RWTH AACHEN UNIVERSITY

Calibration-Free Gaze Tracking

an experimental analysis

Diploma Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Maximilian Möllers

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Dr. Roel Vertegaal

Registration date: March 12th, 2007
Submission date: November 6th, 2007

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Aachen November 5, 2007

Contents

List of Figures

List of Tables

Abstract

Gaze tracking systems are widely used for research in psychology, neurology, and human-computer interactions. A calibration-free system with sufficient accuracy for post-desktop interactions has not yet been built. To explore interactions beyond "pointing through gaze" or "gaze-enhanced video conferencing" the system must meet the following requirements:

- allow free head movement
- accurate tracking
- real time processing of input images
- free of user calibration

To explore why current systems do not fulfill these requirements, we developed our own gaze tracking system and thoroughly evaluated its performance.
We based our work on the proposal by Shih et al. [2000] and enhanced their approach using least square error methods and multiple cameras. We implemented this approach and built our own gaze tracking system to reflect the state of current gaze tracking research. To evaluate the system's behavior we fed synthetical errors (e.g., add noise to the images) into the system and measured its fault-tolerancy.
We calculated the ratio between each error input and its resulting error. This way we were able to quantitatively predict the influence of an error and to sort the error sources by their damage potential. We discovered that especially the extrinsic parameters of the cameras (i.e., position and rotation) have to be measured very precisely or the system will always produce errors, like every gaze tracking system does up to now — even after being calibrated to the user.
To compensate for errors resulting from imperfect system calibration, we designed and implemented an adaptive online calibration algorithm. This approach was tested and it was shown to be a promising way forward. Although it was not able to protect the system from all error sources, it was able to vastly reduce the impact of intrinsic calibration errors and imprecise camera positions.

Überblick

Gazetracker werden in der Psychologie, Neurologie und im Bereich der Mensch-Maschine-Interaktion zur Forschung eingesetzt. Heutige Systeme erfordern eine Anpassung des Systems an den Nutzer um zufriedenstellende Genauigkeit liefern zu können. Bei vielen Einsatzgebieten ist es jedoch nicht möglich oder zumutbar, erst eine Benutzeranpassung durchzuführen (z.B. bei spontanen Geschäftsmeetings). Ein System, das die Blickrichtung des Benutzers nicht nur als Ersatz für Eingabegeräte oder zur Verbesserung von Videokonferenzen nutzt, muss folgende Anforderungen erfüllen:

- Der Kopf darf sich frei bewegen.
- Die Blickrichtung wird exakt ermittelt.
- Das System arbeitet in Echtzeit.
- Das System benötigt keine Anpassung an den Benutzer.

Um herauszufinden warum die aktuellen Systeme diese Anforderung nicht erfüllen, haben wir unseren eigenen Gazetracker entwickelt und dessen Leistungsverhalten genau analysisert.
Der Vorschlag von Shih et al. [2000] wurde mittels der Methode der kleinsten Fehlerquadrate verbessert und unterstützt jetzt beliebig viele Kameras. Das Verfahren wurde implementiert und ein eigener Gazetracker entwickelt, welcher aufgrund der verwendeten Bildverarbeitungsroutinen und mathematischen Modelle ähnlich zu gängigen Systemen ist. Unsere Resultate sollten sich somit auf andere Systeme übertragen lassen. Um das Verhalten unseres Trackers zu analysieren, wurden die Rahmenparamter variiert, Eingabedaten künstlich verfälscht (z.B. Verrauschen der Kamerabilder) und die Abweichungen in der Blickrichtugn des Benutzers gemessen.

Das Verhältnis zwischen synthetischem Eingabe- und daraus resultierendem Ausgabefehler wurde als Kennzahl ermittelt. Wir waren somit in der Lage die Auswirkung einer Fehlerquelle quantitativ vorherzusagen und die Fehlerquellen nach ihrem Schadenspotential zu sortieren. Die Resultate ergaben, dass insbesondere die extrinsischen Kameraparameter (d.h. Position und Orientierung) sehr genau gemessen werden müssen. Dies erklärt, warum bei allen bisherigen Systemen die berechnete Blickrichtung nicht fehlerfrei war.
Um diese Fehler auszugleichen, haben wir einen adaptiven Kalibrierungsalgorithmus entworfen und implementiert. Unsere Tests wurden erneut durchgeführt, und es wurde gezeigt, dass dies ein Schritt in die richtige Richtung ist. Auch wenn nicht alle Fehler eliminiert werden konnten, war der Algorithmus in der Lage die Einflüsse von intrinsischen Kamerakalibrierungsfehler und falsch gemessenen Kamerapositionen deutlich zu reduzieren.

Acknowledgements

First of all, I want to thank Prof. Dr. Jan Borchers for his great lectures, which were a fresh breeze in the otherwise dry computer science studies. Working on this thesis was made especially enjoyable by the encouraging and friendly atmosphere and people at the Media Computing Group.
I especially want to thank David Holman and Thorsten Karrer for encouraging me to pursue this topic. They both provided valueable feedback and helped me throughout the development of the system and its evaluation. Fresh Canadian thoughts and German thoroughness are a good combination after all.
I want to thank Dr. Roel Vertegaal for supplying us with the latest Xuuk cameras and inspiring David and me to research gaze-based systems.
I want to thank all of the numerous reviewers of my thesis including René Boulnois, Christian Brüffer, Jonathan Diehl, Andreas Ganser, Christian Mattar, and once again David Holman and Thorsten Karrer. A special thank goes to Sarah Mennicken as my graphics expert and as an amusing roommate. Speaking of the room, I want to thank Milton Waddams for leaving his room to me and moving his desk one more time to another floor.
Aside from study-related help, I want to thank my family and friends. They were there to pick me up when things went wrong, and made the last five challenging years a great time.
Thank you all!

Conventions

Throughout this thesis we use the following conventions.

Definitions are set off in orange boxes and have a marginal note attached.

2:
The cardinal number equal to the sum of 1 and 1.

Definition:
2

Theorems are set off in blue boxes,

Theorem 0.0.1. *The equation $x^n + y^n = z^n$ has no solutions in positive integers for n greater than 2.*

and are followed by a proof.

Proof. For this, I have found a truly wonderful proof, but the margin is too small to contain it. □

Source code and implementation symbols are written in typewriter-style text.

```
myClass
```

The whole thesis is written in American English.

Chapter 1

Introduction

Computer science has been around for several decades and most research focused on theoretical models, efficent algorithms, and effective ways to develop software products. Based on these results computers became faster, usable by non experts, and are, today, part of everyone's life.
Since the 90's another field of research has become more popular: human-computer interaction. It is concerned with the design, evaluation, and implementation of interactive systems for human use and studies the major phenomena surrounding them. It simplifies computer systems for everyday use.

An important information for human interaction is the user's viewing direction, or "gaze". Prior to most interactions, humans tend to look at things of interest. Especially in multiparty human-human interaction, the interaction partners will look at each other to show their attentiveness, and thereby simplifying the communication. Although gaze tracking is an easy task for a human being, it is not easy for a machine.

Current research uses the viewing direction of the user to enhance human-computer interaction. Typical uses of gaze are: as substitute for other pointing devices like a mouse, to give gaze cues in video conference systems, or to de-

tect eye contact towards real world objects (see Chapter 3—"Related Work").
Since most of the gaze tracking systems need to be calibrated for each specific user, the application of gaze tracking to other interesting research fields is hampered. These research fields include gaze-enhanced systems, which could support community and social activities at public places.
Even with user calibration, systems are still not able to replace the mouse for interaction tasks like text selection.

Another example of a gaze-enhanced system is a gaze augmented tabletop PC or conference room (see Figure 1.1).

Figure 1.1: The system is aware of the user's gaze direction and can react on it

Possible use of this information includes:

- *Text Orientation*
  Tabletops are researched as a frequent meeting point for group collaboration. However, in a multi-user tabletop scenario, displaying information that is properly oriented for each user is challenging.

Rotation-sensitive components, such as menus and text, are problematic for horizontal displays. To address this, text can be automatically oriented on the angle that is most readable to the user [Rekimoto and Saitoh, 1999]. In the case of multiple users looking at the same document a "best" angle for the whole set can be found.

- *Generating Video Metadata*
  If the system knows that user A's gaze is directed to user B's eyes, it is clear that when he is speaking he addresses his talk to user B. Knowing who is talking to whom makes it easy to add this information to a video take of this scene, enhancing the browseability for later review.

For such a system, the need to calibrate the system to each user definitively reduces its application range.

We are interested in user calibration-free gaze tracking systems that are able to support these kinds of interactions. The system must also have a good accuracy to reliably differentiate between real world objects. We attempt to find the reasons for the absence of such a system. Knowing the factors that degrade the performance of a gaze tracking system and being able to minimize their influences, future researchers will be able to explore new interactions.

1.1 Overview

In Chapter 2—"Fundamentals", we will show how the eye works and see its similarities to the camera model we use in our calculations. We also show a simple tracking system to present typical problems.
In the first part of Chapter 3—"Related Work", we show severalsystems that have been developed in the last 30 years to track the user's viewing direction and use this information to control a computer system in particular ways. The second part is used to present the interactions

The engineering problem has been researched intensively, but is not solved yet

they were able to support.

We then present our own gaze tracking system which is an extension to the work of Shih et al. [2000]. The theoretical model and the hardware setup is shown in Chapter 4—"System Design and Implementation".
In Chapter 5—"Evaluation" we thoroughly analyze the performance of the system. Synthetical errors are fed into the system (e.g., by adding noise to the images) and its fault-tolerance is measured.
This way we can quantitatively predict what kind of input error influences the system most. We will attempt to enhance our system to cope with these error sources.
Chapter 6—"Summary and Future Work" closes this thesis with a summary of our work and an outlook on future research that will be possible based on our contributions.

Chapter 2

Fundamentals

For visual contact, each of our eyes has to properly focus at an object of interest. The light travels from the object through the eye's lens-system into the eye. Since the human's field of sharp view is very narrow (6° [Rayner, 2000]), people are able to recognize where other people are looking.

2.1 The Human Visual System

Only people with healthy eyes are well-trackable

In order to track the user's viewing direction, it is necessary to have working knowledge of the human eye. Since we will often refer to parts of the human eye, we now introduce its most relevant parts and their processing steps to generate an image in our mind. Afterwards we present a very simple eye tracking system, to demonstrate how this information can be used.
For our work we (as every other researcher) focus on humans with two healthy eyes, because even a human cannot find out the viewing direction, e.g., of a squint-eyed test subject. For people with strong cornea curvature, gaze tracking systems usually perform badly, although the system should be able to at least give a good hint on the viewing direction.

The human eye functions very similar to a camera (see Figure 2.1). The light is bundled by the cornea and the lens onto the retina. Photoreactive cells on the retina generate an electric signal which is transported along the optic nerve to the brain where an image is generated.

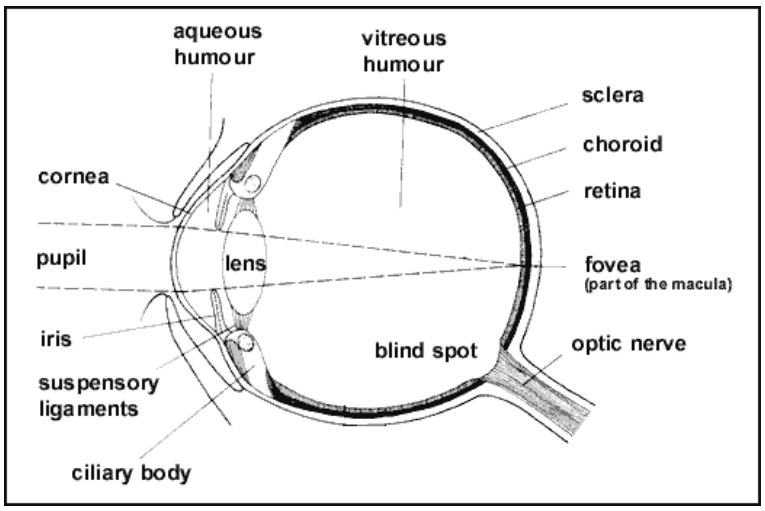

Figure 2.1: Structure of the human eye, ©Editure 2007

2.1.1 The Lense System

Cornea and lens bend the light

The refraction of the incoming light is done in two steps: the major refraction comes from the cornea, a transparent refractive layer that covers the front of the eye.
Together with the crystalline lens, the cornea works to focus incoming light onto the retina. The cornea performs most of the refraction of incoming light, but has no means of adjusting its curvature, so the amount of refraction performed by the cornea is fixed.

The lens sharpens the image

The crystalline lens resides behind the pupil and refracts light onto the retina. The crystalline lens is flexible and its curvature is controlled by ciliary muscles. By changing the curvature of the lens, the eye can change its focus among objects at different distances. In this way, the cornea can be thought of as the "coarse" lens and the crystalline lens can

be thought of as "fine". To see an object clearly, the eyeball has to be rotated so that the incoming light is refracted onto the fovea spot — the most sensitive area of the retina.

2.1.2 The Pupil and the Iris

To compensate for different lighting situations, the amount of light that reaches the retina can be controlled. The transparent opening of the eye is called "pupil" and can usually be seen as a black circle. The iris is the set of muscles around the pupil. By flexing und relaxing the iris, the size of the pupil and therefore the amount of light that reaches the retina, is changed.

2.2 A Simple Eye Tracking System

Fur further discussion, we define gaze as follows.

GAZE:
The line from the cornea center through the pupil center onto an object of interest.

Definition: *Gaze*

This is similar to the optic axis.
We show a simple eye tracking system to demonstrate typical challenges and solutions. Although it is able to reconstruct the point where the user is looking on the screen, this is not a gaze tracking system — it does not calculate the gaze direction at all.
The system setup consists of a user sitting in front of a display and letting a camera take images of his eyes. The camera is mounted on top of the display and takes images like Figure 2.2 or 2.3.

Images are taken from a camera, typically mounted on top of the user's display

As we can see in the first image, the user is looking straight onto the middle of the display. In the second one, he is looking to the left, so he must be gazing at some point in the left area of the display.
To track the user's gaze one could take five images of the eye, with a user looking at each corner and the middle of

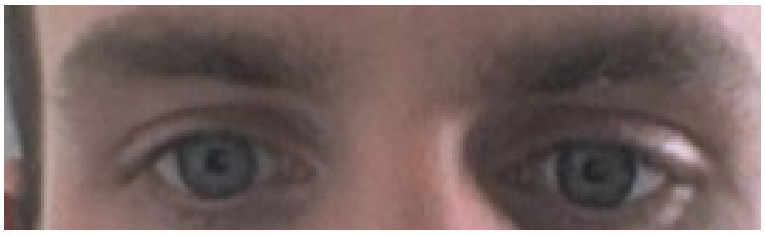

Figure 2.2: User looking straight onto the middle of the display

Figure 2.3: User looking straight onto the left side of the display

the display. With this calibration, we can input arbitrary images of the user and compare the input with our five sample images. The best fitting image is chosen and the position is returned (i.e., one of the corners or the middle). We now have a simple eye tracking system that can differentiate between five different regions of the display.

Sophisticated image-processing allows more detailed output

Unfortunately, this is not accurate enough to, e.g., substitute a pointing device like the mouse with eye-based input. For higher accuracy, more intelligent and complex hardware setups and methods are required. A possible extension of our simple idea would be to use sophisticated image-processing to not only find one best fitting image, but to get statements like:
"The current image looks 50 percent like the image with the user looking at top-left corner and 50 percent like the image with the user looking at lower-left corner."
This statement suggests that the user is looking at a spot in the left half and middle height of the display. This extension would significantly improve the system, since it is now possible to return an arbitrary point on the display for

a given camera image — the actual accuracy of such an approach is another topic. A drawback, which is common to other systems is that the system depends on a calibration step (i.e., the five images) before usage, which has to be done for each user.

Chapter 3

Related Work

"The common fly has eight legs."
–Aristotle

Eye tracking has been studied from two points of view. The first is a more psychologically and physiologically focused view:
How is our eye working? How is information recorded by the eye and then processed by the brain?
Here eye tracking was the technology to help find results like the discovery of saccades — very fast and quick movements of both eyes which "step" through information like text. The second view is more interaction oriented:
How can we use the eyes as an input device for some kind of computer system?
To give a broad overview of the recent studies in the interaction area, we will differentiate the systems the researchers used by their capabilities and the necessary preparations for eye tracking. More information about psychological, physiological, and other aspects can be found in the extensive survey by Duchowski [2002a].

3.1 Eye Tracker

During the general research of eye tracking, the term has come to mean several different things:

- 2D eye tracking systems, that map 2D images of the pupil via heavily user- and system-dependent mapping to the 2D screen position, the user is looking at.
- Eye contact sensors, that are only able to detect, whether a user is looking at them.
- 3D eye tracking systems, that use monocular information to calculate the eye's position at first, and then the 2D pixel position.
- 3D gaze tracking systems, that are able to calculate the 3D point both eyes are focusing on.

The items are ordered by increasing difficulty and decreasing age.
Fur further discussion, every system that tracks the user's eye is an eye tracker. If an eye tracker calculates the viewing direction of the user, we will call it gaze tracker or gaze tracking system.

Early eye trackers used invasive techniques

Duchowski [2002b] gives a good overview of the older eye tracking systems. A major design-decision for an eye tracker is the invasiveness of the system. The first eye trackers were simple electrodes underneath the skin next to the eye. They detected the muscles' movement and the attached system could calculate the gaze of the eye relative to the user's head.
Another invasive technique is to place a contact lens with a magnetic coil on the user's cornea. As soon as the user moves the eye, the magnetic coil will induce a current into the magnetic field which is produced around the eye by other coils. Measuring the current will yield the desired eye movement.

Current eye trackers use vision-based techniques

Most of the newer eye tracking systems are much less invasive. They use vision-based techniques to track the user's eye, and are mounted to either the user's head, the desktop, or somewhere else in the room. They typically use two

pieces of information. The reflection of a static light (e.g., a LED) on the cornea and the position of the pupil. Because the surface of the cornea is a more or less sphere-shaped object the movement of the eye will not alter the reflection spot of the light source. This way the system can determine the position of the eye. With this information the system can then use the position of the pupil to find the direction in which the user is looking.
The ultimate goal of eye tracking is to pinpoint in real-time the three-dimensional point at which both eyes are focussing, using a calibration-free non-invasive system. This goal has not yet been reached under realistic circumstances, but the research community is making step-by-step progress. We discuss common systems and the trade-offs they chose to give good results for the proposed interaction task.

3.1.1 Head-mounted

Although we are focussing our research in the area of gaze tracking, there are some revelant papers in peripheral areas.
ViewPointer, introduced by Smith et al. [2005], is a wearable eye contact sensor that detects the orientation of the head towards small embedded devices attached to real world objects. The system consists of a headset with a small camera that monitors the eye, see Figure 3.1.

ViewPointer detects eye contact to real word objects

The real world objects have IR emitters attached, that send uniquely identifiable binary encoded tags through blinks into the scene (see Figure 3.2). As soon as the camera detects these blinks as reflections on the eye it can calculate the tag's ID. If these reflections appear on the center of the cornea (i.e., the pupil), the user must be looking at that specific tag. Another benefit of the IR tags is that they are inexpensive and easy to attach to everyday objects and thereby augmenting the world with contextual information. It is also possible to send meta data in addition to the ID, but due to prototypical bandwidth limitations (the framerate of the camera limits the number of blink identifications per second) even small chunks take several seconds.

Starburst accurately detects the pupil's position

Another head-mounted eye tracker was built by Li et al.

Figure 3.1: The ViewPointer headset, ©ACM 2005

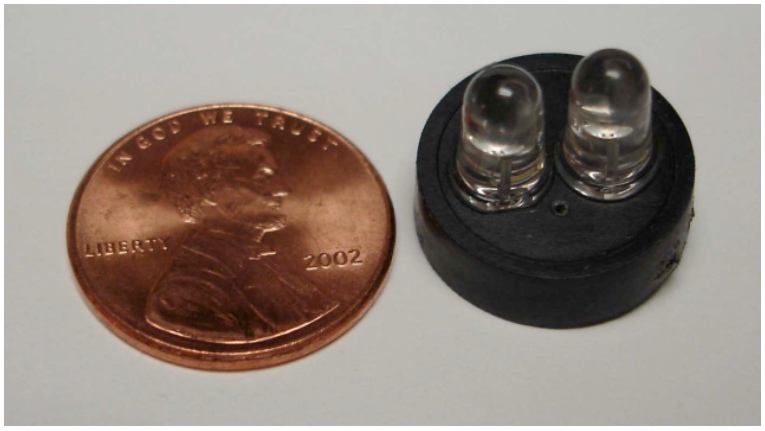

Figure 3.2: A ViewPointer tag compared to an US penny, ©ACM 2005

[2005]. It consists of two cheap CCD cameras that are mounted on a pair of safety glasses. The underlying algorithm achieves a good trade-off between run-time performance and accuracy. This means an approximately 1° error of visual angle and real-time performance using general purpose hardware affordable at that time. The algorithm attempts to find the corneal reflection by adaptive thresholding and removes it from the image. It then detects the center of the pupil by starting with an initial guess of the pupil center (Figure 3.3). It shoots rays in every direc-

tion and searches for the pupil's edges (clearly identifiable as black-to-gray gradients) — hence the algorithm's name "Starburst".

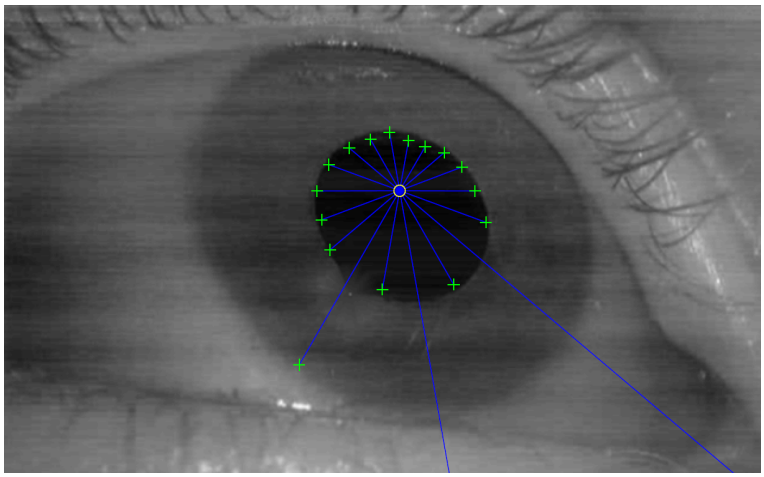

Figure 3.3: Feature detection using an initial guess with the Starburst algorithm, ©IEEE 2005

Iteratively fitting an ellipse to the detected edge points finally yields the exact shape of the pupil and its center (Figure 3.4. The point of gaze can than be calculated using a linear mapping from the exact 2D position of the pupil to the screen coordinate the point of gaze is calculatead. This mapping has to be calculated before system use and this relationship is measured with a 3×3 grid of calibration points.

3.1.2 Remote

There also exist eye contact sensors for remote eye tracking. One example is the EyePliance system by Shell et al. [2003]. It is a set of appliances and devices that detect and respond to human visual attention. Knowing whether the user is focusing on the device or not can reduce the explicit user input. They proposed an attentive TV that detects when the user is standing up from the couch to get something to drink from the refrigerator. As soon as he turns his face away from the TV, the movie stops and will resume as soon

The EyePliance's awareness of the user's eye contact reduces explicit user input

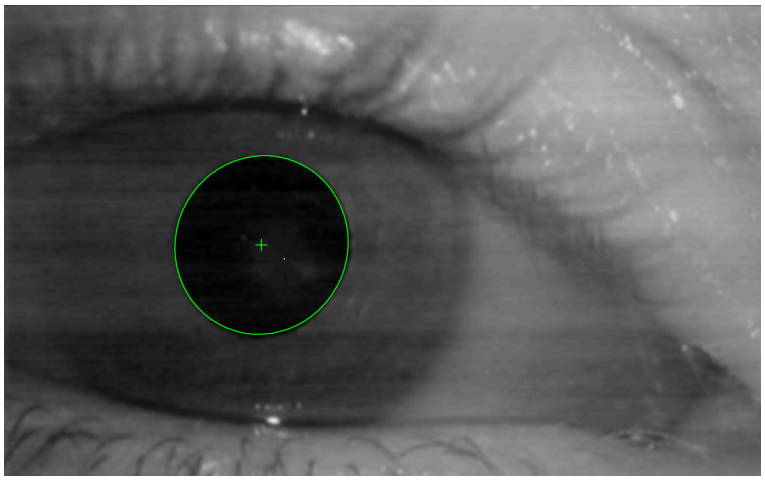

Figure 3.4: Model-based ellipse after fitting, ©IEEE 2005

as the user is back on the couch (Figure 3.5).
With the EyePliance system there is less much context needed for specific speech recognition tasks. For example, looking at a device and saying "on" or "off" is sufficient enough to activate the correct device, whereas with traditional speech recognition the user would have to name the device first, and then give the order to switch it on or off[1].

Mapping the camera image to a complex 3D model of the head yields the head's 3D pose

A system that is capable of tracking the 3D pose of the user's head was developed by Stiefelhagen et al. [1997]. The colored input image is searched for pixels with face color, and the largest connected region of face-colored pixels is considered as the face region. Afterwards, the search for the facial features begins with an iterative thresholding (starting with a low threshold) for the pupils, which are the darkest spots in the image. As a first filter, geometrical constraints are applied: valid candidate pairs must be in proximity to each other at approximately the same height in the picture. Using geometric constraints, (minimum and maximum distance of the dark spots, etc.) the candidate spots in the thresholded image are mapped to the pupils.
The next facial feature are the lip corners. Using a model of the head and the just acquired positions of the eye, a region-of-interest is defined for the corner search. A row-wise his-

[1] More research on "Look-to-interact" in Section 3.2—"Interaction-techniques and Applications"

Figure 3.5: Attentive TV reacting to the user facing away, ©ACM 2003

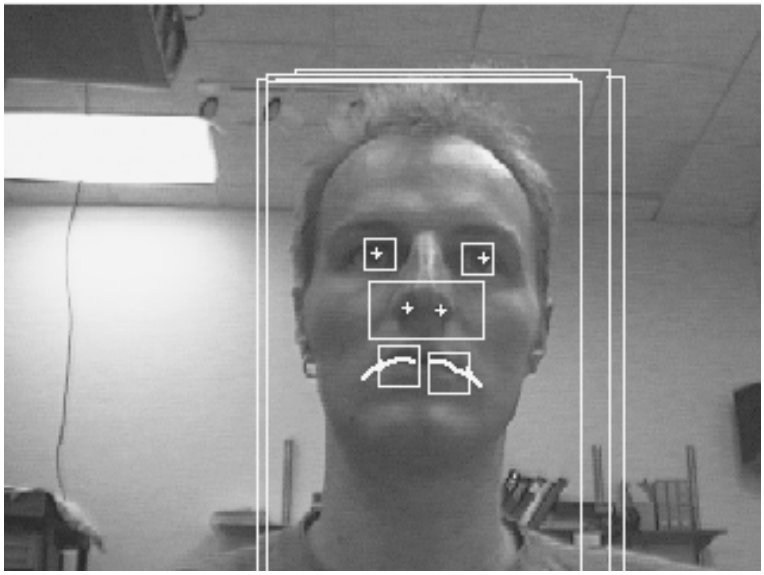

Figure 3.6: Input image after tracking of the facial features, ©ACM 1997

togram is built to find the location of the lips, which resemble a dark line. Afterwards, an edge operator is applied to a smaller area around the estimated line position to find the left and right boundaries of the lips. Similar to detecting the eyes, the nostrils can be found by the iterative thresholding algorithm already used for the pupils. An additional geometric constraint is their expected position below the eyes and above the lips.
To track the user's head position, the system uses small search windows around the last feature position. These search windows are predicted using linear extrapolation over the two last frames. If the tracking fails, the system resets to search mode and detects the features again.
The system was capable to calculate a correct head orientation with a negligible error of 1° rotation around the z-axis (which goes from top to bottom) and an error of 5° along the other two axis at 15 frames per second. However, it did not calculate the actual gaze direction of the eyes at all.

The latest gaze tracking systems use feature-based methods to avoid complex models of the user, but just search for simple features in the image.

Gaze can be tracked exploiting the different brightness of sclera and iris

A simple eye location detection algorithm for unconstrained greyscale images is the work of Kothari and Mitchell [1996]. Figure 3.7 shows the gradients of the grayscale image. The gradients point from dark regions to bright regions. As can be seen in the figure the gradients point outwards to the edges of the iris' ellipsoid shape. The algorithm extrapolates the field in a direction opposite to the gradient (the gradient points in the direction of increase of a function, hence it points outwards — from the darker iris to the lighter sclera). A 2D array of bins serves as an accumulator. A line is drawn at each point along the direction opposite to the gradient. This line passes several bins and if a bin is hit its accumulator is increased by one. Bins with high accumulators have lots of lines passing them, so the center of the iris will have a very high amount of hits. The algorithm uses these bins as candidates for the iris. As a first filter, geometrical constraints are applied: valid candidate pairs must be in proximity to each other at approximately the same height in the picture. The final step uses temporal information by adding a (less than 1) weighted array of the last frame(s) to the current array. This results in a

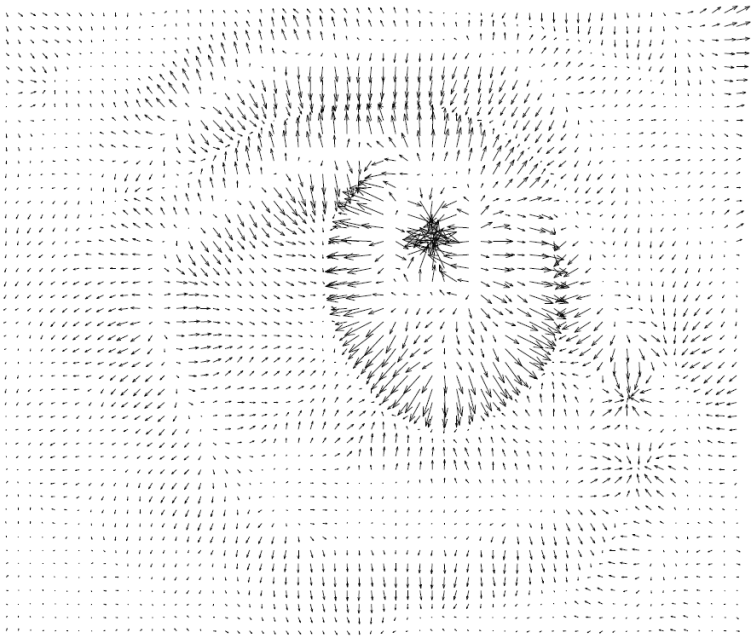

Figure 3.7: Gradient directions around the pupil, ©IEEE 1996

robust detection of (2D) eye locations over a wide range of subjects.

Another trackable feature is a specular reflection on the cornea from a light source, typically a LED (Figure 3.8). Because of the spherical shape of the cornea, the position of the reflections, or "glints", stay the same even if the user moves his eye. Common algorithms use triangulation on this static glint position to calculate the 3D position of the eye.

Current gaze tracking systems use reflections of light sources to triangulate the eye

An idea, that uses an alternating lighting scheme was developed by Ebisawa [1995]. If a human pupil is lit with a near-infrared light source coaxial to the camera, then the light is reflected off the retina and escapes back through the pupil — an effect known to most photographers as the "red-eye" effect. By switching between an uncoaxial and coaxial light source the camera sees two images except, that only differ in a red pupil in the odd and a dark pupil in the

The "red-eye" effect is an outstanding feature to track the pupil

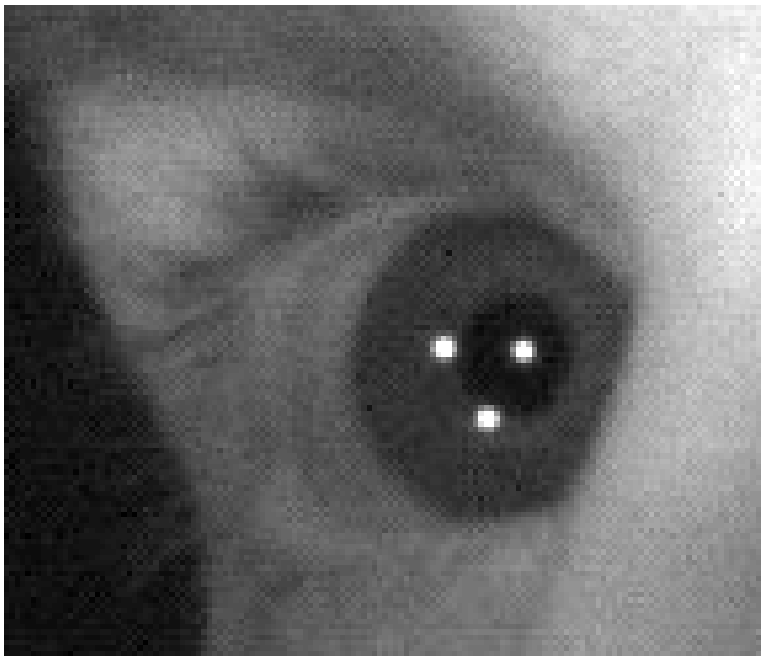

Figure 3.8: Cornea illuminated by three LEDs

even frame. Subtracting these images yields a black image with a white spot — the pupil (Figure 3.9). Using simple thresholding, the system can filter out the noise in the images to find the pupil area.

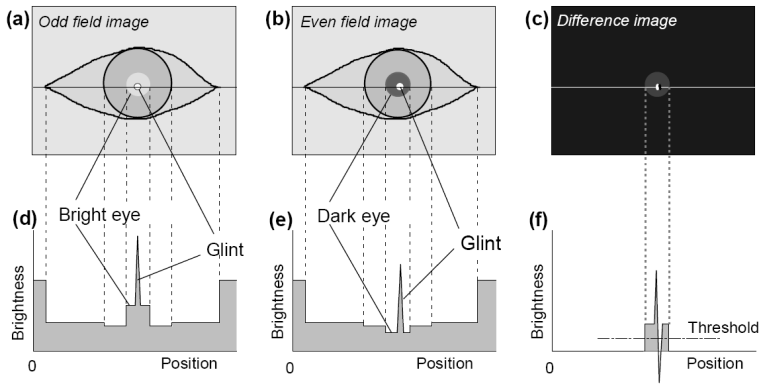

Figure 3.9: Coaxial and uncoaxial illumination and the resulting difference image, ©WIT Press 1995

The system is able to track multiple pupils at the same time using the "red-eye" effect

Based on this differencing technique, Morimoto et al. [2000] built a pupil detection and tracking system. One major improvement was the placement of the two LEDs as rings around the camera. This way the glint generated from the outer ring (for the dark pupil image) is at the very same position as the glint generated from the inner ring (for the bright pupil image), thus increasing the quality of the difference image.
To compensate for motion artifacts due to users moving their heads, the pupils are detected from the differences between all consecutive pairs of dark and bright images and not only from one pair. Most motion artifacts only appear between some of the frames and will vanish in the overall average difference frame.
As soon as the pupils are reliably detected, they are tracked by applying thresholding and blob detection to the dark and bright image. This leads to better results than tracking with the difference image, because the difference image can include motion artifacts that blur the appearance of the pupil. The system was able to reliably track several pupils

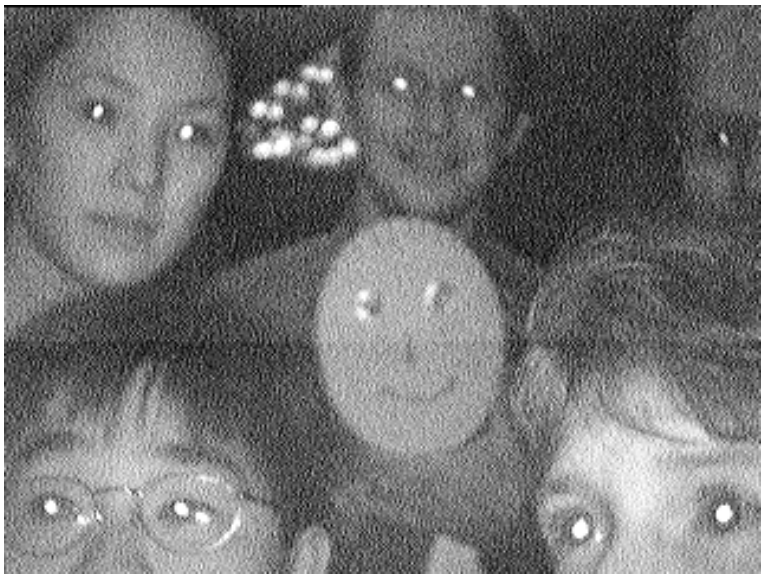

Figure 3.10: Multiple pupil tracking, including people wearing glasses, ©IEEE 2000

at once at 15 frames per second using a Pentium with 200 Mhz (Figure 3.10).

Two cameras and two light sources are needed for the first calibration-free gaze tracking algorithm

The first real gaze tracking algorithm that does not need a user-specific calibration was proposed by Shih et al. [2000]. The method employs multiple cameras and light sources to determine cornea center and gaze direction. The authors concluded that two light sources and two cameras are needed for their model. They ran a computer simulation on this model (by generating computer images and adding noise to them) and yielded good results regarding the position of the cornea. The gaze direction itself was quite sensitive to noise. This algorithm formed the basis of our own work. For a detailed description, see Section 4—"System Design and Implementation".

A new eyeball model leads to less than 1° average error angle

Next in the evolution of gaze tracking systems was the FreeGaze system by Ohno et al. [2002]. The physical setup consists of a narrow angle camera mounted below the user's desktop at roughly 60 cm distance to the user and an infrared light source to generate a glint, which is invariant to rotation of the eye.
They introduced a new eyeball-model and sophisticated image processing to get quite accurate data after per-user calibration has been done. The common performance measurement for gaze tracking systems is the angle between the actual gaze direction and the calculated gaze direction. It is usually given in degree or radian — for this system it was less than 1° on average.
The system searches for the eye in the camera image with a connected component analysis. For each region, its boundary shape is calculated to detect pupil candidates. The neighboring pixel values are used to distinguish between pupil and other circular objects. If a good candidate is found, the system does a model-based ellipse fitting to perfectly match the shape of the pupil.

User calibration is required

This ellipse is mapped onto the eyeball model, which includes the user-specific radius of cornea curvature and the user-specific distance between the pupil and the center of cornea curvature. This allows to compensate for the refraction of the pupil image due to the curvature of the cornea, which lies between the camera and the pupil.

Narrow angle cameras do not allow much head movement

Fortunately, the system only needs the user to look at two points before he can start using it, which is an acceptable trade-off for the suggested interaction task of a user sitting in front of a desktop. The initial system did not allow the

user to move their heads more than in a 4 cm^2 area or they would be out of sight of the narrow angle camera.

Free head movement possible, but still per user calibration required

This was improved with the follow-up system two years later [Ohno and Mukawa, 2004]. The researchers combined the old system with wide angle stereo cameras, which were mounted on top of the user's monitor, tracking the user's face. The gaze tracking camera was mounted on a pan-tilt stand. When the stereo camera finds the user's pupil the gaze tracking camera is directed to the eye. They showed that the system was still accurate enough for operation (less than 1° of error angle) but now had a wider application range, because the user is allowed to freely move his head. However, for interactions like selection of text accuracy is still lacking. In addition, the need for user calibration is a problem in various situations (gaze tracking at public places, spontaneous meetings in gaze-enhanced conference rooms, etc.).

Replacing the pan-tilt stands with rotating mirrors in front of the narrow cameras results in a more accurate gaze tracking system

Beymer and Flickner [2003] developed a similar system which also uses a pair of wide angle stereo cameras to track the face and two narrow angle cameras for the gaze tracking (Figure 3.11). They also have a complex 3D model of the eye, but they use a (more realistic) ellipsoid for the corneal ball. Due to the narrow field of view, quick head motions would outpace the pan-tilt heads. For this reason, pan and tilt of the narrow angle cameras are controlled using rotating mirrors on high performance galvos, that can rotate and settle in 2 ms. The motion of the mirrors is synchronized to the frame rate of the camera, so their motion does not cause image blur.

First mention of the need for good system calibration

Due to the multitude of cameras with their own coordinate systems, there is a need for a good calibration algorithm. The system uses multiple views of a planar calibration target with a checkerboard pattern on it. Because of the narrow field of view, the checkerboard is not fully visible, leading to a correspondence problem. To address this issue, the authors added a 2D dot code to map each square in the image to its 3D equivalent (Figure 3.12).

Like the camera calibration algorithm of Zhang [2000], their systems takes several images from different viewpoints of the scene. It then minimizes the reprojection

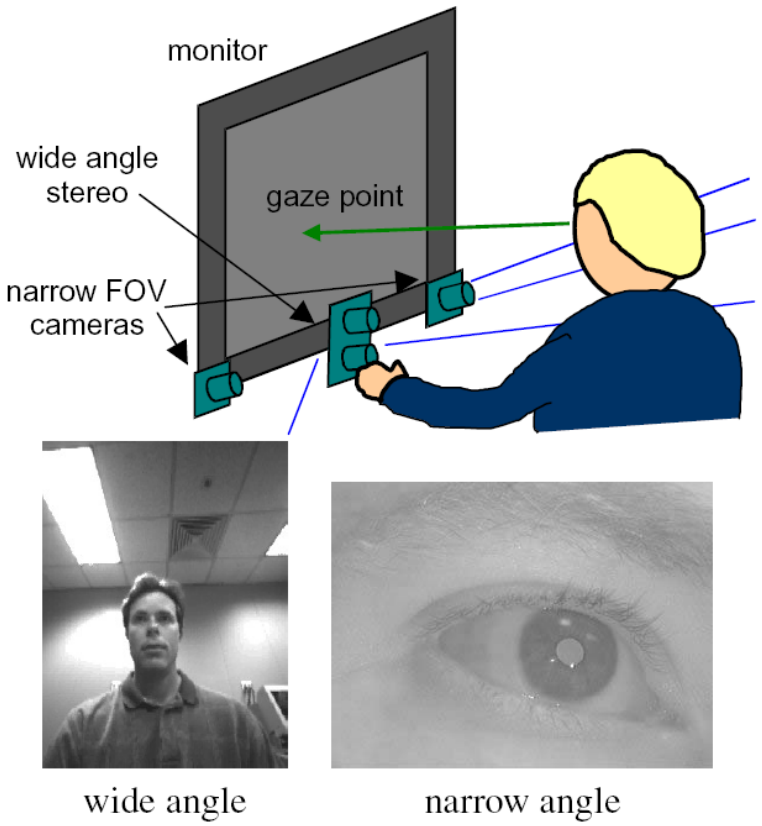

Figure 3.11: Combination of wide angle stereo for head detection and narrow angle stereo for high resolution eye tracking, ©IEEE 2003

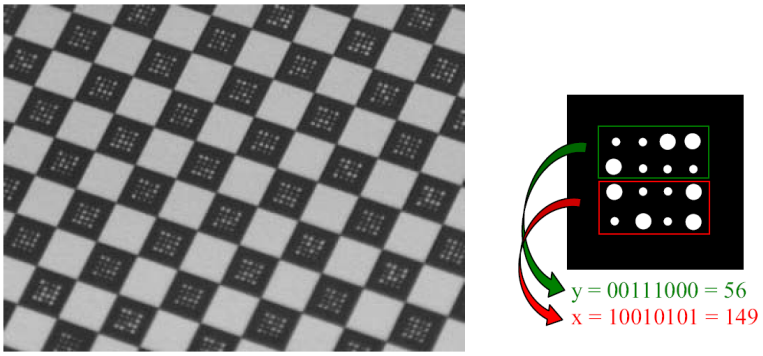

Figure 3.12: Example checkerboard image acquired by one of the narrow FOV cameras and the 2D dot code, ©IEEE 2003

error to find a well-suited projection matrix to map the 3D targets to 2D positions.
Although the system was calibrated to each user's eye, the estimated foveal angle — the angle between the optic and the visual axis of the eye — could not be stabilized. "The other angle seems to be less stable with higher variance on its estimation; stabilizing this parameter is the subject of future work."[Beymer and Flickner, 2003]
Since the foveal angle is key component in their eyeball model, the system was not able to achieve perfect gaze tracking (even with the user specific calibration). They were able track the gaze with an error angle of $0.6°$. To our knowledge this is the most accurate gaze tracking system. Unfortunately, the small looking error angle of $0.6°$ results in several mm deviation on the screen in a typical desktop scenario. This is enough for selection of big objects, but even standard desktop tasks like selection of text are impossible to realize accurately.

Zhu's neural network-based gaze tracking system is calibration-free, but inaccurate

An alternative approach for the gaze tracking problem was pursued by Zhu and Ji [2004]. Instead of using a complex 3D model to map the input data to screen coordinates, the authors use a neural network. Sufficient 2D input data (pupil, glint, reference gaze point) was used to train the network. Performance was measured with a 4×2 grid. The system was able to detect the right region 90 percent of the time, which translates to roughly $5°$ of maximum error angle. Although the gaze tracker was not as accurate as others, it achieved sufficient accuracy under head movement and — more importantly — without any user specific calibration.

Generalized theory for gaze tracking systems summarizes the previous models

As a summary of vision-based eye trackers Guestrin and Eizenman [2006] developed a "General Theory of Remote Gaze Estimation Using the Pupil Center and Corneal Reflections" (see Figure 3.13). Their theory covers the full range of combinations (one or more cameras and one or more light source). They showed that as system complexity (i.e., the number of cameras and light sources) increases, the number of user-specific parameters, that have to be estimated through calibration, can be reduced and the con-

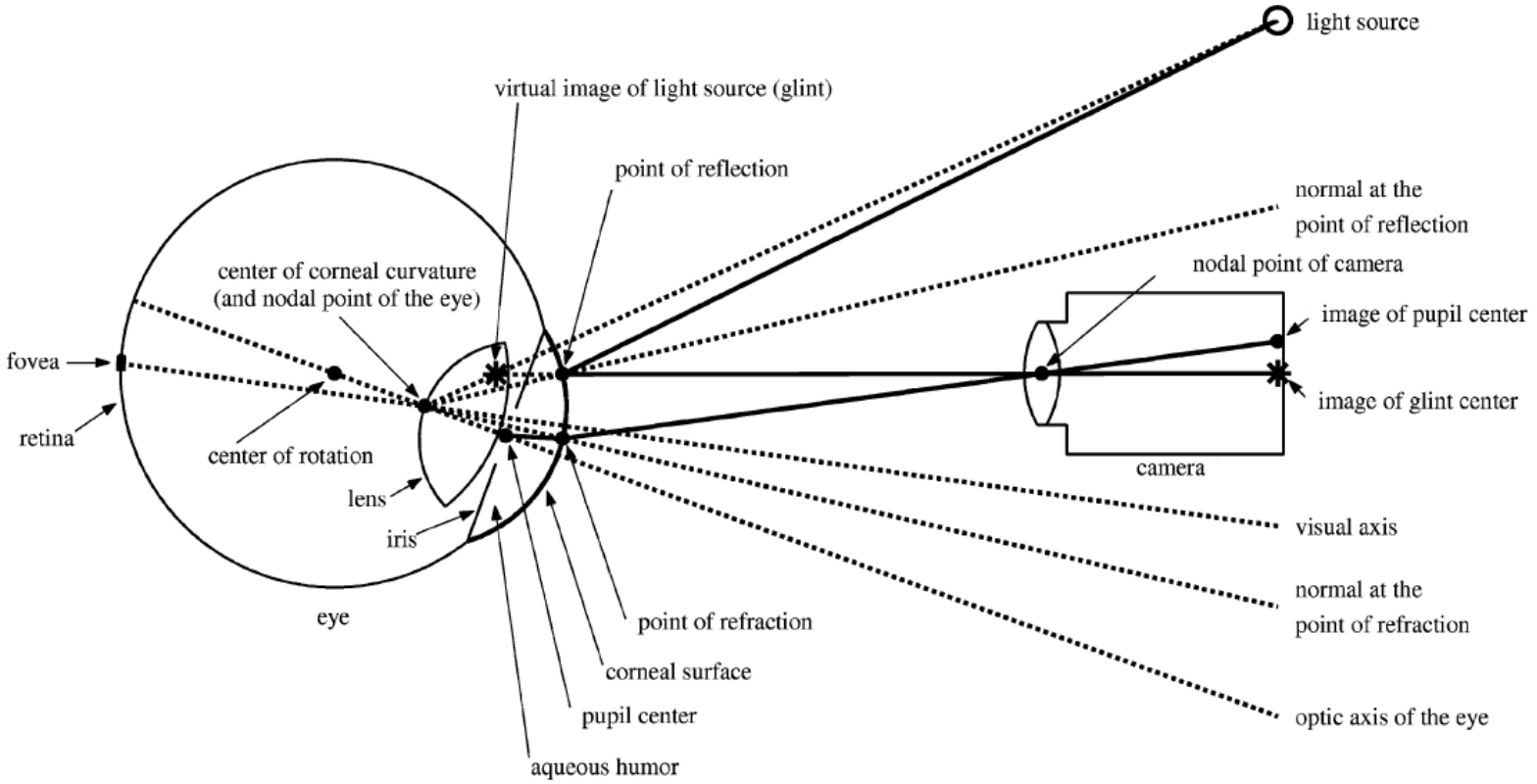

Figure 3.13: Ray-tracing diagram, showing schematic representations of the eye, a camera, and a light source, ©IEEE 2006

straints on head movement can be lifted.
Using only one camera and one light source, the head must be stationary or its distance be measured to detect the gaze point.
To allow free head movement, at least one camera and two light sources are needed. To find the gaze point in this setup, estimation or calibration of user-specific data is needed, typically including corneal radius, distance between pupil center and corneal center, refraction index of the cornea, and the angle between optic and visual axis.
The authors implemented this setup and could achieve the usual accuracy of less than 1° error angle. More interesting was their analysis on the cause of this error. They added noise to the detected 2D pupil and glint positions (gaussian distribution with standard deviation of 0.1 pixel) and found out that it had quite an impact on the quality of the output (roughly 6 mm error in the gaze point). No additional analysis concerning sensitivity of the gaze point to other calibration or measuring errors was performed.
If at least two cameras and two light sources are used, it is possible to calculate the optic (!) axis of the eye without user-specific calibration — a calibration-free system. To find the gaze point, only the difference angle between the optic and the visual axis needs to be estimated and added to the optic axis. Our system is based on a similar approach with the same advantages and drawbacks.

Noise in the image has quite an impact on the accuracy of the system

An overview of the latest gaze tracking systems can be found in Table 3.1. Except for the system by Zhu and Ji [2004], all implemented systems need user-specific calibration. As we mentioned in the introduction, user calibration reduces the application range of gaze tracking. We therefore specifically built our system to not require user calibration and investigated what hampers a good accuracy with such a system.

Paper	Glasses	Head movement	Accuracy	Calibration
[Shih et al., 2000]	no	big	n/a	no
[Ohno et al., 2002]	yes	small	$< 1°$	yes
[Beymer and Flickner, 2003]	no	big	$0.6°$	yes
[Ohno and Mukawa, 2004]	yes	big	$< 1°$	yes
[Zhu and Ji, 2004]	no	small	$5°$	no
[Guestrin and Eizenman, 2006]	yes	big	$< 1°$	yes

Table 3.1: Gaze tracking system comparison

3.2 Interaction-techniques and Applications

One goal of engineering a gaze tracking system is to support further interaction research. One of the pioneers who dealt with gaze interaction techniques was Jacob [1991]. He investigated the utility of eye movements as a mode for human-computer communication. The first idea was to use the user's gaze as a substitute for a pointing device like the mouse. Jacob concluded that this is an infeasible approach, because users are accustomed to look at items without having the look "mean" something. He coined the term "Midas Touch".

Only using gaze as input modality is insufficient for desktop-based interactions

The system cannot discern whether actually someone wants to activate an object when glancing at it. Therefore an activation input is needed; this can be, e.g., an eye blink or a special key. Jacob suggested a short dwell-time to select the object of interest. Although time-based interactions can be problematic, this worked well in user trials.
He also analyzed a dragging trask. The best performance was achieved when using the eye to select and move the object. A pushbutton was used to pickup and drop the object. After applying smoothing filters on raw gaze point data, this worked well as the test subjects agreed.
Another task was selecting an item in a menu. Here again pointing with the eye was a good choice for opening the menu and selecting an item, but the final activation should be done with a pushbutton. He concluded that the overall approach for designing interaction techniques based on gaze tracking should be to obtain information from a user's natural eye movement, rather than forcing him to do specific and unnatural eye movements. The system should

be designed to be able to cope with artifacts like fixation jitter.

Reduce cursor movement by warping the cursor to the user's point of gaze

Based on this work, Zhai et al. [1999] presented a new pointing technique called "Manual and Gaze Input Cascaded (MAGIC) Pointing". The idea is to eliminate a large portion of the cursor movement by warping the cursor to the eye gaze area and then clicking with a classical pointing device like a mouse on the desired target. Although some prototypical problems arose, the MAGIC pointing technique was slightly faster in user studies than conventional mouse pointing.
Overall, the subjects liked the MAGIC pointing technique for its responsiveness and less fatigue, when compared to conventional mouse pointing.

Tracking the user's gaze in a virtual environment can support pointing tasks

Another work that follows the approach of basing command input on the natural behavior of the user and not artificial command sequences was presented by Tanriverdi and Jacob [2000].
Their belief was that navigating through a virtual environment already exploits the user's existing "navigational commands", such as positioning his head, turning his body, or walking to some point of interest. Therefore enriching this VR interface with an eye movement based interface would obviously be the next step. They developed a new interaction technique for eye movement interaction in a virtual environment and compared it to more conventional 3D pointing in the form of a Polhemus 3Space FASTRAK magnetic tracker [Polhemus, 2007]. One of the Polhemus receivers was mounted on the user's head to identify VR camera positioning and one was on a cardboard ring around the subject's finger to detect the pointing. For eye tracking they added a head-mounted gaze tracker to the head-mounted VR display.
The test task was to point at specific objects in a complex virtual environment with lots of objects like cans, spheres, vases, etc. (see Figure 3.14). As soon as the user looks or points at the object, the program enlarges the object, fades its surface color out, and exposes its internals. When the user looks or points away from the object, the program gradually zooms out and restores the initial color of the ob-

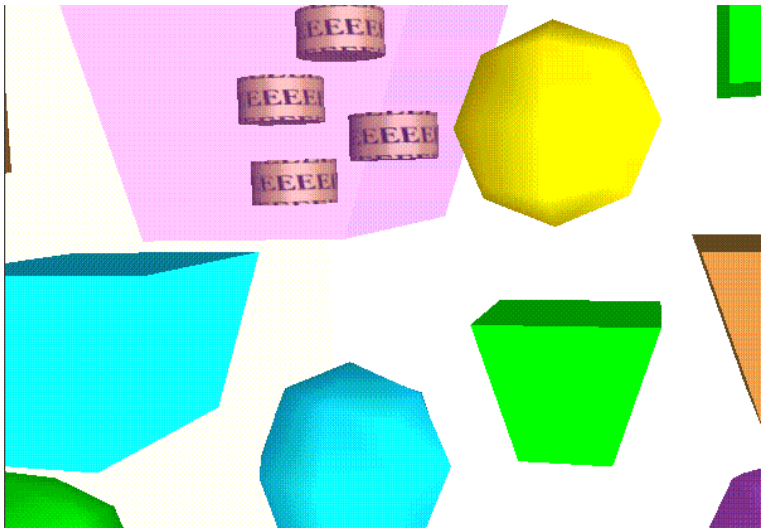

Figure 3.14: The purple object in the top-left corner is selected, and its internal details are shown, ©ACM 2000

ject — the object gets deselected.
The results are similar to Zhai's experiment for his MAGIC pointing technique. Eye movement was faster than interacting with pointing, especially when distant objects should be selected. As a trade-off to this gain, it was found out that the second task which included spatial memory of the scene was not in favor of the eye movement based technique. In this second task the users had to recall objects by their internal data they already had interacted with — similar to a pairs game. Using traditional pointing the user had to spend extra physical effort to reach out to the objects and interact with them instead of just looking at the object of interest. This way they spent more time with the virtual environment and built a more detailed model compared to traditional pointing. Hence the performance, measured as the number of right pairs, was better.

Gaze in attentive user interfaces can improve the usability of the whole system

Switching from virtual reality to augmented reality, we find the work of Maglio et al. [2000] about "Gaze and Speech in Attentive User Interfaces". An *attentive user interface* pays attention to what users do so that they can attend to what users need. They observed users with test tasks in an "office of the future", where information is accessed on dis-

plays via verbal commands. The office consisted of several flat screens, labeled as "Calendar", "Map/Direction", "Address", a printer, and a futuristic looking orb labeled "Dictation". To compensate for technological shortcomings, Wizard-of-Oz techniques were used to timely react on user input.

Results suggest that gaze is a robust source to find out which device the user wants to interact with

Maglio et al. found out that the human-computer communication was very similar to human-human communication in his tests. This correspondence is also known as the "Media Equation", a term coined by Nass et al. [1995]. The majority of requests were issued directly to the device, i.e., looking at the printer and then saying "give me a copy" and only less than two percent specified the device like in "Printer, give me a copy".

Mode	Activate	Feedback	Deactivate	Feedback
PTT	Switch the microphone to "on"	Physical status of the switch	Switch the microphone to "mute"	Physical status of the switch
LTT	Turn head toward Sam	Sam shows listening expression	Turn head away from Sam	Sam shows normal expression
TTT	Say "computer"	Special beep	Automatic (after 5 sec)	None

Figure 3.15: How to activate and deactivate the speech interface using push-to-talk, look-to-talk, and talk-to-talk, ©ACM 2002

Comparison of look-to-talk, talk-to-talk, push-to-talk interactions

Based on these results, Oh et al. [2002] evaluated the usefulness of a gaze-aware speech interface in a collaborative environment. Their hypothesis was that using gaze as an interface to activate the speech recognition would enable a natural human-computer interaction in the collaborative entertainment. To test the hypothesis they compared three methods to activate the speech recognition; gaze-driven "look-to-talk", spoken keyword-driven "talk-to-talk", and a classical key press-driven "push-to-talk" (Figure 3.15). They ran the first series of experiments with then state-of-the-art eye tracking and speech recognition software and the second series with a Wizard-of-Oz setup. The

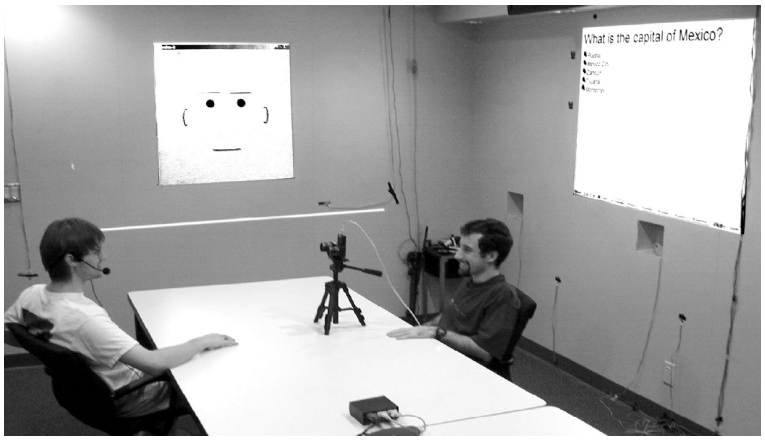

Figure 3.16: Subject A (left) can read the questions from the wall. Subject b (right) discusses the task with subject A and acts as a collaborator. In the background one can see the quiz master Sam, who changes his expression depending on whether he is listening or not, ©ACM 2002

experiment was set up with two subjects and a software agent (Sam) — (see Figure 3.16). During the experiment, Sam reads quiz questions through a text-to-speech module. Each pair of subjects was confronted with three sets of six trivia questions, each set using a different mode of interaction.

Users preferred LTT and TTT over PTT

Test results show that under ideal conditions (i.e., Wizard of Oz), users preferred LTT and TTT over PTT. A remarkable answer of one the subjects, whether he preferred LTT, was "I just turned my head to answer and noticed that Sam was already in listening mode".

Enhancing video conferencing with gaze cues of the users to make it feel more realistic

A more realistic collaborative environment was built by Okada et al. [1994]. They constructed a multi-party video conferencing system that supports multiple eye contact among the participants and includes gaze awareness. This "Multi-Attendant Joint Interface for Collaboration" (MAJIC) works in a multi-site and multi-user environment. It projects life-size video images of conference participants onto a large curved screen with boundaries between them in an effort to give the users the look and feel of a meeting at a real table with all persons sitting next to each other.

Figure 3.17 shows an example three-way setup of the

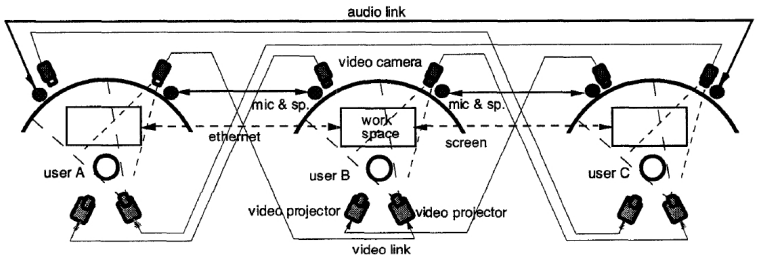

Figure 3.17: Three-way videoconferencing using MAJIC, ©ACM 1994

MAJIC system.
Sellen [1992] estimated that 60 percent of conversation involves gaze and at least 30 percent includes mutual gaze. For that reason MAJIC users can make eye contact with an individual participant and be aware of the direction of each others gaze. When user A turns his head to the right to look straight at user B, user B sees user A full face and user C sees the left profile of user A. This means that user C becomes aware of user A gazing toward user B and so on. In adittion to the video stream modifications the audio was changed. The users hear the voices more from the left or the right, depending on where the others are facing.
To evaluate whether this gaze-aware system enriches the user's experience, a sample system was set up at a collaboration fair in Tokyo. 20 visitors participated in the three-way conferencing system. They were asked to play a modified version of poker. Everyone was able to make multiple eye contacts and to become aware of the gazes of each other.
Many of them provided positive answers concerning a feeling of togetherness. Although the images on the screen had a low resolution and the users were not allowed to move their head, they were impressed by the overall experience.

User tests show that the system was able to deliver a more realistic conference experience

Another videoconferencing system was presented by Vertegaal [1999]. They based their research on the idea that "designing mediated systems is a problem of conveying the least redundant cues first" [Vertegaal, 1999]. Their GAZE

Minimizing turntaking problems by giving cues of the users's gaze

groupware system also is aware of the user's gaze and thereby tries to convey this important cue for multiparty communication in order to minimize turntaking problems. The GAZE system simulates a four-man round-table meeting room. As can be seen in Figure 3.18, the camera images

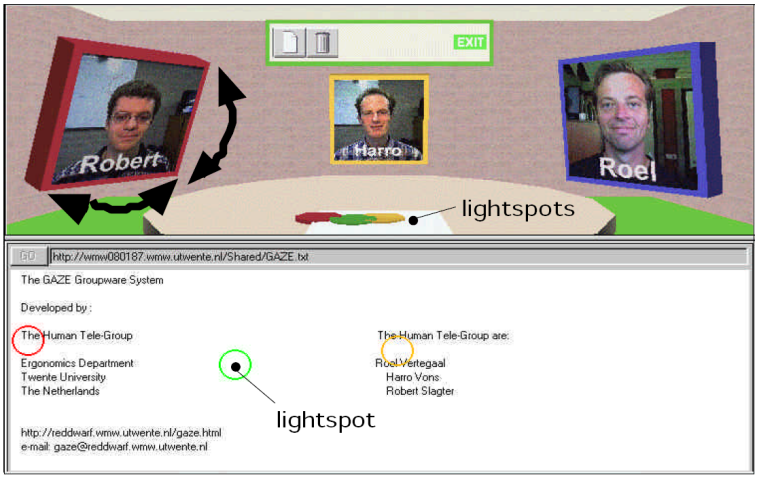

Figure 3.18: Screens rotate according to where users look, ©ACM 1999

of the users are mapped onto virtual screens. They rotate according to the user's gaze. A virtual light is attached to the top of each virtual screen to visualize the spot the user is looking at — the so called lightspot. It even allows the users to see where other participants gaze at in a shared document.
Evaluation of the system took place at ACM Expo'97 with informal sessions of several hundred novice users. Most users did not even seem to notice the gaze tracking and just sat in front of the system to talk to other participants and enjoyed the system.

More formal study of their system

A more thorough study on the "Effects of Gaze on Multiparty Mediated Communication" was done one year later by Vertegaal et al. [2000]. Groups of two actors and one test subject solved language puzzles in three different audiovisual communication conditions. They first presented a frontal motion video of the actors to the subject, then the second included motion video with gaze direction cues

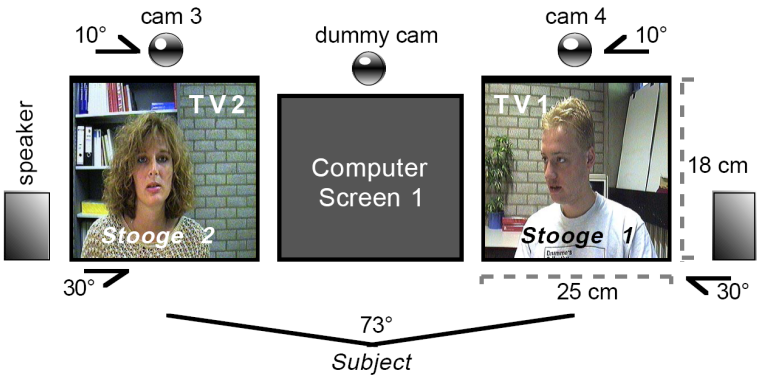

Figure 3.19: The video-mediated system used by the subjects, ©ACM 2000

(i.e., the actors were facing the computer, the subject, or the other actor). The third one had still images of the three aforementioned head poses, but also gaze direction cues.

Gaze-awareness supports efficient turntaking

The results show that gaze is an important factor for successful collaborative work — even more important than moving from still images to video as there were no significant differences between setup 2 and 3. Subjects used twice as many references to persons when gaze cues were present. When gaze was lacking, the turn-taking efficiency decreased by 25 percent.

Gaze-aware level of detail rendering improves performance

But even in non-collaborative environments gaze can be a useful input modality. One can reduce the level of rendering detail in the area off the gaze spot. Parkhurst and Niebur [2004] developed a system that needed users to search for a named object in several different complex home interiors. Results indicated that using a medium degree of level of detail reduction off the gaze spot resulted in a decrease of overall search time. Because of the reduced number of polygons the system would even run faster and provide a much more realistic experience due to smoother rendering.

To conclude the related work section we present a special use for an eye tracker — as a control device for video games. Smith and Graham [2006] evaluated gaze as substi-

tute for the mouse in three games from different genres:
A first-person shooter where the user controls orientation, a role-playing game in which an avatar is moved through an interactive environment through pointing, and an arcade game in which moving objects are targeted through pointing.
A large majority of the users felt more immersed to the gaming experience. Therefore they preferred the look-to-walk interface of the role-play game over the standard mouse interface. In the first-person shooter the "midas touch" was a large problem. If a player was walking throug a long plain hallway and an interesting object appeared the use would turn to that object, although most of the time he still wanted to move on straight. Therefore those players favored the mouse for such an input. Conclusively, the users voted for future use of gaze tracking in games where it supports the input metaphor, but not as a general pointing substitute.

Chapter 4

System Design and Implementation

"When solving problems, dig at the roots instead of just hacking at the leaves."
–Anthony J. D'Angelo

In the previous chapter, we presented a selection of research on the engineering problem of gaze tracking systems. The main focus was to achieve accurate gaze tracking for single-user desktop or multi-user videoconferencing applications, which have the same requirements from the engineering perspective. They usually focused on persons with two healthy eyes, i.e., no squint-eyed people, no abnormal curvature of the corneal surface, and so will we. A more severe problem for most (and our) systems, that are based on reflections of specific light sources, are people wearing glasses. For the current implementation, we only considered people without glasses.
Considering the interactions, most of the papers focussed on gaze as substitute for pointing devices. More current ones dealt with using the visual cues gaze tracking can give for interacting with users, but they were very focused on multi-site video-conferencing.

Most gaze tracking systems require user calibration

Unfortunately, current gaze tracking systems are not capable of tracking users' gaze (with acceptable accuracy)

without a cumbersome precalibration step.

Current research thinks that the main reason for inaccurate tracking is due to differences in the user's eyes

Current research argues that a precalibration step is necessary due to the difference between the optic and the visual axis, which is different from user to user. But even with precalibration, systems still have an error of more than $0.5°$ (e.g. [Guestrin and Eizenman, 2006]). This accumulates to roughly 1 cm deviation on a 1 m distant screen — not accurate enough for tasks like text selection.

System calibration is at least as important as user calibration

We are confident that these errors have another source — the calibration of the system. By this we mean the location and orientation of the hardware components (camera, light source).
Every system that wants to track the 3D point that both eyes are focusing on — the focal point — has to calculate the gaze of each eye. Using vision-based methods, the cameras have to extract features from the images and then calculate some intermediate points, lines, or planes for further use in a mathematical model. Due to spatial proximity of the tracked features, even slight errors in the complex mapping from the 2D to the 3D position will have an impact on the accuracy of the gaze.
One could in some way calculate cornea center and pupil center and then use these anchorpoints for the focal point calculation. These two points are not more than 1 cm distant, so if the cornea center position is wrong by 1 mm, the gaze will be rotateted around the pupil center and the focal point will be off by 10 cm.

Our gaze tracking system is thoroughly evaluated

As we mentioned in Chapter 1—"Introduction", the goal of our work is to explore what kind of calibration errors have the biggest impact on the system's accuracy. To analyze these dependencies, we start with building a user calibration-free gaze tracking system. We add noise to several of the system calibration variables (rotation of the cameras, position of LEDs, etc.) and will show in Chapter 5—"Evaluation" how unstable a gaze tracking system can react to minor variations of some of the variables.

A POV-Ray based setup was tested

To make sure that our results are not flawed by undetermined measuring errors or other deficiencies in the

hardware setup, we used POV-Ray to design a computer model of the human eye. Our gaze calculation algorithm was slightly modified to work on these computer-generated images and the same tests as on the real setup were performed.
We calculate an output error per input error ratio and can then order the parameters by their impact. We think that the types of error that "move" the key components (like the cornea center) of the gaze direction calculation the most will lead to the biggest errors in the accuracy of the system.

We focus on the main reasons for inaccuracies

Other researchers [Shih et al., 2000], [Guestrin and Eizenman, 2006] have shown that adding noise to the 2D positions of the detected features has an influence on the gaze direction. A zero-mean gaussian distributed position error with 0.1 pixel standard deviation yielded a $0.3°$ and $0.26°$ respectively error angle.
Although the numbers are quite high we think that in the days of high resolution cameras and more than sub-pixel accurate calibration systems the intrinsic camera parameters are not that important anymore.
Instead, initial tests showed that especially the extrinsic camera parameters have a high impact. We design an adaptive calibration algorithm that attempts to reduce the impact of these error sources and increase the robustness of the system.

Noisy images can reduce the accuracy

We first present the hardware setup. We then describe the underlying theoretical model and after that we will talk about the implementation details.

4.1 Hardware Setup

Based on Shih et al. [2000] we built our own gaze tracking system. We use three narrow angle cameras from Xuuk [2007], which are mounted on a table using video stands and six near infra-red light sources. They are connected to an Apple MacBook Pro (Intel Dual Core 2 2.3 GHz Processor with 2 GB of RAM).
As is shown in Figure 4.1, each video camera has a ring

Figure 4.1: The hardware setup consisting of three cameras and three additional off-axis LEDs

of on-axis LEDs and a set of off-axis LEDs connected to it to generate the dark and bright pupil images. Using a triangular lighting pattern makes it very easy to distinguish between the different corneal reflections. The lighting alternates between all off-axis LEDs illuminated and the on-axis LEDs not illuminated and vice versa.
Although the algorithm itself is capable of tracking a moving user, the current prototype limits his movement. The user bites on a spoon, so head movement is minimized and cannot handicap the tracking process, due to too much noise in the difference images. The main reason for this decision is that the cameras have a low framerate due to high resolution images. Another problem is that these high resolution images are connected by a shared USB and have to be processed on the same pc.

Our system does not require a user-specific calibration, but it is necessary to measure the position of the LEDs, the display, and the cameras. For the cameras also the rotation w.r.t. an anchor coordinate system is required.
For calibration, we use a rigid checkerboard pattern (see Figure 4.5). The lower left corner of the pattern is then de-

Figure 4.2: Image-cutout with off-axis illumination

Figure 4.3: Image-cutout with on-axis illumination

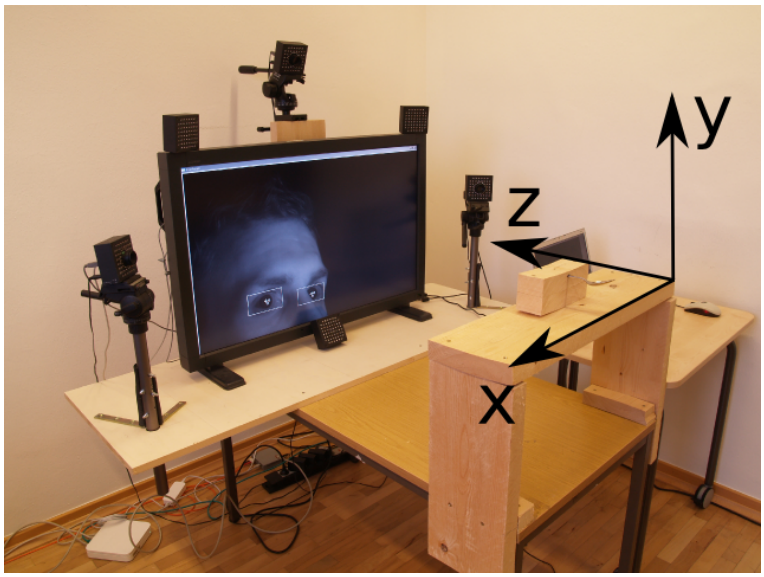

Figure 4.4: The coordinate system

fined as the origin of our coordinate system.
To calculate the position we define a global coordinate system. As can be seen in Figure 4.4 the x-axis represents horizontal shifts, the y-axis represents vertical shifts, and the z-axis resembles the depth. 1 mm in a direction equals 1 in the coordinate system.

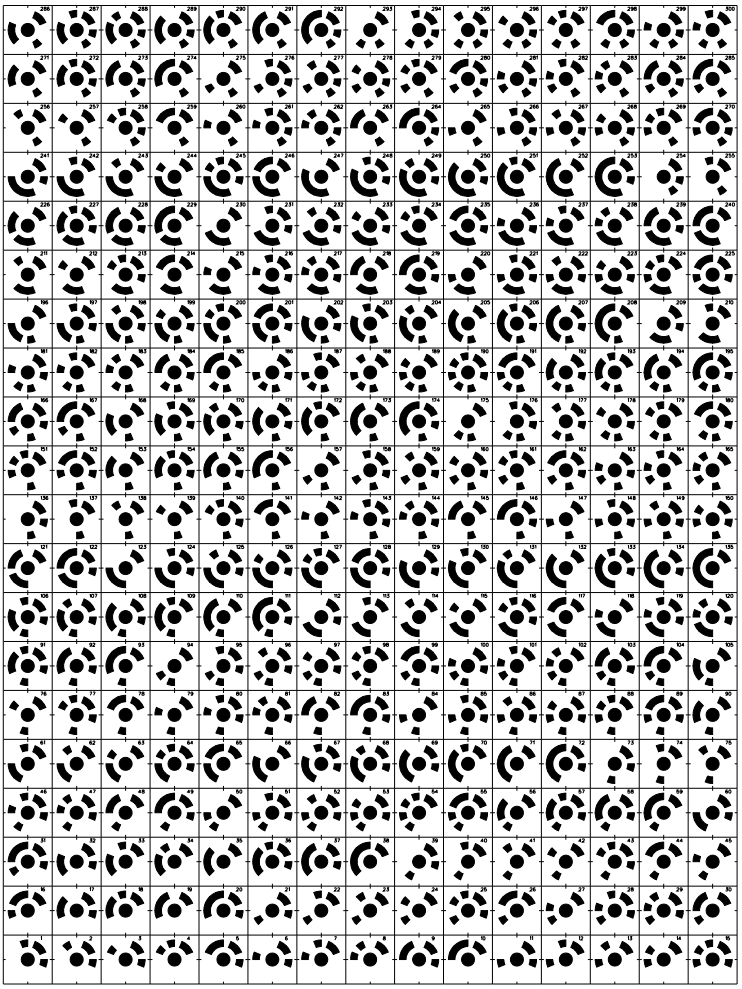

Figure 4.5: Calibration pattern used for setup of the coordinate system

We use PHIDIAS[1] for the camera calibration. This is a software toolkit for close range photogrammetry and was developed by a spin-off of the RWTH Institute for Geodesy. It returns intrinsic parameters and extrinsic parameters w.r.t. the calibration pattern. The average reprojection error of the determined parameters is lower than $0.4\mu m$.

[1] More information at http://www.phocad.de/Produkte/PHIDIAS_Info_en.pdf

4.1.1 POV-Ray Model

To make sure that our results are not falsified by flaws in the calibration of the hardware, image requisition, or similar external factors, we decided to also build two computer models of the human eye. One consisting of only an eyeball

Two POV-Ray models are designed to evaluate with perfect data

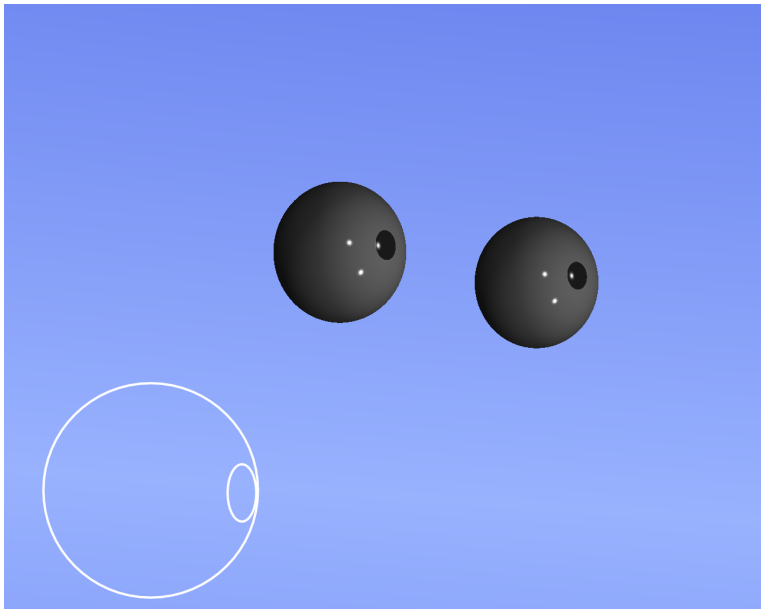

Figure 4.6: The simple eye model rendered by POV-Ray

and a pupil (Figure 4.6) and another one also modeling the anterior chamber of the eye before it (Figure 4.7). The models use appropriate sizes for cornea radius, refraction index, and eye radius. Both eyes are rotated to a focal point which is specified in the POV-Ray data file. The whole hardware setup was remodeled, i.e., every calibration parameter (position of cameras, LEDs, etc.) was the same for the computer generated model.
The same parameters were also used for when performing eye tracking on real-world eyes.

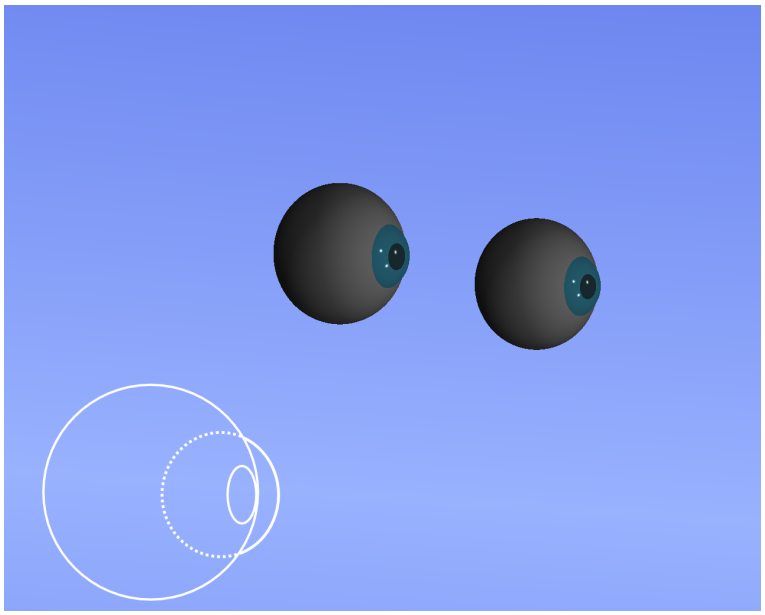

Figure 4.7: The more complex eye model rendered by POV-Ray

4.2 Gaze Model

Research has brought up different eye models

As we have shown in Chapter 3—"Related Work", most gaze tracking systems internally remodel the human eye. Each system has a different abstraction of the very complex human visual system. Therefore they all behave slightly different when confronted with (synthetic) input errors. Analysing each of these models is beyond the scope of this work.
To be able to apply our research and our results to these systems we compared their eye models and chose a generalized model. The eye is modeled as a sphere with a little bulge on it, representing the aqueous humor with the cornea on top of it.

Similar systems should behave similar

Although our results will primarily be applicable to our system, we are confident that other gaze tracking systems, which are based on more detailed models, behave similar. Since most systems use comparable feature tracking, image processing, and mathematical models, the absolute num-

bers (output error per input error) will be different, but the magnitude and especially the ordering between different types of error should stay the same.
To determine the gaze of the user, we need to find the cornea center and the pupil position (see Definiton 2.2—"A Simple Eye Tracking System", starting with the cornea center.

4.2.1 Cornea Center

LED reflections are used for calculation of the cornea

In every odd frame the on-axis LEDs are lit and in every even frame the off-axis LEDs are lit. Some of the emitted rays hit the cornea of the user's eye. They are reflected and are eventually recognized as small bright spots by the video camera.

Theorem 4.2.1. *Let V be the video camera center, L the LED position, and R the spot on the cornea where the light gets reflected. Then the cornea center C is also on the plane defined by (V, L, R) (see Figure 4.8).*

Proof. According to the law of reflection the incident ray $\overrightarrow{LR}$, the reflected ray $\overrightarrow{RV}$, and the cornea's normal n, which stands on the reflection spot R, are all in the same plane.
If we assume a healthy eye, $-n$ will point from the reflection spot R to the cornea center C.

$$\overrightarrow{OC} = \overrightarrow{OR} - \alpha n$$

for some scalar α. Multiplying by the normal n_p of the (V, L, R) plane yields

$$n_p^t \overrightarrow{OC} = n_p^t \overrightarrow{OR} - \alpha \underbrace{n_p^t n}_{=0}$$

This means that R and C lie on the same plane. Therefore C also lies on the plane defined by (V, L, R) □

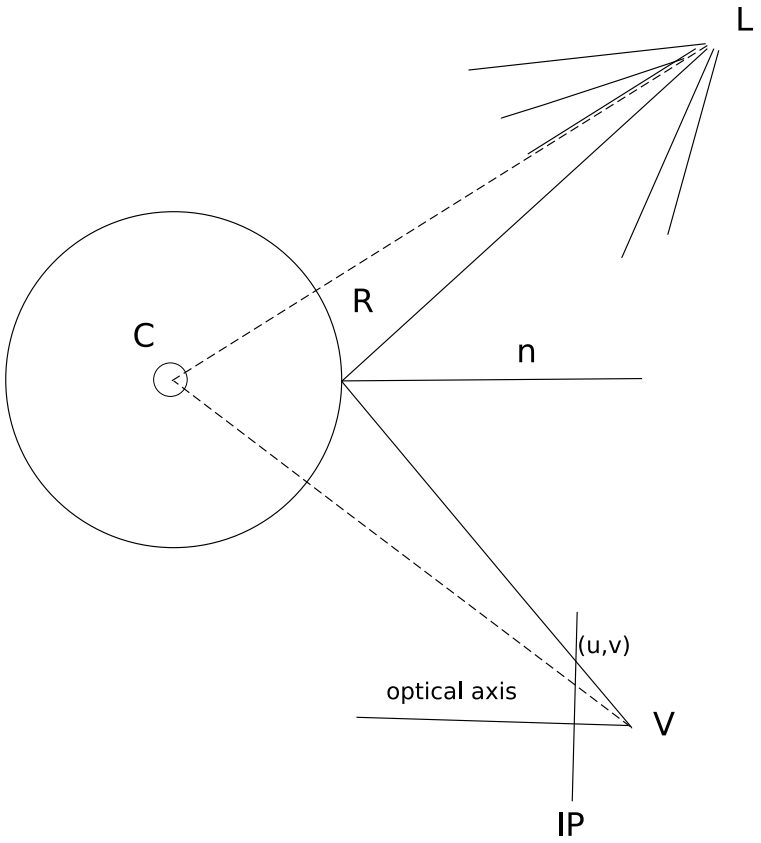

Figure 4.8: Center of the cornea

If we use more than one video camera and more than one LED, we get multiple planes defined by $(V_i, L_j, R_{i,j})$, for $i \in 1, .., k$ video cameras and $j \in 1, .., l$ LEDs.
If we assume perfect data, the intersection of at least three non-coplanar planes yields one point. This is the cornea center C.
By precise measuring of our hardware setup, we get the video camera centers V_i, and the positions of the LEDs L_j.

The reflection spots at the cornea $R_{i,j}$ are more difficult to calculate. We know that they lie on the ray that goes from V_i to $R_{i,j}$ intersecting the image plane at $(u_{i,j}, v_{i,j})$, where $u \in [0, w-1], v \in [0, h-1]$, w is the image width and h is

the image height.
By unprojecting the image pixel $(u_{i,j}, v_{i,j})$:

$$x_{i,j} = \frac{(u_{i,j} - c_x) z_{i,j}}{f_x}$$
$$y_{i,j} = \frac{(v_{i,j} - c_y) z_{i,j}}{f_y}$$

we get the direction of the ray $\left[\overrightarrow{V_i R_{i,j}}\right]_L$: $\begin{pmatrix} \frac{(u_{i,j}-c_x)}{f_x} \\ \frac{(v_{i,j}-c_y)}{f_y} \\ 1 \end{pmatrix}$
w.r.t. the local coordinate system L of the video camera, where (c_x, c_y) is a principal point and f_x, f_y are focal lengths of the camera expressed in pixel-related units (see the appendix for more information about the pinhole camera model).
These video camera-specific parameters were precalculated using the PHIDAS software. The vector $\left[\overrightarrow{V_i R_{i,j}}\right]_L$ then has to be converted from the video camera coordinate system L to the global coordinate system G. The necessary rotation matrix and the translation vector can also be precomputed for each video camera. Because vectors are not affected by a translation we can invert the rotation matrix R:

$$\left[\overrightarrow{V_i R_{i,j}}\right]_G = R^{-1} \left[\overrightarrow{V_i R_{i,j}}\right]_L = R^{-1} \begin{pmatrix} \frac{(u_{i,j}-c_x)}{f_x} \\ \frac{(v_{i,j}-c_y)}{f_y} \\ 1 \end{pmatrix}$$

We have all vectors and points given w.r.t. the global coordinate system. As aforementioned, the cornea center C is at the intersection of the planes $(V_i, L_j, R_{i,j})$. Every point P on a plane with normal n has to satisfy:

$$n^t P = d$$

with d being the distance from the plane to the origin (of our global coordinate system). We get n via the cross-product of $\overrightarrow{V_i R_{i,j}}$ and $\overrightarrow{V_i L_j}$, and $d = n^t V_i$.
That means for one pair of video cameras and LEDs we get an equation of the form:

Linear constraint for the cornea center is found

$$(\overrightarrow{V_i R_{i,j}} \times \overrightarrow{V_i L_j})^t C = (\overrightarrow{V_i R_{i,j}} \times \overrightarrow{V_i L_j})^t V_i \qquad (4.1)$$

In this LES (linear equation system) only C is unknown, but can be calculated as soon as we have at least three linear independent equations i.e. different pairs of video cameras and LEDs.

If we have more than three pairs of video cameras and LEDs, the LES is overdetermined. In theory the three or more planes should still intersect at one point, but because of noise in the real data they do not. Therefore, we attempt to find a point which is as near (in a least square error sense) as possible to all of the planes.

The least square error approach returns the cornea center

The distance of a point $p = (p_x, p_y, p_z, 1)$ to a plane N, defined by its normal $n = (n_x, n_y, n_z, 0)$ and one point $q = (q_x, q_y, q_z, 1)$ on it, is $n^t p - n^t q$. By summing up the distances to each plane we get the distance to all planes $\Phi = \{N : N = (n_x, n_y, n_z, -n^t q)\}$. Thus

$$\sum_{N \in \Phi} (n^t p - n^t q)^2 \rightarrow min \tag{4.2}$$

should be minimized. Calculating the partial derivatives of this function for all three varying dimensions (x, y, z) and setting them to 0 gives a new LES with three equations. As long as we have at least three non coplanar planes we only need to solve the system and then have the cornea center.
This least square error approach will give a more robust cornea center than the naive three plane intersection approach.

Because we always have the same kind of function we can even optimize the minimization [Horn, 1987].
The distance from the point p to the planes Φ can be written as:

$$\begin{aligned} distance(p, \Phi) &= \sum_{N \in \Phi} (N^T p)^2 \\ &= \sum_{N \in \Phi} p^t (N N^t) p \\ &= p^t (\sum_{N \in \Phi} N N^t) p \\ &= p^t (\sum_{N \in \Phi} Q_N) p \end{aligned}$$

where Q_N is the error quadric of the plane N. For a plane $N = (n_x, n_y, n_z, d)$, with $d = -n^t q$ the *symmetric* quadric is:

$$Q_N = \begin{pmatrix} n_x^2 & n_x n_y & n_x n_z & n_x d \\ n_x n_y & n_y^2 & n_y n_z & n_y d \\ n_x n_z & n_y n_z & n_z^2 & n_z d \\ n_x d & n_y d & n_z d & d^2 \end{pmatrix}$$

For a set of planes we just have to add each quadric:

$$Q_\Phi = \sum_{N \in \Phi} Q_N$$

We now can reformulate our optimization problem:

$$p^t Q_\Phi p \rightarrow min$$

For a general symmetric quadric

$$Q = \begin{pmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{pmatrix} =: \begin{pmatrix} A & l \\ l^t & m \end{pmatrix}$$

with $A \in \mathbb{R}^{3x3}, l \in \mathbb{R}^3, m \in \mathbb{R}$
we calculate the partial derivatives of $F = (p_x, p_y, p_z, 1)\ Q\ (p_x, p_y, p_z, 1)^t$.

$$\begin{aligned} \frac{\partial F}{\partial p_x} &= 2(ap_x + bp_y + cp_z + d) \stackrel{!}{=} 0 \\ \frac{\partial F}{\partial p_y} &= 2(bp_x + ep_y + fp_z + g) \stackrel{!}{=} 0 \\ \frac{\partial F}{\partial p_z} &= 2(cp_x + fp_y + hp_z + i) \stackrel{!}{=} 0 \end{aligned}$$

Back to matrix notation this can be written as:

$$\begin{pmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \leftrightarrow \begin{pmatrix} a & b & c \\ b & e & f \\ c & f & h \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} -d \\ -g \\ -i \end{pmatrix} \leftrightarrow Ae = -l$$

Solving this 3×4 LES yields the point nearest to all planes without computing any derivatives at all.

4.2.2 Gaze Direction

The calculation of the gaze direction is done similar to the cornea center calculation. To find the gaze, we need the location of the pupil center P_r (Figure 4.9). Because of the convex appearance of the cornea, our video cameras see, not the real pupil, but a refracted image it.
The video camera sees the projection of the virtual pupil center (u, v) in an image. By unprojecting this pixel (as seen in Section 4.2.1) we get the ray $\overrightarrow{V_i P_{v_i}}$ from the video camera to the virtual pupil.
If we had only one camera we would need user-specific information (like the refraction index of the cornea, the distance between pupil and cornea center, etc.) to calculate the real pupil center [Guestrin and Eizenman, 2006]. But with the help of two cameras and theorem 4.2.2 we can find it without this data — hence our system does not need user-specific calibration.

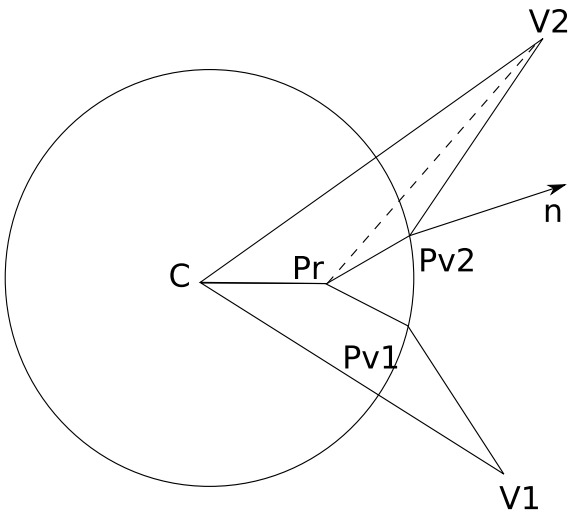

Figure 4.9: Virtual pupil image

Theorem 4.2.2. *Let P_r be the real pupil center and P_{v_i} the virtual pupil center seen by video camera V_i. Then the cornea center C lies on the plane defined by (P_r, P_{v_i}, V_i) (see Figure 4.9).*

Proof. The proof works analog to the one for theorem 4.2.1. The light travels through the cornea and eventually hits the real pupil center. It is reflected outwards and w.l.o.g. ends up at video camera V_2. According to the law of refraction the incoming ray $\overrightarrow{P_r P_{v_2}}$ the perpendicular n, and the refracted ray $\overrightarrow{P_{v_2} V_2}$ are on the same plane. Assuming a healthy eye and therefore a sphere-shaped cornea, the perpendicular $-n$ points from the virtual pupil center to the cornea center:

$$\overrightarrow{OC} = \overrightarrow{OP_{v_2}} - \alpha n$$

for some scalar α. Multiplying by the normal n_p of the (P_r, P_{v_i}, V_i) plane yields

$$n_p^t \overrightarrow{OC} = n_p^t \overrightarrow{OP_{v_2}} - \alpha \underbrace{n_p^t n}_{=0}$$

□

Linear constraint for the gaze direction is found

Because (C, P_r, P_{v_i}, V_i) all lie on the same plane we get a linear constraint for the gaze direction i.e. $\overrightarrow{CP_r}$.

$$(\overrightarrow{V_i P_{v_i}} \times \overrightarrow{V_i C})^t P_r = (\overrightarrow{V_i P_{v_i}} \times \overrightarrow{V_i C})^t V_i \tag{4.3}$$

Only the position of the real pupil center P_r is unknown, but if we have at least two video cameras (and therefore two equations) the LES yields a ray which is the desired gaze direction.

For the case of exactly two video cameras, we can even find an easier solution. Taking a closer look at (4.3), we see that

we are trying to find the intersection of two planes.
Let n_1, n_2 be the two normals of the planes. All lines within one plane have to be orthogonal to the plane's normal, thus the intersection line has to be orthogonal to both. Therefore $l = n_1 \times n_2$ is parallel to the intersection line. Because we know that C is on the intersection, we have a parametric representation of the gaze direction of one eye:

$$\begin{aligned} gaze(\alpha) &= \overrightarrow{OC} + \alpha(n_1 \times n_2) \\ &= \overrightarrow{OC} + \alpha((\overrightarrow{V_1 P_{v_1}} \times \overrightarrow{V_1 C}) \times (\overrightarrow{V_2 P_{v_2}} \times \overrightarrow{V_2 C})) \end{aligned}$$

4.2.3 Combining Gaze

Gaze direction of each eye is unstable

Using the above algorithm we get a gaze direction for each eye. The gaze direction is the intersection of the planes that are constructed using the cornea center, the video camera position, and the unprojection of the virtual pupil center (as seen by each video camera). This unprojection passes close to the cornea center. This means that even slight errors in the calculation of the cornea center, or the unprojection will have an impact on the plane and therefore the gaze direction.

Extension of the calculations uses both eyes

To compensate for this, we expand the algorithm to take into account that both eyes are looking at the same point in space. This means the focus point F is on the gaze ray of the left and the right eye.
If we remember the calculation of the gaze direction, which was the intersection of two planes, we see that the focus point F is on all of these planes P. They are defined by the triplet:

$$(V_i, P_{v_i}, C_j), i \in \{1 \dots v\}, j \in \{1, 2\}$$

where V_i is the video camera position, v the number of video cameras, P_{v_i} the virtual pupil center and C_j the center of the cornea.
Using the least square error approach that we also use for the cornea center calculation, we find the point which is

closest to all of these planes — the focal point of the eyes. The gaze direction (from the eye to the focal point) is now based on not just an unstable two plane intersection but by an "intersection" of all $2v$ planes. Using the more stable calculated point F, we can now define the two eye gaze direction.

COMBINED GAZE:
The combined gaze vector is the vector from the center of gravity of the cornea centers $(C_1 + C_2)/2$ to the focal point F.

Definition:
Combined Gaze

4.2.4 Gaze on Screen

We now have the combined gaze ray and need to find out where it hits our display. Let $C + \lambda v$ be the gaze ray where v is the gaze direction and C the center of gravity of the cornea centers C_1, C_2. The rectangular display is given by three points P_1, P_2, P_3 (that is top-left, top-right, bottom-left), thus the normal of the display-plane is:

$$n = (P_2 - P_1) \times (P_3 - P_1)$$

To get the point I, where gaze ray and display-plane intersect (see Figure 4.10), we insert the equation of the ray into the normal form of the display plane:

$$\begin{aligned} n^t x - n^t P_1 &= 0 \\ n^t(C + \lambda v) - n^t P_1 &= 0 \\ \lambda &= \frac{n^t P_1 - n^t C}{n^t v} \\ I &= a + \frac{n^t P_1 - n^t C}{n^t v} v \end{aligned}$$

We now have the 3D intersection point and the only thing left to do is to map it to the 2D coordinate system of the display. Therefore we use $P1 + \alpha(P_2 - P_1)$ as the X direction and $P1 + \beta(P_3 - P_1)$ as the Y direction. By projecting the intersection point I on each direction vector

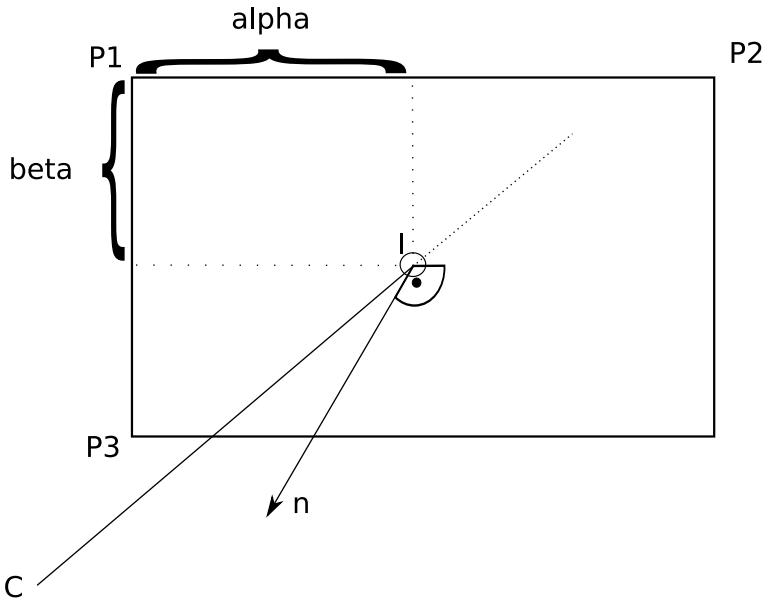

Figure 4.10: Calculation of the line-plane intersection

we get values for α and β which lie between 0 and 1. These virtual screen coordinates can then be transformed into real screen coordinates.

4.2.5 Adaptive Calibration

When we did our first tests, we observed that accuracy of the system was heavily influenced by even small errors in the camera calibration. To be able to further research this problem, we developed an algorithm that tries to compensate for camera calibration errors.

If it does not fit, make it fit

The algorithm adds an adaptive calibration step to the standard algorithm. The idea is simple: To make sure that the cameras' viewing rays for a specific point intersect at a point, we just update the cameras' rotation matrices accordingly — thereby postcalibrating the camera (see Figure 4.11).
We calculate the cornea center as described previously, but

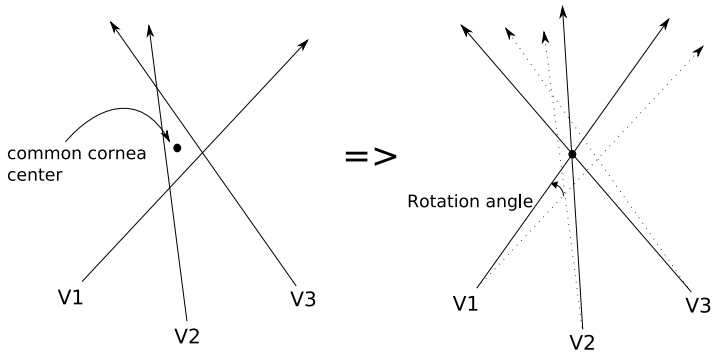

Figure 4.11: Postcalibration of the cameras by rotating onto the LES solution

then start our adaptive calibration. The specific point we want to rotate to is the cornea center. We know its position using the common knowledge of the cameras and our least square error approach.

The least square error approach does not work

We now calculate the cornea center as seen from every camera using only the knowledge of this camera. Each camera i has three planes $(V_i, L_j, R_{i,j})$, which are constructed from the reflections $R_{i,j}$ of the LEDs j on the cornea (see Section 4.2.1—"Cornea Center").
If we use the least square error approach on only the three planes of one camera, we would not get the cornea center. This is because they all share the camera position V_i and (theoretically) also each point on the line from V_i to the common cornea center. The least square error approach searches for the point which is closest to all planes, which is in this case the camera position V_i.

Instead of the least square error approach we construct the intersection of each plane pair and then average these rays. Because each plane passes next to the cornea center, so does the pair's intersection. The average of these rays is very similar to the ray from V_i to the camera's own cornea cen-

ter.
We now have a good guess for the desired ray and construct a rotation matrix that rotates the guessed ray onto the ray from the camera to the common cornea center. This matrix gets applied to the camera's rotation matrix and now that all cameras "agreed" on a point the following gaze calculations have a more stable ground.

4.3 Image Processing

The cameras do not deliver 2D coordinates of the glints and the pupil, but just 1280×1024 grayscale images. To determine the 2D coordinate, the grayscale images are processed by a pipeline of image processing methods (see Figure 4.12).

4.3.1 Detecting the Pupil

We use the differencing technique to acquire the pupil

The video cameras have a set of LEDs around them. When these LEDs are lit the frames taken by the video camera have the "red-eye" effect.
Because the LEDs are around the optical axis, we call these on-axis frames. If the LEDs are not lit they are called off-axis frames. Due to the "red-eye" effect, the on-axis frames are also called bright frames. The off-axis frames are called dark frames.
Because the major difference between the on- and off-axis frames are the red eyes, the subtraction of an off-axis from an on-axis frame yields the pupils.

Due to the noise in the difference image, simple tracking of the biggest conncted pixel components (from now on called blobs) on this image leads to many false-positives. Since for each detected blob an object is created by the blob detection library, the system runs slow on noisy difference images. To compensate for these problems, we do the following for each on- and off-axis frame pair :

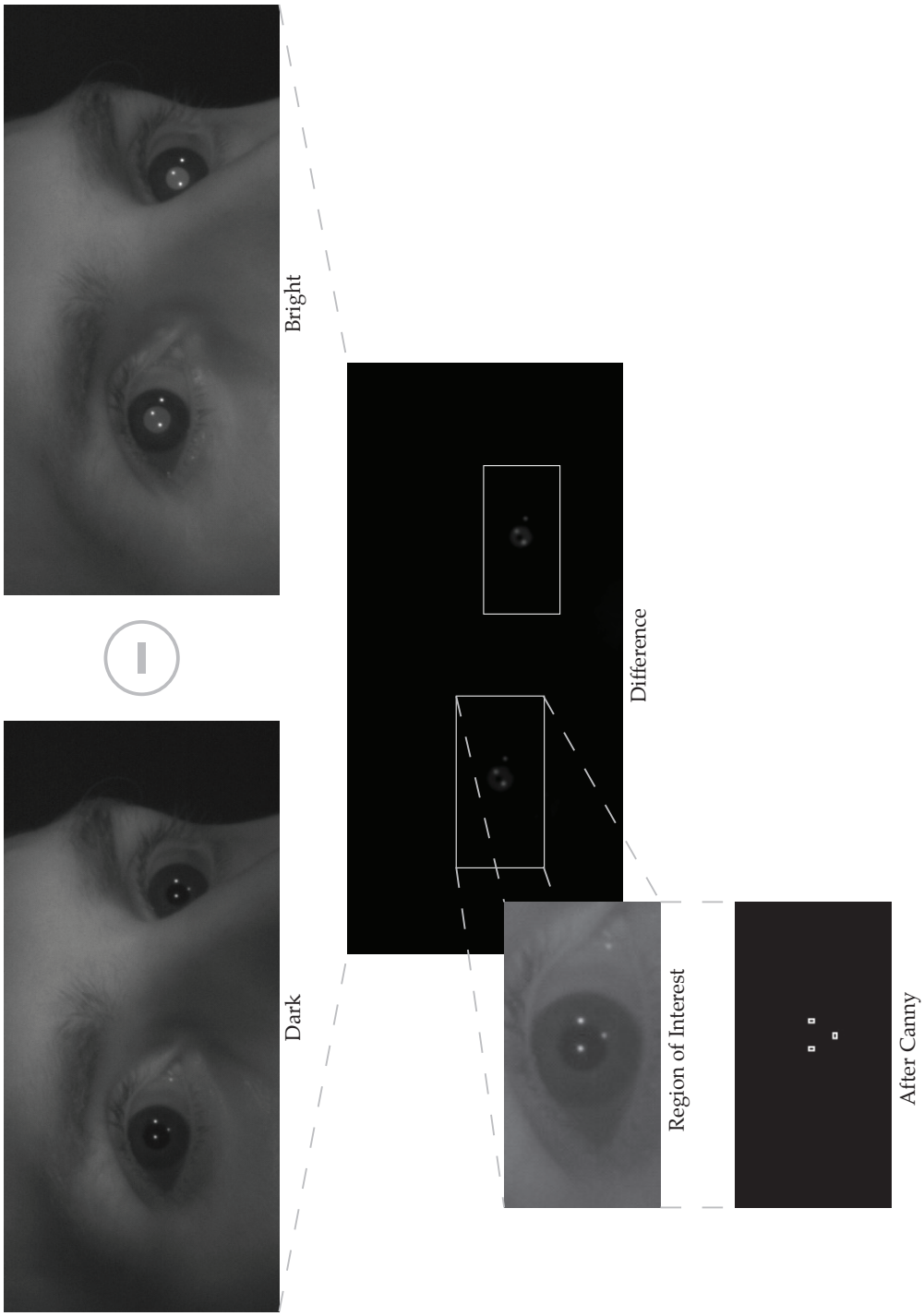

Figure 4.12: The image processing pipeline

1. Scale down the average brightness of the bright frame to the average brightness of the dark frame.
2. Subtract the dark frame from the bright frame.
3. Use an adaptive thresholding technique to filter out the noise:
   A histogram of the brightness is calculated and every pixel which is among the lower 95 percent is set to black, i.e., only the brightest 5 percent of the pixels remain in the image.

Thresholding, blob detection, and filtering distinguishes between noise and pupils

This way we get a black frame with only the brightest spots left — the pupils. The frame is fed into the blob detection routine from the `cvBlobsLib` (see Section 4.4—"Software Implementation") and we get a set of blobs. To prevent false-positives due to head movement, etc. we filter the blobs as follows:

1. Discard blobs which are smaller than 25 pixel or bigger than 1000 pixel.
2. Make a list of all blob pairs (i.e. pupil pair candidates)
3. For each blob pair
   (a) Discard all pairs with distance from blob a to b smaller than 120 pixel
   (b) Discard all pairs who are not on the same height, i.e., the line between the blob centers has an angle of more than $20°$
4. Now select the brightest blob pair; the brightness of a pair is the brightness of each blob normalized over its size

Although the second step takes $O(n^2)$ time, where n is the number of blobs, this is not a problem because the image usually has less than ten blobs after thresholding and filtering.

A ROI rectangle includes the eye

We now draw a region of interest (ROI) in the form of a

rectangle around each pupil so that it contains the whole eye and can afterwards search for the glints, which have to be somewhere on the eye (i.e., next to the pupil center). The ROI is six times as wide as the pupil and three times as high.

Exact pupil position improves accuracy

As Li et al. [2005] showed, an exact pupil position is necessary for good tracking results. The current pupil position is based on the difference image and therefore can contain errors due to small head movement. We therefore use a cutout (in the size of the ROI) from the dark frame to improve the pupil detection accuracy (see Figure 4.12).
The image is inverted and thresholded, so the (once) dark pupil remains as a bright spot. Blob detection yields the position and a new region of interest, which can be used for the glint detection. The algorithm yielded good results — we were not able to find better pupil centers manually in our sample images.

4.3.2 Finding the Glints

Because of the reflective nature of the cornea, the specular reflections of the LEDs on the cornea are very bright spots in the images we get from the video camera.
Our initial idea was plain thresholding for the glints, but that had two drawbacks. Reflections on the moist area of the eye next to the lacrimal glands were confused with the real reflections (see Figure 4.13). Therefore we prioritize glints next to the (2D) pupil center. Furthermore, the glints can be in front of the black pupil or in front of the iris, which is similiar grayish like the glints. There was no perfect thresholding value to guarantee glint detection.

Canny Edge detection gives reliable glint detection

The most distinctive feature of a glint is its brightness compared to its surroundings. Using an edge detection filter [Canny, 1986] we get a black and white image containing the glint edges (see Figure 4.12). A subsequent blob detection yields two blobs for each glint (one for the edge and one for the hole). We remove the interior blobs and yield the glints.

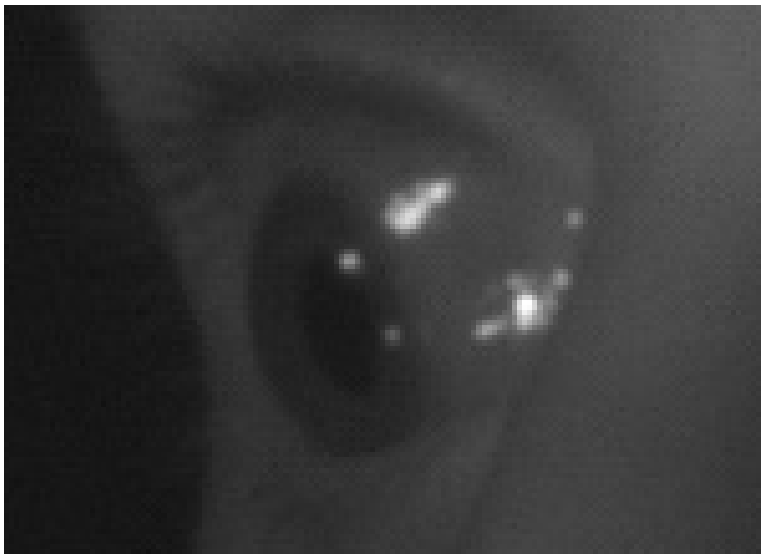

Figure 4.13: Additional reflections next to the lacrimal glands

After we found the centers of the glints in the images, we need to know from which light source they originated. Thanks to our triangular LED setup (see Figure 4.2 and 4.3) this is easily done by sorting the glints by their x-value.
We then input the 2D values to our mathematical model and calculate the gaze direction.

4.4 Software Implementation

The software resembles the image processing pipeline (see Figure 4.12) and is therefore based on procedural programming. C++ was used as a programming language mainly due to the fact that the video cameras from xuuk have an API that is based on C. To accelerate the development process, we used OpenCV[2] for the basic components of our image processing.
The only three (real) classes in the software are `GTCamera`, `CBlobResult` and `CBlob`. The latter two are part of the cvBlobsLib which is a blob detection Library based on

[2]Intel Open Source Computer Vision Library opencv.org

OpenCV. Inputting an grayscale image to `CBlobResult` does a connected component analysis on the image with an user-specified thresholding level. One can filter out blobs according to pre- or self-defined constraints (e.g. size of the blob) or sort the result set (e.g. distance to a specific pixel). The result set is a vector of `CBlob` which can be queried for the center of the blobs.

Our self written class is `GTCamera`. For each camera an instance is generated to store the camera-specific values. These include:

- the intrinsic parameters (focal length, principal point),
- the extrinsic parameters (rotation matrix w.r.t. the calibration pattern),
- a hardware handle to the camera,
- the current bright and dark frame,
- the glint and pupil positions (2D and 3D) as seen from the camera,
- and the planes we construct for the least square error calculations.

To store data that is equal for all cameras (like calculated cornea center) static variables are used, e.g. `GTCamera::right_eye_gaze`.

We will now present the important files of the software project and explain how the gaze direction is calculated. We will skip over insignificant code which is only there for initialisation of variables, exception handling in case of `NULL` pointers, etc. The same goes for all blob* files which are part of cvBlobsLib, and the lucam* files which are required to connect to the cameras.
The naming convention is as follows:

- `foo_` for global variables

- `foo` for local variables
- `_foo` for function parameters

The program can read precaptured images from files and work with computer generated images in `8` bit grayscale format.
To change the mode modify the preprocessor definitons `POVRAYIMG` and `STATICIMG` in stdafx.h accordingly. For further explanation, we will assume `STATICIMG == 0` and `POVRAYIMG == 0`.
There also exist a `_DEBUG` flag, that when activated adds several output windows to the image processing, where, e.g., the difference image is shown.
There is a lot of code in order to do the evaluation, that mainly pertuberates parameters like camera rotation angle before calculation of the gaze and compares the resulting gaze with a reference gaze, see Chapter 5—"Evaluation".

4.4.1 Algebra.cpp

This file includes many routines to create and manipulate matrices and vectors. A linear equation system (LES) solver is also included. Matrices are specified as 2D arrays of doubles and sorted row wise.

`void adaptive_calibration(std::vector<GTCamera> &_cams,std::vector<plane> &_left_eye_planes)`
A runtime postcalibration algorithm for the cameras `&_cams`. It compensates for initial calibration errors (see Section 4.2.5—"Adaptive Calibration").

`void add_plane(GTCamera &_cam, CvPoint2D64f &_glint, CvPoint3D64f &_light, std::vector<plane> &_planes, int _store_at)`
Calculates the plane from the given points (camera position, unprojection of the glint, LED position) on it and adds it to `&_planes`, See Section 4.2.1—"Cornea Center".

`double angle_between( CvPoint3D64f &_veca, CvPoint3D64f &_vecb)`
Returns the angle between `&_veca` and `&_vecb`.

`void calc_gaze( std::vector<GTCamera> &_cams,Table& _table)`
Calculates the gaze direction for each eye and the resulting gaze point on the display, see Section 4.2.2—"Gaze Direction".

`double finddet( double a1, double a2, double a3, double b1, double b2, double b3, double c1, double c2, double c3)`
Returns determinant of the matrix specified in row order.

`bool linsolve(double **A,double *l)`
Uses Cramer's rule to get the solution of a 3×3 LES.

`void matrix_mul( double **_A, double **_B, double **_out)`
Calculates $\_A * \_B = \_out$.

`bool point_nearest_to_planes(plane *_planes,int _numplanes, CvPoint3D64f &_point)`
Calculates the `&_point` which is nearest (in a least square error sense) to all of the planes in `*_planes`. It implements the algorithm from Section 4.2.1—"Cornea Center".

`double random_flat(double _a, double _b)`
Returns a uniformly distributed random number $\in \{\_a, \_b\}$.

`double random_gaussian(double _mean, double _dev)`
Returns a random number of a gaussian distribution.[3]

`void RotationMatrix( CvPoint3D64f &_rotate_around, double _angle, double **_matrix)`
Calculates rotation matrix for a rotation around a given vector with _angle.

`void RotationMatrix(double x_angle, double y_angle, double z_angle, bool _invert_rotation, double** _matrix)`
Calculates right-hand-sided rotation matrix for successive

[3] http://www.taygeta.com/random/gaussian.html

rotations around the origin, i.e. $R\_z * R\_y * R\_x(*p)$. If `_invert_rotation` is true, then the matrix is initialized with a z flip.POV-Ray uses a left-hand-side coordinate system so we have to flip in the case of POV-Ray-generated images.

`void unproject( GTCamera &_cam, CvPoint2D64f &_glint, CvPoint3D64f &_cam_to_glint)`
Unprojects the `&_glint`. This viewing ray from the camera center through the image plane is stored in `&_cam_to_glint`.

4.4.2 GazeTop.cpp

This is the main file, where the `main` function resides, the image processing is started and the results from the computations are visualized.

`void _tmain( int _argc, _TCHAR* _argv[])`

- connect to the cameras, read the scene from the .pov file, and start the main loop
  - set the right lighting scheme
  - acquire images from each camera
  - create a thread for each camera, and start the image processing for each one
  - wait till the threads are finished
  - call cornea center calculation for both eyes
  - call gaze calculation, then display the images with additional information (like a rectangle for the region of interest, a cross at the pupil center, and a circle around glints)

`void process_image( IplImage *_image,const int _camnumber)`
This function implements the algorithm discussed in Section 4.3—"Image Processing":

- calculate difference image from current and last frame
- detect the pupils (adaptive thresholding + blob detection)
- draw a region of interest (ROI) around them
- find the glints inside this ROI (`cvCanny` from OpenCV)
- for each glint store a plane for the cornea center calculation

OpenCV is used for the image manipulation, e.g. the `cvCanny` routine gets called. CvBlobsLib is used for the blob detection.

4.4.3 GTCamera.cpp

The main purpose of this class is to store camera-specific data. Apart from the usual constructors and destructors it contains the following routine:

`void GTCamera::calculate_focal_dist( double _angle)`
Calculate focal distance for a given horizontal opening `_angle` of the (POV-Ray) camera. Assumes 1280×1024 picture.

4.4.4 Image_processing.cpp

This files includes some self-written image processing functions.

`void adaptiveThreshold( IplImage *_src, IplImage *_dst, double _percentage)`
A histogram is calculated and every pixel darker than the top (`_percentage`) brightest pixels is set to 0.

`int find_pupil_pos( IplImage *_image, CvBox2D &_pupil)`

The function gets an image cutout that contains the eye. The images is inverted and thresholded, so the (once) dark pupil remains as a bright spot. Blob detection yields the position. The function draws a region of interest (`&_pupil`) around the pupil.

`void invertImage( const IplImage *_src, IplImage *_dst)`
Inverts the image: `_dst[x][y]=255-_dst[x][y];`

`void subtract( const IplImage *_src1, const IplImage *_src2, IplImage *_dst)`
Calculates pixel-wise difference : `_dst=_src1-_src2;`

4.4.5 Table.cpp

Stores real world coordinates of the display's corners p1 = top left, p2 = top right, p3 = bottom left. A rectangular display is assumed.

`void Table::calc_gaze_points( CvPoint3D64f &_gaze, CvPoint3D64f &_origin)`
The intersection between the gaze vector and the display is calculated — it implements algorithm 4.2.4— "Gaze on Screen". Stores the screen coordinate in `Table::impactpoint`.

`void Table::load_position_data(char *_filename)`
Loads the 3D position of the display.

Chapter 5

Evaluation

"Statistics will prove anything, even the truth."
–Noel Moynihan

Discrepancy between optic and visual axis raises problems

Although gaze tracking has been researched for over 50 years now, there is no calibration-free system that can accurately track several users who are, e.g., having a meeting in some gaze-enabled room. Even tracking the gaze of a single person is unreliable.
Current research argues that the main reason for this is that the optic axis of the eye (which goes through the pupil center and the cornea center) and the visual axis (which goes from the fovea through the cornea center) are slightly different, see Figure 3.13. Unfortunately, the former is the only thing one can find with image-based methods, and the latter is where the user is focusing at.
Even systems that measure user-specific parameters to calculate this difference angle are flawed. It is not possible to get an accuracy that is sufficient to replace the mouse as a pointing device. Therefore, the difference angle can not be the only reason for inaccurate gaze tracking systems.

We believe that an important and overlooked source of error is the calibration of the system, e.g., positioning of the cameras. We developed our own gaze tracking system to explore what impact calibration errors can have on the accuracy of the system.

5.1 Test Setup

First performance measure is not applicable to both test setups

To test the performance of our algorithm on the computer model, we compared the calculated focal point, as a result of our gaze tracking algorithm, with the one defined in our model.
The accuracy on the simple eye model was less than 0.5 cm average error in the focal point and slightly more (< 1 cm) for the more complex eye model.
To apply this performance measure to the real setup, we would need the precise position of an object the user is looking at. Unfortunately, we had no access to a 3D tracking system that is able to accurately measure the position of a fixation object.

Deviation angle is used as performance measure

For that reason, we decided to use another performance measure, which, in addition, is common in the gaze tracking community. We use the angle between two gaze points — the first being an initial reference value, calculated with no measuring errors, and the second being the one calculated with synthetic measuring errors.
This angle can be calculated for the computer model and the real setup. To make sure no problems arise with different lighting conditions, head movement, or other factors, still images were taken for the real setup.

We took images of nine different viewing directions (top-left, top-middle, top-right,..., to bottom-right gazing user). This was also done for the two computer models by modifying the focal point in the .pov file.
This way we ended up with three models: simple POV-Ray, complex POV-Ray, and real setup.

5.1.1 Independent Variables

We ran the program with the measured system calibration and stored the resulting gazepoint as a reference. We then changed a parameter every second frame and compared the resulting gazepoint to the reference value. The parameters in Table 5.1—"Independent Variables" were modified using gaussian distributed random numbers.

Parameter	Variation	Mean	Std. Deviation
camera (1..3) position	add vector (x,y,z)	0 mm	1 mm
off-axis LED (1..3) position	add vector (x,y,z)	0 mm	10 mm
camera (1..3) rotation	rotate around x-axis	0 °	0.5 °
camera (1..3) rotation	rotate around y-axis	0 °	0.5 °
camera (1..3) rotation	rotate around z-axis	0 °	0.5 °
camera (1..3) center point	scale by factor	1	0.01
camera (1..3) focal distance	scale by factor	1	0.01

Table 5.1: Parameter variations using gaussian distributed random numbers

To recall the coordinate system:

- The x-axis represents horizontal shifts,
- the y-axis represents vertical shifts,
- and the z-axis resembles the depth.

We also tested the dependency of the system on resolution. To simulate different resolutions, the image from the video camera was blurred using a gaussian smoothing filter. The size of the smoothing kernel was varied from 1 to 15, using a gaussian distribution.

The magnitude of the standard deviations was chosen according to what is possible to measure and what still gave reasonable results.
For each parameter, 100 random modifications were inserted and the result was measured.

5.1.2 Dependent Variables

For each parameter variation we got another gaze direction. We calculated the angle between this gaze direction and the reference gaze direction, which was calculated without synthetical errors. The bigger this angle is the greater is the impact of this parameter's variation.

5.2 Results

Change of resolution had the expected influence

We did not focus on the influence of different resolutions to the accuracy because the results met our expectations. As long as all glints are detected the impact is very low (less than $0.001°$).
If some glints are not visible, or the system is unable to detect them correctly, the gaze direction is far from accurate (more than $50°$).

Graphs are smoothed for better readability

To better interpret the results of the $21 \times 100 \times 9 \times 3 = 56700$ parameter variations we cluster them and remove outliers (i.e. results which are more then three times the standard deviation distant to the mean).
As a first test, we changed the parameter "camera (1..3) rotation" using still camera images from the real setup we get Graph 5.1. As we can see the impact of a wrong camera rotation matrix is quite huge.
Because scatter plots tend to get confusing with more and more scatters we will stick to line plots with a moving average filter (over five points).
As a first step, we compare whether the two computer models behave like the real setup. As we can see in Figure 5.2, our models resemble the real setup well — for rotation, camera translation, and several other parameters the figures look similar.

The slope of the error graph relates output to input error

We further investigated what parameter variation had greatest impact.
If we take a look at Figure 5.2 we can see that the data can

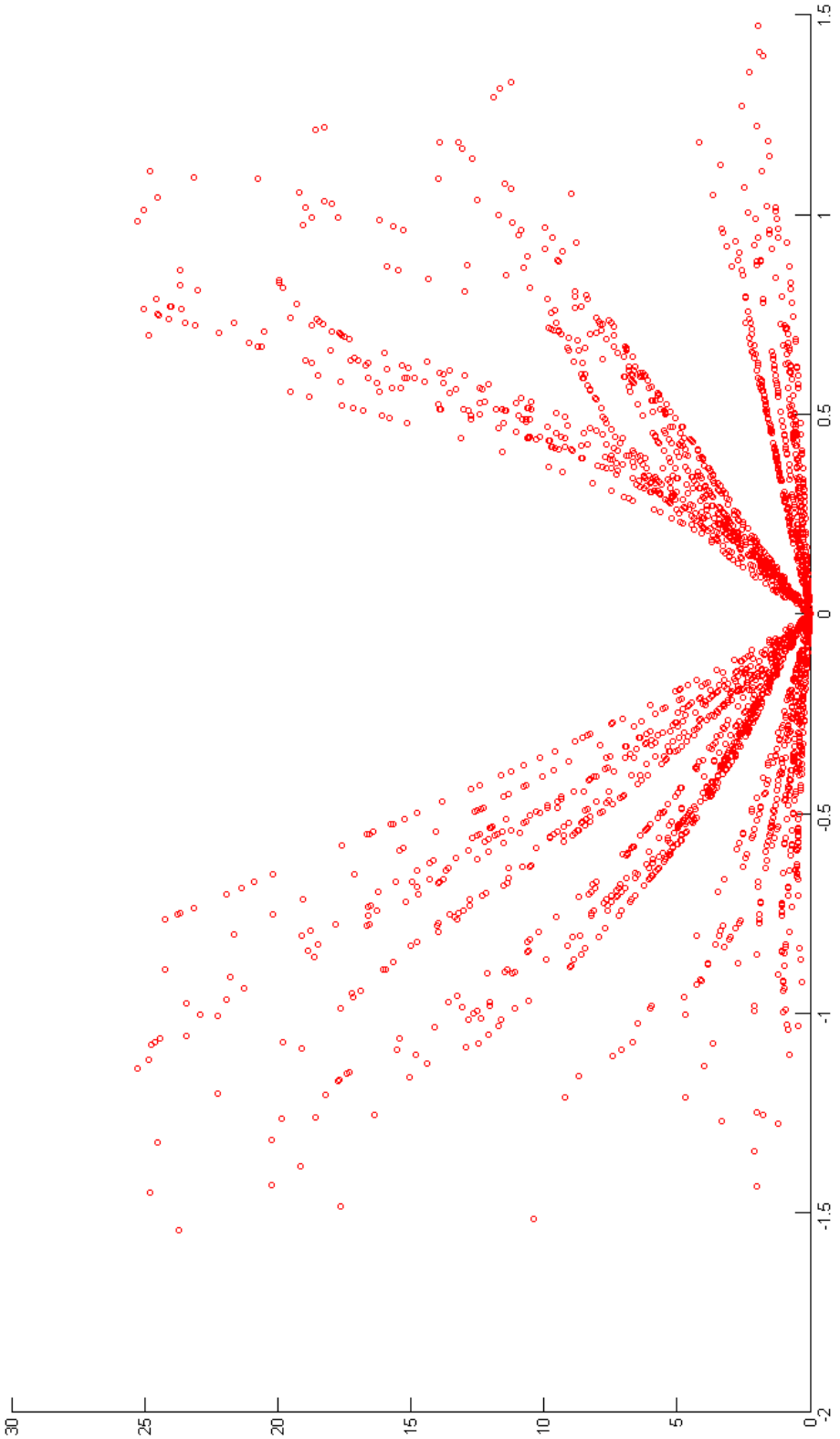

Figure 5.1: Camera rotation around x-Axis (°) vs. Error (°).
Test results for nine views and three different cameras

be approximated with a straight line $y = m * x + b$ (using linear least square error methods) quite well. Because the line has to go through the origin, b equals 0 and therefore the slope m is the ratio of error increase per parameter variation.
We calculated this ratio for each of the testbeds. The results can be found in Table 5.2.

Parameter	Unit	Pov1	Pov2	Real
camera (1..3) position	° error / mm translation	0.5802	1.696	0.4562
off-axis LED (1..3) position	° error / mm translation	0.01059	0.0193	0.03357
camera (1..3) center point	° error / ‰ increase	0.09926	0.3385	0.08600
camera (1..3) focal distance	° error / ‰ increase	0.02113	0.06496	0.03284
camera (1..3) x-rotation	° error / ° rotation	16.48	55.42	11.23
camera (1..3) y-rotation	° error / ° rotation	10.03	47.68	10.17
camera (1..3) z-rotation	° error / ° rotation	0.6025	2.053	0.2592

Table 5.2: Ratio of error increase per parameter variation

Results indicate that our initial hypothesis is right

Our hypothesis was that the types of error that "move" the key components of the gaze direction calculation the most will lead to the biggest errors in the accuracy of the system. The correct position of the LEDs is considerable less important than the correct measuring of the camera's position. Introducing a 1 cm error in our POV-Ray model gives $5.8°$ for the cameras and only $0.1°$ error for the LEDs.
The absolute value indicates that LED position accuracy is not the reason for a faulty system, whereas the correct camera position is crucial for high system accuracy.
The position of the LEDs is only used for the cornea center calculation, whereas the camera's position is used for cornea center and gaze calculation. A (real) translation of a LED by 1 cm would move its glint by some pixel at most, but a camera translation of 1 cm would translate the whole picture by a far greater value, due to the narrow angle of the camera.

Although center point and focal distance accuracy differ in their impact, they are both not sensitive enough to really affect the system's performance, supporting our initial hypothesis that intrinsic camera parameters are not the main problem.

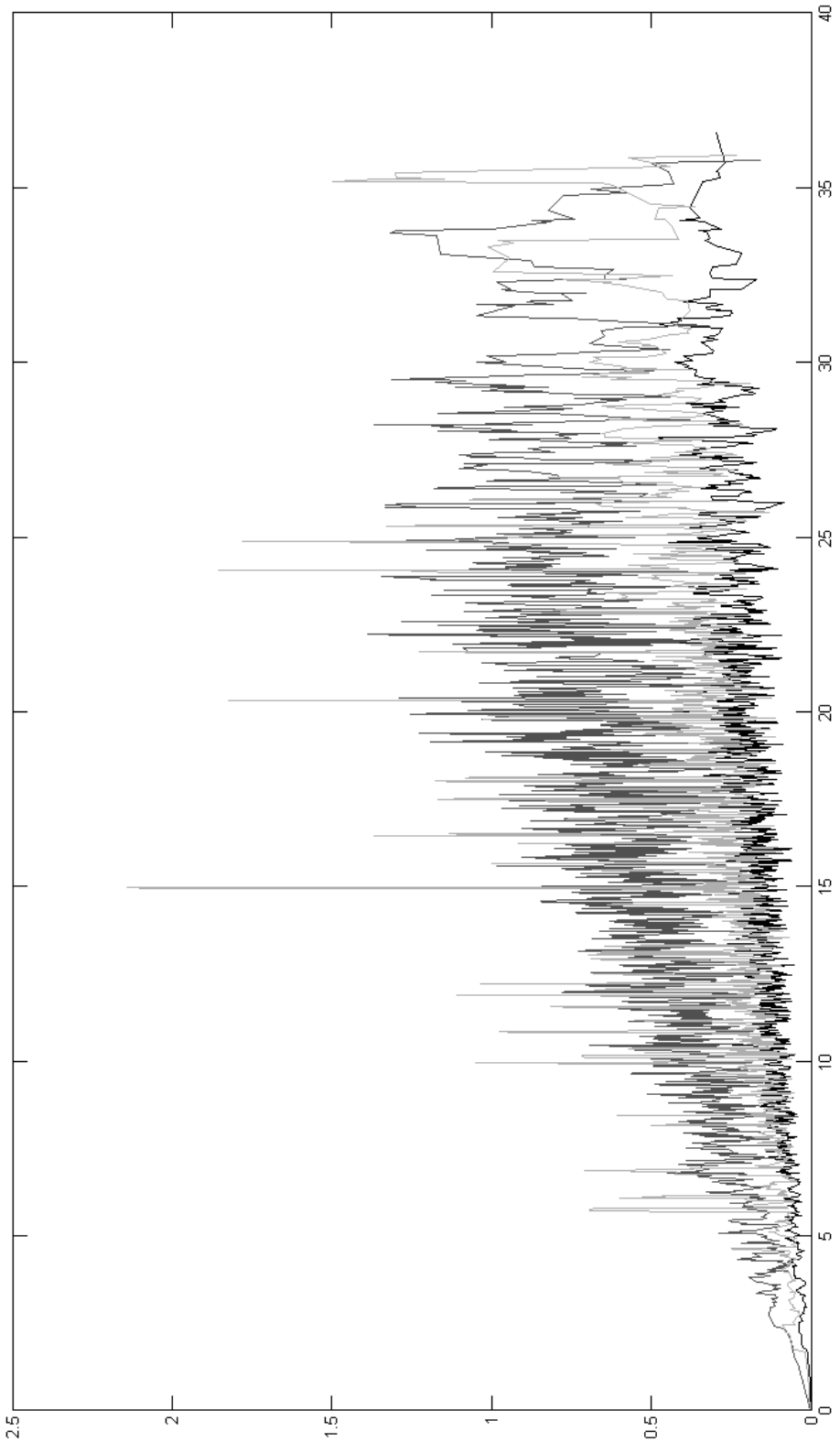

Figure 5.2: Off-axis LED translation (mm) vs. Error (°).
The computer models (black = simple POV-Ray, bright = complex POV-Ray) behave like the real setup (in grey).

For a serious error of more than 0.1° one would have to measure the focal distance or the center point wrong by several $‰$ of the initial value. For our setup, this would mean several pixel for the center point and several mm for the focal distance. However, both values can be calculated quite exactly.

Extrinsic camera parameters vastly influence the system's accuracy

We already saw that the correct position of the camera, as part of the extrinsic parameters, is crucial for the system's performance. Our results indicate, that the rotation matrices are even more important.
Even the slightest rotation around the x or y axis has a huge impact on the accuracy. Rotation around the z axis n is more fault-tolerant.

According to our eye model, two points played an important role when calculating the gaze — cornea center and pupil center. Both points where calculated with the data from all three cameras.
If an error exists in the extrinsic parameters the cameras do not see the same point at the same place (w.r.t. the global coordinate system). Obviously, the calculation of the gaze point based on this deviating position has to be wrong.

For further discussion we take a look at two cameras V_1, V_2 looking at the same glint. Unprojecting the 2D position of the glint yields two viewing rays r_1, r_2. Ideally they intersect at the real 3D position of the glint.
In practice they will not intersect at all in 3D. We see the 2D case of this problem in Figure 5.3. The higher the distance d from the real position to the intersection G of the cameras' viewing rays, the worse each calculation based on G must get. It should be clear that rotation and translation of the camera has a great impact on the distance d, and therefore on the overall calculation. This also explains that a rotation around the camera's z-axis, which is nearly parallel to the unprojections r_1 and r_2 respectively, does not have such a high impact as a rotation around one of the other axis.

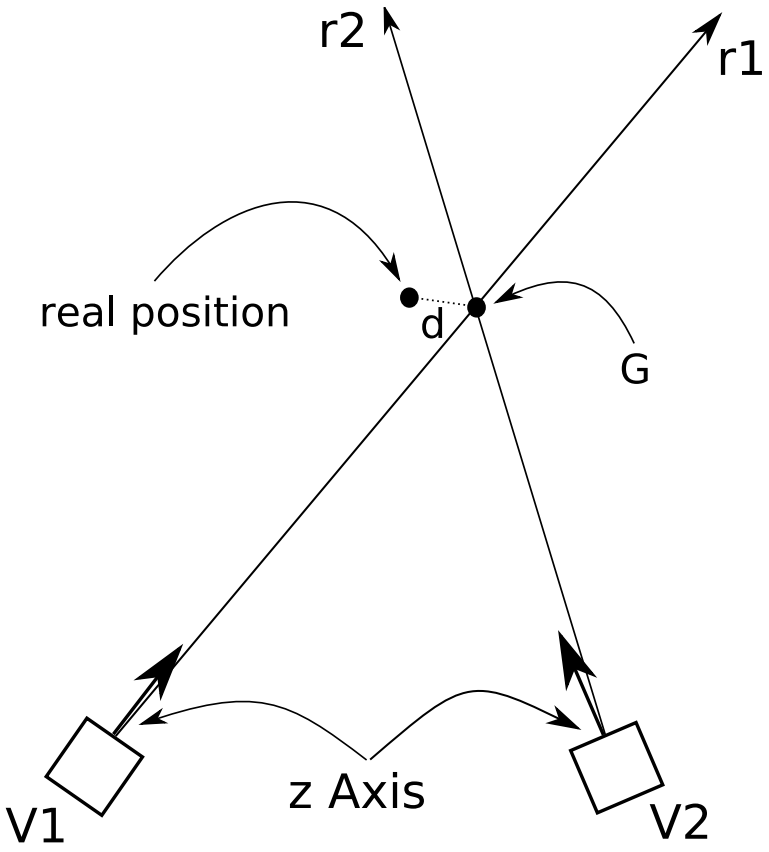

Figure 5.3: Unprojection of pixel positions does not always hit the object's spatial position due to imperfect calibration

5.3 Adaptive Calibration

Online calibration is added to the system

To further explore this problem, we developed an algorithm that tries to compensate for extrinsic parameter calibration errors, implemented it in our gaze tracking software, and compared it to our standard algorithm.
We reran all tests using the adaptive calibration and got encouraging results. Because the data is quite consistent (i.e., both computer models behave the same, the rotation variations (x,y,z) behave the same, etc.) we only present some typical results:

Adaptive calibration vastly reduces errors from some sources

As we can see in Table 5.3 the adaptive calibration works well for compensating camera translation errors and center

Variation	Unit	No Adapt	Adapt
camera (1..3) position	° error / translation in mm	0.5802	0.02617
center point scale	° error / ‰ increase	0.09926	0.003371
camera (1..3) x-axis rotation	° error / ° rotation	16.48	19.66

Table 5.3: Comparison of performance with and without the adaptive calibration for the simple POV-Ray model

Variation	Unit	No Adapt	Adapt
camera (1..3) position	° error / translation in mm	0.4562	1.244
center point scale	° error / ‰ increase	0.086	0.2147
camera (1..3) x-axis rotation	° error / ° rotation	11.23	95.32

Table 5.4: Comparison of performance with and without the adaptive calibration for the real setup

point errors on the computer models. Unfortunately it cannot help with rotation errors.

Adaptive calibration does not always help

Taking a look at Table 5.4, that shows the perfomance of the system on the real setup, we see another trend: the calibration is not helping at all. The authors believe that although the calibration of the real setup has been done very accurately, it still has some errors in it that prevent the adaptive calibration to work like it should (and as it does in the case of perfect calibration data). Further discussion of this point is subject of future work.

Chapter 6

Summary and Future Work

In this final chapter we will summarize our research, state the contributions to the gaze tracking community, and give an outlook on future research ideas.

6.1 Summary and Contributions

Lack of accurate user calibration-free gaze tracking systems

We started with an introduction to the field of eye tracking. We showed what recent eye tracking systems are capable of and that a calibration-free system with sufficient accuracy for post-desktop interactions has not yet been built.
To explore interactions beyond pointing through gaze or gaze-enhanced video conferencing the system must meet the following requirements:

- allows free head movement
- accurate tracking
- real time processing of input images
- user calibration-free

To find out why current systems do not fullfill these requirements, we developed a comparable gaze tracking

system and thoroughly evaluated its performance.

We based our work on the proposal by Shih et al. [2000] and hardened their approach using least square error methods and multiple cameras. We implemented this approach and built a system with three cameras and six light sources.
To evaluate the system's behaviour and find reasons for the absence of a system with the above mentioned capabilities, we fed synthetical errors (e.g., add noise to the images) into the system and measured its fault-tolerance.

The error ratio allowed a comparison of the error sources

We calcaluted the ratio between each error input and its resulting error. This way we were able to quantitatively predict the influence of an error and to sort the error sources by their damage potential. Using this ratio we discovered that especially the extrinsic parameters of the cameras (i.e., position and rotation) have to be measured very precisely or the system will always produce errors, like it does in every gaze tracking system up to now — even after calibration to the user.

Adaptive calibration can improve the accuracy

To compensate for errors due to less than perfect system calibration we designed and implemented an adaptive online calibration algorithm. It uses the images from the camera to calculate more fitting rotation matrices for the cameras. This approach was tested and it was shown that it was able to reduce or even nullify some kinds of errors.

As we can see from our test results, it is difficult to measure the system setup well enough to achieve very good (i.e., way less than 1°) tracking results, even in a highly controlled research setup. As can be seen with our adaptive calibration algorithm, it is possible to use the images from the cameras to improve upon an initial calibration.

The main contributions of this thesis to the gaze tracking research community are:

- our gaze tracking system, which can support an arbitrary number of cameras and light sources

- our evaluation method and its results
- our adaptive online calibration algorithm

As a benefit of this thesis, future researchers know that their system calibration and especially a precisely measured orientation of the cameras is crucial for accurate gaze detection. This insight should be applicable to every research field that attempts to calculate 3D data based on 2D camera images. These include 3D model reconstruction, geodesy, motion capturing, and so forth.

6.2 Future Work

It should be possible to expand the current calibration algorithm to compensate for more types of calibration errors. There are several real-time 3D tracking algorithms that could be combined with our gaze tracker to calibrate the system during its use. With such a self-calibrating and user calibration-free gaze tracking system, more interesting interactions could be explored.

Another research idea would be to apply our research method to other 3D tracking systems like motion capturing or 3D model reconstruction.

One could setup the system using several cameras in an ordinary conference room and analyse the gaze direction of the users. Because most people look at the people they are talking to, it would be possible to enrich a video of the meeting with this metadata for easier browseability.

To use gaze trackers in public places at a long range, it would be necessary to have a lot more than three cameras. It would be interesting how to solve the correspondency problem, because the cameras initially do not know what persons they are looking at and which light source generated the glints on the person's eye.

With our results, future researchers will be able to explore new interactions, because they know what kind of factors negatively influence the performance of their gaze tracking system and can take care to circumvent these problems.

Appendix A

The Pinhole Camera Model

The pinhole camera model is the standard model for the generation of 2D images from 3D scenes (see Figure A.1). It can also be used to calculate 3D viewing rays from 2D image points.

Let $[P]_G = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$ be a three dimensional point given in the global coordinate system G.

To "see" this point the camera is rotated and translated from its initial position. Because we want the camera's position to be static we instead translate and rotate the point.

$$[P]_L = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = R \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + t$$

with R being a 3×3 rotation matrix and t a 3×1 translation vector. We now have $[P]_L$ given in the local coordinate system of the camera. The so called extrinsic parameters (rotation and translation) are saved for further point conversions from the global to the local coordinate system. The point is then projected onto the image plane, which is defined to be at $z = 1$ (w.r.t. the local coordinate system) using perspective transformation, which is a simple divison by z in our case.

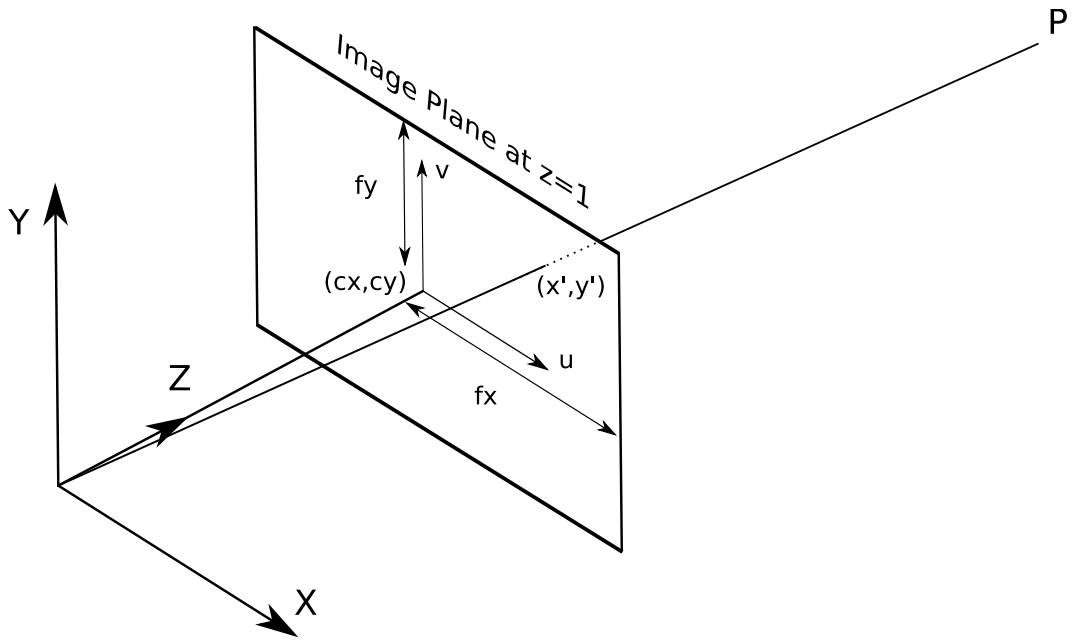

Figure A.1: The pinhole camera model

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x/z \\ y/z \\ z/z \end{pmatrix}$$

To map this point onto a pixel in the final image we need the focal distances f_x, f_y, expressed in pixel-related units, and the center position (c_x, c_y) of the image. We then scale by the focal distance, shift by the center,

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_x * x' + c_x \\ f_y * y' + c_y \end{pmatrix}$$

and yield the coordinates of the point projection in pixels.

Appendix B

CD Contents

On the enclosed CD you will find two folders “GazeTop” and “Tests” and a pdf of this thesis. The first folder contains the gaze tracking algorithm written in C++ using Visual Studio 2005 including some sample images. The second folder contains our test results, some matlab skripts to analyze the textfiles, and images of the evaluation graphs.

Bibliography

David Beymer and Myron Flickner. Eye gaze tracking using an active stereo head. In *CVPR (2)*, pages 451–458, 2003.

J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986. ISSN 0162-8828.

Andrew T. Duchowski. A breadth-first survey of eye-tracking applications. *Behavior Research Methods, Instruments and Computers*, 34(4):455–470, 2002a.

Andrew T. Duchowski. *Eye Tracking Methodology: Theory and Practice*. Springer, November 2002b.

Yoshinobu Ebisawa. Unconstrained pupil detection technique using two light sources and the image difference method. *Visualization and intelligent design in engineering and architecture*, II:79–89, 1995.

Editure. Worksheet 2: The human eye - structure and function, 2007. URL `http://www.schools.net.au/edu/lesson_ideas/optics/images/eye_structure.gif`. [Online; accessed 03-November-2007].

E.D. Guestrin and M. Eizenman. General theory of remote gaze estimation using the pupil center and corneal reflections. *IEEE Trans Biomed Eng*, 53(6):1124–33, 2006.

Berthold Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4:629–642, 1987.

Robert J. K. Jacob. The use of eye movements in human-computer interaction techniques: What you look at is what you get. *ACM Trans. Inf. Syst.*, 9(2):152–169, 1991.

Ravi Kothari and Jason L. Mitchell. Detection of eye locations in unconstrained visual images. *IEEE International Conference on Image Processing*, III:519–522, 1996.

Dongheng Li, D. Winfield, and D. J. Parkhurst. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. volume 3, pages 79–79, 2005. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1565386`.

Paul P. Maglio, Teenie Matlock, Christopher S. Campbell, Shumin Zhai, and Barton A. Smith. Gaze and speech in attentive user interfaces. In *ICMI*, pages 1–7, 2000.

Carlos Hitoshi Morimoto, David Koons, Arnon Amir, and Myron Flickner. Pupil detection and tracking using multiple light sources. *Image Vision Comput.*, 18(4):331–335, 2000.

Clifford Nass, Youngme Moon, B. J. Fogg, Byron Reeves, and D. Christopher Dryer. Can computer personalities be human personalities? In *CHI 95 Conference Companion*, pages 228–229, 1995.

Alice Oh, Harold Fox, Max Van Kleek, Aaron Adler, Krzysztof Gajos, Louis-Philippe Morency, and Trevor Darrell. Evaluating look-to-talk: a gaze-aware interface in a collaborative environment. In *CHI Extended Abstracts*, pages 650–651, 2002.

Takehiko Ohno and Naoki Mukawa. A free-head, simple calibration, gaze tracking system that enables gaze-based interaction. In *ETRA*, pages 115–122, 2004.

Takehiko Ohno, Naoki Mukawa, and Atsushi Yoshikawa. Freegaze: a gaze tracking system for everyday gaze interaction. In *ETRA*, pages 125–132, 2002.

Kenichi Okada, Fumihiko Maeda, Yusuke Ichikawa, and Yutaka Matsushita. Multiparty videoconferencing at virtual social distance: Majic design. In *CSCW*, pages 385–393, 1994.

Derrick Parkhurst and Ernst Niebur. A feasibility test for perceptually adaptive level of detail rendering on desktop systems. In *APGV '04: Proceedings of the 1st Symposium on Applied perception in graphics and visualization*,

pages 49–56, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-914-4. doi: http://doi.acm.org/10.1145/1012551.1012561.

Polhemus. Polhemus fastrak: The motion tracking industry standard, 2007. URL `http://www.polhemus.com/?page=Motion_Fastrak`. [Online; accessed 03-November-2007].

Keith Rayner. The perceptual span and peripheral cues in reading. *Cognitive Psychology*, 7(1):65–81, 2000.

Jun Rekimoto and Masanori Saitoh. Augmented surfaces: A spatially continuous work space for hybrid computing environments. In *CHI*, pages 378–385, 1999.

Abigail J. Sellen. Speech patterns in video-mediated conversations. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 49–59, New York, NY, USA, 1992. ACM Press. ISBN 0-89791-513-5. doi: http://doi.acm.org/10.1145/142750.142756.

Jeffrey S. Shell, Roel Vertegaal, and Alexander W. Skaburskis. Eyepliances: attention-seeking devices that respond to visual attention. In *CHI Extended Abstracts*, pages 770–771, 2003.

Sheng-Wen Shih, Yu-Te Wu, and Jin Liu. A calibration-free gaze tracking technique. In *ICPR*, pages 4201–4204, 2000.

J. David Smith and T. C. Nicholas Graham. Use of eye movements for video game control. In *Advances in Computer Entertainment Technology*, page 20, 2006.

John D. Smith, Roel Vertegaal, and Changuk Sohn. Viewpointer: lightweight calibration-free eye tracking for ubiquitous handsfree deixis. In *UIST*, pages 53–61, 2005.

Rainer Stiefelhagen, Jie Yang, and Alex Waibel. A model-based gaze tracking system. *International Journal on Artificial Intelligence Tools*, 6(2):193–209, 1997.

Vildan Tanriverdi and Robert J. K. Jacob. Interacting with eye movements in virtual environments. In *CHI*, pages 265–272, 2000.

Roel Vertegaal. The gaze groupware system: Mediating joint attention in multiparty communication and collaboration. In *CHI*, pages 294–301, 1999.

Roel Vertegaal, Gerrit C. van der Veer, and Harro Vons. Effects of gaze on multiparty mediated communication. In *Graphics Interface*, pages 95–102, 2000.

Xuuk. eyebox2, 2007. URL `http://www.xuuk.com/Products.aspx`. [Online; accessed 03-November-2007].

Shumin Zhai, Carlos Morimoto, and Steven Ihde. Manual and gaze input cascaded (magic) pointing. In *CHI*, pages 246–253, 1999.

Zhengyou Zhang. Video-based eyetracking methods and algorithms in head-mounted displays. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

Zhiwei Zhu and Qiang Ji. Eye and gaze tracking for interactive graphic display. *Mach. Vis. Appl.*, 15(3):139–148, 2004.

Index

Typeset November 5, 2007