Head Position Invariant Gaze Tracking

Diplomarbeit

vorgelegt von Andreas Tarnowsky

angefertigt am
Lehrstuhl Graphische Datenverarbeitung
Institut für Mensch-Maschine-Kommunikation
Gottfried Wilhelm Leibniz Universität Hannover

Erstprüfender:	Prof. Dr. Franz-Erich Wolter
Zweitprüfender:	Prof. Dr.-Ing. Holger Blume
Betreuer:	Dipl.-Math. Rasmus Buchmann

30. Dezember 2011

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ausschließlich mit Hilfe der angegebenen Quellen und Hilfsmittel angefertigt habe.

Hannover, den 30. Dezember 2011 Andreas Tarnowsky

Abstract

This thesis presents the conception and implementation of a low-cost gaze tracker, i. e. a device that allows to track one users eye movement and map the actual point of gaze to screen coordinates. Within this work, the construction of a head-mounted hardware device as well as various algorithms for the tracking- and mapping procedure are discussed and several improvements to existing approaches are presented. The mapping procedure uses a novel approach utilizing so called radial-basis functions for an interpolation task. In order to allow natural head movement, this approach is extended to generic function approximation in higher dimensional spaces. By adding external markers that cause reflections on the users cornea, the system is able to compensate for head movement while reaching interactive frame rates at any time.

Danksagungen

Ich möchte mich an dieser Stelle – ausnahmsweise nicht in englischer Sprache – bei allen Personen bedanken, die mich während der Anfertigung dieser Arbeit unterstützt haben.

Bei Herrn Professor Wolter möchte ich mich für die Möglichkeit bedanken, diese Arbeit an seinem Institut zu schreiben. Ebenso möchte ich meinen Dank gegenüber Herrn Professor Blume zum Ausdruck bringen, an dessen Institut ich mehrere Jahre arbeiten durfte und der sich als Zweitprüfer für diese Arbeit zur Verfügung stellte.

Des weiteren möchte ich Rasmus Buchmann danken, der sich bereit erklärte, diese Arbeit zu betreuen und viele Ideen beisteuerte. Auch seine Korrekturen und Anmerkungen zu den vielen sprachlichen Feinheiten haben mir sehr geholfen. Auch Holger Flatt möchte ich dafür danken, mir im Laufe meines Studiums einen Einblick in die Hardwareentwicklung ermöglicht zu haben; einem Themengebiet, dem ich sonst – laut dem regulären Studienplan – vermutlich nicht begegnet wäre. Leider hätte eine FPGA-Implementierung der vorliegenden Arbeit den Rahmen einer Diplomarbeit gesprengt.

Großen Dank gebührt meinen Eltern für ihre ideelle und finanzielle Unterstützung während meines gesamten Studiums. Ebenso danke ich meiner Freundin Katrin für die moralische Unterstützung während meiner Arbeit.

Schließlich möchte ich mich auch bei Alexander, Dennis, Felix, Manuel, Maximilian, Philipp und Rasmus bedanken, die sich den – doch teils anstrengenden – Probandentests unterzogen haben.

Hannover, im Dezember 2011

Andreas Tarnowsky

Contents

Chapter 1

Introduction

Right from the beginning, when the first personal computers were available, users were confronted with various input methods. Common devices, like keyboard and computer mouse, are only a small subset of the progresses made in Human-Computer Interaction (HCI) research. Nowadays, these very basic input devices are used on a regular basis by almost every computer user.

◄ HCI

Within the last years, many additional interaction-methods have been developed. Aside from almost trivial computer gear like joysticks and other gaming hardware, great effort has been spent on making computer interaction easier and more intuitive. For example, IBM introduced in 1996 the software *VoiceType* that allowed to omit the keyboard to a certain degree by interpreting the user's spoken words as text input[1]. Another invention that turned out to be very successful was made in 2006 by Nintendo. The gaming console *Wii* uses a wireless controlling device utilizing various sensors to keep track of changes in position and acceleration of the user's hand. These are only two examples for innovative and commercially successful input methods.

Commercial inventions

The science community around HCI is still searching for new innovations in the field of input hardware design, allowing to interact with e. g. virtual reality (VR) environments more naturally. However, another very large research topic has not been mentioned yet: Using the user's line of sight - also known as gaze direction - one can adjust a VR-scene to aid concentration on specific details or to trigger interactive options. Up to now, several commercial tracking devices do exist that allow high precision gaze tracking. Unfortunately, these devices are very costly and therefore not affordable for the general public. During the last decades, many attempts have been made by the science community to create low cost gaze tracking

◄ VR

Gaze tracking

1 For a more complete history of speech recognition research by IBM see `http://www.research.ibm.com/hlt/html/body_history.html`

systems and increasing their accuracy. Up to now – to my knowledge –, none of them have been introduced to a broader audience by making it commercially available.

Goal of this thesis

This thesis describes the conception of a low cost gaze tracking device that allows for natural head movement within a certain range. It is made from freely available components with total material cost lower than 100 €. Additionally, several algorithms have been tested and extended to provide a software framework for tracking the user's gaze direction and mapping it to a computer-screen. In order to evaluate the system and validate its robustness, several proband-tests were made.

Document structure

The document is structured as follows: *Chapter one* describes related work regarding this thesis, including a brief overview of the history of gaze tracking. In the *second chapter* the fundamentals of gaze tracking are described and a simplified algorithm is discussed that requires the user's head position to be fixed while tracking. This restriction is then omitted in *chapter three* allowing more natural head movement. Several techniques are described in this part of the thesis. Thereafter, the actual implementation of the algorithms is presented in *chapter four*. The chapter also includes the technical specifications for the tracking hardware as well as some short assembly instructions. In order to be able to compare the results of this thesis with related projects, the technical setup for the evaluation procedure is described in *chapter five*. Finally, *chapter six* concludes this work.

1.1 History of gaze tracking

The need for tracking a persons eye movement goes back to the late 19th century. At this time, psychologists tried for the first time to categorize the various types of eye movement that may happen e. g. when reading a text like this. This early research formed a foundation for many applications in neuroscience and modern psychology, trying to understand human vision more completely. Over time the need for more accurate eye-/gaze tracking[1] methods emerged.

Three eras of gaze tracking

According to *Rayner*, the history of eye movement research can be splitted up into three eras [Ray98]:

1 The terms *eye tracking* and *gaze tracking* are often taken synonymously. In this thesis, *gaze tracking* will mean getting the actual line of sight whereas *eye tracking* denotes the localization of the eye within a facial image

1879 - 1920 First observations During the *first era*, many fundamental observations were made. In this period, keywords like *saccadic supression*, *saccade latency* and *perceptual span* appeared - just to name a few. The American experimental psychologist *Raymond Dodge* was a well known person in the area of eye movement research at this time. His work on classifying eye movements into five distinctive types [Dod03] has been cited extensively. During his research he also demonstrated some remarkable engineering qualities by creating a device to track human eye motion (see Figure **??**). This construct uses an evenly moving photographic plate (**D**) mounted to a camera (**C**) to take a continouus photograph of the eye. The user's chin is fixed on a head-rest (**A**) during this procedure. One can consider this apparatus as a precursor for modern gaze tracking devices.

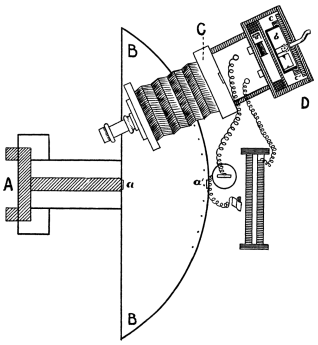


Figure 1.1: The original plan of the apparatus described by *Dodge* as viewn from above [Dod03].

1935 - 1958 Applied phase In the *second era*, only very little research has been done. The reason for this may be that "almost everything that could be learned about reading from eye movements (given the technology at the time) had been discovered" [Ray98]. Within these years, a behaviorist movement in experimental psychology was in progress, resulting in a more applied research rather than progress in eye movement analysis.

1970 - today Technological breakthroughs When the first programmable computer systems appeared, eye movement research experienced a renaissance. The improvements in recording systems allowed easier obtainment of proband data and much higher precision, thus leading to the *third era* of eye movement research. Upon today, eye movement analysis – now widely known under the term of *gaze tracking* – spreaded from its psychological origins quickly across many different fields of research.

Neuroscience and psychology

For example, gaze tracking found many applications in neuroscientific research as well as modern psychology. Interdisciplinary research achieved interesting results by combining gaze tracking with brain imaging equipment [Duc02]. In the context of psychology, *Rayner*also mentions many different domains where gaze tracking is used [Ray98]:

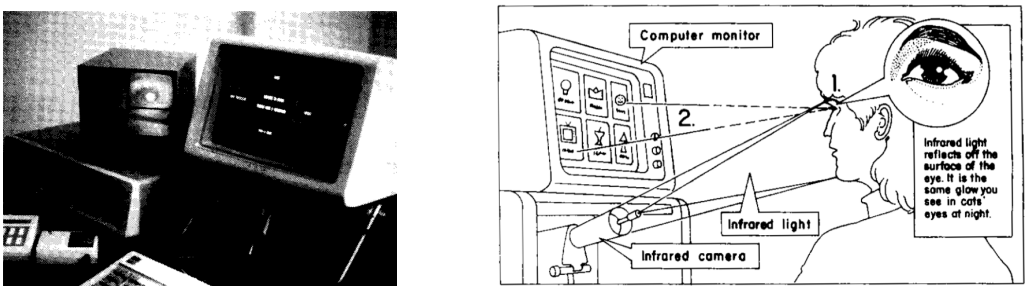
- mathematics, numerical reading and problem solving.
- dual task situations
- face perception
- dynamic situations
  (e. g. driving, walking in uneven terrain, human-computer interaction)

Computer sciences

Another very large interest in gaze tracking originates from computer sciences, which was involved in the beginning of the third era. One of the first gaze trackers that has been made commercially available was invented in 1989 by a research group around *Thomas E. Hutchinson*. The development of this machine – called *"Erica"* – took many years and over one-hundred undergraduate and graduate students were engaged in the project [HWJM+89]. The system (see *figure 1.2*) was able to distinguish among nine reference areas arranged in a 3×3 grid on a computer screen. Despite its low accuracy compared to more recent gaze trackers, many ideas and techniques used in the underlying tracking algorithms are still essential parts of modern approaches.

Over time and with increasing accuracy of tracking systems, more and more interactive tracking applications appeared. One idea that comes to mind naturally is to replace the computer mouse of a personal computer with a gaze based device. However, as *Jacob* pointed out [Jac90], even simple actions like a mouse-click are hard to model when using a gaze tracker. Additionally, eye movement is very spontaneous and not every movement is intended to trigger a computer command[1]. Therefore, many established conventions in user interface design have to be reconsidered when dealing with gaze based interaction.

POG ► In contrast to the active usage of the point of gaze (POG) in "mouselike" interaction



(a) Hardware system **(b)** Schematic illustration

Figure 1.2: The *Erica* gaze tracking system [HWJM+89].

1 *Jacob* describes this behaviour as the *"Midas Touch" Problem*

applications, the POG also can be used as a passive indicator for which region of a graphical display the user is looking at. So called *gaze-contingent displays* (GCDs) allow to change the information density adaptively, depending on the user's line of sight. For example, such displays can be used to minimize bandwidth usage in video telephony applications [Duc02]. Only the parts the user is actually looking at are encoded with high quality. Regions lying in the peripheral field of view are encoded using a reduced quality to save overall bandwidth.

◄ GCD

Last but not least, gaze tracking and eye movement analysis are used for optimizing various processes in marketing and advertising [Duc02]. gaze tracking allows an insight into the consumers visual attention and behaviour when searching for specific information. This allows to e. g. arrange advertisement in a way the consumer will not overlook.

Marketing and advertising

1.2 Related work

The foregoing section indicates that there is a great need for gaze tracking devices in many different fields of application. This may be the reason for the huge amount of techniques that appeared during the last decades. This section only contains a small outline of the approaches that have been developed over time and tries to categorize them in order not to get lost in the variety of tracking methods.

As *Duchowsky* [Duc02] pointed out, gaze trackers can be coarsely divided into two areas of application. The first area contains *diagnostic* tracking applications that provide a quantitative measure for the user's visual and attentional processes. Diagnostic applications often can be processed offline after a video of the user's eye-movement has been recorded. The second area mentioned by *Duchowsky* covers the field of *interactive* gaze tracking applications. Here special attention is devoted to low latencies of the tracking system, since the gaze-data is used for (near) realtime applications that e. g. give a direct response as the user's gaze position changes. Gaze contingent displays – that already have been mentioned before – are an example for an interactive tracking application.

Classification of use cases

The different types of hardware also can be divided into two groups. *Morimoto et al.* classify them as either *intrusive* or *non-intrusive* gaze trackers [MM05]. A characteristic property of intrusive tracking systems is their need for physical contact with the user. For example, *Robinson* uses a so called *search coil* integrated into a contact lens that allows for a very high precision [Rob63]. But also less intrusive techniques like head mounted tracking devices fall into this category. Recent research concentrates mainly on non-intrusive tracking methods, that use a camera

Classification of hardware devices

system to monitor eye movements. These camera systems usually consist of one or more digital video cameras that are set up remotely e. g. besides the computer monitor.

Semi-intrusive tracking systems

Of course, non-intrusive trackers have better potential for everyday use than systems that require physical contact. However, their accuracy is in most cases somewhat limited compared to intrusive devices. Since head mounted tracking systems still have potential for everyday use – as long as they are lightweight enough –, I would like to denominate such systems as *semi-intrusive* within the context of this thesis. Most gaze tracking devices of that kind are built in form of eyeglasses like the ones by *Li, Winfield and Parkhurst* or *Babcock and Pelz* [LWP05, BP04].

Methods of illumination

Concentrating on non- and semi-intrusive tracking systems, the class of camera based techniques can be further divided depending on the way the video stream is analyzed. Most tracking systems designated for indoor use utilize infrared light for illuminating the eye region [HJ10]. Additionally, such *active illumination* schemes allow for an easier detection of the user's pupil since the infrared light enhances the contrast of the pupil obverse to the iris region. Active illumination schemes can be classified either as dark- or bright-pupil methods. If the light source is close to the optical axis of the camera, the pupil appears to be bright since the light is reflected back to the camera [HJ10]. As the distance of the light source to the camera increases, the brightness of the pupil quickly decreases. In many different tracking systems, a combination of both methods is used since the difference of the bright- and dark-pupil image makes it relatively easy to find the user's pupil (cf. e. g. [ZJ04]). However, the intensity of the bright-eye effect varies across different persons and ethnic groups [NWKF02] and therefore might reduce the reliability of such systems.

Regardless of the illumination scheme, there are even more ways for classifying the actual gaze tracking method [HJ10]:

Appearance-based methods use a high dimensional input vector for acquiring the gaze position. These input vectors often consist of whole images containing the eye region of the user. A typical method for processing such large vectors is proposed by *Xu, Machin and Sheppard* [XMS98]. They use images of the user's eye which form a 600-dimensional input vector for a neural network. *Baluja and Pomerleau* present a similar approach, using up to 20 hidden layers within a feed-forward network [BP94]. As these numbers may indicate, the performance of such systems is often not sufficient for interactive usage. To my knowledge, no system of this kind has been reported to allow for free head motion during the tracking process [HJ10].

Feature based methods only take specific features within the eye region into account for further analysis. These features typically contain the position of the user's pupil and the so called *"glint"*. The glint is the bright reflection on the user's eyeball or its cornea caused by the infrared light source used for illuminating the scene.

◄ Glint

Interpolation-based mapping

◄ PGV

Both features then can be used to create a mapping function that uses the difference vector between the user's pupil-center and the glint – the so called *pupil-glint-vector* (PGV). Since the PGV does not move linearly with the eye-movement due to the complex shape of the human eye, several techniques do exist to compensate for this nonlinearity. Many of these methods use linear or higher order polynomial interpolation methods [RDD08]. For that reason, this approach is known as *interpolation-based* mapping. The major downsides of interpolation-based techniques are their sensitivity to head movement and the need for an initial calibration that usually forces the user to fixate on a series of test-points [MM05].

Model-based mapping

The so called *3D model-based* approaches try to circumvent these limitations by trying to create a 3D model of the user's eye using the given feature vectors. They typically involve more complex camera and lighting setups. According to *Shih, Wu and Liu*, the user-dependent parameters that have to be modelled include – among others – the radius of the cornea, the position of the pupil and the position of the user's head [SWL00]. They also describe a system that allows to estimate the user's line of sight without an explicit calibration procedure [SL04]. However, this method needs multiple cameras and light sources and the accuracy of the system is rather limited. Using information taken from both eyes, *Hennessey and Lawrence* present a tracking system that is able to estimate the user's fixation point within 3D space and also allows for free head movement within a working space of $30 \times 23 \times 25\,cm$ [HL09]. Unfortunately, the maximum latency of the system ($1.5s$) disqualifies the device for interactive usage.

Techniques applicable to this thesis

A very elegant technique has been proposed by *Yoo and Chung* [YC05]. They describe a model-based approach that allows free head movement while needing only four infrared light sources placed at the edges of the computer monitor and a relatively simple camera system composed of two cameras and an additional infrared light source. This method only needs very little calibration and has been further analyzed by *Coutinho and Morimoto* who improved the accuracy of the system significantly [CM06]. Since the ideas of this publication also apply to certain parts of this work, a more detailed discussion of this technique is given in *chapter 3*.

1.3 Notation and general conventions

Throughout this document, the notation specified in *table 1.1* will be used. Additionally, some general conventions apply to this work:

- All vectors and matrices used in this document are real valued if not stated otherwise.
- If not explicitly indicated, $\|\cdot\|$ denotes the $L2$-norm (euclidean norm).
- Distances between vectors are always measured using the metric induced by the $L2$-norm, so $d(\boldsymbol{x},\boldsymbol{y}) := \|\boldsymbol{x}-\boldsymbol{y}\|_2$ holds.
- When searching for a set of points that maximizes a given function $f(\boldsymbol{x})$ for an argument $\boldsymbol{x}$, the arg max notation is used:

  $$\arg\max_{\boldsymbol{x}} f(\boldsymbol{x}) := \{x \mid \forall y : f(y) \leq f(x)\}$$

  For the set of minimizing values, arg min is defined analogously.

Description	Sample
Scalar values are denoted using lowercase slanted Roman or Greek letters. If not noted elsewhere, scalar values are always real valued.	a, b, c α, β, γ
Vectorial valued variables are always typesetted in lowercase boldface letters.	$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$
Matrices are written in uppercase boldface letters.	$\boldsymbol{M}$
Images are treated as a special form of a matrix. The element-wise access stays always within the bounds of the image by implicitly applying a modulo operation to the positions x and y.	$\boldsymbol{I}[x,y]$

Table 1.1: Notation used in this work

Chapter 2

Fixed head gaze tracking

This chapter deals with a simplified approach to gaze tracking. Instead of allowing natural head movement, a fixed head position is assumed. Most of the algorithms presented in this chapter are also used for the more complex case of head position invariance and allow to evaluate the tracking hardware in a more reproducible way (see *chapter 5*).

The basic algorithm flow presented here has some overlappings with already existing approaches [YC05, PCG+03]. However, various improvements have been made regarding the overall robustness to unintentional specular reflections. This is done by introducing two original methods for determining the pupil-center. In addition, I present a novel way for mapping the pupil-glint vector to screen coordinates.

Chapter outline

This chapter is subdivided into four sections. At first, some fundamental techniques on digital imaging are explained in short. Readers who are familiar with topics like *connected-component labeling*, *edge detection* and *ellipse fitting* can safely skip this section. The second one gives a short outline about the algorithm in general. The individual parts of this outline are then examined more closely in the last two sections. These parts describe how to analyse the input images to obtain the *pupil-glint-vector* and how to map this vector to the coordinate system of the computer monitor.

2.1 Preliminaries

Before I start to describe the tracking algorithm, some common digital imaging methods and algorithms have to be discussed. At first some basic image processing algorithms are shown in section 2.1.1. These include *template matching*, *connected-component labeling* and *discrete convolution*. Afterwards I will depict two popular techniques for finding edges within an image, namely the *Marr-Hildreth-Operator* and the *Canny-algorithm*. Lastly, I give a short introduction to *ellipse fitting*.

2.1.1 Common techniques

For the sake of completeness and to prevent misunderstandings in nomenclature, some fundamental concepts in digital imaging are recapitulated.

Template matching is a well known technique for locating a template image within a larger one. The naïve approach is to use a *brute force* strategy to minimize the *sum of square differences*. Let w and h denote the width and height of the input image $\boldsymbol{I}$, w' and h' the width and height of the template $\boldsymbol{T}$. The offset q of the template relative to the image can be determined as follows:

$$q = \arg\min_{k} \sum_{x',y'} (\boldsymbol{I}[x'+k_1, y'+k_2] - \boldsymbol{T}[x', y'])^2$$
$$\text{with} \quad 0 \leq k_1 < w, 0 \leq k_2 < h \quad \text{and} \quad 0 \leq x' < w', 0 \leq y' < h'$$

The position q then denotes the position of the template image that matches the original image best. It is quite obvious that this procedure is very computationally intensive. Therefore it is advisable to downsample both images if the loss of accuracy is tolerable. Additionally, multiresolution approaches do exist that operate at different scales of both images in order to find the exact template position. Regardless which implementation is chosen, this technique is very sensitive to rotations, scaling and changes in illumination of the image $\boldsymbol{I}$.

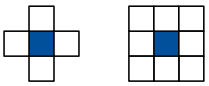


Figure 2.1: Two possible pixel neighborhood patterns: *4-connected* (left) and *8-connected* (right).

Connected-component labelling allows to detect groups of pixels within an image whose elements are *connected* with each other (see *definition 1*). There are basically two possibilities to define the *neighborhood* of

a pixel. Depending whether diagonal connections are allowed or not, two pixels are called *8-connected* or *4-connected* when they are lying within the same region as depicted in *figure 2.1*.

Connected pixels

Definition 1. *Let $\boldsymbol{I}$ be a binary image whose size is given by $w,h \in \mathbb{N}$. Two pixels p_0 at (x_0,y_0) and p_n at (x_n,y_n) are called* connected *if and only if a sequence of pixels $(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$, $0 \leq x_k < w$, $0 \leq y_k < h$ $\forall k \in \{0\ldots n\}$ does exist, such that the pixels at (x_k,y_k) and (x_{k+1},y_{k+1}) are neighbored and $\boldsymbol{I}[x_k,y_k] = \boldsymbol{I}[x_{k+1},y_{k+1}]$ holds for all $k \in \{0\ldots n-1\}$.*

Typical implementations

In order to find such regions, several techniques do exist. These techniques can be coarsely subdivided into two classes. The first one needs two passes for identifying all labels within an image, the second class allows to do the same within a single pass [CCL04]. In contrast to two step approaches like [FBH+08], single pass algorithms often require random access patterns for establishing relations between the pixels, thus making them less favourable e. g. for embedded hardware designs.

Discrete convolution can be applied to an image $\boldsymbol{I}$ in conjunction with a $n \times n$ filter kernel $\boldsymbol{D}$ in order to perform various discrete filter operations. These include *lowpass-* and *highpass-filters* as well as approximations of partial derivatives of the input image. The filtered image $\boldsymbol{I}'$ can be computed as follows:

$$\boldsymbol{I}'[x,y] = (\boldsymbol{I} * \boldsymbol{D})[x,y] := \sum_{k,l=0}^{n-1} \boldsymbol{I}\left[x+k-\frac{n-1}{2}, y+l-\frac{n-1}{2}\right] \boldsymbol{D}[k,l]$$

For example, to approximate the horizontal and vertical derivatives, the so called *Sobel operator* can be used:

◄ Sobel-operator

$$\boldsymbol{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, \qquad \boldsymbol{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Separable kernels

In order to save computation time, *separable kernels* allow to reduce the number of memory accesses. Instead of using a $n \times n$ filter kernel, a $n \times 1$ kernel is used. Afterwards the intermediate result is convoluted with a $1 \times n$ kernel matrix.

For example, $\boldsymbol{G}_x$ is a separable filter kernel, since it can be separated into two vectors $\boldsymbol{g}_x^1 = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^\top$ and $\boldsymbol{g}_x^2 = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$, so that $\boldsymbol{g}_x^1 * \boldsymbol{g}_x^2 = \boldsymbol{G}_x$ holds. Now

the image $\boldsymbol{I}$ can be filtered using the separated kernels, since the convolution operation is associative:

$$\boldsymbol{I}'[x,y] = (\boldsymbol{I} * \boldsymbol{G}_x)[x,y] = (\boldsymbol{I} * (\boldsymbol{g}_x^1 * \boldsymbol{g}_x^2))[x,y] = ((\boldsymbol{I} * \boldsymbol{g}_x^1) * \boldsymbol{g}_x^2)[x,y]$$

2.1.2 Edge detection

A very common task in digital image processing is to find prominent features within an image. *Edges* inside an image are natural candidates for local features and can be described as discontinuities in the brightness of the image.

Many algorithms do exist to detect edges within images. Each of them has different characteristics regarding sensitivity to noise, isotropy and ease of computation. In this work, two well known edge-detection techniques are used in order to obtain the contour of the user's pupil.

Laplacian based edge detection

Edge detection using the *Sobel operator* is very common and trivial to implement. A certain disadvantage of using the Sobel operator is its anisotropy, i. e. the edge detection process is not rotation invariant [AR05]. To overcome this limitation, second order differential operators like the *Laplacian operator* can be used.

Laplace operator

Definition 2. *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a function of class* $\mathscr{C}^k$, $k \geq 2$. *The* Laplace operator $\Delta : \mathscr{C}^k(\mathbb{R}^n) \to \mathscr{C}^{k-2}(\mathbb{R}^n)$, $k \geq 2$ *is then defined as the divergence of the gradient of* f. *In cartesian coordinates, the* laplacian *of* f *can be expressed as follows:*

$$\Delta f := \nabla^2 f = \sum_{k=1}^{n} \frac{\partial^2 f}{\partial x_k^2}$$

A discretization of the Laplace operator can be used to gain direction invariant edges from an image. To acquire such a discretization, finite differences can be

applied to the two dimensional form of the Laplace operator:

$$\begin{aligned}\Delta f(x,y) &= \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \\ &\approx \frac{\partial(f(x+1,y) - f(x-1,y))}{2\partial x} + \frac{\partial(f(x,y+1) - f(x,y-1))}{2\partial y} \\ &\approx f(x+1,y) - 2f(x,y) + f(x-1,y) \\ &\quad + f(x,y+1) - 2f(x,y) + f(x,y-1)\end{aligned}$$

Laplace filter kernel

Conveniently, this expression can be transferred to a 3×3 matrix that can be used for discrete convolution with the image to be analyzed:

$$\boldsymbol{D}_{xy} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

After the discrete Laplace operator has been applied to an image, edges can be found by searching for zero crossings within the resulting image [AR05].

As already stated, the main advantage of using the discrete Laplace operator for edge detection is its isotropy. Since $\boldsymbol{D}_{xy}$ contains only five coefficients, the convolution can be computed very efficiently.

The *Marr-Hildreth*-operator

When working on grayscale images, the discrete Laplace operator suffers from high sensitivity to noise, leading to a high rate of false edge detections. A slightly more advanced edge detection method that is less sensitive to noise can be obtained by convolving the image with a Gaussian filter before applying the Laplace operator. This technique is widely known as the *Laplacian of Gaussian* (LoG) or the *Marr-Hildreth*-operator. Since convolution is an associative operation, a discrete filter kernel of the Marr-Hildreth-operator can be obtained directly through discretization of the following term [AR05]:

$$m(x,y) = \Delta \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

In order to reduce the sensitiveness to image noise, a relatively high value for σ has to be chosen. This leads to a very large discretization matrix thus making this edge detection method rather inefficient. *Figure 2.2* shows a comparison of the

Marr-Hildreth-operator and the Canny-algorithm described in the next section. It is easy to see that the results of the Canny-algorithm are far more accurate for gaining the pupil contour.

The *Canny*-algorithm

A very popular edge detection algorithm that offers a high noise immunity has been proposed by *Canny* several years ago. *Canny* stated three criteria that an optimal edge detector has to fulfil [Can86]:

Criteria for optimal edge detectors

1. The error rate (including false positives and false negatives) during edge detection must be low. This criterion corresponds to maximizing the signal-to-noise ratio of the image gradient.
2. The detected edge should be localized as accurately as possible.
3. For each edge in the input image *only one* response should be given.

In his original work, *Canny* also gives a mathematical foundation for determining such optimal edge detectors. Additionally, more recent studies show relationships between the Canny-algorithm and Laplacian based methods by using a variational-geometric formulation of the problem [KB03]. However, explaining the whole mathematical background would exceed the scope of this work. Therefore, only a short summary of the practical algorithm is given.

Practical implementation

The Canny-algorithm is typically implemented by performing several steps on the input image $\boldsymbol{I}$:

Figure 2.2: The detected edges of the Marr-Hildreth-Operator ($\sigma = 1.4$, left image) and the Canny algorithm (right image). The latter method is less sensitive to noise and the pupil contour is far more accurate.

Gradient calculation In order to reduce image noise, $\boldsymbol{I}$ is convolved with a Gaussian filter first. Afterwards, the partial derivatives in horizontal and vertical direction are obtained by convolving it with its respective Sobel filter kernels (see 2.1.1). Let $\boldsymbol{G}_h$ and $\boldsymbol{G}_v$ be the resulting gradient images. Then the gradient direction (ϕ) and the gradient magnitude (τ) can be calculated for each pixel at position (x,y) as follows:

$$\phi_{xy} = \arctan\left(\frac{\boldsymbol{G}_h[x,y]}{\boldsymbol{G}_v[x,y]}\right) \qquad \tau_{xy} = \sqrt{\boldsymbol{G}_h[x,y]^2 + \boldsymbol{G}_v[x,y]^2}$$

Non-maxima suppression The lines produced by the gradient magnitude computation are typically wider than one pixel. In order to fulfil the third criterion, these potential edge candidates have to be thinned. An often used technique is to compare the gradient directions of two adjacent edge pixels. If both directions are similar to each other, the pixel with lower gradient magnitude is discarded. Otherwise, both pixels are kept, since they belong to different edges.

Hysteresis thresholding The gradient magnitude information itself still contains many incorrectly detected edge points. In order to select only dominant edges and to recover whole edge segments, a special thresholding procedure is used. Given two threshold values t_{high} and t_{low}, only edge points are selected whose gradient magnitude exceed t_{high}. For all of these edge points, the contour of the actual edge is traced in both directions until the gradient magnitude falls below t_{low}. The sum of all these edge segments forms the result of the Canny-algorithm.

2.1.3 Least squares ellipse fitting

Fitting an ellipse to a given set of points is a highly complex task when it comes to robustness of the used method. Partial occlusions of the ellipse to be fitted or noisy input data may degrade the quality of the result. Therefore, many different approaches to handle such difficulties do exist [Ros99, GGS94, FF96]. In order to give a short introduction to this topic, this section describes an exemplary technique as shown in [FF96]. The problem of fitting an ellipse to a given set of data points may be stated as follows:

Formulation of the problem

Let $P = \{\boldsymbol{x}_i\}_{i=0}^{n}$, $\boldsymbol{x}_i \in \mathbb{R}^2$ be the set of data points the ellipse should be fitted to. Further, let $C(\boldsymbol{a}) = \{\boldsymbol{x} \in \mathbb{R}^2 \,|\, x_1^2 a_1 + x_1 x_2 a_2 + x_2^2 a_3 + x_1 a_4 + x_2 a_5 + a_6 = 0\}$ be a general conic section in implicit form with a given metric $d_C(\boldsymbol{a}, \boldsymbol{x})$ that measures

the distance between the point $\boldsymbol{x}$ and $C(\boldsymbol{a})$. The parameter $\boldsymbol{a}_{min}$ that minimizes the error function

$$\varepsilon^2(\boldsymbol{a}) = \sum_i d_C(\boldsymbol{a}, \boldsymbol{x}_i), \quad \boldsymbol{x}_i \in P$$

then describes the conic that fits P in respect to the chosen metric d_C best. As *Fitzgibbon and Fisher* state, different constraints can be applied to the vector $\boldsymbol{a}$ that influence the robustness to outliers of the fitted ellipse [FF96]. They show that using a constraint such that $\boldsymbol{a}_2^2 - 4\boldsymbol{a}_1\boldsymbol{a}_3 = 1$ holds leads to good results while the algorithm complexity remains relatively low.

2.2 Algorithm outline

The algorithm used to do the actual gaze tracking consists of several smaller steps of varying complexity. They can be coarsely grouped into two distinct parts. The first one analyzes the image stream taken from the camera of the tracking device. Within these images, the position of the pupil and the glint is acquired. The
PGV ► difference vector of both points – the so called *pupil-glint vector* (PGV) – is then used to find the position on the screen the user is looking at. Assuming the gaze tracker has been calibrated properly, the PGV can be mapped to the coordinate system of the user's monitor. This is done in the second part of the algorithm, by using the calibration data for an interpolation procedure in order to map between *camera space* and *screen space*. Since the interpolation method greatly influences the accuracy of the tracking result, a more sophisticated method than simple linear interpolation may be reasonable. In section 2.4.2 I present a new way to map between both spaces by using *radial-basis functions* for interpolation.

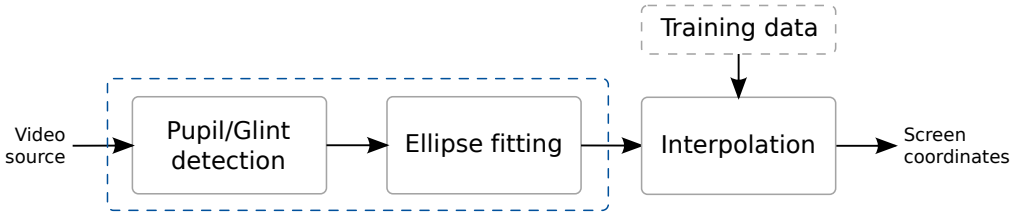


Figure 2.3: A simplified outline of the tracking algorithm. The input of the algorithm consists of a continuous stream of images that are analyzed in two distinct sub-parts. Both parts of the algorithm are discussed more closely in the following subsections. The final result consists of the screen coordinates the user is currently looking at.

2.3 Obtaining the Pupil-Glint-Vector

The most critical part when tracking one persons eye movement is to find the exact position of the pupil. Since the effective resolution of the analyzed image is in general much smaller than the resolution of the used computer screen,[1] the algorithm has to achieve *subpixel accuracy* in order to allow for a sufficient mapping.

Indeed, there are more problems to overcome that are not immediately evident. Noise and a slow shutter speed are very likely to reduce the quality of the acquired images. Additionally, unintentional specular reflections – often caused by environmental lighting or glasses the user may be wearing – can occlude parts of the pupil region.

Dealing with all these situations leads to a balancing act between the robustness of the algorithm and its computational complexity. Since the tracking system shall be used in an interactive environment, low latency is required. Furthermore, when taking embedded hardware into consideration, the complexity of the algorithms may be limited to the capabilities of the hardware environment.

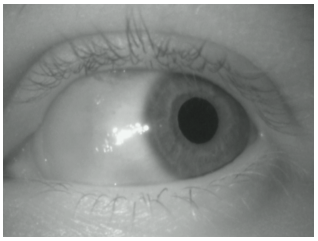


Figure 2.4: A severely distorted glint due to extensive eye movement.

Distortion of the glint

Using the PGV has been proposed in most papers dealing with fixed head gaze tracking. Nevertheless, there may be situations where the position of the glint cannot be

1 Additionally, the pupil moves only in a small area *within* the analyzed image

determined accurately. *Figure 2.4* shows such an incident. If the user's position is too close to the monitor, eye movements become more extensive. As soon as the glint leaves the area of the iris, it becomes severely distorted. The only practical way to deal with such situations is to omit the glint position. Screen space mapping still can be done when using only the absolute position of the pupil. However, this may result in a degradation of accuracy and requires the camera to be fixed to the user's head more firmly.

Figure 2.5 shows a flow-chart of the different steps that have to be undertaken to acquire the pupil and the glint position. Again, each step can be broken up into smaller groups of algorithms that are presented in the following sub-sections.

2.3.1 Preparation of the input image

Before an image is analyzed, it has to be converted into a more suitable format. Since the proposed tracking hardware uses infrared light and is insensitive to visible daylight, the RGB-images delivered by the camera show false-colors. To simplify the algorithms and to save computing time, the images are converted to a single channel 8-bit grayscale format.

Reduction of search area

In order to reduce computation costs of subsequent processing steps, the input image is cropped to the approximate boundary of the eye region. This is done by template matching using a pre-cropped eye region image as a template. In order to avoid quick jumps of the eye region due to misdetections, the eye-area is averaged over several frames. Since this step does not interfere with the accuracy of the following actions, the template matching process operates on a downsampled version of the input image due to its high processing costs.

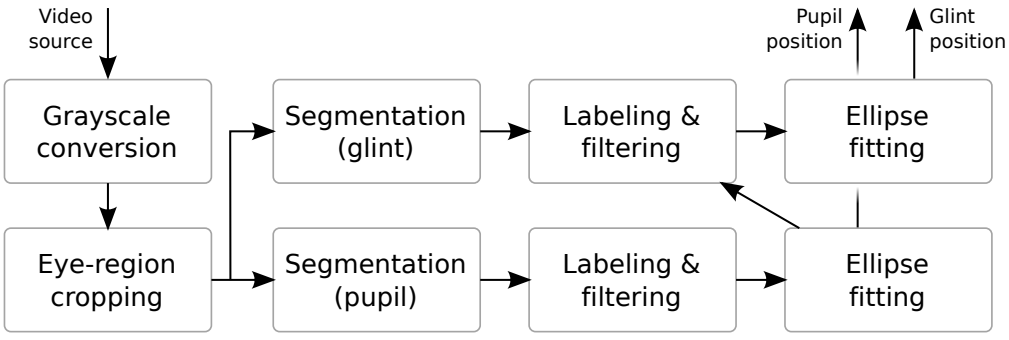


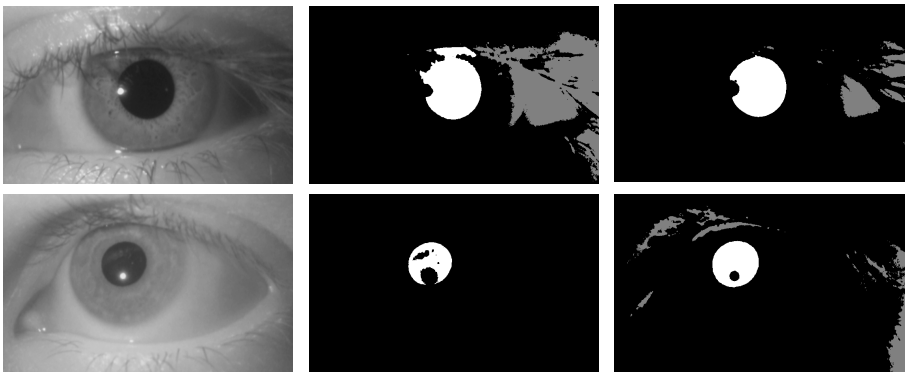
Figure 2.5: A more detailed graph showing the individual steps for gathering the pupil and glint positions.

2.3.2 Image segmentation

In order to search for the pupil and glint regions within the cropped image, it has to be segmented. This is done for each feature (pupil and glint) separately using a simple method known as *thresholding* or *binarization*. Given a *threshold value* t, a binarized version $\boldsymbol{I}_B$ of the input image $\boldsymbol{I}$ is generated as follows:

$$\boldsymbol{I}_B[x,y] = \begin{cases} 0 & \text{if } \boldsymbol{I}[x,y] < t \\ 1 & \text{otherwise} \end{cases}$$

Compared with more sophisticated algorithms like *Region growing* or *Level set* methods, this technique is trivial to implement. Since the features that are searched for can be assumed to be the darkest, respectively the lightest ones within the image, thresholding is a sufficient method for segmentation. However, choosing the right threshold value can be complicated. *Yoo and Chung* suggest using a fixed value, that has been determined experimentally [YC05]. This approach has some major drawbacks: Global changes in illumination or variations in the brightness of the pupil region may result in an insufficient segmentation. Therefore, an optimal threshold value has to be determined for each situation manually. As shown in *figure 2.6*, the segmentation quality varies significantly between different persons when using a fixed threshold value.



(a) Grayscale image **(b)** Fixed threshold **(c)** Proposed method

Figure 2.6: Segmentation results for a fixed threshold value **(b)** and an automatically adjusted one using the proposed method **(c)**. A fixed value may be either to low (*bottom line*) or to high (*upper line*) for segmenting the pupil area exactly. The pupil region (white) has been highlighted for illustration only. It is distinguished from the remaining regions at a later stage.

Automated thresholding

To allow for a more robust segmentation process without the need for determining "magic numbers", I propose an automated thresholding process. For the identification of the optimal threshold value, a gray value histogram $\boldsymbol{h}^{\boldsymbol{I}} \in \mathbb{R}_{\geq 0}^{256}$ of the input image $\boldsymbol{I}$ is created. It consists of 256 bins that correspond to the probability of occurrence of each gray value in the image. The histogram is created by iterating over all pixels of the image and incrementing the value of the corresponding bin by one. Afterwards, each bin is divided by the total number of pixels within the image so that $\sum_k \boldsymbol{h}_k^{\boldsymbol{I}} = 1$ holds. Assuming that the average size of the pupil stays fixed across several persons, the coverage ratio of the pupil compared to the rest of the image may be set as a fixed constant $r_{pupil} \approx 0.06$. This constant is now used to determine the optimal threshold value, separating the pupil from the background:

$$t_{pupil} = \underset{k \in \{0 \ldots 255\}}{\arg\min} \left\| r_{pupil} - \sum_{i=0}^{k} \boldsymbol{h}_i^{\boldsymbol{I}} \right\|$$

As experimental results indicate, the constant r_{pupil} can be used without any adjustment for the majority of probands who tested the device. This approach performs better than fixed value thresholding as shown in *figure 2.6*. The technique can also be used for segmentation of the glint when a ratio-constant of $r_{glint} \approx 0.98$ is chosen.

2.3.3 Pupil and glint detection

In most cases, the segmented image contains numerous defects caused by areas with brightness similar to the pupil region. In order to remove this defective regions, several steps are performed.

At first, connected regions are identified. This is done by using a *connected-component labeling* algorithm as described in *section 2.1.1*. Then, the labeled regions are subject to a series of filters that cancel out erroneous objects. Since the criteria for pupil and glint selection differ, they have to be treated separately.

Detecting the pupil region The pupil detection procedure consists of a chain of three filter criteria that allow for a reliable detection. In the first step, all regions whose sizes are too small are neglected. Afterwards, the areas of the remaining regions are calculated by counting the number of pixels belonging to the label. If the area of the region is larger than 75% of the average pupil-area it is kept for further analysis. The last step removes all objects whose *fill-ratio* differ too much

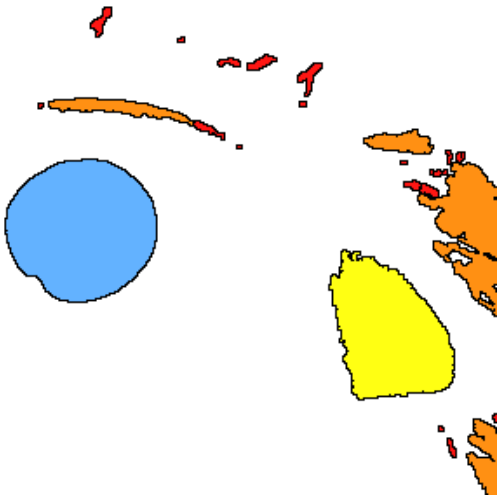


Figure 2.7: When searching for the pupil among the labeled region, very small regions (*red*) are neglected. Also regions that do not fulfil the area criterion (*orange*) are removed. Lastly, the object whose fill-ratio matches the average one best (*blue*) is chosen.

from the average one of the pupil. The fill-ratio is the relation between the number of object pixels and the number of pixels covered by the rectangular bounds of the analyzed region. It is easy to see that for a perfectly circular shaped object this value is close to $\pi/4$.

In order to find the initial pupil region when the average values are not known, the object with the lowest mean gray value and a fill-ratio close to $\pi/4$ is chosen. The average pupil-area is set to zero, effectively disabling the area criterion.

Detecting the glint region Compared with the retrieval of the pupil region, the glint can be obtained relatively easily. Since the region near the iris is mostly free from disturbing reflections, the object next to the current pupil position is chosen. Before doing this, objects that are to small to be considered as a potential glint are removed.

2.3.4 Ellipse fitting

After the regions of pupil and glint have been extracted from the segmented image, the *exact* centres of these objects have to be determined. Of course, taking just the pixel-wise centres of the particular features would never end up in accurate results. Consequently, more sophisticated methods have to be evaluated.

In this section I will introduce two common techniques for gaining subpixel accuracy. The first one utilizes the so called *centroid* of the pupil area, the second one tries to fit an ellipse to the given input data. As it turned out, both techniques have

some severe limitations regarding robustness and stability. Therefore I present two original methods based on the ellipse fitting approach. Depending whether processing speed or accuracy is prioritized, one of these methods can be chosen for operating the gaze tracking device.

Image moment based fitting

A straightforward way to acquire the exact center of an object is to compute its center of mass by using the so called *image moments*. When using discrete input data – like grayscale images –, the *geometric moment* of order p, q of an image $\boldsymbol{I}$ can be determined as follows:

Geometric moment

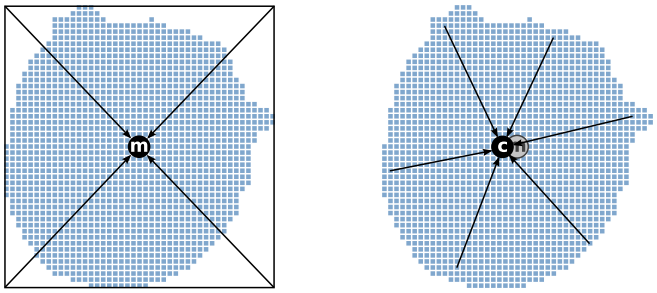
$$m_{p,q} := \sum_{x,y} x^p y^q \boldsymbol{I}[x,y], \qquad p,q \geq 0$$

By using and combining moments of different orders, additional information about the image can be acquired that are translation-, scale- or rotation-invariant [AR05]. However, for determining the center of mass it is sufficient to use first order moments to obtain the so called *centroid* of an image:

Centroid calculation

$$c := \begin{pmatrix} \bar{x} & \bar{y} \end{pmatrix}^\top = \begin{pmatrix} \frac{m_{1,0}}{m_{0,0}} & \frac{m_{0,1}}{m_{0,0}} \end{pmatrix}^\top$$

As shown in *figure 2.8*, using the centroid is very tolerant regarding smaller errors within the input image.



(a) Bounding box **(b)** Centroid

Figure 2.8: Example for calculating the centroid of a binary image. On the left side **(a)** the bounding box of the image contour is used to obtain the center (**m**). The illustration on the right **(b)** shows the calculated position of the centroid (**c**). For acquisition of the centroid *every* pixel of the image is taken into account.

Calculating the centroid of an image is almost trivial to implement and can be done very quickly when utilizing e.g. a dedicated hardware for image processing [Fla11]. However, when very bright spots – like the glint – interfere with the pupil, the centroid no longer depicts its exact center. For the glint region however, such disturbances are very unlikely to happen, making this method an excellent choice for determining its center. For obtaining the pupil position, a more robust technique has to be found.

Least squares ellipse fitting A completely different approach for obtaining the exact centers is taken by fitting an ellipse to the contours of the analyzed object. In order to retrieve a contour of the pupil or the glint, many different techniques are feasible. One of the possible solutions is to use the *Laplace-operator* for edge detection. Like introduced in *section 2.1.2*, the discretized Marr-Hildreth-operator $\boldsymbol{M}$ is used in order to obtain the edges of the input image $\boldsymbol{I}$:

$$\boldsymbol{I}'[x,y] = (\boldsymbol{I} * \boldsymbol{M}_{\sigma=2.0})[x,y]$$

Zero crossing detection

Afterwards, the zero crossings within $\boldsymbol{I}'$ have to be found. This is done using the procedure described in *algorithm 1*. *Figure 2.9* shows an example result of this procedure.

◄ Outlier

Depending on the input image, the result usually is far from being optimal. Points that do not belong to the pupil contour (so called *outliers*) can influence the fitted ellipse. These outliers can be restricted to the inner area of the pupil region by masking the edge points with the segmented pupil-area. This is done by using the binary *and* operator on both images. Nevertheless, there are still numerous outliers within this region. If bright spots (like the glint or other reflections) occupy the pupil area, the fitting result quickly becomes unusable (see *figure 2.9*). Therefore, more effort has to be spent on outlier elimination.

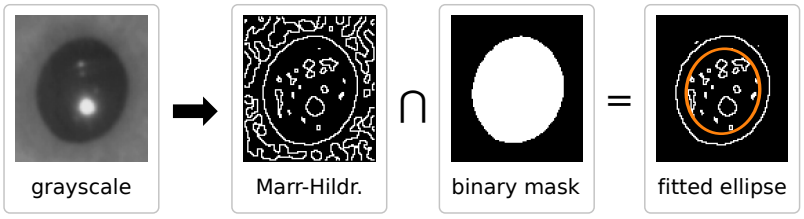


Figure 2.9: A typical situation where outliers deform the fitted ellipse. By masking the edge segments with the binarized pupil area, the outliers are confined to the inner region of the pupil.

Algorithm 1 Determining zero crossings of an image I

```
for all pixel-positions (x,y) in I do
    ref ← sgn(I[x,y])
    I'[x,y] ← 0 {I' will contain the result}
    if ref = 1 then
        skip pixel
    end if
    for all 8-connected neighbors (p,q) of (x,y) do
        if ref ≠ sgn(I[p,q]) then
            I'[x,y] ← 1
        end if
    end for
end for
return I'
```

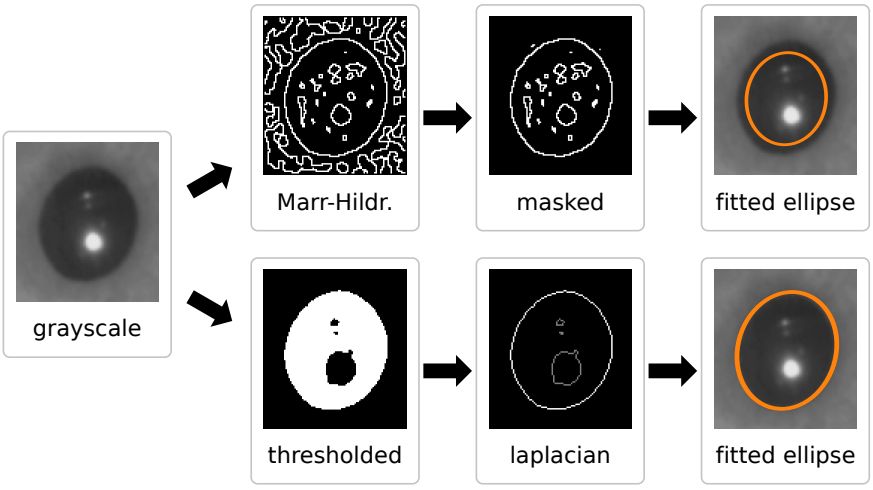


Figure 2.10: Compared with the straightforward ellipse fitting method, the modified method outperforms the former one in most cases. Only the extremity contour taken from the thresholded image is used for fitting the ellipse.

Modified least squares ellipse fitting

The basic idea for the first modification I made to the former approach is, that the segmented/thresholded region of the pupil already appears elliptical. It is easy to see that solely closed contours are created, when the discrete Laplace operator $\boldsymbol{D}_{xy}$ is applied to this binarized image. Depending on how many bright spots are located within the pupil area, additional closed contours may appear inside this region. Since this approach does not operate on the grayscale image like the former one, high frequency noise that may lead to outliers is no longer present.

Basic outlier removal

Next, the contour image is further analyzed to retrieve a set of separated closed contours. For example, this can be done with the algorithm described by *Suzuki et al.* [S+85]. Since the internal contours would influence the ellipse fitting process, only the contour with the largest perimeter is chosen for further processing. This technique performs considerably better than the first one, as can be seen in *figure 2.10*. Furthermore, this method requires less computation time, since no Gaussian smoothing induced by the Marr-Hildreth-operator is required.

Glint induced outliers

However, when the glint[1] comes close to the pupil contour, both contours will be merged and therefore reduce the accuracy of the fitting process. Fortunately this issue can be solved easily, since the position of the glint is already known. *Figure* **??** illustrates how the outliers caused by the glint can be removed. All points lying inside a circular region around the glint position are removed from the pupil contour. The radius for this area can be determined empirically and varies very slightly between different persons. In case of doubt, a larger radius than essentially needed can be chosen.

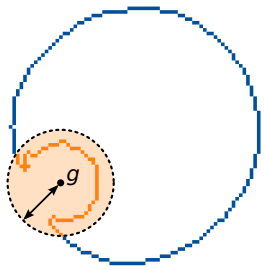


Figure 2.11: Erroneous contour-points caused by the glint can be removed easily since its position is already known. Only the rest of the contour (blue) is used for ellipse fitting.

Enhanced outlier elimination

While the former method for acquiring the pupil position is fast to compute and sufficiently accurate in most cases, there are still situations where this technique breaks down. Possible occasions are poor segmentation of the pupil region or partial occlusion while blinking (see *figure 2.12*). To achieve better accuracy and stability in such difficult situations, I present

1 Of course, this also holds for other bright reflections

a more complex algorithm that includes a different technique for gathering the pupil contour as well as additional filtering for removing outliers.

Instead of using a Laplacian based edge detection method, which performs poorly when operating on noisy data, the Canny-algorithm is used (see *section 2.1.2* for further details). This algorithm tends to be very robust regarding image noise and always returns connected contours.

Outlier elimination

After retrieval of the contours around the pupil region, several criteria are used to determine whether a pixel lies on the pupil ellipse or not. Only the parts of the contour that fulfil *all* these conditions are taken into account in the ellipse fitting process. For these criteria, additional information about the contour pixels are needed. They are acquired by taking the partial derivatives for the pixels into account, thus gaining information about the normal for the points on the contour. The partial derivatives in horizontal and vertical directions can be approximated by convolving the image with the so called *Scharr-operators* which offer a higher rotational symmetry than the Sobel-operators [Sch00]:

Scharr-operator ►

$$\boldsymbol{S}_x = \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}, \qquad \boldsymbol{S}_y = \begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

Next, the gradient information for each contour point is used to feed the following three filters:

Directed contrast filter

This filter is based on the observation that the pixels lying on the pupil contour have a certain characteristic regarding the gray-value of their surrounding pixels. When averaging the pixel values on a line with negative gradient direction, this value is very likely to be near the average pixel value of the pupil itself. The averaged values in opposite direction on the other hand are usually much brighter. In other words: Pixels lying on the pupil contour have a salient characteristic regarding the contrast falloff in gradient direction.

The following term demonstrates the calculation of the *directed contrast* for a contour-point $\boldsymbol{p}$. The gradient vectors taken from the gradient images $\boldsymbol{I} * \boldsymbol{S}_x$ and $\boldsymbol{I} * \boldsymbol{S}_y$ are denoted as $\boldsymbol{g}_{xy}$. The number of sampling points is constituted by n and should be smaller than the average radius of the pupil:

$$v^{\pm}(\boldsymbol{p}) := \frac{1}{n} \sum_{k=1}^{n} \boldsymbol{I} \left[\boldsymbol{p} \pm k \frac{\boldsymbol{g}_{p_1 p_2}}{\|\boldsymbol{g}_{p_1 p_2}\|} \right], \qquad dc(\boldsymbol{p}) := v^{+}(\boldsymbol{p}) - v^{-}(\boldsymbol{p})$$

Now, a point $\boldsymbol{p}$ is considered a pupil-contour point if its directed contrast $dc(\boldsymbol{p})$ is higher than a certain threshold t_p. Empirical results indicate that for a 8-bit input image a value of $t_p \approx 15$ eliminates most outliers inside and outside the pupil region.

Direction filter The former filter criterion eliminates the majority of outliers. This allows for a first guess of the pupil center $\boldsymbol{c}$ by applying the least squares ellipse fitting algorithm to the remaining contour points. Using this estimated pupil position, the second filter can be specified.

The *direction filter* compares the normalized gradient vector for each remaining point with its direction to the center $\boldsymbol{c}$. If the absolute value of the dot-product of both directions is smaller than a certain value[1] the point is kept for the subsequent filtering steps. It is discarded otherwise. Afterwards, $\boldsymbol{c}$ is updated using the remaining edge points.

Magnitude and distance filters The last two filters remove occasional outliers inside the pupil region, where the previous filters were ineffective. At first, the standard deviation for the magnitude of the gradient vectors is calculated. Then the points whose gradient magnitude differs from the average gradient magnitude more than four times the standard deviation are declared as outliers. Afterwards, a similar procedure removes all points whose distance from $\boldsymbol{c}$ differs from the average distance.

The remaining contour points are then used to finally fit the ellipse. Compared with the previous approaches, this method works fairly well even on difficult situations (see *figure 2.12* for a direct comparison). However, this algorithm is far more resource-intensive since the Canny-algorithm is quite complex and gradient information for the contour points has to be acquired.

1 By adjusting this value, the selectivity of the filter can be adjusted. A value of $0.1 \approx \pm 18°$ is appropriate for most situations.

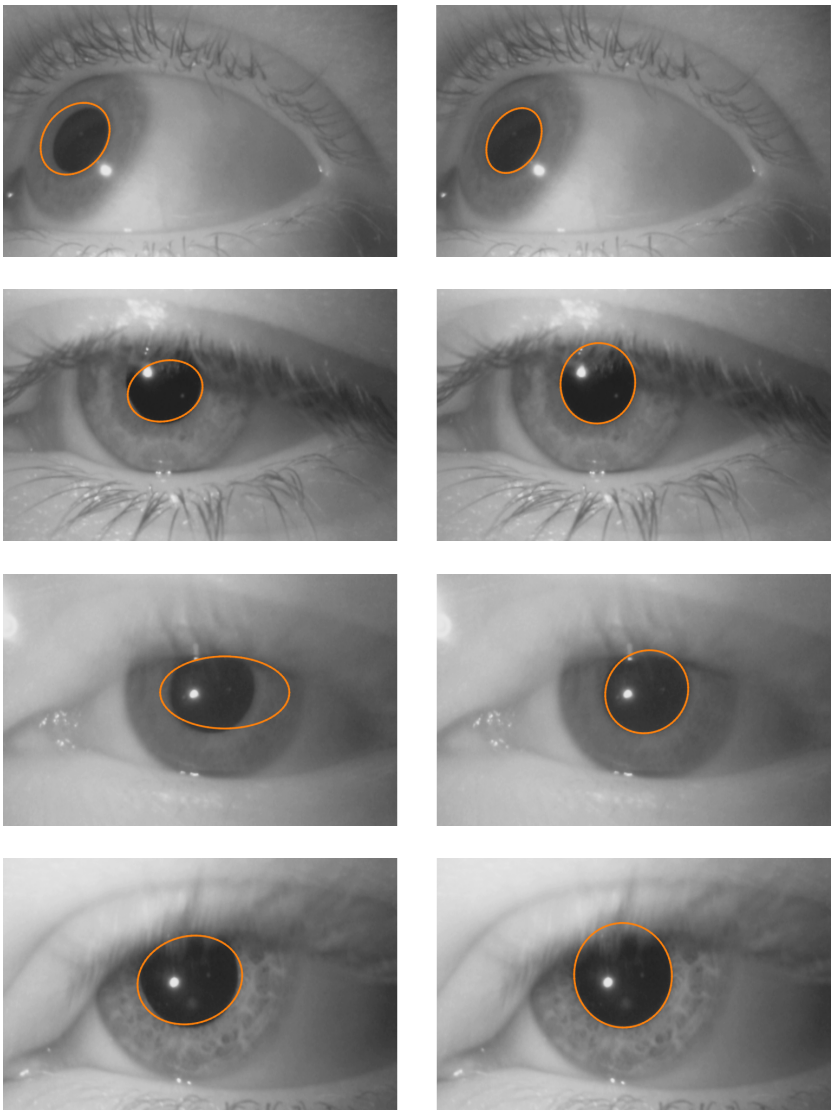


Figure 2.12: Comparison of the last two pupil detection methods. The enhanced outlier elimination procedure allows to fit the pupil more exactly under difficult situations: Poor segmentation of pupil region (first line), partial occlusions (second line) and artifacts caused by the rolling shutter of the camera (third and fourth line).

2.4 Screen space mapping

Once the position of pupil and glint have been determined, the pupil-glint vector (PGV) can be calculated. This vector is then mapped to screen coordinates, allowing to determine the user's fixation point.

Popular approaches

Basically, two popular approaches for doing this mapping are commonly used [HJ10]. The first one – the so called *model-based* approach – tries to solve the problem geometrically by establishing a 3D model of the human eye to determine the point of gaze. The second approach uses a set of calibration points to create a (non-)linear mapping between the input- and the screen-space. This concept is often referred to as *interpolation-based*. Both concepts have individual advantages and drawbacks. Whereas the model-based approaches normally do not need extensive calibration, the attainable accuracy is relatively limited. This is due to the many unknowns like individual eye geometry and the exact orientation of the head relative to the screen. Compared with model-based approaches, the second concept allows for much higher precision, as long as the head position is fixed. However, the calibration of such systems can be very time consuming, since it has to be done for each user separately and requires the involved person to fixate on a series of test-points.

In this thesis I will concentrate on the latter approach. All probands are told to look on a test pattern consisting of nine predefined points shown in random order. The exact test specifications are described in *section 5.2*. Using this calibration data, a mapping function $m : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is established that tries to map the PGV to screen coordinates as precisely as possible.

Section outline

This section describes two techniques for creating this mapping function. The first one – the "classical" linear interpolation – utilizes the so called *Delaunay triangulation* for interpolating between vertices placed on arbitrarily chosen positions in the twodimensional plane. The second method uses a set of *radial-basis functions* for interpolation on scattered data sets. I introduce a specialized kernel-function for the RBF-functions that outperforms the linear interpolation and also works fairly well when using less than nine calibration points.

2.4.1 Linear interpolation

In various digital imaging applications, *bilinear* interpolation is a common and frequently used technique for e. g. resizing images. However, this interpolation method can only be applied if the sampling points are arranged on a regular grid.

Unfortunately, for the sampling points gained from the calibration procedure this is not the case (see *figure 2.13*). Therefore, a more general procedure for multivariate linear interpolation on scattered data sets is needed.

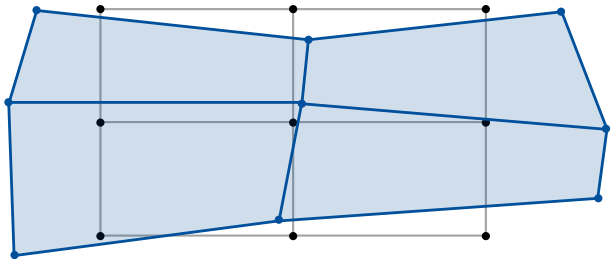


Figure 2.13: An example for a set of sampling points obtained by a calibration procedure: The blue points denote the pupil positions gained by the algorithm presented in *section 2.3*. The black points represent the positions on the computer screen the user has been looking at.

In the first instance the term *interpolation* has to be described more precisely, leading to the definition of the *interpolation problem*:

Interpolation problem

Definition 3. *Given a set of k distinct sampling points* $\{\boldsymbol{x}_i \in \mathbb{R}^n | \; i = 0, \ldots, k\}$ *and their corresponding sampling values* $\{d_i \in \mathbb{R} | \; i = 0, \ldots, k\}$*, the* interpolation problem *is defined as the finding of a function* $f : \mathbb{R}^n \rightarrow \mathbb{R}$ *that fulfils the interpolation condition:*

$$f(\boldsymbol{x}_i) = d_i, \qquad i = 0, \ldots, k$$

In order to find such a function, a *Delaunay triangulation* based interpolation can be used. The Delaunay triangulation of a given set of points can be defined as follows:

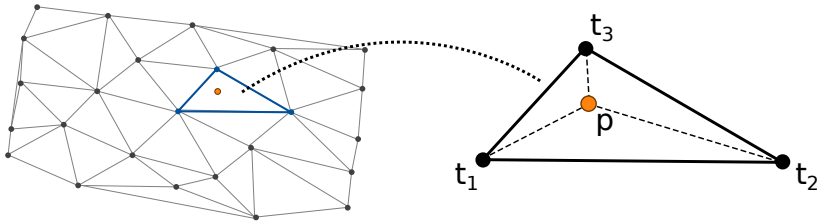


Figure 2.14: Principle of Delaunay triangulation based linear interpolation: The triangle corresponding to the given point $\boldsymbol{p}$ is selected (left side). Then, the interpolation weights are determined (right side).

Definition 4. *Let $P = \{\boldsymbol{x}_i\}_{i=0}^{n}$ be a set of distinct points in $\mathbb{R}^2$. The* Delaunay triangulation *consist of a set of edges that are acquired by connecting any two points $p, q \in P$ for which a circle exists that passes both points but does not contain any other points of P.*

Delaunay triangulation

A comprehensive overview regarding different methods for acquiring such a tessellation is given by *Aurenhammer and Klein* [AK96].

Now, the triangulation of the sampling points gained by the calibration procedure can be used to do the actual interpolation. For this purpose, the triangle that contains the point to be interpolated ($\boldsymbol{p}$) is determined (see *figure 2.14*). Let $\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3 \in \mathbb{R}^2$ denote the vertices of this triangle and $v_1, v_2, v_3 \in \mathbb{R}$ their corresponding sampling values. Then, a function $f(\boldsymbol{x}) = \lambda_1 \boldsymbol{x}_1 + \lambda_2 \boldsymbol{x}_2 + \lambda_3, \lambda_{\{1,2,3\}} \in \mathbb{R}$ exists that interpolates $\boldsymbol{p}$ within this triangle. Since the interpolation condition must be fulfilled, the following equation system can be established:

$$\begin{aligned}
\lambda_1 \boldsymbol{t}_{1,x} + \lambda_2 \boldsymbol{t}_{1,y} + \lambda_3 &= v_1 \\
\lambda_1 \boldsymbol{t}_{2,x} + \lambda_2 \boldsymbol{t}_{2,y} + \lambda_3 &= v_2 \\
\lambda_1 \boldsymbol{t}_{3,x} + \lambda_2 \boldsymbol{t}_{3,y} + \lambda_3 &= v_3
\end{aligned}$$

This linear equation system can be solved easily in order to obtain the three unknowns within $f(\boldsymbol{x})$.

2.4.2 RBF interpolation

The linear interpolation method described in the previous section has two main disadvantages:

- The mapping function $m_{Lin} : \mathbb{R}^2 \rightarrow \mathbb{R}$ obtained by using this type of interpolation is not smooth.
- Points lying outside the Delaunay triangulation cannot be extrapolated.

Many different techniques to access these drawbacks are known including fitting higher order polynomials to the given dataset or using piecewise polynomial functions like splines (cf. [Buh03]). However, in this thesis I want to focus on so called *Radial-basis functions* for interpolation. These can be adapted for function approximation in higher order vector spaces thus becoming very handy when dealing with *head position invariant gaze tracking* as described in *chapter 3*.

Other interpolation methods

RBF ►

A short introduction to radial-basis functions

At first, I would like to give a theoretical foundation about what *radial-basis functions* (RBF) are and how they can be used for multivariate interpolation on scattered datasets.

Radial-basis function

Definition 5. *A function* $\varphi : \mathbb{R}^n \to \mathbb{R}$ *is called* radial-basis function *if its value only depends on the distance from the origin and therefore a* kernel function $\psi : [0,\infty) \to \mathbb{R}$ *does exist, such that* $\varphi(\boldsymbol{x}) = \psi(\|\boldsymbol{x}\|)$, $\boldsymbol{x} \in \mathbb{R}^n$. *A radial-basis function is called* localized *if*

$$\lim_{x\to\infty} \psi(x) = 0$$

holds. It is called nonlocal *if* $\psi(x)$ *becomes unbounded as* x *reaches infinity.*

Typical and frequently used representatives of this class of functions are depicted in *figure 2.15*. By introducing a second vector $\boldsymbol{c}$, RBFs can be arbitrarily placed within the given vector space:

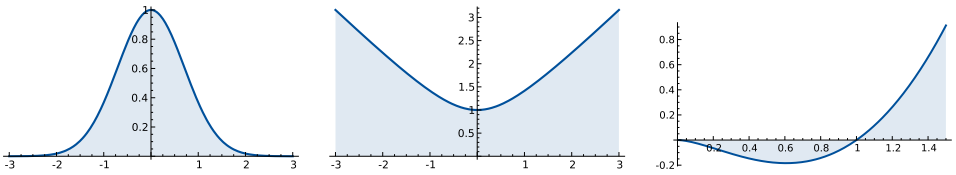
$$\varphi_{\boldsymbol{c}}(\boldsymbol{x}) := \varphi(\boldsymbol{x} - \boldsymbol{c})$$

Interpolation function

The interpolation function then can be obtained by superposition of multiple radial-basis functions placed at the sampling points. Let P be a set of k sampling points as defined in *definition 4*. Then – for a certain *weighting vector* $\boldsymbol{w} \in \mathbb{R}^k$ – the function

$$f : \mathbb{R}^n \to \mathbb{R}$$
$$\boldsymbol{x} \mapsto \sum_{i=1}^{k} \boldsymbol{w}_i \varphi_{\boldsymbol{t}_i}(\boldsymbol{x}) \quad , \quad \boldsymbol{t}_i \in P, \forall i \in \{1, 2, \ldots, k\}$$

satisfies the interpolation condition. In order to acquire $\boldsymbol{w}$, a linear equation system



(a) Gaussian: $\psi_G(x) = e^{-(\varepsilon x)^2}$

(b) Multiquadric: $\psi_M(x) = \sqrt{1 + (\varepsilon x)^2}$

(c) Thin plate spline: $\psi_S(x) = x^2 \ln(x)$

Figure 2.15: Common radial-basis functions.

has to be solved. It can be expressed using the following definition:

Interpolation matrix

Definition 6. *Let* $P = \{\boldsymbol{x}_i\}_{i=0}^{n}$ *be a set of sampling points and* $\varphi(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^m$ *a radial-basis function. Then, the* $n \times n$ *matrix*

$$\boldsymbol{\Phi} = \left\{\varphi(\boldsymbol{x}_i - \boldsymbol{x}_j) \,|\, (i,j) = 1,\ldots,n\right\}$$

is called the interpolation matrix.

Let $\boldsymbol{v} \in \mathbb{R}^k$ be the vector containing the sampling values as stated in *definition 3*. Then, $\boldsymbol{w}$ can be determined by solving the following equation system:

$$\boldsymbol{\Phi}\boldsymbol{w} = \boldsymbol{v} \quad \Rightarrow \quad \boldsymbol{w} = \boldsymbol{\Phi}^{-1}\boldsymbol{v}$$

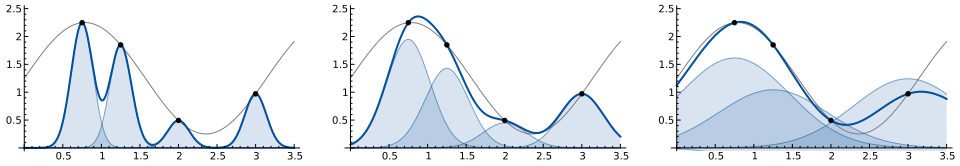
The invertibility of this matrix strongly depends on the chosen radial-basis function φ. Fortunately, a large class of RBFs – including the ones shown in *figure 2.15* – is covered by *Micchellis's theorem* ensuring the existence of $\boldsymbol{\Phi}^{-1}$ [Mic86].

Numerical considerations

The choice of φ leads to another interesting consideration [Hay99]: While *nonlocal* RBFs – like multiquadrics – are known to yield better interpolation results regarding the smoothness of the fitted interpolation surface, they always result in an interpolation matrix that is *not* positive definite. On the contrary, the *localized* Gaussian kernel function always yields a positive definite interpolation matrix, allowing the use of more efficient matrix decomposition methods than e. g. *LU-decomposition*. However, the interpolation matrices that occur by the use of a 9-point calibration procedure are rather small so this behaviour is – in this case – unproblematic.

Influence of the kernel-width

Many radial-basis functions introduce a second parameter ε for controlling the width of the underlying kernel function. This parameter influences the smoothness



(a) $\varepsilon = 5.0$ **(b)** $\varepsilon = 2.5$ **(c)** $\varepsilon = 1.0$

Figure 2.16: Influence of the parameter ε for interpolating a test-function (gray line) using a Gaussian radial-basis function. With increasing width of the kernel function (decreasing ε), the interpolated surface (blue line) becomes more and more smooth.

of the resulting interpolation surface. As seen in *figure 2.16*, a kernel-width that is too narrow may result in separated "peaks" at the sampling points. Of course, this result is still a valid solution of the interpolation problem. With decreasing ε the interpolation function becomes more smooth. Concurrently, the condition of the interpolation matrix $\boldsymbol{\Phi}$ gets worse as ε decreases. Practically, the minimal ε that can be chosen without inducing numerical instabilities depends on the chosen kernel-function and the minimal distance between the sampling points. Therefore, these parameters have to be chosen sensibly in order to compete with other interpolation techniques.

Choosing the right kernel function for the gaze tracker

Spherical distortion

Within this thesis, several radial-basis functions have been evaluated empirically in order to find the best interpolation function for the task stated in *section 2.4*. The central problem of this task is the warp of the PGV induced by the geometry of the surface of the user's cornea that has to be reverted by the mapping function. This surface is known to be *approximately* spherical and varies among different persons.

Due to the lack of additional parameters, the *thin plate spline*

$$\psi_S(x) = x^2 \ln x$$

can be compared directly with the results of the linear interpolation procedure described in *section 2.4.1*. Unfortunately, the performance of the thin plate spline is in most cases inferior to the linear interpolation scheme. Therefore, a more suitable kernel-function has to be found.

Performance of conventional kernel-functions

The interpolation function searched for requires a high degree of smoothness, as *figure 2.13* may indicate. Therefore, when using RBFs like the Gaussian function or the multiquadric, a very small value for ε has to be chosen. In this context the performance of both, the multiquadric and the Gaussian kernel-function, are approximately on a par with the linear mapping approach. For some users who tested the device, the multiquadric

$$\psi_M(x) = \sqrt{1 + (\varepsilon x)^2}$$

yielded better results than the linear interpolation. In all cases its accuracy was higher than the ones obtained with Gaussian or thin plate spline kernels.

Improving the interpolation quality using a customized kernel-function

To improve the quality of the interpolation further, I introduce a new radial-basis

function initially based on the Gaussian kernel ψ_G. It is obtained by deforming ψ_G in a nonlinear way, fitting the whole domain of ψ_G into the interval $[-1,1]$. The kernel-function for this RBF is defined as follows:

$$\psi_B(x) := \begin{cases} e^{\frac{-1}{1-(\varepsilon x)^2}}, & \text{if } \|x\| < \frac{1}{\varepsilon} \\ 0, & \text{else} \end{cases}, \text{ for } 0 < \varepsilon \leq 1$$

It can be shown that this function is infinitely differentiable and has compact support. Functions with these properties are often referred as so called *bump-functions* [Row11]. The origin of this naming becomes evident when looking at *figure 2.4.2* that shows a plot of ψ_B.

Performance of the new kernel function

As experimental results indicate, this function yields much smaller errors than any other method tested within this work and is clearly better suited for the gaze tracker than e. g. the multiquadric. The reason for this good behaviour cannot be stated within the scope of this thesis. However, a noticeable property of ψ_B seems to be its capability to reproduce spherical surfaces better than the Gaussian- as well as the multiquadric kernel-function as seen in *figure 2.18*. Since the possible pupil-centers used for tracking also lie on a sphere (the user's eye-ball), these graphs may give an idea why this kernel-function performs that well. A detailed analysis of the performance of this function can be found in *chapter 5*.

Automatic width-parameter estimation

Finding the right width-parameters for the kernel-functions presented here is not easy. Neither are they constant across different users nor does a simple method exist for determining them.

Determining the optimal kernel-width

In the foregoing paragraph, the parameter has been identified as follows: Let $C = \{\boldsymbol{c}_i\}_{i=1}^{9}$ be the set of calibration-points acquired from the tracking algorithm. Further, let $R = \{\boldsymbol{r}_i\}_{i=1}^{25}$ be a set of reference testpoints as seen in *figure 2.4.2* and furthermore $G = \{\boldsymbol{g}_i\}_{i=1}^{25}$ their corresponding PGVs. For a given RBF φ_ε whose

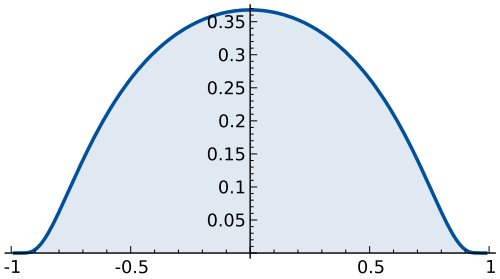
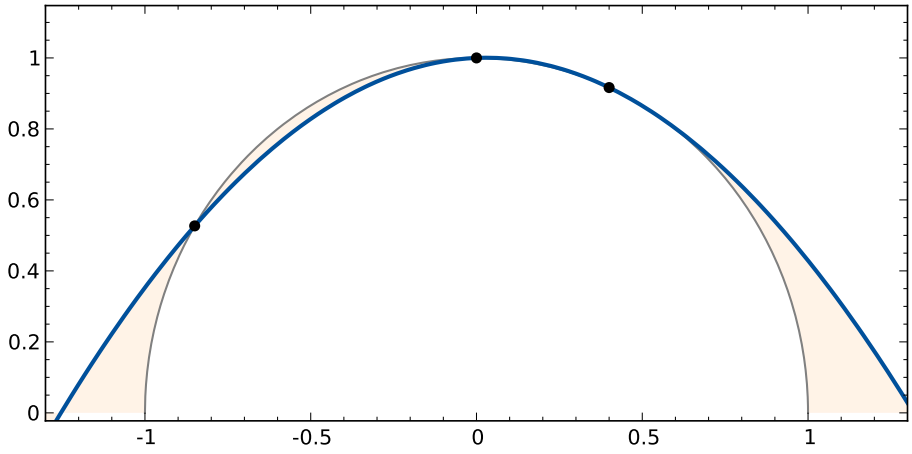
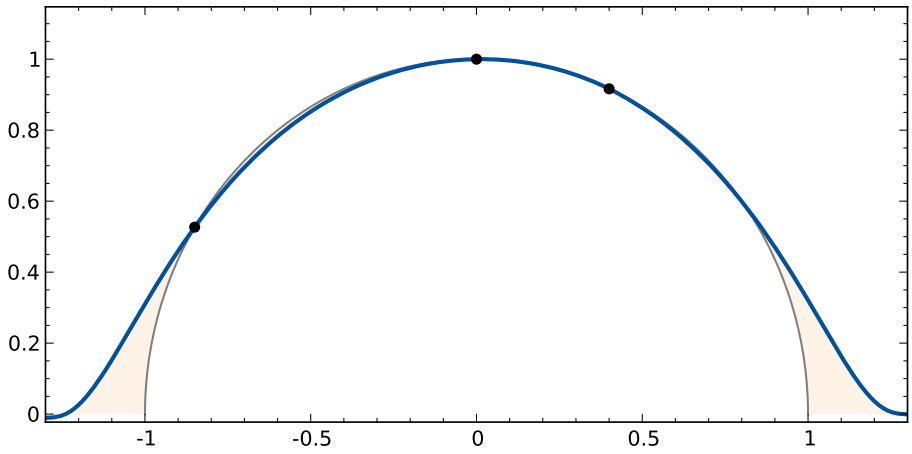


Figure 2.17: The proposed kernel function $\psi_B(x)$ is infinitely differentiable and has compact support. It consists of a Gaussian kernel ψ_G that has been deformed in a nonlinear manner.



(a) Gaussian kernel: $\psi_G(x) = e^{-(\varepsilon x)^2}$, $\varepsilon = 0.1$



(b) Proposed kernel: $\psi_B(x) = e^{\frac{-1}{1-(\varepsilon x)^2}}$, $\varepsilon = 0.73$

Figure 2.18: Comparison of the interpolation performance of the Gaussian kernel-function **(a)** and the proposed one **(b)**. The three sampling points on the test-function $f(x) = \sqrt{1-x^2}$ have been chosen arbitrarily. Even though the width of ψ_B is smaller than the width of ψ_G, the proposed method reproduces the circular shape of $f(x)$ better.

width-parameter depends on ε, let $f_\varepsilon(\boldsymbol{x})$ be the RBF-interpolation function that has been trained with the calibration set C. Now, the optimal width-parameter ε_{min} can be estimated as follows:

$$\delta = \max_{i,j} \|\boldsymbol{g}_i - \boldsymbol{g}_j\|$$

$$\varepsilon_{min} = \arg\min_{0<\varepsilon<\delta} \sum_{i=1}^{25} \|f_\varepsilon(\boldsymbol{g}_i) - \boldsymbol{r}_i\|$$

The search for ε_{min} has been restricted by a parameter δ that ensures a certain smoothness of the interpolation function. This parameter is especially important when using the bump function ψ_B, since a width-parameter that is too large quickly results in a very rough interpolation-surface due to the compact support of ψ_B.

Again, without cheating

Admittedly, this procedure is of no practical value, since it uses the test-data that is designated for examination of the quality of the interpolation function. Therefore, a total of 34 calibration-points have been used effectively. Fortunately, the kernel-function ψ_B still behaves reasonably well when using only five calibration-points (as seen in *figure 2.4.2*: 1,3,5,7 and 9) instead of nine. Now, the remaining points $2, 4, 6$ and 8 can be used for the minimization of the interpolation error as seen before.

Performance of the five-point interpolation

Interestingly enough, the error induced by this five-point interpolation still outperforms the 9-point linear calibration in all tested cases. Therefore, I recommend this type of interpolation since it is easy to set up and performs very well under all tested conditions.

Benefits of radial-basis functions in the context of gaze tracking

Compared to the linear interpolation method as presented in *section 2.4.1*, radial-basis function interpolation performs comparably well as long as the width-parameter has been chosen correctly. This method can be used regardless of the chosen

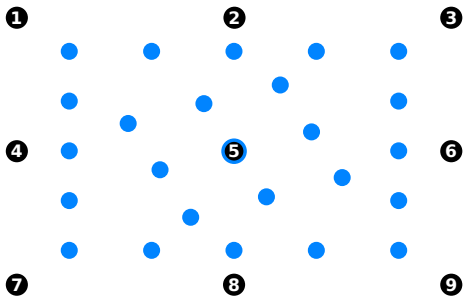


Figure 2.19: Calibration- (black) and testpoints (blue) used to determine the interpolation quality.

calibration pattern, since RBF-interpolation can be performed on any scattered dataset.

Extrapolation properties

Another advantage of this interpolation scheme is its ability to extrapolate screen-coordinates that lie outside the convex hull of the calibration pattern. This allows to reduce the number of calibration points even further: The RBF-interpolation based on the bump-function has been tested to yield acceptable results when using just *three* calibration points (e. g. points $3, 4$ and 9). The Delaunay triangulation based interpolation does not allow for such flexibility without further modification.

Further research

Lastly, RBF-functions are not restricted regarding the dimensionality of the input vectors. Therefore it is possible to integrate additional information into the interpolation process, like the eccentricity and orientation of the fitted pupil-ellipse. This may lead to more accurate tracking results and leaves enough room for further research.

Chapter 3

Head position invariance

The algorithm presented in *chapter 2* has one critical drawback: After the calibration procedure is finished, even faint variances of the head position or its orientation result in displacements between the point of gaze and the mapped screen-coordinates. As the head movement gets more extensive, the tracking result becomes almost unusable. The naïve approach of recalibrating when necessary is impracticable, since it is uncomfortable and time-consuming.

This chapter extends the concepts described in the previous part of this work to operate under natural head movement without inducing such errors. In order to achieve head position invariance, the concept of radial-basis function interpolation is extended to a more general one allowing function approximation in higher dimensional input-spaces. In order to allow a comparison with already existing techniques, the geometrical approach of *Yoo and Chung* [YC05] is presented.

Chapter outline

The current chapter is organized as follows: In the beginning, the reader is introduced to *artificial neural networks* and the structure of the *human eye*. Afterwards, the extension of the algorithm presented in *chapter 2* is described in detail. These modifications allow to compare the tracking method proposed by *Yoo and Chung* with the newly presented method of using *radial-basis function networks* for gaze estimation. Both methods are described and discussed in their respective sections.

3.1 Preliminaries

The modifications done to the video analysis algorithm described in this chapter do not require additional concepts in digital imaging. However, the interpolation algorithm described in *section 2.4.2* has been extended to allow for general approximations. In this context, the bundle of radial-basis functions can be interpreted as an *artifical neural network* . For this reason, the interrelated concepts of such networks are discussed briefly in this section. Additionally, a simplified model of the human eye is described, forming the base of the tracking method proposed by *Yoo and Chung*.

3.1.1 Artificial neural networks

Artificial neural networks are generally known as computational models that have a certain capability to learn and are often said to be inspired by biological neural networks like the human brain. There exists a variety of different network structures and paradigms. Since this section only covers the minimal theoretical foundations required for the rest of this chapter, the interested reader is referred to [Kri07, Hay99] for further reading.

ANN ►

Network structure An *artificial neural network* (ANN) can be described as a set of small independent processing units – the so called *neurons* – that are connected with each other in a certain topology. The connections between the different neurons may have specific *weights* that are often represented in matrix form. The entries of the so called *Hinton diagram* describe the weights between the neurons. An advantage of this representation is that non-connected neurons can be described as a connection with a weight of zero [Kri07]. All in all, an ANN forms a weighted directed graph whose vertices are represented by its neurons.

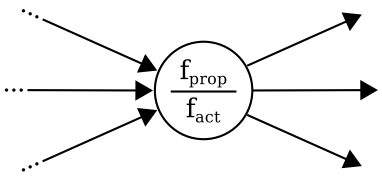


Figure 3.1: Simplified schematic model of an artificial neuron.

Depending on the type of network, the neurons can be described in various ways. A common definition, that covers so called *Perceptron-* and *RBF-networks*, can be stated as follows:

Artificial neuron

Definition 7. *An* artificial neuron *represents a function* $f : \mathbb{R}^n \to \mathbb{R}$ *(with* n *being the individual number of inputs) that is composed of three functions* $f_{prop} : \mathbb{R}^n \to \mathbb{R}$*,* $f_{act} : \mathbb{R} \to \mathbb{R}$ *and* $f_{out} : \mathbb{R} \to \mathbb{R}$*:*

The propagation function f_{prop} *receives the output values of neurons whose outputs are connected with the actual neuron. These values are usually transformed to a single scalar value that is passed to the* activation function*. A commonly used function for doing this is the* weighted sum*, that takes the weights of the Hinton diagram into account when summing up the input values.*

The activation function f_{act} *computes the response for a given scalar input as calculated by the propagation function. For some kinds of ANNs, this function depends on external parameters that are determined during the learning process. In case of so called* Perceptron-networks*, sigmoidal functions are quite common for use as activation functions.*

The output function f_{out} *modifies the result of the activation function to form the final response of the neuron. This function usually implements the identity function and is therefore often omitted for the sake of simplicity [Kri07].*

Such an artificial neuron can be symbolized e. g. in a form as seen in *figure 3.1*. Here, the output function is assumed to be the identity function and is therefore not displayed explicitly. In case the neuron implements only trivial functionality – like the identity function or the weighted sum of inputs – the representation of f_{act} is also omitted.

Common topologies

Commonly used topologies can be divided into two classes, depending on the type of graph the ANN does implement [Kri07]:

Feedforward networks The neurons in a *feedforward network* are organized as a directed acyclic graph. Such networks can be arranged in different layers that may be *completely linked*, i. e. the output of each neuron in a layer is connected with the input of each neuron in the following layer. The first of these layers is commonly called the *input layer*, whereas the last layer is often referred to as the *output layer*. The neurons between these layers form a number of *hidden layers*.

Recurrent networks In case the ANN implements a directed graph that contains cycles, the network is called a *recurrent network*. Depending on the configuration of cycles, the training of such a network can be very difficult. For that

reason, the cycles within a recurrent network are often restricted to the neuron itself (*direct recurrence*), to other neurons within the same layer (*lateral recurrence*) or to the neurons of the foregoing layer (*indirect recurrence*).

Learning process For the training of an ANN, different learning strategies do exist. They are commonly divided into *supervised-*, *unsupervised-* and *reinforcement-*learning strategies. These learning paradigms differ in the type of feedback that is given to the network during the learning phase.

While *supervised* learning strategies provide training-vectors and the desired response-vectors at the same time, *unsupervised* strategies provide the network solely with training-vectors and therefore encourage it to find patterns within this data. The *reinforcement* method may be seen as an intermediate stage between the other paradigms: Here, the network is trained using only training-vectors. After the network has returned a response for the given input stimulus, it is told whether its result was right or wrong.

Further reading

Not all training paradigms can be applied to all kinds of ANNs. Also, the practical implementation depends on many different factors and is far from being trivial. For further information, the reader is referred to the book *Neural networks - A comprehensive foundation* by *Simon Haykin* [Hay99].

3.1.2 A closer look at the human eye

The human eye is a very complex visual organ – not only in terms of geometrical aspects. In the beginning of the 20th century, *Gullstrand* modelled a simplified version of the human eye similar to the one shown in *figure 3.2*. Due to its simplicity, many model-based tracking systems assume such a simplified eye for establishing geometrical relations.

The following description is based on [CRM09]: As seen in *figure 3.2* the outer eyeball consist of the *sclera* ⑤ and the *cornea* ①. The sclera is the white tissue around the eyeball and merges at the frontal side with the transparent cornea that acts as a fixed lens. It is often assumed to be spherical [Spe08]. The *optical axis* ② is the imaginary line that goes through the central points of the cornea and the *lens* ④. Interestingly enough, the *fovea* ⑥ – which is the center of the visual field – does not lie exactly on this axis. Contrary to this spot, that exhibits the highest resolution within the whole visual system, the *macula* ⑦ is insensitive to visual stimuli, since it marks the point where the *optic nerve* ⑧ leaves the eyeball.

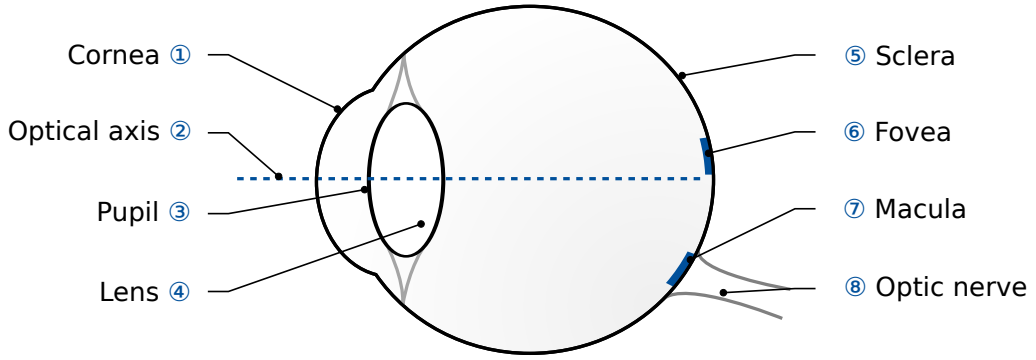


Figure 3.2: A simplified model of the human eye.

3.2 Extension of the algorithms

The former tracking procedure only took the two dimensional pupil-glint vector into account for the whole screen-space mapping. Obviously, this information is not sufficient for compensating head movement or changes in orientation due to the lack of external reference points.

Conceivable setups

An almost self-evident approach to increase the amount of available information is to add additional cameras to the tracking setup, monitoring e. g. the head movement over time. However, this would lead to additional costs regarding both the investment into additional hardware as well as the increased computational load. Another possible solution is to use inertial sensors[1] attached to the user's head that can be quite accurate. Using such an inertia sensor adds additional weight to the tracking device and also induces higher costs of the overall system.

Method used in this work

In the context of this thesis, I want to introduce another approach that is easy to set up, induces only very little costs and appears to be quite elegant. This method adds four additional infrared-LEDs that act as markers at the edges of the computer screen. Since these markers – similar to the glint – cause reflections on the user's cornea, no supplementary cameras are needed. Regarding the additional hardware, similar approaches have been tested successfully and allowed natural head movement to a certain degree [PCG+03, YC05, HL09]. However, to my knowledge the method presented here is the first one incorporating a low cost, head mounted, single camera setup.

1 Commercial vendors like *InterSense Incorporated* offer a variety of different mobile sensor systems: `http://www.intersense.com`

Changes in the tracking algorithm

Since the additional reflections on the user's cornea need to be detected reliably, the algorithm presented in *section 2.3* has to be extended. In order to acquire the positions of these reflections, an additional step within the algorithm-flow is needed just after the PGV has been obtained.

Restricted search-window

In order to limit the additional computation time needed for the detection of the four reflections, the search is restricted to a rectangular region around the glint. The size of this region reflects the aspect-ratio of the computer monitor as well as the approximate size of the user's cornea. Within this region, an iterative thresholding and labeling procedure is carried out consisting of four steps:

1. The image is binarized with a certain threshold value t_r. This value is initialized with the same threshold value as used for the binarization of the glint.
2. A connected component labeling procedure is applied on the binarized image.
3. All labeled regions that are smaller than the glint are kept for further analysis.
4. If the number of valid regions is smaller than 3, t_r is decremented and the whole procedure is repeated.

This loop is prematurely aborted if t_r drops below a lower threshold (e. g. the threshold value used for segmenting the pupil image). In this case, no reflections could be found and the actual frame is discarded. This incident may happen if the user is not looking on the screen or the infrared markers get occluded.

The former procedure results in at least three potential reflection candidates. Depending on the used threshold value and due to unintentional reflections, additional points may appear that have to be filtered out. Before doing this, the subpixel-accurate positions are determined by calculating the centroid for each reflection individually. Now, a certain heuristic is used to find a set of three points among all candidates that appear most likely to be the reflections caused by the infrared-markers. This heuristic is based upon the following assumptions:

- The four markers form the edge-points of a convex quadrilateral that appears almost rectangular as long as the user does not move too far away from the front of the computer screen.
- The size of the quadrilateral does not change abruptly, since the velocity of the user's motion is limited physiologically.

- As long as the user's head faces to the monitor, the distance between the midpoint of the quadrilateral and the glint remains relatively small.

Hardware limitations

Due to constructional limitations of the tracking device (see *section 4.1.2*), one of the four reflections may get occluded by the camera body. Therefore, the algorithm for finding the reflections is restricted to find only three of them. If appropriate reflections have been found, the fourth reflection is approximated by assuming that the quadrilateral caused by the reflections is a parallelogram. Within a circular region around this approximated point, the fourth reflection is searched for. If the initial set of reflection-candidates contains a valid reflection in this region, the approximated point is replaced with the real one. Otherwise, the approximated point is kept. As a matter of cause, this point cannot reproduce the distortion caused by the cornea.

Finding the right set of reflections

Based on the foregoing assumptions, three rating-criteria can be stated that act on a 3-tuple of potential reflection points:

Orthogonality The reflections searched for are arranged almost rectangular in most cases. The first rating allows to measure how close the triangle spanned by the three points is to be right-angled. This is done by calculating the dot-product of the two shortest difference-vectors among the three points that have been normalized. Obviously, points lying on the edges of a rectangle result in a better rating than points that are collinear.

Covered area As stated before, the user only moves slowly in front of the monitor. Therefore the rate of change regarding the area covered by the triangle – that is spanned by the three points – is rather small. Therefore the second rating calculates the difference between the area covered by the given points and the area of the last valid set of reflections from the previous frame.

Rectangle position Lastly, the average euclidean distance of the given points to the glint is calculated and directly used for rating the set of points.

Using these criteria, the total rating of a set is determined by combining them. Since each individual rating should have the same influence to the overall rating, the constants α, β and γ have to be determined empirically in order to yield good results:

$$R_{\text{total}} = \alpha R_{\text{ortho}} + \beta R_{\text{area}} + \gamma R_{\text{distance}}$$

Among all 3-tuples within the set of candidate-reflections, the tuple with the best (lowest) total rating is chosen for the final mapping procedure. Since the candidate reflections are determined using an iterative thresholding technique, the amount of

candidate-reflections usually is very small. Therefore, this straightforward approach does not induce that much additional computation costs.

3.3 Cross-ratio gaze estimation

Model-based tracking approaches often use complex camera setups consisting of multiple video cameras and light sources. *Yoo and Chung* propose a method that does not need such a complex setup and just utilizes the four infrared-markers that are also used in the context of this thesis. They are able to estimate the point of gaze by projecting the four reflections and the pupil-center onto a virtual tangent-plane on the user's cornea. Then, the screen-coordinates the user is fixating at are calculated by using the so called *cross-ratio* that is invariant under projective transformations. These are needed for transforming the virtual tangent-plane to the coordinate system of the camera.

3.3.1 The approach proposed by *Yoo and Chung*

System configuration

The hardware for tracking the user's gaze as described by *Yoo and Chung* consists of several parts. Similar to the setup in this work, four infrared light sources are attached to the corners of the monitor the user is intended to look at. An additional fifth infrared-LED is located at the camera system, causing a bright-eye effect for simpler pupil detection. The camera system consist of two cameras that are mounted on a pan-tilt unit. The first camera, which has a wide field of view, tracks the user's head in order to find the exact location of his eyes. A second camera that is equipped with a zoom lens then provides the system with images showing one of the user's eyes in detail. In order to segment the pupil area more easily, the four monitor-LEDs and the camera-LED are switched on and off by turns, resulting in a dark- and a bright-pupil image. Of course, this procedure is synchronized with the camera system.

Face tracking subsystem

In order to follow the user's head movement, a face tracking system controls the pan-tilt unit. The position of the user's head is determined using a Gaussian modelling of face color [YC05]. This modelling process allows to compute the probability of belonging to the facial region for each pixel in the input image. The center of the user's face is then determined by calculating the centroid (see *section 2.3.4*) using these probabilities. Since the adjustment of the cameras is done mechanically, the tracking process is interrupted for a couple of frames when the user moves his head.

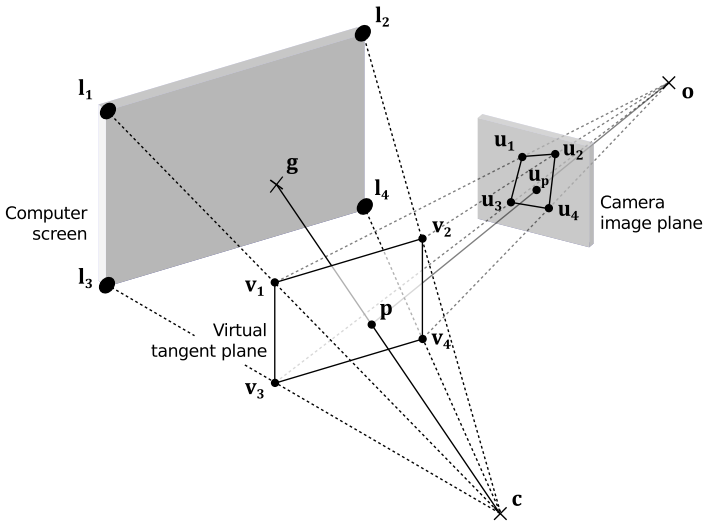
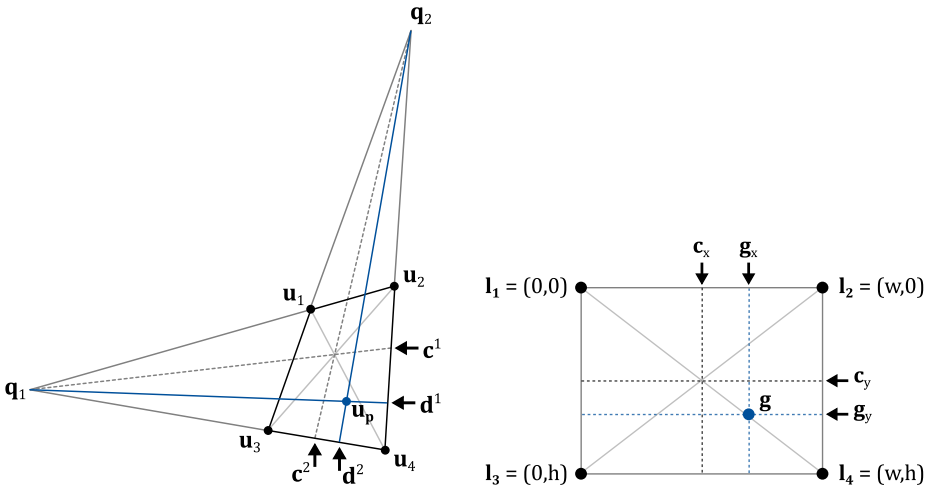


Figure 3.3: Relation between IR-markers and virtual projection points.



(a) Camera image plane **(b)** Screen-space

Figure 3.4: Cross-ratio calculation for different situations.

Assuming the user's eye has been tracked correctly and the pupil-center $\boldsymbol{p}$ as well as the locations of the four reflections $\boldsymbol{r}^1, \ldots, \boldsymbol{r}^4$ have been located within the image, the screen coordinates the user is looking at can be obtained using a cross-ratio calculation.

Cross-ratio calculation

Virtual projection of the monitor

In order to motivate their approach, *Yoo and Chung* assume a virtual tangent plane that is located at the surface of the user's cornea. *Figure 3.3* shows the relation between the different elements of the scene. The four infrared LEDs $\boldsymbol{l}_1, \boldsymbol{l}_2, \boldsymbol{l}_3$ and $\boldsymbol{l}_4$ are located at the edges of the computer screen. Following the lines to the projection center that is the center of the user's cornea $\boldsymbol{c}$, the virtual tangent plane is crossed, forming the *virtual projection points* $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$ and $\boldsymbol{v}_4$. When the user looks at the point $\boldsymbol{g}$ on the monitor, the virtual pupil-center $\boldsymbol{p}$ can be obtained analogously.

Projection to camera plane

Using a second projective transform, the projected monitor screen and the pupil-position can be projected further into five points $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3, \boldsymbol{u}_4$ and $\boldsymbol{u}_p$ lying on the image plane of the camera. As long as the points $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_4, \boldsymbol{p}$ and $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_4, \boldsymbol{u}_p$ are approximately coplanar, the screen coordinates given by $\boldsymbol{g}$ can be obtained since the cross-ratio is invariant under projective transformations. If the dimensions of the computer screen are known, the absolute screen coordinates the user is fixating at can be obtained easily.

Determining screen coordinates

For determining these screen coordinates, two cross-ratio vectors are calculated that correlate directly since the cross-ratio is invariant in projective space. In order to obtain the cross-ratio within the camera image plane, the two vanishing points $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are acquired by using the points $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3$ and $\boldsymbol{u}_4$. As depicted in *figure 3.4 (a)*, the points $\boldsymbol{d}^1$ and $\boldsymbol{d}^2$ can be obtained by intersecting the lines $\overline{\boldsymbol{q}_1\boldsymbol{u}_p}$ and $\overline{\boldsymbol{u}_2\boldsymbol{u}_4}$ ($\overline{\boldsymbol{q}_2\boldsymbol{u}_p}$ and $\overline{\boldsymbol{u}_3\boldsymbol{u}_4}$ respectively). The points $\boldsymbol{c}^1$ and $\boldsymbol{c}^2$ can be acquired in the same manner by using the intersection point of $\overline{\boldsymbol{u}_1\boldsymbol{u}_4}$ and $\overline{\boldsymbol{u}_2\boldsymbol{u}_3}$ instead of $\boldsymbol{u}_p$. These points then can be used to obtain the cross-ratio vector $\boldsymbol{CR}_{\text{image}}$ [YC05]:

$$\boldsymbol{CR}^x_{\text{image}} = \frac{(\boldsymbol{u}_{3,x}\,\boldsymbol{c}^2_y - \boldsymbol{u}_{3,y}\,\boldsymbol{c}^2_x)(\boldsymbol{u}_{4,y}\,\boldsymbol{d}^2_x - \boldsymbol{u}_{4,x}\,\boldsymbol{d}^2_y)}{(\boldsymbol{u}_{3,x}\,\boldsymbol{d}^2_y - \boldsymbol{u}_{3,y}\,\boldsymbol{d}^2_x)(\boldsymbol{u}_{4,y}\,\boldsymbol{c}^2_x - \boldsymbol{u}_{4,x}\,\boldsymbol{c}^2_y)}$$

$$\boldsymbol{CR}^y_{\text{image}} = \frac{(\boldsymbol{u}_{4,x}\,\boldsymbol{c}^1_y - \boldsymbol{u}_{4,y}\,\boldsymbol{c}^1_x)(\boldsymbol{u}_{2,y}\,\boldsymbol{d}^1_x - \boldsymbol{u}_{2,x}\,\boldsymbol{d}^1_y)}{(\boldsymbol{u}_{4,x}\,\boldsymbol{d}^1_y - \boldsymbol{u}_{4,y}\,\boldsymbol{d}^1_x)(\boldsymbol{u}_{2,y}\,\boldsymbol{c}^1_x - \boldsymbol{u}_{2,x}\,\boldsymbol{c}^1_y)}$$

As the computer screen is rectangular, calculating the cross-ratio $\boldsymbol{CR}_{\text{screen}}$ does not involve vanishing points. According to *figure 3.4 (b)* it can be calculated as follows:

$$\boldsymbol{CR}^{x}_{\text{screen}} = \frac{\boldsymbol{g}_x}{w - \boldsymbol{g}_x}$$
$$\boldsymbol{CR}^{y}_{\text{screen}} = \frac{\boldsymbol{g}_y}{h - \boldsymbol{g}_y}$$

Since both cross-ratios are equal, the user's gaze-position can finally be determined by combining both cross-ratio terms:

$$\boldsymbol{g}_x = \frac{w \cdot \boldsymbol{CR}^{x}_{\text{image}}}{1 + \boldsymbol{CR}^{x}_{\text{image}}}$$
$$\boldsymbol{g}_y = \frac{h \cdot \boldsymbol{CR}^{y}_{\text{image}}}{1 + \boldsymbol{CR}^{y}_{\text{image}}}$$

Acquisition of the virtual projection points

The former calculations assumed that the virtual projection points $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$ and $\boldsymbol{v}_4$ are already known. Unfortunately, only the reflections on the user's cornea can be localized directly within the *camera image plane*. Determining the points $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3$ and $\boldsymbol{u}_4$ from the given positions of the four reflections is a highly complex task that cannot be accomplished directly since it includes various unknowns. By restricting the allowed head movement and making a series of simplifications, *Yoo and Chung* accomplished to describe the complex relations between camera orientation, cornea radius, head position relative to the camera, etc. by using only a very simple term:

$$\boldsymbol{u}_n = \boldsymbol{u}_c + \alpha_n(\boldsymbol{r}_n - \boldsymbol{u}_c), \quad n = \{1,2,3,4\}$$

Here, $\boldsymbol{u}_c$ denotes the position of the glint (with the corresponding infrared LED located near the optical axis of the camera) and $\boldsymbol{r}_n$ marks the positions of the four reflections within the camera image plane. The scalar values α_n are acquired using a relatively short calibration procedure where the user has to look directly at the individual infrared LEDs.

Further reading

For further details about the calibration process and the geometrical relations that have been simplified, the reader is referred to the papers by *Yoo and Chung* [YC05]

and *Coutinho and Morimoto* [CM06] who improved the accuracy of the system and also give further details about the geometrical relations.

3.3.2 Adaption to own tracking hardware

In order to allow a comparison between the approach used in this thesis (see *section 3.4*) and the method by *Yoo and Chung*, the previously described approach has been implemented and tested with the available tracking hardware. The results of the method in conjunction with the tracking device will be summed up in *chapter 5*. However, the performance of this combination is inferior to the results of the original paper. The following points may give an explanation for this behaviour:

- In the original paper, a remote tracking system with a fixed camera origin relative to the computer screen is used. The camera origin of the head mounted gaze tracker is fixed relative to the user's head. Therefore, some of the assumptions made for the simplification of the geometrical relations do not hold anymore and induce inaccuracies.
- The tracking system used in this thesis utilizes the dark-pupil method for obtaining the pupil-center. Therefore, the infrared LED that causes the glint is not located near the optical axis of the camera as assumed by *Yoo and Chung*. This results in an (unknown) offset for the vector $\boldsymbol{u}_c$ that is used for the calculation of the virtual projection points.
- As already mentioned before, one of the four reflections gets frequently covered by the camera of the tracking device. In this case, the approximated reflection position is used for calculating the cross-ratio. Of course, this induces further inaccuracies especially under large head movement.

3.4 RBF-Networks for gaze estimation

Motivation for using ANNs

The former – geometrically inspired – approach uses many strict assumptions and limitations regarding the head position of the user. The simplifications made to the geometrical relations are absolutely essential in order to overcome the complexity of the problem. A completely different approach for handling this problem is to use a neuronal network that is able to model these relations *implicitly* over time by learning from a given set of training vectors. If the data fed to the net is sufficient for modelling the problem and the training database is large enough, there is hope

for finding an approximation of the geometrical relations that induces smaller errors than the simplifications done in the geometrical approach.

Multivariate interpolation

Such an artificial neural network can be modelled upon radial-basis functions that have been introduced in *section 2.4.2*. Since the RBF-interpolation is able to handle higher dimensional input vectors easily, one may ask why the training-data is not directly used for creating an interpolation function as done before for fixed-head tracking. Indeed, using such a high dimensional RBF-interpolation is possible and leads to so called *RBF-networks* that are discussed later.

Choosing the right input vectors

Before describing the different kinds of networks that can be used for the task of finding an appropriate approximation function, the number of dimensions used for the input-vectors has to be determined.

Appearance-based methods

Some papers using an appearance-based tracking procedure propose to use whole images of the user's eye that may be scaled down or are further preprocessed [BP94, XMS98]. However, using such large input vectors has certain disadvantages. Obviously, the computation time for processing the large ANNs needed for analyzing the images is very high and may be insufficient for interactive usage of the tracking systems. Additionally, such direct methods are very sensitive to e. g. changes in illumination [HJ10].

Model-based methods

Geometrical motivated approaches on the other hand – just like the method by *Yoo and Chung* – allow for a decent tracking accuracy while being much more computationally efficient. The technique described in *section 3.3* uses a total of six input vectors, describing the position of the five reflections and the pupil-center within the camera image. This 12-dimensional feature-vector is much smaller than the ones used in appearance-based approaches and allows for creating a more efficient artificial neuronal network design. However, it is possible to use even smaller input vectors. For example, *Guestrin and Eizenman* present a tracking system that is able to compensate for head movement using just two infrared light sources in addition to the pupil-center. They also give a mathematical explanation why this setup is the minimal possible one allowing for tracking the user's point of gaze under head movement [GE06].

Feature vectors used in this work

The tracking-algorithm as described in *section 3.2* is able to obtain a total of six subpixel-accurate feature vectors: The positions of the pupil-center, the glint caused by the camera-LED and the four reflections caused by the infrared-markers at the

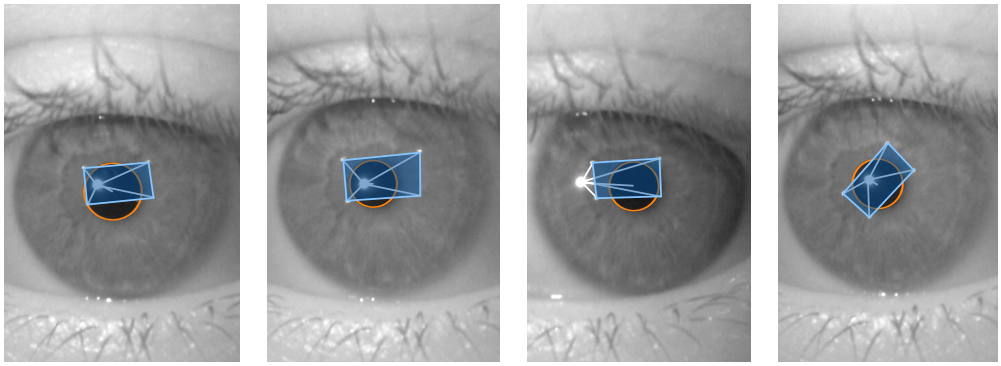


Figure 3.5: Positions of the different extracted feature-points under head-movement while fixating the same point on the screen. From left to right: Initial position, after sagittal movement, transversal movement and rotation of the head.

monitor. Since one of the four reflections is frequently covered by the tracking-device, only a total of five vectors can be acquired reliably. These vectors can be used as a combined input-vector for artificial neural networks. However, as experimental result indicate, the accuracy of the system can be increased when the feature vectors are transformed in a way such that the glint is the origin of their coordinate system instead of using absolute coordinates. This also allows to omit the glint-vector thus reducing the total dimension of the input-vector to 8 dimensions. In *figure 3.5* an example for the relations between the different feature-vectors is shown.

Radial-basis functions for strict function interpolation

The direct RBF-interpolation method in conjunction with the Gaussian radial-basis function – theoretically – can also be used for interpolation in higher dimensional spaces. Similar to the method described in *section 2.4.2*, a set of training-vectors is used to build up the interpolation matrix $\boldsymbol{\Phi}$. For the sake of simplicity, all RBFs share a common kernel-width $\varepsilon > 0$. By inverting this matrix, the weights for the final linear combination of RBF-responses can be obtained.

Since the Gaussian function is used as a kernel-function, the interpolation matrix is always positive definite and can therefore be inverted efficiently. However, the practical use of this approach is rather limited. As the number of input dimensions increases, more training-vectors are needed in order to obtain an interpolation function that represents the underlying – and also unknown – mapping function

sufficiently. This results in some unintended side effects limiting the applicability of this approach:

- For each individual training-vector, a separate RBF is used. When using large training-sets (that may be needed when modelling complex problems) a large amount of RBFs is needed, thus increasing the size of the interpolation matrix. As N increases, the time needed for inverting the $N \times N$ interpolation matrix grows polynomially.
- If the interpolation matrix gets larger, it is more likely to be *ill conditioned*. As a direct consequence, numerical instabilities may occur.
- Until now, the training-vectors have been supposed to be accurate, i. e. they do not contain outliers. Since the tracking-algorithm may sometimes misdetect the user's pupil or one of the four reflections, this is *not* a justifiable assumption. Due to the fact that all training-vectors are taken into account when doing a strict interpolation, the resulting mapping-function carries these defects over from the misdetected training-vectors.

The latter effect is due to a missing procedure for detecting and removing outliers within the training-set. Nevertheless, detecting outliers reliably without removing valid training-vectors is a non-trivial task. The negative effect of outliers can be reduced by using a so called *regularization-network* for approximating the function searched for [Hay99]. In the special case of the RBF-function being the Gaussian one, the result of the strict interpolation can be *regularized* by introducing an additional factor λ that controls the smoothness of the result. Using an appropriate value for λ, the weights for the approximated mapping-function can be determined as follows:

◀ Regularization-network

$$\boldsymbol{w} = (\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{v}$$

Where $\boldsymbol{I}$ is the $N \times N$ identity matrix. Regardless if the result is regularized or not, the size of the resulting matrix still prohibits practical usage.

3.4.1 RBF-Networks for function approximation

The approach presented in the previous section – as long as regularization is applied – is able to yield good results in terms of accuracy. However, the amount of training-data that has to be used in order to find an adequate mapping function is too large for this method in general. Additionally, the runtime performance of

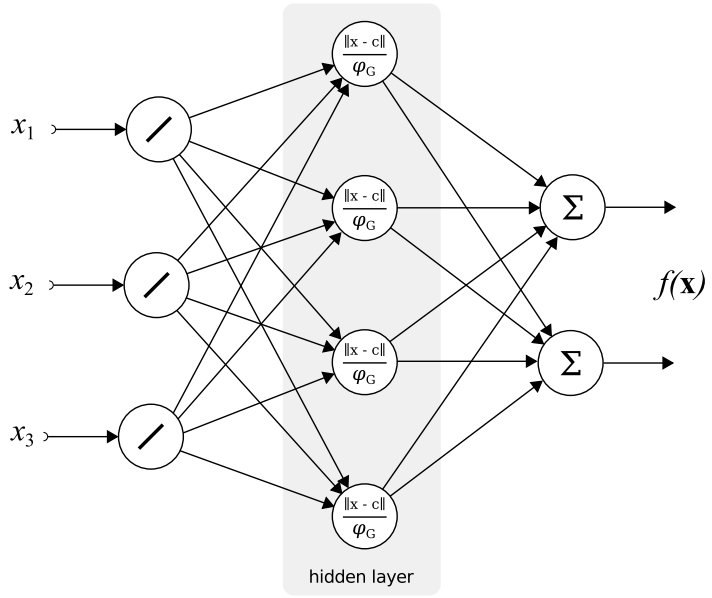


Figure 3.6: An example RBF-network consisting of three input-, four hidden- and two output-neurons.

the system would not be suitable for interactive use, since *all* radial-basis functions have to be queried for obtaining a single gaze-estimation.

Both problems can be solved by limiting the amount of RBFs used in the mapping-function. Apparently, this approach results in a search for a suboptimal solution in a lower dimensional space that approximates the mapping function as best as possible. In order to allow a real-time evaluation of the function, the amount of RBFs typically should be much smaller than the amount of training-data available. Obviously, a strict interpolation cannot be achieved when limiting the amount of used sampling points.

Function approximation

RBF-network ►

When generalizing the concept of radial-basis functions for doing function approximation instead of strict interpolation, it is reasonable to treat the resulting construct as a special form of artificial neural network. Such a *RBF-network* consists of *exactly* three different layers (see *figure 3.6*):

The input layer just distributes the input vector across the hidden neurons in order to form a *completely linked* network. The neurons of this layer represent the *identity function*.

The hidden layer consist of $n \ll N$ radial-basis functions with a Gaussian kernel (where N denotes the number of training-vectors). In terms of ANNs, the neurons representing the RBFs have a certain *center* $\boldsymbol{c}_\varphi$ that represents the point in input-space where the related RBF is located.[1] The *propagation function* of such an *RBF-neuron* measures the distance of the input-vector $\boldsymbol{x}$ and the neuron-center by evaluating $\|\boldsymbol{x} - \boldsymbol{c}_\varphi\|$. The *activation function* consists of the underlying RBF-kernel-function ψ_G that creates the response of the neuron.

◀ RBF-neuron

The output layer contains one or more neurons returning a weighted sum (i. e. a linear combination) of the responses of the neurons in the hidden layer. Each output neuron uses an individual set of weights and is connected with *every* neuron in the hidden layer.

In contrast to other types of artificial neural networks – such as the *multilayer perceptron* – RBF-networks only need a single layer of hidden neurons to be able to approximate a given function. Indeed, *Park and Sandberg* give proof that such networks are *universal approximators*, i. e. they are able to approximate any given continuous function on a compact set [PS91]. To be more precise, the *universal approximation theorem for RBF-networks* can be stated as done by *Haykin*:

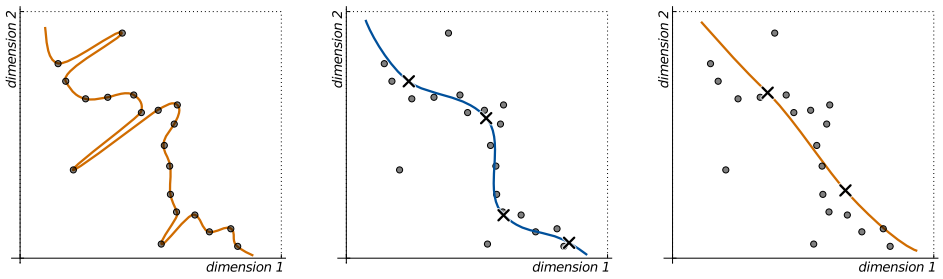
> *"For any continuous input-output mapping function $f(\boldsymbol{x})$ there is an RBF network with a set of centers $\{\boldsymbol{t}_i\}_{i=1}^{m_1}$ and a common width $\sigma > 0$ such that the input-output mapping function $F(\boldsymbol{x})$ realized by the RBF network is close to $f(\boldsymbol{x})$ in the L_p norm, $p \in [1,\infty]$."* [Hay99, p. 291]

Strategies for training RBF-networks

Unlike the "training" in the case of strict function interpolation,[2] an RBF-network contains many unknowns that have to be determined in order to find the optimal approximation. These variables affects several properties of the RBF-network:

Neuron count The number of used neurons directly influences the maximum accuracy of the approximation function. Obviously, a larger amount of RBFs allows for a higher precision than a smaller one. Since this parameter also influences the runtime performance of the network (once it has been trained), the most favorable strategy is to keep this value as small as tolerable. As

1 This center does *not* have to correspond with any training-vector
2 The training only implicates determining the unique solution of a system of linear equations



(a) Overfitted result **(b)** "Correct" result **(c)** Underfitted result

Figure 3.7: Depending on the number and centers of neurons, the approximation-surface may emphasize on outliers (a) or may be too smooth (c). Only an adequate configuration leads to the desired result (b).

figure 3.7 illustrates, outliers may also induce a negative effect if the neuron count is higher than practically needed.

Position of neuron centers Unlike the positions of the RBFs used for interpolation, the centres of the RBF-neurons do not need to be placed at the exact positions of the training-vectors. Of course, the structure of the input-data does influence the optimal positions for the given amount of neurons. However, it is often more reasonable to cover the input-space uniformly while focusing the neuron concentration on areas within input-space that require higher precision than others.

Width of RBF-kernels As already seen in *section 2.4.2*, the width-factor used for the kernel-functions has a great impact on the overall quality of the mapping-function. This also holds for higher dimensional input-spaces. Indeed, as the dimension of the input-space increases, a common width for all neurons may not be the optimal choice. On the one hand, using individual width-factors for each neuron would be a reasonable diversification since some regions in input-space may require a more smooth approximation than others. On the other hand, this would also increase the complexity of the minimization problem excessively.

Linear weights of output neurons Fortunately, the computation of the linear weights does not add much additional complexity to the overall minimization problem since the hidden- and the output-layer are typically computed separately [Hay99].

Of course, the determination of the linear weights also can be incorporated into the whole minimization process. In this case, the RBF-network takes on its most flexible and generalized form. However, finding a combination of parameters that minimizes the approximation-error as best as possible is a highly complex and nonlinear task. The typically used gradient-descend approaches often do not yield convincing results in this context [Kri07].

Reducing the complexity

Due to the difficulties when trying to determine all parameters at the same time, the training process is usually subdivided into separate steps while still being able to yield acceptable results. Such an approach is also used in the context of this work. In this case, the positions, the widths and the linear weights of the neurons are determined sequentially, thus reducing the overall complexity of the problem.

Simplified training procedure

The training procedure used in this thesis assumes that the number of neurons has already been set to a fixed value that depends on the computation power that is available as well as the desired smoothness of the approximation function. At first, the n available neurons are placed at fixed positions within the input-space. These positions are determined using a cluster analysis technique that examines the given training-vectors as described in *section 3.4.2*. After the neurons have been positioned, a common width ε is used for all neurons that depends on the maximum distance d_{max} across all neurons as well as the number of neurons:

$$\varepsilon = \frac{d_{max}}{\sqrt{2n}}$$

This width-factor ensures that the underlying kernel-functions are neither too peaked nor too wide [Hay99]. As experiments indicate, this common width-factor yields good results in most of the cases, depending on the neuron positions. Lastly, using the fixed positions and widths of the neurons, the linear weights for the output-layer have to be determined. The technique for doing this is known as the *pseudoinverse method* [Bro88].

Moore-Penrose pseudoinverse

Definition 8. *For a given matrix $\boldsymbol{A} \in \mathbb{K}^{m \times n}$, the* Moore-Penrose pseudoinverse *of $\boldsymbol{A}$ is defined as the matrix $\boldsymbol{A}^+ \in \mathbb{K}^{n \times m}$ that satisfies the following criteria:*

1. $\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{A}$
2. $\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}^+$
3. $(\boldsymbol{A}\boldsymbol{A}^+)^* = \boldsymbol{A}\boldsymbol{A}^+$
4. $(\boldsymbol{A}^+\boldsymbol{A})^* = \boldsymbol{A}^+\boldsymbol{A}$

Obtaining the weight-vector

Using the centres of the individual neurons $\{\boldsymbol{c}_i\}_{i=1}^{n}$ and the given training-vectors $\{\boldsymbol{t}_j\}_{j=1}^{m}$, a matrix $\boldsymbol{\Theta} \in \mathbb{R}^{m \times n}$ can be established that is quite similar to the already known interpolation matrix:

$$\boldsymbol{\Theta} = \left\{ \varphi_G(\boldsymbol{t}_j - \boldsymbol{c}_i) \,|\, i = 1, \ldots, n;\ j = 1, \ldots, m \right\}$$

Now, the weights $\boldsymbol{w}$ for the output-neurons can be determined by using the Moore-Penrose pseudoinverse of this matrix and the vector $\boldsymbol{v}$ containing the sampling-values:

$$\boldsymbol{w} = \boldsymbol{\Theta}^{+} \boldsymbol{v}$$

Assuming that all $\boldsymbol{c}_i$ are distinct and $m > n$, $\boldsymbol{\Theta}$ leads to an overdetermined system of linear equations. In this context, the Moore-Penrose pseudoinverse has the desirable property of solving this system optimally in a least-squares sense [Pen56].

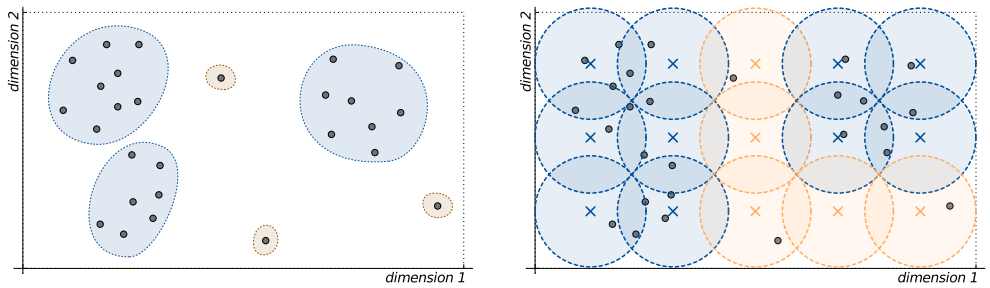
Determining the pseudoinverse

Many efficient implementations for determining the pseudoinverse of a matrix do exist. They usually involve a *Singular Value Decomposition* (SVD) of the matrix for calculating it. Suppose the matrix $\boldsymbol{\Theta}$ has been decomposed using the SVD, such that $\boldsymbol{\Theta} = \boldsymbol{U}^{\top} \boldsymbol{\Sigma} \boldsymbol{V}$ holds with $\boldsymbol{U}$ containing the left- and $\boldsymbol{V}$ containing the right-singular vectors. Then, the Moore-Penrose pseudoinverse of $\boldsymbol{\Theta}$ can be obtained by evaluating $\boldsymbol{\Theta}^{+} = \boldsymbol{V} \boldsymbol{\Sigma}^{+} \boldsymbol{U}^{\top}$ where $\boldsymbol{\Sigma}^{+}$ is the matrix obtained by replacing all nonzero diagonal entries of $\boldsymbol{\Sigma}$ with their respective reciprocals and transposing the result.

Finally, after all parameters of the RBF-network have been acquired, the network can be used for estimating the user's point of gaze by feeding the network directly with the data obtained by the tracking-algorithm.

3.4.2 Determining the optimal neuron positions

Even if all neurons share a common width, i. e. share the same ε for the underlying kernel-functions, placing the neurons in a way that yields good results and covers the training data sufficiently is a non-trivial task. An example for this problem is given in *figure 3.8*. For the sake of clarity, this example has been reduced to a two dimensional problem. The illustration on the left (a) shows a set of training-vectors that contains valid training-data (blue) as well as some isolated outliers (red). Now, the neurons of the RBF-network have to be placed in a way that covers the training samples as best as possible without emphasizing the isolated outliers.



(a) Training data in 2D input space

(b) Even coverage of input space

Figure 3.8: Example situation in 2D input-space: The given training-vectors can be either classified as valid data (left image, blue) or outliers (red).

Even neuron distribution

A straightforward approach for covering the whole training data is to place the neurons in a way that covers the whole input space.[1] The illustration on the right side (b) depicts such an strategy applied to the example situation. Here, a total of 15 neurons have been placed evenly, such that all training-vectors are covered. It is easy to see that even in such a simplified situation, some neurons are "wasted" since they do not cover any relevant information (marked in red) and therefore do not contribute to the quality of the final approximation function.

Curse of dimensionality

When increasing the dimension of the input space, the situation gets worse: The volume that has to be covered by the neurons grows exponentially with the number of dimensions. On the contrary, the available data provided by the training-vectors becomes more and more sparse. This phenomenon is often referred to as *the curse of dimensionality* due to *Richard E. Bellman* [Bel61]. Obviously, such a strategy is only applicable if the input-dimension is small enough.[2]

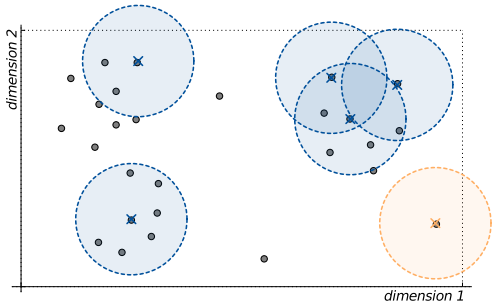


Figure 3.9: Random positioned neurons may emphasize on outliers (bottom right) or do not cover the whole training set (top left).

1 Of course, the intervals the neurons are placed in have to be restricted in some way.
2 As a reminder: The tracking-algorithm yields 8-dimensional input-vectors.

Random positions

Another simple – and fairly inelegant – approach for placing the neurons is to position them at random points based on the training-vectors as seen in *figure 3.9*. Here, the neurons have been placed exactly at the point in input-space that is given by the respective training-vector. As long as the number of neurons placed in this way is high enough and a sufficient amount of training-data is given, this method can perform quite well. However, depending on the distribution of training-vectors, chances are high that single outliers are chosen as neuron position or some regions are not covered evenly enough.

Clustering techniques

In many situations, the training-vectors show a certain distribution within the input-space. Some regions may contain a relatively high concentration of vectors, forming so called *clusters*. *Figure 3.8 (a)* shows an example for a (manually done) clustering of training-vectors. This illustration suggests that the centres of these regions are reasonable neuron-positions.

For a given set of training-vectors, a variety of clustering techniques do exist. Within the extend of this work, two methods have been evaluated. The relatively ordinary *K-means clustering* method can be seen as an entry point for more sophisticated clustering techniques. The so called *regional and online learnable fields* presented in *section 3.4.3* are reported to be useful in the context of RBF-networks [Kri07].

Mean ►

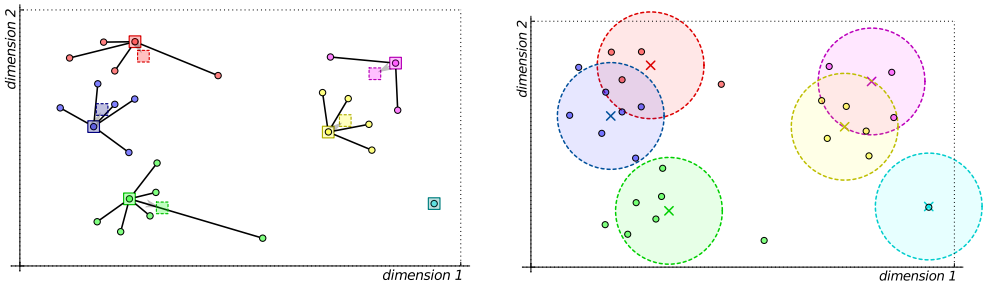
K-means clustering The *K-means clustering* method is an iterative two step algorithm that tries to find K so called *means* $\boldsymbol{m}_k \in \mathbb{R}^m$ that represent the centres of clusters covering the N given training-vectors $\boldsymbol{t}_n$ [Mac02]. At first, the means are initialized with arbitrary positions, e. g. with random positions taken from the training-vectors. Then, the following two steps are repeated until convergence is reached, i. e. the means $\boldsymbol{m}_k$ stay at fixed positions:

Assignment step For each training-vector $\boldsymbol{t}_n$, the mean that has the smallest euclidean distance to $\boldsymbol{t}_n$ is assigned to it.

Update step The *center of mass* for each group of training-vectors belonging to the same mean-vector is calculated. Afterwards, the respective mean is set to this new position, that can be calculated as

$$\boldsymbol{m}_k^{(t+1)} = \frac{1}{|R_k^{(t)}|} \sum_{\boldsymbol{r} \in R_k^{(t)}} \boldsymbol{r}$$

with t being the iteration step and $R_k^{(t)}$ the training-vectors assigned to $\boldsymbol{m}_k^{(t)}$.



(a) First iterations step **(b)** Associated neuron-positions

Figure 3.10: A visualization of *K-means clustering* showing the training-vectors associated for each mean and their respective update-position **(a)**. After convergence has been reached, the means are used for positioning the neurons **(b)**.

Algorithm 2 K-means clustering

for $k = 0$ **to** K **do**
  initialize $\boldsymbol{m}_k$ with an unique training-vector $\boldsymbol{t}_{\text{rand}}$
  initialize S_k as an empty set
end for
repeat
  for all $\boldsymbol{t}$ **in** available training-vectors **do**
    $i \leftarrow \arg\min_k \|\boldsymbol{m}_k - \boldsymbol{t}\|$
    add $\boldsymbol{t}$ to S_i
  end for
  for $k = 0$ **to** K **do**
    $\boldsymbol{sum} \leftarrow 0$
    for all $\boldsymbol{t}$ **in** S_k **do**
      $\boldsymbol{sum} \leftarrow \boldsymbol{sum} + \boldsymbol{t}$
    end for
    $\boldsymbol{m}_k \leftarrow \frac{\boldsymbol{sum}}{|S_k|}$
    clear S_k
  end for
until S_k does not change for all $k = 0, \ldots, K$

This procedure is illustrated in *figure 3.10*. A more detailed description of the method is given in *algorithm 2*. When the *K-means clustering* technique is used for positioning the neurons, better results than by using e. g. random positioning are usually achieved. However, the initialization of the means influences the quality of the final result. For example, the cyan-colored neuron in *figure 3.10 (b)* is directly placed on an outlier, since the associated mean never changed its position during the clustering process.

3.4.3 Regional and Online Learnable Fields

Some artificial neural networks with unsupervised learning strategies also can be used for clustering tasks. Such a specialized ANN architecture has been proposed in 2005 by *Schatten, Goerke and Eckmiller* [SGE05]. They present an ANN that allows for clustering a set of given input-data that may be feeded continously to the network, thus making it an online adaptable clustering technique. Despite its high computational efficiency and its benefits in combination with RBF-networks [Kri07], this technique has not drawn much attention yet.

ROLF ►

The so called *Regional and Online Learnable Fields* (ROLFs)[1] allow to combine the adaption rules of *self-organizing maps* with the *K-Means-* and the *ε-Means clustering* techniques. Unlike the latter clustering methods, ROLFs are very efficient in terms of memory usage since they do not need to store the training-data directly. Instead, a set of representatives is used that needs less storage space. In contrast to the K-Means clustering technique discussed in the section before, ROLFs are able to detect "clusters within clusters", i. e. contiguous areas that are enclosed by other clusters.

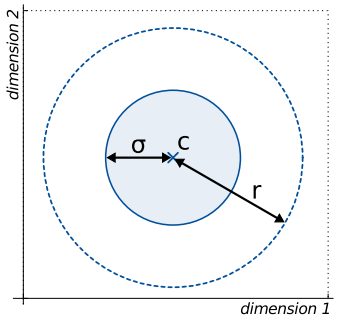


Figure 3.11: A ROLF-neuron is defined through its center c, a width σ and its perceptive area implicitly given by r.

ROLF

Definition 9. *A* Regional and Online Learnable Field *(ROLF) is a set of specialized neurons that are created and modified adaptively to cover the given training-data as best possible.*

1 Fun fact: The authors complete names are *Rolf* Schatten, Nils *Rolf* Goerke and *Rolf* Eckmiller.

The neurons within a ROLF differ from the “classical” neurons as described in *section 3.1.1* since they do not implement a conventional input-/output-scheme. For this reason they have to be treated separately:

ROLF-neuron

Definition 10. *A* ROLF-neuron *is an element of a ROLF with a certain index k. It can be identified as a* 3*-tuple* $(\boldsymbol{c}_k, \sigma_k, r_k)$ *that describes the* center $\boldsymbol{c}_k \in \mathbb{R}^n$ *of the neuron within an n-dimensional input-space, the* width $\sigma_k \in \mathbb{R}_{\geq 0}$ *and its* perceptive area $r_k \in \mathbb{R}_{\geq 0}$*. The radius* r_k *of the perceptive area is implicitly given by* $r_k = \rho \cdot \sigma_k$ *with* ρ *being a constant scalar factor within the whole ROLF.*

Learning procedure

The perceptive area is confined to a hypersphere with radius r_k around the center of the neuron. This area plays a central role for the learning algorithm used for this kind of neural network. For the learning procedure of a ROLF, two different levels have to be distinguished [SGE05]:

Online adaption of ROLF-neurons

For a given training-vector $\boldsymbol{p} \in \mathbb{R}^n$, each ROLF-neuron decides whether it accepts the given stimulus or not. A neuron k accepts $\boldsymbol{p}$ if and only if the training-vector lies within the perceptive area, i. e. $\|\boldsymbol{c}_k - \boldsymbol{p}\| \leq r_k$ holds.

In case the neuron k accepts $\boldsymbol{p}$, the center and the width of the neuron is adapted simultaneously by applying the following learning rule (with t being the current time index):

$$\boldsymbol{c}_k^{(t+1)} = \boldsymbol{c}_k^{(t)} + \eta_c(\boldsymbol{p} - \boldsymbol{c}_k^{(t)})$$
$$\sigma_k^{(t+1)} = \sigma_k^{(t)} + \eta_\sigma(\|\boldsymbol{c}_k^{(t)} - \boldsymbol{p}\| - \sigma_k^{(t)})$$

Here, η_c and η_σ denote the learning rates for adapting the neuron centres and their widths respectively. These learning rules cause the accepting neuron to move towards the position of the training-vector while its width adapts towards the mean distance between the accepted training-vectors and the neuron center [SGE05].

Learning procedure of the whole network

The network itself controls whether a new neuron is needed in order to cover the given training-vector or an already existing neuron can be adapted. For the latter case, a so called *winner-takes-all scheme* is applied: Under all accepting neurons, only the one whose center $\boldsymbol{c}_k$ is closest to the training-vector is allowed to adapt its position and width.

If no accepting neuron does exist, a new neuron is created. The center of this neuron is initialized with the position of the training-vector whereas the

width σ of the newly created neuron is initialized using one of the following strategies:

Init-σ The width σ is initialized with a constant initial value σ_{init}.

Minimum-/Maximum-σ σ is initialized using the minimum/maximum value across all neurons.

Mean-σ The average width across all neurons is used for initializing σ.

The choice of the initialization strategy primarily affects the area covered by the neurons [Kri07]. The original authors propose using *mean-σ* as the favourite strategy.

Once the learning phase has been completed and no additional training-vectors are about to be added to the network, a second phase is started that builds up neighborhood relations between the different ROLF-neurons. However, in the context of RBF-networks this phase can be omitted since the raw neuron positions are of particular interest. Therefore, this phase will not be discussed in detail within this work.

Learning parameters

In order to operate properly, four parameters have to be determined before using the ROLF. These include the global constant ρ, the two learning rates η_c and η_σ and the initial width of the first placed ROLF-neuron. Unfortunately, no generic strategy exists for choosing these values. As *Kriesel* points out, learning rates between 0.005 and 0.1 and values in the interval from 2 to 3 for the constant ρ tend to yield good results. The initial width for the neurons generally depends on the data distribution and therefore has to be determined manually [Kri07].

Combination with RBF-networks

After appropriate learning parameters have been chosen and the learning phase has been finished, the positions of the ROLF-neurons can be used for initializing the centres of the radial-basis functions within the RBF-network. Depending on the training-data and the chosen learning parameters, the ROLF may contain more neurons than scheduled for the RBF-network. In this case, only the largest ROLF-neurons (regarding σ) are selected, since they cover more training-vectors and are less likely to represent outliers than smaller neurons.

Future prospects

In contrast to the previously discussed neuron-placement methods, ROLFs do not rely on a good (random) initialization and always produce neurons that cover relevant information within the input-space. In future applications, the width of the ROLF-neurons could be incorporated to the RBF-neuron initialization. One has to examine if this additional information about the training-data has the potential to improve the quality of the approximation function.

Chapter 4

Implementation

In this chapter I will describe the practical implementation of the algorithms as discussed in the previous sections as well as the gaze tracking hardware to evaluate them. Regarding the development of the tracking hardware, two prototypes have been created, each with different assets and drawbacks. In order to allow gaze tracking across a broader audience, special attention has been given regarding the costs of the devices and the availability of the individual parts. The software framework presented in this section allows an easy integration into follow-up projects and comes with a set of user interfaces to gain calibration- and training-data.

This chapter starts with a discussion of the technical considerations when designing a head-mounted gaze tracker. Afterwards, the two prototypes that have been developed are presented. Thereafter, I give a detailed review about the software framework consisting of the tracking library, a set of testing programs and end-user interfaces.

4.1 Hardware design

As many different techniques for gaze tracking exist, as many hardware devices have been designed. In this section, I will give a short overview of an existing head-mounted tracking device presented in related research as well as the devices created in context of this thesis.

Current research mainly focuses on *remote gaze trackers*, i. e. devices that do not need physical contact with the user. In the last years only few head-mounted gaze trackers have been presented. This is due to the different demands on the respective designs. Remote tracking techniques usually allow for a higher comfort during the tracking process while head-mounted devices are not bound directly to the workspace in front of the computer screen. The following juxtaposition compares the different designs directly:

Remote gaze trackers

- ⊕ High user comfort
- ⊕ Almost instant setup
- ⊖ Workspace confined fixed area
- ⊖ Limited accuracy
- ⊖ Complex hardware

Head-mounted devices

- ⊖ Minimally invasive
- ⊖ Needs user-intervention
- ⊕ Non-restricted workspace
- ⊕ Constantly good accuracy
- ⊕ Easy to build

It is quite obvious that remote devices outplay head-mounted gaze trackers in terms of usability. Once the remote hardware is set up and connected to the computer, no additional user actions are necessary. On the other hand, the user has to put on and arrange the head-mounted device each time it is about to be used.

Restrictions to work-space

However, the higher comfort also induces a major drawback for remote setups. Since the position of the device is fixed – usually near the computer monitor, the user's eyes cannot be detected if the user turns away from it. Additionally, as the distance between user and tracking device increases, the maximum attainable accuracy decreases – as long as no complex zoom-lens camera system is used. These drawbacks do not apply to head-mounted gaze trackers. Since the camera is fixed to the user's head, the quality of the eye-images remain constant under head movement. Also, their operational range is not limited to the area in front of a computer screen. By utilizing an external tracking system that monitors the absolute position and orientation of the user's head, head-mounted devices can be used within a large-scale VR-environment.

Construction and financial aspects

Another advantage over remote tracking devices is the complexity of the hardware itself. While the latter usually need a complex camera setup that often involves multiple cameras mounted on a pan-tilt unit, head-mounted devices only need a single camera and a retainer for fixating it to the head. Of course, the relatively low technical requirements also result in a lower pricing of such devices.

Related projects

Precedent to this work, various other approaches have been done for developing low-cost tracking systems. Most notable is the *openEyes* project initiated by *Derrick Parkhust*. This project describes a head-mounted camera based tracking system consisting of two cameras, one for monitoring the user's eye and another one for recording the whole scene around him. The images are analyzed by an open-source software that utilizes the so called *Starburst*-algorithm for determining the point of gaze [LWP05].

Costs and construction

Their whole system consist of off-the-shelf components that are modified and assembled by hand. This procedure includes desoldering the sensor module of an IEEE-1394 webcam, replacing the infrared-filter of the camera with a *87c Wratten filter* that blocks visible light and adding a 12*mm* zoom lens to the system. As they point out, this procedure is very error-prone and may result in damaging the sensor [WLBP05]. As they claim, the whole system induces a total cost of about 350 US dollars. The average accuracy of the system is stated as approximately *one degree* of visual angle.

Accuracy

Towards a low-cost head-mounted tracking hardware

As *Morimoto and Mimica* point out, a theoretical *ideal gaze tracker* should satisfy the following seven criteria [MM05]:

1. *Accuracy:* The optimal gaze tracker should be precise to minutes of arc.
2. *Reliability:* Tracking results have to be constant and have to exhibit a repetitive behaviour.
3. *Robustness:* The device should work under different conditions, regardless whether the user needs visual aids or the device is used indoors/outdoors.
4. *Non-intrusiveness:* No harm or discomfort must be done to the user.
5. *Head position invariance:* The gaze tracker should allow for free head motion.

6. *Instant setup:* No calibration procedure should be needed.
7. *Interactivity:* The device should have real-time response.

Of course, such an idealized tracking device does not exists, yet. The goal of this thesis is to create an affordable tracking device that matches these criteria as best possible. Regarding the construction of a hardware prototype, especially the points *Robustness*, *Non-intrusiveness* and *Head position invariance* have to be kept in mind while pondering on a sufficient design.

Section outline

The following sections contain detailed descriptions of the hardware devices that were build in context of this thesis. In addition to the assembly instructions, each section contains a short cost overview besides the assets and drawbacks of the prototypes.

4.1.1 First prototype

The first prototype I created in the context of this thesis is heavily inspired by the work of *Parkhust et. al.* [BP04, WLBP05]. This device has to be understood as an early *proof of concept* since it is inferior to the second prototype in terms of accuracy and usability. For the sake of completeness, the construction of the device is described in short. However, it has not been used for the final evaluation of the tracking algorithms.

Device structure

Similar to the related work by *Parkhust et. al.*, this tracking device consists of a single camera mounted on a spectacle frame made from plastic. The camera casing has been replaced with a smaller one made from acrylic glass that can be directly fixated at the frame using a stiff wire. The camera used for the first prototype is a mid-class desktop webcam that has an average price of about 25 € (see *table 4.2* for more information).

Camera modifications

This camera offers a relatively high resolution of 1280×720 pixels. However, in its initial configuration the camera is insensitive to infrared light and the focus point is too far away for obtaining a sharp image of the user's eye. Therefore, a series of modifications have been done to the camera. At first, the infrared-blocking filter of the camera has been replaced with a filter that blocks visible light. This is done by

Figure 4.1: The readily assembled first prototype.

unscrewing the lens-case and removing the small glass filter that is placed behind the lens. While doing so, special attention has to be drawn to the sensor. If dust or lint reaches its surface, the image quality may get affected negatively. In order to keep the costs as low as possible, a very inexpensive filter has been chosen. This filter consists of a small patch of an unexposed processed diapositive. Interestingly enough, this filter offers an acceptable performance at virtually no costs.[1] After the original filter has been replaced, the lens-case is reassembled. During this step, a thin layer of cardboard has been placed between the camera board and the lens-case. This results in a shift of the focus plane thus allowing for gaining a sharp image of the user's eye.

Infrared filter

Focus plane adjustment

The infrared LED for illuminating the eye region and for generating the glint is placed directly besides the camera module. The technical specifications of this LED can be found in *table 4.1*. It is powered directly by the same USB cable the camera uses. In order to adjust the voltage of the LED, a 60.40 Ω series resistor is used. Additionally, a 5 $k\Omega$ potentiometer allows for adjusting the intensity of the infrared light.

Infrared illumination

Performance of the first prototype

The first tracking device has several assets and drawbacks. The main advantage of this device is its low price and the robustness while moving the head. Since a relatively stiff wire is used to join camera and spectacle frame, the whole system is quite robust to smaller vibrations induced by the user. All in all, the total costs can be summed up as follows:

Item	Quantity	approx. Price
Spectacle frame (plastic)	1	3 €
C270 USB HD Webcam	1	25 €
Infrared LED (HE1-240AC)	1	0.25 €
Resistor (60.40 Ω)	1	0.11 €
Potentiometer (5 $k\Omega$)	1	0.65 €
Cables, wiring and casing		< 5 €
total costs		≈ **34** €

1 Especially if compared with the *87c Wratten filter* used by *Parkhust et. al.*

However, the downsides of this device outweigh the low costs: Since the whole camera board is fixated only at one side of the frame, the weight of the whole device is very unbalanced. This leads to a high level of discomfort over time. Additionally, the size of the camera limits the user's field of view. The camera used for this device also only offers a fixed focus lens, leading to slightly out-of-focus images for some users depending on the head-geometry.

Summarising all these aspects, the first prototype is not comfortable enough for everyday use. However, it shows that gaze tracking is principally possible even if low-cost hardware is used.

Assets

- ⊕ Very low costs
- ⊕ Few external parts needed
- ⊕ Relatively insensitive to vibrations

Drawbacks

- ⊖ Unbalanced weight
- ⊖ Restricted field of view
- ⊖ Interfering with user's glasses
- ⊖ Uneven image quality across different persons

4.1.2 Second prototype

In order to address the issues of the first prototype, a second one has been built. The focus on the second device clearly lies on the usability and comfort of the user. Instead of a spectacle frame – which is the common construction for head-mounted gaze trackers, a modified headset has been used. This approach has two salient advantages. At first, the weight of the whole devices now bears on the user's head instead of his nose. This allows for a higher comfort when using the tracking device on a regular basis. Secondly, the position of the camera can be adjusted freely. If no gaze tracking is needed, the camera can be hinged up easily.

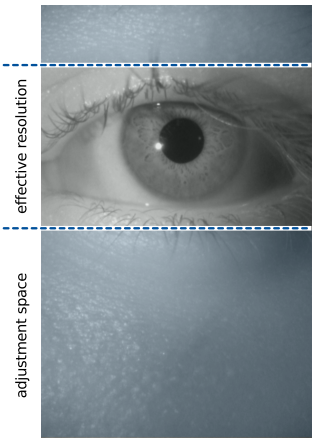


Figure 4.2: Since the camera is rotated, the effective resolution of the image containing the user's eye is reduced.

Camera properties and modifications

The camera used in the second prototype also has changed. This time, a very small webcam for mobile use has been installed (see *table 4.3* for more details). The footprint of the whole camera board – including the lens – is only about $25\,mm \times 25\,mm$ and therefore the camera distracts the field of view less than the one used in the first prototype. Additionally, the camera is equipped with an autofocus lens that allows to focus on the user's eye without the need for further modifications. Analogous to the camera of the first prototype, the infrared blocking filter has to be substituted with one that blocks visible light. When unmounting the lens-case of the camera, one has to exercise caution while desoldering the focusing device since it is surface mounted and may break easily.

Caution advice

Now, the camera can be attached to the headset. This is done by cutting of the plastic-case where the microphone of the headset is located and glueing the

Assembly of parts

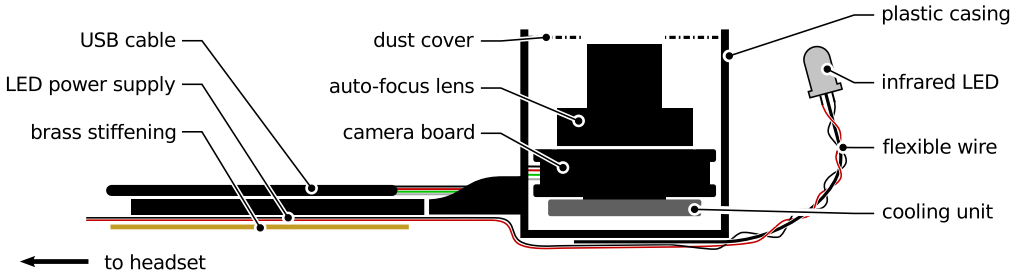


Figure 4.3: The layout of the second prototype showing the camera housing and parts of the microphone boom.

Parameter	Value	Unit
Vendor	Harvatek	
Article number	HE1-240AC	
Packaging diameter	5	mm
Pricing example	0.25	€
Average forward current	100	mA
Typical forward voltage	1.6	V
Wavelength	850	nm
Radiant on-axis intensity	40	$\frac{mW}{sr}$
Viewing angle	40	deg

Table 4.1: Technical specifications of the infrared LED used for eye-illumination.

Parameter	Value	Unit
Vendor	Logitech	
Article number	C270 USB HD Webcam	
Vendor price	29.99	€
Resolution	1280×720	px
Framerate	30	$\frac{\text{frames}}{s}$
Focus method	fixed focus	

Table 4.2: Technical specifications of the webcam used in the first prototype.

Parameter	Value	Unit
Vendor	Microsoft	
Article number	Lifecam HD-6000	
Vendor price	59.99	€
Resolution	1280×800	px
Framerate	30	$\frac{\text{frames}}{s}$
Focus method	auto/manual focus	

Table 4.3: Technical specifications of the webcam used in the second prototype.

cut surface to a small box of plastic that holds the camera module. The camera is mounted in a way that rotates the sensor by 90°. This reduces the effective resolution of the camera as seen in *figure 4.2* but also allows the user to adjust the vertical position of the camera. A layout of these parts can be seen in *figure 4.3*. Then a dust-cover made from cardboard is placed on the box that protects the camera and also blocks the blue operating light emitted from the camera board.[1]

The infrared LED – that is the same type as used in the first device – is attached to a flexible wire that allows to adjust the cone of light for each user individually. The electric cables of the LED as well as the USB-cable is then fastened to the microphone boom that leads to the left ear. In order to make the structure less sensitive to vibrations, a brass section is attached to the microphone boom that increases the stiffness of the whole device. Inside a second box of plastic, the series resistor as well as the potentiometer for adjusting the infrared LED is placed. Since this casing is very small ($18\,mm \times 15\,mm \times 11\,mm$) it is directly attached to the outside of the left ear cup. An image of the readily assembled tracking device can be seen in *figure 4.4*.

Figure 4.4: The readily assembled second prototype.

Total costs

Due to the high quality camera and the headset, the costs of the second prototype are relatively high as seen in the table below. However, the market prices for these components are usually much lower. The actual price of the prototype (build in 2011) amounts to approximately 78 €.

1 Of course, the LED that is causing the light can also be desoldered. However, this induces the risk of damaging the camera.

Item	Quantity	approx. Price
Headset (Creative HS-450)	1	29.99 €
Webcam (Microsoft Lifecam HD-6000)	1	59.99 €
Infrared LED (HE1-240AC)	1	0.25 €
Resistor (60,40 Ω)	1	0.11 €
Potentiometer (5 $k\Omega$)	1	0.65 €
Brass section	500 mm	2.45 €
USB extension cable	1.8 m	4.15 €
Cables, wiring and casing		< 5 €
total costs		≈ **103** €

Performance of the second prototype

As stated at the beginning of this section, the main focus of the second prototype is a high comfort and usability. During the evaluation of the device and the tracking algorithms as described in *chapter 5*, a total of seven persons were asked about their impressions while wearing the device. The probands were told to give out grades ranging from 1 (worst) to 5 (best) for the following criteria:

- Wearing comfort
- Weight of the tracking device
- Restriction of field of view

While the first two criteria have been rated very high by all probands (average rating: 4.6/4.7), the restriction of the field of view has been perceived as distracting by most users (average rating: 2.9).

However, the latter point also induces some implications when doing head position invariant tracking. The size of the camera-box and the width of the microphone boom covers one of the screen markers most of the time. This results in some issues for testing the tracking method by *Yoo and Chung* as seen in *section 3.3*.

Image quality

Technically, the second prototype offers a much higher image quality than the first one. Since the focus lens can be adjusted manually by the controlling software, the camera can be focused *exactly* on the user's pupil. Additionally, the camera offers a better signal-to-noise ratio in low lighting situations. Certainly such situations can be avoided, since the infrared LED can be freely adjusted to illuminate the eye region as uniformly as possible. However, if the user performs very fast head movements, the stiffness of the device is still not high enough to prevent vibrations.

These vibrations can influence the image quality negatively. This aspect should be taken into account for future devices.

Assets

- ⊕ Excellent image quality
- ⊕ High wearing comfort, adjustable camera-arm
- ⊕ No interferences with user's glasses

Drawbacks

- ⊖ Stiffness not high enough under certain situations
- ⊖ Field of view still restricted
- ⊖ Higher costs compared to first prototype

4.1.3 Screen markers

In order to achieve a certain degree of head-position invariance, four corneal reflections are analyzed by the tracking-algorithm (see *section 3.2* for details). These reflections originate from four markers that emit infrared light and are placed at the edges of the computer monitor. Each of them is build from two slices of acrylic glass that have a dimension of $50\,mm \times 10\,mm \times 3\,mm$. Between both slices, a high power infrared LED is embedded that acts as a light source. The relevant technical specifications of this diode are stated in *table 4.4* and are taken from the vendors technical datasheet [Pac]. The sides and back of this package are coated with aluminium foil that acts as a reflector for the infrared light. To allow for an even light distribution, the surface of the lower acrylic glass slice is roughened. In addition, the upper slice contains a series of cavities that deflect the light (see *figure 4.6* for more details). All parts are held together using a transparent glue and two pieces of shrink sleeving at both endings. Lastly, a piece of Velcro fastener is fixated at each marker that allows to attach and detach the markers from the monitor easily.

Assembly instructions

All readily assembled markers are powered by a standard $8\,V/500\,mW$ power supply as used for e. g. charging mobile phones. The LEDs are connected in series with the power source using a $10\,\Omega$ series resistor. To allow for an easy adjustment of the light intensity, a $5\,k\Omega$ potentiometer is added to the circuit.

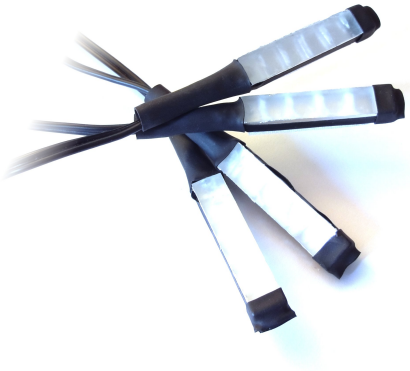


Figure 4.5: The final assembled screen markers.

Power supply

The total costs for the screen markers are summarized below:

Costs

Item	Quantity	approx. Price
Power supply	1	5 €
Infrared LED (HDSL-4230)	4	1.84 €
Resistor (10 Ω)	1	0.11 €
Potentiometer (5 $k\Omega$)	1	0.65 €
Cables	4 m	1.96 €
Acrylic glass 3 mm (10 $cm \times 5\,cm$)	1	1.01€
Velcro fastener, shrink sleeving, etc.		< 5 €
total costs		≈ **16** €

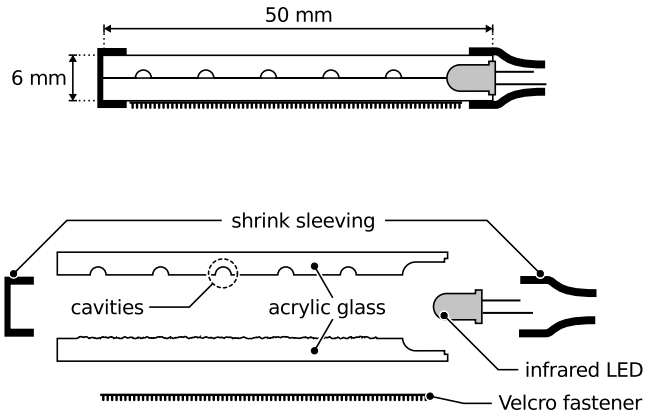


Figure 4.6: Construction of the infrared screen markers. The approximate dimensions are stated in the upper image. The lower one shows a exploded assembly drawing of a marker.

Parameter	Value	Unit
Vendor	Hewlett Packard	
Article number	HDSL-4230	
Packaging diameter	5	mm
Pricing example	1.84	€
Average forward current	100	mA
Typical forward voltage	1.67	V
Wavelength	875	nm
Radiant optical power	32	mW
Radiant on-axis intensity	150	$\frac{mW}{sr}$
Viewing angle	17	deg

Table 4.4: Technical specifications of the marker infrared LEDs.

4.2 Software implementation

In the foregoing chapters, several algorithms were described that are able to process the images delivered from the tracking hardware. These algorithms have been implemented in order to deliver a complete gaze tracking system for interactive use. In addition to the tracking-library itself, a couple of programs that implement this library have been created that allow for testing the performance of the algorithms and also can be used for a realtime-tracking demonstration.

4.2.1 General design choices

Before starting to implement the tracking- and mapping-algorithms, some fundamental design choices have to be discussed. Since the tracking system may be used in various follow-up projects, several demands can be stated:

1. The code should be reusable and easy to maintain
2. The tracking procedure should be robust
3. Latencies should be as small as possible
4. All programs should be platform independent

Choice of programming language

With these requirements in mind, I made the choice of using *C++* as programming language in conjunction with the *OpenCV* library. Since high performance and low latencies are critical factors, *C++* seems to be a good compromise as it supports an object-oriented paradigm while being machine-oriented enough to match these factors.

External libraries

These performance requirements are also concurred by the *OpenCV* (Open Source Computer Vision) library. This library is known to be stable and feature-rich. It allows for acquiring images from camera devices and video-files as well as processing the images in various ways. Using this library, most of the image filters described in *chapter 2* do not have to be implemented explicitly. Additionally, *OpenCV* supports the *CUDA* architecture that allows to speed up some algorithms by utilizing the computers graphics hardware at a later time.

In addition to *OpenCV*, the following libraries are utilized as well:

cvBlobsLib The *cvBlobsLib* is an extension to the OpenCV library that allows to find connected components within a binarized image. It uses the connected-component-labeling technique as described by *Chang et al.* [CCL04].

CGAL The *Computational Geometry Algorithms Library* is an open-source project that provides various structures and algorithms for computational geometry. In context of this thesis, it is used to obtain a Delaunay triangulation for doing linear interpolation as described in *section 2.4.1*.

Eigen 3 Numerical calculations – like singular value decomposition – are done using the *Eigen 3* library. This library is also distributed under an open-source license.

Qt 4.7 For some of the end-user interfaces (e. g. for the calibration procedure) a graphical user interface is needed. *Qt* is an open-source, cross-platform library for creating graphical user interfaces and can also be used for visualization of 2D-graphics.

Reusablity and platform independence

In order to allow for an easy integration of the tracking algorithms into future projects, the gaze tracker itself is organized as a *shared library*. At the moment, this library has only been implemented for Linux platforms. However, adapting the library to other platforms (like *Microsoft Windows* or *Apple MacOS*) should not be too complicated, since all used external libraries support these platforms and no platform-specific calls have been used.

4.2.2 The tracking library

The tracking library consists of two closely related subparts. The *high-level API* contains the concrete implementations of the algorithm-chain as seen in *section 2.2* and the related extensions for head position invariant tracking. If more control over the underlying algorithms is needed or if the existing ones should be extended, one can use the *low-level API* for doing so. This part of the library also contains template-based classes for defining arbitrary RBF-networks for interpolation and approximation tasks.

High-level API

The *high-level API* allows the creation of gaze tracking enabled applications that only need very few additional lines of code. The high-level API contains classes for gathering image-data, representing tracking- and mapping-results and the respective algorithms for obtaining these results.

The following listing shows a minimal example for using the high-level API. The individual parts are discussed afterwards:

```
1  #include <WeGA/wega.h>
2  #include <WeGA/camerareader.h>
3  using namespace WeGA;
4
5  int main(int argc, char *argv[]) {
6     CameraReader cam(0); // 0 denotes the systems default camera
7     TrackingTunables tunables;
8     tunables.pupilSize = 0.06; tunables.glintSize = 0.02;
9     tunables.eyeTemplate = cv::imread("template.png", 0);
10    InvariantGazeTracker tracker(
11         WeGA::BinaryLeastSquares, // Method for pupil-detection
12         WeGA::Centroid,           // Method for glint-detection
13         tunables);
14     while(cam.hasNext())
15     {
16         cv::Mat inputImage = cam.nextFrame();
17         TrackingResult result = tracker.analyzeFrame(inputImage);
18         std::cout << result.toString() << std::endl;
19     }
20     return 0;
21 }
```

Tracking procedure

This example application first opens the system's default camera device as seen in *line 6*. The class `CameraReader` also can be replaced by an instance of `VideoReader` that allows opening video files for offline gaze tracking. The respective class diagramms can be seen in *figure 4.7*. At next (*lines 7 to 13*), the tracking algorithm needs to be initialized correctly. This is done by setting up an instance of `TrackingTunables` that contains various directives for the tracking procedure. In this example, the variables `pupilSize` and `glintSize` are specified that control the thresholding procedure as described in *section 2.3.2*. Additionally, a template image of the eye-region is loaded using built-in functionality of *openCV*. The tracking algorithm itself is intialized by passing the `TrackingTunables` next to the specifications of the pupil and glint detection methods:

`WeGA::BoundingBox` Only the outer bounds of the respective features are determined and the midpoint of this box is used. This method is by far the fastest method, but does not allow for subpixel accuracy.

`WeGA::Centroid` The centroid of the pixels inside the bounding-box is used as described in *section 2.3.4*. This leads to subpixel accurate results though needing more processing time.

`WeGA::LeastSquares`, `WeGA::BinaryLeastSquares`, `WeGA::ComplexLeastSquares` One of the ellipse-fitting based approaches is used as shown in *section 2.3.4 ff.*.

These normally allow for the highest accuracy. However, these algorithms are more complex than the former methods, leading to higher computing costs.

The remaining lines of the example program should be almost self-explanatory: As long as the camera delivers images, they are obtained using the `nextFrame()` method and then passed to the tracking algorithm. The result of the algorithm is stored in a container class that is specified in *figure 4.8*. The class diagram of the tracking algorithms is shown in *figure 4.9*.

Mapping procedure

The last action of the example program dumps the tracking result to the console. However, more complex programs would pass this result to an instance of e. g. `InvariantGazeMapper` to obtain the screen-coordinates the user is looking at. All mapping algorithms realize the interface `MappingAlgorithm` as seen in *figure 4.10*. Before the tracking results can be passed to the mapping algorithm, a set of training-vectors has to be specified. This is done by calling `addTrainingResult` with a `std::vector` that contains pairs of tracking results whose corresponding screen-coordinates are already known. After the training-vectors have been added, the `mapToScreen` method returns an instance of `MappingResult` as seen in *figure 4.8*. Assuming the training-vectors already have been specified, the source code may be extended as follows:

```
// [...]
InvariantGazeMapper mapper(8, // Number of neurons used in the RBF-network
                           WeGA::KMeans); // Neuron placement strategy
// [...] initialization of training-vectors [...]
while(cam.hasNext())
{
    cv::Mat inputImage = cam.nextFrame();
    TrackingResult result = tracker.analyzeFrame(inputImage);
    MappingResult position = mapper.mapToScreen(result);
    std::cout << "Currently looking at: (" << position.toString()
              << ")" << std::endl;
}
```

The second argument in the constructor of `InvariantGazeMapper` specifies the strategy for placing the neurons as discussed in *section 3.4.2* and may be set to `WeGA::Random`, `WeGA::KMeans` or `WeGA::ROLF`. In case the results are mapped with an instance of `FixedGazeMapper`, the interpolation method can be specified in the constructor.

Low-level API

For most applications, the high-level API should be sufficient. However, if more control is needed or the existing algorithms need to be extended, the *low-level API* provides an interface for doing this. Since describing all details of this API would exceed the scope of this document, the interested reader is referred to the documentation within the source code of the library. The following list demonstrates the capabilities of the low-level API:

Extension of algorithms All mapping and tracking algorithms can be extended with additional functionality by simply inheriting from the respective classes. For example, the class `InvariantGazeTracker` inherits from `FixedGazeTracker` and only implements functionality for detecting the marker reflections. By implementing `TrackingAlgorithm`, completely new algorithm-chains can be established.

Modification of algorithms The sub-algorithms of existing implementations can be modified by passing alternative implementations in the respective constructors. For that reason, a set of interfaces is provided that allows for e. g. exchanging the template matching routine with a custom implementation of the interface `EyeLocator`. This way, the approximation method or the neuron-placement strategy of `InvariantGazeMapper` can be modified by implementing the interface `Approximator` or `Neuronplacer`.

RBF framework Additionally, a general purpose framework for modelling RBF-networks for interpolation and approximation tasks is included.[1] The classes of this framework are completely templatized resulting in very efficient code. The input and output dimensions of the RBF-network can be chosen arbitrarily through template parameters as well as the radial-basis functions and metrics the RBF-neurons use.

4.2.3 Testing framework

In order to allow for an objective comparison of the different tracking and mapping techniques across a number of probands, a comprehensive testing-framework is needed. This framework consists of a series of small command-line applications that are based on the WeGA-library as well as a python-script that coordinates these programs. The following applications are included in this framework:

1 This framework also has been created within the extend of this work.

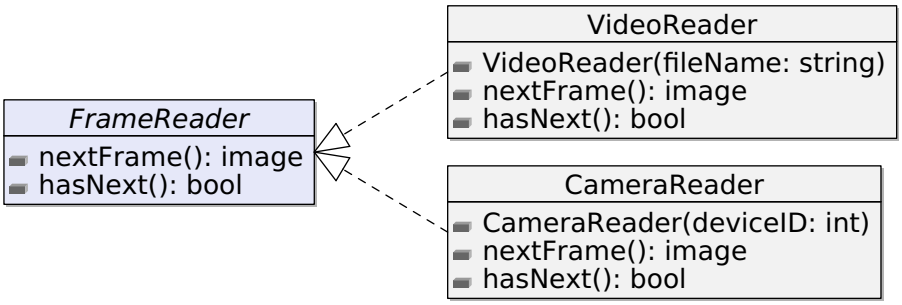


Figure 4.7: Class diagram of input classes.

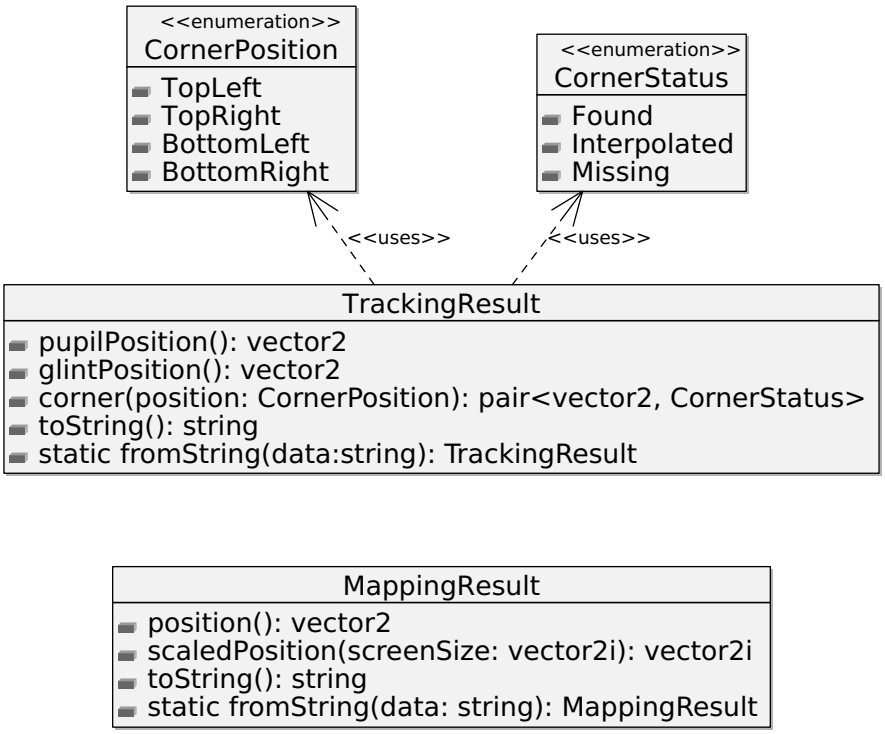


Figure 4.8: Class diagram of container classes.

CalibrationTest This application allows to define two test patterns that are used for calibration of the system (training data) and for testing its performance (evaluation data). The program consists of a graphical user interface for entering the probands data (like name, distance to monitor, etc.) and shows the test patterns on a full-screen window afterwards. During this process, the eye movement of the user is recorded to a video file. Additionally, a file is created that contains information about the screen-coordinates the user should have been looked at for the specific frames of the video file.

AlgorithmTest The algorithm test consists of two separate programs that are similar to the example program as seen before. The programs implement either the fixed-head case or the head position invariant case of the tracking procedure. By setting various command-line arguments when executing the programs, different video resolutions, algorithm combinations and other parameters can be specified. Given these arguments and the output files of the *CalibrationTest* application, the programs process the video files and return the tracking results for each frame.

MappingTest Lastly, the tracking results are processed by two programs that implement the mapping techniques for the fixed-head and the invariant case. In combination with the data given by the calibration procedure, the error vectors between the estimated gaze-position and the position where the user did look to are returned. Again, the behaviour of the algorithms can be specified by using command-line arguments.

Testing script

While acquiring the video data can be done separately, the analysis of the videos is coordinated by an external script written in *Python*. This script allows to specify complex test-scenarios. For example, one can specify a test that compares the performance of two different algorithms for fitting the pupil ellipse over a series of different video resolutions – as it has been done in *chapter 5*. The script automatically chooses the correct command-line arguments for the *AlgorithmTest* and *MappingTest* programs and distributes the program instances across the available processor cores of the current system. This allows to process complex test scenarios within a reasonable expenditure of time.

Evaluation of test-data

The results of these tests are analyzed automatically. At the end, the script returns a series of information about each test, containing e. g. the minimum, maximum and average error of the tracking and mapping procedure. Additionally, various graphical plots of these values are exported to PDF files using the *Sage* mathematics software system.[1]

1 See `http://www.sagemath.org/` for more information.

4.2.4 End-user interfaces

Besides the testing applications, some additional programs have been created for e. g. demonstration purposes.

CameraCalibration This program uses *v4l2* (Video4Linux version 2) for controlling the focus mechanics of the camera used in the second hardware prototype. The application automatically focuses on the user's pupil region by choosing the focus setting that generates the highest contrast around the pupil area. This is done by applying a high-pass filter to this specific region of the image and counting the number of pixels with a certain intensity. The focus setting that implies the highest pixel count is chosen.

This application does only work on Linux platforms.

CalibrationDaemon Since the training procedure for head position invariant tracking tends to be quite extensive, a more user-friendly procedure has been searched for. Similar to an approach described by *Sugano et al.* [SMSK08], this program runs as a background daemon on the user's system. Each time the user performs a mouse-click, the actual camera image and the position of the mouse-cursor is captured. Assuming the user fixates on the cursor – or at least next to it – when performing a click action, this program generates a continuously growing database of training-vectors that can be used for the mapping procedure. Technically, this applications uses the *XRecord* extension of the *X-server* and is therefore restricted to Linux/Unix platforms.

This application does only work on Linux platforms.

LiveDemonstration This application acts as a short demonstration of the fixed-head tracking algorithms. At first it shows 9 calibration points arranged in a 3×3 pattern. Afterwards, a picture is shown and a red spot follows the user's fixation point interactively.

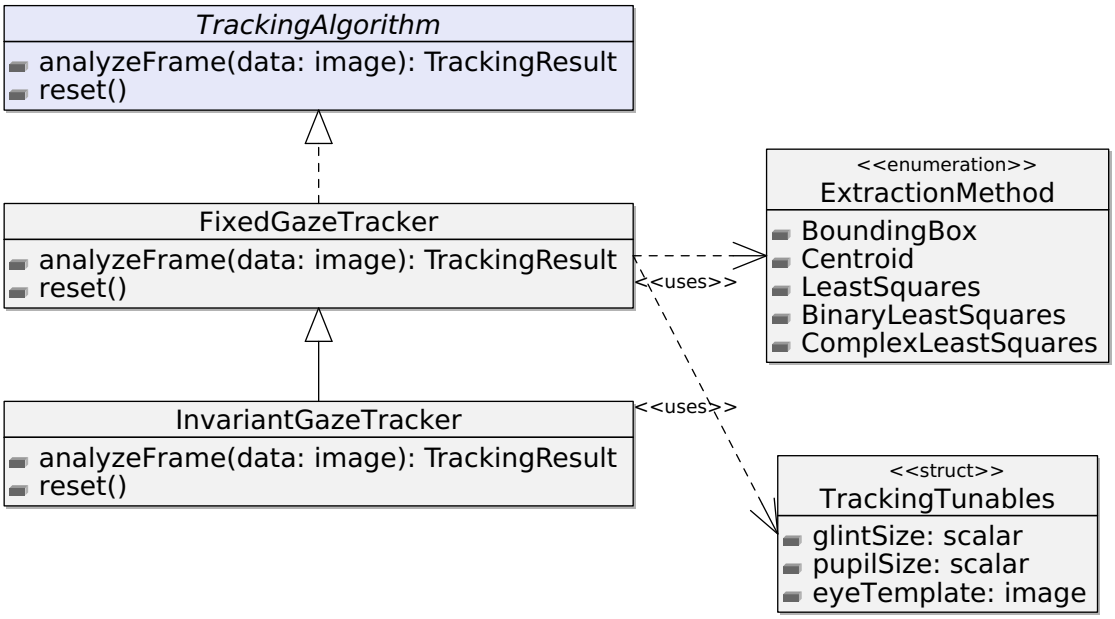


Figure 4.9: Class diagram of the tracking algorithms.

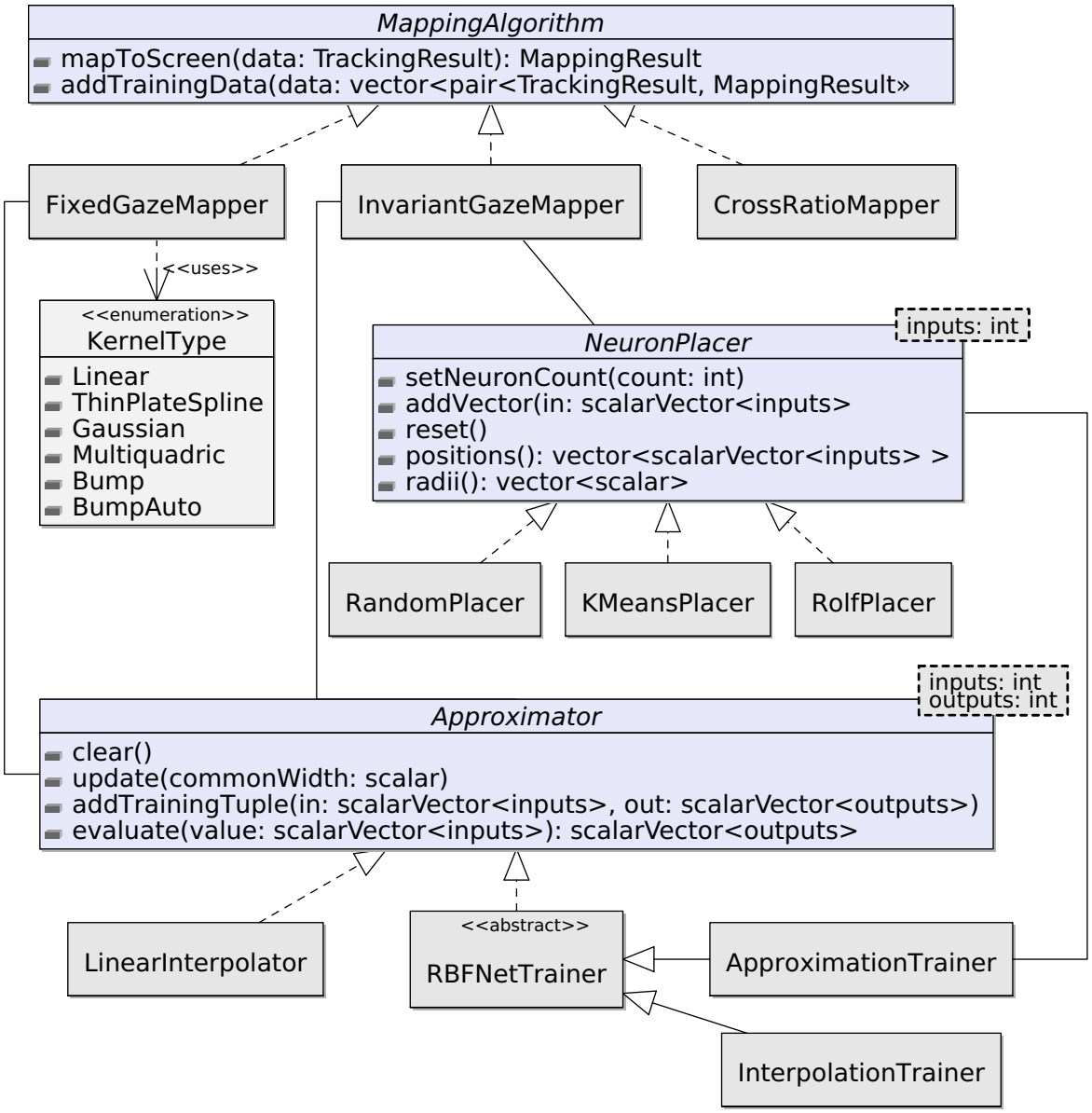


Figure 4.10: Class diagram of the mapping algorithms.

Chapter 5

System evaluation

In the previous chapters various algorithms and theoretical considerations have been envisaged that have to be validated by empirical results. This chapter deals with the evaluation of the tracking hardware as well as the different algorithm configurations. The tests for this evaluation were performed on different persons. Some of these probands wore glasses or contact lenses, posing a challenging task for the tracking system. Several tests were performed on the tracking data for determining the average accuracy, the runtime performance and other statistics for the different tracking and mapping methods.

For the case of head position invariance some separate tests have been done. Here, the optimal parameters for the RBF-network are of fundamental importance. In this context, the question about how to obtain sufficient training-data arises and needs to be discussed.

Chapter outline

The chapter is organized as follows: At first, the choice of probands – who tested the tracking device – is presented. Then, the methodology on how the test-data is acquired has to be discussed, including the setup of the test-environment. Lastly, each of the test results is presented and discussed in detail.

5.1 Choice of probands

Since the tracking algorithms initially have evolved using a test dataset made by myself, it has to be assured that these algorithms perform equally well on other persons. For that reason, six additional persons were asked to test the device under the same test conditions. Each person was given a short questionnaire asking for the following information:

- Age of the person
- Need for vision aids
- Questions about the tracking hardware (as discussed in *section 4.1.2*)
- Perception of the own concentration during the test

The form was given to the probands right after they finished the test. During the testing procedure only artificial light was used to illuminate the test environment. This was done in order to reduce unintentional specular reflections since the amount of infrared light emitted by fluorescent lamps is relatively small compared to direct sunlight.

Evaluation of the questionnaire

The age of the seven male probands ranged from 23 to 34 years with an average age of 27 years. During the test, two of the persons were wearing glasses and a single proband needed contact lenses. The average concentration rating – ranging from 1 (lowest) to 5 (highest) – was about 3.6. This indicates that staring at the screen without being allowed to move the head quickly becomes challenging.

Since the complete testing procedure for the invariant case is very time consuming, the probands only tested the fixed-head algorithms as described in *chapter 2*. The tests for the head position invariant case were performed by only one person. However, since the image processing algorithms of both cases are almost the same, the results obtained for the head position invariant tests should be transferable to other persons as well.

5.2 Methodology and testing setup

In order to attain objective and reproducible results, some effort was spent on defining a sufficient testing procedure. In related work many different strategies have been suggested. In the majority of cases, the probands are told to undergo

a calibration procedure and then evaluate the system by fixating on a series of test-points on the computer screen.

Related work

For example, *Ramanauskas et al.* use a grid consisting of 25 dots that are evenly distributed across the computer screen [RDD08]. These dots are shown one at a time in random order and are visible for two seconds. During this time period, the radius of the particular dot decreases to draw the user's attention. The data obtained within this procedure is then used for calibrating a remote tracking system. Afterwards, the procedure is repeated using 25 dots placed at *random* positions. Each session uses a different set of dots that is used for measuring the accuracy of different mapping methods.

Offline testing

However, this strategy has one drawback. Since each single test is unique regarding its test-points, different algorithms or mapping methods are hard to compare with each other. A large amount of testing sessions are required in order to obtain an even coverage of test-points across the whole computer screen for each test case. Therefore – in the context of this thesis – a different testing strategy is used. Instead of repeating the whole procedure for each single method that is about to be tested, an offline testing process is used. While the proband makes use of the tracking device, his eye movement is recorded to a video-file.[1] In addition, the frame-accurate screen coordinates – that indicate where the user should have been looking at – are stored. This offline testing strategy allows to test and compare different combinations of algorithms directly, without forcing the proband to undergo the procedure multiple times.

Calibration- and test-patterns

The calibration- and test-patterns used for this offline test also differ from the ones *Ramanauskas et al.* use. Since the fixed- and the head position invariant tracking procedure have different requirements regarding the testing setup, they are treated separately.

5.2.1 Fixed-head tracking tests

For evaluating the basic image processing algorithms – such as image segmentation and pupil/glint detection – as well as the performance of the RBF-interpolation, the probands are told to calibrate the system using a total of 9 points that are displayed on the computer monitor. These points are arranged in a 3×3 pattern that covers the whole screen. In order to improve the concentration of the user, the points do not pop up abruptly as suggested by *Ramanauskas et al.* since this implies a short

1 The file is compressed with the lossless *HuffYUV* video codec.

period of reorientation while the user searches for the new position of the dot. Instead, the dot moves smoothly across the – randomly selected – target points of the calibration grid.

During this procedure the calibration application shows a red dot on a black screen. This dot is used to draw the user's attention and guides his eye movement while calibrating. In order to maximize the user's focus on the dot, its movement and behaviour can be subdivided into three phases as seen in *figure 5.1*:

Movement phase At the beginning or when a new point is targeted, the dot moves to this new position following a straight path. The maximum velocity of the dot is limited to a value that does not prolongate the whole testing process but still allows the user's eyes to follow it. The dot is dynamically accelerated and decelerated in a way that results in a sinusoidal easing curve for the velocity. This way, the movement of the dot can be followed easily without the need for reorientation during the whole process.

Attention phase After the dot has reached its destination point, its size is increased temporarily in order to draw the user's attention. Within a short period of time, the dot morphs back to its initial appearance. The attention phase allows the user to prepare himself for the critical *recording phase*, since he is not allowed to blink during that period.

Recording phase This phase lasts exactly two seconds. During this time period, 10 "snapshots" of the user's eye are made – which means that the respective frame numbers are stored including the actual position of the dot. Since the jitter of the fixation point during this phase should be as small as possible, the mid-point of the dot flashes quickly thus attracting the attention of the proband.

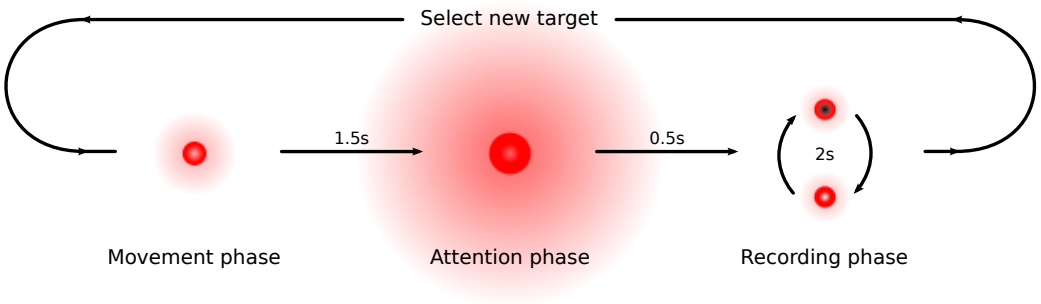


Figure 5.1: The three phases of the calibration procedure for attaining the maximum attention of the user.

These three phases are repeated until all calibration points have been processed. The same procedure is also used for acquiring data for testing the calibration as well as for training and testing the neuronal network in the head position invariant case. A typical calibration procedure consisting of 9 points lasts approximately 36 seconds.

Choice of test-points

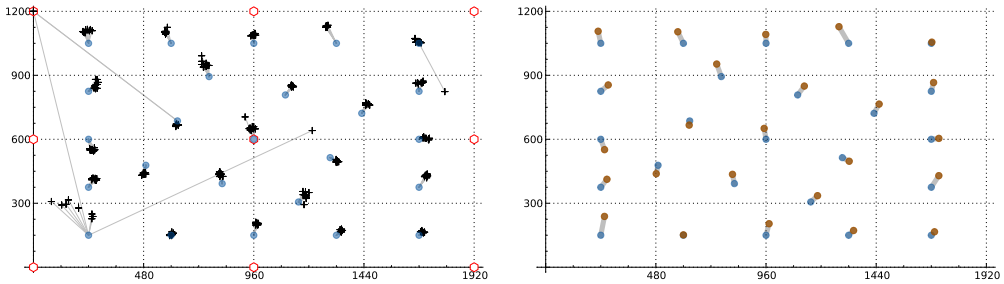
In order to measure the accuracy of the tracking system, a separate set of test points is required on which the calibration is applied to. This test-pattern consists of 25 points that are distributed across the screen. The positions of these points are illustrated in *figure 2.13* (page 30) and are marked by blue points. The placement of these points has been chosen in a way that covers a large area of the computer monitor while not lying directly on the line between two calibration points in the majority of cases. The procedure for obtaining the test-data is done directly after the calibration has been finished.

Processing the calibration data

After all calibration- and test-points have been examined, a total of 10 pupil-glint-vectors are available per point. However, these PGVs cannot be used directly, since they may contain outliers induced e. g. by temporary loss of concentration or blinking. These outliers are filtered out by firstly calculating the mean value of all PGVs. Those points whose distance to this value differs more than the standard-deviation over the complete set of points, are filtered out. Afterwards, the mean value of the remaining points is used as the final PGV for the specific test-point. An example for this procedure is given in *figure 5.2*.

Fixation of the probands' heads

Since the fixed-head mapping scheme is very sensitive to head movement, the probands' heads need to be fixated in some way. This is usually done by using a



(a) Original gaze points **(b)** Averaged points used for testing

Figure 5.2: Illustration of the outlier removal procedure: The left plot shows the originally estimated gaze-positions (black crosses) and their corresponding reference points (blue circles). The right plot shows the final points that are used for further processing.

chin rest or a *bite bar*. Since none of these devices were available, an improvised construction had to be utilized. To achieve at least a loose fixation of the head, a plastic box was used that was placed in front of the monitor. The distance between the box and the monitor was approximately 75 centimetres. In order to make the whole testing procedure more comfortable for the probands, the chin did not lie directly on the box. Instead, the head rested on a piece of foam. Unfortunately – as it turned out later – this construction was not able to compensate for unaware head movements caused by e. g. breathing. A more detailed analysis on this topic is given in *section 5.3*.

5.2.2 Head position invariant tracking tests

Fortunately, for acquiring training-data in the head position invariant case the user's head does not need to be fixated. However, a nine point calibration procedure is not sufficient for training a neuronal network. Therefore, the calibration and testing procedure has to be adapted to fit the needs of the RBF-network.

The procedure for acquiring the gaze-data, as described in the section before, can also be used for obtaining test-vectors for the invariant case. For the tests discussed in *section 5.3.2*, a total of four data sets have been created:

Optimal training set This set of training-vectors has been obtained by repeating the nine point calibration procedure – as described before – ten times, thus gaining a total of 900 training-vectors.[1] During the calibration, the user rearranged his head position and -orientation with each new target point. However, during the recording phase of each point, the user was instructed explicitly not to move his head.

Background set Since the foregoing training procedure is not suitable for practical purposes, a second set of training-vectors has been created. This one has not been obtained using the testing procedure. Instead, a background daemon was started that monitored the system for mouse-clicks. The user was explicitly told to try to look near the mouse-cursor when performing a click action. Each time such an action was performed, the position of the mouse-cursor as well as the actual camera frame was recorded. During a time period of about 30 minutes, a total of 128 training-vectors have been obtained.[2]

1 Each of the 90 points contain 10 individual input-vectors. In this case, no outlier elimination is applied.

2 In order to achieve an even screen coverage while using the daemon, the user played some games of *Mahjongg*.

Evaluation set In order to test the accuracy of the RBF-network trained with the training sets as seen above, a third set of test-vectors has been created. This set consists of 750 input-vectors and their corresponding screen coordinates. It has been acquired by completing the 25 point test-pattern three times. The user was allowed to move his head naturally during this procedure.

Dense set This set has been created in order to test how the number of training-vectors used in the RBF-network correlates with the attainable accuracy of the system. The set has been obtained by repeating the 25 point test-pattern procedure 9 times, thus consisting of a total of 2250 test-vectors.

The distance of the persons head to the computer monitor is assumed to be the same as in the fixed-head tests (about 75 centimetres). However, since the user was allowed to move his head freely, this value is only a rough approximation.

5.3 Results and discussion

Before the individual tests are presented and discussed, some additional notes about the interpretation of the test-results have to be provided. These results usually consist of the average distances between the estimated gaze positions and the associated reference points the user was intended to look at. These distances are measured in pixels, since the mapping algorithm returns the absolute screen coordinates of the current gaze position. In order to allow a direct comparison with related work, these pixel values are converted to *degrees of visual angle*. This value can be approximated by using the following term:

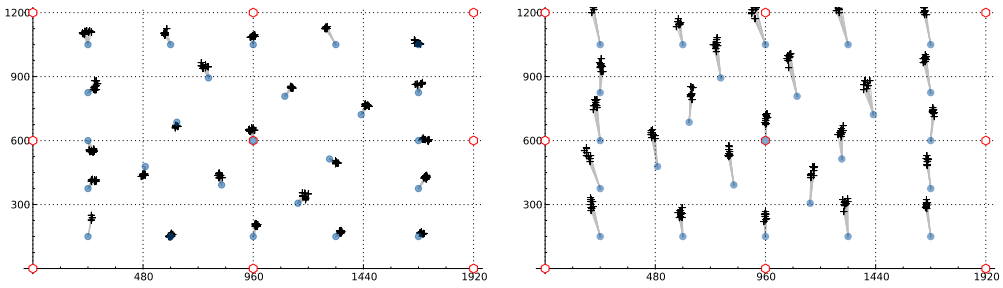
◀ Degrees of visual angle

$$v_{\text{deg}} = \frac{180}{\pi} \arctan\left(\frac{v_{\text{pix}}}{ppc \cdot d}\right)$$

Here, the variable d denotes the distance between the user's head and the computer monitor whereas ppc stands for *pixels per centimeter*. This value has been measured to be approximately $37.7\frac{px}{cm}$.

Compensating for unintentional head movement

During the fixed-head gaze tracking tests, some of the probands moved their head slightly. Some of these movements where periodic – and may be caused by breathing – whereas some of them appeared to be directed. *Figure 5.3* shows two examples for such situations. Especially in the second plot it is easy to see that the shifted gaze-points degrade the overall accuracy of the tracking result significantly. Since this fault *is not* due to the tracking and mapping algorithm, I tried to compensate for this kind of head movement afterwards. This is done as



(a) Continuous breathing movement **(b)** Head shift movement

Figure 5.3: Effects of unintentional head movement on the systems accuracy.

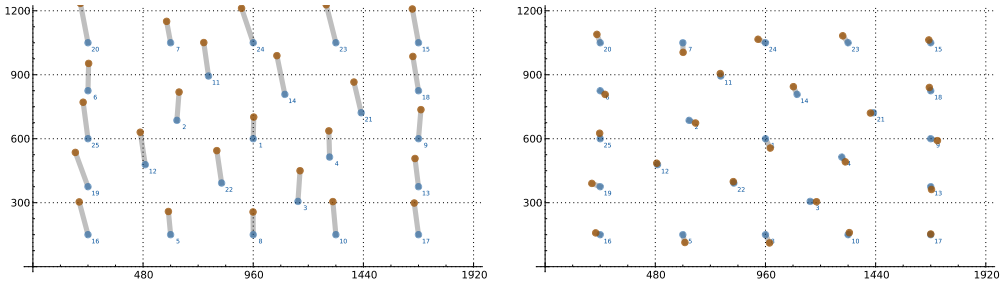


Figure 5.4: By using the average directed error vector, uniform head movement can be compensated up to a certain degree.

follows: At first, outliers are removed as described in *section 5.2.1*. Afterwards, the average error vector is determined. This vector is then used to shift all estimated gaze-points in order to minimize the error induced by the directed head movement. This procedure is depicted in *Figure 5.4*. Unfortunately, when using this approach irregular head movement cannot be compensated completely. Therefore, using a bite bar or a similar construction to fixate the head more firmly may result in a better average accuracy than presented in the following sections.

5.3.1 Fixed-head gaze tracking

In the previous chapters some constant values – like the threshold constant on *page 20* – have not been justified yet. This section describes how these values have been obtained. In addition, the attainable accuracies of different tracking and mapping methods are compared with each other.

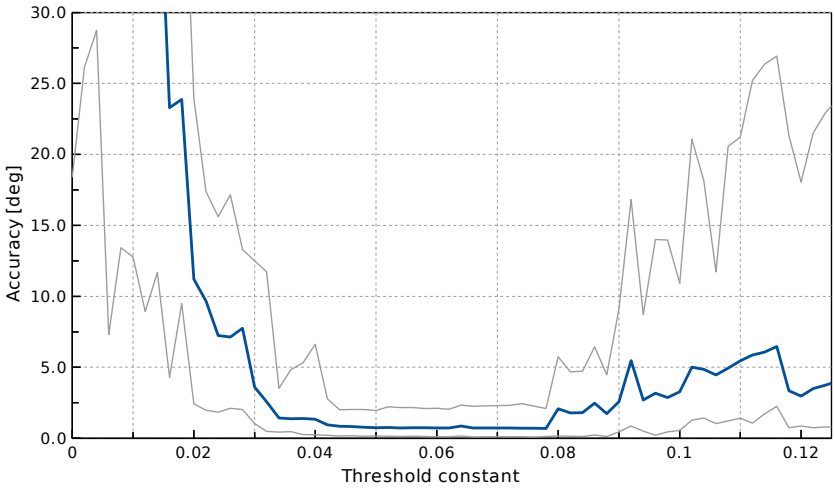


Figure 5.5: Estimation accuracy in relation to thresholding value. The blue line represents the average accuracy across all probands. The gray lines stand for the averaged minimum and maximum error within a 25 point test case.

Determining a common threshold constant

In order to test the accuracy of the various algorithm combinations, a reliable detection of the user's pupil is absolutely necessary. The goal of the first test is to find a thresholding constant that can be used across the majority of persons.

Test description

The test has been conducted using the offline test-data of all seven probands. For each of these datasets, a series of different thresholding values has been applied ranging from 0.0 to 0.2. The images segmented using these thresholding values are then further processed as described in *section 2.2*. For determining the pupil-center the *modified least squares ellipse fitting* method – as described on *page 25* – has been used. Since this method needs a closed contour of the segmented pupil region it is very sensitive to the chosen thresholding value, making it a reasonable choice for this test. The PGVs obtained this way are then mapped to screen coordinates using the *bump RBF-interpolation* as shown in *section 2.4.2*.

Results and discussion

For each proband and each thresholding value, the minimum, maximum and average errors across all 25 test-points have been calculated. *Figure 5.5* shows the mean values across all probands of these three values in relation to the chosen thresholding constant. As it can be seen, the plot features an interval between 0.05 and 0.07 where the average accuracy remains almost constant. Outside this region, the tracking algorithm quickly collapses due to incomplete or erroneous pupil regions. This result immediately suggests to choose a value of approximately 0.06 for use as the default thresholding constant.

Precision of ellipse fitting methods

In *chapter 2* four different methods for determining the pupil-center have been presented. All these methods provide subpixel accurate results but have different characteristics regarding their robustness. In this test case they are directly compared to each other. In addition, a non-subpixel accurate method has been tested that only uses the bounding box of the pupil-region to estimate its center.

Test description

The test has been performed similarly to the former one. The common threshold value has been chosen according to the results of the thresholding test. Except for the "Bounding box" method, the glint has been detected using the centroid of this region. The video data of each proband has been processed using the pupil fitting methods introduced in *section 2.3.4*. These include:

- Centroid calculation (*page 22*)
- Least squares ellipse fitting (*page 23*, referred to as "Ellipse")
- Modified least squares ellipse fitting (*page 25*, referred to as "Binary ellipse")
- Outlier eliminated least squares ellipse fitting (*page 25*, referred to as "Complex ellipse")

Afterwards, the PGVs have been mapped to screen coordinates using the *bump RBF-interpolation* method with optimal kernel-width.

Results and discussion

As can be seen in *figure 5.6* and *table 5.1*, the average accuracy of all tracking algorithms – except for the *Bounding box* method – was better than *one* degree of visual angle. However, both novel methods for determining the pupil-center (*Binary ellipse* and *Complex ellipse*) perform considerably better than the remaining ones. As the standard deviation across the probands indicates, these methods are also less sensitive to the individual differences between the test persons.

Table 5.1 also shows that the best performance was achieved by *Proband B*. Indeed, this proband seemed to be the one who was able to control his head movement best, since the directed error vector of this dataset was the smallest one among all others. The accuracy he achieved translates to approximately *13 pixels* on the computer screen. This indicates that the average accuracy could be much better if the probands' heads were fixated more firmly. However, these results are comparable with the accuracy of the system proposed by *Parkhurst et al.*, that has been stated as approximately *one degree of visual angle* [WLBP05].

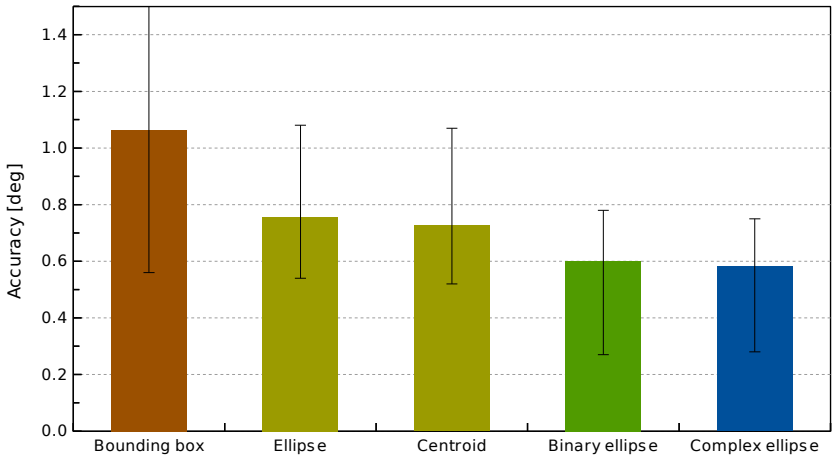


Figure 5.6: Performance of the different ellipse fitting methods. The error bars indicate the minimum and maximum average errors across all probands.

Method	Bounding box	Ellipse	Centroid	Binary ellipse	Complex ellipse
Proband A°	0.82	1.08	0.93	0.78	0.75
Proband B	0.56	0.54	0.52	0.27	0.28
Proband C•	2.86	0.82	0.52	0.60	0.56
Proband D	0.98	0.80	1.07	0.61	0.61
Proband E•	0.70	0.59	0.69	0.60	0.68
Proband F	0.85	0.76	0.78	0.74	0.56
Proband G	0.68	0.70	0.58	0.60	0.63
Average	1.06	0.76	0.73	0.60	0.58
Std. dev.	0.80	0.18	0.21	0.16	0.15

Table 5.1: The achieved accuracy of the different ellipse fitting methods. Probands wearing spectacles are marked with a filled dot (•). *Proband A* was wearing contact lenses (○). All values are stated in degrees of visual angle.

Performance of the different interpolation methods

The previous tests all used the *5-point bump RBF-interpolation* for comparing the tracking algorithms performance. This interpolation method has been introduced in *section 2.4.2, page 34*. In this paragraph, the reasonability of this choice is justified by comparing this mapping method with the remaining ones.

Test description

Again, this test was performed using a common threshold value of 0.6. The pupil-center was determined using the *complex ellipse* method. All the pupil-glint-vectors

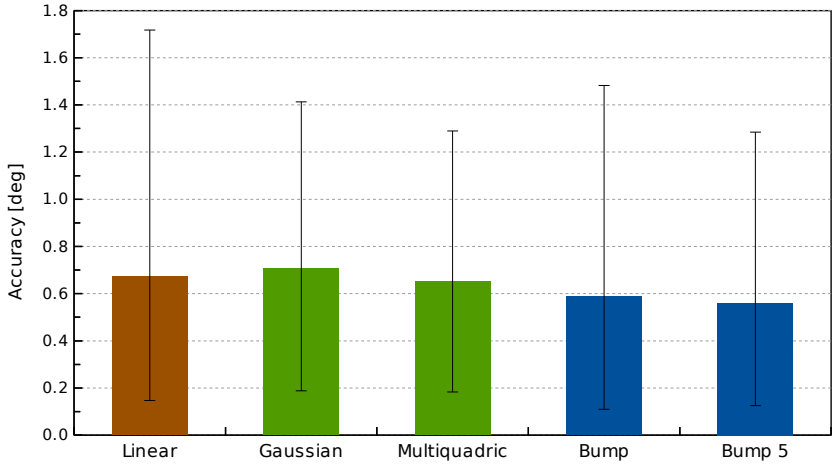


Figure 5.7: Correlation between the attainable accuracy and the mapping method. The error bars depict the minimum and maximum value across all probands for each technique.

acquired this way are then mapped to screen using the following interpolation methods:

Linear The Delaunay triangulation based linear interpolation as presented on *page 29* is used. All nine calibration points were utilized by this method.

Gaussian, multiquadric and bump An interpolation RBF-network has been established using the Gaussian-, multiquadric- or bump-function as underlying kernel-function. This technique has been described in *section 2.4.2, page 31*. For each kernel-function and each dataset, the optimal kernel-width has been determined as shown on *page 35*.

"Bump 5" Similar to the other RBF-interpolation methods, this method uses a *bump*-kernel-function RBF-network. The optimal kernel-width is estimated by using the calibration data as shown on *page 37*, effectively constituting a 5-point interpolation.

Results and discussion

The results of this test are depicted in *figure 5.7*. It is easy to see, that the Gaussian- and multiquadric RBF-interpolation methods are almost on a par with the results of the linear interpolation – that is about 0.67 degrees of visual angle. Interestingly enough, the average accuracy of the 5-point bump-interpolation is slightly better than the 9-point bump-interpolation (0.56 versus 0.58 degrees of visual angle). This relatively small difference may be due to statistical coincidence, since only seven probands participated in the tests. However, the 5-point bump-interpolation performs better than the linear interpolation method on all tested probands. All in

all, compared with the linear interpolation case, it allows to increase the overall accuracy by approximately 20% and should therefore be the preferred choice.

Reducing the number of calibration points

As the foregoing test indicates, the bump-RBF-interpolation performs particularly well, even if the number of sampling points is reduced. The following test addresses this behaviour by reducing the number of calibration points even further.

Test description

Unlike the previous tests, this one only utilizes the data of the *three* probands that exhibit the lowest head-motion induced error. The average accuracy attained by *Proband B*, *Proband C* and *Proband D* is below 0.5 degrees of visual angle. During the test, the number of calibration points used for the interpolation procedure has been reduced successively. The following calibration patterns have been tested on the linear interpolation scheme as well as the bump-interpolation with optimal kernel-width (the numbering is taken from *figure 2.13, page 30*):

9 points Points 1 to 9

5 points Points 1, 3, 5, 7 and 9

4 points The edge points 1, 3, 7 and 9

3 points Points 3, 4 and 9 - This point-set cannot be applied to the Delaunay based linear interpolation since this method is not able to extrapolate values outside the convex hull of the sampling points.

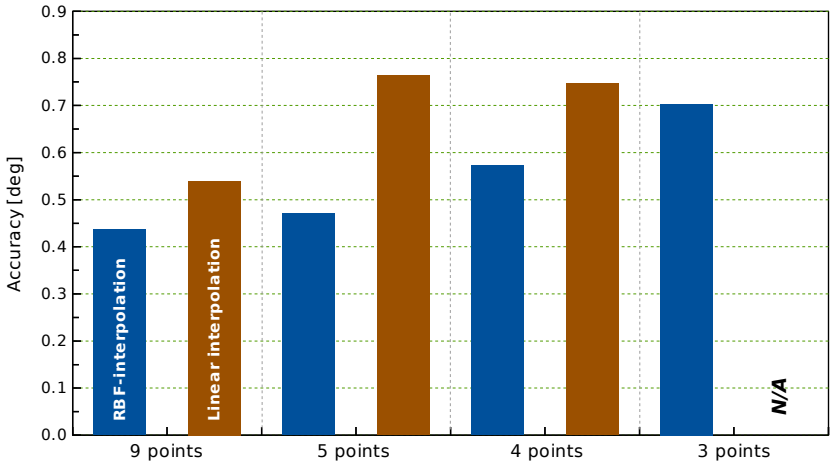


Figure 5.8: Performance of the different interpolation approaches when reducing the number of sampling points. The 3-point linear interpolation cannot be tested and is therefore omitted.

Results and discussion

As it turned out, the RBF-interpolation is still able to attain acceptable results if only *three* sampling points are used. *Figure 5.8* compares the accuracy achieved by the linear interpolation with the results of the bump-interpolation. It was to be expected that the overall accuracies of the mapping methods decrease as the number of sampling points is reduced. Surprisingly, the 3-point interpolation procedure still outperforms the 5-point linear interpolation. However, this result is not directly applicable to the concrete tracking system, since it needs to be verified on a larger base of probands. Also, at least one additional calibration point is required for determining the optimal kernel-width automatically. Still, this result motivates further investigations in the performance of RBF-networks within this context.

Influence of the image resolution

The last test in the context of fixed-head gaze tracking analyzes the influence of a reduced image resolution to the overall accuracy of the tracking system. This question is especially relevant when the costs of the tracking hardware should be reduced further. Low-cost camera systems often exhibit a video resolution that is much smaller than the 1280×800 pixels provided by the actual system. It should be noted that only a small section within the camera image is used for the analyzation process. This is due to the construction of the second prototype as discussed in *section 4.1.2*. In order to calculate the effective resolution of the camera image, its width has to be multiplied with a factor of 0.37

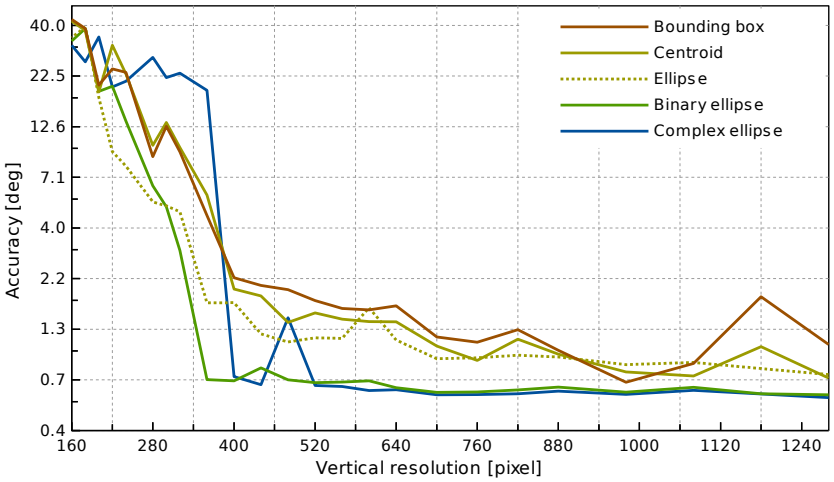


Figure 5.9: Changes in the attainable accuracy as the image resolution increases. It can be seen that both novel tracking methods deliver stable results once the resolution is higher than 640×400 pixels.

whereas the height remains approximately the same. For example, the effective resolution of the 1280 × 800 pixel camera image is about 474 × 800 pixels.

Test description

The setting of this test is almost the same as in the test for comparing the different ellipse fitting methods. This time, the video images were scaled down proportionally before they were analyzed further. The tested resolutions ranged from 1280 × 800 (unscaled) down to 160 × 100 pixels. A total of 24 different resolutions have been tested on each pupil-center detection method.

Results and discussion

As shown in *figure 5.9*, the analyzed tracking techniques exhibit different behaviors as the video resolution decreases. On the one hand, the *bounding box* and the *centroid* methods reveal an unsteady performance. On the other hand, both newly suggested methods – the *binary ellipse* and *complex ellipse* techniques – appear to be very robust regarding changes in resolution. In the interval between 1280 × 800 down to 640 × 400 pixels, the accuracy of both methods steadily diminishes from approximately 0.6 to 0.65 degrees of visual angle. As the resolution decreases further, the accuracy quickly collapses, since the pupil- and glint detection methods are no longer able to provide the fitting algorithms with accurate feature regions. However, stable results can be achieved even with a resolution as small as 640 × 400 pixels – which translates to an effective image size of 237 × 400 pixels. This allows for using cheaper camera systems in future prototypes.[1]

5.3.2 Head position invariant gaze tracking

In *chapter 3*, two methods for head position invariant gaze tracking have been introduced. The performance of the first one, being the cross-ratio method presented by *Yoo and Chung*, is directly compared with the second one that utilizes RBF-networks for function approximation within this section. The latter technique involves some unknowns – such as number of used neurons, structure of training-vectors and the chosen clustering method – that need to be determined empirically.

Both presented techniques need the corneal reflections of the infrared markers to be detected reliably. Therefore, the filter parameters introduced on *page 45* have been adjusted in a very restricted way that blocks the majority of misdetections. However, this may also filter out some correct results. In addition, the structural limitations of the hardware prototype also lead to a certain amount of frames to

1 Of course, resizing a high resolution image to a smaller one is not exactly the same as obtaining an image of a low-resolution camera. For that reason, additional tests – measuring e. g. the robustness to image noise – should be made.

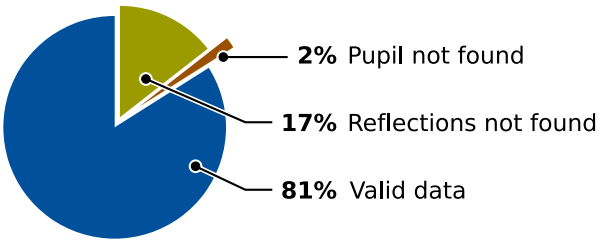


Figure 5.10: Due to missing marker reflections or eye blinks, a certain amount of frames is discarded.

be invalid. In order to analyze this behaviour more closely, the tracking results of all available test-videos have been examined. All in all, a total of 15615 images have been used for the invariant case tracking tests. Due to blinking, initialization phases[1] and – very rarely – misdetections, 292 frames did not contain valid pupil positions. In consequence of the restrictions as described before, the error rate on the reflection detection was much higher. A total of 2688 frames have been discarded due to missing reflections. *Figure 5.10* shows a graphical representation of these values.

Optimal number of hidden RBF-neurons

In *section 3.4.1*, a procedure for training the RBF-network has been presented. In this context, the issue of choosing the right amount of hidden RBF-neurons could not be solved, since this number depends on the underlying function that is about to be approximated. As *figure 3.7* (page 56) indicates, the amount of RBF-neurons may influence the quality of the tracking result. Unfortunately, only little is known about the mapping function. Therefore, the optimal neuron count has to be determined empirically.

Test description

In order to determine the optimal number of RBF-neurons, the *optimal training set* (as described on *page 94*) has been used to train a couple of RBF-networks featuring different neuron counts. For detecting the pupil-center within this dataset, the *binary ellipse* method has been utilized. The same configuration is also used for processing the *evaluation set* that contains three test-session for determining the average accuracy of the RBF-network. For obtaining suitable neuron-positions, the *K-Means* clustering method has been used. Since this method relies on a random initial neuron placement, the whole procedure – including clustering, training the

1 The first few frames of each test-video do not contain valid tracking data since some sub-algorithms need values that are averaged over a certain amount of frames.

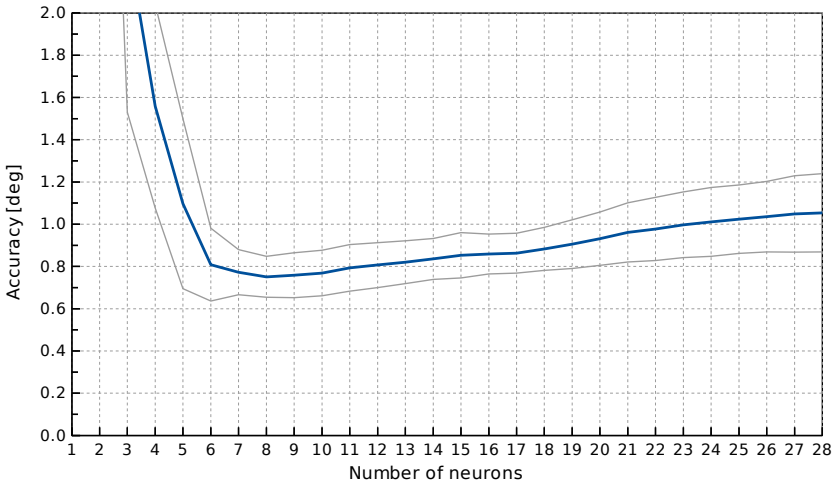


Figure 5.11: The attainable accuracy in relation to the number of neurons used in the RBF-network. The blue line represents the average accuracy across all 750 trials whereas the gray lines stand for the standard deviation.

RBF-networks and evaluating the approximated function – is repeated 250 times for each of the three evaluation datasets. Afterwards, the average accuracy as well as the standard deviation of all this data is determined in order to obtain stochastically accurate results.

Results and discussion

These results are shown in *figure 5.11*. As it can be seen, the curve representing the attainable accuracy in relation to the number of neurons agrees with the theoretical considerations: If the number of neurons is very low, the network is not able to learn from the given training-vectors adequately. Otherwise, as the number of neurons increases, the network slowly looses its generalization capabilities and an "overlearning" situation occurs. This effect also has been described in [LZSC04]. In the latter case, single outliers in the training-data are more likely to influence the overall accuracy.

Interestingly enough, the best accuracy for the *optimal training set* is achieved if only *eight* neurons are used. This number is surprisingly low, especially since the dimension of the input-vectors is rather high. However, this neuron count also works on the other training-sets as seen in the upcoming tests. Therefore, this value may be assumed to be the optimal one.

Clustering methods

When using RBF-networks for function approximation, a reasonable placement of the RBF-neurons is a crucial factor for attaining a good accuracy. It is plausible that neurons with a large distance to an input-vector have

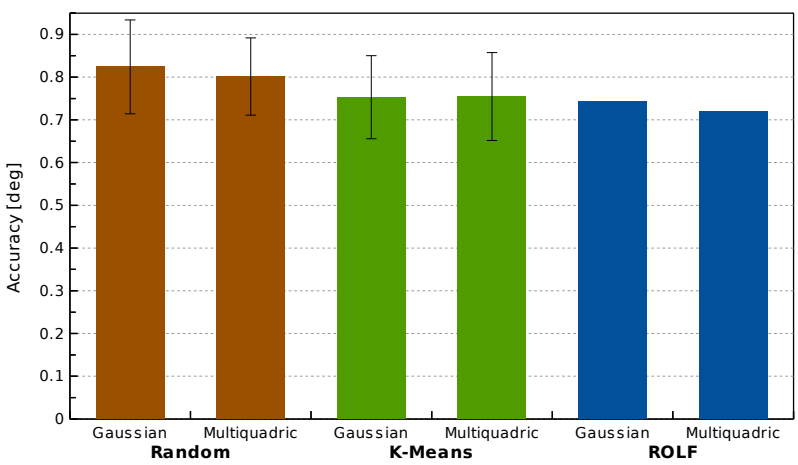


Figure 5.12: The performance of different neuron placement strategies performed on the same dataset. The error bars represent the standard deviation.

less influence to the approximated value than neurons being close to this vector. Therefore, if the neuron positions are poorly chosen – e. g. when they are all placed in the same region of input-space – inputs being outside this region cannot be processed sufficiently. In *section 3.4.2* three different strategies for placing these neurons have been discussed. Within this test, their performance is compared with each other.

Test description

The differences between the available placement strategies are tested on the same datasets as before. In addition to the Gaussian kernel-function that has been used so far, the multiquadric also has been tested. Like in the previous test, the *K-Means* as well as the *random* placement strategies have been repeated 750 times – each time using a different initial neuron placement. Due to its deterministic nature, the tests for the *ROLF* strategy presented in *section 3.4.3* were performed only once. Since to my knowledge no automated procedure for determining the parameters of the ROLF network is known, these parameters have been optimized by hand.

Results and discussion

Figure 5.12 shows the results of these tests. It can be seen, that the performance of the K-Means clustering method is almost on a par with the ROLF network. Since the latter method does not involve the risk of choosing a poor initial configuration, it is – theoretically – the best choice among the others. However, it cannot be used practically until a method for determining the correct parameters is found. As expected, the average accuracy of the random placement method is inferior to the remaining methods. This becomes even more evident when the worst performances among all 750 trials are compared with each other. While the random strategy attained a worst-case accuracy of 1.42 degrees of visual angle, the K-Means strategy

Strategy	Random		K-Means		ROLF	
	Gaussian	multiquadric	Gaussian	multiquadric	Gaussian	multiquadric
average	0.82°	0.80°	0.75°	0.75°	0.74°	0.72°
worst-case	1.42°	1.21°	1.19°	1.28°	0.87°	0.89°
best-case	0.53°	0.54°	0.58°	0.54°	0.66°	0.63°
std. dev.	0.11°	0.09°	0.10°	0.10°	N/A	N/A

Table 5.2: Tabular representation of the performances of different neuron placement strategies and kernel-functions.

always performed better than 1.19 degrees.

As it can be seen in *table 5.2*, the multiquadric kernel-function performs only barely better than the Gaussian one. Since the former kernel-function also implies an unfavorable numerical behavior, it is not recommendable for this tracking scenario.

Influence of training-data

All previous tests used a relatively large set of input-vectors for training the RBF-network. Most certainly, the time-consuming procedure for obtaining this training-data is not appropriate for end-users. Therefore, further observations are needed in order to determine the amount of training-data minimally needed to achieve appropriate results. For that reason, the *dense set* has been created, consisting of training-vectors taken from nine test-sessions, each using the 25-point test pattern.

Test description

Like in the tests before, the K-Means placing strategy is used for training the network consisting of eight hidden neurons. In order to test the relation between attainable accuracy and number of used training-vectors, the following procedure is performed:

1. From the *dense set* – that consists of 2250 training-vectors – a subset of n training-vectors is selected randomly.
2. The RBF-network is trained using these vectors.
3. The remaining test-vectors of the *dense set* are used for determining the average accuracy of the network.
4. Steps 1 to 3 are repeated 75 times and the overall average accuracy is obtained using the accumulated data.

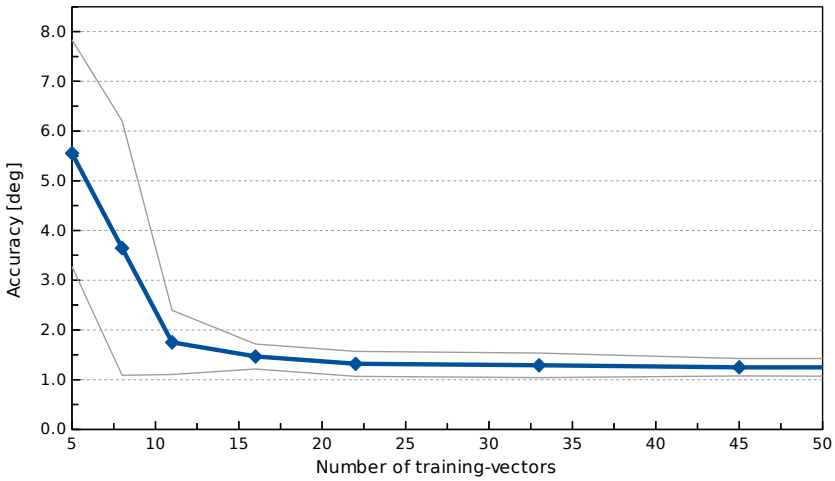


Figure 5.13: Average accuracy of the tracking system in relation to the number of input-vectors used for training the RBF-network. Again, the gray lines represent the standard deviation.

This procedure is then reiterated for different values of n, ranging from 5 to 904. It is important to know that the results obtained this way do not undergo an outlier detection and the estimated gaze positions for each point of the 25-point test pattern are not averaged. Since the training-vectors are taken from random positions, there is no temporal connection between the sampling points. Therefore, averaging their tracking results would not be feasible.

Results and discussion

This missing averaging procedure results in a lower accuracy than attained in the tests before. However, the results still have a generic significance. *Figure 5.13* shows a graphical representation of the accuracies that have been achieved. As it can be seen, up to a training-vector count of 22, increasing the amount of training-data also increases the overall accuracy. For higher vector counts, the additional training-vectors only influence the standard deviation of the result. In the interval between 22 and 904 vectors, the average accuracy stays approximately the same (1.29 degrees of visual angle) while the standard deviation slowly decreases (0.25 down to 0.16 degrees).

The theoretically needed amount of training-vectors is rather low. However, these tests have been done on a expansive database of evenly distributed training-points. In more realistic scenarios, a larger amount of training-vectors would be needed due to the presence of outliers.

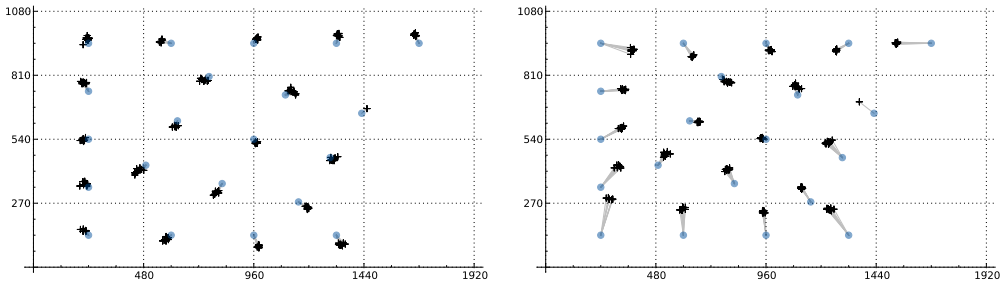
Background aggregation of training-data This rather short test has been done in order to try out an alternative concept for obtaining the training-vectors. For this test, the *background set* has been used. This set has been acquired in a more user friendly way: The proband is allowed to create an increasing amount of test-data during normal computer interaction. Each time he triggers a click-action, one training-vector is added to the training-database. These vectors then can be processed further using the previously described techniques. For evaluating the quality of this training-set, the *evaluation set* has been used once more.

Results and discussion

Compared to the preceding tests, the accuracy attained when using the *background set* is rather low. To be more specific, this set achieved an average accuracy of 2.05 degrees of visual angle, whereas the explicit calibration procedure attained an accuracy of 0.75 degrees. However, there are two possible reasons for this behaviour:

Presence of outliers The tracking algorithm – including the marker reflection detection – is designed to use averaged values of preceding frames for various filtering actions. The data of the *background set* however, does not consist of continuously taken video frames. Instead, only the frames are stored that are associated with a click-action initiated by the user. Therefore, the reliability of the tracking procedure is limited and the number of outliers may be higher than average.

Insufficient head movement When looking at the gaze estimation results shown in *figure 5.14*, the estimated gaze points obtained by using the background calibration are apparently warped to the midpoint of the screen. This distortion may be explained by an insufficient volume in space covered by the user's



(a) Explicit calibration **(b)** Background calibration

Figure 5.14: The estimated gaze positions when an explicit calibration procedure is used **(a)** compared to the calibration data acquired by the background daemon **(b)**. The missing regions of the test-pattern are due to constructional limitations.

head motion during the background calibration procedure. For example, if the user moves his head within a certain region of space during the calibration, but the tests for evaluating the accuracy are taken outside this region, the RBF-network has to extrapolate the gaze positions. This may explain the uniform warp of the estimated gaze points.

Concluding this test, when using a larger amount of background training-data in combination with an advanced outlier removal technique, this approach has the potential of achieving much better results than presented here. However, some adjustments to the tracking algorithm may have to be made.

Comparison of different techniques

The last test regarding head position invariant gaze tracking summarizes the results of the previous tests and also compares the performance of the RBF-network based mapping procedure with the one suggested by *Yoo and Chung*.

Test description

In order to test the latter method, the calibration values α_n, $n = 1, 2, 3, 4$ as described on *page 49* are determined by using the *optimal training set*. Afterwards, the *evaluation set* is used to acquire the average accuracy of this mapping technique. Just in order to emphasize the sensibility to head movement of the fixed-head tracking method, the same datasets have been processed using the algorithms presented in *chapter 2*. In this case, only the pupil-glint-vector is used and the corneal reflections of the infrared markers are omitted.

Results and discussion

The results of this comparison is shown in *figure 5.15*. Unsurprisingly, the result of the conventional fixed-head method is unusable for any practical applications. For clarification, the estimated gaze-positions of this method have been illustrated in *figure 5.16*. Due to the different hardware setup and constructional issues as discussed in *section 3.3.2*, the attained accuracy of the cross-ratio technique is inferior to the one stated by the original authors. They claim to achieve an average accuracy of about 0.9 degrees of visual angle [YC05] whereas the results of this test exhibit an accuracy of 2.62 degrees. The RBF-network based mapping method presented in this work allows for an accuracy of 0.75 degrees of visual angle and is therefore able to compete with the method by *Yoo and Chung*. However, the calibration of this new technique is still quite extensive, which does not apply to the latter method.

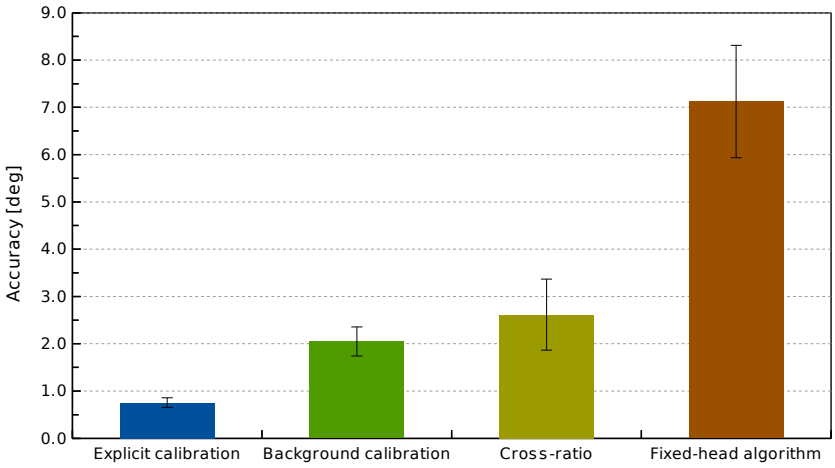


Figure 5.15: Comparison of the attained average accuracies of the presented methods. The error bars represent the standard deviation across the different test-sets.

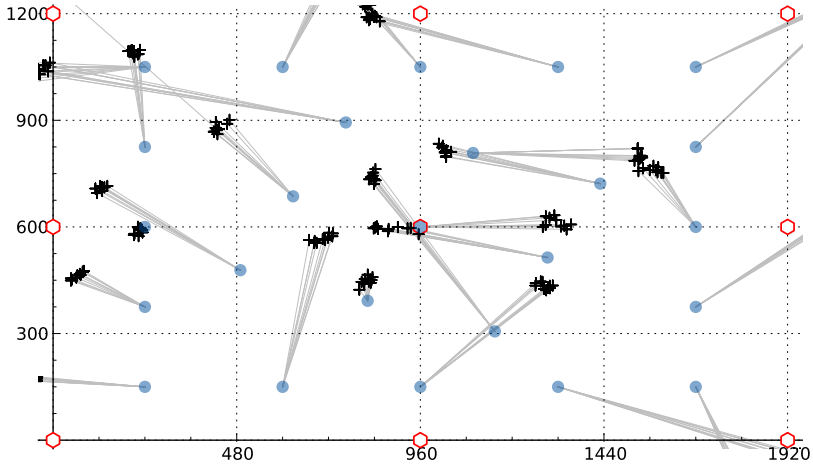


Figure 5.16: The estimated gaze positions under free head movement with the conventional fixed-head algorithms applied.

5.3.3 Computation time and latencies

When using gaze tracking systems for interactive applications, very low latencies are required. In this test, the latencies induced by the different tracking algorithms are analyzed and compared with each other.

Test description

For this test, an exemplary tracking-video consisting of 989 frames has been used. The times for processing all these frames in different combinations of resolutions and tracking methods have been measured using the unix `time` command. Additionally, the time needed for decoding the video files, format conversions and resize operations have been measured separately. All tests have been performed on an *Intel Core 2 Duo* processor running at 2.50 *GHz*. No special optimizations have been done to the algorithms and only a single processor core was utilized. Since the mapping procedure – as soon as the RBF-network has been trained – only needs very few computational resources, it has not been taken into account for these tests.

Results and discussion

All timing information acquired by this test are listed in *table 5.3*. As it can be seen, a vast amount of time is used for reading and pre-processing the video file. This pre-processing includes decoding the HuffYUV encoded video file, converting the images to grayscale format[1] and (optionally) resizing it to a lower resolution. This way, for a 1280 × 800 pixels video frame, a total time of 1.78 milliseconds is spent.

Resolution	Method	milliseconds/frame		frames/second	
		total	processing	total	processing
1280 × 800	Binary ellipse	2.51	0.72	40	138
	Complex ellipse	2.62	0.84	38	119
640 × 400	Binary ellipse	2.24	0.23	45	443
	Complex ellipse	2.26	0.25	44	405
1280 × 800 **invariant**	Binary ellipse	2.56	0.78	39	128
	Complex ellipse	2.66	0.88	38	114

Table 5.3: Time performance for various combinations of resolutions and tracking methods. “Processing” values represent timings without decoding overhead.

1 OpenCV firstly converts the YUV images to an internal RGB format before allowing a grayscale conversion. This way, additional processing time is wasted.

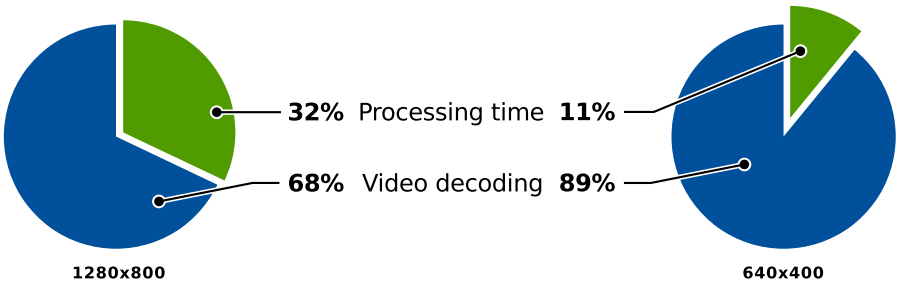


Figure 5.17: Depending on the resolution of the tested video frames, the time needed for decoding and resizing the images varies significantly.

For lower resolutions, the relative amount of time spent for pre-processing grows even further. *Figure 5.17* illustrates this situation.

When including this additional time, the head position invariant tracking system described in this thesis is able to process at least 38 video frames per second. If only the raw image processing time is taken into account, frame rates up to 128 frames per second are reachable. In either case, these frame rates can be considered as being interactive. If more efficient pre-processing techniques are utilized, the algorithms would be efficient enough to run on slower hardware or even on embedded platforms.

Chapter 6

Conclusion and outlook

The goal of this thesis was to conceive and construct a low-cost gaze tracking device for interactive use. This included a hardware device as well as the software for operating it. Within this work, various algorithms for analyzing the camera images and for mapping the results to a computer screen were presented. In order to allow the user to move his head naturally while using the device, a novel approach for compensating for head movement was presented.

This last chapter summarizes this thesis and gives a short outlook about possible future enhancements to the hardware system as well as the different algorithms used for gaze tracking.

Hardware

For acquiring the video frames needed for tracking the user's eyes, two prototypes of a gaze tracking device were constructed. Both devices had different characteristics regarding their construction and the induced costs. The first prototype – despite its very low price and ease of construction – did not meet the demands of a tracking device for everyday use. Especially the wearing comfort was not sufficient for continuous utilization.

First prototype
↪ *Page 68*

In order to cope with the restrictions on the first device, a second prototype was built. Instead of being mounted on a glasses frame – which is the common approach in related work – the camera of the second device has been fixated at a conventional headset. This construction allowed for a much higher wearing comfort that was confirmed by the majority of persons who tested the device. In addition to the higher comfort, the camera used for the second prototype also allowed for a much better image quality.

Second prototype
↪ *Page 71*

Outlook

In the previous chapters, one major downside of the second prototype was indicated. The size of the camera module – including the focus mechanics – interfered with the field of view of the user. Additionally, the box that holds the camera often covered one of the four corneal reflections needed for achieving head position invariance. As it was pointed out in *chapter 5*, the tracking algorithms were able to operate at much lower resolutions than the one provided by the actual camera. For future tracking devices, it may be possible to utilize much smaller camera systems like e. g. the one shown in *figure 6.1*. This would allow for much more lightweight tracking devices that do not distract the user. Since the algorithms presented in this thesis are very robust, less costly cameras could be used for building the tracking device, allowing to reduce the total costs of the system even further.

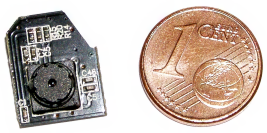


Figure 6.1: Miniaturized VGA-camera module with fixed-focus lens.

Fixed head gaze tracking

In this work, an algorithm chain for determining the point of gaze of the user was described, that assumed the user's head to be fixated. While the basic idea of this algorithm was similar to already existing gaze trackers, some substantial improvements were introduced. For example, the segmentation of the user's pupil was done using a dynamic thresholding technique that allowed for a reliable pupil segmentation across all persons who tested the device. The subpixel accurate pupil center was determined using an ellipse fitting approach. For obtaining the contour points of the pupil needed for this procedure, two original and robust techniques were presented. Both methods had different assets: While the first one allowed for a very fast computation, the second one was more robust to partial occlusions of the pupil – that were caused e. g. by blinking.

Pupil/glint segmentation
↪ *Page 19*

Ellipse fitting
↪ *Page 21 ff.*

Screen space mapping

The results of these algorithms then were used for determining the point of gaze of the user. In order to obtain screen coordinates, a calibration procedure was used to create a function that mapped the pupil glint vectors to the coordinate system of the computer screen. In addition to conventional linear interpolation, a different technique for creating such an interpolation function was presented. This technique utilized so called radial-basis functions. By using a customized kernel-function, the attainable tracking accuracy was always better than the one achieved with linear interpolation. In addition, the new mapping technique also allowed for reducing the number of calibration points while still providing a sufficient overall accuracy.

↪ *Page 29 ff.*

Comparison with related work

The results of this thesis can be compared e. g. with the work of *Parkhurst et al.* [LWP05, WLBP05]. They were able to achieve an average accuracy of about one degree of visual angle using their own hardware construction and algorithms. In this thesis an average accuracy of approximately 0.6 degrees was achieved. However, since the testing procedure of the former device differed from the one used in this work, these results cannot be compared directly. Instead, additional testing of both devices would be feasible for gaining more representative results. It is notable however that the accuracy of the algorithms presented in this thesis was achieved with a device costing around 103 €.

Outlook

The algorithms used for fixed-head tracking achieved stable and accurate results on all probands who participated in the tests. Still – due to lack of time – there are some tests that should be done in future research. For example, the performance of the RBF-interpolation had not been tested against higher order polynomial interpolation schemes. The existing algorithms are easy to adapt to different tracking devices. This allows for e. g. an integration of the system to a head-mounted display for use in a virtual reality scenario. Since no compensation for head movement would be needed in this case, such a combined system would exhibit a high tracking accuracy while inducing only few additional costs. Also, the existing tracker could be extended to utilize an external head-tracking system in order to allow for free head movement within a large working area.

Head position invariant gaze tracking

Cross-ratio gaze estimation
↪ *Page 46*

RBF-network gaze estimation
↪ *Page 50*

The second part of this thesis dealt with the extension of the algorithms summarized above to allow for natural head movement during usage. The idea for achieving head position invariance originated from the work of *Yoo and Chung*. They presented a way to obtain the point of gaze by utilizing four corneal reflections on the user's eye. These reflections originated from infrared LEDs placed at the edges of the computer screen. The method presented in this work also used these reflections. Instead of using a cross-ratio calculation, as proposed by *Yoo and Chung* [YC05], the existing RBF-interpolation approach was extended to allow for a higher dimensional function approximation. As the results shown in *section 5.3.2* indicated, the approximated mapping function provided by the RBF-network allowed for a high accuracy, comparable with the one of the fixed-head tracking tests.

Background calibration
↪ *Page 108*

Unfortunately, a large amount of training-data was needed for achieving this high accuracy, thus forcing the user to undertake a lengthy calibration procedure. In order to circumvent this inconvenience, an automated approach was tested that

assumes that the user looks near the position of the mouse cursor when performing a click action. This way the system is implicitly calibrated without distracting the user. However, the performance of this method was inferior to the explicit calibration.

Comparison with related work

When employing the explicit calibration procedure, the results of the head position invariant algorithms were comparable with the ones presented by *Yoo and Chung*. However, the techniques presented in this work allowed for lower latency: While *Yoo and Chung* stated a framerate of about 15 frames per second (non-optimized), the presented tracking system reached a framerate of at least 38 frames per second. If the video-decoding and format conversion overhead is neglected, framerates up to 128 frames per second were reachable.

Latency tests
↪ *Page 112*

Outlook

Due to a lack of time, the head position invariant tracking system was only tested on one person. Before the system can be used in further applications, additional proband tests would be reasonable. As already stated, the background calibration process did not perform as well as the explicit calibration procedure. However, additional work on this method may allow for better results and therefore for a more user-friendly calibration procedure. In this context, it would be interesting to examine to what extent the RBF-network is able to generalize: If enough initial training-data of different persons were available, the RBF-network may be able to estimate the gaze positions of unknown persons. In this case, a calibration procedure would not be needed at all. In addition to the algorithms presented in this thesis, more sophisticated clustering algorithms may improve the accuracy of the approximated mapping function further. All in all, there is much room for further studies on these highly interesting topics.

List of Figures

List of Tables

Bibliography

[AK96] F. Aurenhammer and R. Klein. *Voronoi diagrams*. Citeseer, 1996.

[AR05] T. Acharya and A. K. Ray. *Image processing: principles and applications*. Wiley-Interscience, 2005.

[Bel61] Richard Bellman. *Adaptive control processes: a guided tour*. Princeton University Press, Princeton, N.J.„ 1961.

[BP94] S. Baluja and D. Pomerleau. Non-intrusive gaze tracking using artificial neural networks. *Advances in Neural Information Processing Systems*, pages 753–753, 1994.

[BP04] J. S. Babcock and J. B. Pelz. Building a lightweight eyetracking headgear. In *Proceedings of the 2004 symposium on Eye tracking research & applications*, pages 109–114, 2004.

[Bro88] D. S. Broomhead. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, 1988.

[Buh03] M. D. Buhmann. *Radial basis functions: theory and implementations*, volume 12. Cambridge Univ Pr, 2003.

[Can86] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[CCL04] F. Chang, C. J. Chen, and C. J. Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93(2):206–220, 2004.

[CM06] F. L. Coutinho and C. H. Morimoto. Free head motion eye gaze tracking using a single camera and multiple light sources. 2006.

[CRM09] N. A. Campbell, J. B. Reece, and J. Markl. *Biologie, 8. aktualisierte Auflage*. Pearson Studium-Verlag, München, 2009.

[Dod03] R. Dodge. Five types of eye movement in the horizontal meridian plane of the field of regard. *American Journal of Physiology–Legacy Content*, 8(4):307, 1903.

[Duc02] A. T. Duchowski. A breadth-first survey of eye-tracking applications. *Behavior Research Methods*, 34(4):455–470, 2002.

[FBH+08] Holger Flatt, Steffen Blume, Sebastian Hesselbarth, Torsten Schünemann, and Peter Pirsch. A parallel hardware architecture for connected component labeling based on fast label merging. In *International Conference on Application-specific Systems, Architectures and Processors, ASAP*. IEEE, 2008.

[FF96] A. W. Fitzgibbon and R. B. Fisher. A buyer's guide to conic fitting. *DAI RESEARCH PAPER*, 1996.

[Fla11] Holger Flatt. *Eine konfigurierbare RISC/Coprozessor-Architektur zur Echtzeitverarbeitung von Objekterkennungsverfahren*. Shaker Verlag, 2011.

[GE06] Elias Daniel Guestrin and Moshe Eizenman. General theory of remote gaze estimation using the pupil center and corneal reflections. *IEEE Trans Biomed Eng*, 53(6):1124–33, 2006.

[GGS94] W. Gander, G. H. Golub, and R. Strebel. Least-squares fitting of circles and ellipses. *BIT Numerical Mathematics*, 34(4):558–578, 1994.

[Hay99] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice hall, 1999.

[HJ10] D. W. Hansen and Q. Ji. In the eye of the beholder: A survey of models for eyes and gaze. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(3):478–500, 2010.

[HL09] Craig Hennessey and Peter Lawrence. Noncontact binocular eye-gaze tracking for point-of-gaze estimation in three dimensions. *IEEE Trans Biomed Eng*, 56(3):790–9, 2009.

[HWJM+89] T. E. Hutchinson, K. P. White Jr, W. N. Martin, K. C. Reichert, and L. A. Frey. Human-computer interaction using eye-gaze input. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(6):1527–1534, 1989.

[Jac90] R. J. K. Jacob. What you look at is what you get: eye movement-based interaction techniques. In *Proceedings of the SIGCHI conference*

on Human factors in computing systems: Empowering people, pages 11–18, 1990.

[KB03] R. Kimmel and A. M. Bruckstein. Regularized laplacian zero crossings as optimal edge integrators. *International Journal of Computer Vision*, 53(3):225–243, 2003.

[Kri07] David Kriesel. *Ein kleiner Überblick über Neuronale Netze*. 2007. erhältlich auf http://www.dkriesel.com.

[LWP05] D. Li, D. Winfield, and D. J. Parkhurst. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches (pdf). 2005.

[LZSC04] Y. Liu, Q. Zheng, Z. Shi, and J. Chen. Training radial basis function networks with particle swarms. *Advances in Neural Networks–ISNN 2004*, pages 317–322, 2004.

[Mac02] David J. C. Mackay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, June 2002.

[Mic86] C. A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2(1):11–22, 1986.

[MM05] C. H. Morimoto and M. R. M. Mimica. Eye gaze tracking techniques for interactive applications. *Computer Vision and Image Understanding*, 98(1):4–24, 2005.

[NWKF02] K. Nguyen, C. Wagner, D. Koons, and M. Flickner. Differences in the infrared bright pupil response of human eyes. In *Proceedings of the 2002 symposium on Eye tracking research & applications*, pages 133–138, 2002.

[Pac] Hewlett Packard. Hdsl-4230 technical data sheet.

[PCG+03] A. Perez, M. L. Cordoba, A. Garcia, R. Mendez, M. L. Munoz, J. L. Pedraza, and F. Sanchez. A precise eye-gaze detection and tracking system. In *11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2003.

[Pen56] R. Penrose. On best approximate solutions of linear matrix equations. In *Proc. Cambridge Philos. Soc*, volume 52, pages 17–19, 1956.

[PS91] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

[Ray98] K. Rayner. Eye movements in reading and information processing: 20 years of research. *Psychological bulletin*, 124(3):372, 1998.

[RDD08] N. Ramanauskas, G. Daunys, and D. Dervinis. Investigation of calibration techniques in video based eye tracking system. *Computers Helping People with Special Needs*, pages 1208–1215, 2008.

[Rob63] D. A. Robinson. A method of measuring eye movemnent using a scieral search coil in a magnetic field. *Bio-medical Electronics, IEEE Transactions on*, 10(4):137–145, 1963.

[Ros99] P. L. Rosin. Further five-point fit ellipse fitting. *Graphical Models and Image Processing*, 61(5):245–259, 1999.

[Row11] Todd Rowland. "bump function". From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein, 11 2011. `http://mathworld.wolfram.com/BumpFunction.html`.

[S+85] S. Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[Sch00] H. Scharr. *Optimale Operatoren in der digitalen Bildverarbeitung*. PhD thesis, Universitätsbibliothek, 2000.

[SGE05] R. Schatten, N. Goerke, and R. Eckmiller. Regional and online learnable fields. *Pattern Recognition and Image Analysis*, pages 74–83, 2005.

[SL04] S. W. Shih and J. Liu. A novel approach to 3-d gaze tracking using stereo cameras. *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, 34(1):234–245, 2004.

[SMSK08] Y. Sugano, Y. Matsushita, Y. Sato, and H. Koike. An incremental learning method for unconstrained gaze estimation. *Computer Vision–ECCV 2008*, pages 656–667, 2008.

[Spe08] E. J. Speckmann. *Physiologie: Mit Zugang zum Elsevier-Portal*. Elsevier, Urban & FischerVerlag, 2008.

[SWL00] S. W. Shih, Y. T. Wu, and J. Liu. A calibration-free gaze tracking technique. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 4, pages 201–204, 2000.

[WLBP05] D. Winfield, D. Li, J. Babcock, and D. J. Parkhurst. Towards an open-hardware open-software toolkit for robust low-cost eye tracking in hci applications. 2005.

[XMS98] L. Q. Xu, D. Machin, and P. Sheppard. A novel approach to real-time non-intrusive gaze finding. In *British Machine Vision Conference*, pages 428–437, 1998.

[YC05] D. H. Yoo and M. J. Chung. A novel non-intrusive eye gaze estimation using cross-ratio under large head motion. *Computer Vision and Image Understanding*, 98(1):25–51, 2005.

[ZJ04] Z. Zhu and Q. Ji. Eye and gaze tracking for interactive graphic display. *Machine Vision and Applications*, 15(3):139–148, 2004.