Utrecht University

Dimenco

Master Thesis

Accurate Eye Tracking For Autostereoscopic Displays

Author:
Erik Maessen

Student Number:
3848914

Supervisors:
Robby T. Tan
(Utrecht University)

Jan van der Horst
(Dimenco)

Thesis Number: ?????????

August 26, 2013

Abstract

Autostereoscopic displays have the potential to play a significant role in the multimedia industry in the coming years. Their big advantage is that the viewer does not need glasses to experience 3D. However, static view cones in which the 3D looks best, restrict the viewer to very specific positions in front of the display. For single viewer usage, Dimenco developed dynamic clear view, which allows the viewer to move freely in front of the display and still experience good quality 3D on each position. This is obtained by tracking the eyes of the viewer and adjusting the rendering of the 3D content accordingly. However, tracking the viewers eyes for this purpose means the tracker has to be fast enough to keep up with the content and accurate enough to retrieve the best possible viewers location. The speed required is 30 frames per second and this is particularly difficult because state of the art methods that obtain good tracking result, often lack speed and vice versa. Obtaining good accuracy from the eye tracker is a challenge because this tracker has to work under many different circumstances. This thesis presents an eye tracker that meets the requirements set by Dimenco in both speed and accuracy. These results are obtained by combining the state of the art face tracker introduced by Viola and Jones with a particle filter for fast face tracking, and by using the local minimum, adaptive gradient masks and isocenters to determine the pupil location. This combination of methods contributes to this field by running at at least 30 frames per second and by tracking as accurate as needed. The accuracy that is needed was determined by an experiment with an autostereoscopic display from Dimenco and for that accuracy this method outperforms other eye tracking methods to which it is compared to. Problems with eye tracking still remain when the viewer wears glasses, mostly when light reflection is visible on these glasses.

Contents

1 Introduction **3**
- 1.1 Motivation . . . 3
- 1.2 Goal . . . 3
- 1.3 Approach . . . 4
- 1.4 Related Work . . . 6
  - 1.4.1 Tracking Faces . . . 6
  - 1.4.2 Tracking Eyes . . . 8
  - 1.4.3 Closed source . . . 9
- 1.5 Overview . . . 9

2 Requirements **10**
- 2.1 Speed . . . 10
- 2.2 Conditions . . . 10
  - 2.2.1 Lighting changes . . . 10
  - 2.2.2 Occlusion . . . 11
  - 2.2.3 Transformations . . . 12

3 Face Tracking **14**
- 3.1 Viola Jones . . . 14
  - 3.1.1 Haarlike Features . . . 14
  - 3.1.2 Integral Images . . . 15
  - 3.1.3 Feature Selection . . . 17
  - 3.1.4 Cascade Classifiers . . . 18
- 3.2 Particle Filter . . . 19
  - 3.2.1 Setting the particles . . . 19
  - 3.2.2 Weighting the particles . . . 19
- 3.3 Fast Tracking . . . 21
  - 3.3.1 Random sampling . . . 22
  - 3.3.2 Weighting based on color . . . 22
  - 3.3.3 Comparison . . . 22

4 Eye Tracker **23**
- 4.1 Determine eye region . . . 23
- 4.2 Determine pupil location . . . 24

4.2.1 Local minimum . . . 25
4.2.2 Adaptive gradient mask . . . 25
4.2.3 Isocenters . . . 26
4.3 Cost function . . . 29

5 Implementation **30**
5.1 Tracker . . . 30
5.2 Switch faces . . . 31
5.3 Tilted faces . . . 31
5.4 Interface . . . 33

6 Results and Evaluation **34**
6.1 Face Tracker Results . . . 34
6.1.1 Speed . . . 34
6.1.2 Conditions . . . 35
6.2 Eye Tracker Results . . . 36
6.2.1 Speed . . . 39
6.2.2 Comparison . . . 41
6.2.3 Error Margin . . . 42

7 Conclusion **45**
7.1 Future Work . . . 45

Chapter 1

Introduction

This thesis project is conducted at Dimenco. A company in Veldhoven that makes autostereoscopic 3D displays. These displays allow the viewer to experience 3D without having to wear glasses. One can imagine that this might be the next big step in watching 3D and the applications are very broad. And because this technology is rather young, the research field is still wide open. This papers covers a computer vision project to track the viewer using a webcam. The tracking information is used for single viewer usage, so that this user is not restricted to static viewing position but can freely move around and still experience 3D.

1.1 Motivation

Proper eye tracking will be very important for the autostereoscopic displays that Dimenco makes. With good eye tracking, the 3D quality will improve as well. Of course, there are numerous applications for face and eye trackers, but the main motivation here is to develop a method that is specific for the autostereoscopic displays. The requirements for such a tracker are very specific and will have to be met, because there is a change of commercial use. This opens the possibility to research and compare existing methods and determine whether they are sufficient for this task and whether these can be altered or approved to better meet the ultimate requirements. This thesis shows this research and tries to come up with improved methods that meet the requirements of Dimenco as closely as possible.

1.2 Goal

The goal of this project is to research and develop an accurate and fast eye tracker that Dimenco can use for their autostereoscopic displays. The need for such an eye tracker comes from the fact that the lenticular technology they use, restricts the viewer to static positions in front of the display in which the 3D

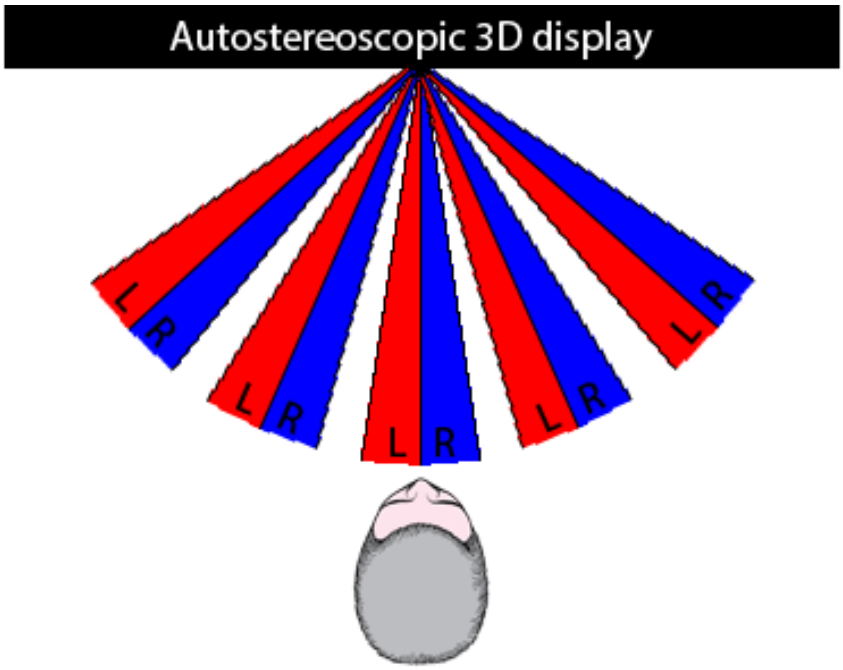


Figure 1.1: Cones

effect is optimal. These positions are called cones and can be seen in figure 1.1. This cone is split up into two parts. One for the left eye and one for the right eye. If these cone parts correspond to the viewers eyes, the 3D effect is optimal. Once the viewer moves out of the cone, the content on the display becomes vague and blurry, just like when you remove your 3D glasses during a movie. Dimenco overcomes this problem for single viewer usage by tracking the viewer in front of the display so that the rendered content can be adjusted according to the viewers position. That way the cone follows the viewer when he or she moves. To achieve such a tracker the project can be roughly split up into two parts. The first part is the face tracker, which follows the viewers general face position in a fast way. The second part is to find the eye position within the face region. How fast and how accurate the tracker has to be, and under which circumstances it should work properly, is stated in the requirements chapter.

1.3 Approach

The general approach is to first locate the face of the viewer and then, from within the face position, look for the exact eye positions. That means there are

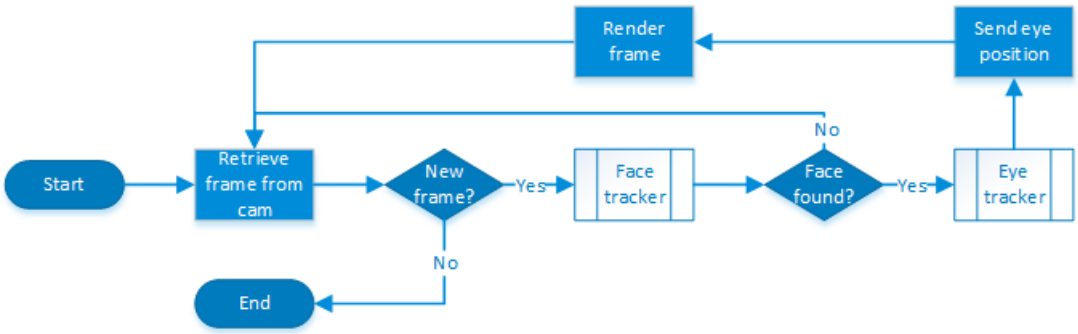


Figure 1.2: Total pipeline

two problems the solve which will both be shortly discussed below. In figure 1.2 the flowchart for the general algorithm can be seen.

Face Tracker The first problem, finding the face position, is solved by combining the Viola Jones approach for face detection [14] with a particle filter according to [9]. The Viola Jones approach uses haarlike features and a cascade classifier to find faces in an image. In OpenCV, an implementation of the Viola Jones algorithm is present but this method searches the whole image at many different scales for each frame and this significantly slows down the process. By using a particle filter and an adjusted implementation of Viola Jones, faces can be tracked much faster. First the particles will be set. Each particle is a rectangular section of the input image. This set of particles is then sent to the face detector, which checks each particle for a face. Positive particles that are close together will most likely represent the same face, so these rectangles are grouped together. This results in one rectangle per face. If multiple faces are found, the biggest is chosen for further tracking. More details about the particle filter and the face detector are in the face tracking chapter below. See figure 1.3 for a graphical impression of how the tracker works.

Eye Tracker If the face tracker has found a face, the rectangle that represents the face is send to the eye tracker. Within this rectangle the eye tracker searches for the pupil positions by first selecting the eye regions (roughly the top-left and top-right part of the face rectangle) and then looks for the exact pupil position in these smaller regions. The exact position is searched by using a combination of three different methods. First the eye region is scanned for isocenters [13]. An isocenter basically is the middle point of a circle like shape (for example the pupil). Secondly, a adaptive gradient center mask is used, because the pupil is usually in the middle of the eye. It is adaptive to take into account the small movements the pupil can make. The third and last method searches for the

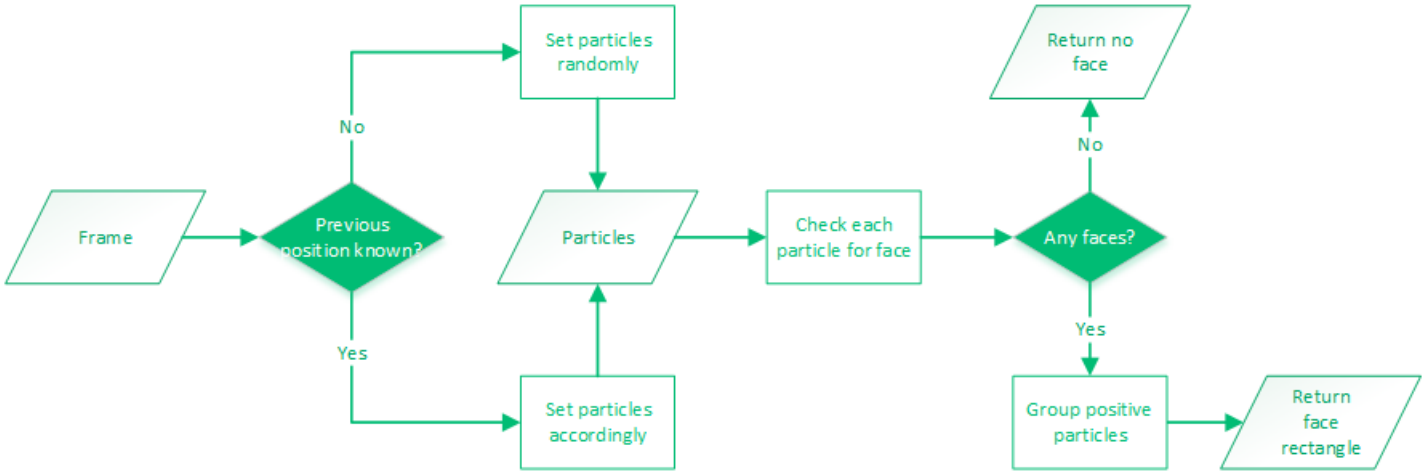


Figure 1.3: Pipeline For Facetracker

minimum in the eye region (which should represent the pupil of the eye). All these methods have their strengths and drawbacks, but together they form a very robust pupil detection algorithm. The outcomes of the three methods are analysed in a cost function that gives a cost to every position in the eye region. The region with the lowest cost is chosen as the pupil position. The pipeline for the eye tracker is shown in figure 1.4. A more detailed description about this eye tracker can be found in chapter 4.

1.4 Related Work

Computer vision is a very interesting field of research in which various methods are developed for face and eye tracking. This section looks at these methods and briefly discusses the strengths and drawbacks of these methods. The rest of this chapter is split up into two parts. One for face tracking and one for eye tracking.

1.4.1 Tracking Faces

Many different approaches for detecting and tracking faces are present in academic literature. Earlier methods often use color-based techniques to distinguish faces [5] [6] [2] [4], but these techniques are very sensitive for illumination changes or pose changes and have difficulty to cope with backgrounds that have skin-like color. Another problem with these techniques is that the skin color can vary significantly between different races. The positive side of these techniques is that they are computationally cheap and when the conditions are good, they

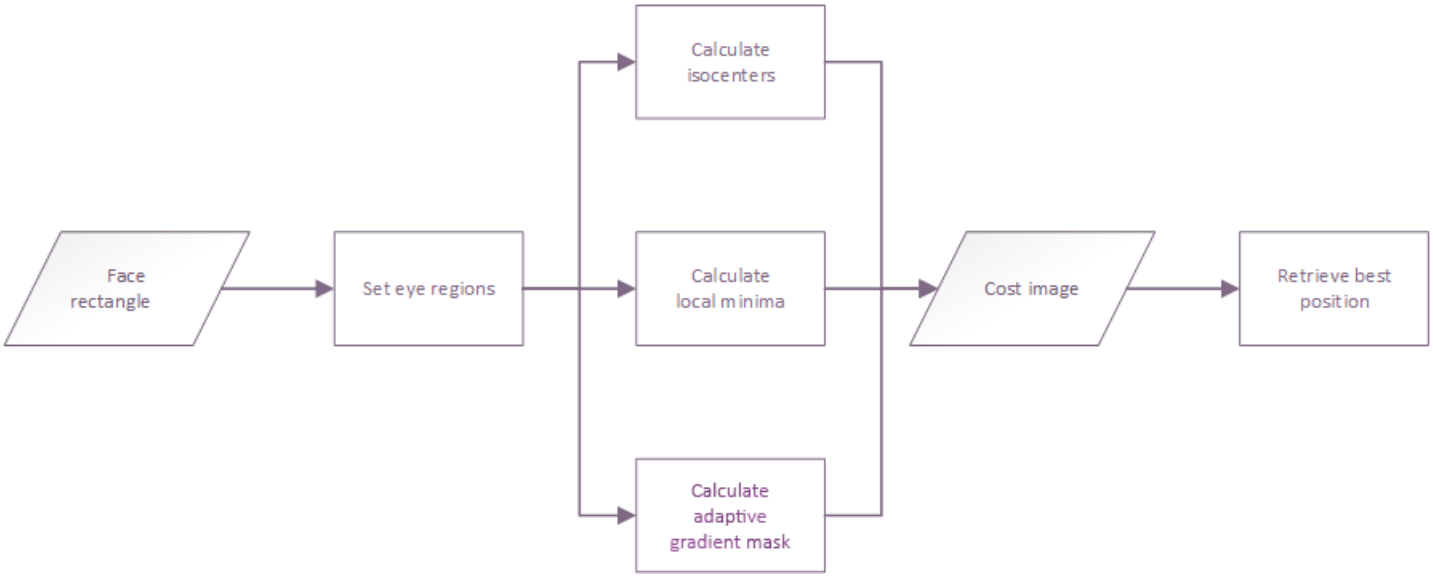


Figure 1.4: Pipeline For Eyetracker

provide good results, even with low resolution or bad quality images [15]. For this project, a color-based method is insufficient because the tracker needs to work in a lot of different situations described in the next chapter. Another type of face detection uses edge detection. A well known edge detection algorithm is the Canny algorithm. In [8], edges are used to detect faces. These kinds of methods are proven to work well if the background is not too complex. For scenes where the background is very complex, with a lot of edges, these types of detector can easily fail to recognize a face shape from all these edges. So edge based techniques are also insufficient for this project, because the background cannot be controlled. Methods that are very invariant to illumination changes or background changes are machine learning algorithms, but these often require an extensive training set and are computationally more expensive. Feature based approaches are methods that look for very specific features within the face, for example the eyes or mouth and use these to define the face. These methods are very dependent on the type of feature that is used and how easily these features can be computed and compared. For example, some papers such as [3] use template matching to detect features such as eyes or mouths. But eyes and mouths tend to change when a person talks or moves, and these are again very sensitive to illumination changes. On the other hand, Viola and Jones [14] use haar-like features which are easy to compute and easy to compare. This method is still seen as the state of the art in object detection. The haar-like features can express certain features very well and once trained properly, the algorithm

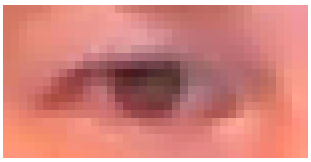


Figure 1.5: Magnified image of an eye

can detect objects very well. The drawback of this method is that is originally is too slow to apply directly in this thesis. Therefore, the algorithm is slightly altered and incorporated into a particle filter to speed it up. More details about the face tracker can be found in chapter 3.

1.4.2 Tracking Eyes

The first distinction that can be made in eye tracking is the kind of hardware that is used. A lot of methods rely on expensive infra-red cameras, or head mounted cameras. Infra-red cameras have the advantage that pupils tend to light up when shone on by infra-red light. This makes the detection and tracking of pupils a lot easier. The disadvantage of the infra-red camera's is that they are a lot more expensive then regular cameras. In this case, for autostereoscopic displays, the hardware used cannot be too expensive, so the aim is to use regular hardware, such as webcams. Another type of camera that is widely uses for pupil tracking are head mounted cameras, which monitors the eyes of the viewer from a very short distance. This way, the resolution of the eye is very high, and can easily be analysed. Head mounted cameras generally produce good results, but just as the infra-red cameras, they are more expensive, and more importantly, the ultimate goal of autostereoscopic displays is to be able to watch 3D without any extra accessories. So it would be rather strange to make a glasses free 3D dispaly and then put on a head mounted camera. The type of hardware used for this project is a simple webcam on a 640x480 resolution. One can imagine that this leaves rather low resolution images for the eye region. See figure 1.5 for an example of how pixelated an eye looks when the viewer is sitting approximately 1 meter from the display.

To deal with such images, a number of different approaches are used in academic literature. Just as with face detection, edge-based techniques are also applied on eyes. These can be able to neatly put an edge around the pupil and then use an ellipse fitting algorithm to determine the exact pupil location [1]. However, these methods tend to fail when the viewer is further from the screen, or for example when the viewer wears glasses. These are all conditions that have to be face when applied to autostereoscopic displays, so they are insufficient for this research. Some papers simply use a local minimum to determine the pupil location, because in most cases, the pupil is the darkest area of the eye region.

This is a very simple approach, but lacks robustness especially when the scene itself becomes dark, or the viewer has very dark skin color, dark eyelashes or make-up. The most popular method however, is finding circular like shapes within the eye region. Some of these approaches use blob tracking in which particular parts of the image that have the same color or intensity belong the one blob. These blobs are then analysed for any circular shapes. The main drawback of blob tracking is that it is color-based, so if the scene is very dark or the viewer wears glasses with a dark frame, these can easily be mistaken for a pupil. More complex approaches are also used to detect circular shapes. One of them is the use of Isocenters as in [13]. This method uses isophotes to detect circle like shapes. This is a very robust method and is one of the main approaches used in this project. The drawbacks of this methods are that it has the tendency to mistake the eye corners or the eyebrows for the pupil. It can also fail when the user wears eyeglasses. More on the evaluation of this method can be found in chapter 6.

1.4.3 Closed source

Before this research was done, Dimenco used another eye tracker that came from a third party developer. Details about the methods behind this tracker are unknown so it is very hard to analyse how the tracking is done. Dimenco was already quite satisfied with the performance of this tracker, but because it is closed source, Dimenco cannot further develop or customize this piece of software.

1.5 Overview

The rest of this paper is organized as follows. The next chapter is about the requirements of the system. Because it might be commercially used in the future, it is important that the system meets the requirements. The third chapter contains more details about the face tracker used in this research. The fourth chapter discusses the different methods in detail that are used in the eye tracker. The fifth chapter is an evaluation of the complete method and will compare the results of both trackers with other methods. The last chapter concludes this report.

Chapter 2

Requirements

This chapter covers the requirements of the system. That means the speed it has to obtain, the different conditions it has to work under and of course more general things such as the user interface.

2.1 Speed

The system has to run at atleast 30 frames per second to obtain the best result. If it will run any slower, the tracker will adjust the rendering too late and this affects the quality of the 3D. Since this research uses normal webcams, it is important to notice that most webcams can actually shoot on 30 fps but are set to drop the framerate when the scene becomes darker to let in more light. This can of course be switch off. Another important asset that has to be discussed here is the delay of the system, since too much delay can also cause bad quality 3D. Delay can come from the hardware itself, for example the USB. This type of delay is out of the scope of this project, since it has nothing to do with the actual tracking. On the other hand, delay can also be introduced by the software by using information from the past. This has to be taken into account when developing the eye tracker to make sure the delay stays to a minimum.

2.2 Conditions

The eye tracker has to have a good performance under different conditions that can occur in real-world use of the system. For this, a list of different conditions has to be determined and all the conditions on this list have to be tested. The results of the tests will be discussed in the evaluation chapter.

2.2.1 Lighting changes

Different illumination conditions have always been a problem with face and eye tracking. Between various lighting conditions, the face can appear very

differently. First, the colour of the face can change dramatically and secondly the contrast within the face can change as well. This depends mainly on what kind of light is emitted and what the light source is. Different types of light are: Daylight: This is probably the best kind of light because it usually is more ambient than other types, but it can also be directional. Cameras usually perform best under this kind of natural light. Office light: A lot of bright white lights from above. Home light: Usually semi-bright light that can come from different directions Screen light: The light that comes from the TV or computer screen. This light can have many different colours and is dependent on the content shown on the screen. The source of the light stays the same, namely in front of the viewer, so the face is lit from the front.

The source of the light is also important. So the above types of light have to be tested from different sources if applicable. Different sources are: Ambient: The light is everywhere around the viewer and does not visually effect the face of the viewer. Frontal light: When the light comes directly from the front, the viewers face can be highly effected by it. The face can gain a different color from this light and also the contrast and contours on the face can change dramatically. Light from above: When light comes from above, the viewers face will also be likely to change due to it. Mostly different shadows will be cast onto the face in this kind of lighting condition. Light from behind: This usually means that the viewers face will become darker and the contrast of the face will drop. Light from the side: When light comes sideways, it can change one half of the face more than the other. So there can be a lot of difference between the two halves. The system has to work under as much different lighting conditions as possible. So a tests have to be done with different types of light and different light sources.

2.2.2 Occlusion

When occlusions occur, parts of the face, or the whole face is obstructed by another object. Occlusion can be permanent or temporary.

Permanent Occlusion Permanent occlusions can for example be accessories that the viewer is wearing, such as glasses, jewellery, piercings, tattoos or some sort of headwear. But it can also be due to hair that falls down the face, maybe partially occluding the forehead and eyes, or facial hair such as moustaches or beards. It is very important that the eye tracker works with these kinds of occlusions, because one does not want the viewer to change his or her appearance just so that the eye tracker can be used. The first problem with these occlusions is to correctly track the face. Once a face is to extreme, for example very hairy and full of tattoos, it can vary too much from any face in the training set and it is likely that the face will not be detected. The second problem with these kinds of occlusions is to determine the exact location of the pupils. This is mainly a problem when the viewer is wearing glasses. The glasses can obstruct the pupil tracking in two ways. First, the frame of the glasses can vary much between viewers and this frame can have somewhat the same colour as the eyes. And

secondly, which is the biggest problem, is that the glass from the eyeglasses can reflect light. If that is the case the appearance of the eye behind the glass can be altered dramatically which can cause a lot of difficulty finding the exact location of the pupil. So in short the main difficulties are that the training data has to be sufficient to detect faces with many different appearances such as facial hair or accessories. And that due to reflections from the viewer's glasses, the pupil location can be hard to determine.

Temporary Occlusion These are occlusions where another object or a part of the viewer other than the face obstructs the face partially or fully for a relative short amount of time. For example when the user scratches his or her eye, so that the hand obstructs the face, or when another person walks in front of the main viewer. These occlusions are temporary so when the occlusion is gone, the eye tracker has to successfully keep tracking the same face as it was before the occlusion. While the occlusion is occurring, the tracker should track the eyes for as long as possible obviously, but at some point the occlusions can be too severe for the tracker to remain tracking the eyes correctly. At that point, it is important that the tracker does not immediately search for another face but remains stationary for a certain amount of time until the occlusion is over. If the occlusion takes too long, the tracker starts to find a new face.

2.2.3 Transformations

Transformations are when the face of the viewer changes due to movement of the viewer (or camera). The transformation conditions can be divided into 3 types:

Scaling This basically means that the viewer moves towards the screen or away from the screen. For autostereoscopic displays, the viewer is never too close to the screen so that the tracker fails. The biggest problem here lies in the maximum distance. The optimal viewing distance is highly dependent on the type of display used. A small display has a much smaller optimal viewing distance than a large screen. The tests conducted in this project are all done on a 21" screen. The optimal viewing distance for these screens is approximately 1 meter. For larger displays, such as a 50" display, the viewing distance is approximately 3.5 meters.

Rotation When the viewer rotates the face can move along different axes. For example looking to the right or left, or looking upwards or downwards. When this happens, the viewer is no longer looking at the display, so the position of the eyes is not that important when these kinds of rotations occur. On the other hand, when a viewer tilts the face, he or she can still be watching the content on the display, so the eye tracker has to be able to keep tracking the eyes. Since the 3D effect of the displays will disappear when the viewer tilts the

face for more then 45 degrees, the tracker has to be able to track the eyes up to 45 degrees of tilting.

Translation These transformations mean that the face is moving through the field of view of the camera, but the face itself stays the same, so no rotations or scaling or any other transformations. The eye tracker has to be able to track the viewer during any of these movements.

Of course, a combination of the above transformations can occur at the same time, so the tracker has to deal with that as well.

Chapter 3

Face Tracking

This chapter describes the method used in this research to detect and track faces. Two existing methods are combined to have robust detection and fast tracking. For detection the method of Viola Jones is used. This is a very robust object detection method that can detect any object if it has a training set. In OpenCV, a training data file for face detection with Viola Jones is included and will be used here. For the tracking part of the algorithm, a particle filter is used. The particle filter used originated from rob hess and is very flexible in its way of tracking objects. Both methods will be explained below as well as the combination of both.

3.1 Viola Jones

The method that Viola and Jones describe in their paper is still seen as one of the best object detectors around as it is invariant to a lot of different situations that will be discussed later on. It can be trained to detect any kind of object. The method Viola and Jones have introduced basically has three major contributions that make their method state of the art. The first is their introduction of the integral images, which allow the user to very quickly compute the rectangular features. The second contribution is the selection of a small number of important features using AdaBoost [7] and the third contribution is the use of a cascade-structure for the classifiers, which increases the speed of the algorithm dramatically. In this section the Viola Jones algorithm will be described in more detail.

3.1.1 Haarlike Features

The detector of the Viola Jones method uses features to detect objects of any kind it is trained for. The features it uses are based on the haar features used in papgeorgiou et al [11]. These are rectangular features which are divided into different parts. The first two features are so-called two-rectangle features and

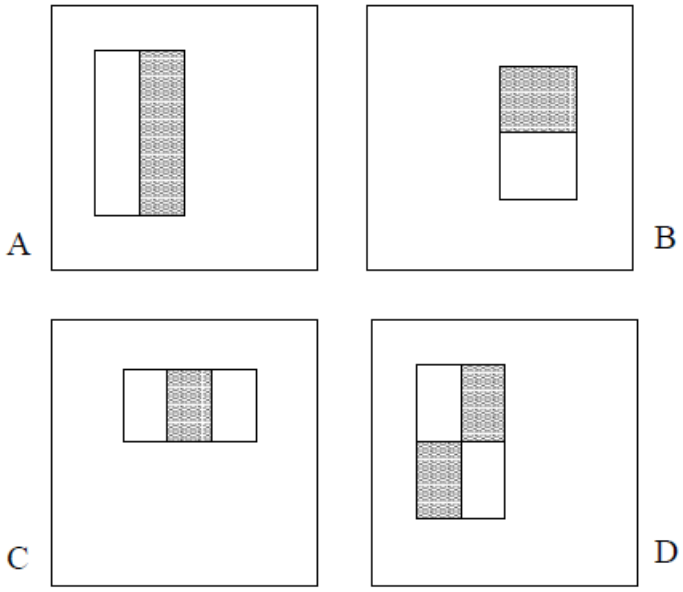


Figure 3.1: Rectangular features

the value of these features simply is the difference between the sum of pixels within two rectangles (see figure 3.1a) and figure 3.1b). As can be seen in the image, the rectangles have the same size and shape and should be horizontally or vertically aligned. The second feature is called a three-rectangle feature and its value is the sum of the pixels of the two outer rectangles subtracted from the sum of the pixels of the inner rectangle (see figure 3.1c). The last feature in the image is a four-rectangle feature and expresses the difference of diagonal pairs of rectangles (see figure 3.1d).

3.1.2 Integral Images

As mentioned in the previous section, the Viola Jones method uses rectangular features to detect objects. These rectangular features represent the difference of the sum of pixels of two or more rectangular areas in the image. One can imagine that adding up all the pixel values for all the features used is a very expensive task if it would be done normally. To make sure that the algorithm runs very fast, Viola and Jones have used a clever method of summing up the pixel values within an image. This method allows the user to compute the sums for the entire image only once, and after that, the user can simply access the correct summed up values required. This is achieved by using integral images. In an integral image, the value at x, y represents the sum of all pixels that are

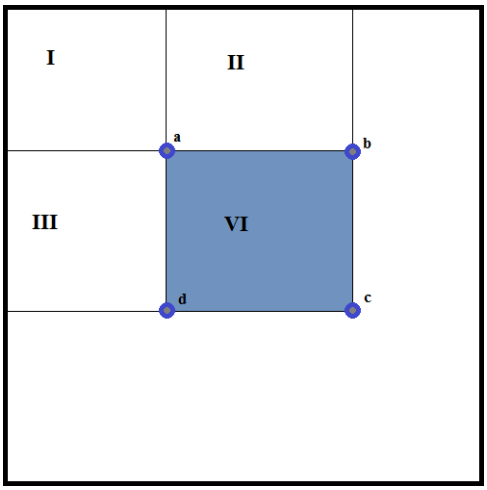


Figure 3.2: Example for calculating the sum of a rectangle using Integral Images

to the left of x and above y, including x and y themselves:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{3.1}$$

Here *ii(x,y)* is the integral image and *i(x,y)* represents the original image. As mentioned before, the integral image can be computed in one sweep over the original image using a cumulative row sum with the following two equations:

$$s(x,y) = s(x,y-1) + i(x,y) \tag{3.2}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{3.3}$$

Here $s(x,-1) = 0$ and $ii(-1,y) = 0$.

Now that the entire integral image is set up, it can be used to compute the values of the features. The first thing that has to be clarified is that the sum of one specific rectangle in the image can be computed with only four references to the integral image. An example of this is shown in figure 3.2. The sum of the pixels in the blue rectangle *VI* can be computed with only four references to the integral image. The value at position *a* (top-left corner of the rectangle that has to be computed) represents the sum of pixels in rectangle *I*. The value at position *b* represents the sum of pixels in rectangles *I + II*, the value at position *c* represents the sum of pixels in rectangles *I + III* and the value at position *d* represents the sum of pixels in rectangles *I + II + III*. All that needs to be done now is subtracting the sum of *b* and *c* from the sum of *a* and *d*: *d + a - (b + c)*. This same method can be applied to calculate the features discussed earlier.

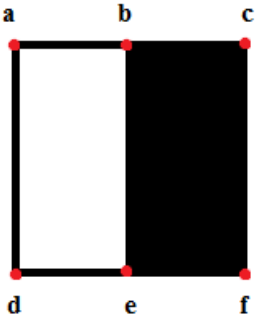


Figure 3.3: Here the value of this feature can be computed as follows: *(f + b - (c + e)) - (e + a - (b + d))*

For two rectangles, only eight references to the integral images are needed, but because the rectangles in the features are adjacent, two of these references are the same, so only six different values have to be looked up from the integral image to compute the *two-rectangle features*. In figure 3.3 this is graphically explained. For the *three-rectangle feature* a total of eight references is needed and for the *four-rectangle feature* a total of nine reference is required to get the required result.

Viola and Jones have shown with this method that the computation of rectangular features can be done very quickly. The pixels of the input image only have to be summed up once to calculate any feature on that image.

3.1.3 Feature Selection

Now that the features have been declared and the integral images allow for very fast computation of these features, the next step in the process is to choose which features should be used. This is necessary because there are a huge amount of feature to choose from and computing them all would drastically slow down the algorithm. Because the Viola Jones method uses a base patch of 24x24 pixels, the total number of possible features would be 45.396. To select only the most important features from this huge set, Viola and Jones use a feature selection process with AdaBoost learning to select a small number of features. This training comes up with a number of important features that describe a face in the best way possible. Figure 3.4 shows the two first features that Adaboost

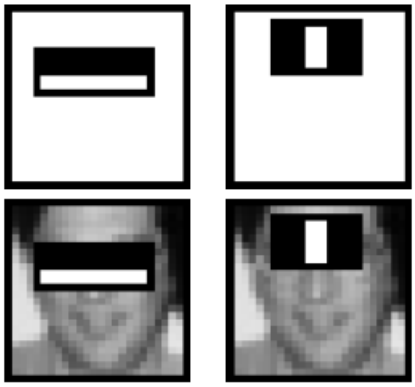


Figure 3.4: The two first features chosen by AdaBoost

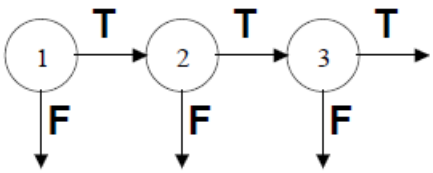


Figure 3.5: Cascade classifiers

has chosen. The pictures in this image are 24 by 24 pixels and the feature in the first picture resembles the difference of intensity of the eye region and the region of the upper cheeks. Often, the eye regions is darker than the upper cheek region of a face. The second picture resembles the difference of the intensity of the eyes compared to the intensity of the nose bridge, which often is a lot brighter.

3.1.4 Cascade Classifiers

The next step is to use these chosen features on an input image. To quickly decide whether the input image contains a face or not, Viola and Jones have made a set up where the classifiers are in a cascade structure. If the first classifier classifies the input as not a face, that means that the process for that input is directly cut off. This way, at an early stage, a lot of negative inputs is directly discarded and only possible positive inputs are processed further. Only the input that manages to go trough each and every classifier, is labelled as a face. Figure 3.1.4 shows the schematic representation of the cascade classifiers.

3.2 Particle Filter

A particle filter is a very commonly used method for object tracking. What a particle filter basically does is looking for an object in a number of defined locations (particles) that have some correlation with the previous frame. Rob Hess has developed a particle filter that was partially used in this research to do the face tracking. This particle filter will be described below and consists of a few different parts. The first part is how to chose the location of the particles and how to distribute them. The second part is how to determine for each particle if it is a face or not, the actual weighting of the particles.

3.2.1 Setting the particles

Particles are normally set according to some information in the past, but for the initial frame, there is no previous data. Therefore the first set of particles will be set randomly in the input image. These particles are then weighted and resampled. The weighting tells us how much that particle resembles a face and different kinds of weighting will be explained in the next part. The resampling is done according to the weights. First the particle with the highest weight (particle that most resembles a face) is chosen. This weight is then multiplied by the number of particles to get the number of particles that will be resampled at the location of the best previous particle. Then the next best particle is chosen, and again multiplied by the weight and so on... This way, in the next frame, a lot of particles are at the same location (namely the best location of the previous frame). This way, these particles have the best chance of getting a high weight in the next frame. Figure 3.6 shows a graphical example of how this weighting and resampling works.

The example shows the initial situation where the particles are set up randomly across the input image. The second image shows the weights of the particles (The size represents the weight). The third image shows the resampling done on the previous particle set, here the number in the particles resembles how many particles are in that location. Now these particles can be weighted again.

3.2.2 Weighting the particles

The weighting of the particles basically is figuring out how much that particle resembles a face. The particle filter that Rob Hess created uses color histograms to determine the weight. In the first frame of the videosequence, the user has to select an area that needs to be tracked. The particle filter then creates a color histogram of this region which acts as a reference. Then for the first frame the particles are set randomly across the image and the weighting begins. The weighting simply is creating a color histogram for each of the particles and compare these with the reference. The resemblance values are then normalized and that determines the weight for the particle. So what Rob Hess uses in his method, is a color based detection method, but basically any object detection

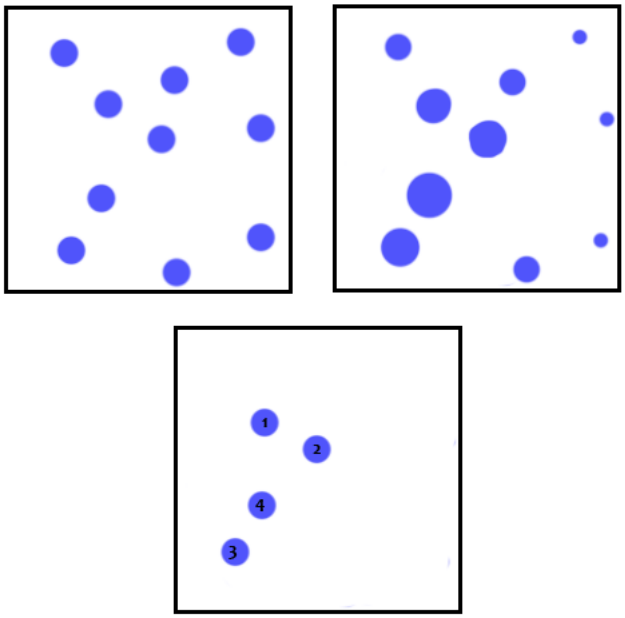


Figure 3.6: Example of particle filter

method can be used here. Color based detection is proven to be very sensitive for intensity and lighting changes in the scene so for this research, the color based face detection is not applicable because the scenes that this research has to work with are very likely to change a lot in intensity and color.

3.3 Fast Tracking

Inspired by the robustness of the Viola Jones object detector and by the flexibility of the particle filter, this research combines both method to obtain fast and robust face tracking. The basics of the particle filter are used but instead of using the color based histogram comparison method, an adaption of the Viola Jones method is used to determine for each particle if it is a face or not. In the first frame, the normal Viola Jones algorithm is applied to find a face. The normal Viola Jones looks for faces in the entire image and on every scale, so if this would be done for each frame, it would simply not be fast enough. After this first frame, the face position is known. Then the particle filter starts working. It sets it's particles according to the face position in the previous frame. But Viola Jones does not result in any weights, so the particles in this second frame are set randomly with a gaussian distribution around the previous face position. Because faces move much more in the horizontal direction then in the vertical direction, the standard deviation of this distribution is larger in the horizontal direction then in the vertical direction. So now the particles are set up around the previous face position. A particle in this case, basically is a rectangle in which a possible face can be found. To address the change in distance from the viewer (scale invariance), the particles are set up in three scales. The first scale is the same as de size of the rectangle of the previous found face. The second scale is 10% bigger than the first one and the third is 10% smaller than the first one. This allows the tracker to adjust for distance changes of the viewer. The next step is to check for each particle whether it is a face or not. Instead of using the color histograms from Rob Hess' implementation, here an adaption of Viola Jones is used. As mentioned earlier in this section, Viola Jones looks for objects in the entire image and at a lot of different scales. This drastically slows down the process. All that is needed here is that Viola Jones looks for objects in the particle locations(which are only a few hundred at three different scales). The OpenCV implementation of Viola Jones was slightly altered so that the search location could be specifically set. So now all the particles are checked for a face and all particles that contain one are returned as positive. The next problem is that this method does not return any weights for the particles. It either is a face or not. So the particles cannot be resampled the same way they were in the implementation of Rob Hess. At this point, two different solutions were tried and compared.

3.3.1 Random sampling

The first solution is to combine all positive particles into one, to get the best face positions and to simply skip to the next frame and sample the particles as if we had no previous information except for the previous face location. The particles are again sampled randomly using the same gaussian distribution as before. This resulted in a very fast and robust face tracker, that despite it was not weighted, could still track very fast movements.

3.3.2 Weighting based on color

The second method tried as a solution was to make a color histogram of each particle, just as in the original implementation, but only use it for resampling the particles. That way, in the next frame, there were a lot of particles on the previous face position and so a lot of particles returned as positive. The only drawback of this method was that it was drastically slowed down. Not due to the fact that it had to calculate a lot of extra histograms and compare them, but due to the fact that the Viola Jones algorithm simply found a lot of faces. As mentioned in the section about Viola Jones, a possible face location goes through a cascade of classifiers. If there are a lot of faces, the whole cascade has to be traversed a lot of times. So this solution turned the strong point of Viola Jones into a weak point.

3.3.3 Comparison

Based on the result described above, it is logical that the first solution was applied to the final implementation. It did not show any drawbacks for not resampling the particles according to a weight. By arranging the particles in a more horizontal way, the horizontal movements of the viewer were easily picked up and processed.

Chapter 4

Eye Tracker

The goal of the eye tracker is to locate the pupil of the viewer. Remember that the face region is already known, so the next step for the system is to look for the pupil location within this face region. In the introductory chapter, a pipeline for the eye tracker was presented. The first step in this pipeline is to obtain the rough eye region from within the face region. After that, a number of three different methods is used to determine the best possible pupil location. These three results are put into a cost function which then returns the best guess for the pupil location.

4.1 Determine eye region

So the first step in the eye tracking algorithm is to determine the rough region of the eyes. Since the face region is already known, it is quite easy to extract the eye region from it. Some papers simply use the top-left and top-right quarters as the regions for the right eye and the left eye respectively [3]. Others use more complex methods such as Viola Jones, but with eye training data instead of face training data (this eye training data is also available in the OpenCV implementation). The first try for this research was also the Viola Jones method and while it worked quite well, the training data did not include any sufficient support for eyes with glasses, so once a viewer with glasses entered the scene, the eye regions often proved hard to find. This could be solved by manually training the system, but that required a large training set of eyes with glasses. Another downside with this method is that it is rather slow. The better solution in this case was to try the first method, by simply taking the top-left corner. This however turned out to be even worse. Although the eye region was know in every frame, it was simply to big to do any good pupil detection. The eye region contained a lot of hair from the user and also a piece of background. These are both things that can easily mess up the pupil tracking. The obvious next step is to reduce the size of this eye region so it only roughly shows the eye of the users. And by taking into account that the average eye distance is

Figure 4.1: A set of images where the rough eye region is correctly determined

6.8 centimeters, this method proved to work quite well. For different viewers, the rough eye region is chosen correctly (see Figure 4.1).

4.2 Determine pupil location

Once the eye regions are known, it is time for the pupil detector to determine the pupil location. As input, the eye region image is used. This input image is then send into three different methods that all return a cost image. These cost images are combined to come up with the best possible location. The three methods of use are the local minimum, isocenters and the adaptive gradient mask. The inspiration for these three methods came from the properties a pupil has. First, a pupil is dark (black), so the intensity value of a pupil is very small (hence the local minimum). Second, a pupil is round (hence the isocenter method). And third, when a viewer is watching content on a screen, the pupil of that user is mostly is the center of the eye and can only move in a rather

Figure 4.2: A grayscaled an blurred eye image

contained area (adaptive gradient mask).

4.2.1 Local minimum

The first method is the local minimum. This is simply determining which part of the eye region image is the darkest. As a stand alone method it is already quite promising, but can be erroneous when the viewer has very dark eyebrows, or glasses with a very dark frame. The same also applies for scenes with very low light and shading. This method has the eye region image as an input. This image is then converted to a grayscale image to be able to obtain the intensity values and the image is blurred to reduce noise. That is basically all this method does. See figure 4.2 for an example.

4.2.2 Adaptive gradient mask

The adaptive center gradient mask method takes into account that the pupil of a viewer is mostly in the center of the eye and it is made adaptive so it can react to the movements of the pupil. It is basically a circular gradient image, that is dark in the center (low cost) and light at the edges (high cost). Of course, the pupil is not always exactly in the center, especially when the viewer is not sitting straight in front of the display, the viewers head is slightly rotated or more importantly, when the viewer looks at content displayed on different corners of the screen. To handle such situations the gradient mask image shown in the image has to be adaptive. That means it has to be able to adjust according to the situation. To make it adaptive, the center mask uses information from the previous frame. So if the pupil was off center in the previous frame, the changes are big that it will be off center in the current frame as well. The center of the gradient will be the previous pupil location. Because the image is a circular gradient, when the pupil changes its relative location, the gradient image can still pick this up and correctly adjust the mask again. Figure 4.3 shows the adaptive center mask for different pupil locations. This method did not come from any literature, but was experimentally used and proved to work very well. It can be best looked at as a restriction for the local minimum, so that it can not so easily suddenly jump toward a dark location other then the pupil.

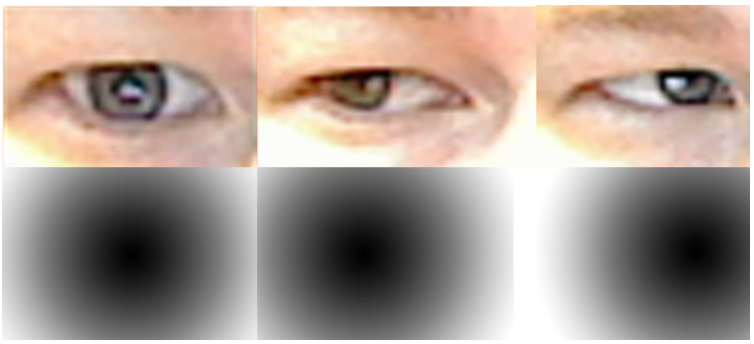
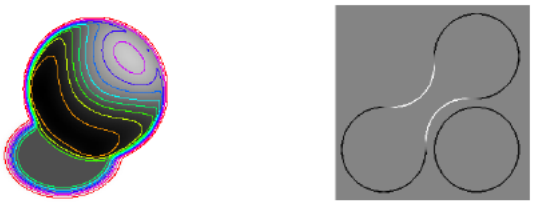


Figure 4.3: The adaptive center mask



(a) (b)

Figure 4.4: Isophote images

4.2.3 Isocenters

The third, and most sophisticated method is the detection of Isocenters. The paper of Valenti and Gevers [13] introduced this method and is very promising for detecting the pupil location. This method uses isophotes as a basis. An isophote is a curve in an image that has the same intensity all around (see figure 4.4a).

From this isophote, the isophote curvature can be computed. The curvature of another shape can be seen in figure 4.4b. On the positions where this image is white(negative), the curvature is outwards, where the image is black(positive), the curvature is inwards. First lets discuss how this curvature is calculated. With the use of the intrinsic geometry the curvature K can be ultimately expressed as follows:

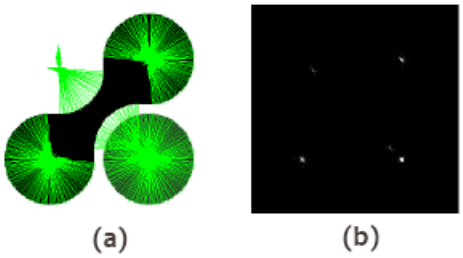


Figure 4.5: a) dispacement vectors, b) centermap

$$K = -\frac{L_y^2 L_{xx} - 2L_x L_{xy} L_y + L_x^2 L_{yy}}{(L_x^2 + L_y^2)^{3/2}} \tag{4.1}$$

Where L_x is the first order derivative in the x dimension and L_y is the first order derivative in the y dimension. This curvature equation is then reversed to get the radius of the circle that provided the curvature in the first place. This reversed equation is multiplied with the gradient to determine the orientation of this radius, because the gradient points towards the direction with the greatest change in luminance. Remember from the curvature image that a positive curvature means we have a curve that is light on the outside and dark on the inside. That is exactly like a pupil, so we only take into account the positive values here. The equation then looks like this:

$$D(x,y) = -\frac{\{L_x, L_y\}(L_x^2 + L_y^2)}{L_y^2 L_{xx} - 2L_x L_{xy} L_y + L_x^2 L_{yy}} \tag{4.2}$$

Here *D(x,y)* is a set of displacement vectors that point towards the estimated positions of the centers. In figure 4.5a can be seen how these vectors point towards all circular centres in this example image. Figure 4.5b shows the so-called centermap for that image. The bright spots in this image represent positions with high likeliness to be a center of a circle.

When this method is applied to any real world scenarios, it does not produce the same perfect results shown in the previous images. That is because the used examples are very simple and most of all very gradual. If we look at the isophote image (Figure 4.4a) we see that there are multiple isophotes and each smaller one, is more or less a scaled version of another isophote. So when this method is applied, all these isophotes have a weight in the final estimate. However, in real world scenarios, when tracking eyes, the input images are not as simply shaped and gradual. That means that the isophote image will look very different and if all isophotes are taken into account for the final estimate, the displacement vectors for all these isophotes will point in a lot of different directions. Therefore, Valenti and Gevers came up with a additional method that determines which

Figure 4.6: The curvedness of an example image

Figure 4.7: A real time centermap of an eye (left). A blurred version of the same centermap (right)

displacement vectors will be used for the final centermap. For this, they use the so-called curvedness of an image. The curvedness is calculated as follows:

$$curvedness = \sqrt{L_{xx}^2 + 2L_{xy}^2 + L_{yy}^2} \tag{4.3}$$

The curvedness detects sharp edges in an image. An example of the curvedness can be seen in figure 4.6. This image shows that the curvedness is high in places where the isophote density is high as well (see figure 4.4a). So now this curvedness value can be used to select which displacement vectors are used in the final result.

Figure 4.7 shows a real time centermap image of an eye(left) and the blurred version of that same image(right). This shows that it is necessary to blur the images after doing the centermapping. Because an eye is very complex and because of noise, the original centermap image looks very distored.

The obtained centermap finally is inverted, so that it are the dark areas that most resemble the pupil. This is done so it corresponds to the other two techniques when adding up the three.

4.3 Cost function

The cost function receives three cost images from the three methods and adds them up and normalizes them. Adding them is is done according to certain weights that are experimentally determined.

Chapter 5

Implementation

This chapter talks about the implementation of the tracker and how the face and eye tracker are combined into one system and how to deal with very specific problems such has multiple viewers, or what to do when a face is lost. The tracker is implemented using C++ and the OpenCV libraries to do the computer vision part. OpenCV is chosen because it has all the state of the are computer vision techniques that are required for this project. The main class of the implementation calls the tracker and once it retrieves the information it needs (pupil location), it gives this information to the Dimenco library. This Dimenco library then does all the rendering so that the content on the screen looks 3D for the given eye location.

5.1 Tracker

The first thing to discuss are the different states the tracker can be in. Below is a list of states and an explanation.

FACE This state means there is a known face

FACE_NO This state means there is no face and a new face has to be found

FACE_NEW This is the state the tracker enters when the user pressed the 'n' key. This key allows the user to switch between faces when multiple viewers are in the scene. The tracker looks for a new face on every location except the location of the old face.

FACE_WAIT This is the state the tracker enters once it loses a face due to an occlusion for example. The tracker then looks for a face in the previous know location for 2 seconds before it looks for a new face in the entire scene. This state is implemented because it can occur that the viewer for example scratches his or her nose and the tracker loses the face due to the hand occlusion. When the viewer removes the hand, the tracker has to be able to quickly respond and refind the same face.

The initial state of the tracker obviously is FACE_NO. The tracker tries to detect any faces in the first frame. If it does not find any faces, it simply stays in the same state and retries in the second frame. Once the face tracker finds a face, it switches to the FACE state and the eye tracker is called to determine the pupil location from within the found face. The eye tracker will always produce a result. This result is then send back to the main class. Once the face tracker loses the face, the state FACE_WAIT is entered. As mentioned in the state description, this state will stay at the same searching position for 2 seconds. When no face is found within the 2 second window, the tracker goes to FACE_NO and looks for faces in the entire scene again. If it does finds a face within these 2 seconds the tracker goes back to the FACE state and continues.

5.2 Switch faces

When the user presses the 'n' key on the keyboard, the tracker goes to the FACE_NEW state and should switch to another face in the scene. It looks for a new face in the entire scene except for the old face location. This is done by using a mask. OpenCV allows masks to be used on a lot of methods. The mask basically is a binary image, that tells the tracker where in the scene it can (white) or cannot (black) look for new faces. So once the user presses 'n', a black square is placed on the mask where the old face location was. This black square stays there for 2 seconds before it disappears again. That means that the tracker has 2 seconds to find another face, if it fails to, the tracker switches back to the old one. This setup works well for two faces, but it can occur that more faces are in the scene and the user wants to be able to switch between all faces, and not just two. Therefore, each 'old' face has its own counter of 2 seconds. Lets say there are three faces A, B and C. The tracker is following face A, when the user presses 'n'. The tracker then puts a black square on face A for 2 seconds and finds face B. If the user then presses 'n' again within 2 seconds, face A will still be masked out, and face B will now also be masked out for 2 seconds. This allows the tracker to switch to face C. After 2 seconds, all three faces are again available to be tracked again. So by giving each face its own timer, the user is able to switch between more than two faces.

5.3 Tilted faces

The tracker also has to be able to track tilted faces. But since the Viola Jones algorithm used here is only trained for upright faces, the tracker loses the face when it is tilted for more then 20 degrees (See figure 5.1). As stated in the requirements, the tracker has to track faces that are tilted up to 45 degrees.

To address this problem, the tracker uses the previous eye position to determine the degree of tilting. If the left eye is slightly higher then the right eye, that means that the face is slightly tilted. The degree of tilting is calculated from this and this degree is then rounded up.

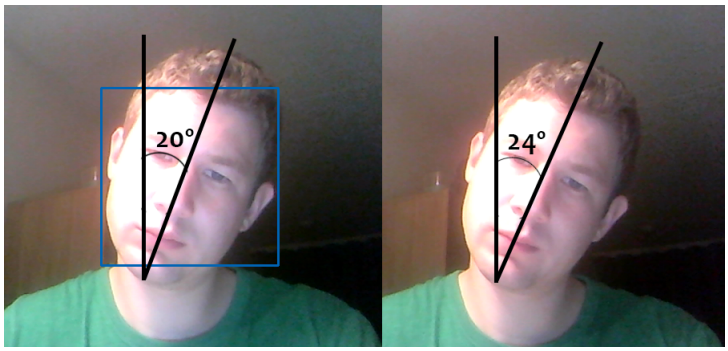


Figure 5.1: The angle of tilted faces

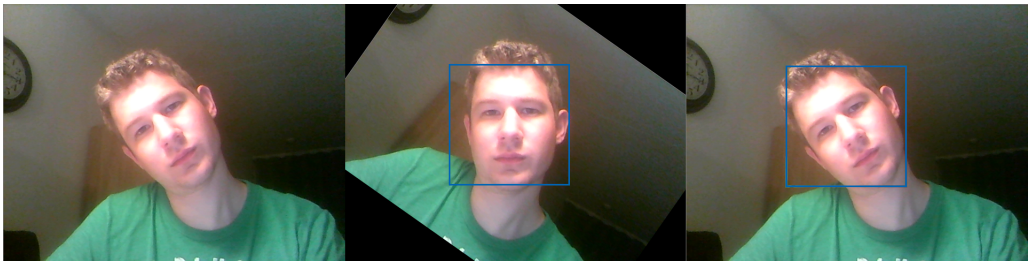


Figure 5.2: A tilted face is first rotated against the direction of tilting. Then the face tracker can detect the face. Finally the rectangle is backprojected onto the original image

This rotation value is then used to rotate the input image for the face tracker. If the rotation value is 15 degrees, the input image will be rotated -15 degrees (or 345 degrees) to adjust for the tilting. That way, the face will always appear almost upright for the face tracker. Because small steps are used for the rotation (1 degree), fast tilting movements are picked up and processed correctly. The result section in the evaluation chapter will show this. Any tracking data the face tracker returns at this point, has to be back projected onto the original image after the tracking. See figure 5.2 for an example of how the tilting is done.

This tilt fix helps the tracker to track faces up to even 90 degrees. Even although a face that is tiled at 90 degrees cannot see any threedimensional content anymore, it is still important to be able to track the viewer, so when the viewer tilts the face back upright, the tracker maintains the viewers face.

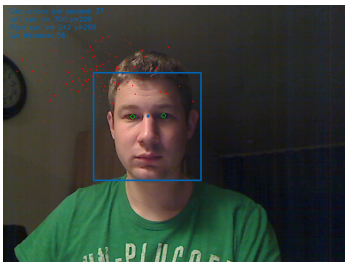


Figure 5.3: A printscreen from the eye tracker

5.4 Interface

The interface of the implementation is also built to the requirements of Dimenco. The screen shows all information required in the top left corner. This information is the speed of the algorithm, the eye position and the eye distance. The pupil locations are also drawn on the face of the viewer so the accuracy can be easily observed. This information can also be switch on and off by pressing certain buttons. The user can also display other information by pressing buttons such as the face rectangle and the particle cloud. These are used for debug purposes or demonstration purposes. The table below shows a keymapping on how to control the tracker.

Key	**Action**
E	Toggle pupil location
I	Toggle information text
F	Toggle face rectangle
P	Toggle particle cloud
N	Switch tracking face
Y	Lock tracker
arrow keys (in locked mode)	move the eye position
-, + (in locked mode)	increase or decrease the eye distance

As can be seen in the keymapping table, the button 'y' lets the user lock the tracker. Locking the tracker means that the tracker does no longer track the eyes but it locks the last known eye position and uses that to send to the renderer. This locking of the tracker is built in so that the whole system can easily be configured. Once in lock mode, the user can use the arrow keys and the minus and plus sign to adjust the eye position. During configuring, it is important that the eye position is as stable as possible. Figure 5.3 shows a frame of the tracker with all information turned on.

Chapter 6

Results and Evaluation

The evaluation chapter shows all results and compares these with other methods. First the face tracker will be evaluated, and secondly the results for the eye tracker will be analysed.

6.1 Face Tracker Results

Here some result are presented from the face tracker discussed above. The first result to show is the speed of the face tracker. After that some results are shown for different conditions discussed in chapter 2.

6.1.1 Speed

Of course the face tracker has to be fast enough to do 30 frames per second, but it also has to leave sufficient room for the eye tracker. If the face tracker is already at its limit, there will not be enough left for the eye tracker to work within these 30 frames per second.

Tests show that the face tracker can perform well within the bounds of 30 frames per second. One important variable here is the number of particles used. More particles means better tracker for fast movements and less jitter, but obviously costs more computational power. When the number of particles is reduced, fast movement is tracked less good and the jitter can go up, but the computational speed is much greater. Figure 6.1 shows a graph in which the framerate is compared with the number of particles. The vertical red threshold line indicates the lower boundary for when the number of particles is so low that the tracker cannot track fast movements any more. The system used in this test is an second generation Intel i7 2.0GHz quad-core notebook.

This graph tells us that for a number of 200 particles the fast movement is still picked up by the tracker and it achieves a framerate of approximately 47 fps. Even when 750 particles are used, the tracker still runs above 30 fps. The conventional Viola Jones Algorithm on the same system runs at approximately 6

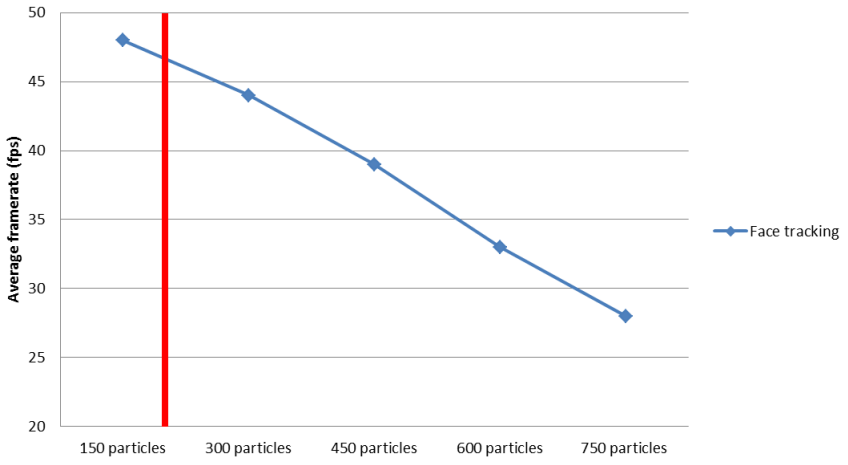


Figure 6.1: A graph that shows the speed of the algorithm according to the number of particles used

frames per second. So the increase in speed is significant. whether this framerate is high enough for the eye tracker, will have to be determined in the next section. The other requirement in the speed section was the amount of delay introduced by the software. This face tracker obviously uses information from the past (to resample the particles), but because the particles are spread out around this previous face location and the actual information is not used for detection, it does not influence the delay in any way.

6.1.2 Conditions

Here some result are posted from the face tracker. These results cover the situations described in the requirements chapter.

Illumination Figure 6.2 shows three images in which the first face is normally lit, the second face is lit very much and the third face is barely lit. In all three scenario's the tracker is still able to follow the face.

The images in figure 6.3 show various directions of light shining onto the face. The tracker shows to have some trouble in determining a face when a face is unevenly lit. The illumination from the side proves to be too much to handle once it passes a point where one side of the face looks too different from the other side. In the image where the face cannot be tracked anymore, the right side of the face is barely visible. The same applies to the images where the

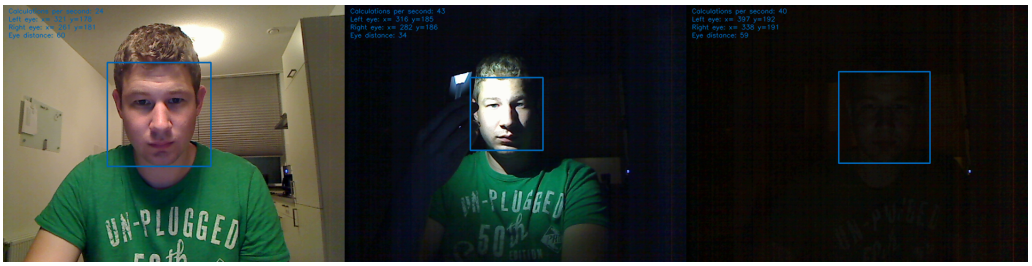


Figure 6.2: Different amounts of face brightness

light is shun from above. As soon as the face is too unevenly lit, where only the nose still receives light, the tracker fails. Remember that the second feature from Viola Jones looked at the difference between the eye regions and the nose bridge. In these images this difference is too great to be labelled a face.

Occlusion Figure 6.4 shows some permanent and temporary occlusions where permanent occlusions being sunglasses or hats and temporary occlusions being hands or other objects. Accessories such as glasses are handled perfectly by the face tracker. Even the face with the sunglasses is still recognized as well as the face that is largely occluded by a drinking glass. The occlusion with the hand shows that the tracker can handle these types of occlusions up to a certain point. In the image where the face could not be tracked anymore, the hand is just covering up too much of the face to still be recognized.

Transformation Figure 6.5 shows a number of images with different kinds of transformations such as scaling and rotating in different directions. As these images show, the tracker can handle the transformations very well. The maximum distance for the tracker is approximately 3 meters (dependent on the situation), so that is not quite enough for the large displays at Dimenco. After 3 meters the tracker often loses the face. This is purely a resolution issue, because the training data for faces used in this project are trained at 24x24 pixels. So faces smaller than that, or even faces that come near that size are not or barely recognized. And of course, the smaller the image is, the more it suffers from motion blur. When higher resolution images are used for the tracker, the maximum distance will be increased. However, higher resolution means more computation and less speed.

6.2 Eye Tracker Results

This sections discusses the results of the eye tracker. The eye tracker is mainly tested by using the BioID testset. This is considered a difficult data set that

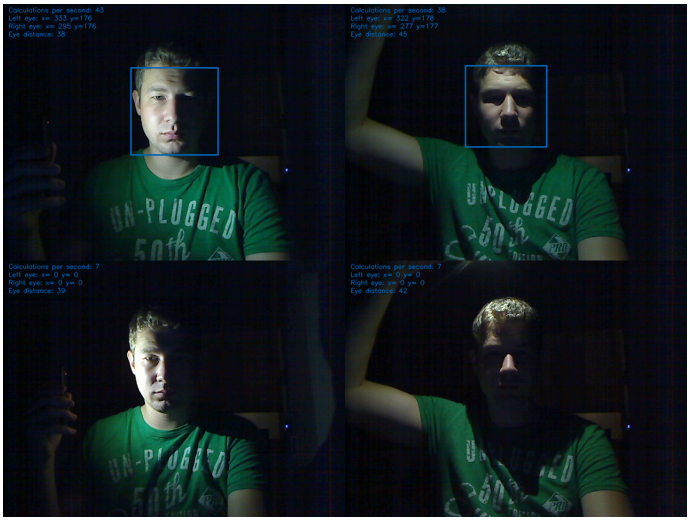


Figure 6.3: Different amounts of face brightness

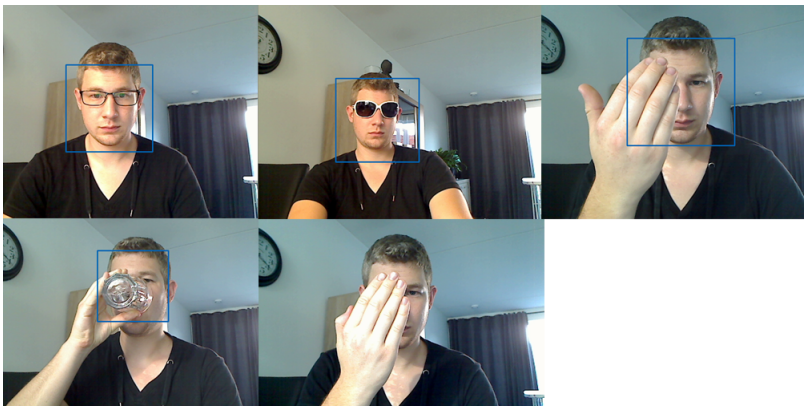


Figure 6.4: Different types of occlusion

Figure 6.5: Different types of rotation

is widely used for facial feature detection testing. The test set consists of 1521 grayscale images of 23 different subjects at a resolution of 384x288 pixels. It is considered difficult because the pictures were taken at different times and under different illumination conditions. Also the scale and pose of the subjects vary widely. Finally, many subject are wearing glasses. The data set also comes with a complete set of ground truths for each image, so this set is ideal for testing the eye tracker. Many other papers also use this test set, so it is also easy to compare the results with other papers.

Figure 6.6 shows a number of images from the BioID test set where the eye tracker is applied. In these eight images, the difficult situations for the eye tracker are presented here as well. For example when a person closes their eyes, the pupil is not visible anymore. In those cases it is not strange that the eye tracker does a wrong estimate. A possible solution to partially overcome this is to have some sort of blink detection, that simply selects the middle of the shut eye as the pupil location. In de case of a person wearing glasses, two major issues can occur. The first is that the frame of the eyeglasses is very dark. This can trick the eye tracker into thinking that this frame is actually the pupil. Frames can have a lot of different shapes and sizes, so this really is a difficult problem to overcome. An example of this is shown in figure 6.8. The second major issue is glare that shines upon the glasses. This glare can distort the image very drastically, which results in a difficult situation for the tracker. Figure 6.8 also shows an example in which there is glare on the glasses. For most of the situations however, the eye tracker estimates the pupil location very accurately. In the comparison section below, more details are presented about

Figure 6.6: A number of results taken from the BioID dataset

the accuracy of the eye tracker.

6.2.1 Speed

Earlier in this report, the speed of the face tracker discussed. This showed promising result, but the question remained if there was enough computation time left for the eye tracker to stay above the threshold of 30 fps. The same graph from the previous speedtest is displayed in table 6.9, but this time a line for the full algorithm (face and eye tracking) is added.

This graph shows that the algorithm runs an average of more than 30 fps for up to about 500 particles. This number of particles is more than enough to be

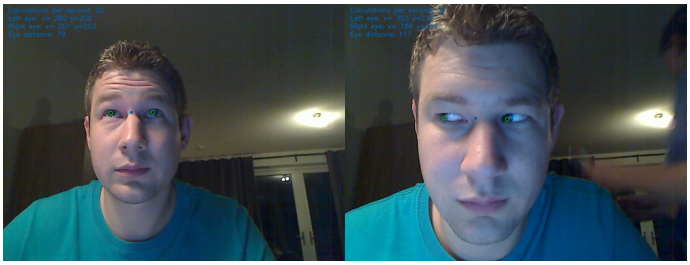


Figure 6.7: Some more results of the eye tracker

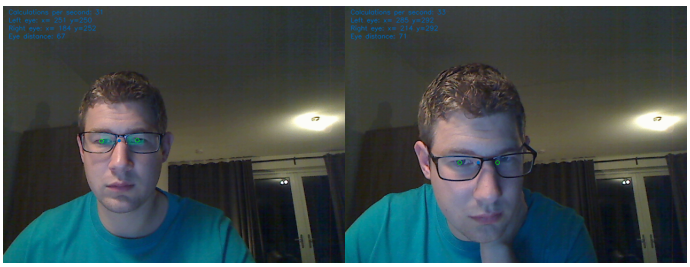


Figure 6.8: Different situations where the eye tracker tends to fail

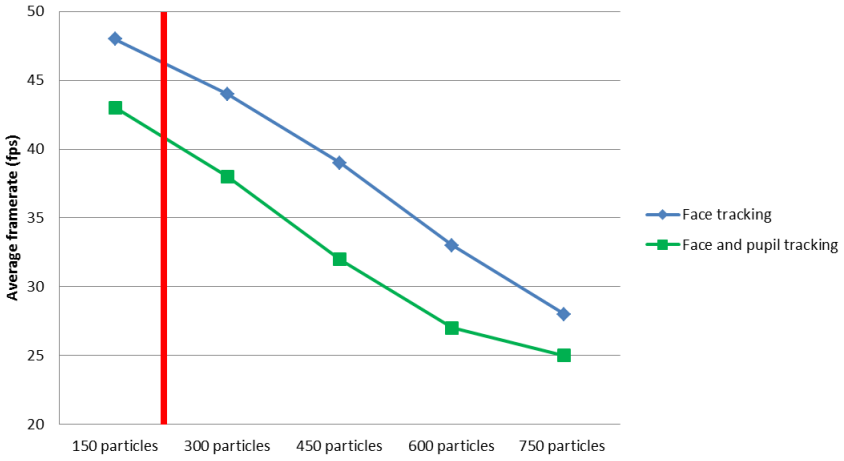


Figure 6.9: Different amounts of face brightness

able to track fast and sudden movements. This speedtest is done on the same system as stated in the previous speedtest. As for the delay, information form the previous frame is used to make the gradient mask. This is however only a tool to roughly help the tracker towards the correct position. The information from the previous frame will not be average with or calculated into the new frame in any way. And because it is a gradient, the mask allows the new position to anywhere in the eye region, but positions closer to the previous frame position will have lower cost.

6.2.2 Comparison

To be sure that the method proposed in this paper has any contribution, it is important to compare the method to others. Therefore, a comparison result is used from the paper of Valenti and Gevers [13]. They also do a comparison at the end between a lot of different methods that used the same data set for testing. They showed a table which shows how good each of these methods is. To measure how good a tracker is, the normalized error is used. It is calculated as follows:

$$e = \frac{max(d_{left}, d_{right})}{w} \tag{6.1}$$

Where d_{left} and d_{right} is the Euclidean distance between the estimated pupil location and the ground truth, and w is the Euclidean distance between the eyes in the ground truth. This error value is often used in these kinds of papers and was introduced by [10]. The table below shows what these error values roughly mean when applied to eyes.

Error value	**Description**
≤ 0.25	Distance between eye center and eye corner
≤ 0.1	Range of the iris
≤ 0.05	Range of the cornea

The method proposed in this paper is also applied to the BioID data set and is compared with the ground truth to come up with results that can easily be compared to the ones in the table from the paper of Valenti and Gevers. These results were added to this table.

Method	**Accuracy** ($e \leq$ **0.05**)	**Accuracy** ($e \leq$ **0.10**)	**Accuracy** ($e \leq$ **0.25**)
MIC	77.15%	82.11%	96.35%
MIC+MS	79.56%	85.27%	97.45%
MICs+SIFT	84.10%	90.85%	98.49%
Asteriadis	74.00%	81.70%	97.40%
Jesorsky	40.00%	79.00%	91.80%
Cristinacce	56.00%	96.00%	98.00%
Turkan	19.00%	73.68%	99.46%
Bai	37.00%	64.00%	96.00%
Campadelli	62.00%	85.20%	96.10%
Hamouz	59.00%	77.00%	93.00%
This paper	**76.74%**	**96.50%**	**99.93%**

These result show that the method proposed in this paper scores extremely high for $e \leq 0.10$. The score for $e \leq 0.05$ is relatively low compared to the result that Valenti and Gevers preseted. These error values mean nothing if it is not clear, how accurate the system has to be in the end. It is known from experience that the eye tracker does not have to be 100% accurate to produce good quality 3D. To determine how inaccurate the tracker is allowed to be, the error margin was calculated by doing an experiment explained below.

6.2.3 Error Margin

The error margin experiment is a test to see how far 'off' the eye tracker is allowed to be without losing any 3D quality. The setup used for this test can be seen in figure 6.10. This image shows an autostereoscopic 3D display with a camera on top of it. Approximately 2 feet from this display is a stand with a ruler attached to it. This stand has another camera on top of it. The meaning of this setup is that the camera on the display does the tracking while the other camera captures the content on the display. For this experiment a special piece of content was created. An image of the content can be seen in figure 6.11. This image is a so-called side-by-side image. This images is split up into two halves by the renderer and these two halves are placed on top of each other. When this image is displayed on a autostereoscopic display and the viewer is in one of the viewing cones, the top and bottom of the image on the screen will both look gray, because black and white are blended together. In theory, when the viewer then closes the right eye, the image appears to be fully black at the top and fully white at the bottom. This happens because the display creates two views to create the 3D effect. These two views are the two parts of the side-by-side image and each of these views is displayed in it's corresponding cone part(left eye and right eye). So trough the left eye, the viewer will see only the left image of the total side by side image. This principle can now be used to determine how inaccurate the eye tracker is allowed to be. In this experiment, the second camera will be a one-eyed 'viewer'. This second camera will capture the content, which is broadcasted on another screen.

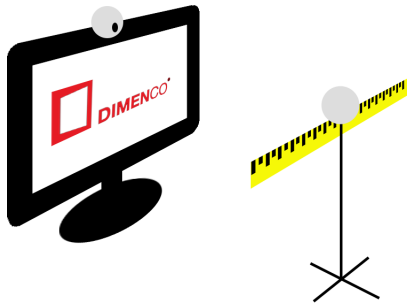


Figure 6.10: The setup for the error margin experiment

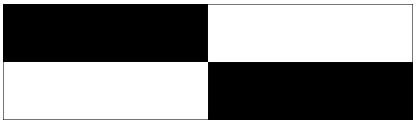


Figure 6.11: A black and white side by side image

The first step of the experiment is to lock the eye tracker. Remember that the implementation allowed the user to lock the tracker by pressing 'y'. After that the user can alter the eye positions and distance with other key commands. Using this the tracker has to be locked, so that the position of the left eye is set to the lens of the observing camera. To obtain the correct eye distance, a marker is put next to the camera at 6.8cm from the lens. This distance is used because that is the average eye distance of an adult human. Figure 6.12 shows how the tracker has to be locked during the experiment.

Once the position and the distance of the tracker are correctly set, the observer looking at the second screen should see content from the 3D display that is clear black at the top and clear white at the bottom. The next step of the process is to first move the camera slowly towards one side until the content on the display changes (the camera moves out of the cone), mark that point and slowly move towards the other side until the content changes again. This second changing point is also marked. The final step in this experiment is now to count how many pixels are in this no-changing range. This number of pixels is the range in which the viewer will experience good quality 3D. This range can be used to calculate what the maximum error margin on either side is by using the two outer points of this range as the tracked position and the starting point of the camera as the ground truth in the normalized error equation (equation 6.2.2).

This experiment is not yet done so the exact error margin value cannot

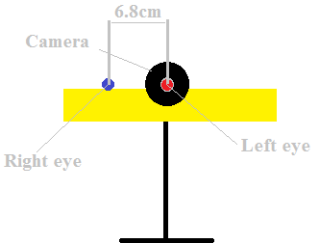


Figure 6.12: How to lock the tracker for the error margin experiment

be stated here yet. However, the final version of this paper will contain this the result of this experiment. With this result, the comparison table from earlier can be used to see how good the proposed eye tracker is for the use of autostereoscopic displays. The error margin is expected to be somewhere between 0.1 and 0.25.

Chapter 7

Conclusion

This thesis presents an eye tracker method specific for autostereoscopic displays. Through experiments and comparison, this paper has shown that the eye tracker proposed the qualities to meet the requirements. The main contributions of this thesis are the combination of the the robust Viola Jones algorithm with a particle filter to achieve robust and most of all very fast face tracking. The second contribution is the combination of methods to determine the pupil location from within the eye region. One existing and proven method is used for this as a basis (isocenters) and together with two other methods, the accuracy that is needed for the autostereoscopic displays has improved. There are some situations where the tracker fails, such as eyeglasses or very extreme lighting conditions as can be seen in the result section. The eyeglasses problem is due to the fact that the frame is often seen as a pupil or that there is a highlight on the glass which obstructs the eye behind it. In these cases this method can fail to guess the correct pupil location because there is either too much (glasses frame) or too little (highlight) information. And for the extreme lighting conditions, the face is just too different on different parts. The face detector is trained with certain features that a face has. These features disappear when the face is lit too extremely. These situations are still open problems in this field of research and will be very difficult to solve.

7.1 Future Work

The biggest issue to overcome is to correctly track the eyes when the viewer is wearing glasses. Especially when there is a lot of glare on the glasses, the tracker often fails. There are methods that are able to remove glare [12], but these are not applicable yet on real time systems such as this one. However, there are easier methods to detect highlight (the intensity goes up), so this information can be used to switch to a different kind of pupil detection that is more robust to these highlights when a glare is detected. Also, the cost function weights were experimentally determined, but only for one situation. It would

be a nice addition if these weights were dynamic and could be adjusted in real time according to the current conditions such as the brightness or whether the viewer wears glasses. To do this, it is necessary to understand which situations call for which method to weigh the heaviest.

Bibliography

[1] C. Akinlar and C. Topal. Edcircles: A real-time circle detector with a false detection control. *PATTERN RECOGNITION*, 2013.

[2] AA. Argyros and MIA Lourakis. Real-time tracking of multiple skin-colored objects with a possibly moving camera. *LECTURE NOTES IN COMPUTER SCIENCE*, 2004.

[3] H. Baltzakis, M. Pateraki, and P. Trahanias. Visual tracking of hands, faces and facial features of multiple persons. *MACHINE VISION AND APPLICATIONS*, 2012.

[4] R. Belaroussi and M. Milgram. A comparative study on face detection and tracking algorithms. *EXPERT SYSTEMS WITH APPLICATIONS*, 2012.

[5] GR. Bradski. Real time face and object tracking as a component of a perceptual user interface. In *FOURTH IEEE WORKSHOP ON APPLICATIONS OF COMPUTER VISION - WACV'98, PROCEEDINGS*, pages 214–219, 1998.

[6] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 2002.

[7] Y. Freund and RE. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *JOURNAL OF COMPUTER AND SYSTEM SCIENCES*, 1997.

[8] DC. Guix. Real-time face tracking methods. -, June 2009.

[9] R. Hess and A. Fern. Discriminatively trained particle filters for complex multi-object tracking. In *CVPR: 2009 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, VOLS 1-4*, pages 240–247, 2009.

[10] O. Jesorsky, KJ. Kirchberg, and RW Frischholz. Robust face detection using the hausdorff distance. In *AUDIO- AND VIDEO-BASED BIOMETRIC PERSON AUTHENTICATION, PROCEEDINGS*.

[11] Papageorgiou. A general framework for object detection. In *Computer Vision, 1998. Sixth International Conference on*.

[12] RT. Tan, K. Nishino, and K. Ikeuchi. Separating reflection components based on chromaticity and noise analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2004.

[13] R. Valenti and T. Gevers. Accurate eye center location and tracking using isophote curvature. In *2008 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, VOLS 1-12*, pages 1452–1459, 2008.

[14] P. Viola and MJ. Jones. Robust real-time face detection. *International Journal Of Computer Vision*, 2004.

[15] J. Zheng, GA. Ramirez, and O. Fuentes. Face detection in low-resolution color images. In *IMAGE ANALYSIS AND RECOGNITION, PT I, PROCEEDINGS*, pages 454–463, 2010.