue values [Eq. (4)]. As an example, the result of this classification is shown on a given input image in Figure 17.

Once a flow is applied to the source image, setting classification criteria leads to an image where only pixels who met the criteria are visible. In terms of matrices this means that after classification one has a matrix filled with either 0's (indicating not meeting the criteria) or 1's (does meet the criteria). Since these matrices are usually sparse, calculations can be very fast and efficient [35].

3.4.7 Validation

Validation is the process of combining multiple classifications into a single representation. It is possible to invert the classified results before applying the validation process to it. This way, results can be re-used in different ways to make powerful combinations. Let A be the result of classifying pixels with Speed > 5 pixels / frame, X be the result of classifying pixels with x-location < 120, and Y be the result of classifying pixels with a y-location > 400 in an image whose dimensions are $m \times n$. Then the validated Image is calculated as followed:

$$I_{validated} = A_{speed>5} \cap X_{x<120} \cap Y_{y>120} . \tag{7}$$

In words, this expression means, look in the upper left corner of an image and find pixels that are moving with a speed higher than 5 pixels per frame.

Here's what the corresponding matrices *could* look like in an 4×4 image:

$$I_{validated} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cap \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \cap \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

3.4.8 Histograms

Histograms are another important feature. Histograms are a common way to visualize statistical data and to perform statistics on large data sets. Histograms are used to do statistical calculations. The way the calculations are performed can be configured. Figure 18, shows a histogram and the parameters and need to be configured. All histograms have a minimum value (*min*), a maximum value (*max*), a median, a highest peek (*posrmax*), the value of the highest peek (*rmax*) and the amount of energy of the function (*nbpix*). The *bval* parameter has to be specified enable partial selection of the histogram.

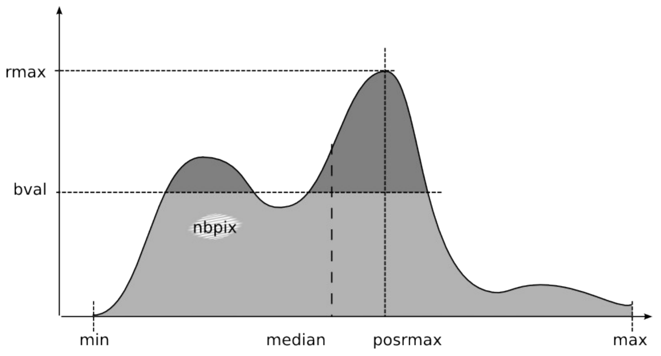


Figure 18: Histogram with main parameters.

Essentially, histograms offer additional tools to select certain pixels. They can be configured to focus on the highest peek, and disregard the rest of the histogram. This removes pixels from the image in the same fashion as classification does, but histograms parameters can directly be used in the classification for other flows. E.g. the boundaries of a histogram can be used as criteria for classifying another flow.

Figure 19 shows a classified image, where Hue and Saturation flows have been used to select those pixels in the range of Hue[0,200] and Saturation[50,150]. The Y projection of this image, is the Y histogram that you can see on the left most part of the image. The X histogram is a projection of the classified image (visible in the bottom of the image). The *bval* parameter is the one that has to be configured on the board, in the example it is defined as follows:

$$bval_x = \frac{rmax_x}{2} \text{ and } bval_y = \frac{rmax_y}{2}. \tag{9}$$

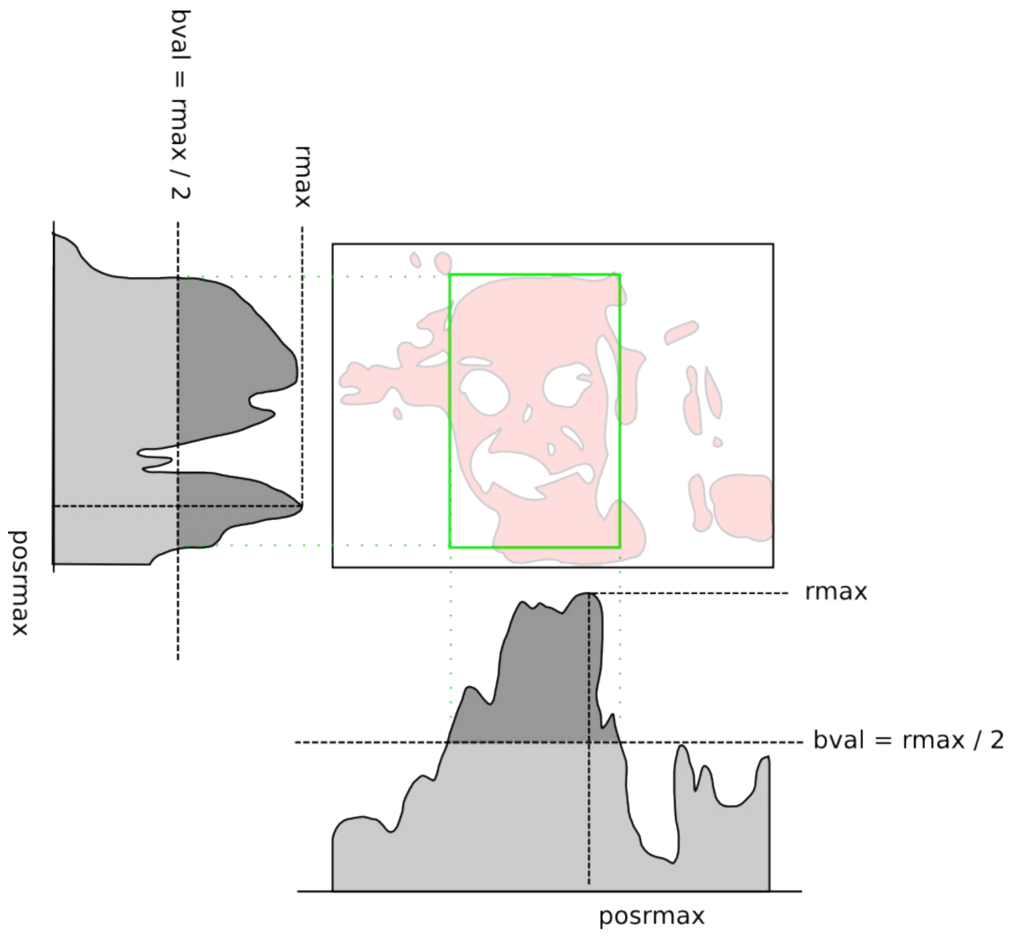


Figure 19: X and Y histograms as obtained from analyzing a classified image.

where $rmax_x, rmax_y$ are the value of the maximum peek in the X and Y histograms respectively. Now the new minimum and maximum values of the X histogram define the position and width of the selected pixels while the Y equivalent defines the height of the selecting pixels, forming a square of pixels.

3.4.9 Cells

All of the concepts mentioned are used within a single cell. Thus, one cells needs to be configured to use spatial flows and a temporal flow. The classification process is applied to the configured input flows and the resulting pixels will be used to calculate the histograms. The current processor version uses a total of 16 cells. The output of a single cell, can be used as the input of another cell as well. By using inhibition and cross validation, cells can be configured to look within the results of another cell. Figure 20 shows the layout of one such cell.

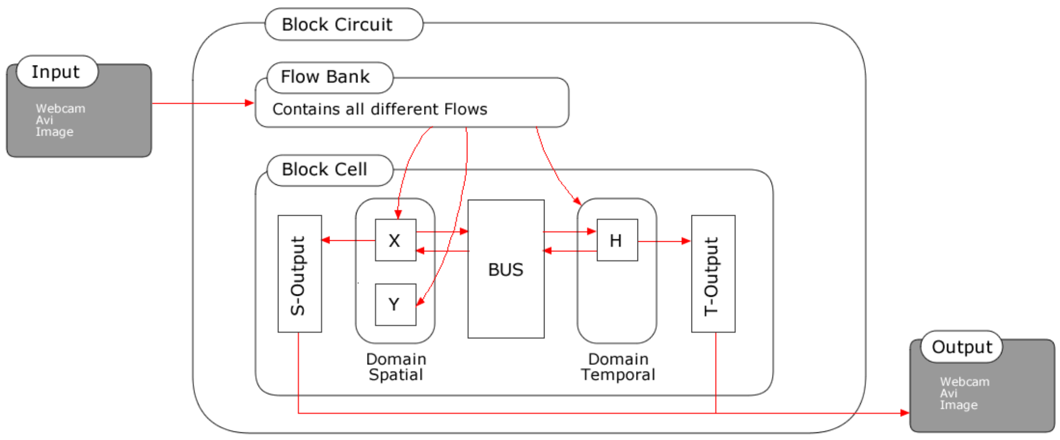


Figure 20: Layout of a single cell. A total of 16 cells are present on the perception processor.

This kind of layout can lead to hierarchical structures which can be used to search in depth for specific geometrical features like, for instance, the pupils within an eye, or teeth within a mouth region.

3.5 Interaction

Using all the before mentioned techniques, the processor can be configured for different tasks. Some of them can be used to create an independent configuration. For example searching for moving objects, one only has to send a few commands (see Figure 11). The processor will only select pixels that are moving more than the specified number of pixels. From this classified image, histograms are extracted. Those histograms are used to draw rectangular boundaries around the classified pixels, based upon the histogram configuration. The next frame, the boundaries are expanded, and the pixels inside the boundaries are classified again. This will reduce the area to search for pixels. This new expanded area will be used to calculate histograms again, to draw the new boundary. This process causes the perception processor to track and follow the moving object. Result registers are used to store the position of the classified region, as well as statistical data about the number of pixels inside the classified region. These result registers can be used directly by other cells, to search within that region. This direct usage of the result registers is the preferred way to connect multiple cells together like a chain, because it allow the processor to run at it's highest speed. This setup is shown in Figure 21.

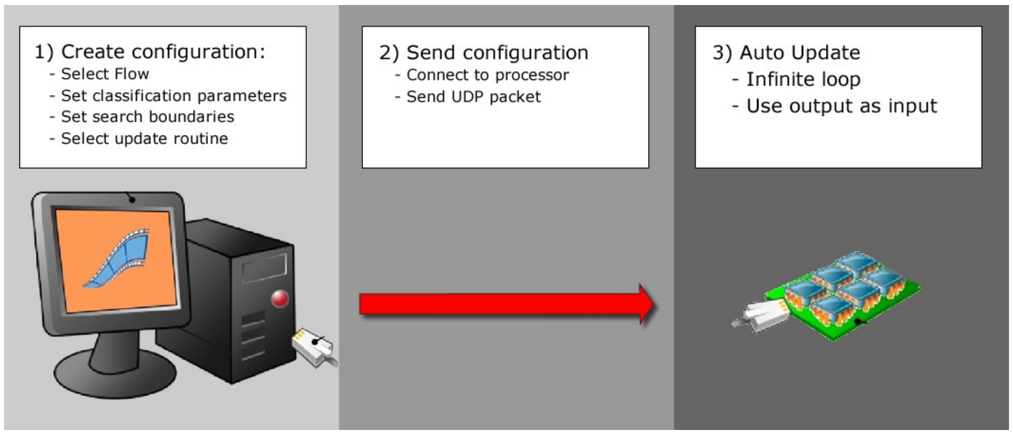


Figure 21: Stand alone setup. Requires a single configuration being send, where after the processor can run stand-alone. This can only be achieved when the result registers do not need to be evaluated for further analysis.

If, however, the result registers cannot be used directly, it is possible to retrieve the results, and perform additional calculations on those results. Followed by sending new parameters to the processor. This can be the case when you need to interpret the results of a cell and take appropriate follow up actions. Think about situations where you want to validate a result. For example, when a face is found we want to start looking for the eyes. However, it might be possible that the region classified as a face, is actually a window. Without a check, we would look for the eyes within the falsely classified window yielding false results. If we where to check the location and dimensions of the classified region, we might conclude that indeed we are dealing with a falsely classified region, and start looking for the actual face elsewhere. Thus we need to check to location and dimension and send updated parameters based upon the whether or not the classified region seems valid. An example of such a setup can be seen in Figure 22.

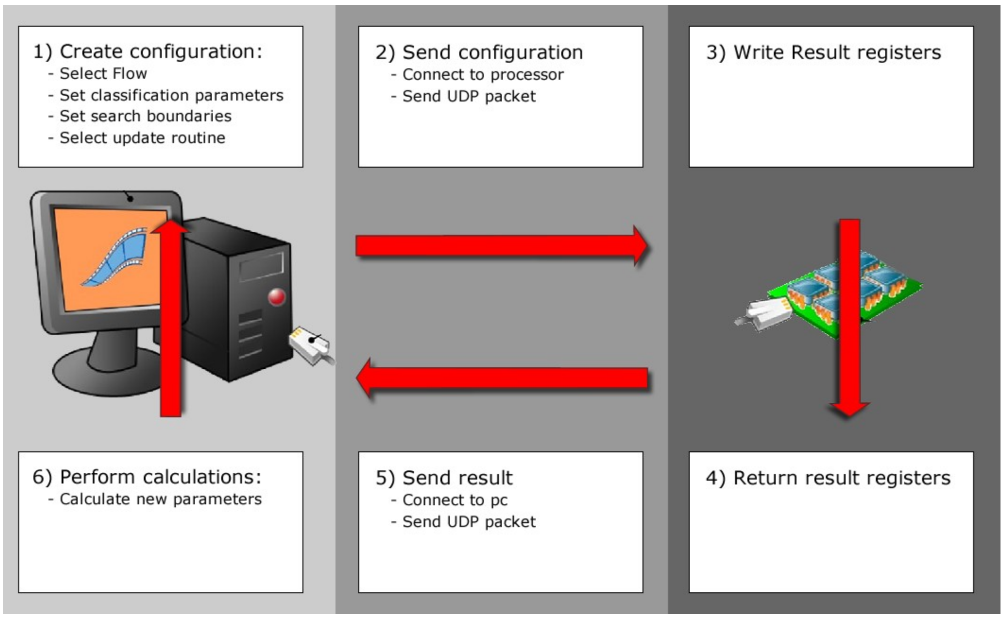


Figure 22: Intervention setup: The results are being retrieved, and based upon some new calculations appropriate actions can be taken.

This second setup, reduces a lot of the performance of an application. Since the time to retrieve the results, perform the calculations, and send the new parameters back takes a significant amount of time, this slows down the overall performance of the processor.

3.6 Implications of interaction

There are some downsides to the setup where the results are being retrieved, used for calculations and then being resend. The main disadvantage of this technique is that it affects the operational speed by intervening in the processors work flow by constantly sending newly calculated configurations.

Once the board is configured to perform a specific tasks, results are send back from the board at a maximum of 100 times per second, *c*. When those results need to be interpreted by some high level analysis system, the time left, *t*, available for the system without missing any frames is given by:

$$t = \frac{1}{f_{max}} - \frac{1}{c} . \qquad (10)$$

where f_{max} is the operational frame rate of the camera. And c is the upper bound of the perception processor (100 in our project). Additionally, this means that when the camera captures images with a frame rate equal of higher than c, some frames cannot be processed real time.

Thus, if the time of performing the calculations on the computer exceeds *t,* frames will not be processed. In the case of blink detection with a 25 fps camera, this is not acceptable. Ideally, as much logic as possible should be performed on the processor side. The current version of the processor used for this Thesis, was only a prototype, and therefore not all functions where implemented and/or available yet. Hence, sometimes we really need to interact with the processor in order to get stable results while taking the risk of loosing frames.

3.7 Conclusion

Concluding, the processor is not an image processor. All traditional algorithms used for optimizing images can be implemented on a compute running along side the processor. Standard techniques like increasing contrast, flood fills and noise cancellation are not part of the standard features of the perception processor. It uses matrix transformations instead to classify regions of transformed images, and uses Histograms to do statistical analysis on those regions. By combining tasks, a chain of cells can be connected to detect blinks within a sequence of images. Since the area within an image where blinks occur is very small, the suggested solution is to find the eyes before looking for blinks. With similar reasoning it is suggested to start by finding the face even before the eyes. A solution to finding blinks, hence consists out of a chain (see Figure 23). The main advantage of the processor is that it is very fast and that it is flexible for multiple purposes. We would like to find out what the limits are with respect to complexity and speed. Obviously there is a trade off between the complexity and the operational speed of the processor.

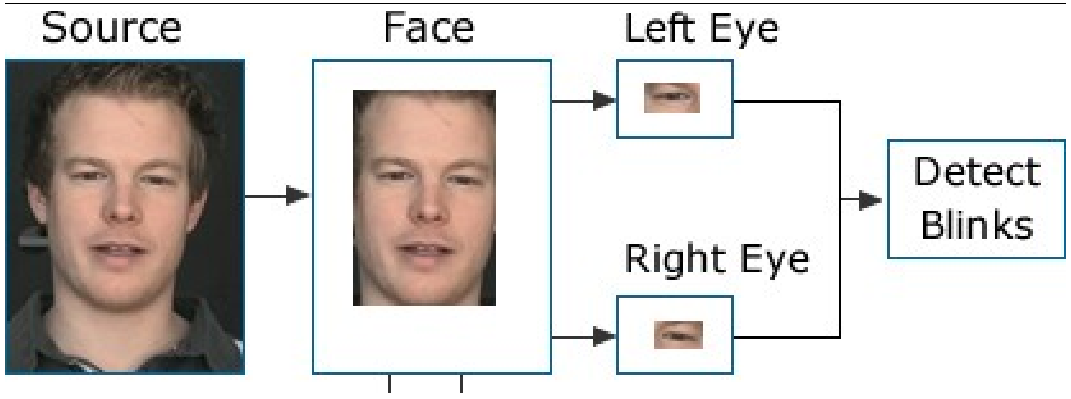


Figure 23: Chain for detecting blinks. First the face has to be detected. Next a search for the eyes is performed in the face portion. Finally, the width/height of the eyes will be used to determine whether or not a blink has occurred.

4 Model & Approach

Summary. *This chapter describes the architecture, design of the model as well as some preliminary tests performed to get an indication of which approach to take for the final set of tests.*

So far we discussed the underlying concepts of the perception processor as well as the terminology used. The ultimate goal is to create an application capable of detecting blinks within a car environment. Two specific reasons steer the project towards blink detection. First off all, blinks are directly related to fatigue [4], which by itself is a thread to safety on roads [1]. Since AW, as a producer of navigation systems with an increasing amount of functions, is itself also a part in the distraction problem, it is very interesting for AW to look at counter measures to increase safety. Basically, it is a very lively and interesting subject. Second, blinks are hard to detect. Pupils are relatively small, blinks occur very fast, and the lighting conditions change very rapidly within a car environment. Therefore, creating a blink detector generates a good understanding about how well the perception processor is able to perform under these conditions.

The problem of blink detection is divided into three sub problems. First of all, the face must be located. Once the location of the face is known, the eyes must be located. Finally the characteristics of the eyes must be extracted over time to see whether or not we are dealing with a blink.

4.1 Face detection

Face detection is the first step in our application. Different approaches have been used to create a robust face detector. Namely classification based upon color, shape and movements.

4.1.1 Introduction

Skin color is a very characteristic for most humans. Even though it comes in a wide variety of colors, skin portions are easy to detect for humans. One might think that skin color can occur in a wide variety of colors, but since the perception processor uses the HSV colour scheme (see Figure 13), looking at a combination of Hue and Saturation satisfies for most skin color. Earlier research, has shown that S+V/H provides a constant value for skin colour independent illumination [36].

4.1.2 Classification through color

It seems logical to detect faces based upon the HSV model. One of the problems using this method is that the perception processor currently does not have an option to combine input flows mathematically. Therefor it is currently not possible to

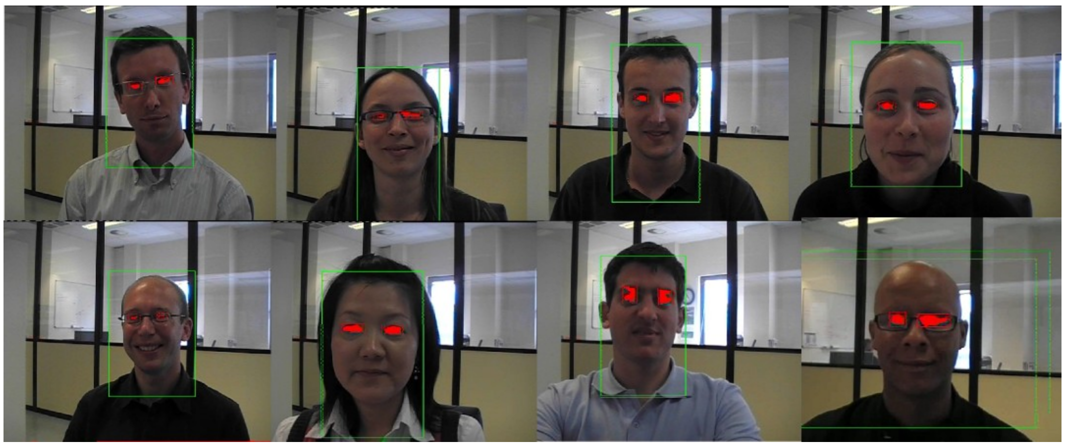


Figure 24: Results of Face Detection.

implement the formula S+V/H without a computer intervention. Hence, a combination of Saturation and Hue is used to detect portions of skin, making it sensitive to illumination changes. Early tests using this method provides the results shown in Figure 24. Note that all these images where taken under similar lighting conditions. The African male's face was not detected right, but the European and Asian faces are detected right.

4.1.3 Classification through shape

One might argue whether or not color can provide a stable classifier for skin portions. Due to the fact that color depends upon the lighting conditions, other methods have been studied as well. Like mentioned before, the processor currently does not provide an easy way for combining color information into a constant. A series of tests looking at a particular shape where done to see if shape could provide a more robust classifier. For these tests the Oriented Edge flow is used (see 3.4.3 Gradient transformations (temporal)). This classifier can look for edges in a specified orientation. Currently, the perception processor is based on rectangular structures, making it hard to detect circles or oval shaped occurrences.

4.1.4 Classification through movements

Another classifier could look for the biggest moving object within a scene. This idea would be build upon the assumption that the head would be the biggest moving object within a scene. Early tests clearly show that this assumption does not hold. Especially in the dynamic environment of the car, where background movements can occur. Even without the distracting background this method will not provide robust results. The classifier is triggered by movement, so whenever the person behind the steering wheel is not moving for some time, the classifier will loose the head. Hence, classification through movements seems not very suitable for a robust face detector.

4.1.5 Eye detection

Eye detection has proven to be the most prudent task in the chain. Since the eyes are quite small and look a like features exist in the surrounding of the face. Since a lot of similar features, like portions of hair, eyebrows and even the nostrils (see Figure 25, above), all share some of the characteristics of the eyes, there is need for an extra step to reduce the area in which to search for the eyes. A number of different approaches have been tested to study the effects. All approaches will be explained within the rest of this chapter.

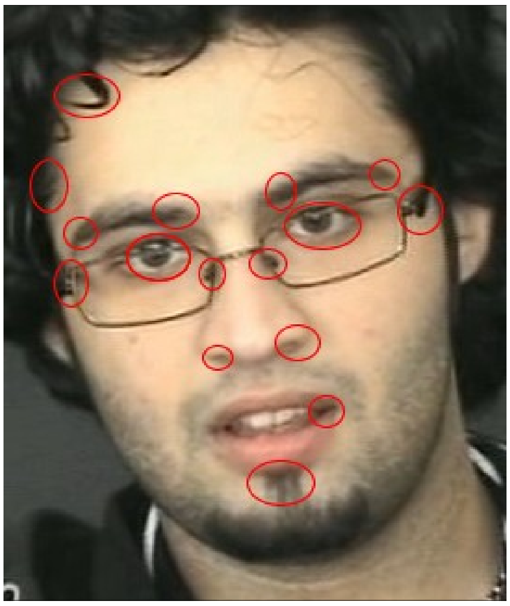


Figure 25: The difficulty in finding the eyes, a lot of eye-like features exist.

4.1.6 Approach I: Side rectangles

Instead of directly looking in the face for dark round features, an intermediate step is needed to prevent miss classification of the eyes. To be able to find the eyes, we assume that the area we found in the previous step is indeed a face. If it is a face, we have some knowledge about the geometry of the object. Since the current perception processor only deals with squares, we can check if the square we have found is indeed a face by looking at the projection histograms. We use this knowledge to find out if we indeed have a face, and at the same time reduce the potential area of where the eyes can be located. First a cell is positioned at the left boundary of the face. We classify all pixels that are the opposite of our face, and validate pixels that are between the maximum and half of the maximum. All columns of pixels, starting from the left, with more than half of the height of the face of non-

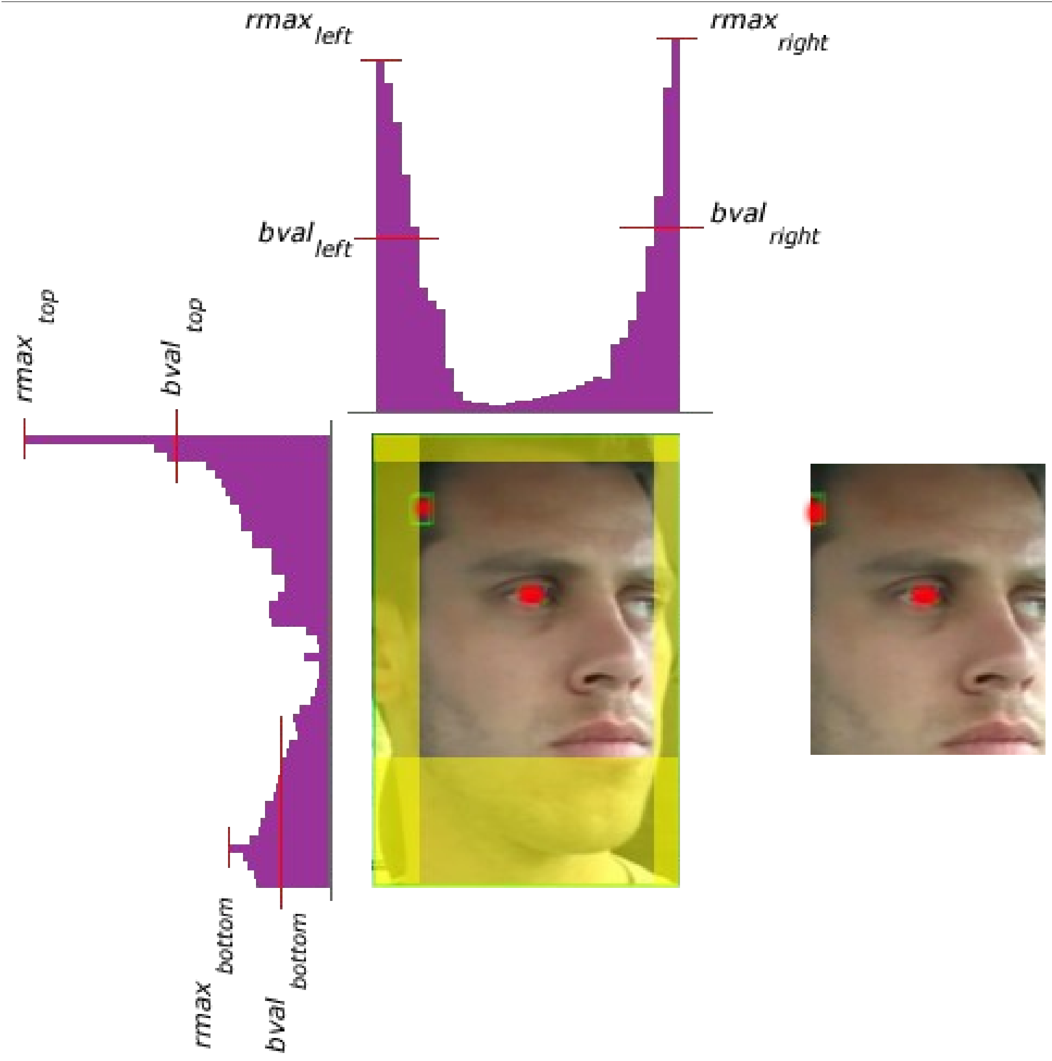


Figure 26: Errors can occur in situations where the head is rotated.

face colours will be removed. We repeat this process starting from the opposite side as well. After that, the process is repeated on the y projections, removing all rows from top and bottom which consists of mostly non-face pixels. A graphical explanation can be found in Figure 27. By excluding the four boundaries from the actual facial pixels, we get the reduced area where the eyes should be located, and which is clear from noise belonging to the hair of the subject. The advantage of this approach is that it requires no interaction with the processor. The boundaries of the face are used for the starting point of each of the four side rectangles. The four side rectangles remove portions of the side of the face, resulting in a smaller area for the potential eyes. On the other hand, this approach is not robust to head tilting. Once the head is tilted, the removal of the sides is not optimal, and the resulting region is not free of eye-shaped features. This in turn, can lead to false recognition of the eyes.

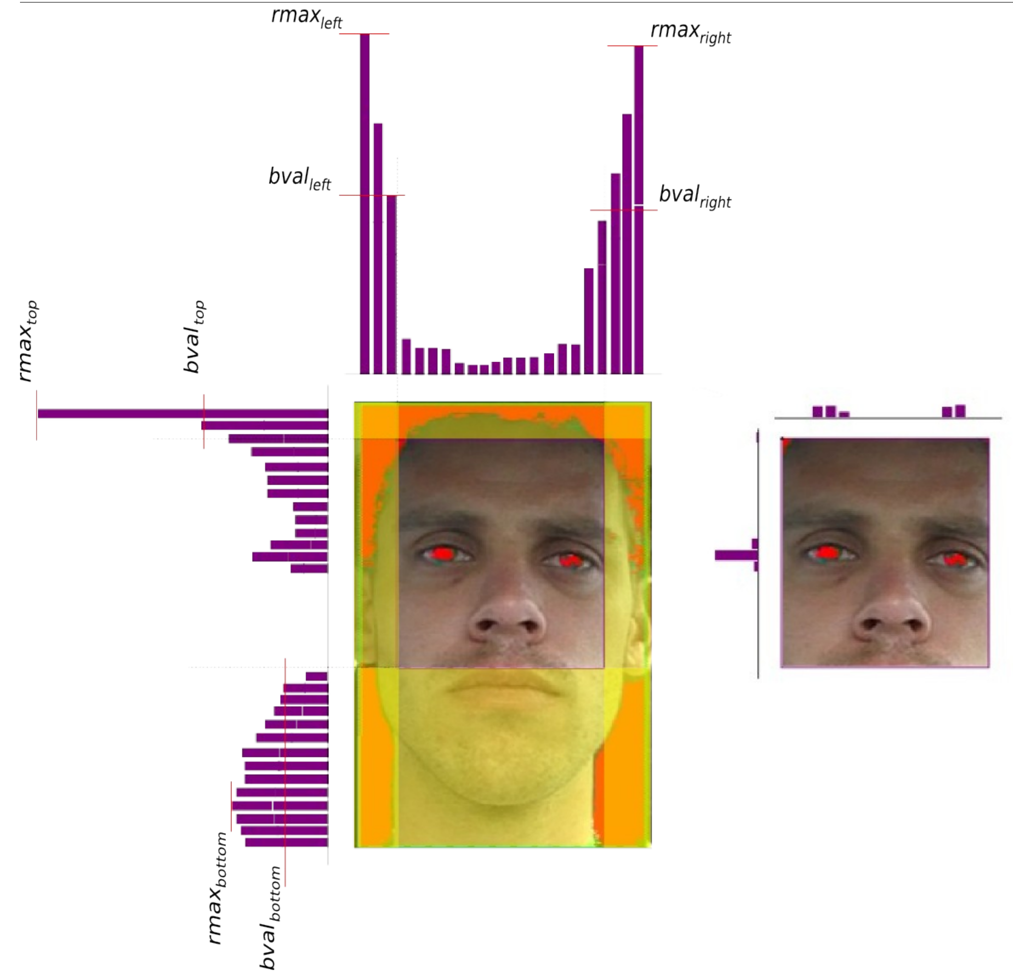


Figure 27: Eye Detection.
Due to the characteristic geometry of a face, we know what the histogram projections should look like. By using four cells to remove these non-face pixels, the resulting area is smaller, and will probably contain the eyes.

So this approach is very sensitive for head tilting/ rotation, but it requires no additional computational power on the computer side. Therefor, the processor does not have to wait for some computations on the computer to finish. This enhances the speed at which the processor runs.

4.1.7 Approach II: Prediction

Another approach to reduce the potential area for the eyes is by using the face in a different way. In this approach, the face is first detected. Instead of using the face

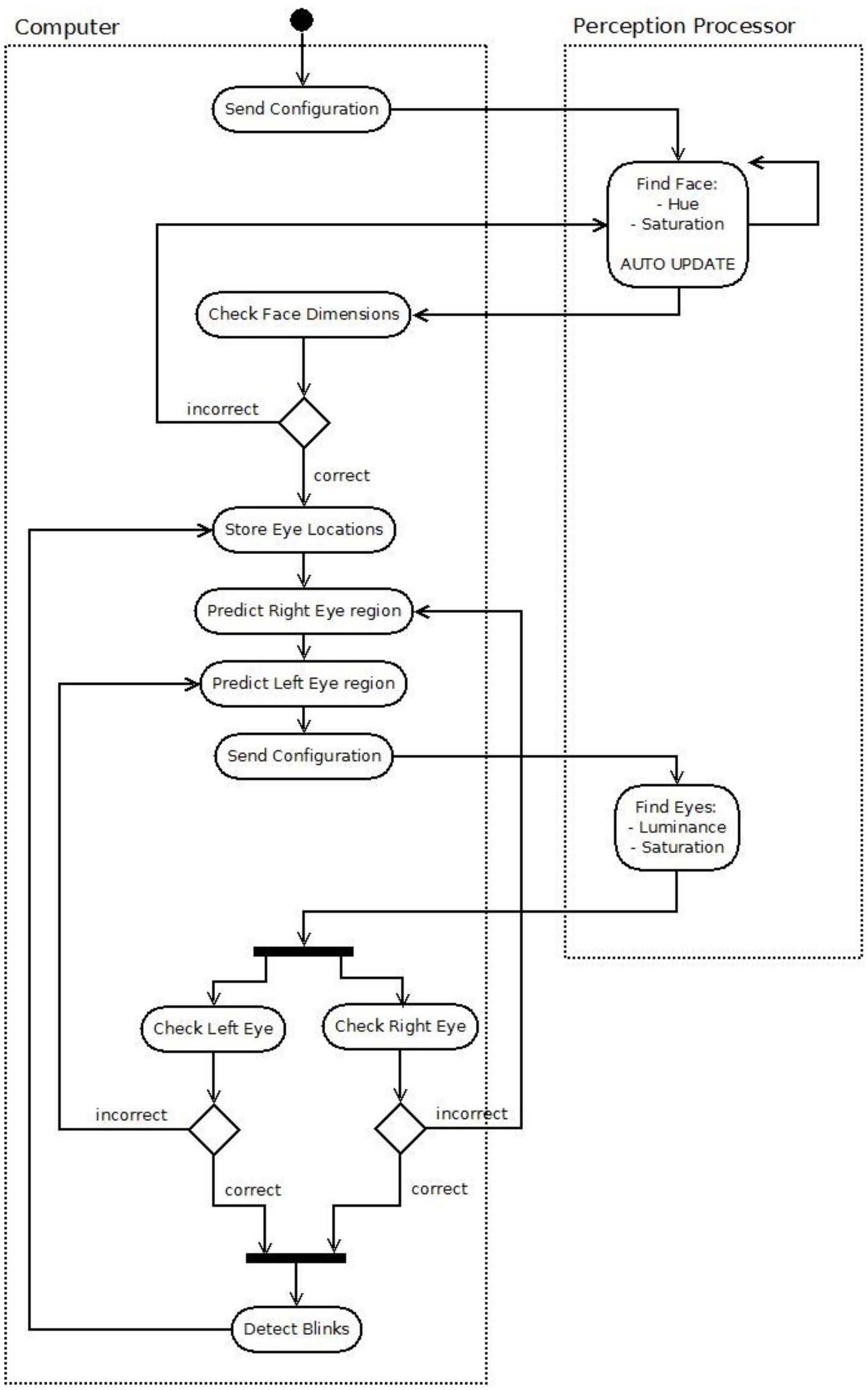


Figure 28: Activity diagram of the Prediction Approach.

boundaries directly, the region where the eyes will most likely can be found is predicted. Within this region, two eye like regions are being retrieved. If no eyes are found, we move our predicted region, and we try again.

The obvious advantage of this method is that it allows for searching within smaller potential regions, given less false positive results for the eyes. As a side effect, since the location of the eye region is calculated, we can store the location for future reference. If we are to find two eye like features, we can store those locations as well, giving the possibility to easily account for occlusion. The down side of this method is that some of the speed is lost, due to the complexity of the algorithm to predict the region.

4.2 Eye blink detection

After the step described in the previous section, the eyes can easily be located by looking at the two biggest regions that are not part of the skin. Taking a close look at the image shown in Figure 27, one might wonder why the two eyes are selected rather than the two nose holes, that have approximately the same shape, size and colour? It appears that, even though the nostrils seem to be the same colour, they have different hue. Note that the skin was selected by looking at a specific hue. Recall that Hue the value of a colour where it submitted to bright white light. By turning down the light, the appearance of the colour changes, but the base remains the same. The holes of the nose in fact are skin colour. They appear very dark, due to the lack of light. Yet, the eyes, are not skin colored. Problems do occur, when the eyebrows are very dark. Since they have a different hue from skin colour, and they also appear in the same region of the face, there exist situations where the eye brows are mistakenly recognized as being the pupils.

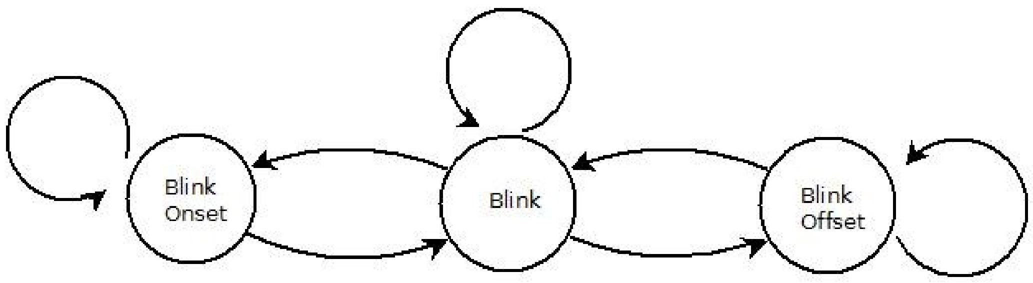


Figure 29: State diagram of a blink.

Assuming the pupils are detected correctly, the following method is used for detecting blinks. For every frame the height of the pupils is retrieved and stored. During a blink, when the eyes are completely closed, the pupils cannot be found anymore, since there is no dark shaped area within the eye region. At this time, we can assume that a blink is going on. To actually determine whether or not a blink is occurring, we look back in the stored heights, and if we see that the height of the pupil was getting smaller, we assume a blink actually is occurring. Furthermore, if the same pattern can be seen in the other pupil as well, we can safely conclude that a blink occurred. Figure 29, shows the state diagram for a blink. We use the onset of the blink to check if we actually have a blink, or whether we simply lost the location of the pupil.

4.3 Test case : Driving in a car

To test the different methods a simple test has been performed. Using a camera mounted inside an actual car, some recordings have been made. Given the fact that the application is for demonstration purposes only, the performance is good. Goal was to find out the best methods for blink detection and to get an idea of the usefulness of the processor in real time applications. The camera used for this test was a regular PAL Camera of 720 x 540 resolution at a frame rate of 25 fps. Figure 30 shows the recording setup which in this case yields pupil sizes of 10 x 5 pixels. The processor has proven to give promising results, even under these harsh conditions.

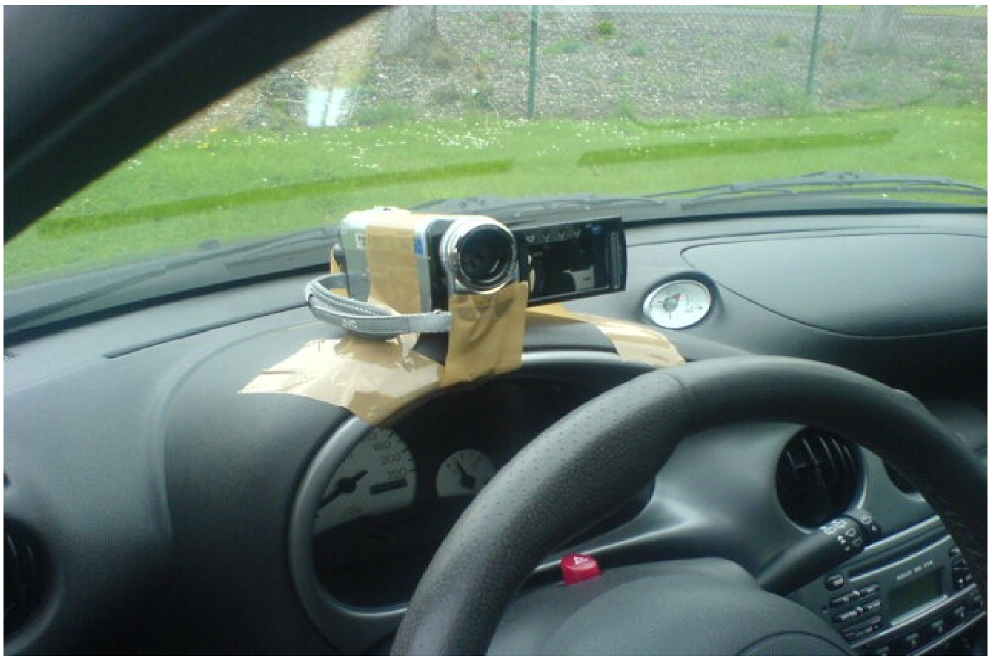


Figure 30: A camera (720 x 540 pixels) is mounted on the dashboard.

Sequences of images are recorded under poor conditions. Shaky images at middle resolution with varying lighting conditions and rapid moving backgrounds can still be used to produce valid results for the blink detector. Manual recordings with "little" head movement produce a 97% recognition rate for a sequence of 2 minutes,

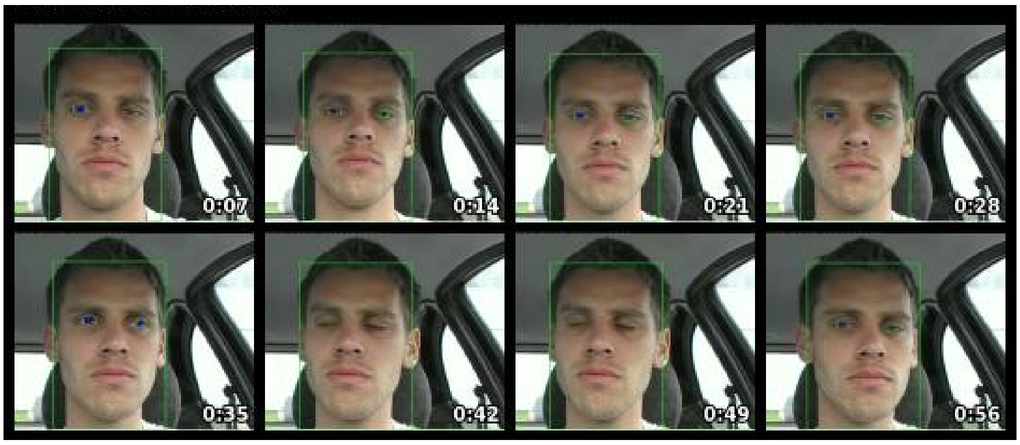


Figure 31: Result of the blink detector prototype. This clip with little head movement lasts approximately 1 minute, and has a correct recognition rate of around 97%.

while a 2 minute recording with “heavy” head movement provide a recognition rate of 82%. Some screen captures from a clip with little head movement is shown in .

4.4 Test case evaluation

From the simple test case we can learn which of the proposed methods for detecting respectively the Face, Eyes and Pupils will provide the most stable results. Though, one must keep in mind, this test case was performed on a single test subject only.

4.4.1 Errors using side rectangles

Some errors in classification occur in this first test. Especially using the Side Rectangles Method (described in 4.1.6 Approach I: Side rectangles) a lot of errors occur. Both images in the left of Figure 32 show a common error with this approach.

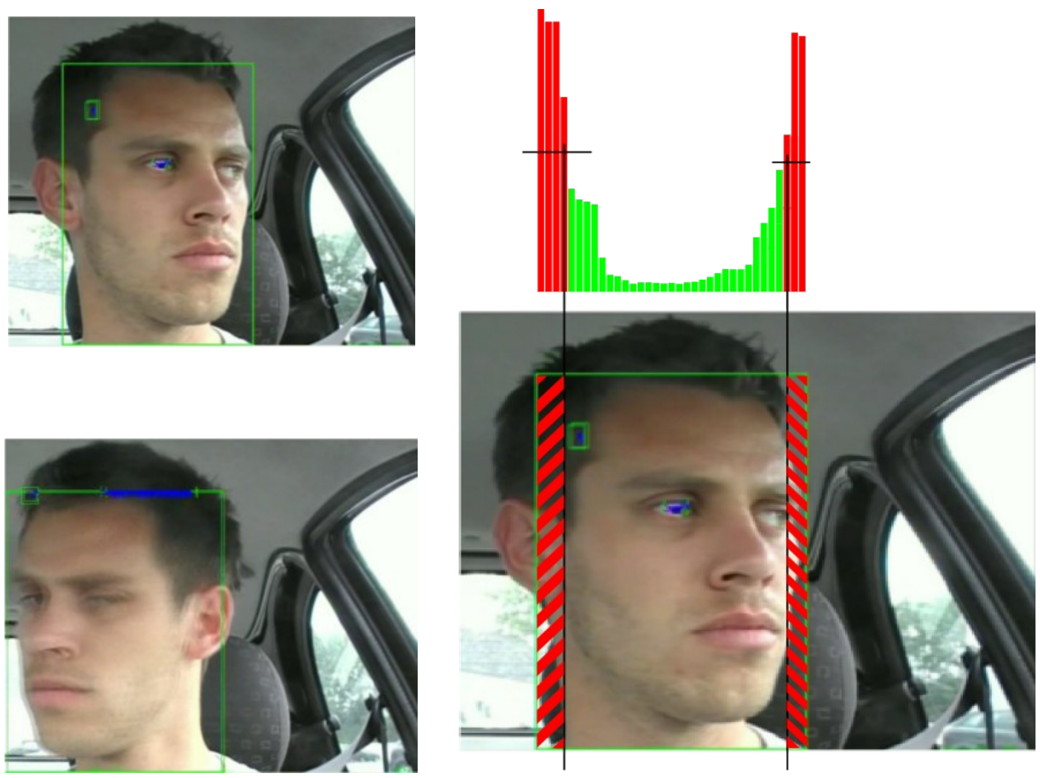


Figure 32: Common errors using the Side Rectangles approach. Left: Errors in classification. Right: The explanation of why this type of error occurs.

Reason for this miss classification is due to the *bval* parameter, which has to be send to the board. The current demonstration application uses Eq. (9) to determine to what extend portions from all sides of the face boundaries are removed. Currently this value is set to the height of the face divided by two (or half of the maximum). As can be seen in the image in the right side of Figure 32, the misclassified portion is outside the area removed (looking from the left). Furthermore, the hair is of a different hue than the skin, and the region is bigger than the pupils, making it the first candidate to be a pupil.

4.4.2 Test case conclusion

From the initial test case we learned that the best way to create a blink detector is to use the Color Method for face detection (See 4.1.2 Classification through color) and to use the predictive model for locating the eyes (4.1.7 Approach II: Prediction). As for determining the blinks themselves, we will use the model using the onset of a blink to double check if we observed a blink.

Detect	Approach	Recognition Rate	
		little movement	normal movement
Face	color	99.00%	99.00%
	shape	81.00%	56.00%
	movements	67.00%	86.00%

Detect	Approach	Recognition Rate	
		little movement	normal movement
Eye region	Side rectangles	85.00%	59.00%
	Prediction	87.00%	82.00%

Table 33: Results of the preliminary test to get an indication of the best approach for the eye blink detector.

Looking at the accuracy, we obtained a 87% recognition rate for the eyes when the test subject is hardly moving, and 82% recognition rate for the eyes when the driver is actually driving on a real road. Note, that this test helped to decide on which method to use for the final tests, and there was no blink detection during this early test.

5 Implementation

Summary. *Description of the implementation used to work with the processor as well as a description of the requirements for the software. All implemented software is used in conjunction with the perception processor.*

The first goal for the testing phase was to design and implement a graphical user interface (GUI), to communicate with the board without using the command line tool. Initially, the perception processor comes with some command line tools to send the configuration packets and for retrieving the results send in return from the processor. Requiring manually typing hexadecimal codes needed to configure the processor is very sensitive to errors which are hard to detect. To facilitate easy communication as well as creating an easier way to configure the processor, a GUI has been designed and implemented.

5.1 GUI

The GUI will allow to create, open and edit files that contain configurations. Editing the configurations will be done using GUI elements such as drop down menus, combo boxes and sliders. Changes made to the configurations can be saved, and send to the perception processor as well. Furthermore, the GUI will have the option to retrieve the results of the processor at run time and optionally reply with adjusted configurations.

5.1.1 Goal of the GUI

Like before mentioned (See 3.3 Communication), the perception processor comes with some command line tools. Since using the processor this way requires a lot of experience, a sub goal of the project was to facilitate working with the processor. All desired settings have to be looked up within the packet reference manual, and typing them requires precise work. Goal is to facilitate the creation of configurations as to minimize to changes for errors whilst creating new configurations.

5.1.2 Requirements

A GUI can facilitate the way of working with the processor. Goal was to create a tool that allows for easy selecting some configurations, and saving those configurations for later usage, or for sending them directly to the perception processor. Furthermore, the results being send from the processor would have to be retrieved for gaining insight in the results.

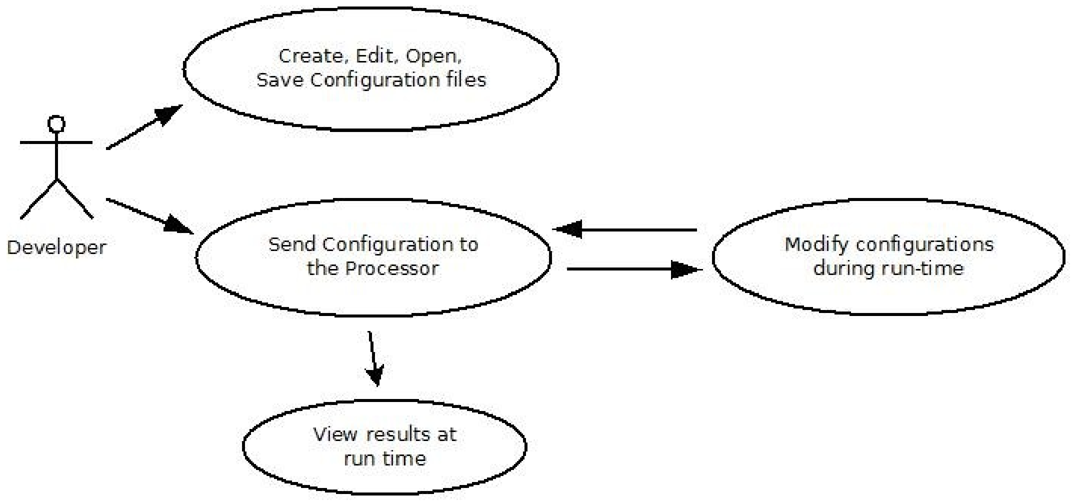

Create, Edit, Open, Save Configuration files
Developer
Send Configuration to the Processor
Modify configurations during run-time
View results at run time

Figure 34: Use Case diagram for the GUI

Since the accompanying tools are all written in C++, it seems like a natural choice to implement the GUI in C++ as well. As a platform, Qt Creator was chosen, since it offers some nice functionality for using plug and play components such as sliders, drop-downs and combo boxes.

5.1.3 Functionality

The GUI should comprise of the following functions:

- Easy to use through the usage of common GUI elements
- Create, edit and save files to modify them later on
- Connect to the perception processor
- Receive data packets from the perception processor.

Additionally, before the user can actually work with the GUI, the tools described in the next section need to be installed.

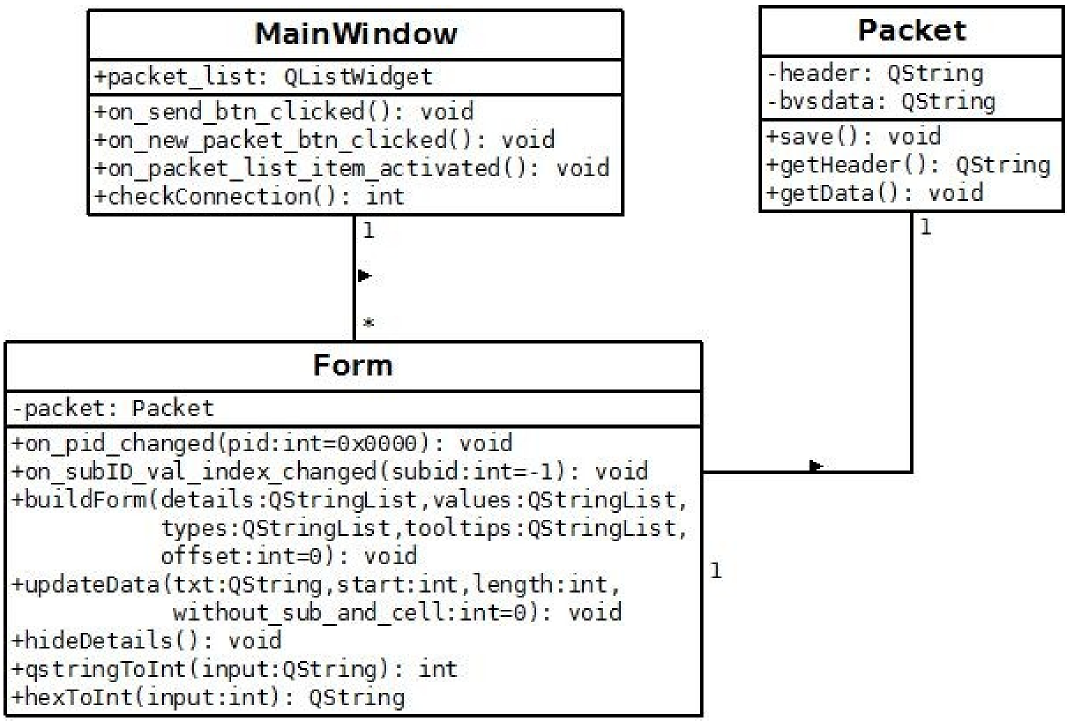
5.1.4 External software

The GUI will be able to use the following external software to communicate with the perception processor. Two applications that come with the processor and NetCat. The applications that come along with the processor are two command line tools. One for sending packets to the processor, while the other is a listener, that catches all results send back from the board. NetCat is an open source utility for Linux to read and write data across network connections. This is necessary to send the updated predictions of the eye locations to the processor. It is possible to use NetCat instead of the two accompanying applications.

To be able to watch the output of the perception processor, the RTP stream from the perception processor needs to be captured and displayed. For this thesis, the open source version of Mplayer has been used, for it offers good functionality on displaying RTP streams over an Ethernet connection.

5.1.5 UML

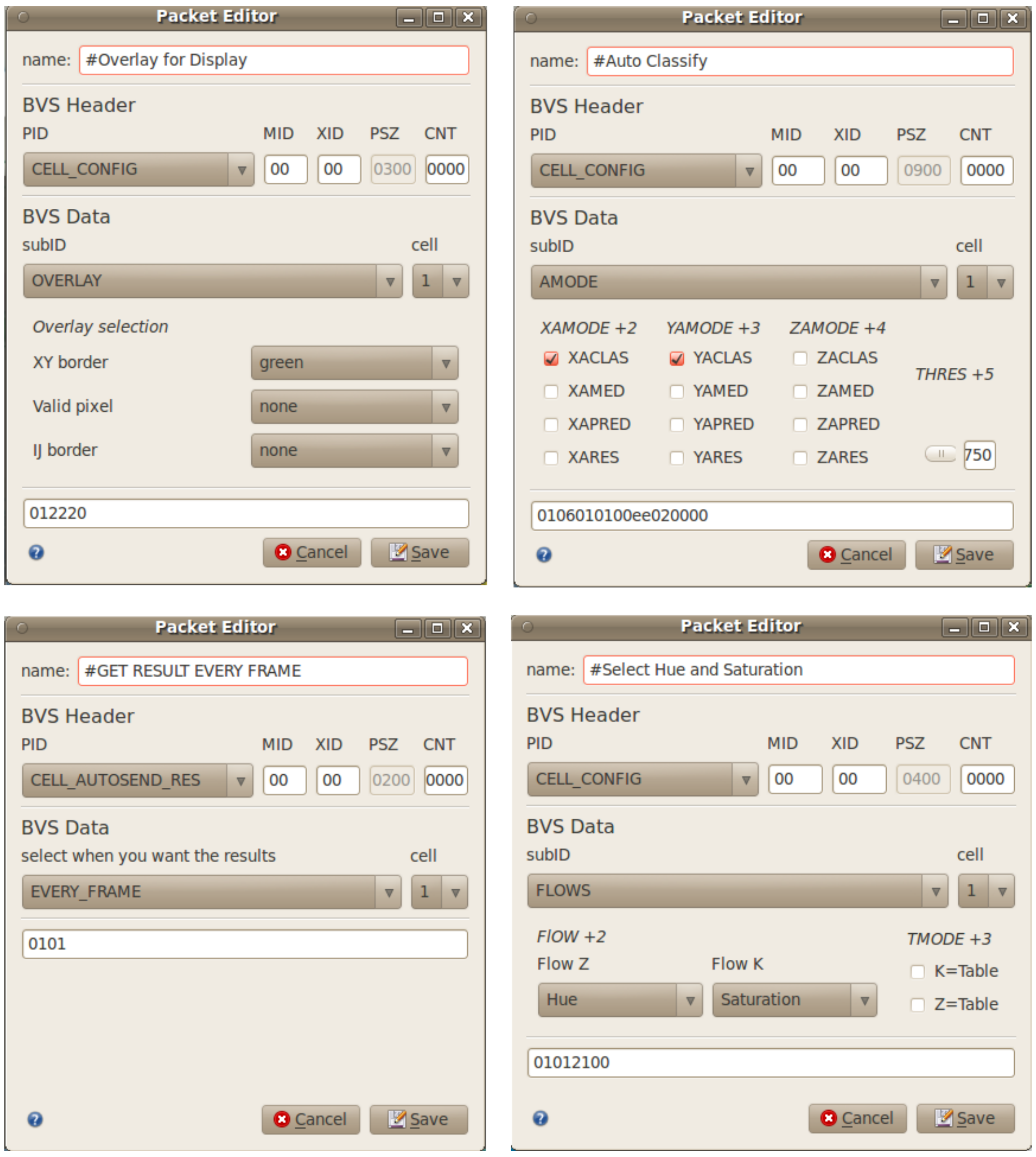
Figure 35 Shows the UML diagram of the GUI design. It contains only the most important classes and functions. The `MainWindow` is loaded at start up and contains a list. Packets can be added to the list and packets referring to the same cell can be



MainWindow
+packet_list: QListWidget
+on_send_btn_clicked(): void
+on_new_packet_btn_clicked(): void
+on_packet_list_item_activated(): void
+checkConnection(): int
Packet
-header: QString
-bvsdata: QString
+save(): void
+getHeader(): QString
+getData(): void
1
*
1
1
Form
-packet: Packet
+on_pid_changed(pid:int=0x0000): void
+on_subID_val_index_changed(subid:int=-1): void
+buildForm(details:QStringList,values:QStringList, types:QStringList,tooltips:QStringList, offset:int=0): void
+updateData(txt:QString,start:int,length:int, without_sub_and_cell:int=0): void
+hideDetails(): void
+qstringToInt(input:QString): int
+hexToInt(input:int): QString

Figure 35: UML diagram of the most important classes of the GUI.

grouped together. Double clicking a packet in the list opens the `Form` that enables the user to edit the packet by selecting the options in the Form. The bottom line of the Form is the actual representation of the hexadecimal packet that is going to be send to the board. Note that only the most relevant classes and methods are described within the UML diagram.



Packet Editor

name: #Overlay for Display

BVS Header

PID	MID	XID	PSZ	CNT
CELL_CONFIG	00	00	0300	0000

BVS Data

subID: OVERLAY — cell: 1

Overlay selection

XY border: green

Valid pixel: none

IJ border: none

012220

Cancel Save

Packet Editor

name: #Auto Classify

BVS Header

PID	MID	XID	PSZ	CNT
CELL_CONFIG	00	00	0900	0000

BVS Data

subID: AMODE — cell: 1

XAMODE +2	*YAMODE +3*	*ZAMODE +4*	*THRES +5*
✓ XACLAS	✓ YACLAS	ZACLAS	
XAMED	YAMED	ZAMED	
XAPRED	YAPRED	ZAPRED	
XARES	YARES	ZARES	750

0106010100ee020000

Cancel Save

Packet Editor

name: #GET RESULT EVERY FRAME

BVS Header

PID	MID	XID	PSZ	CNT
CELL_AUTOSEND_RES	00	00	0200	0000

BVS Data

select when you want the results: EVERY_FRAME — cell: 1

0101

Cancel Save

Packet Editor

name: #Select Hue and Saturation

BVS Header

PID	MID	XID	PSZ	CNT
CELL_CONFIG	00	00	0400	0000

BVS Data

subID: FLOWS — cell: 1

FLOW +2

Flow Z: Hue

Flow K: Saturation

TMODE +3

K=Table

Z=Table

01012100

Cancel Save

Figure 36: The dynamic interfaces for editing an individual packet.

5.2 Creating configurations

Creating and editing of configurations is done using the GUI shown in Figure 37, below. It provides ways to create a configuration file by adding packets for each cell to a list. The order in which the packets are being send to the processor is important, so the interface provides some utilities for organizing the packets.

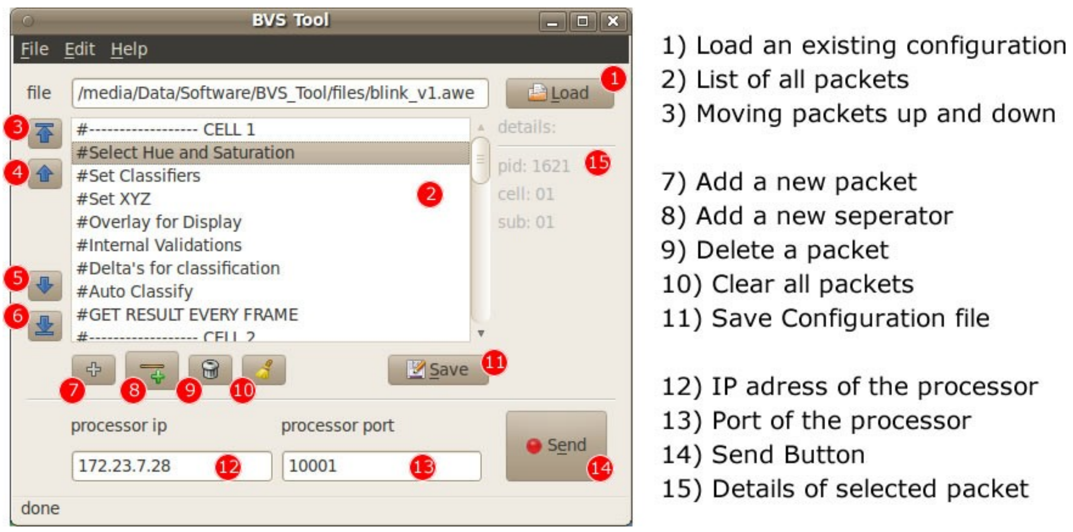
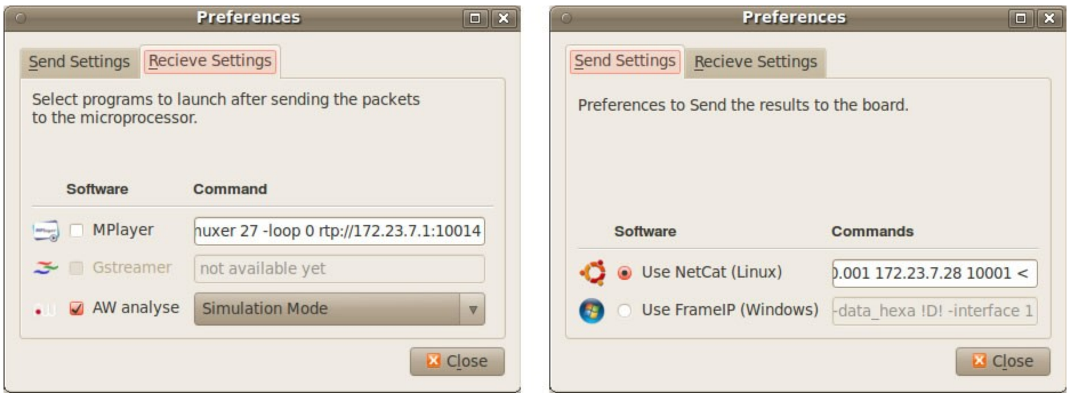


Figure 37: The interface for creating and editing configurations.

Upon selecting a specific packet out of a configuration, the user can edit that specific packet. The interface for editing individual packets is a dynamic form, based upon the underlying code of the packet. Some elements provide sliders to adjust boundaries, while others present selection boxes to the user. Figure 38 Shows four different examples of the forms to edit a single packet. From the UDP protocol we have to set a header (top part in Figure 38). Most of the time this does not need to be changed. All header data is filled in automatically based upon the packet to be send. Below the header part, the actual data can be filled in. Depending on the kind of configuration it is possible to slide, select or enter values to configure the processor. By Saving the packet, you will return to the main window where all the packets are listed in the order the user wants to send them to the processor.

5.3 Sending configurations

Once all packets have been created, it is time to send them to the processor. This is done by simply clicking the “Send” button (nr 14 in Figure 37). As a default action, the GUI launches a Blink-Detector class, which monitors the blinks of the newly started session. It is possible though, to adjust this in the “Preferences Menu”.


Preferences

Send Settings | Recieve Settings

Select programs to launch after sending the packets to the microprocessor.

Software	Command
☐ MPlayer	nuxer 27 -loop 0 rtp://172.23.7.1:10014
☐ Gstreamer	not available yet
✓ AW analyse	Simulation Mode

Close

Preferences

Send Settings | Recieve Settings

Preferences to Send the results to the board.

Software	Commands
● Use NetCat (Linux)	).001 172.23.7.28 10001 <
○ Use FrameIP (Windows)	-data_hexa !D! -interface 1

Close

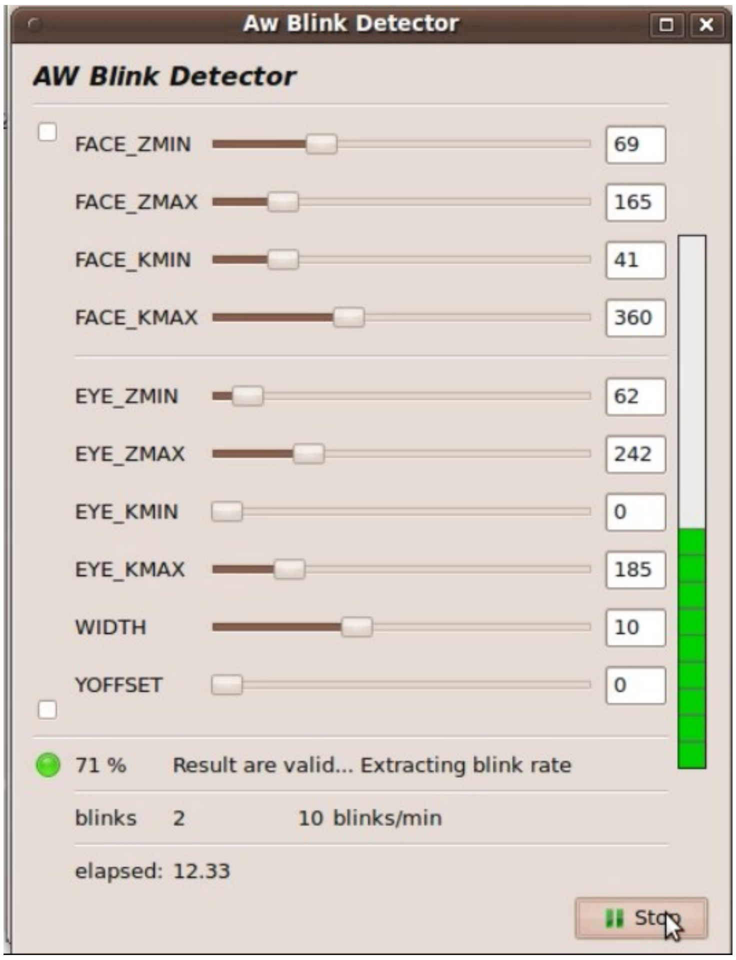
Figure 38: Preferences menu, for selecting which class to load after sending the configuration. And for selecting the ip and port address of the perception processor as configured in the internet connection settings.

In the preferences menu, alternative class names can be started. It is up to the developer to implement that specific class, but the possibility is there for facilitating future work. It seems logical to start a new class depending on the kind of task performed by the processor. While this Thesis focus lies in blink detection, the Blink Detection class is started by default.

5.4 Viewing results at run-time

Once the processor is running, the user should be able to observe what happens and see in an instance how many blinks occur. The top part of Figure 39 provides some tools for fine tuning the configurations. Each time one of the sliders is released, it's corresponding value is being send to the processor. The processor in turn, immediately updates the configuration. At the same time, the lower part of Figure 39 shows a resume of the actual process.

It is here, in the lower part, where the user can see important information. The info right of the green dot in Figure 39 shows a percentage of spatial checks that have passed as well as a message explaining what goes wrong when not all the checks have been passed.


Aw Blink Detector

AW Blink Detector

FACE_ZMIN 69
FACE_ZMAX 165
FACE_KMIN 41
FACE_KMAX 360
EYE_ZMIN 62
EYE_ZMAX 242
EYE_KMIN 0
EYE_KMAX 185
WIDTH 10
YOFFSET 0

71 % Result are valid... Extracting blink rate

blinks 2 10 blinks/min

elapsed: 12.33

Figure 39: Top part of the figure above is used to fine tune the current configuration. While the lower part of the interface provides feedback of the current session. Finally, the bar on the right is the first step towards an indicator for the level of drowsiness.

Even lower, a resume of the number of blinks, the number of blinks per minute and the amount of time since the session started is shown. These values are also used to drive the alertness meter (vertical bar on the right in Figure 39). Once all packets are send to the board, the GUI starts the `BlinkDetector` class which is responsible for asking the results, and processing the results as can be seen in 4.1.7 Approach II: Prediction.

6 Experiments

Summary. *This chapter will present the experiments performed for the thesis as well as the data needed to replicate the results for future reference.*

While the initial results look promising, the real question was how the perception processor would perform in more realistic experiments. Good results have been achieved in controlled environments with fixed lighting conditions, and clean backgrounds. But is it fast and accurate enough to give acceptable results in a specific situation, where the lighting conditions are uncontrolled. To get answers to these questions, a final series of tests is performed in which people where asked to drive laps in a racing simulator game. Followed is a description of the tools and platforms used for performing the tests, as well as a description of the test goals and approach. The results are denoted in Chapter 7 Results.

6.1 Tools and platforms

To being able to perform the tests, a set of tools have been used. To mimic an in car environment, TORCS is used as a driving simulator along with a steering wheel (6.1.1 TORCS (The Open Racing Car Simulator)). Furthermore, to be able to extract the number of blinks during a race, chosen is to use an EEG scanner during some of the tests (6.1.2 EEG Scanner & TruScan). At the TU Delft the EEG scanner has been used in different experiments along the way, and has proven that it can indeed be an valuable asset, given that it's used right. Once the EEG data and the output of the processor is acquired, all data has to be compared to draw any conclusions. Primarily MATLAB (6.1.3 MATLAB) will be used for that task. Followed is a description of these tools and platforms.

6.1.1 TORCS (The Open Racing Car Simulator)

The Open Racing Car Simulator, or TORCS (see Figure 40), is a open source race game developed for racing but for research as well. TU Delft has extended this game with races and billboards, making this platform suitable for performing such a test. Two different tracks have been used for testing. Most of the tests where performed at the "Road Track 2" created by . Details of the track are shown in Figure 41. This track is chosen because it's approximate length (in minutes). We wanted to have recordings of around 2,5 minutes, as to make sure the test subjects actually blink during the race, but not have too much data to analyze on the other hand.

Figure 40: The Open Racing Car Simulator.

Name: CG Track 2

Creator: Christophe Guionneau

Type: road

Length: 3185.83m

Width: 15.00m

# Pits: 16

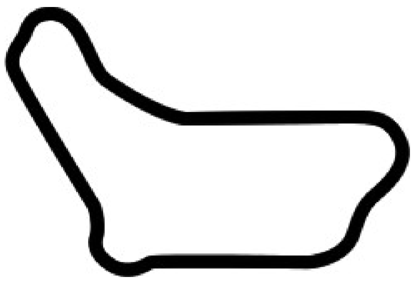


Figure 41: CG Track 2, the one used in the tests.

6.1.2 EEG Scanner & TruScan

Electroencephalography (EEG) is the recording of electrical activity along the scalp produced by the firing of neurons within the brain.[2] It can measure the activity of the brain using a brain cap that is connected to a computer. Figure 42, below shows the hardware available at the TU Delft.

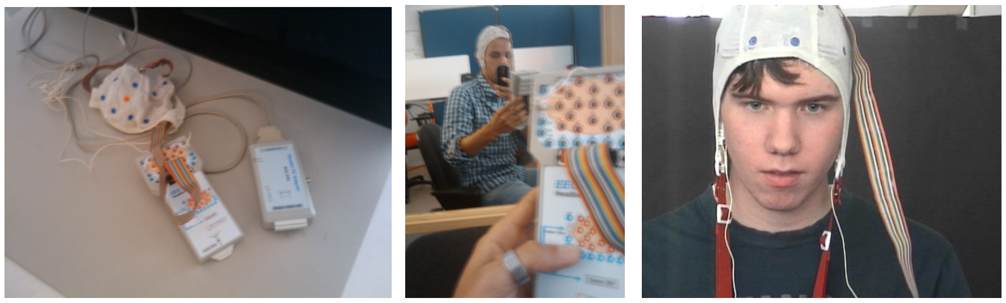


Figure 42: Attributes used for the EEG scans.

As with all EEG systems, the signal is very sensitive. A detailed description of how to work with the EEG scanner at the TU Delft can be found in [37] . There is a chapter about removing artifacts from the EEG recordings. For this project we are interested in one of those particular artifacts, Blinks. Hence, we will have to remove all artifacts, whilst keeping the blink data intact. Given the nature of blinks in EEG data, this

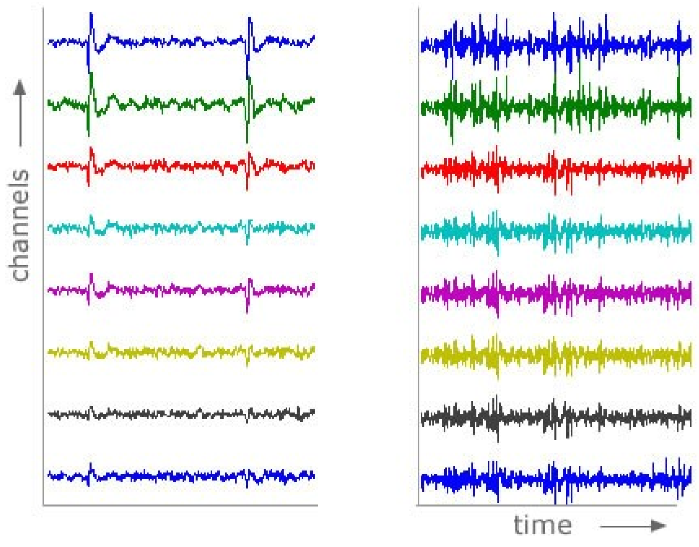


Figure 43: Samples of EEG data. Left: clean data where two blinks can easily be identified. Right: A noisy sample caused by (in this case) movement of the test participant.

should be apparent, for the blinks can easily be detected in the EEG data, if the data is free from noise. EEG data is very sensitive to external factors. Muscle movements, or even the net tension can heavily influence the recorded data as can be seen in Figure 43. A lot of these artifacts can be removed using the techniques described in [37]. The main difference is that we do not want to remove the blinks, instead the aim is to remove as much as possible except the blinks. More detailed information about conducting EEG experiments and Brain Computing in general, can be found within the Appendix

6.1.3 MATLAB

MATLAB is a powerful mathematical framework for solving engineering problems. Many extensions have been written to give tools in specific solution areas. The data obtained from the EEG scans, will be exported to use within MATLAB. The export functions in TruScan Explorer result in two different files. One is an ASCII text file, containing all the data recorded from each of the 19 channels. While the other is an ASCII *.Ini file, containing all manual annotations during the test.

The manual annotations are entered during the recordings by pressing a predefined key. The annotations can be used to set references, and to see the corresponding EEG response at that given moment.

6.1.3.1 EEGLAB

EEGLAB is an interactive toolbox for MATLAB. Is contains lots of useful functions to process EEG data. EEGLAB is used to pre-process the data as obtained from TruScan for usage in MATLAB. EEGLAB is already equipped with the exact scalp map used for our testing, giving the possibly to gain insight in the locations of the responses on the brain. Blinks typically show the biggest response in Fp1 and Fp2 (see Figure 44).

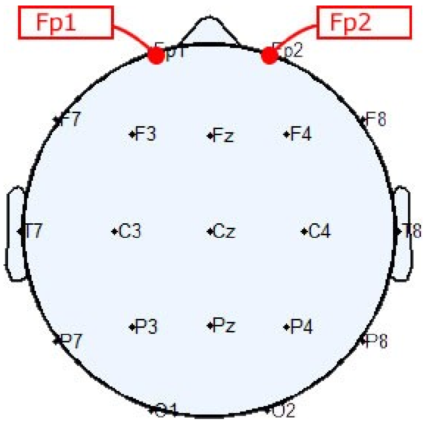


Figure 44: Locations most sensitive to eye blinks, Fp1 and Fp2.

6.2 Test Goals & Expectations

The main goal of the test is to find out whether the number of blinks during a race according to the perception processor is consistent with both the manual and EEG data. Since using the EEG cap influences the behavior of the participants, two runs of the test will be done. The first one without EEG, as to recreate normal driving conditions. The second run with an EEG cap, to facilitate the comparison with respect to number of blinks observed. Even under these conditions, the tests are far from a real driver, since driving on a computer screen is limited to a relatively small screen.

6.2.1 Goals

Below is a description of the goals of the tests:

- Test Face detection robustness
- Test Eye detection robustness
- Test blink detection robustness
- Find overall robustness of the blink detector.

By analyzing the individual parts in the blink detection chain, hoped is to learn what parts of the chain work well and what parts need improvement.

6.2.2 Expectations

Furthermore, we expect the tests in which the participants are asked to sit still provide better results in the blink detection domain. Since there is less movement, it should be easier to find and track the pupils and extract the blinks. As for the performance, given the results from the early tests inside a car (see 4.3 Test case : Driving in a car) we expect the results to be almost the same.

6.3 Test approach

Goal of the final tests is to determine whether the perception processor is capable of producing good results for multiple test subjects. Tried is to create a blink detector that has the potential to be used within a real car environment. The area of blink detection is chosen since it is:

- Challenging since blinks occur fast, and the pupils are small
- Of great interest in the new developments within cars.

One of the most difficult areas of application is the area of blink detection. Blinks typically occur very fast, and give a good indication about the alertness of a driver [4]. To be able to test the processor under these circumstances a total of 15 test persons have been recruited. People are asked to drive 3 laps in the game of TORCS (see 6.1.1 TORCS (The Open Racing Car Simulator)), using a steering wheel. The complete test is recorded using a standard analogue video camera. After racing three laps to get acquainted with the racing options, people are asked to drive a second time. The second race laps are recorded as well, but this time the test persons wear an EEG headset as well, to get an indication of the number of blinks. Finally, after all tests where performed, the recorded videos where used as an input for the perception processor. The results in the number of blinks obtained from the EEG can than be compared with the results according to the processor.

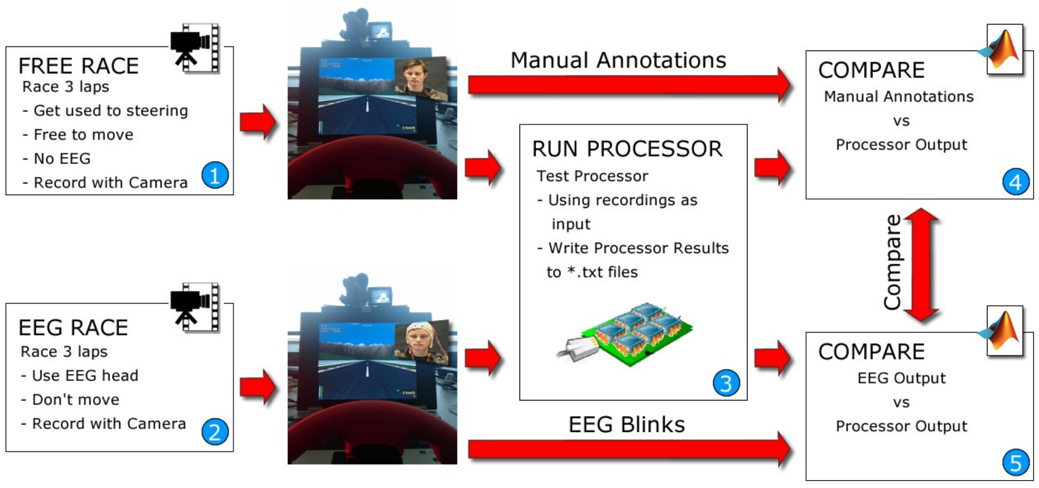


Figure 45: Diagram of the test approach.

1) First run where subjects are allowed to move freely.

2) Second run, where people have to sit still, and being EEG recorded.

3) Both tests are processed by the processor after wards.

4) The first run has to be annotated manually, since there is no EEG data. These annotations are compared to the Processor output.

5) The Second run has EEG data, so the EEG output is compared to the processor output.

6.4 Performed tests

Followed is a list of all the tests performed for this thesis, along with a reference to the section where the results of that specific test are presented. Also, they are listed in Table 2.

Table 2: List with all the experiments performed.

Experiment	Goal	Result
Free Race Experiment	Test influence of rotation/ movements on the results	Chapter 7.1
EEG Experiment	Compare detected blink rate of the processor with the blink rate from the EEG scans	Chapter 7.2.

6.4.1 Free race experiment

During the “Free Race Experiment”, participants are asked to race 3 laps in the simulator, with no moving constraints. Goal is to find out if the perception processor is able to find and track the eyes if subjects are allowed to move freely. By running the recorded session through the perception processor, the number of blinks is compared to a manual reference. Since during this experiment, no EEG cap is worn, the comparison is done with respect to manual counts of occurring blinks.

6.4.2 EEG experiment

During the EEG Experiments, participants are asked to sit as still as possible while racing 3 laps in the simulator. Since moving would influence the EEG data, people are asked to move a little as possible. Goal is to compare the results of the EEG scans with the results of the perception processor as to get an indication of how well the blinks are extracted. This time, the EEG data shows clearly when blinks occur. After filtering the data using EEGLab and Matlab (see Section 6.1.3 MATLAB), the number of blinks are compared with the number of blinks according to the perception processor.

7 Results

Summary. *This chapter will present the results and statistics of the experiments mentioned in the previous chapter.*

The results of the tests will be described within this section. At first a single test is reviewed in detail. After that findings on the complete test set are presented. The results are split up into different categories. First an analysis of the face detection part is given, followed by an analysis of the results in Eye detection. Finally, the results of the overall blink detector are provided. Table 3 shows a brief summary of the experiments. A total of 13 participants volunteered for the tests, and the average

Table 3: Summary of the results.

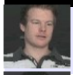
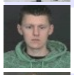
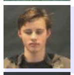
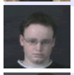
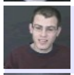
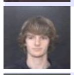
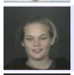
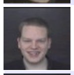
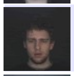
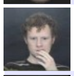
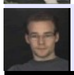
		Test 1	Test 2
	code name	Free Race	EEG Race
Stats	Participants #	13	13
	Male %	92.5	92.5
	Female %	7.5	7.5
	Facial hair %	15.4	15.4
	Glasses %	30.7	30.7
	avg. blinks #	19.31	12.77
	avg. detection rate %	50.37	45.71
Details	total blinks	251	166
	false positive #	110	157
	false negative #	36	93
	blinks detected #	112	60

number of blinks recognized correct is 50.37% for the Free Races, and 45.71% in the races with the EEG Scans.

7.1 Free race results

The results of the experiments in which the participants are allowed to move freely are presented within this section. While the overall blink detector works like a chain of multiple sub tasks, those sub tasks will be analyzed separately to get an idea of the overall performance of the blink detector. By doing so, a better understanding of the individual parts in the chain is obtained. First the overall results for the “Free Race Experiment” are presented. The following sections will explain each part of the

Table 4: Results for the Free experiments.

participant	test type	Recognition Rate (%)		Blinks					
	Free race	Face	Eyes	Actual	Processor	+	-	CORRECT	%
	FREE Race	100.00%	86.07%	**3**	5	3	1	**2**	**0.67**
	FREE Race	100.00%	99.49%	**6**	4	0	2	**4**	**0.67**
	FREE Race	100.00%	64.04%	**21**	37	27	11	**9**	**0.43**
	FREE Race	100.00%	99.39%	**14**	2	1	13	**1**	**0.07**
	FREE Race	100.00%	91.75%	**7**	30	24	1	**6**	**0.86**
	FREE Race	100.00%	99.97%	**26**	28	5	3	**23**	**0.88**
	FREE Race	100.00%	82.50%	**92**	32	5	65	**24**	**0.26**
	FREE Race	100.00%	96.81%	**0**	9	9	0	**0**	**0.00**
	FREE Race	100.00%	96.41%	**20**	22	5	3	**17**	**0.85**
	FREE Race	100.00%	98.39%	**6**	8	3	1	**5**	**0.83**
	FREE Race	100.00%	97.87%	**2**	1	0	1	**1**	**0.50**
	FREE Race	100.00%	100.00%	**48**	16	1	33	**16**	**0.33**
	FREE Race	100.00%	46.19%	**6**	31	27	2	**4**	**0.19**
	Overall Average:	**100.00%**	**86.44%**	**251**	**225**	**110**	**136**	**112**	**50.36%**

chain separately with examples of common errors. In Table 4 the results are shown for the Free Experiment. On average, 50% of the blinks that actually occurred are detected correct.

7.1.1 Face detection

The third column of figure Table 4 shows the recognition rate for the face. The face is detected using the Color method described in 4.1.2 Classification through color. The face detector works very well. It has been tested on people of different races, and with different levels of facial hair. In all cases the application is able to continually find and track the face. Since the face is the beginning of the detection chain, it is comfortable that face tracking works well. One point to mention is that at the current configuration it is only possible to detect a single face. Since a single cell in the processor is responsible for locating and tracking the biggest area that contains skin colored pixels, it is limited to one face only. If two faces appear in the scene, depending on the distant between the two faces, it either sees them as one giant face, or it selects the biggest and ignores the other. All faces presented to the blink detector can be found and tracked over time.

7.1.2 CLEAN Race: Eye region prediction

To get an indication of the robustness of this method, take a look at the fourth column of Table 4. The time in which the eyes can be found is measured compared to the total time of the recorded clips. On average, the application successfully finds and tracks the eyes in 86,44% of the time when participants are allowed to move freely. The rest of the time the application is scanning for a suitable region, as explained in section 7.3.1 Vertical scanning for eyes.

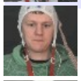
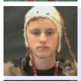
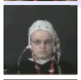
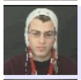
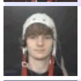
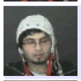
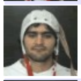
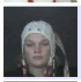
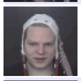
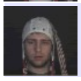
7.1.3 CLEAN Race: Blink detection

While the region of the eyes can be found most of the time, the same is not true for counting the blinks. The last columns in Table 4, show the number of blinks that actually occurred, the number of blinks according to the processor. Additionally the number of False Positives and False Negatives (e.g. blinks missed by the processor) are depicted. The resulting percentage of correct identified blinks is only 50,36%. This number is not what was expected, but note that 9 participants have really bad blink detection stats, while 4 have a recognition rate of more than 80%. Section 7.3 Interpretation of the results will describe some of the most common errors and discuss some of the aspects that work very well. Note that since the clean races are performed with the absence of an EEG scanner, the recordings have manually been checked for blinks. Those manually detected eye blinks are used as a reference for the performance of the eye blink detector.

7.2 EEG race results

Results for the experiments with the EEG cap are shown in Table 5. Using the EEG cap as a reference, the number of correctly recognized blinks drops even further to 45,71%. Since the face detection works equally well on both the Free Experiments as on the EEG Experiments, the section of face detection is skipped.

Table 5: Results for the EEG experiments.

participant	test type	Recognition Rate (%)			Blinks				
	Free race	Face	Eyes	Actual	Processor	+	-	CORRECT	%
	EEG Race	100.00%	99.30%	**1**	1	0	0	**1**	**1.00**
	EEG Race	100.00%	78.81%	**9**	8	1	3	**6**	**0.67**
	EEG Race	100.00%	77.48%	**23**	13	10	15	**8**	**0.35**
	EEG Race	100.00%	99.91%	**9**	2	7	0	**2**	**0.22**
	EEG Race	100.00%	79.16%	**5**	63	59	1	**3**	**0.60**
	EEG Race	100.00%	74.75%	**24**	14	3	13	**8**	**0.33**
	EEG Race	100.00%	74.73%	**61**	20	4	45	**14**	**0.23**
	EEG Race	100.00%	82.28%	**2**	36	34	0	**2**	**0.06**
	EEG Race	100.00%	76.80%	**6**	4	1	3	**3**	**0.50**
	EEG Race	100.00%	99.99%	**4**	4	1	1	**3**	**0.75**
	EEG Race	100.00%	88.88%	**4**	7	3	0	**4**	**1.00**
	EEG Race	100.00%	99.52%	**12**	1	0	11	**1**	**0.08**
	EEG Race	100.00%	63.22%	**6**	39	34	1	**6**	**0.15**
	Overall Average:	**100.00%**	**82.72%**	**166**	**212**	**157**	**93**	**61**	**45.71%**

7.2.1 EEG Race: Eye region prediction

Like the data illustrated, the eye region prediction for the EEG Experiments works correct 82,72% of the time. This is a little lower than for the Free Experiments. This is the opposite as was expected, since during the EEG Experiments, participants where asked to move still. This would indicate that the eyes would move less than in the Free Experiments and thus a more accurate prediction of the eye region could be made. In the Interpretation of the results section an explanation is given for this strange result.

7.2.2 EEG Race: Blink detection

As can be seen in Table 5, more specifically in columns five through 10, the average percentage of correctly detected blinks is around 45,71%. This result is disappointing on one hand, but is also promising given the fact that no special algorithms to increase the performance have been implemented yet. This means that our blink detector works in approximately 50% of the cases out of the box. And there are lots of potential techniques available to improve this number (see 9 Future work).

7.3 Interpretation of the results

While the rough results are presented, a discussion of those results will be given. Some of the implemented techniques work very well, while others are extremely disappointing. Tried is to give an explanation for some of the phenomena that can be observed while taking a closer look to the results of the experiments.

7.3.1 Vertical scanning for eyes

Based upon the results of the face detector, the processor will predict where the eyes should be located. The prediction is based upon the geometric properties of the human face [38]. If no eye can be found in the predicted area, a vertical scan is started until the eyes indeed are found. This process can be seen in the series of screen shots in Figure 46. The series of screen shots show how the initial prediction, shown in the top left image, is a little bit above the center of the middle of the face. No eye like features are found within this region, so the vertical scan is initialized. At first the predicted region moves down. At each step, the area is scanned for eye shaped features. No such features are present, so the scan continues all the way to the bottom of the face. Upon hitting the lower boundary of the face region, the scan direction is reversed to go up again.

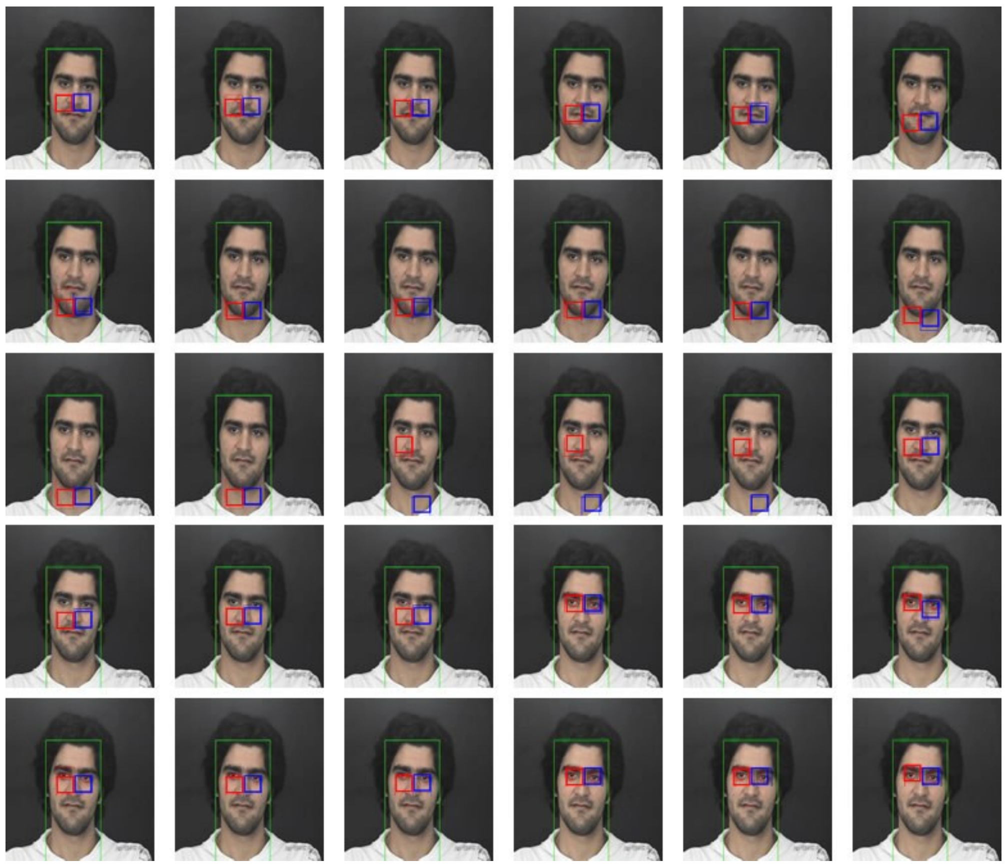


Figure 46: Vertical scanning for a potential eye region.

This method of scanning the face region for eye-shaped features works great. In the above case, it takes the processor less than one second to find the eyes within the face. Ultimately, when no eyes are present in the face, the vertical scan will continue to sweep up and down the face until it can positively identify eye shaped features.

The method makes a prediction for both eyes (left and right) separately. This can be seen in some of the frames in Figure 46 above. Reason behind this is that a person might be winking, so there might actually be only one eye. Results from both predictive areas are used for the next prediction.

7.3.2 Pupils

It is clear from the results that the performance of the processor in the fist two stages is acceptable. However, detecting the pupils themselves is not going correct in most of the cases. One possible explanation is that with the current perception processor it is only possible to search for Structure, Color, and Movements of the biggest object. However, the pupils tend to be very small, and to make it even harder, surrounded by similar features. This causes the processor to falsely classify the eyebrows as pupils in some cases. The current version uses some checks on the computer side to check for these falsely classified features, by looking at geometrical and positional properties. However, exactly finding and tracking the pupils over time

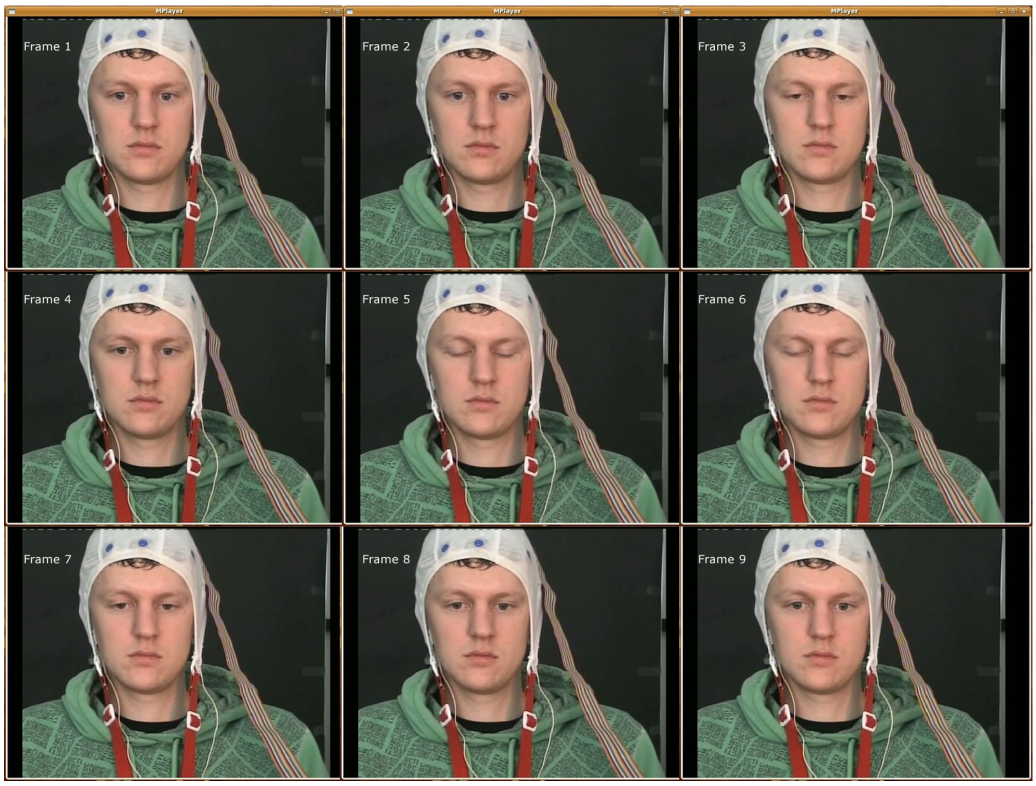


Figure 47: Example of a near blink, followed by an actual blink.

seems to be the most challenging problem of the eye blink detector. It does not really matter if subjects are moving constantly or sitting motionless, the presence of similar features continue to distract the perception processor. The best solution might be to look in another direction for classifying pupils. Or to use classical algorithms, like Hough transforms and Kalman filters to improve tracking of the pupils.

7.3.3 Near blinks

Another issue when trying to detect the blinks are near blinks. The EEG scanner notices heavy eye movement, which results in a blink. On the images recorded with a 25fps camera however, the people do not seem to blink. Since the processor only registers a blink when the eyes are completely closed, e.g. a height of 0, those blinks are missed by the processor. One such example is shown in Figure 47. Frame three results in a blink in the EEG scan, but on the images, the eyes are not completely closed, resulting in a non-blink according to the processor.

7.3.4 False negatives & False positives

A lot of the blinks that actually occur in during the experiments are being missed by the blink detector. It is really interesting to note that there are clear cases when absolute no blinks are missed, while other generate false negatives all the time. This is an indication that the perception processor has no preference, or flaw, in either missing or adding blinks that are not present. This implies that the processor can deal with both situations, but depending on the type of person in front of the camera, acts either as with a tendency to miss blinks, or with the tendency to register extra blinks.

7.3.5 Glasses

30% Of the participants wore glasses during the test. This comes down to 4 participants. In 3 of those cases, the glasses of the test subjects where very light or transparent, like shown in Figure 48 below. All those experiments resulted in a relatively large amount of False Positives.

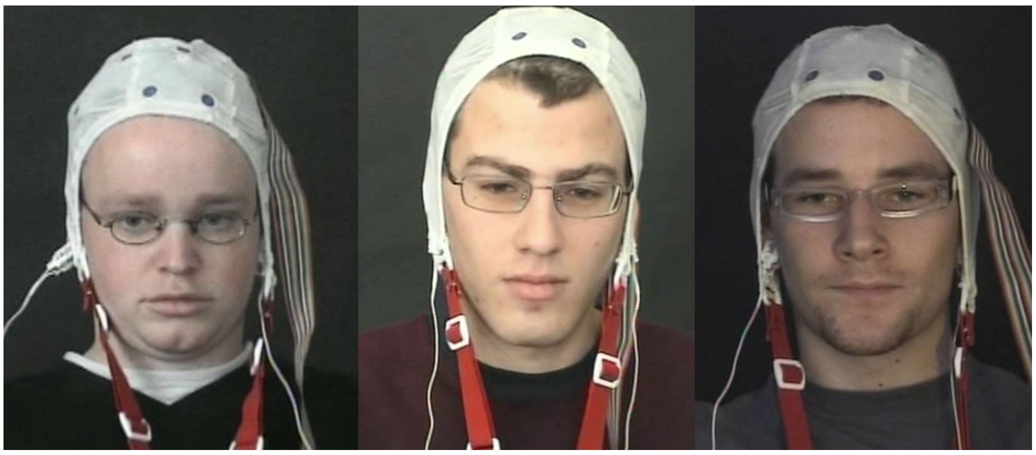


Figure 48: Examples of people with light, or transparent glasses.

E.g. the light glasses cause the wrong features to be tracked, and once they move very fast, this results falsely to registering a blink. Especially since the glasses are quite often positioned on the border of the Eye region, a sudden move of the head will move the glasses outside of the eye region, causing to loose the eyes and register a blink. By improving the pupil detection part of the eye blink detector, this type of error might be circumvented as well. The question remains as to what is the best discriminator for usage with the limited options available on the perception processor.

7.3.6 Dark eyebrows

All test participants with very dark eyebrows hair have higher than average percentage of incorrect blink detection. Figure 49 shows some of those participants with very dark eyebrows.

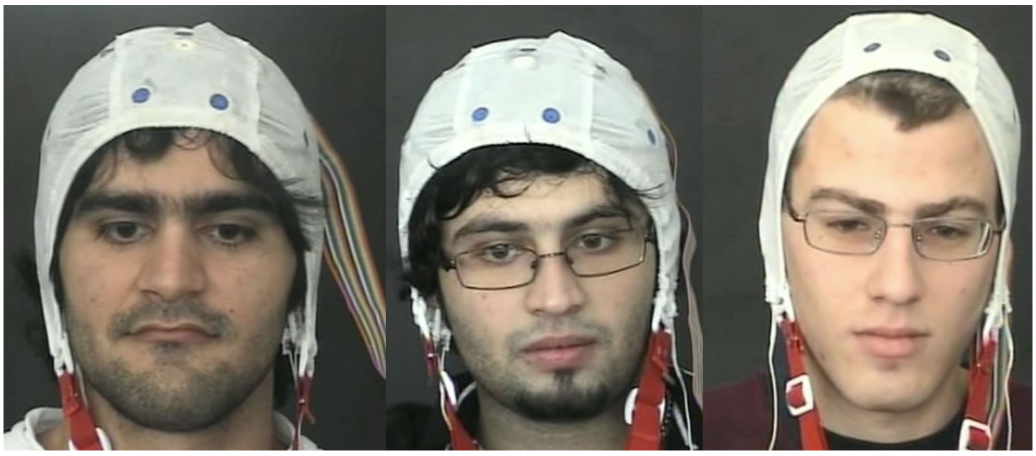


Figure 49: Examples of people with very dark eyebrows.

Since the eyebrows usually share some of the characteristics of the pupils, they are often mistakenly being detected as the pupils. In these cases we see a very high percentage of blinks missed, indicating that when an actual blink occurs, the detector will think the eyes are still open, since it is tracking the eyebrows.

8 Conclusion

Summary. *All conclusions will be presented here.*

Looking at the results and experiments tried is to answer the questions declared in the introduction of this thesis (1.2 Research questions). Although the results of the final experiments are a bit disappointing, the perception processor can definitely be an added value to many problems as it is. And only a few added features can lead to a more robust tool for usage in many computer related vision problems.

8.1 What is the perception processor?

The perception processor is a fast and flexible utility that can be used in a wide range of applications. Since it is configurable during run time, the behavior of the processor can adapt to different situations. While the perception processor is not specifically designed for the problem of eye blink detection, it's flexibility allows to tackle such a problem. Instead of creating a fully robust eye blink detector, the perception processor can be used as a tool to solve the problem. Domain specific algorithms can be implemented on a computer to enhance the results within a specific domain without loosing the opportunity to tackle other problems with regard to computer vision.

8.2 Applicable Domains

During the tests different problems have been solved. Some of the applicable domains in which the perception processor can be an added value are:

- Monitoring
- Surveillance
- Object tracking/ recognition
- Virtual Reality
- Man machine interaction

8.3 Interface for interaction

The interface designed to work with the perception processor is implemented with future work in mind. It can load different classes after sending a specific configuration, and it has a gentle learning curve compared to manually creating configurations. Improvements can be made by implementing a queuing mechanism to provide better performance. As for, the prototype created for this thesis, has to wait for the algorithmic calculations to finish before sending an updated configuration. Implementing such a queuing mechanism, possible in a separate thread could speed up the methods where direct interaction with the perception processor is necessary.

8.4 Create a tool to monitor eye blinks

The blink detection class is one of the classes that can be launched after sending the eye blink detection configuration. The classes' sole responsibility is to count the number of blinks based on the data received and to visualize those results. A drowsiness indicator is available as well, but it remains to be investigated what eye blink frequency is correlated to what level of drowsiness. Additionally, the class provides means to tweak some of the most important configuration settings at run time. This feature allows to find the human face under any lighting condition. However, an automatic approach, without manual tweaking is preferred. Possibly this can be achieved by newer version of the perception processor, that includes more advanced combinations of flows. Or even a special illumination flow to get information about the illumination of a scene.

8.5 Detect and track human faces?

The perception processor in its current state, can readily be used to find and track human faces in many environments. When tweaking some of the settings for any face in any lighting condition a suitable configuration can be created to robustly find and track a human face over time.

What is currently lacking, is the robustness to create a single configuration that works under all lighting conditions on people of all races. Since different lighting condition have a huge effect on the hue and saturation of the faces, a single configuration is currently not capable to track all possible faces under all possible lighting conditions. If, in advance, the characteristics of the environment are known, the perception processor is well suited to perform such a task. To really come to the point of creating a single configuration capable of doing so, more features need to be added to the perception processor itself, like mathematically combining the color input flows (See 9.2.3 Combining flows mathematically). An alternative is to send a configuration that analysis the properties of the scene first, followed by an adjusted configuration to perform tracking under the given conditions. A computer algorithm should be present to analyze the current conditions and to make decisions about the appropriate configuration to send.

8.6 Detect and track pupils?

To fully track the pupils is more difficult than expected. Some of the evident methods for pupil detection fail for different reasons. A down side of the perception processor is the preferred direction to find and track the biggest objects within an image sequence. By expanding the boundaries of tracked objects, and re-analyzing the histogram information, the perception processor tends to “snap” to the larger object. For the pupils, this is a disadvantage, since the much bigger eyebrows, with similar characteristics are present many of times, near the actual pupils.

Either a better discriminator to make a clear distinction between pupils and eyebrows needs to be found, or extra features need to be added to the perception processor. One of those features should be the ability to search for the most circle like shape. This is currently quite difficult since the classifiers work based upon limited knowledge obtained from calculating histograms.

8.7 Optimal Approach

The optimal approach to detect the eye blinks with the current version of the perception processor is start by looking for the face. Face detection and tracking can be done with the current version. With regard to eye region detection, the best approach is to predict and store the eye regions. By storing previous successful identifications, a better prediction can be made. The use of a Kalman filter can also be an added value to track the eye regions, once a successful identification has been made. A major drawback of the prototype is that it has to wait for some calculations and geometrical checks to finish, before an updated configuration can be send. Ideally, more and more of these calculations and checks should performed by the processor itself, without creating a specific blink detector, and thus, loosing it's flexibility.

8.8 Determine blink rate?

Concluding we can state the the current version used throughout this project is not quite ready for application in complex systems such as blink detection. It simply lacks too many necessary features to provide stable and reproducible results. Features like searching for elliptic shapes is of great importance for applications that operate in biological environments. On the other hand, applications that perform in non biological environments can already benefit from the speed and ease of the functions provided by the perception processor. For example, finding and tracking a face is already performing at a recognition rate of 100 fps, with good accuracy. But using that face as a starting point for more advanced types of recognition requires a little more details.

Especially the lighting conditions are very hard to deal with. Image processing techniques can be used to increase the robustness by intervening in the classification process, but they also intervene in the operational speed of the overall application. By combining flows in a mathematical way, a future version could provide better results on this area as well, without slowing down the overall speed. All experiments with a correct eye blink detection rate of more than 80% come from participants that have a fairly light skin color and light eyebrows. In those cases, the tacking of the eyes went excellent. This proves that the current application is indeed able to be used as a blink detector when some of the shortcomings have been fixed. Those shortcomings appear especially in the final stage of the blink detector, namely, when it is time to track the actual pupils. Whether this step will be improved from the side of the computer running smarter algorithms, or whether the improvements are driven by updates of the processor itself is to be seen. When the perception processor matures, it can definitely be an added value in all applications that need to observe objects.

9 Future work

Summary. *Things to improve, and steps needed to continue presented here.*

From the experiment analysis, we have learned that the processor has great potential. However, the prototype used for the thesis is not yet ready to provide stable results to be used as a blink detector. Looking at the sub problems arising in trying to create a real time blink detector we can conclude that the final part of the chain is not up to the task yet. Most can be gained by the last step, namely improving pupil detection.

Two logical solutions to improve the results of the blink detector are available. Firstly enhancing the high level reasoning performed on the computer side can improve the results. The other solutions steers towards using an updated version of the processor. Both solutions will be discussed here.

9.1 Computer Algorithms

By enhancing the methods and algorithm on the computer side, better results for the blink detector can be achieved. For instance to increase the correct detection of the pupils, Hough transforms or Eye projection methods can be used. A more advanced system for storing previous locations of the eyes can definitely improve the problems that occur when dealing with occlusion. Whatever methods used, all methods will decrease the operational speed of the processor. Since the calculations on the computer side have to be performed before updating the configuration to the perception processor. Therefor, the preferred direction is to move to the other end, towards the perception processor itself. If no future updates of the processor are available, the following items should be considered when expanding the research.

9.1.1 Threads

One of the main reason that the processor is slowed down by calculations performed on the computer side, is that the calculations lead to new insights in better configurations of the processor. Hence, the processor has to wait until the calculations are finished and the newly calculated configurations are being send to the processor. A possible work around might be an intelligent threading and queuing system on the computer side. By reserving multiple threads in combination with a queuing system, the waiting time might be reduced.

9.1.2 High level reasoning

The High level reasoning implementation can be improved as well. Currently a total of 10 checks are performed upon the results that are being send from the processor to the computer. By using more advanced reasoning upon those results, a more accurate location of the pupils can be calculated. This should lead to a more robust pupil tracking algorithm, which spends more time counting blinks than trying to find the eyes.

9.2 Processor update

Because of the reason given, the preferred way to increase the overall performance of a blink detector is to use a new processor version. Ideally, all logic and reasoning should happen on the processor side. Since flexibility is one of the key features of the processor, this is not possible. To place all logic on the processor would implicate that the blinks themselves would be detected on the processor as well. Off course this is possible, but it would only be useful for a blink detector. No other application would benefit from the opportunity to count blinks on the processor itself. Thus, what do we want to move to the processor side?

9.2.1 Search at relative offset

One of the must have features for a new perception processor would be the ability to search relatively within another cell. The current prototype only offers the way to search within the boundaries of a complete cell. Or to position a cell absolute using screen coordinates.

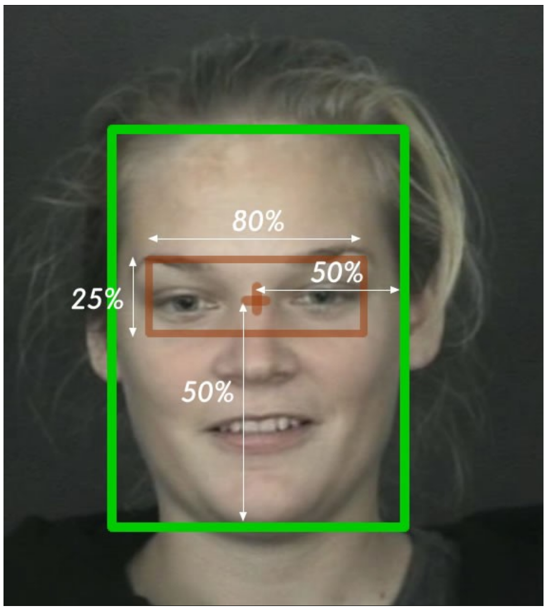


Figure 50: An example of how relative search would work.

Figure 50 Shows an example of the idea behind relative searching within cells. The green cell is responsible for searching the face. Based upon the human anatomy, an initial guess would be to look for the eyes in a block centered in the face and with a

width of 80% of the face and a height of 25% of the face. By using such automatic settings, the extra steps needed to predict the eye location would become redundant.

9.2.2 Search for circular shapes

Yet another missing function the current prototype is missing, is to search for circular shapes. Given the current options available to configure the processor, it seems to be optimized for usage with squares. Indeed, it is possible to find and track a specific edge, as to follow the edge of a mill for example. However, finding a circular feature is currently very tricky. Especially since the difference between a real circle and a distorted circle is not very big (when looking at the resulting histograms).

9.2.3 Combining flows mathematically

The last improvement a new version of the prototype must have is to ability to mathematically create combinations of flows. Right now the possibilities are limited to traditional AND, OR combinations. Especially for skin detection, it might be interesting to look at the classification process of combing the hue, saturation and value flows as described in a paper by Lu et al. [36]. The research states that according to their findings, (Hue + Saturation / Value) provides a constant for skin color independent of races. This could provide a more robust starting point for face recognition which is independent of the illuminating properties of the scene.

9.3 Hardware updates

Even though AW put a constraint on the hardware used for the blink detector, e.g. no wearables and a fairly cheap camera, it might be extremely interesting to see the performance of the current application by using a high speed camera or an infra red camera. Ideally, a combination of both would be interesting. Considering this, it might be more efficient to adjust the type of equipment based upon the function of the processor. By using different types of cameras for different problems, better results might be accomplished without loosing the flexibility and operational speed of the processor.

References

[1] U.S. Department of Transportation (2009), "Driver Distraction in Comercial Vehicle Operations" - *U.S. Department of Transportation Federal Motor Carrier Safety Administration.*

[2] Horne (1992), "Stay awake, stay alive" - *New York State Department of Health*

[3] Mitler (1988), "Catastophes, sleep and public policy." - *Sleep.*

[4] Schleicher, Galley, Briest, Galley (2008), "Blinks and sacades as indicators of fatigue in sleepiness warnings: looking tired?" - *Ergonimics, Volume 51.*

[5] Viola, Jones (2001), "Robust Real-time Object Detection" - *Second Intl. workshop on statistical and computational theories of vision.*

[6] Intel (2000), "Intel's OpenCV Library - http://opencv.willowgarage.com/wiki/" - *Website.*

[7] Feraud, Bernier, Viallet, Collobert (2001), "A fast and accurate face detector based on neural networks" - *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

[8] Rajagopalan, Kumar, Karlekar, Manivasakan, Patil, Desai, Poonacha, Chaudhuri (1998), "Finding faces in photographs." - *Sixth International Conference on Computer Vision.*

[9] Morimoto, Koons, Amir, Flickner (2000), "Pupil Detection and Tracking Using Multiple Light Sources" - *Image and Vision Computing, Volume 18, Issue 4.*

[10] Lalonde, Byrns (2007), "Real-time eye blink detection with GPU-based SIFT tracking" - *CRV '07. Fourth Canadian Conference.*

[11] Zhu, Ju (2004), "Robust Real-Time Eye Detection and Tracking Under Variable Lighting Conditions and Various Faces" *- Journal of Information science and engineering.*

[12] Yuille, Hallinan, Cohen (1992), "Feature extraction from faces using deformable templates" *- Computer Vision and Pattern Recognition, 1989. Proc. CVPR '89., IEEE Computer Society Conference.*

[13] Feng, Yuen (2001), "Multi-cues eye detection on gray intensity image" *- Pattern Recognition, Volume 34, Issue 5.*

[14] Nixon (1985), "Eye spacing measurement for facial recognition" *- Proc. SPIE Applications of Digital Image Processing.*

[15] Pentland, Moghaddam, Starner (1994), "View-based and modular eigenspaces for face recognition" *- IEEE Conference on Computer Vision & Pattern Recognition.*

[16] Huang, Wechsler (1999), "Eye detection using optimal wavelet packets and radial basis functions (rbfs)," *- International journal of pattern recognition and artificial intelligence.*

[17] Kawato, Tetsutani (2002), "Detection and tracking of eyes for gaze-camera control" *- Proceedings from the 15th International Conference on Vision Interface.*

[18] Waite, Vincent (1992), "A probabilistic framework for neural network facial feature location" *- 2nd International Conference on Automatic Face and Gesture Recognition.*

[19] Reinders, Koch, and Gerbrands (1997), "Locating facial features in image sequences using neural networks" *- Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference.*

[20] Sirohey, Rosenfeld (2001), "Eye detection in a face image using linear and nonlinear filters" *- Pattern Recognition, Volume 34, Issue 7.*

[21] Kawato, Ohya (2000), "Real-time detection of nodding and head-shaking by directly detecting and tracking the between-eyes" - *Automatic Face and Gesture Recognition, 2000. Proc. Fourth IEEE International Conference.*

[22] Tian, Kanade, Cohn (2000), "Dual-state parametric eye tracking" - *Fourth IEEE International Conference on Automatic Face and Gesture Recognition.*

[23] Lucas, Kanade (1981), "An iterative image registration technique with an application to stereo vision" - *Proc. of the 1981 DARPA Image Understanding Workshop.*

[24] Cajochen, Brunner, Kräuchi, Graw, Wirz-Justice (1995), "Power density in theta/alpha frequencies of the waking EEG progressively increases during sustained wakefulness" - *Sleep.*

[25] Gevins, Leong, Dur, Smith, Le, DuRousseau, Zhang, Libove (1995), "Toward measurement of brain function in operational environment" - *Biological Psychology.*

[26] Akerstedt, Folkard (1994), "Prediction of intentional and unintentional sleep onset." - *Sleep onset. Normal and abnormal processes.*

[27] Horne, Baulk (2004), "Awareness of sleepiness when driving" - *Psychophysiology.*

[28] Hargutt (2003), "Das Lidschlagverhalten als Indikator für Aufmerksamkeits- und Müdigkeitsprozesse bei Arbeitshandlungen" - *Düsseldorf: VDI Verlag.*

[29] Papadelis, Chen, Kourtidou-Papadeli, Bamidis, Chouvarda, Bekiaris, Maglaveras (2007), "Monitoring sleepiness with on-board electrophysiological recordings for preventing sleep-deprived traffic accidents" - *Clinical Neurophysiology.*

[30] SeeingMachines (2006), "http://www.seeingmachines.com/product/faceapi/" - *Website.*

[31] Beta Face (2008), "http://www.betaface.com" - *Website.*

[32] Janssen (2009), "Literature Study: Vision Project" - *TU Delft, literature study.*

[33] Pantic, Rothkrantz (2000), "Expert system for automatic analysis of facial expressions" - *Image and Vision Computing Journal.*

[34] Datcu, Rothkrantz (2008), "Automatic bi-model emotion recognition system based on fusion of facial expressions and emotion extraction from speech" - *Automatic Face & Gesture Recognition, 2008. FG '08. 8th IEEE International Conference.*

[35] Gibbs, Poole, Stockmeyer (1976), "A Comparison of Several Bandwidth and Profile Reduction Algorithms" - *ACM Transactions on Mathematical Software (TOMS) .*

[36] Lu, Liu, Guo, Kong, Chang, Shan (2007), "Features of human skin in HSV color space and new recognition parameter" - *Optoelectronics Letters, Volume 3, Issue 4.*

[37] Horlings (2008), "Emotion Recognition using brain Activity" - *Proc. of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing.*

[38] He , Chia , Yang (2006), "A Geometric Invariant Approach to Human Face Verification" - *Journal of Information science and engineering.*

A Appendix

Summary. *This part of the Appendix contains all kinds of links, resources, terms and abbreviations used throughout the report. For a list of references see References.*

A.1 Links & Resources

Res. 1: Eclipse IDE – *http://www.eclipse.org*

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is often called Eclipse CDT for C/C++ development.

Res. 2: MPlayer - *http://www.mplayer.org/*

MPlayer is an open source movie player which runs on many systems. It plays most video formats.

Res. 3: Netcat - *http://netcat.sourceforge.net/*

Netcat is a featured networking utility which reads and writes data across network connections, using the TCP/IP and/or UDP protocol.

Res. 4: OpenCV - *http://opencv.willowgarage.com/wiki/*

OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision.

Res. 5: TruScan Exporer - *http://www.deymed.com/truscanqEEG.html*

TruScane Explorer is the software available at the TUDelft to work with the EEG scanner.

A.2 Terms & Abbreviations

Gloss. 1: AW

AW is the company where I did my Master Thesis project.

Gloss. 2: SDK

Software Development Kit

Gloss. 3: STN

Spatio-Temporal Neuron is a building block of the circuits in the perception processor.

A.3 Brain Computing Interfaces

Summary: *Article from wikipedia:*

A brain–computer interface (BCI), sometimes called a direct neural interface or a brain–machine interface, is a direct communication pathway between a brain and an external device. BCIs are often aimed at assisting, augmenting or repairing human cognitive or sensory-motor functions. Research on BCIs began in the 1970s at the University of California Los Angeles (UCLA) under a grant from the National Science Foundation, followed by a contract from DARPA. The papers published after this research also mark the first appearance of the expression brain–computer interface in scientific literature. The field of BCI has since advanced mostly toward neuroprosthetics applications that aim at restoring damaged hearing, sight and movement. Thanks to the remarkable cortical plasticity of the brain, signals from implanted prostheses can, after adaptation, be handled by the brain like natural sensor or effector channels. Following years of animal experimentation, the first neuroprosthetic devices implanted in humans appeared in the mid-nineties.

A.3.1 Invasive BCIs

Invasive BCI research has targeted repairing damaged sight and providing new functionality to persons with paralysis. Invasive BCIs are implanted directly into the grey matter of the brain during neurosurgery. As they rest in the grey matter, invasive devices produce the highest quality signals of BCI devices but are prone to scar-tissue build-up, causing the signal to become weaker or even lost as the body reacts to a foreign object in the brain. In vision science, direct brain implants have been used to treat non-congenital (acquired) blindness. One of the first scientists to come up with a working brain interface to restore sight was private researcher William Dobelle. The second generation device used a more sophisticated implant enabling better mapping of phosphenes into coherent vision. Phosphenes are spread out across the visual field in what researchers call the starry-night effect.

BCIs focusing on motor neuroprosthetics aim to either restore movement in individuals with paralysis or provide devices to assist them, such as interfaces with computers or robot arms.

A.3.2 Partially evasive BCI's

Partially invasive BCI devices are implanted inside the skull but rest outside the brain rather than within the grey matter. They produce better resolution signals than non-invasive BCIs where the bone tissue of the cranium deflects and deforms signals and have a lower risk of forming scar-tissue in the brain than fully-invasive BCIs. Electrocorticography (ECoG) measures the electrical activity of the brain taken from beneath the skull in a similar way to non-invasive electroencephalography (see below), but the electrodes are embedded in a thin plastic pad that is placed above the cortex, beneath the dura mater. ECoG technologies were first trialed in humans in 2004 by Eric Leuthardt and Daniel Moran. In a later trial, the researchers enabled a teenage boy to play Space Invaders using his ECoG implant.[26] This research indicates that control is rapid, requires minimal training, and may be an ideal tradeoff with regards to signal fidelity and level of invasiveness.

Light Reactive Imaging BCI devices are still in the realm of theory. These would involve implanting a laser inside the skull. The laser would be trained on a single neuron and the neuron's reflectance measured by a separate sensor. When the neuron fires, the laser light pattern and wavelengths it reflects would change slightly. This would allow researchers to monitor single neurons but require less contact with tissue and reduce the risk of scar-tissue build-up. This signal can be either subdural or epidural, but is not taken from within the brain parenchyma itself. It has not been studied extensively until recently due to the limited access of subjects. Currently, the only manner to acquire the signal for study is through the use of patients requiring invasive monitoring for localization and resection of an epileptogenic focus.

ECoG is a very promising intermediate BCI modality because it has higher spatial resolution, better signal-to-noise ratio, wider frequency range, and lesser training requirements than scalp-recorded EEG, and at the same time has lower technical difficulty, lower clinical risk, and probably superior long-term stability than intracortical single-neuron recording. This feature profile and recent evidence of the high level of control with minimal training requirements shows potential for real world application for people with motor disabilities.

A.3.3 EEG

Electroencephalography (EEG) is the most studied potential non-invasive interface, mainly due to its fine temporal resolution, ease of use, portability and low set-up cost. But as well as the technology's susceptibility to noise, another substantial barrier to using EEG as a brain–computer interface is the extensive training required before users can work the technology. Another research parameter is the type of waves measured. Birbaumer's research with Jonathan Wolpaw has focused on developing technology that would allow users to choose the brain signals they found easiest to operate a BCI, including mu and beta rhythms.

A further parameter is the method of feedback used and this is shown in studies of P300 signals. Patterns of P300 waves are generated involuntarily (stimulus-feedback) when people see something they recognize and may allow BCIs to decode categories of thoughts without training patients first. By contrast, the biofeedback methods described above require learning to control brainwaves so the resulting brain activity can be detected.

Attached electronic data

Below is a DVD with all materials of this thesis. It contains a PDF version of this thesis, as well as the final presentation. Additionally, all test data and videos are present on the medium. Finally, the GUI application developed for this thesis is available. The application runs currently under Linux only.