High Speed Eye Tracking Using The Vision Chip

Master Thesis

by

Björn Werkmann

Submitted to the Department of Computer Science at the Technical University of Darmstadt, Germany

written at the Ishikawa Namiki Komuro Laboratory,
Department of Information Physics and Computing, Graduate School of Information Science and Technology

University of Tokyo, Japan

Tokyo, August 2005

Supervisor: Prof. Dr. O. v. Stryk

This work has been written by myself, independently, solely based on sources available at the department and mentioned in the text. It is submitted to the Technical University of Darmstadt only.

Toyko, August 30, 2005

Contents

Abstract

This work demonstrates the implementation of a high speed eye tracking system capable of tracking saccadic eye movement. Setting out to capture saccades with velocities of up to $700^o/sec$ and durations as short as 20ms, the current system provides a frame rate of 100Hz. To achieve this frame rate, the system utilizes low cost, small size tracking hardware: the Vision Chip.

The system tracks eye movements based on the pupil position. Segmentation to isolate the pupil is mainly performed by binarization, and the position is determined based on *moments* of the pupil area within the binary image. The main problem solved in this work, is to avoid cluttering of the binary image due to binarization artifacts, and to detect distortions of the pupil that affect the accuracy of the positional information. A natural cause for distortions is the occurrence of eye blinks, others are caused by the variation of the appropriate binarization threshold for different eye positions.

To deal with these problems in realtime, the tracking is preceded by in depth analysis of grayscale images of the eye, to locate the pupil and to determine a pupil related feature vector, including a binarization threshold. This information is acquired for several eye positions, where each threshold is *ideal* for the position, i.e., few binarization artifacts appear.

Using these thresholds, the binarization threshold can be dynamically adjusted during tracking to reduce the amount of clutter. To detect distortions, the system also computes a vector of pupil related features at runtime. Using the precomputed features as constraints, distortions can be detected by comparison.

This is achieved with the required frame rate, by taking advantage of special Vision Chip functions that support moment calculations. The thesis demonstrates the calculation of moments up to the second order and gives *area* and *eccentricity* as examples for features, the eye position is computed as the pupil *centroid*.

These positions are available in realtime or can be stored for off-line analysis. They are provided as 2D positional information that can optionally be mapped into screen coordinates to acquire *Point Of Regard* information. The positional resolution has been estimated to be between $<0.14^o$ and $<2.3^o$ over a range of 45^o.

Chapter 1

Introduction

This master thesis describes the implementation of a high speed eye tracking system based on special purpose tracking hardware, the Vision Chip.

The following section explains the need for an eye tracking system with higher than usual video frame rates. Section 1.2 describes eye tracking systems that already provide such frame rates and related research in general. Next, a background section provides information that is relevant for the understanding of the presented system. The final section gives an overview of the remaining thesis.

1.1 Motivation

In general, the purpose of an eye tracking system is, simply stated, to monitor eye movements and report the state of the eye as accurately as it is necessary for an application. The purpose of this thesis is to demonstrate the application of the SR3300 Vision Chip to the task of tracking eye movements, taking advantage of its high frame rates.

A possibly reported information is the point of regard (POR), i.e., 2D positional information as to what the user is regarding in a given plane such as the computer screen. The presented system provides an approximation to the POR based on the position of the pupil in the image plane of the Vision Chip.

The high speed requirement as it is stated in the thesis title, is justified by the fact that eye movements are by no means as smooth as they appear to us from our viewing experience. Large parts of the movement are determined by saccades, high velocity changes of our visual focus, with durations of few tens of milliseconds [1, 2] and velocities of up to $700^o/sec$ [2]. Because of the resulting high speed changes of the pupil position, high speed eye tracking is necessary in order to accurately capture the position and in particular to capture the occurrence of saccades. Timely accuracy of such scale is demanded by existing, diagnostic [3] and interactive applications [4, 5]. Many applications analyzing visual and cognitive processes, do not require the exact trajectory during a saccade, as little visual processing is performed at this time. Nevertheless, awareness of saccades is still important, to, e.g., reduce the size and complexity of eye movement protocols [6].

Although sufficient for many applications, eye tracking systems that capture

the image of the eye with normal video frame rates around 50Hz, cannot capture saccadic eye movement. For purposes where the eye movement has to be monitored accurately during the performance of a saccade, or when the beginning of a saccade has to be detected and reported as soon as possible, such systems are not sufficient, as large parts of the eye movement are missed between consecutive frames. Also, For gaze-contingent displays [4] or innovative display techniques [5], where saccades are used to trigger the update of a display, the delay between occurrence and reporting of a saccade is crucial and affects overall system performance. For gaze-contingent displays in particular, shorter delays have been shown to be advantageous and allow for greater flexibility in choosing system parameters [4].

Existing high speed tracking systems provide saccadic eye tracking at rather high expense, e.g. [7, 8], which makes them unavailable for some applications. While these systems usually provide features that justify the cost of several $10,000, some applications might suffice with a simpler system. For example high accuracy and a wider variety of tracked parameters, as well as a suite of software tools for data analysis might not be needed. For applications that do not require maximum spatial accuracy but require a high frame rate, the presented system provides an alternative at lower cost.

Furthermore, the presented system does not require any changes to the host workstation beside the installation of a Universal Serial Bus (USB) device. Other systems usually require integration of special hardware extensions into the workstation. While this is also a reason for their increased accuracy and larger set of features, some applications such as wearable computer systems that integrate eye tracking [6], might be able to take advantage of a tracking system available as USB device.

The use of the Vision Chip, with matchbox size dimensions, also recommends the system for applications with tight requirements regarding size and weight. Further miniaturization is rather a question of improving the production process of the chip and does not require principle changes. Unobtrusive integration into normal eye glass frames would be possible. The tracking system has been designed with such applications in mind and the use of additional hardware, e.g. a half transparent mirror for better illumination, has been avoided. The current system closely resembles such attempts for integration.

1.2 Related Work

As described in Section 1.3.2 there are many different techniques for tracking eye movements and just as many different systems. In the past, the developments of eye tracking system was directly inspired by the need coming out of research projects that require information about eye movements [9, 10]. Today, companies provide ready solutions, and many different companies provide varieties of systems for different applications. A listing [11] from August 2001 that is still quite accurate, gives a total of 27 vendors of eye tracking systems. A variety of tracking techniques are used by these systems. The different techniques are described in detail in Section 1.3.2.

Regarding the achieved frame rates, Dual Purkinje eye tracking systems mark

the top of the field with 4000Hz. While these system are leading in most aspects, the second purkinje image is of low intensity and can be problematic to capture, especially under difficult lighting conditions. This is one of the reasons why the presented system does not use this technique. Search coil based systems provide frame rates up to 1000Hz but cause substantial inconvenience that is only acceptable in laboratory settings. IROG based systems provide frame rates up to 360Hz but do not cope well with eye blinks.

Most of the system, as well as the presented work, are based on video images of the eye. An interesting approach is presented by [12] that uses images that have a similar resolution as the images used by the presented systems, but that uses artificial neural networks and achieves a frame rate of 15Hz. The accuracy is limited to 1.5-2^o. A very robust way of locating the pupil is implemented in the system presented by [13]. Two infrared light sources, one located close to the optical axis of the eye, are used in turn to illuminate the eye. The system takes advantage of the retro-reflectivity of the eye, that causes a very bright pupil for the image illuminated with the on-axis source. Subtraction of both images yields an image containing only the pupil area. Tracking the position of the pupil, the system provides eye movement data with normal video frame rates. The system presented by [14] focuses on the hardware that is necessary to build eye tracking headgear and gives valuable instructions regarding safety and usability.

Concerning video based, or VOG (Video Oculography), systems that provided frame rates as they are aspired by the presented work, the listing from 2001 gave the Alphabio Eyeputer with 480Hz [15] as the fastest system. Today, this system is surpassed by the EyeLinkII with 500Hz [16]. The Chronos Eye Tracker [17, 18] provides a frame rate of 400Hz.

All these systems have been made possible by the advent of programmable CMOS image sensors. Only these sensors made saccadic eye tracking achievable using video based methods. Facilitating such special purpose hardware, parts or all of the image analysis can be performed by the image sensor at the required frame rate. This way, the data load can be largely decreased as only relevant data has to be processed by the workstation. A common approach is to have the CMOS sensor identify a region of interest (ROI), e.g., a small area containing the eye or pupil, that is subsequently transferred to the host workstation for further analysis. As the data load is decreased the frame rate can be increased accordingly.

These VOG systems are capable of tracking 3D eye coordinates, including the torsional component around the pupil visual axis in addition to the translational components. The translational components are given in degrees of rotation around the remaining two axes of the eye. As all systems are video based, the rotational angles have to be determined using 2D features extracted from the image of the eye. The x and y position of the pupil center as it is reported by the Vision Chip Eye Tracker can be used for this purpose.

The torsional component is commonly computed based on the grayscale signature of a circular strip within the iris. As this feature could not be extracted in high enough speed and quality using the Vision Chip, the torsional component is not reported by the presented system.

Table 1.1 gives the core performance values for the mentioned systems in com-

parison to the Vision Chip Eye Tracker. The column *Mobility* summarizes size and power consumption of the eye tracking device. All systems but the presented work are intended to be used solely in a stationary environment. The range for the VC resolution is explained in Section 5.1.

System	Frame rate	Resolution	Price	Mobility
EyeLinkII	500Hz	$<0.01^o$	\$80,000	stationary
Eyeputer	480Hz	$<0.016^o$	\$36,000	stationary
Chronos Eye Tracker	400Hz	$<0.1^o$	\$17,000	stationary
Vision Chip Eye Tracker	100Hz	between $<0.14^o$ and $<2.3^o$	\$1,000	mobile

Table 1.1: Comparison of the Vision Chip Eye Tracker to the leading high speed eye tracking systems.

Although the presented system is no competition in these areas, price and the possibility for miniaturization can make it an alternative.

All mentioned systems require the installation of PCI (Peripheral Component Interconnect) extensions into the host workstation to achieve real time availability of the tracking data. Due to the system architecture of the used Windows system, the presented work is currently limited in this respect. This is addressed in more detail in Section 4.1

1.3 Background

1.3.1 The Eye, Eye Movements And Eye Tracking

> "If we were to order a moveable electronic eye from an engineer, we might think it a little odd if its lens turned out not to be lined up with its primary axis, if it wobbled as it rotated, and if the mechanism for elevating its gaze also made it twist to one side. Yet all of these are features of the eyes that nature has given us." [2]

Such are the features of the eye that affect the task of tracking eye movements, determining the position of the eye and inferring the point or object regarded. In building a system for this task, it is necessary to simplify and use workable models of the eye. It is important to judge these simplifications and the performance of a system for a given application, and regard every measured characteristics separately [2].

Different applications may differ in how closely the tracking system has to adhere to the actual model of the eye or its movements. Some information might not even be available if the most precise model available is considered. For example, neurological effects bias the relation between monitored eye position and regarded object [2]. These effects cannot easily be accounted for by means of measurement made by common tracking device without monitoring neural activity as well.

An example that can be personally verified is the *spotlight metaphor* [19]. These findings show that, very similar to the sweep of a spotlight, we are able to move

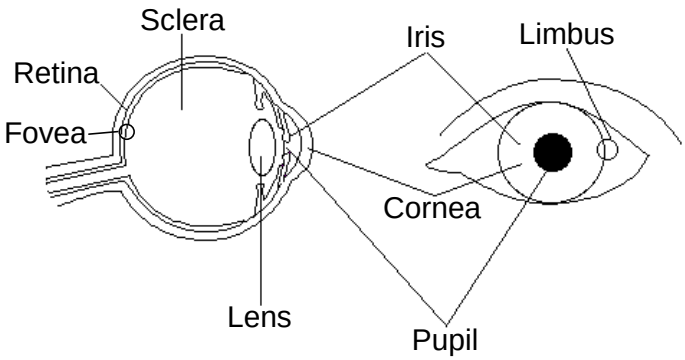


Figure 1.1: Eye anatomy [20].

our attention within our field of view, while constantly focusing on a fixed location. Hence, with a fixed eye position, we are regarding different objects.

Luckily, many of these effects are usually small and can be neglected for many applications [2].

Eye Terminology. To talk about the eye, it is necessary to introduce some common terms [21] to refer to different parts of its anatomy. Figure 1.1 depicts a cross-section and a frontal view of the object of interest.

The sensing part of the eye is the *retina*. It lines the back of the eye and contains photoreceptors that relay information about incident light to the brain. A particular area of the retina is the *fovea*. It is of about 2mm diameter and is the most sensitive area that provides most acute vision.

From the anterior (front) of the eye, incident light first passes through the *cornea*, the first refracting surface on the way to the retina. In the cross-section it is visible as a slight bulge with a greater curvature than the rest of the eye. The light then passes through the *pupil*, a circular hole whose size can change to regulate the intensity of incident light. The area surrounding the pupil is the *iris*.

The white area that makes up the most of the visible surface of the eye ball is the *sclera*. In the front view, a thin area has been marked that connects the cornea to the sclera, the *limbus*.

Viewing. When we are reading, viewing the symbols on a computer screen or regard any other scene, our eyes are constantly moving. Similarly, our eyes move when we are fixating on a point while we are moving our head. The purpose of this movement is to position the image of the object we are regarding on the fovea.

Eye Movements. These movements are brought about by the muscles that are attached to the eye ball as depicted in Figure 1.2. These rotate the eyes and thus determine the nature of the rotation. While a technical system would usually prefer a fixed center of rotation for the eye ball, the actual center moves along a line in space, the *space centrode*. Over, e.g., the whole range of horizontal movements, the center has been measured to move about 2mm [2].

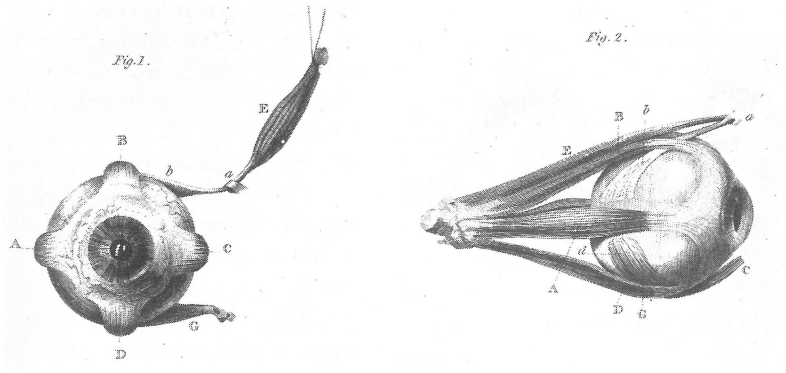


Figure 1.2: Eye muscles responsible for the movements of the eyes [2].

Saccades. From our own viewing experience it seems that our eyes are rather smoothly scanning our environment. This is by no means the case. An important aspect of our eye movements are *saccades*, which are movements of short duration and very high velocity. These movements are ballistic, i.e. cannot be changed while they are performed. Their purpose is to make the image of the regarded object fall on the fovea. This is usually achieved by a larger movement and subsequent *correctional saccades*, that position the image more accurately.

A saccade can reach velocities of up to $700^o/sec$ and be as short as 20ms in duration [2]. Both velocity and duration vary with the rotational angle, or *amplitude* of the saccade. The peak velocity of a saccade increases exponentially with the amplitude, while the duration increases linearly with the amplitude.

A saccade of large amplitude can have a duration above 200ms while the noted majority has a duration below 75ms. The highest velocities are only found in saccades with the largest amplitude. A large saccade has an amplitude of about 90^o, while 85% of all saccades are rather below 15^o [2].

Speed Profile. Figure 1.3 shows the speed and position of the eye over the course of a saccade. The visible symmetry can be found with all saccades. It is particularly relevant that the maximum velocity is reached half way through the saccade. This fact can be exploited by an eye tracker to detect saccades by monitoring the speed profile.

Frequency Of Saccades. Before further evaluating the impact of saccades for tracking devices, a short note should be given on saccades in a larger context, i.e., as part of a sequence of several saccades. Here, two characteristics of saccades are important. The *latency* and the *refractoriness*. These can be seen as delays before and after the saccade, respectively. The latency is the delay between the, or *entrance pupil* stimulus that triggers the saccade and the actual onset of the saccade [2]. The refractoriness gives the minimum time by which saccades can follow each other. These times are rather long in comparison to the short duration of saccades. As a consequence, about 3 saccades per second is a common value [1], which is a very low frequency or long period of approximately 300ms.

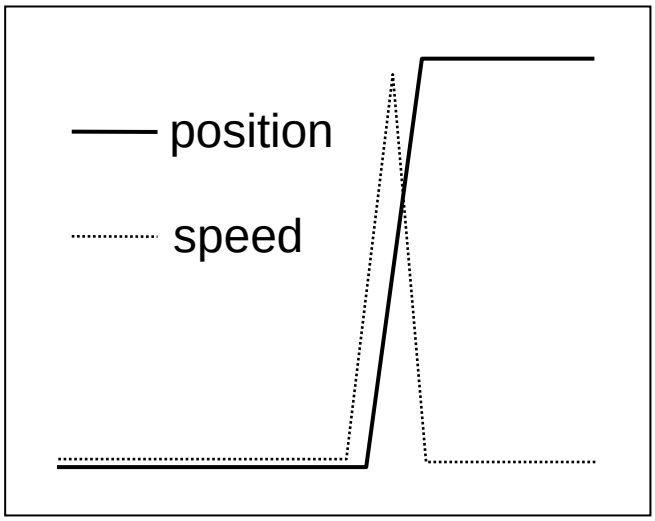


Figure 1.3: Eye position and speed over time. During saccades the speed shows the depicted symmetrical profile. The positional information is on logarithmic scale, hence linear despite the linear increase and decrease in speed [1].

Impact Of Saccades. When building an eye tracking device, the impact of saccades has to be considered. In tracking devices that use contact lenses for tracking, the lens might even slip due to the quick acceleration of the saccade [2].

While image based systems do not have this kind of problem, saccadic eye movements have to be accounted for in respect to the sampling rate. Depending on the application of the eye tracker, the velocity of saccades is of more or less importance. Because of the mentioned latencies of saccades for example, the eye position hardly changes for a long time between saccades. Therefore it is quite possible to acquire usable results with normal video frame rates below 50Hz.

If the application requires detection of saccades or positional information even during the course of a saccade, normal video frame rates are not sufficient. To detect "most saccades" a system with 250Hz should be used [3].

1.3.2 Overview Of Eye Tracking Techniques

Recordings of eye movements have been made as early as 1936 [22], but the technique used for the recordings only gradually improved. In particular since the availability of video cameras, the non-intrusiveness of tracking systems has substantially improved.

Since the beginning, a great variety of systems have been developed. According to [23] there are four broad categories to classify measurement methodologies for eye movements:

1. Electro-oculography (EOG):
   Involves measurement of electrical potential at the skin around the eye.

2. Scleral contact lens/search coil methods:
   Involves measurement of the position of a special contact lens inserted into the user's eye. The measurement can be mechanical, electromagnetic or visual. This is the most precise method, allowing a precision as fine as 10 arcseconds.

3. Photo-oculography (POG) or video-oculography (VOG):
   Involves photo or video image analysis for "the measurement of distinguishable features of the eye under rotation/translation, e.g., the apparent shape of the pupil, the position of the limbus (iris-sclera boundary), and corneal reflections of closely situated directed light sources."

4. Video-based combined pupil and corneal reflection methods:
   Involves video image analysis to detect two points of reference, e.g., pupil center and the corneal reflection of a stationary light source. The eye position is determined based on the difference between these points. As the difference remains relatively constant with "minor head movements" but changes with eye movements, the eye position can be measured relative to the source or a computer screen, respectively. The position within the screen is called POR (Point of Regard).

As wearing contact lenses is quite inconvenient, few current eye tracking systems belong to category two. Most current eye tracking systems are less invasive by relying only on video image. The majority of these systems belongs to category four [23]. The main advantage in comparison to systems in category three is the increased convenience when measuring the POR.

VOG techniques require the user's head to be fixed at a certain location or involve additional measurements, to determine the head position and orientation. For the presented system, the user's head is expected to be at a fixed location. For this reason, and because it does not use any corneal reflection it belongs to the VOG systems in category three.

The following gives a more detailed description of video based tracking techniques [22]:

Limbus Tracking. This technique determines the eye position based on the position and shape of the limbus relative to the head. As the contrast between the white sclera and the dark iris is usually large, this feature is easy to detect. A disadvantage is the rather large size of the iris which leads to frequent occlusion by the eye lids. This is in particular problematic for the tracking of vertical eye movements. The user's head has to be fixed in a position.

Pupil Tracking. Very similar to the previous technique in most other respects, this technique copes better with vertical eye movements due to the smaller size of the pupil. A disadvantage is the lower contrast between pupil and surrounding iris.

Corneal and Pupil Reflection Relationship. This technique uses the reflection of an infrared light source from the cornea, the *glint*, and the retro-reflectivity of the eye. The latter causes the retina, which is visible through the pupil, to appear as a bright disk in a captured video image. The relative position of the pupil in respect to the highlight is used for measuring eye movements. As already explained this technique allows good usability due to free head movements. Naturally this is also a restriction as the image of the eye cannot be captured if larger head movements occur that leave the field of view of the camera. An additional problem is caused

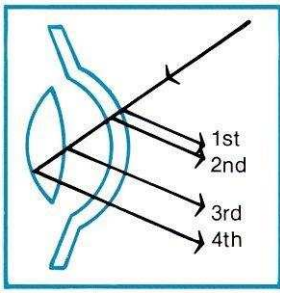


Figure 1.4: Purkinje images [24]. Four refracting surfaces of the eye cause reflections: front and backside of cornea and lens.

by the eye movements themselves. If the glint is not reflected from the cornea but the surrounding eye, the mathematical computations to infer the eye position are complicated.

Purkinje Image Tracking. The previously mentioned glint is only the first reflection that a ray of light causes on its way into the eye. Further reflections are cause by the internal structure of the eye. Figure 1.4 shows all four occuring reflections, the Purkinje images. This technique uses the relative positions of the glint and the fourth image, that is caused by the back of the lens. It is generally more accurate than other techniques and allows very high sampling rates up to 4000Hz. One problem is the rather dim fourth image, that is difficult to capture.

Infra Red Oculography (IROG). This technique is also based on the reflection of infrared light, but rather on the amount of light than video images. If the eye is illuminated with IR light, the amount of light that is registered by a sensor, varies with the orientation of the eye. This way, the eye position can be determined. One advantage of using IR light is the low sensitivity to changes in environmental lighting. As no semantic analysis is performed, eye blinks can be problematic with this technique.

Corneal Reflection And Eye Image Using An Artificial Neural Network (ANN). This technique uses an ANN to determine the eye position based on video images. The network requires a rather lengthy calibration procedures for training of more than 30 minutes. With 1.5-2^o, the accuracy is not very good, but this technique offers a high head mobility for the user [12].

1.3.3 Health And Safety Considerations

Eye tracking systems commonly use IRLED (Infrared Light Emitting Diodes) to illuminate the eye. Using a light source that is invisible to the user has the advantage that no disturbing light is in the field of view that might blind the user. Exactly this advantage is also one of the greatest risks when using invisible light sources: we do not have any natural aversion to over exposure [25]. For this reason, the user can not naturally protect himself or herself from possibly harmful effects, hence great

care has to be taken to assure that the use of the eye tracking system does not cause harm to the user.

The ICNIRP (International Commission on Non-Ionizing Radiation Protection) guidelines chosen for this thesis are described in [25]. These guidelines have been chosen as they provide a good safety measure while not making unreasonable worst-case assumptions that do not apply to the tracking system and the used near infrared illumination.

The main focus for the hazard assessment is the eye itself, the surrounding skin is not likely to be affected [25]. Although the mere energy incident to the eye is the basis for the assessment, the effect on the eye is highly dependant on biological effects. There are two main interaction mechanisms by which the eye can be affected, and that are taken into account by the guidelines. Photochemical interaction and thermal interaction. The guidelines provide a compact set of factors and limits that are relevant for the assessment, but that still reflect such biological characteristics that are relevant for safety.

When assessing the risk posed by IRLED illumination the following factors are relevant:

1. Distance of the light source to the eye
2. The size of the light source
3. The brightness of the light source
4. Exposure time (chronic or acute)

A factor that is also important but that is already determined by choosing an infrared light source is the wavelength of the source, as the effects on the eye are highly wavelength dependant [25].

An important aspect that can be derived from the listed factors is the *retinal image size* of the light source, i.e., the size of the source after it passed through the refracting surfaces of the eye. This is in particular relevant as the cooling efficiency of the retinal tissue depends on the size of the heated area. Small areas are more efficiently cooled than larger ones [25].

Another effect that influences the retinal image size is the blurring of the image at the near point of accommodation. If the light source is closer than this point, the eye cannot focus on it anymore and the blurring additionally enlarges the retinal image. Under normal conditions this point is about 10-20cm from the eye. As the eye tracking system places the light source at 5cm from the eye, the standards limit for the maximum source size is used to provide a worst case assessment for the retinal exposure limit.

Overall the standard provides two different limits. For *cornea and lens*, and for *retinal* exposure. Taking the wavelength and the maximum source size into account, the limits are as follows:

$$\begin{aligned} \textit{Cornea and lens}: \quad & E_{IR} \leq 10\,{}^{\mathrm{mW}}/_{\mathrm{cm}^2} \\ \textit{Retina}: \quad & E_{IR} \leq 6\,{}^{\mathrm{mW}}/_{\mathrm{cm}^2} \end{aligned}$$

with E_{IR} as the irradiance caused by the IR light source.

At the distance of the eye one of the used LEDs was measured[1] to provide with $1.5\,{}^{\mathrm{mW}}/_{\mathrm{cm}^2}$. This yields a sum of $3\,{}^{\mathrm{mW}}/_{\mathrm{cm}^2}$ for the irradiance of both LEDs. With the common use of the device at the stage of development, no negative side effects were experienced over the last 6 months.

1.4 Thesis Overview

The remainder of the thesis is structured as follows:

Chapter 2 describes the most important aspects of the implemented system, the implemented functionality and the relevant aspects of the hardware from a *bird's eye perspective*. Chapter 3 and Chapter 4 give a detailed description of all involved hardware and software components, respectively. Chapter 5 summarizes the achieved results, describes limiting factors and gives and outlook on future improvements.

[1] The Measurement was performed using a Hewlett Packard 8153A Lightwave Multimeter with a range from 800 to 900nm.

Chapter 2

The Eye Tracking System

This chapter gives an overview over the presented eye tracking system and highlights the most significant aspects that are taken up in more detail in the following chapters. Section 2.1 describes how the eye tracking is performed and which steps were taken to make the tracking possible. Section 2.2 describes the precomputations that provide the data that is necessary for the tracking at runtime. Section 2.3 describes how the SR3300 Vision Chip is used for tracking. The SR3300 is connected to a host workstation via USB. Section 2.4 describes a Windows application that runs on the workstation and demonstrates the use of the eye tracking system.

2.1 Tracking Eye Movements

The presented work implements a high speed eye tracking system capable of tracking eye movements with a frame rate of 100Hz. Such frame rates are required if the system is to be aware of saccadic eye movement, as described in Section 1.3.1.

The Used Eye Tracking Technique. The system tracks eye movements based on the position of the pupil, or more accurately, the position of the center of mass, the centroid. To be able to determine this point, the pixels that are covered by the pupil area have to be determined from a grayscale image of the eye. This is done by performing the segmentation of the image into background and into the foreground that contains the pupil area.

At runtime, i.e. during tracking, the segmentation is performed using *binarization* followed by *region growing* using the *Self Fill* algorithm described in Section 3.3. The centroid is then computed as the center of mass of the area of the pupil within the binary image. Using binarization for segmentation is based on the fact that the pupil is darker than the rest of the image, which is called a *dark pupil* approach in eye tracking terminology.

To perform region growing, a starting point has to be provided. The first point is provided by the host workstation and the pupil area is covered by growing from this point. The centroid determined for this area is then used as starting a point in the next frame. Ideally, the growing only covers the pupil and possible binarization artifacts are eliminated. The pupil centroid is continuously tracked for every frame.

Problems. At runtime, no sophisticated image analysis is readily available, and it is not possible to perform any semantic analysis to identify the area of the pupil. The tracking algorithm cannot distinguish between pupil area and undesired binarization artifacts, or *clutter*, around the pupil. Also, the Self Fill algorithm would fail in performing the segmentation if clutter is connected to the pupil. It would be desirable that binarization should yield binary images that only contain the area of the pupil, and no clutter.

Due to the general color distribution within the image of the eye, the possibly dark pigmentation of the iris, and the characteristics of the used camera system, this optimal result is not to achieve. Large amounts of clutter appear if the image of the eye is binarized with an inappropriate threshold. Together with the fact that the threshold varies for different eye positions, this comprises a serious obstacle to successfully tracking the eye position. Clutter frequently connects with the pupil area. Not being able to perform semantic image analysis to circumvent this obstacle, or to cope with natural problems such as eye blinks at runtime, contributes to the problem.

Dynamic Threshold Adjustment. The presented system does not completely solve the problem, but succeeds in keeping the amount of clutter at a low level. In particular, the clutter does not connect with the pupil area and Self Fill can successfully perform the second segmentation step to eliminate clutter.

To solve the problem, grayscale images were taken for different eye positions. Inspection of these images, showed rather large changes in the color values that were associated with the pupil area. Due to the continuous nature of the changes, they can most likely be attributed to the effects of the *lens distortion* described in Section 3.3 and to non-uniform illumination. Compensating these effects keeps the binary image of the pupil relatively stable and reduces clutter.

The microcontroller component described in Section 4.3.2 achieves the compensation by dynamically adjusting the binarization threshold as the eye moves and the pupil changes its position.

It does so by performing bilinear interpolation between threshold values that are determined before tracking is started. Each threshold is associated with a specific position within the image space, and the value is chosen to provide a good binarization result if the pupil center is located at that position. Using these *ideal threshold* values to interpolate threshold values for other positions, segmentation can be performed with good results for all pupil positions.

Result. Figure 5.6 compares a tracking session without adjusting the threshold to the presented system. It is visible that the tracking fails if the threshold is not adjusted.

Another consequence of the small amount of clutter in the binary image is of advantage when the pupil is lost during eye blinks. As the pupil makes up a large part of the overall number of set pixels, the center of mass of the whole binary image is very likely to be located within the area of the pupil. Due to the working of the *region growing* the pupil can be correctly recaptured for the next frame, by starting the growing from this center.

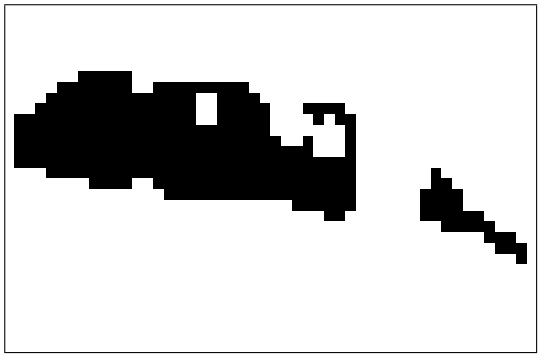


Figure 2.1: Binary image showing sudden increase in clutter during eye blink.

Naturally this is only the case if the pupil stays in the vicinity of the last position that was used to interpolate the binarization threshold. However, the short duration of eye blinks and the high frame rate make this a very likely case. Whether the pupil was actually recaptured can be verified by the general capability of the system to detect distortions.

Detecting Distortions. A simple way to detect distortions is again based on having a known amount of clutter in the binary image. This makes it possible to detect distortions, based on the amount of clutter present in the binary image. This is in particular true for the most common distortion, an *eye blink*. In this case, another kind of distortion besides the pupil loss, has been found. During the blink, the binary image shows a sudden increase of clutter as depicted in Figure 2.1. Comparing the actual amount of clutter to an expected value, the tracking component can detect the blink and report the distortion.

Another way to detect the distortions caused by an eye blink, is to monitor the area of the target, i.e. of the pupil. If clutter connects as it is shown in the figure, the pupil area is also drastically increased. However, if the clutter does not connect with the pupil, the tracking results remain valid even though the overall amount of clutter might be quite large. Therefore this approach is in general to favor, as it copes better with other kinds of distortions. In these cases that cannot be completely avoided, clutter can appear without distorting the pupil.

The loss of the pupil target, is reflected by the target area being zero, hence the distortion is easily detected. After the mentioned recapturing of the target, it is checked for distortions, and rejected if distortions are present. Otherwise the coordinates of the target are regarded as valid.

The implementation of a more sophisticated method to detect distortions is demonstrated in Section 4.3.3. In general, any feature based on the described moments can be computed for the target, and compared to a constraint vector computed before tracking. If appropriate, the precomputed values can be included in the bilinear interpolation. This way, a more robust detection of distortions can be implemented. Due to memory size limitations of the SR3300 microcontroller, these methods could not be included into the presented system.

2.2 Precomputations For The Tracking Phase

Besides the calibration steps that are commonly encountered in eye tracking systems, using the Vision Chip requires additional precomputations. These are performed not during tracking, but are performed by the host workstation before tracking is started, and are the basis for the overall solution presented in this work. Both the dynamic threshold adjustment and the detection of distortions performed at runtime rely on these precomputations.

A crucial point of the precomputations is the image analysis that identifies the pupil using a modified Hough Transform. All subsequent steps rely on this information to determine features associated with the pupil.

Image Features. One step to the solution was to identify such image features that are available at runtime, but can be associated with the pupil before tracking is started. The area of the pupil for example, can easily be computed by the workstation, as the location of the pupil is known. At runtime on the other hand, the location is not known, but the area can be computed from the binary image.

This way elaborate image analysis is not necessary at runtime, but the validity of the tracking results can be guaranteed by runtime comparison to the precomputed features. Distortions can be detected as previously described.

While this comparison is a means to detect the failure to perform successful segmentation, i.e., that clutter affects the tracking results, it is in turn made possible by adjusting the threshold, to keep clutter generally at a low level.

To make these runtime steps possible, the precomputation extracts at least the following features from grayscale images of the eye:

1. Pupil area
2. Ideal binarization threshold

Other moment based features, e.g. the mentioned *eccentricity*, can easily be computed using the information about the pupil location. Having these features available for different eye positions, the tracking system can successfully track the position at runtime. Section 4.2.3 describes the component that performs the precomputations to extract the features.

Point Of Regard Computation. Another step performed before tracking is started, is the calibration of the eye tracking system. It allows the mapping of eye positions into the screen space of the workstation to indicate the position the user is currently looking at. During this phase, different position on the screen are associated with eye positions. This information is used at runtime to approximate a Point of Regard by mapping the pupil position into screen space. The actual mapping from the eye tracking systems frame of reference, is performed by the host workstation after the pupil position has been received. The mapping is described in Section 4.2.4.

2.3 Hardware For Tracking

The core component of the system that makes the high frame rates possible is the Vision Chip (VC). It is mounted in front of the user's eye using a pair of safety glasses and fulfills the job of a camera and also performs the actual tracking and moment computations with the given frame rate.

Vision Chip Functions. The Vision Chip is controlled by a microcontroller (MC) that executes the previously described algorithms. The MC uses VC functions to perform the following tasks:

1. Tracking of a selected target
2. Centroid calculation
3. Grayscale and binary pixel acquisition

The first two items are related to binary images only, as tracking and calculation of the target position access only binary information to be able to achieve the required frame rates.

Using the VC functions, the MC determines the pupil position as the centroid of the tracked target and uses the USB connection to transfer it to the host workstation. If no distortions are present, the pupil position is continuously tracked.

VC functions are described in more detail in Section 3.3.

2.4 The Working System

The use of the eye tracking system is demonstrated by a Windows application that performs the calibration and offers different kinds of visual stimuli that can be traced by the user. The tracked eye position is indicated on the screen and written to a file.

Before the calibration begins, the program displays a video of the images currently captured by the eye tracking system. The corresponding dialog is depicted in Figure 4.1. The blue dot within the pupil indicates the center of the pupil as it is determined by the host image analysis. This step is necessary to adjust the safety glasses to position the pupil so that it can be located successfully.

During the next step, the eye tracking system is calibrated by displaying a set of visual stimuli on screen. As the stimuli appear on screen, the user is required to focus on each stimulus in turn and trigger the calibration for this eye position. Figure 4.3 depicts a calibration screen with five stimuli. The surrounding rectangle indicates the extent over which the stimuli are displayed and defines the area in which the eye movements are tracked. Figure 4.6 shows a set of grayscale images acquired during the calibration phase. These pictures are processed by the host image analysis. Figure 4.7 shows the corresponding binary images created using the computed threshold.

Once the calibration was performed, tracking is started. The program displays some visual stimuli to demonstrate the tracking. The current POR is indicated by a

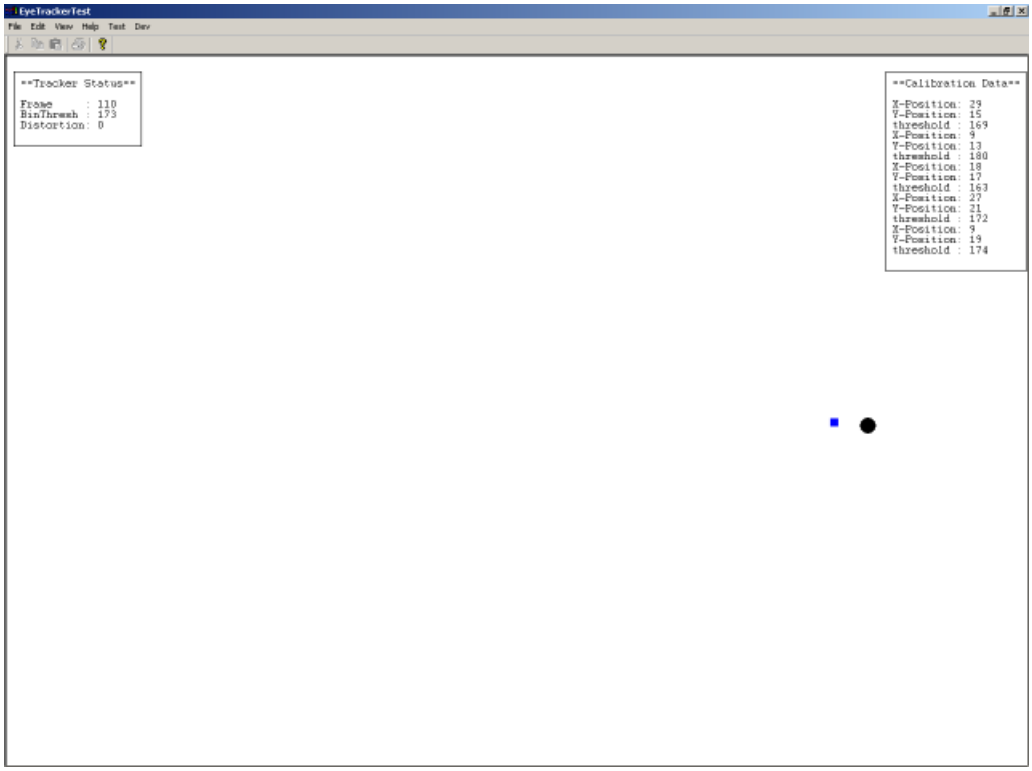


Figure 2.2: Vision Chip Eye Tracker demonstration application. The user was looking at the black circle, the blue square indicates the POR reported by the eye tracking system.

small blue marker as shown in Figure 2.2. The image was taken while the black circle was traced by the user. For the given stimulus the circle moved along a rectangle. Figure 5.5 shows the tracked eye position and the POR as it was mapped into screen space to display the marker. In both images the corner points of the traced rectangle have been indicated. While tracking is performed, the user can inject a marker by pressing the space bar. The marker is injected into the stream of tracking data and can be used to identify certain screen positions in the tracking data file.

Chapter 3

Hardware

This chapter describes the hardware that is used for the eye tracking system. First, the experimental setup involving all components is described. In subsequent sections each component is described in detail.

3.1 Hardware Setup

Figure 3.1 shows the experimental setup of the eye tracking system, and Figure 3.2 gives a schematic overview of the setup. The main component is the *SR3300 Vision Chip* that captures the image of the eye and performs the tracking. For simplicity, the SR3300 is directly attached to a pair of safety glasses, occluding large parts of the user's field of view. The safety glasses in turn, are mounted in front of a computer screen to fix the head position.

To illuminate the user's eye, two infrared LEDs are used. For the experimental setup the LEDs are powered by an external source, but the SR3300 provides a 5 Volt power supply.

A USB cable connects the SR3300 to the host workstation controlling the computer screen.

3.2 SR3300 Vision Chip

Figure 3.3 shows the main component of the setup, the *SR3300 Vision Chip*. The depicted circuit board integrates the Cypress AN2131SC chip and the actual Vision Chip (VC) component. The Cypress chip contains a 8051 compatible microcontroller and circuits necessary to connect to the chip via USB. The MC is connected to the VC component and controls its operation as well as the USB connection to the host workstation.

The connection to the tracking component is established using port pins. The MC program uses regular port commands to control the VC via these connections. The SR3300 also integrates an ADC (Analog Digital Converter) external to the Vision Chip component. This converter is used to acquire grayscale values from the analog values provided by the VC.

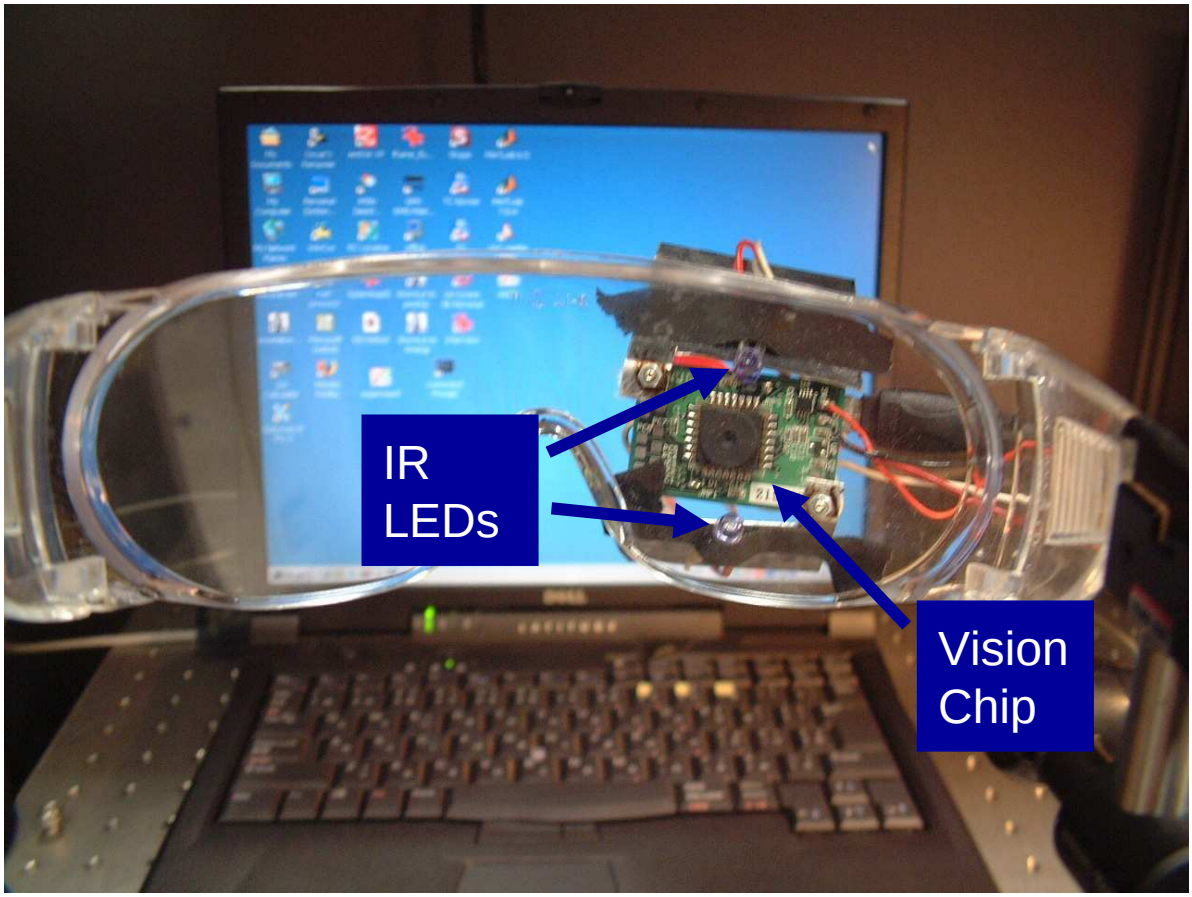


Figure 3.1: Hardware setup for the eye tracking device.

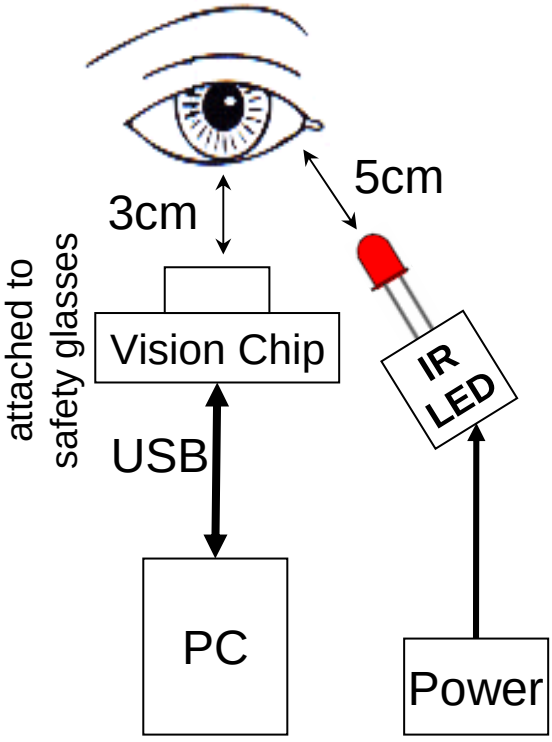


Figure 3.2: Schematic of the hardware setup for the eye tracking device.

Figure 3.3: SR3300 Vision Chip.

The lens visible in the figure has a focal length of 3.0mm. The image is projected onto a sensor array of 1.568mm by 2.352mm, with a pixel size of 49μm.

The facilitated VC component is described in Section 3.3. It also mentions some of the problems that revealed itself during development.

The SR3300 was chosen for the following reasons:

- The VC component makes frame rates of up to 1000Hz possible which is ideal for saccadic eye tracking.
- The offered functionality for high speed calculation of the centroid position seems ideal for pupil tracking.
- The sensitivity to infrared illumination is almost a requirement in respect to non-intrusive eye tracking.
- The low price is an important advantage in comparison to other high speed eye tracking systems.
- Due to its USB connection, the SR3300 is easily integrated into a computer system.
- The small size make it attractive for applications that require high mobility. The same is true for USB connection.
- The integrated MC and the available ports, suggest the use of the eye tracker as a stand alone system, without a direct connection to a PC.

3.3 Vision Chip Tracking Hardware

The Vision Chip [26] is special purpose tracking hardware that enables the eye tracking system to track the movements of the eye with the required frame rates. In the simplest sense the VC is a camera system with integrated image processing capabilities.

This section describes how the images of the eye are taken, how the tracking of a target is performed and how its position is calculated. In particular the functionality that is used to perform segmentation by binarization and region growing is addressed.

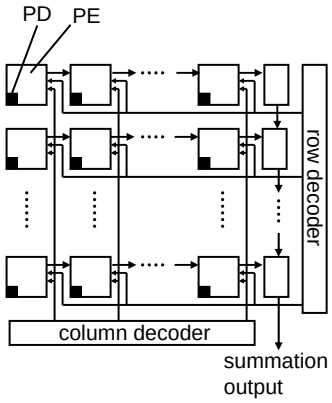


Figure 3.4: Array of Processing Elements.

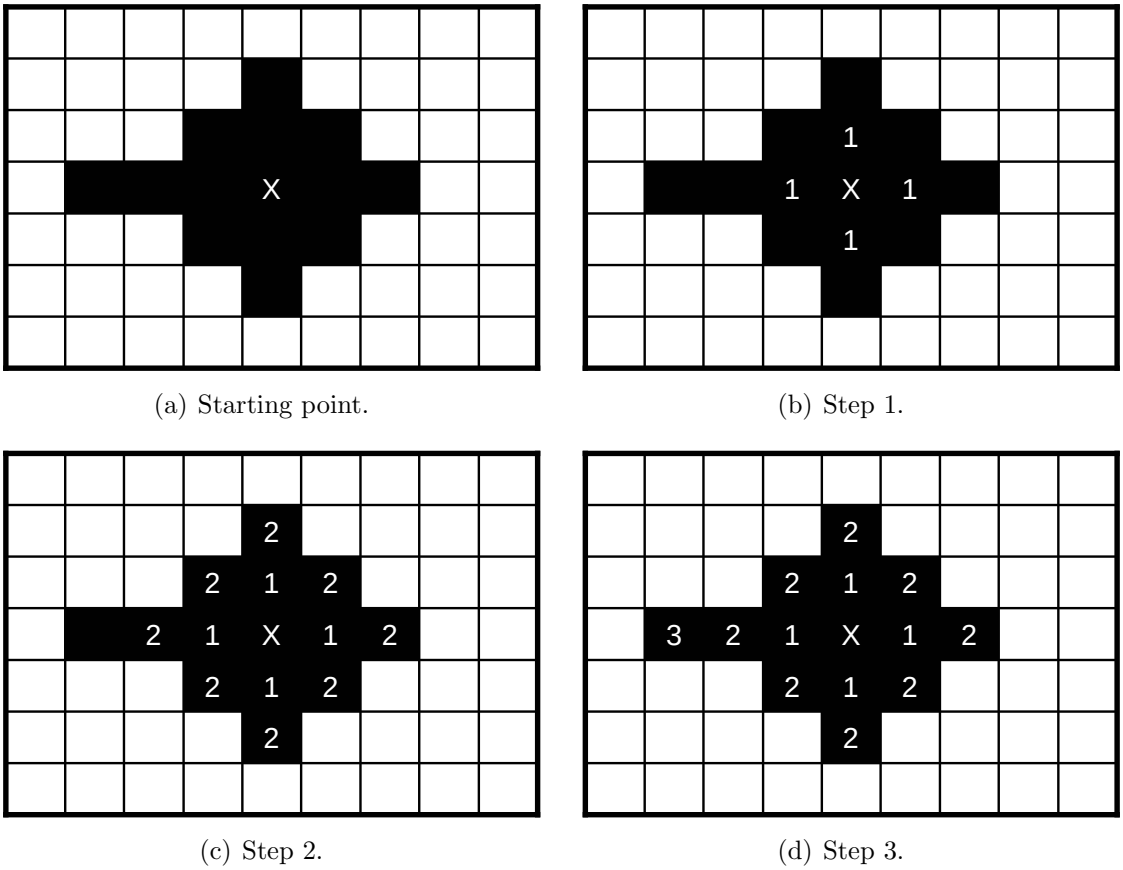
Relevant Vision Chip Functions. The relevant functions performed using the VC are the following:

1. Tracking of a selected target
2. Binarization of analog intensity levels
3. Area calculation
4. X Moment calculation
5. Y Moment calculation
6. Centroid calculation
7. Grayscale and binary pixel acquisition

The tracking and all calculations are based on the binary image that results from the binarization. The binarization is performed by the chip itself with sufficient speed to provide tracking frame rates of up to 1000Hz. The acquisition of binary image data, for example for external use by a MC program on the other hand, is by magnitudes slower. The analog to digital conversion necessary for grayscale image acquisition is performed by an ADC (Analog Digital Converter) external to the VC and consequently takes even longer time.

Internal Structure. To understand these functions it is necessary to take a closer look at the internal structure of the VC. Figure 3.4 depicts the array of *processing elements* (PE) that is responsible for acquiring and processing the image. Each PE corresponds to a pixel of the image. The VC has a pixel resolution of 48 by 32. The *photo detector* (PD) that actually measures light intensity is also depicted as a small black square. The analog value of a PD can be binarized and is available for processing within the PE. The entirety of this array of binary values from the PD, is referred to as the image *f*. Seen from a higher level of image analysis, f represents the result of the first segmentation step, the foreground of the image, i.e., the target.

Each PE also contains memory to store 1 bit. The entirety of the array of bit values is called *target window* as this memory represents the target that is used for



(a) Starting point. (b) Step 1.

(c) Step 2. (d) Step 3.

Figure 3.5: Self Fill algorithm. The grid of the *target window* has been overlayed over binary image f that contains the pupil area as black pixels. The pixel initially set in the target window is marked with "x". The pixels set in subsequent iterations are marked with the number of the iteration. The pixel set in the last iteration is an indication for the presence of clutter.

all subsequent calculations during tracking. A pixel of the target window is set if it is part of the target, or not set otherwise. It is to be noted that, although related, f and the target window will generally be different. This will become clear when looking at how the second segmentation step, the region growing, can be applied to f, or more specifically, how the target is created in the *target window*.

Target Creation using Self Fill. The approach used by the eye tracking system is the *Self Fill* region growing algorithm [26]. The algorithm is implemented in hardware and partially in MC code. Figure 3.5 illustrates the steps of the algorithm for the area of the pupil. For every frame it starts out with an initially empty target window, and assumes that the centroid coordinates of the target from the previous frame are known. The target window at this location is then set as the starting point for the filling. In the next step, all 4-neighbors of this pixel are set, but only if the image f is also set at that location. These steps are implemented in hardware and are iteratively invoked by the MC.

In subsequent iterations the 4-neighbors of already set pixels are set in the same manner and the region of set pixels keeps growing. As the setting of pixels is

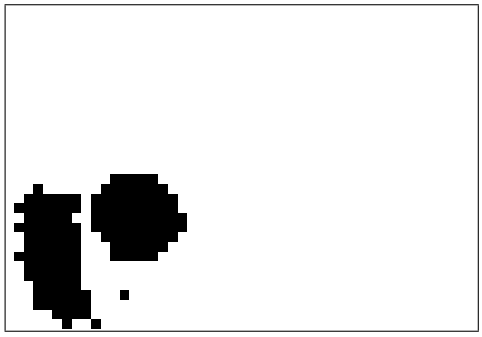


Figure 3.6: Clutter in binary image. To the lower left of the pupil area, appear binarization artifacts in the binary image of the eye.

bounded by the image f that represents the current location of the target, this can be seen as the filling of the target. The meaning of the last iteration depicted in the figure will be explained shortly.

The working of this algorithm is based on the high frame rates of the VC. Because the target cannot move more than one pixel between consecutive frames, it is guaranteed that the centroid from the previous frame is still enclosed by the target area at the new position. Therefore, the area filled using Self Fill will normally represent the new location of the target. It is also clear that the initial centroid coordinate has to be provided to the VC and, in general, cannot be computed by the chip itself.

As previously noted, the resulting target window is not necessarily equal to f. In fact, for the eye tracking system, it is an important requirement that they differ. This becomes clear when looking at Figure 3.6 that represents an image f during tracking. Although it is desirable that the result of the segmentation performed by binarization contains only the target, this cannot always be guaranteed. The specks to the lower left of the large circular area are undesired clutter that is not to be visible in the target window. Hence, applying region growing by means of the Self Fill algorithm is another image processing step required for segmentation.

A crucial assumption of this algorithm is that the target does not connect with any of the clutter. If this is still the case, the tracking proceeds just the same, but yields wrong results as the filling will proceed beyond the boundary of the actual target. Whether the assumption is true, largely depends on the tracking task and the color distribution of the involved images. The case of connected clutter is indicated in Figure 3.5(d) by a single clutter pixel marked with 3.

Having created the target within the target window, the binary information is used to compute the required tracking data.

Calculating The Target Position. The target position is calculated as the center point of the target area, its center of mass, or centroid according to the

following formula:

$$\overline{x} = m_{10}/m_{00}$$
$$\overline{y} = m_{01}/m_{00}$$

where $\overline{x}$ and $\overline{y}$ are the centroid coordinates within the target window that are computed from different order moments. The most simple moment, m_{00} or the area of the target, is calculated using the internal circuits of each PE and their interconnections along a row.

Conceptually, the sum is first calculated over each row, and another summation unit provides the sum of all rows as a serial bit string that can be retrieved from the *summation output* marked in Figure 3.4. The actual circuits perform the summations in parallel to gain speed and to reduce the need for memory to store intermediate results.

As calculating the area is actually only a special case of calculating a moment, these steps have to be preceded by a step required by the general function to calculate moments. For the area, the row and column decoders indicated in Figure 3.4 simply have to be programmed to include the whole target window into the summation.

This is necessary as the higher order moments required to perform summation over only certain columns or rows of the window. The exact algorithm for the computation of higher order moments is described in Section 4.3.3.

Table 3.1 gives the performances for the basic VC functions relevant for tracking.

Function	Computation time
0th moment	$2.75\mu s$
1st moment	$12.5\mu s$
centroid	$27.75\mu s$
Self Fill	Area dependant. Dominated by the MC iterations (4Mhz clock).

Table 3.1: Performance of Vision Chip functions

As previously mentioned, f and the *target window* usually differ. If it is necessary to calculate moments for the whole image, the VC can be programmed to take all values of f into the target window and the necessary calculations can be performed as before.

Grayscale And Binary Image Acquisition. Grayscale and binary image acquisition is performed by the MC by iteratively querying the VC for the pixel values. To provide grayscale information, the analog value provided by a PE is converted to a byte value by an external ADC. For binary images the MC uses a VC function to check if a pixel is set and packs 8 bits into one byte. The grayscale values are simply stored as one byte, the way they are provided by the ADC.

Both types of acquisition are affected by the underlying hardware implementation of the photo detectors. The PDs are implemented as CMOS Photo Diode Active Pixel Sensors (APS). The analog voltage level that represents the light intensity, as

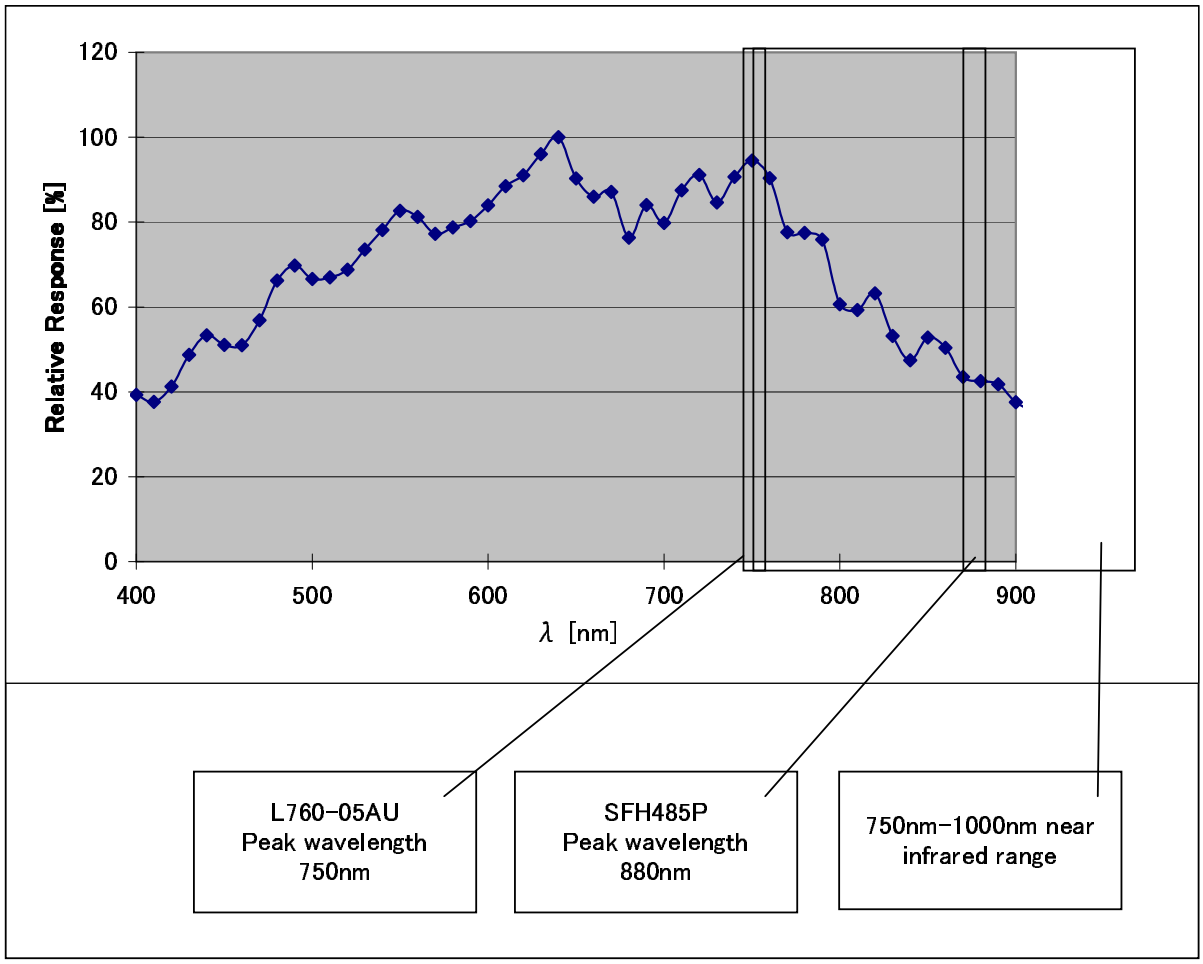


Figure 3.7: Spectral sensitivity graph for Vision Chip photo detectors.

determined by the APS, is stored in a capacitor. Both grayscale and binary image acquisition are based on this voltage level. Because of a leak current of the capacitor, the stored voltage degrades over time and it is not possible to capture the whole image at once. To provide a stable image, it is necessary to refresh the voltage after one line of the image, and consequently it takes several frames to capture the images.

It is to be emphasized that in the two cases, the quantization of the voltage level is performed in different places. In the case of binary image, it is performed internally by the VC, and in the case of grayscale images, by ADC. Due to variations between the reference voltages for both operations, problems can occur. This is described in more detail in Section 4.2.3 as part of the threshold color calibration.

Spectral Sensitivity. The APS also determines the general brightness of the captured image by its *spectral sensitivity*. Figure 3.7 shows the sensitivity for the SR3300 Vision Chip as well as the range of infrared light and the peak wavelength for some near infrared LEDs. It is visible that the sensitivity is best for wavelength close to the visible spectrum of light and degrading for wavelength closer to the infrared spectrum. The implications of this fact and the mentioned LEDs are described together with the facilitated light sources in Section 3.4.

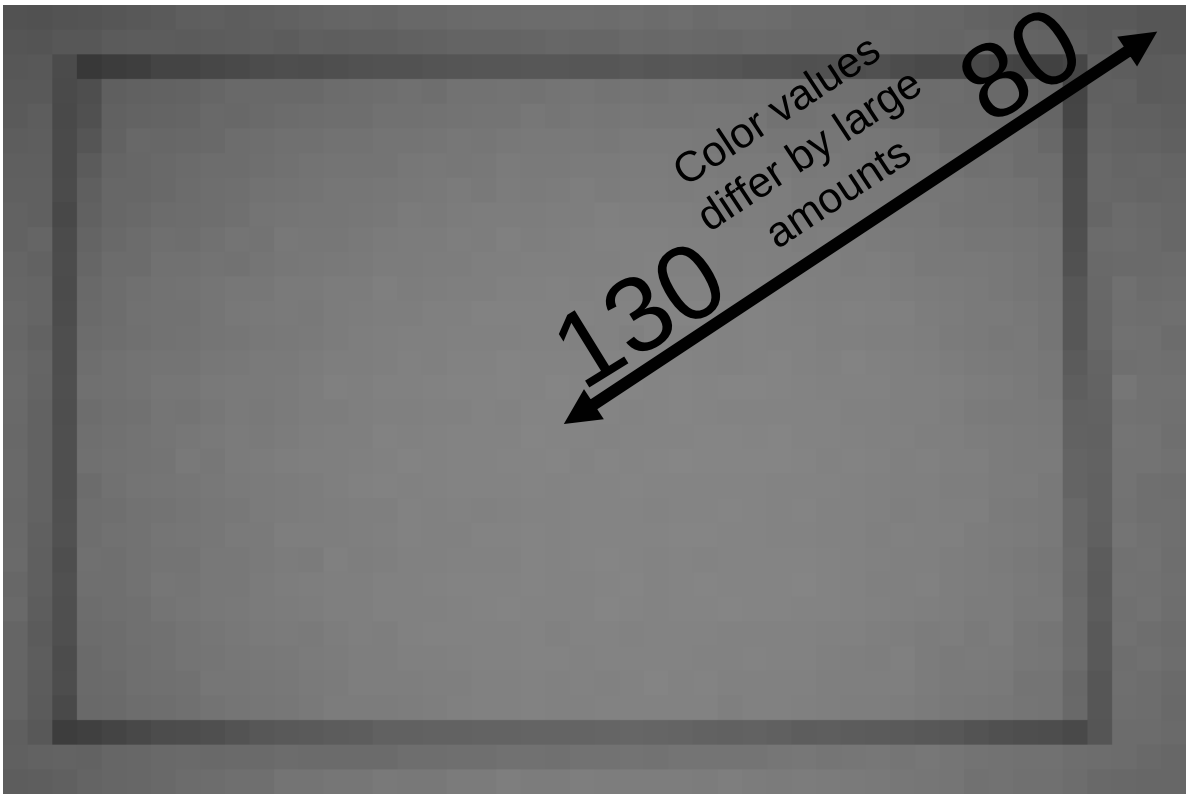


Figure 3.8: Lens distortion: A black rectangle on white computer screen. It is visible that corner areas are darkened. This appears due to the lens used by the SR3300. The grayscale image was captured at 100 frames per second.

Lens Distortion. Another factor that affects image quality is the distortion introduced by the lens. Figure 3.8 shows a grayscale image captured using the SR3300. Although the captured screen was completely white, the image shows a radial color gradient with brighter pixels at the center and darker pixels in the corners. As indicated, the color values can differ by 50 between the center and the corner.

If target and background are far apart on the color spectrum, such as it is the case for a black target in front of a white background, this is not a serious problem. For the presented system however, pupil and the surrounding background are not as clearly separated.

As part of Figure 4.4, the histogram of an image of the eye is depicted. Although the histogram is bimodal and a binarization threshold can be found for segmentation, problems occur during tracking. If one of the darker areas, such as the iris, would move into an area of the image that is affected by lens distortion, binarization can not be performed successfully. The color values of the iris would be shifted across the binarization threshold and would be part of the *target window*.

3.4 Illumination

The choice for the illumination was influenced by the following points:

1. Obtrusiveness
2. Safety
3. Image quality
    - Contrast
    - Homogeneity of Illumination
    - Disturbance caused by light source
    - Disturbance caused by other light sources

While the first two items are related to the usability of the system and the impact on the user, the remaining items are related to the impact on the image analysis that has to be performed for the tracking.

Obtrusiveness and Safety. As the overall goal is to design a system that is least obtrusive, irritating effects of light sources that are constantly in the field of view of the user have to be avoided. Therefore, light sources with wavelength invisible to the user are a good choice. As the Vision Chip is sensitive in the near infrared range and safety is more easily achieved with longer wavelengths, near infrared LEDs are recommended and frequently used by eye tracking systems in general. For the presented system in particular, the proximity of the light sources to the eye would make visible light an inconvenient choice for the user.

Contrast. To be able to perform the binarization to separate the pupil from the rest of the image, a threshold has to be found. The higher the contrast of the images is in general, the more reliable the threshold can be found, and the less sensitive the binarization is to other effects such as the distortion caused by the lens, that is described in Section 3.3.

Two aspects of the system can affect the contrast in a negative way. The first is inherent to the task of high speed tracking that requires very short exposure times to achieve the required frame rates. During the short exposure time only little light can be integrated by the photo diodes. As a consequence, the color spectrum is compressed as the frame rate increases. The second aspect is the degrading spectral sensitivity of the VC for wavelengths close to the infrared part of the spectrum. Choosing these wavelengths for illumination also impairs the contrast.

As the latter aspect is a matter of choice, it would be natural to choose the wavelength as low as possible to be able to take advantage of the higher sensitivity in this range. The spectral sensitivity is depicted in Figure 3.7 together with the peak wavelengths of two tested LEDs. Unfortunately, the LED in the range with the highest sensitivity is clearly visible.

Homogeneity of Illumination. In the presented system the light sources have to be placed at close proximity to the eye. Under these conditions first attempts with narrow beam LEDs showed negative effects due to inhomogeneous illumination. For this reason LEDs with a wide beam angle are more suitable and also avoid the need to realign the LED fixture for different user's.

Disturbance caused by light source. Another effect of narrow beam LEDs was the occurence of over exposure in the center of the image, while areas in corners where hardly illuminated. Moving the source further away did not solve the problem. Even with increased intensity the necessary brightness of the image could not be achieved. This is naturally affected by the close attachment of the SR3300 that shields the eye from illumination, if it is placed at a distance from the eye, behind the circuit board.

Experiments with narrow beam LEDs and infrared diffuser did not yield the required results either. Wide beam LEDs gave the best results.

Disturbance caused by other light sources. Environment light can be a major obstacle to successful image processing [27]. Changing light conditions can for example affect the brightness of the processed images or cast shadows. In most situations it is not possible to control environment light such as overhead illumination in an office surrounding. An approach that copes well with many situations is to use infrared illumination of a distinct peak wavelength and employ filters to filter out all but the used wavelength range.

The Selected Light Source. For the presented system the Osram infrared LED SFH485P (Table 3.2) is used. Two of these LEDs are connected as it is depicted in Figure 3.9. The corresponding circuit board is attached to the SR3300 circuit

SFH485P Infrared LED	
Package	5mm
Peak Wavelength	880nm
Beam Angle	80^o
IF(max)	100mA
Radiant Intensity at IF(A)	3.15 - 6.3mW/Sr

Table 3.2: SFH485P technical specification.

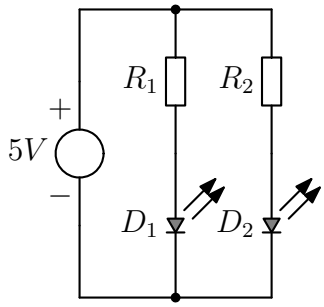


Figure 3.9: Extension board circuit for attaching infrared LEDs.

board. For the experimental setup an external power source is used, but the SR3300 provides a 5V output. The external power source is set to 3.1 Volt and the resistors have 36Ω each.

Chapter 4

Software

Conceptually, the system consists of two software components, running on a Windows 2000 workstation and a microcontroller (MC), respectively. The workstation component downloads the MC component to the SR3300 microcontroller and starts execution. Because of memory size limitations of the SR3300, the MC part is implemented in several distinct programs that are downloaded and executed separately. This is transparently handled by the Windows library to access the eye tracking system.

The eye tracker is controlled and accessed using the interface provided by this library. The library implements all necessary functions to make the tracking hardware go through the necessary phases of calibration and initialization until tracking. Once tracking is started, tracking data is stored on harddisk and is available at runtime.

The most relevant tracking data provided by the library is the pupil centroid. The x and y position of the centroid are available in tracker image coordinates or can optionally be mapped into screen space to provide POR information to the using application.

The following sections describe the communication between microcontroller and host, the host library and the microcontroller programs. The latter are implemented in C, while the host library uses C++ and compiled Matlab code as well as few portions in Assembler. Matlab is used for image analysis to simplify the task.

4.1 Microcontroller-Host Communication

For the host to be able to exchange data with the eye tracking hardware, a common communication protocol was implemented on top of the available USB protocol. The host component uses the available operating system calls while the microcontroller relies on the framework provided by the EZUSB library that is used with the AN2131SC.

The design of the protocol was largely determined by the limitations of the microcontroller, i.e., simplicity for the implementation of the MC program has been favored where possible. The aim was not only to reduce code size, but also to keep the portion of communication code as low as possible to avoid any impact on the

tracking code, whose execution is suspended during USB-transfers.

One restriction that is opposed by the underlying USB protocol is the necessity for periodically polling the SR3300 to receive its data. As a consequence, the frequency with which the host can poll for data is a limiting factor for the real time availability of tracking data. To some extent arising problems can be solved by customizing the thread priority for the polling thread, but the final limit is the operating system scheduler and the used time slices. In the case of the used Windows 2000 systems the shortest time slice possible is 20ms [28].

A faster system cannot solve this problem, only changing to a real time, or single task operating system allows to take advantage of the full real time performance of the eye tracker.

Nevertheless, the tracking data is continuously available, due to buffering performed by the MC program. This is one reason that makes more complicated variable length messages necessary.

4.2 Eye Tracker Host Library

The host library consists of several components whose functionality is available via a single interface provided by shared libraries. These components provide the following functionality:

1. Communication
2. User Interface
3. Image Analysis
4. Tracking Interface

4.2.1 Communication

The first component is described as a part of Section 4.1, it is responsible for communication with the SR3300 USB device.

4.2.2 User Interface

The second component provides user interface components that are useful or necessary to integrate the eye tracker into an application. It provides dialogs for showing a video stream of grayscale images of the eye, streams of the binary image that the tracker uses for tracking, and components to perform the calibration of the eye tracker.

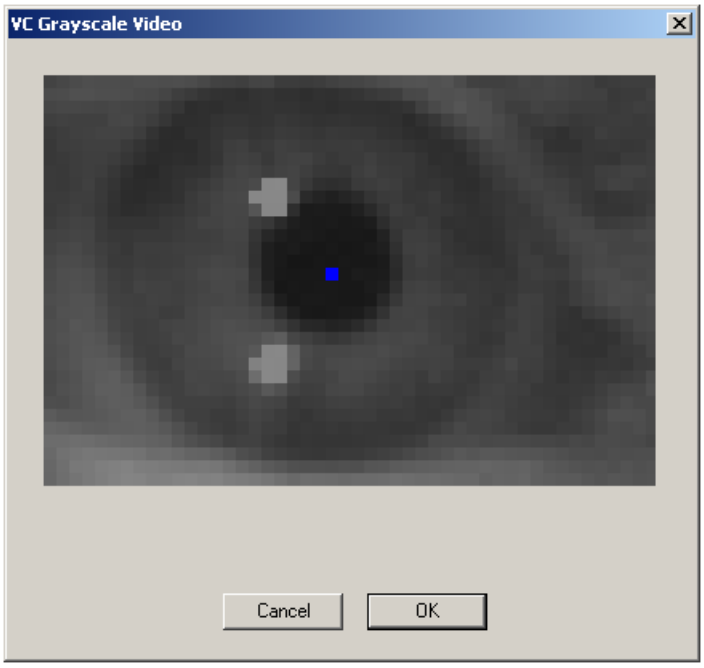


Figure 4.1: Video stream of grayscale images captured by the SR3000 Vision Chip. The pupil center is is indicated by by blue pixel.

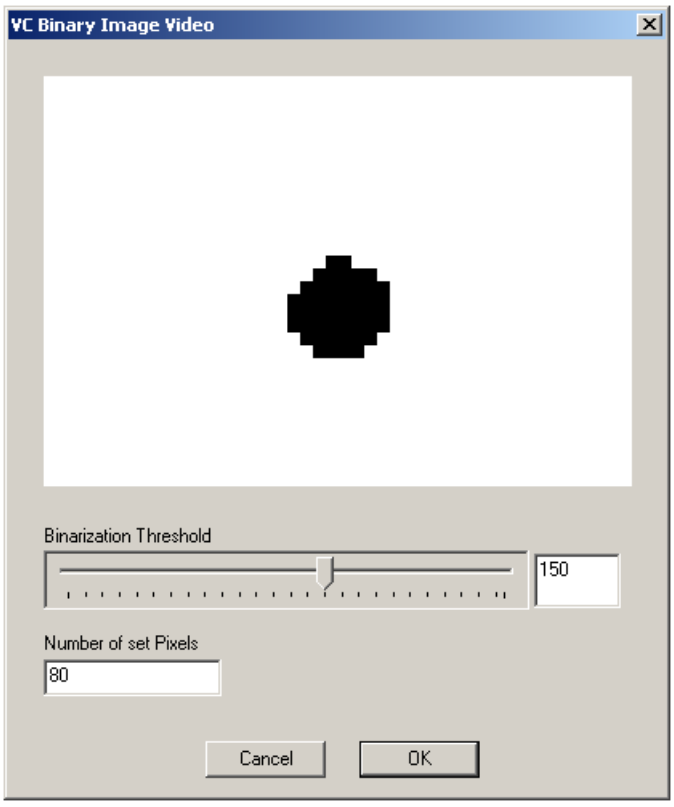


Figure 4.2: Binary image stream captured by the SR300 Vision Chip.

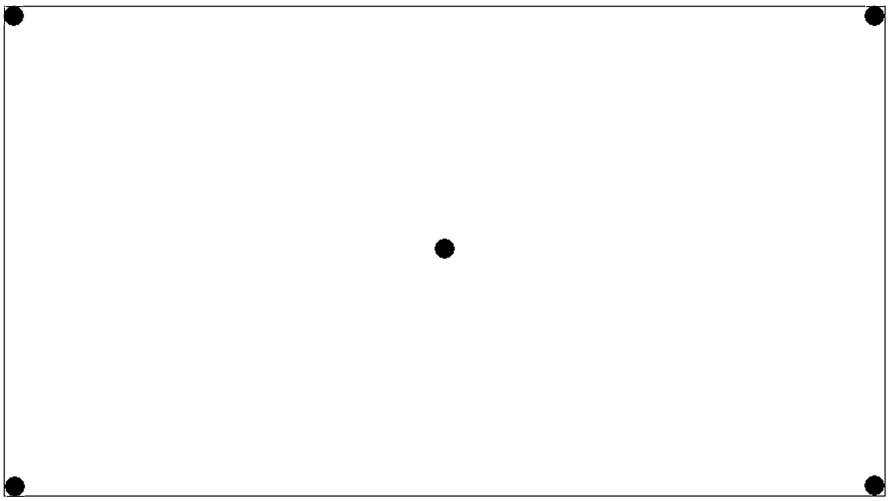


Figure 4.3: Visual stimuli presented during calibration.

Video Streams. Figure 4.1 shows the dialog that displays a video stream of the image of the eye. The blue dot marks the center of the pupil as it is determined by the host image analysis. The analysis is continuously performed to aid the adjustment of the glasses. Using the video stream the user has to adjust the glasses until the pupil is roughly centered and the pupil can be successfully located as indicated by the dot. The binary image stream shown in Figure 4.2 can be used to verify that binarization is performed as required.

Calibration. The calibration component displays a grid of visual stimuli on the screen as it is depicted in Figure 4.3. As each stimulus is displayed and the user focuses on the stimulus, the eye tracker interface is used to invoke the necessary calibration procedures as described in Section 4.2.4.

4.2.3 Image Analysis

The third component provides image analysis functionality that is needed to determine a set of features from the grayscale images during calibration. The functions are performed for different images of the eye as the user focuses on the stimuli on the calibration screen. The performed functions are:

1. Locating the pupil
2. Determine features associated with pupil
3. Determining an ideal threshold

Locating The Pupil. To locate the pupil, a modified Hough Transform is implemented that takes advantage of known properties of the analyzed images. The original algorithm to detect circles is described, e.g., in [29]. The implemented image analysis determines the same features, the pupil center and the pupil radius, but also uses these results to determine other properties of the pupil.

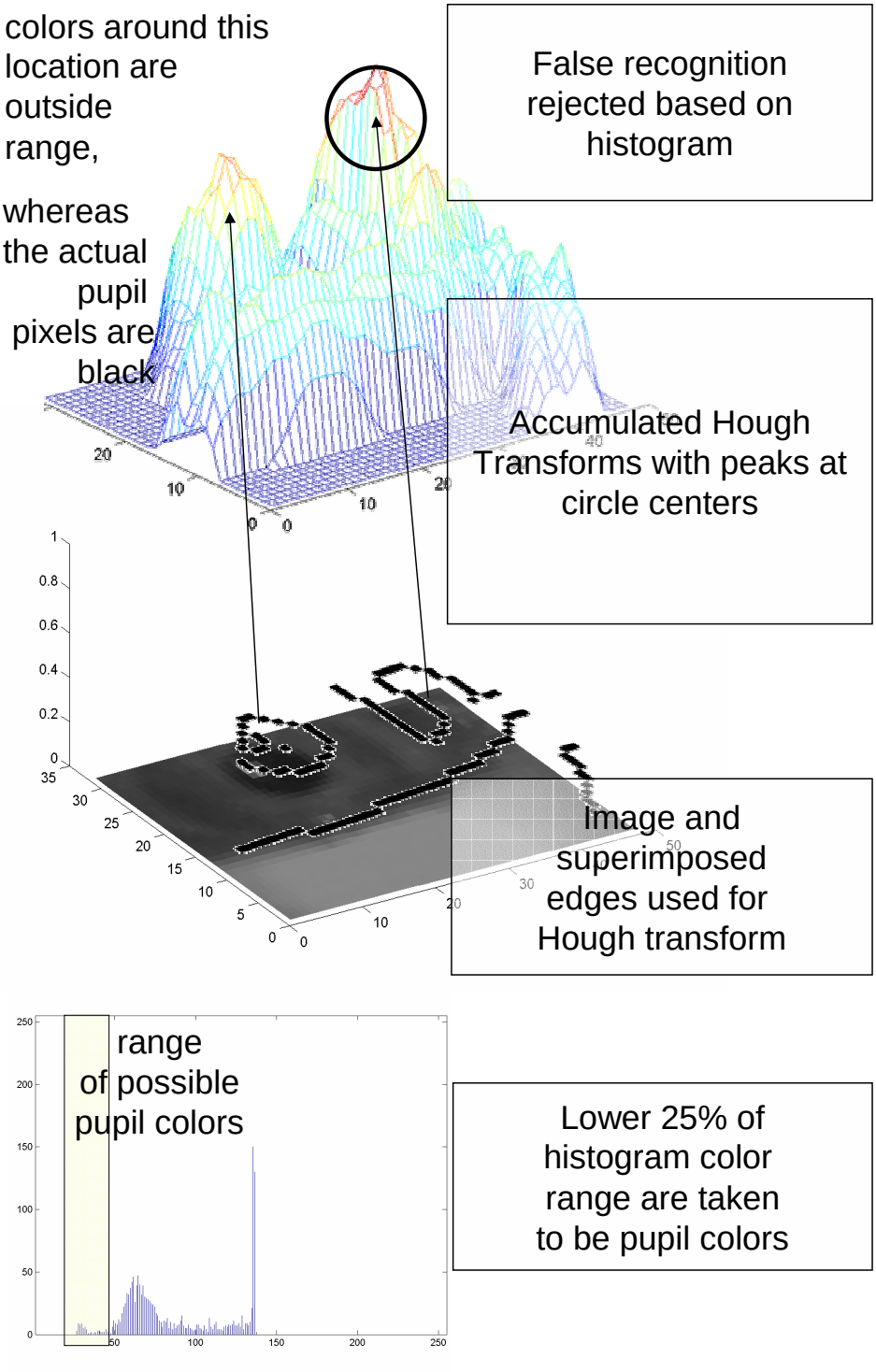


Figure 4.4: Rejecting false recognitions.

The Hough Transform is applied to a binary image containing the edges found in the grayscale image. To stay more flexible in respect to pupil diameters, the algorithm is relatively tolerant and accepts a wider range of radii for the circle detection. For this reason it is possible that a circle different from the pupil is identified as the pupil. Figure 4.4 illustrates the problem and the taken solution.

Generally the pupil is identified as the highest peak within the accumulator map created by the Hough Transform. In the depicted case however the pupil does not create the highest peak. For this reason the vicinity of the peak, as defined by the radius of the circle candidate, is inspected. If the average color in the vicinity does not fall into the range of possible pupil colors, the candidate is rejected. The range is simply defined as the darkest 25 percent of the histogram of the grayscale image. This way many false identifications can be rejected while staying more flexible.

To be able to take advantage of the sub-pixel resolution of the VC and to base the position on the binary image that is actually used by the VC, the determined pupil position is further optimized as described in Section 4.3.1.

paragraphPupil Features Having pupil center and radius available, many different pupil features can be computed. The most simple feature being the pupil area, as it is acquired by this library. Currently this is the only feature that can be used by the microcontroller components. As this is only due to a memory size limitation of the SR3300, this is not a principle limitation. If the problem is solved, other features, such as the eccentricity of the pupil can be included into the set of calculated features, to make them available to the microcontroller.

Ideal Threshold. To be able to perform the segmentation at runtime, a threshold for binarization has to be determined. This is done by first locating the pupil and then finding the ideal threshold for binarization. In this context, ideal means a threshold that successfully performs the segmentation of the image of the eye to show as much of the pupil as possible, while reducing the amount of clutter from the background of the image.

Using the information which part of the image is background and which part belongs to the pupil area, different binarization thresholds are tested. The threshold is varied so as to include as many pixels from within the pupil area in the binary image. Maximizing this criteria, the threshold is chosen that still keeps the amount of clutter in the background below a limit.

Threshold Color Calibration. For technical reasons this threshold that is determined by the host, based on grayscale images from the SR3300, cannot directly be used for the binarization that is performed at runtime.

To illustrate why this is the case, Figure 4.5 compares the thresholds found by the host, to the actual threshold of the VC, or more precisely, to a range of thresholds that reflect the quantization noise of the VC. The line fitted to the median of each threshold range shows the variation of the offset between the two measured values. As mentioned in Section 3.3, the offset occurs due to the fact that grayscale images are created using an ADC external to the VC, while the binarization performed during tracking is performed internally. For this reason the binarization performed by the host workstation yields different results.

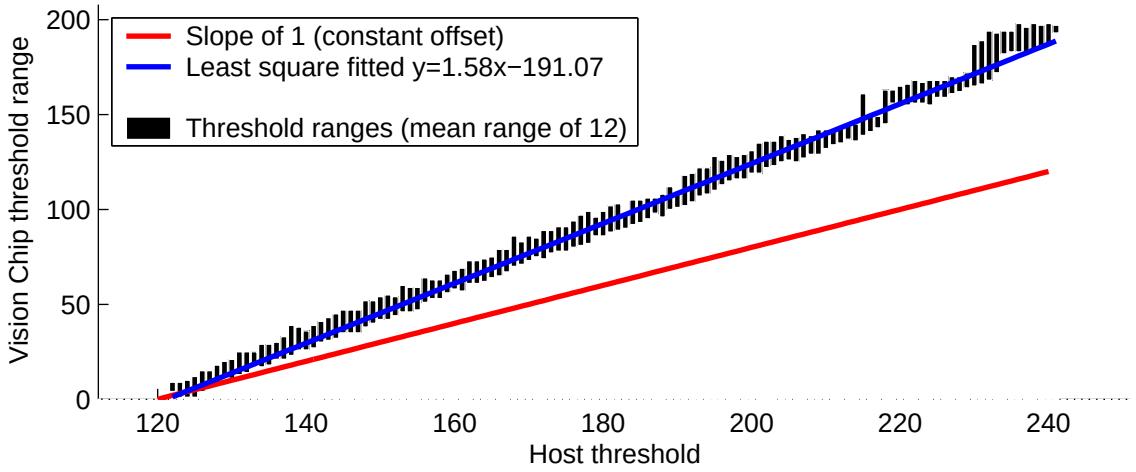


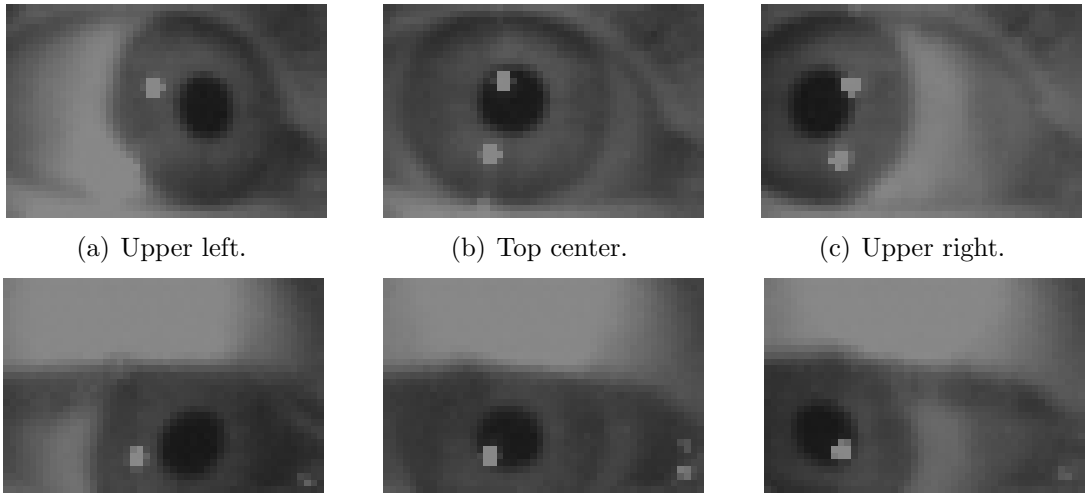
Figure 4.5: Comparison of binarization thresholds of host and Vision Chip. Due to noise a range of thresholds is measured.

Data Acquisition For Threshold Calibration. The data depicted in the figure has been acquired by capturing several grayscale images with values spanning the relevant range used during tracking. For each image the SR3300 performed binarization with all 256 possible binarization thresholds. The depicted threshold ranges were then determined by checking the binary images for each pixel in the grayscale images. For a given pixel and its grayscale value, a threshold was included in the range, if the pixel was set in the binary image for this threshold, in other words, if the grayscale value would be visible with that threshold. The check was performed beginning with the threshold value that yields no set pixel at all, and only the first threshold that yielded a set pixel at the checked pixel position was included in the range. By having several pixels with the same grayscale value, noise was still taken into account.

Threshold Optimization. Due to the noise, compensating the offset by applying the function defined by the fitted line does not yield the required results. To much clutter appears in the resulting binary images used for tracking. As the amount of clutter, and therefore the threshold, is crucial for the tracking result, further steps are taken to optimize the threshold, that should now more appropriately be called estimated threshold.

To bypass negative effects of noise, the optimization is performed at the SR3300 by the calibration component. For this purpose the Image Analysis component performs the binarization based on the grayscale image using the estimate, and determines the corresponding number of pixels set in the binary image. This value, together with the estimated threshold is transferred to the tracking hardware for optimization.

The microcontroller code for optimization starts with the estimate, and simply changes the binarization threshold until the binary image has the required number of pixels. This way, the optimization is based on a global image property, that is less sensitive to noise and that is similar for the grayscale images, and the analog image values that are used for binarization at runtime. Taking advantage of this fact, the



(a) Upper left. (b) Top center. (c) Upper right.

(d) Lower left. (e) Bottom center. (f) lower right.

Figure 4.6: Grayscale images acquired during calibration. The bright spots are caused by reflection of the infrared light sources. The Resolution is 48 x 32 pixels.

microcontroller program is kept as simple as possible while the more complicated image analysis can be performed by the host, despite the noise.

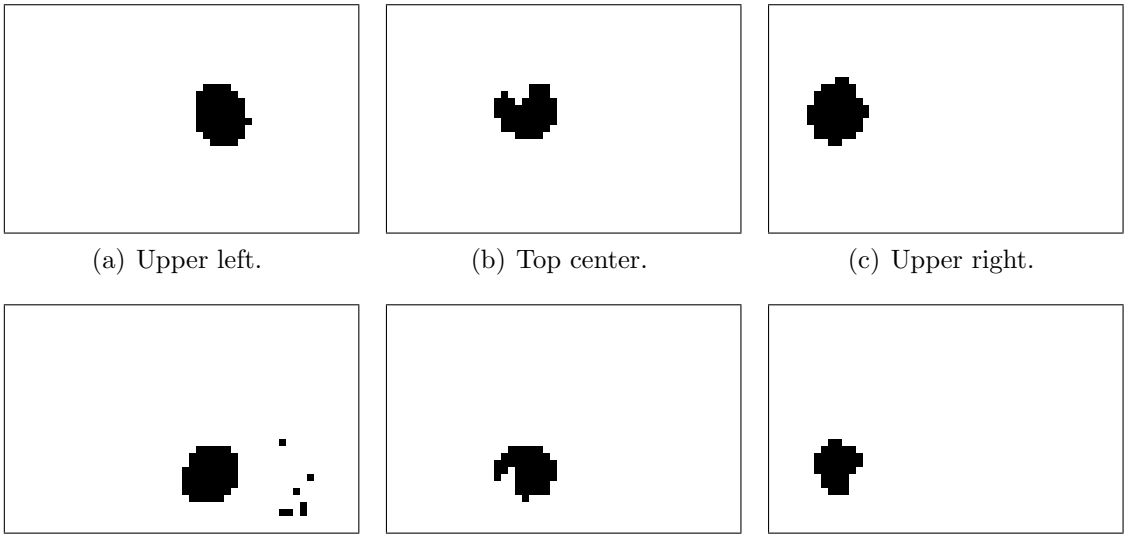
4.2.4 Tracking Interface

The fourth component is the actual interface that an application uses to access the eye tracking hardware. It relies on all other components. The following functionality and data is provided by the tracking interface:

1. Calibration of the eye tracker
2. Pupil position
3. Point of Regard information
4. Storage of realtime data

Before the tracking can begin, the necessary precomputation and calibration steps have to be performed, that are required by the MC tracking component described in Section 4.3.2.

Calibration. During the calibration phase the user interface component displays a set of visual stimuli on screen. The user is asked to focus on each one in turn. Each eye position is captured and image analysis is performed as described in Section 4.2.3. Figure 4.6 depicts examples for the captured images. Figure 4.7 shows the binary image for each position, that was created using the threshold determined by the image analysis. For each position the image analysis yields a *calibration point* that is later transferred to the MC.



(a) Upper left. (b) Top center. (c) Upper right.

(d) Lower left. (e) Bottom center. (f) lower right.

Figure 4.7: Binary images acquired during calibration. Resolution is 48 x 32 pixels.

One information contained in a calibration point is the pupil center location in VC image space. For each position this information is associated with the x and y position of the stimulus on screen. During tracking this association is used for mapping eye tracker image space coordinates into the screen space of the application.

Screen Mapping. The mapping is performed to provide Point of Regard (POR) information to the application. Mapping VC image space coordinates into screen space is performed as follows:

On the screen, the calibration stimuli form a rectangular grid as depicted in Figure 4.3. Assuming that the images of the stimuli form a similarly rectangular grid in VC image space, a coordinate t of a tracked position can be mapped into screen space using the following formula (see, e.g., [23]):

$$screen = c + \frac{(t-a)(d-c)}{(b-a)}$$

If the mapped coordinate t is the x coordinate, $b-a$, and $d-c$, are the width of the grid in VC image space and screen space, respectively. The values of a and c give the left bound of the grid in the respective space. Coordinates along the y axis are mapped accordingly.

This way the mapping can be performed with the assumed rectangular grid in VC image space. In reality however the rectangular grid on the screen is distorted in VC image space. This is partially due to the fact that the tracker image plane is not necessarily absolutely in parallel to the screen nor to the face plane. Figure 4.8 depicts the nature of this distortion. For visual simplicity, the depicted tracker image plane is only tilted along one of its axes, while in reality both axes might be affected.

To compensate the distortion, the VC mapping range $b-a$ is computed for every mapped point. In the figure this is illustrated for the y range, which is interpolated based on the x position.

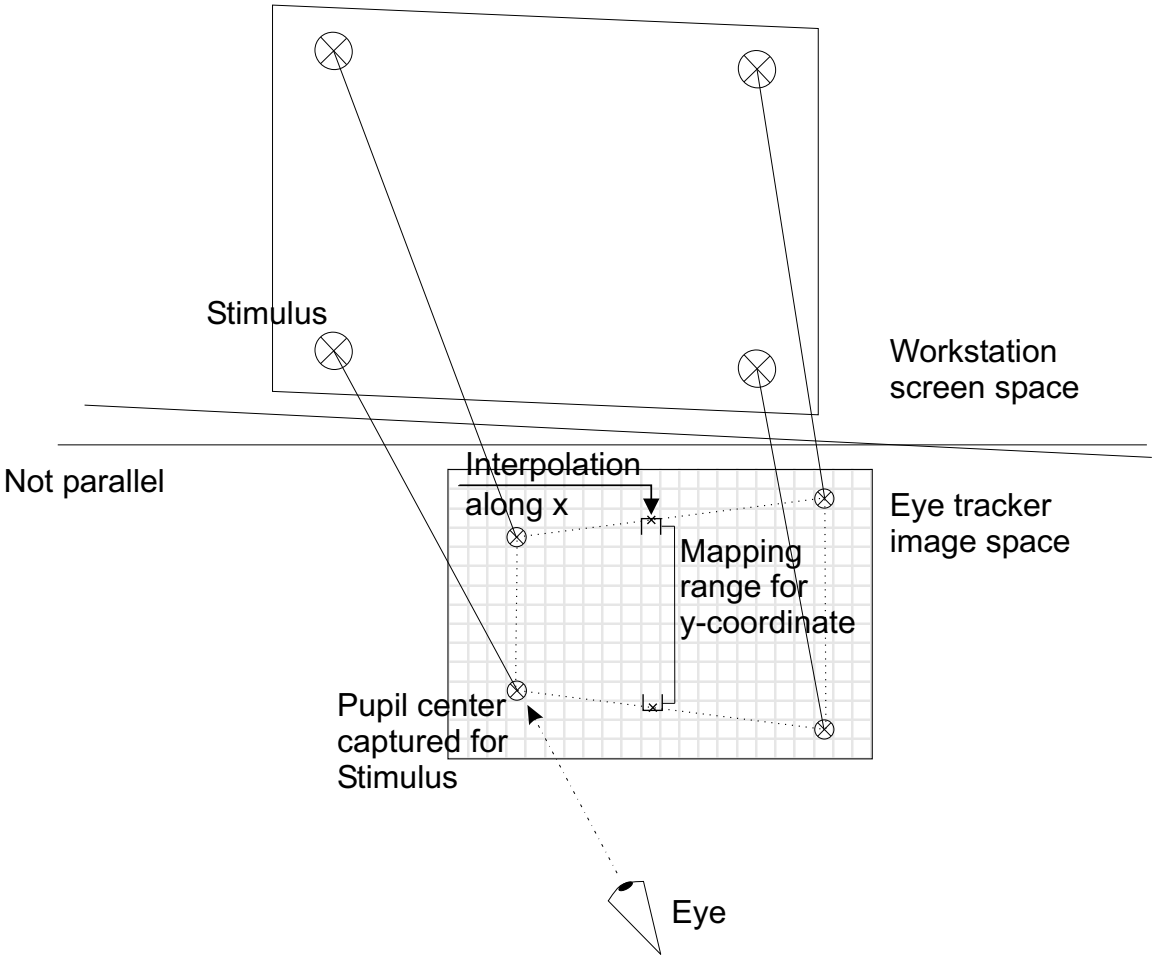


Figure 4.8: Perspective distortion of the calibration grid.

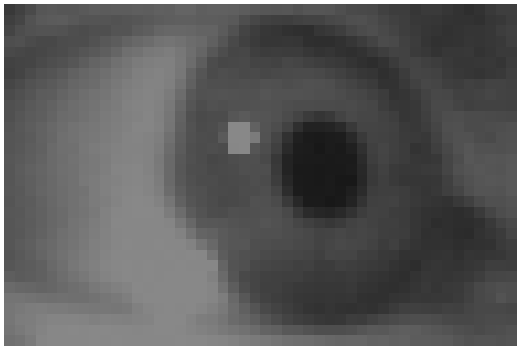


Figure 4.9: Grayscale Image 48 x 32 pixels.

As the tilt is affected by the user's head position, this part of the library could be modified to incorporate head tracker information. Currently, the head is fixed and corner points of the depicted quadrilateral are assumed to be fixed.

Any rotational component around the straightforward direction, is not in particular compensated. Also, the perspective distortion that is caused by the projection of the eye onto the VC image space [30] is neglected.

4.3 Microcontroller

4.3.1 Calibration Component

The calibration program is the first binary downloaded into the SR3300. While it is running it performs the following tasks relevant for the host library:

1. Streaming grayscale and binary image video
2. Capturing grayscale and binary images
3. Optimizing binarization threshold
4. Optimizing positional information of calibration points

Streaming grayscale images is important just before calibration is started, to properly adjust the glasses, so that the pupil is clearly visible. The binary image video can be used to verify the thresholds that are automatically calculated. The function to capture single grayscale and binary images, is used during calibration while the host displays visual stimuli on screen, and requests the corresponding image of the eye for analysis. Figure 4.9 shows a typical grayscale image as it is captured during calibration. Figure 4.10 shows a binarization of the grayscale image.

Both streaming and single image capturing are based on Vision Chip functions to evaluate a single pixel of the tracker image space, to get a grayscale or binary value, respectively.

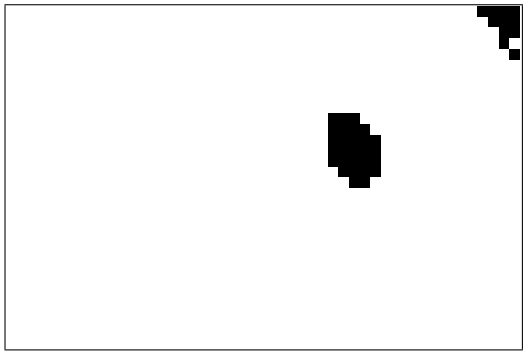


Figure 4.10: Binary Image 48 x 32 pixels.

Optimizing Binarization Threshold. The function to optimize a given binarization threshold is also important during calibration, when the threshold estimated by host image analysis is improved. From the host, the microcontroller receives the estimated threshold along with the number of set pixels that is expected in the resulting binary image.

After that, the MC sets the given threshold for the next exposure. Then, using a Vision Chip function, it determines the number of set pixels after binarization. If this number is still larger than the expected value, the binarization threshold is changed and the procedure repeated.

Optimizing Positional Information. The optimization of the positional information for each calibration point is also based on an estimate determined by the host image analysis. This analysis provides the coordinates of the pupil in pixel resolution. These coordinates are received by the calibration component, which selects the area at the given position as target. As this is performed while the user is focusing on the on screen stimulus for the respective calibration point, the pupil centroid can be computed in sub-pixel resolution the same way it is done during tracking.

4.3.2 Tracking Component

This program is downloaded into the SR3300 after calibration and is active during the rest of the tracking. It is supplied with an initial position that lies within the pupil area and continues to track the selected area. The greatest difficulties that this program has to solve, arise from the limited computational capabilities that are available at runtime. It is not possible to perform any semantic image analysis. The only means of image analysis is the segmentation performed using binarization, and region growing using the Self Fill algorithm. The means of segmentation internally used by the Vision Chip.

While the eye position is continuously tracked, the binarization threshold is adjusted and the tracked pupil checked for distortions.

Tracking Algorithm. Although the actual implementation details of the program are complicated by the underlying system, the general working of the algorithm is summarized by the pseudo code given in Listing 4.1. The given code is executed

```
1  if (!Distortion)
2  {
3      // recapture pupil
4      f.CalculateFeatures();
5      if (f.SatisfiesConstraints())
6      {
7          Distortion = NONE;
8          Centroid = f.GetCentroid();
9      }
10     else
11     {
12         // try again in next frame
13     }
14 }
15 else
16 {
17     tw.SelfFill(centroid);
18     tw.CalculateFeatures();

20     if (tw.SatisfiesConstraints())
21     {
22         Distortion = NONE;
23     }
24     else
25     {
26         Distortion = tw.GetDistortion();
27     }



30     if (!Distortion)
31     {
32         Centroid = tw.GetCentroid();
33         Threshold = BilerpThreshold(Centroid);
34     }
35     else
36     {
37         // try to recapture in next frame
38     }
39 }



42 ReportToHost(Centroid, Distortion);
```

Listing 4.1: Tracking pseudo code

for each frame, i.e., after each exposure, after capturing an image of the eye.

Besides global variables to save data between consecutive frames, the code uses two pseudo object variables, **f** and **tw**. These objects represent the binary image f and the binary image of the *target window* that were described in Section 3.3. They provide the following methods:

CalculateFeatures() Calculates the image features such as area, centroid and eccentricity. This function takes advantage of the high speed of the VC function to calculate moments.

SatisfiesConstraints() Compares the image features to the constraints provided by the host precomputations and returns true if the current features satisfy the constraints, i.e. the target is the pupil without recognizable distortions. False is returned if distortions are found.

GetDistortion() Returns a code for the found distortion to be able to report eye blinks to the host.

GetCentroid() Returns the centroid of the image, usually the x and y coordinates of the pupil center.

The object representing the target window also provides the method **SelfFill()** to execute the corresponding VC function. The function takes a parameter to define the starting point for the filling. The following global variables and functions are available:

Distortion Represents the distortion found in the frame, if any exists.

Centroid The x and y coordinates of the center of mass.

Threshold The binarization threshold, adjusted for each frame based on the determined centroid.

BilerpThreshold(Centroid) Performs bilinear interpolation for the current pixel position of the pupil and returns the computed binarization threshold.

ReportToHost(Centroid, Distortion) Timestamps the data, assigns a frame number and injects markers before the tracking data is reported to the host workstation.

The Code assumes initialization with a centroid from the host before tracking is started.

4.3.3 Moment Calculation

The calculation of moments is of great importance for the eye tracking system. The MC components take advantage of the VC functions that allow the computation of moments within the time of a single frame. Only using these functions is it possible to compute the pupil centroid with the required frame rate. This section explains

how moments up to the second order can be computed using these functions. The computation of the pupil *eccentricity* will be given as an example using second order moments.

First, some definitions are necessary. For a binary image $g(x, y)$, the geometric moments of 0th, 1st and 2nd ($p + q = 2$ for $m_{p,q}$) order, can be defined as follows:

$$
\begin{aligned}
m_{0,0} &= \sum_x \sum_y g(x, y) \\
m_{1,0} &= \sum_x \sum_y x g(x, y) \\
m_{0,1} &= \sum_x \sum_y y g(x, y) \\
m_{1,1} &= \sum_x \sum_y xy g(x, y) \\
m_{2,0} &= \sum_x \sum_y x^2 g(x, y) \\
m_{0,2} &= \sum_x \sum_y y^2 g(x, y)
\end{aligned}
$$

The 0th order moment $m_{0,0}$ is simply the area. It is equal to the number of all pixels with value 1.

Using these moments, the centroid, or center of mass coordinates can be computed:

$$
\begin{aligned}
\overline{x} &= m_{10}/m_{00} \\
\overline{y} &= m_{01}/m_{00}
\end{aligned}
$$

These values define the pupil position as it is reported by the VC. Furthermore, they can be used to define central moments, of which only the second order moments will be relevant:

$$
\begin{aligned}
\mu_{1,1} &= \sum_x \sum_y (x - \overline{x})(y - \overline{y}) g(x, y) \\
\mu_{2,0} &= \sum_x \sum_y (x - \overline{x})^2 g(x, y) \\
\mu_{0,2} &= \sum_x \sum_y (y - \overline{y})^2 g(x, y)
\end{aligned}
$$

These moments in turn can be calculated using the geometric moments and the centroid coordinates. For example:

$$\mu_{1,1} = m_{1,1} - \overline{y}m_{1,0} - \overline{x}m_{0,1} + \overline{x}\ \overline{y}m_{0,0}$$
$$\mu_{2,0} = m_{2,0} - m_{1,0}^2 + m_{1,0}^2 m_{0,0}$$

This is important, as the VC supports only the calculation of geometric moments. But as central moments can be expressed as above, this does not pose a problem. The central moments can be computed by the microcontroller.

Using central moments, *eccentricity* can be defined [31]:

$$\varepsilon = \frac{\mu_{2,0} + \mu_{0,2} + \sqrt{(\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2}}{\mu_{2,0} + \mu_{0,2} - \sqrt{(\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2}}$$

The eccentricity is the ratio of longest and shortest line that span the region. Consequently, for the circular shape of the pupil, ε would be 1.

Implementation. Moment calculation is performed partially by the VC and partially by microcontroller routines that take advantage of special VC functionality. The relevant functions are:

- Area(): Computes the area, i.e., counts set pixels, over the area of the binary image.
- SetRow(): Selects all pixels of a given row or set of rows, to be included into the computation performed by Area()
- SetColumn(): Selects all pixels of a given column or set of columns, to be included into the computation performed by Area()
- SetAnd(): Affects the selection performed using SetRow() and SetColumn() as explained below.

As mentioned in Section 3.3, the VC column and row decoders can be used to select certain columns and rows of the VC image. Selecting, e.g., a certain row, includes this row into the area computation that is performed in hardware. All binary image pixels in this row, that have value 1, will provide to the sum that the VC computes. Using this functionality it is possible to select a certain set of rows at once. This is performed by the function SetRow(), or SetColumn(), respectively.

As previously mentioned, to compute $m_{0,0}$, these function are simply used to include all rows into the area computation. For the computation of higher order moments, it is relevant, in which ways, e.g., a set of rows can be specified. Which rows are contained in the set depends on the bit pattern of the row number.

The binary representation for a row x can be defined as $x_n x_{n-1} \cdots x_1$ and

$$\begin{aligned} x &= x_n 2^{n-1} + x_{n-1} 2^{n-2} + \cdots + x_1 2^0 \\ &= \sum_k x_k 2^{k-1} \end{aligned}$$

with k as the index of the binary string.

For the moment calculation, the VC offers the possibility to select all rows that have a bit set at position k of their binary representation. Why this is relevant will be explained shortly shortly.

Keeping the notion of a set of rows, it is also possible to select rows that correspond to the intersection of two sets. Hence, it is possible to select only rows that have a bit set at two positions k *AND* l. It is furthermore possible, to use the same functionality to intersect sets or rows and columns, which will yield a pattern of selected pixels that are part of a row *AND* a column from the sets. This is performed by calling the function SetAnd() before selecting the respective rows and columns.

Having selected the required pixels, the VC can perform the area computation. To see how this is of advantage for the calculation of higher order moments, a different set of formulas has to be derived from the original formulas for moment calculation.

By replacing the coordinates x and y with their binary representation, the following formulas can be defined:

$$m_{1,0} = \sum_k 2^{k-1} \sum_x \sum_y x_k g(x,y)$$
$$m_{0,1} = \sum_k 2^{k-1} \sum_x \sum_y y_k g(x,y)$$
$$m_{1,1} = \sum_k 2^{k-1} \sum_x \sum_y x_k y_k g(x,y)$$
$$m_{2,0} = \sum_k \sum_l 2^{k+l-2} \sum_x \sum_y x_k x_l g(x,y)$$

The formula for $m_{1,0}$ was originally proposed by [26].

These functions are more closely related to the way the moment can be computed using the VC. Using the formula for $m_{1,0}$ as an example, it is visible that the rightmost summations over the x and y is performed for a fixed position k of the binary representation of x. As x_k is either 0 or 1, it selects the pixels that are included in the summation. In other words, pixels in all rows x are included in the summation, if x has a bit set at position k of its binary representation.

This is where the particular way the VC can select rows based on their binary representation becomes relevant. In iteration k of the leftmost summation, when selecting all rows that have a bit set at position k, the following is equivalent to the rightmost summation:

$$\sum_x \sum_y g(x,y)$$

The selection of the rows being performed by calling SetRow(), the rightmost summation can be performed very efficiently by the VC Area() function. Conse-

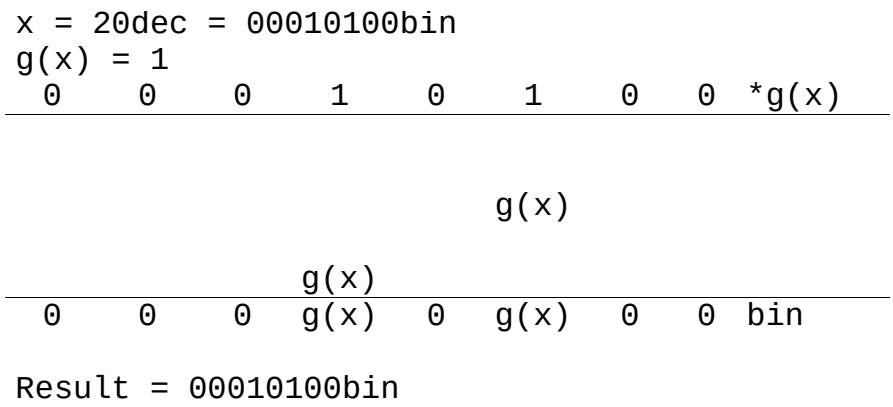


Figure 4.11: Binary multiplication.

quently, moment calculation can mostly be delegated to the VC, the MC only has to perform the remaining left part of the equation.

The computation for $m_{0,1}$ is performed in the same way. To compute $m_{1,1}$ the VC SetAnd() function is called before calling SetRow() and SetColumn() to select the rows *and* columns as defined by k. This is necessary, as the rightmost summation includes only pixels that have a bit set at row x_k and column y_k.

The computation of $m_{2,0}$ is performed in a similar way, only SetRow() and SetColumn() are invoked with different values, k and l, respectively.

This way, the moments can be calculated in $log(N)$ iterations, with N being the maximum of the number of rows and columns.

Validity. To show that the calculated results are valid, it is now only to show that the used formulas are valid, i.e., for example, that the following equality holds for the formulas of $m_{1,0}$:

$$\sum_x \sum_y x g(x,y) = \sum_k 2^{k-1} \sum_x \sum_y x_k g(x,y)$$

By simply substituting x by its binary representation the following formula is derived from the left part:

$$\sum_x \sum_y \left[\sum_k 2^{k-1} x_k \right] g(x,y)$$

where the part marked with square brackets, calculates the value of x from its binary representation. This formula is equal to

$$\sum_x \sum_y \sum_k 2^{k-1} \left[x_k g(x,y) \right]$$

with the definition of the *binary multiplication* explained in Figure 4.11. The variable x_k has the purpose of selecting $g(x,y)$ and 2^{k-1} performs the shift. Therefore, each iteration over k corresponds to a new line in the depicted summation. The final formula follows with commutativity:

$$\sum_k 2^{k-1} \sum_x \sum_y x_k g(x,y)$$

All other formulas can be derived in a similar fashion.

Chapter 5

Conclusion

This thesis demonstrated the implementation of a high speed eye tracking system using the Vision Chip. Section 5.1 summarizes the results acquired with the system and illustrates the problems that were solved. Section 5.2 gives an outlook on possible improvements of the system itself and the SR3300 Vision Chip.

5.1 Results

In respect to assessing the value of the eye tracking system, two properties are of particular interest:

1. Time resolution
2. Spatial resolution or precision

The time resolution is determined by the frame rate. The system provides a frame rate of 100Hz, that is sufficient to capture the occurrence of most saccadic eye movements. The lack of a measuring device as it is described in [17] does not allow to objectively assess the accuracy of the system. Instead the precision is given, the smallest change that the system is able to detect. The precision has been estimated to $<2.3^o$ over a range of 45^o, which is a rather conservative estimate.

Time Resolution. Figure 5.1 shows tracking data recorded while the user was tracing a wave curve with his eyes. The time resolution is visible from Figure 5.2 that shows only the y position over time. The occurrence of saccades is visible in areas with almost vertical slope and very few sample points. This illustrates the need for fast eye-tracking and the inadequacy of smaller frame rates, in respect to saccadic eye movements. Also, the importance of saccades for accurate positional information is visible, as most positional changes are brought about by saccadic movements.

The area marked within Figure 5.2 is shown again in Figure 5.3. This enlarged subsection shows the achieved time resolution of 10ms between consecutive frames, or a frame rate of 100Hz.

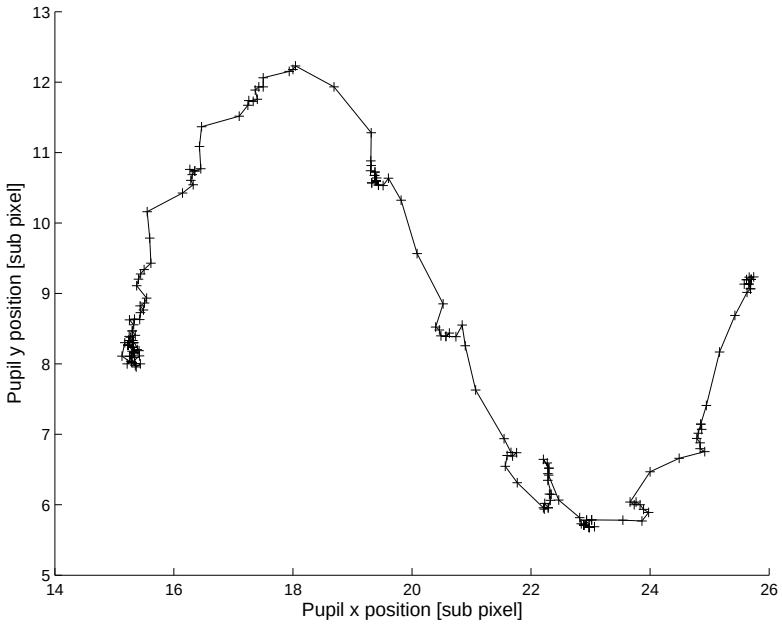


Figure 5.1: Trajectory of pupil xy-position.

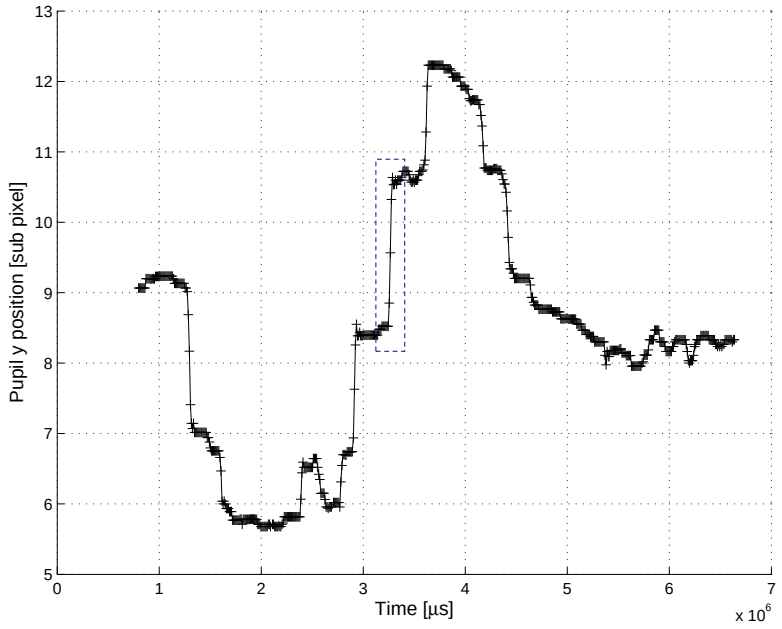


Figure 5.2: Trajectory of pupil y-position over time.

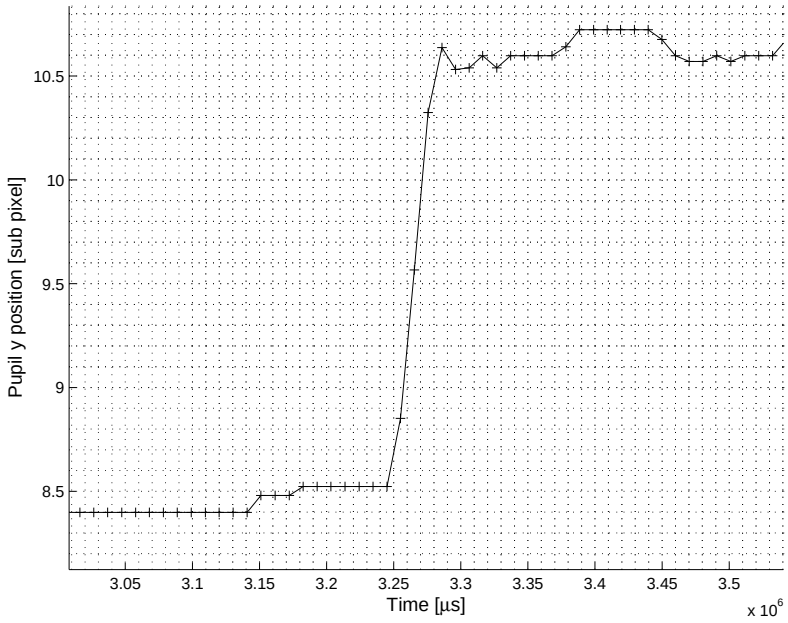


Figure 5.3: Zoomed in trajectory of pupil y-position over time.

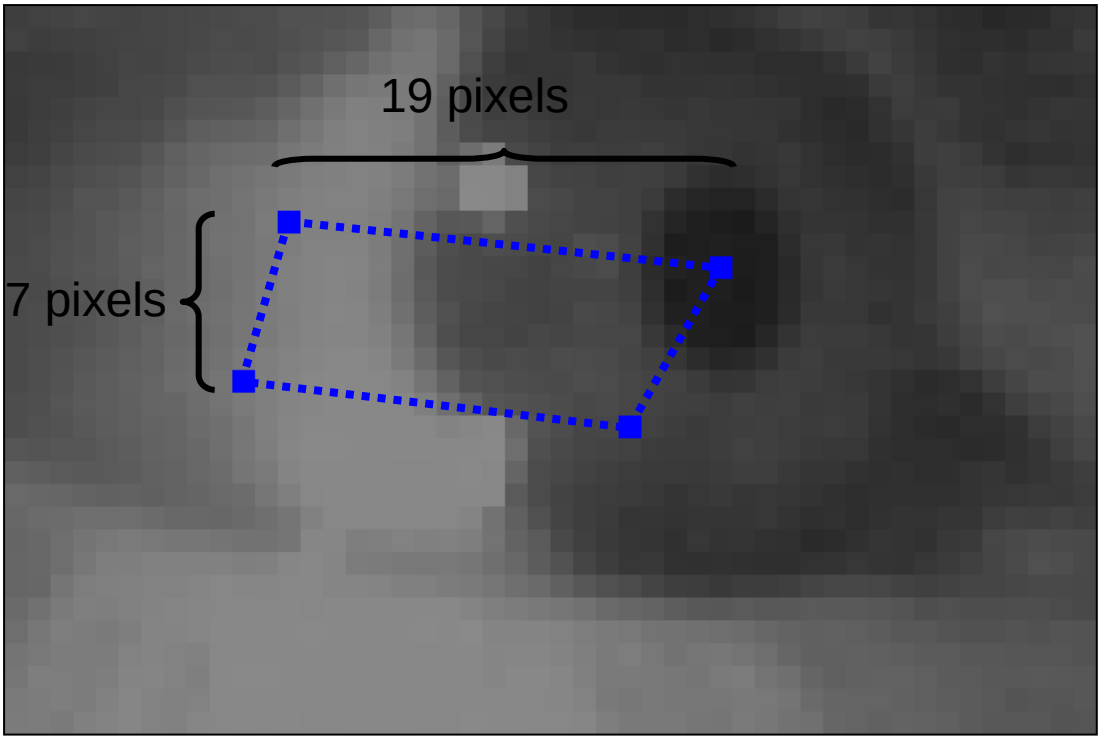


Figure 5.4: Grayscale image captured during calibration. The shape is formed by the four corner calibration points.

Spatial Resolution. Figure 5.4 shows a grayscale image as it is acquired during the calibration, while the user looks at a far corner of the computer screen. The overlayed polygon indicates the maximum extent to which the eye can move into the far corners of the image, as determined by additional calibration images. As mentioned before, the calibration points do not form a rectangle due to misalignments that cannot be generally avoided.

The width and height of the bounding rectangle are given in pixel. Although sub-pixel resolution is used for tracking and mapping into screen space, this indicates the resolution of the tracking device. The depicted polygon with edges of about 19 and 7 pixels length, is mapped onto a computer screen rectangle of about 850 by 450 pixels (25 by 14 centimeters) at a distance of 30 centimeters. Consequently a change by 1 pixel in tracker space corresponds to about 45 pixels in horizontal and 64 pixels in vertical direction in screen space. While the actual positional information in tracker space is provided in sub-pixel resolution, the small extent of the polygon severely limits the resolution of the device. Based on the pixel resolution, the precision can be estimated to $<2.3^o$.

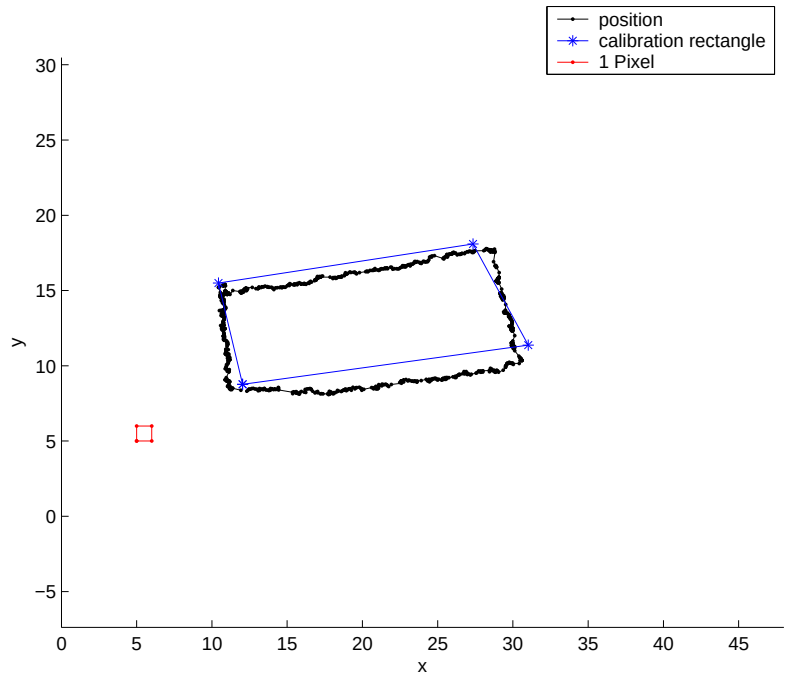
To take the sub-pixel resolution into account is not as straightforward as it may seem. Although the fixed point decimals returned by the VC suggest a precision of $1/256 = 0.004$ with 8 fractional bits, this precision is not achieved.

As the centroid is calculated using area and first order moments, the precision of the calculation depends on the area and shape of the pupil. The degree to which the shape is affected by binarization noise cannot easily be assessed. Ignoring the influence of binarization and assuming a perfectly round shape, the precision can be estimated to $1/4r$ using the double of the number of pixels along the edge of a bounding rectangle. With a radius $r = 4$ this yields a precision of $1/16 = 0.0625$. The resulting estimate for the eye tracking accuracy would be $<0.14^o$ which is a very optimistic value. The actual value will be closer to the conservative estimate.

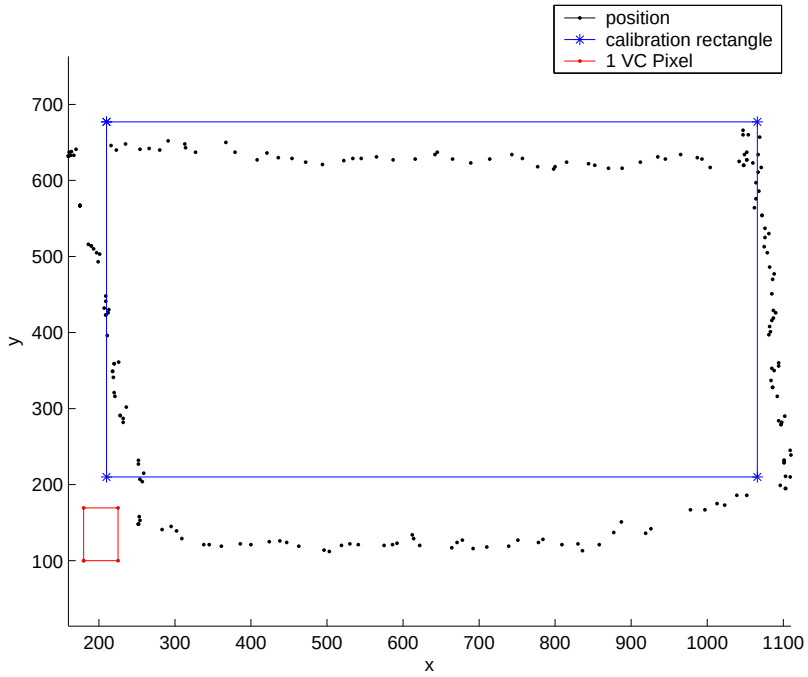
This is also visible in Figure 5.5. It shows tracking data that was taken while the user was tracing the calibration rectangle on screen. The corner calibration points are indicated and connected with lines. Figure 5.5(a) shows the x and y position in VC image space while Figure 5.5(b) shows the mapping into screen coordinates. The size of a VC pixel is indicated in both figures.

Threshold Adjustment. Figure 5.6 compares the tracking application that dynamically adjusts the binarization threshold to one with a fixed threshold. The tracking session is the same as depicted in Figure 5.5. The user injected a marker into the stream of tracking data when his gaze swept across the corner points. Both figures show the measured area of the pupil and the area of the whole binary image. The difference between these two measurements is the amount of clutter contained in the binary image besides the pupil.

While both versions cannot avoid undesired clutter completely, the version with adjustment shown in Figure 5.6(b), can keep it at a reasonable low level. Over large parts of the tracking there is no clutter present in the image, and the pupil area makes up the whole area. In particular, the clutter surrounding the pupil never merged with the pupil area, so that the tracking results remain valid.

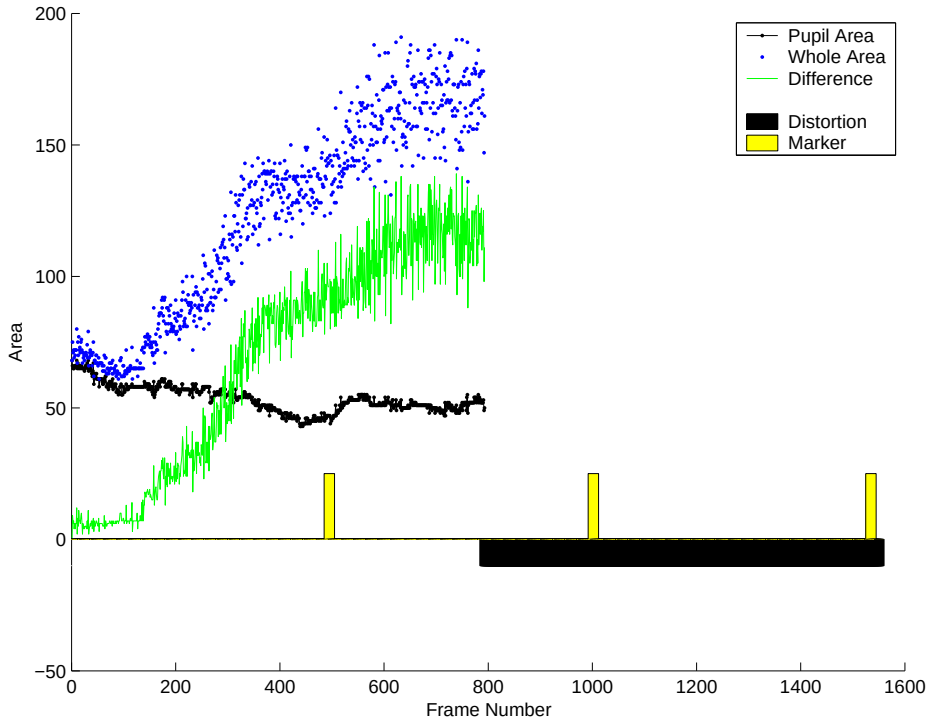


(a) Coordinates in Vision Chip image space.

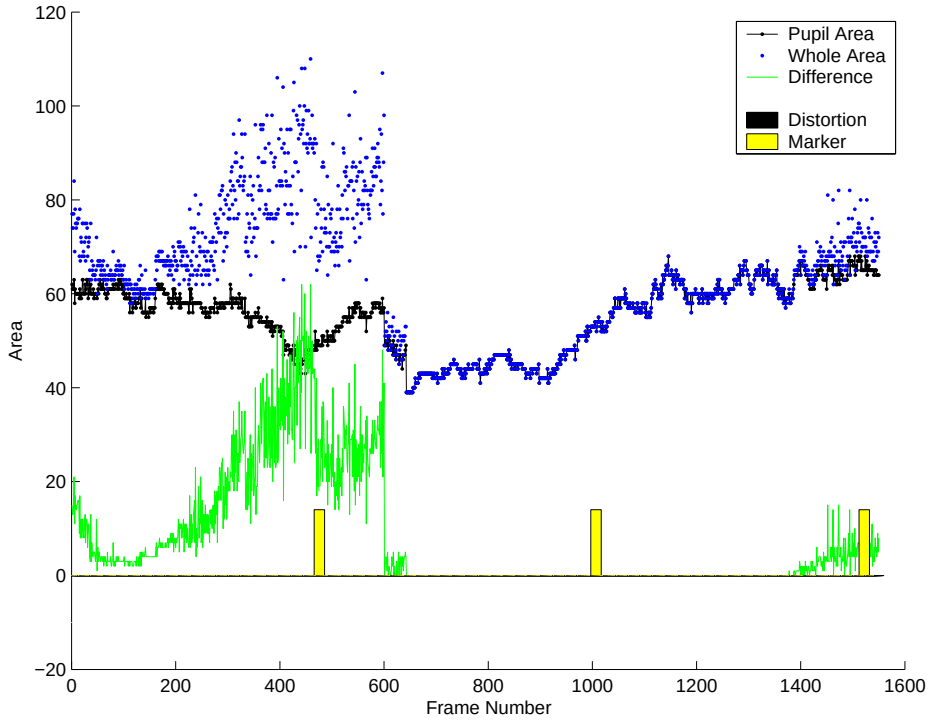


(b) Coordinates in screen space.

Figure 5.5: Tracking data captured while user was tracing rectangle on screen.



(a) Tracking without threshold adjustment.



(b) Tracking with threshold adjustment.

Figure 5.6: Measurement of the pupil area and whole area of the binary image. The samples were taken while the user traced a rectangle. Figure 5.6(a) shows a failed tracking attempt when the threshold is not adjusted. Figure 5.6(b) shows a successful tracking session with dynamic adjustment of the binarization threshold.

Figure 5.6(a) on the other hand, shows an attempt to track the same rectangular movement without adjusting the binarization threshold. Here, the tracking begins successfully, but soon the amount of clutter increases, and half way to the second marker, the pupil is distorted by the clutter and the tracker appropriately reports the distortion.

5.2 Future Work

5.2.1 Improving The Eye Tracking System

Many things remain to be done, some interesting solutions remain to be evaluated. Some of the following points would require hardware changes as they are described in the next section. Generally it is likely that increased pixel resolution is a requirement to justify any further work. Increased MC memory is a definite precondition. The next section gives more details in that respect.

Presuming some of the asked for hardware changes, the following would be worthwhile to pursue:

1. Increase accuracy by improving calibration. Use more calibration points. Requires speed improvement for host image analysis.

2. Increase accuracy by compensating effects of sphere like shape of the eye [30].

3. Reduce load on host workstation by reporting only detected saccades. In the current system, the host polls the USB connection "in a tight loop" to keep up with the MC that transfers frames at 100Hz. This is necessary to make the host aware of saccades, that can occur at any time. Detecting saccades on the MC would allow for substantially lower transfer rates. Only during saccades the high frame rate is necessary to capture the movement, otherwise a lower frame rate can be chosen. As saccades occur only about 3 times per second, the overall load on the host workstation could be largely reduced.

4. Use higher grid resolution for bilinear threshold interpolation and evaluate impact on robustness, e.g. in relation to increased frame rate.

5. Increase frame rate. More research is needed to analyze reasons for failed tracking with higher frame rates. Necessarily requires larger programs, e.g. to acquire grayscale images from within tracking component.

6. Reduce sensitivity to choice of binarization threshold by evaluating other approaches than dark pupil. For example the Blueeyes approach [13] could be interesting but requires grayscale image analysis.

7. Change tracking technique to use first Purkinje image. Simply capturing the highlight every other frame would be a solution but cuts frame rate in half. More sophisticated solutions might be able to keep the frame rate. Making this improvement, it might be possible to fix VC at screen and gain more usability without the need to fix the user's head position.

8. Compensate the effect of the LED highlights on the moment calculation. Currently the area of the LED is missing in the binary image and distorts the shape of the pupil. Compensation might be possible by taking advantage of the separability of moment calculation. The moments of the highlights could be precomputed and included in the bilinear interpolation to adjust the moment at runtime.

5.2.2 Improving The SR3300

Using the SR3300 for the implementation of the presented eye tracking system has revealed certain disadvantages but also showed that high speed eye tracking is possible with hardware that was not specifically designed for the task. Although the sale of the SR3300 has been suspended, development and improvement of the underlying concept continues, and new chips are being designed by the Ishikawa Namiki Komuro Laboratory.

The experience made during the development of the eye tracking system suggest improvements in the following area:

1. Pixel resolution
2. Memory available for program code
3. Lens distortion
4. Grayscale image access
5. Development board
6. Spectral sensitivity

For the presented system, and similar tracking tasks in general, improvements in these area would be beneficial.

Pixel Resolution. The pixel resolution of 48 x 32 pixels that is available from the SR3300 was the most limiting factor in respect to providing a competitive eye tracking system. Experiments have been done to work around this problem by replacing the lens with an optical zoom, but the problem remains, as the extent of the eye movements within the image limits the zooming factor. Only increasing the pixel resolution can make the system competitive in comparison to other systems that use resolutions of 640 by 480 pixels [16].

Memory. The factor that most severely limited robustness and implemented features is the small memory size available to the microcontroller program. The limitation is partially due to the fact that the USB framework, VC library and application code share the available memory of 8KB. Nevertheless, if the eye tracking system is to be improved, increasing the size of the memory is inevitable. Although the microcontroller code is implemented in separate programs that are exchanged

at runtime, even these programs do not allow for further code addition due to the size limitation.

Lens Distortion. Only tracking tasks like tracking a black target on a white background are not affected by the lens distortion. If the VC is to be used for applications that are rather "grayscale" in nature, the lens distortions complicates the software and therefore reduces the number of possible features. Reducing the distortion would be of great value.

Development Board. A Development board integrates the hardware with debugging and simulation applications on the host. The availability of such a board would speed up development and would also aid in tracking problems in the hardware itself. Even if the sold product does not include more microcontroller memory, the development board should. The low level nature of the tracking task makes it difficult to theoretically develop algorithms without putting them to the test. Once a solution is verified, an attempt to optimize the code and reduce code size is more likely to be successful.

Grayscale Image Access. With the current system, grayscale images can not be used for time critical aspects. The pixel access is by magnitudes to slow and the tracking is restricted to use only binary images. Speeding up grayscale image access to a level that makes the microcontroller speed the limitation, would already be an improvement. Escalation procedures such as recapturing the pupil could implement more sophisticated image analysis. Also, occasional checks with more elaborate computations could increase robustness while tracking is performed at higher speed.

Illumination and Spectral Sensitivity. Safety standards and usability limit the illumination and the high frame rates limit its effect. Increasing the spectral sensitivity would enable the system to use higher frame rates while leaving other parameters unchanged.

Bibliography

[1] Paul C. Knox. The parameters of eye movement. `http://www.liv.ac.uk/~pcknox/teaching/Eymovs/params.htm` [Last Accessed Aug. 12, 2005].

[2] R.H.S. Carpenter. *Movement Of The Eyes (2nd edition)*. Pion Limited, 1988.

[3] Keith S. Karn. "saccade pickers" vs. "fixation pickers": the effect of eye tracking instrumentation on research. In *ETRA*, pages 87–88, 2000.

[4] Stavri G. Nikolov, Timothy D. Newman, David R. Bull, Cedric Nishan Canagarajah, Michael G. Jones, and Iain D. Gilchrist. Gaze-contingent display using texture mapping and openGL: system and applications. In *ETRA*, pages 11–18, 2004.

[5] Junji Watanabe, Hideyuki Ando, Taro Madea, and Susumu Tachi. Gaze-triggered selective information display. In *Proceedings of International Conference on Advances in Computer Entertainment Technology ACE*, pages 10–17, 2004.

[6] Asim Smailagic and Daniel Siewiorek. Application design for wearable and context-aware computers, 2002.

[7] Monte System Corporation. Price list on file.

[8] Staff. Sales presentation of 100 Hz eye tracking system.

[9] D.F. Nodine, L. Toto Kundel, and E.A. Krupinski. Recording and analyzing eye-position data using a microcomputer workstation. *Behavior Research Methods*, 24(3):475–584, 1992.

[10] H. Collewijn, F. van der Mark, and T.C. Jansen. Precise recording of human eye movements. *Vision Research*, 15(3):447–450, March 1975.

[11] Sohel Merchant. Eye movement research in aviation and commercially available eye trackers today. `http://66.102.7.104/search?q=cache:cck0ySFXivQJ:arrow.win.ecn.uiowa.edu/56245/FinalEyeTrackingReportAug17.pdf+eye+tracking+companies&hl=en` [Last Accessed Aug. 12, 2005], August 2001.

[12] Shumeet Baluja and Dean Pomerleau. Non-intrusive gaze tracking using artificial neural networks. Technical Report CMU-CS-94-102, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, January 1994.

[13] C.H. Morimoto, D. Koons, A. Amir, and M. Flickner. Frame-rate pupil detector and gaze tracker. http://www.ime.usp.br/~hitoshi/framerate/node1.html [Last Accessed Aug. 12, 2005].

[14] Jason S. Babcock and Jeff B. Pelz. Building a lightweight eyetracking headgear. In *ETRA*, pages 109–114, 2004.

[15] Alphabio. Alphabio's eyeputer technical details. http://www.electronica.fr/alphabio/page1.html#prin [Last Accessed Aug. 12, 2005].

[16] SR Research. EyeLink II technical specification. http://www.eyelinkinfo.com/mount_tech_spec.php [Last Accessed Aug. 12, 2005].

[17] Clarke A.H., Ditterich J., Drüen K., Schönfeld U., and Steineke C. Using high frame rate CMOS sensors for three-dimensional eye tracking. *Behavior Research Methods, Instruments, & Computers*, 34(4):549–560, November 2002.

[18] Chronos Vision GmbH. Eyetracking. http://www.chronos-vision.de/eyetracking/default_start_eyetracking.htm [Last Accessed Aug. 12, 2005].

[19] M. I. Posner, C. R. R. Snyder, and B. J. Davidson. Attention and the detection of signals. *Journal of Experimental Psychology: General*, 109:160–174, 1980.

[20] EnchantedLearning. Eye diagram. http://www.enchantedlearning.com/subjects/anatomy/eye/label/labeleye.shtml [Last Accessed Aug. 12, 2005].

[21] George Mather. The eye. http://www.lifesci.sussex.ac.uk/home/George_Mather/Linked%20Pages/Physiol/The%20Eye.html [Last Accessed Aug. 12, 2005].

[22] Arne John Glenstrup and Theo Engell-Nielsen. Eye controlled media: Present and future state. http://www.diku.dk/~panic/eyegaze/article.html [Last Accessed Aug. 12, 2005].

[23] Andrew T. Duchowski. *Eye Tracking Methodology. Theory and Practice.* Springer-Verlag, 2003.

[24] Fourward Technologies Inc. Dual-purkinje-image concept. http://www.fourward.com/dconcept.htm [Last Accessed Aug. 12, 2005].

[25] Guidelines on limits of exposure to broad-band incoherent optical radiation (0.38 to 3m). *Health Physics*, 73(3):539–554, 1997.

[26] Takashi Komuro, Idaku Ishii, Masatoshi Ishikawa, and Atsushi Yoshida. A digital vision chip specialized for high-speed target tracking. *IEEE transaction on Electron Devices*, 50:191–199, 2003.

[27] Bernd Jähne. *Digital Image Processing: Concepts, Algorithms, and Scientific Applications.* Springer-Verlag, 4th edition, 1997.

[28] Mark Russinovich. Windows 2000 quantums. `http://www.sysinternals.com/Information/Windows2000Quantums.html` [Last Accessed Aug. 12, 2005].

[29] Linda G. Shapiro and George C. Stockmann. *Computer Vision*. Prentice Hall, 2001.

[30] Kai Schreiber and Thomas Haslwanter. Improving calibration of 3-d video oculography systems. *IEEE Transactions on Biomedical Engineering*, 51(4), April 2004.

[31] David Marshall. Statistical region description. `http://www.cs.cf.ac.uk/Dave/Vision_lecture/node36.html` [Last Accessed Aug. 12, 2005], 1997.

Appendix A

User's Manual

This appendix explains how to use the demonstration program that demonstrates the eye tracking system. The program is compiled from the EyeTrackerTest project. For system requirements see Appendix B.1.

The SR3300 Eye Tracker has to be connected before the program is started. Once the program is started, all options are available from the Test menu:

- Mockup Test
- Rectangle Trace

The Mockup Test was used during development and can be used to verify the tracker if problems occur while tracking a human eye. It simulates the pupil on screen for a tracker mounted in front of the screen. See Appendix B.4 for more information.

If a demonstration is started, first a video image of the eye is shown in a dialog. The user has to adjust the glasses until the pupil is centered in the image and a blue dot within the area of the pupil indicates that image analysis is performed successfully. If everything is alright, a click on OK will start the calibration.

During this phase, 5 black circles, the stimuli, are shown on screen. For each circle, calibration has to be performed. Once the user focuses on the displayed stimulus, the calibration is performed by pressing the space bar. The user has to keep focusing on the stimulus until the next stimulus is displayed.

After calibration, a dialog with a binary image video is shown. To start tracking, the user has to focus on the last displayed stimulus and press space again.

If the option "Rectangle Trace" has been chosen, the black circle will move along the edge of a rectangle.

During tracking, the tracked eye positions is indicated by a small blue square on the screen. A marker can be injected into the stream of tracking data by pressing the key "m". This can be used to identify a certain screen position in the tracking data file. The data is written to the file tracking.log. The following shows several lines from this file, followed by an description of each field:

```
120: (18) 134037, 4.5, 24.7695, 10315, pA44, wA44, nA28, dis0, t131, 7992,
```

```
121: (18) 144357, 4.5, 24.7695, 10320, pA44, wA44, nA28, dis0, t131, 12017,
122: (18) 154648, 4.5, 24.7695, 10291, pA44, wA44, nA28, dis0, t131, 7970,
123: (18) 165064, 4.5, 24.7695, 10416, pA44, wA44, nA28, dis0, t131, 12009,
124: (18) 175269, 4.5, 24.7695, 10205, pA44, wA44, nA28, dis0, t131, 11998,


frame number: (message length) timestamp, x, y, time since last frame

The message length is in bytes, the timestamp in microseconds.
The coordinates x and y are given in sub-pixel resolution.
The time since last frame is given in microseconds.

The  meaning of the remaining fields is as follows:
pA:  Pupil area.
wA:  Whole area. The number of all set pixels in the binary image.
nA:  Normal area. An expected value for the pupil area.
dis: Distortion. 0 indicates no distortion, 1 and 2 indicate blink
t:   Threshold that was used for the frame

The last value gives a time measure performed by the host.
It gives the time in microseconds since the last polling of data.
```

The demonstration displays the following information on screen:

```
==Calibration data==
x and y: The centroid coordinates in VC image space
threshold: The ideal threshold that was computed for this position

==Tracker status==
frame: The current frame number (not updated with 100Hz)
binThresh: The binarization threshold that is currently used
distortion: The distortion that was detected
```

Appendix B

Maintenance Manual

This Appendix is for developers who want to compile and execute the source code. On the `Release CD` the source code is located in directory `Source`. For convenience the source is expected to be located on drive `S:/` on the development system. There are the following directories with source code:

```
host/
    EyeTrackerTest: The demonstration program.
    EyeTrackerLib:  Library Encapsulating the access to the tracker.
    ImageAnalysis:  Library for host image analysis.
    CustomControls: Library for user interface functionality.
    common:         Source commonly used in all windows programs.
    AlgorithmTest:  Project for testing microcontroller source.
    mlfTest:        Project to test matlab functions of
                    ImageAnalysis library.
target/
    Tracking:       The component performing the actual tracking.
    Calibration:    The calibration component.
shared:             Files shared by host and target applications.
```

The Windows code is located in directory host, all microcontroller code is located in directory target. Code that is shared by Windows and microcontroller applications is located in the directory shared.

A release version is not provided. The following instructions refer to building and executing the Windows programs from MS Visual Studio as debug versions. The microcontroller programs are built using Keil uVision2. The resulting microcontroller binaries with the ending bix have to be located in the directory S:/EyeTrackerTest, from where they are read by the Windows application to be downloaded into the SR3300. There are two such files:

- Calibration.bix
- Tracking.bix

For convenience the following drives are expected to be present in the system:

S:/ Contains the source code as described above L:/ Contains all libraries M:/ Contains the project matlab image analysis

Because of a bug in the matlab mcc compiler the drive M:/ has to refer to S:/host/ImageAnalysis. This is described in Section B.2.1.1.

The remaining sections describe the following:

- Section B.1: Required system setup.
- Section B.2: How the source code is built.
- Section B.3: How the source can be modified and extended.
- Section B.4: How the system can be tested.

B.1 Prerequisite System Setup

To be able to compile, but also to execute the programs, the system has to be set up appropriately. This section describes all necessary steps. First all required libraries are described and where they should be placed. Next, all required installations are described, including the libraries. This gives information where the needed tools and libraries are available.

B.1.1 Required Libraries

The development system requires certain libraries. Where the required libraries can be found is described in the next section. This section gives details about libraries that might require some attention and describes to which locations the library should be installed.

The following list indicates where the paths are hardcoded (`[MC]`: microcontroller project, `[WIN]`: Windows project, `[PATH]`: system environment path):

```
[MC]   L:/Keil/C51/BIN/
[MC]   L:/Keil/C51/INC/
[MC]   L:/Keil/C51/LIB/


[MC]   L:/Cypress/Target/Lib/ {for Ezusb/Ezusb.lib and USBJmpTb.OBJ}
[MC]   L:/Cypress/Target/Inc/

[WIN]  L:/libs/GraphicsMagick-1.1.4
[WIN]  L:/libs/GraphicsMagick-1.1.4/Magick++/lib
[WIN]  L:/libs/GraphicsMagick-1.1.4/VisualMagick/lib
[PATH] L:/libs/GraphicsMagick-1.1.4/VisualMagick/bin/

[WIN]  L:/Matlab/extern/include
[WIN]  L:/Matlab/simulink/include
[WIN]  L:/Matlab/extern/lib/win32/microsoft/msvc60/
```

```
[WIN]  T:/Debug/dlls
[WIN]  T:/Debug/libs
[WIN]  T:/Debug/includes
```

The last three directories have to be *created* before compilation. The others are created when the library or software is installed. The respective libraries are described in the following sections.

B.1.1.1 Graphics Magick

The Windows applications need the magick library. A compiled version is already included on the `Release CD` in directory `Misc/Libs`, but the latest version can be downloaded from `http://www.graphicsmagick.org/www/download.html`. It is expected in the directory L:/libs/GraphicsMagick-1.1.4. You have to add the following to the systems environment path:

L:/libs/GraphicsMagick-1.1.4/VisualMagick/bin/. The following dlls are relevant:

```
CORE_DB_magick_.dll
CORE_DB_Magick++_.dll
CORE_DB_zlib_.dll
CORE_DB_bzlib_.dll
CORE_DB_lcms_.dll
CORE_DB_ttf_.dll
```

DB refers to the debugging libraries, RL refers to the respective release libraries.

Compiling The Magick Source. If recompilation is necessary, Visual Studio 6.0 Service Pack 5 has to be installed to avoid the internal compiler error C1001.

To compile the library, do the following or look at L:/libs/GraphicsMagick-1.1.4/INSTALL-windows.txt:

Configuration. Go to L:/libs/GraphicsMagick-1.1.4/VisualMagick/configure/ open the configure.dsw file, build release (AND DEBUG) and execute. That starts the configuration wizard you just built. Select checkboxes

- Include all demo and test programs
- Generate all utility projects with full paths rather then relative paths

paragraphCompilation After finishing the configuration, add the line

```
#define snprintf _snprintf
```

to the file L:/GraphicsMagick-1.1.4/magick/magick_config.h To avoid the link error

```
module.obj : error LNK2001: unresolved external symbol  \_snprintf
```

when compiling the Core_magick project. Now open the file L:/libs/GraphicsMagick-1.1.4/VisualMagick/VisualDynamicMT.dsw, and Select the "All" project, Clean, and build as Release.

Testing. For testing the build, go to L:/libs/GraphicsMagick-1.1.4/VisualMagick/bin/, and look at

- GraphicsMagick/Magick++/demo
- GraphicsMagick/Magick++/tests

B.1.1.2 Keil Libraries

The Keil libraries are required by the MC code. The compiler libraries are installed together with the Keil uVision2 IDE. They are expected to be located at L:/Keil.

B.1.1.3 Cypress EZUSB

This library is required by the MC code. The library is expected to be located at L:/Cypress.

B.1.1.4 Matlab libraries for mcc

This library is required by the image analysis component of the windows application. The files are installed as part of Matlab. Matlab version 6.5 is required with toolboxes for image analysis, and the matlab C compiler mcc. It is expected to be located at L:/Matlab.

B.1.2 Required Installs

The following tools, drivers and libraries have to be installed to the system:

Microsoft Visual Studio 6.0 It is required for the compilation of the source code located in the host directory.

Keil uVision2 For binaries larger than 4KB, the trial version supplied with the EZUSB development kit, cannot be used. It is necessary to purchase the version available from `http://www.keil.com/c51/ca51kit.htm`.

EZUSB SDK This sdk is required by the MC code. It can be found on the `Release CD` in directory the `Misc/Tools`.

SR3300 USB driver To be able to access the tracking hardware, the USB driver has to be installed from the directory `Misc/SR3300/Driver` on the `Release CD`.

Cygwin It is required as the projects execute some post-build commands that are not available on a standard Windows system. It can be found at `http://www.cygwin.com/`

Perl It is required by some Matlab scripts to extract data from the tracking data file. This is only required if the tracking data is to be visualized, but not for the demonstration of the tracking system. The version installed on the development system is available from `http://aspn.activestate.com/ASPN/Downloads/ActivePerl/`.

Graphics Magick This library is required by the Windows application. It can be found on the `Release CD` in directory `Misc/Libs`. The previous section gives some more details if the library has to be rebuilt.

B.2 Building The Programs

B.2.1 Building The Windows Applications

If all libraries are installed in the previously described locations, the Windows source can be built using the provided Visual Studio project files. The remaining warnings during compilation are due to a Visual Studio bug (`http://support.microsoft.com/default.aspx?scid=kb;EN-US;q167355`).

First the libraries have to be built:

1. ImageAnalysis
2. EyeTrackerLib
3. CustomControls

Each library copies certain files to the following directories that have to be created beforehand:

```
T:/Debug/dlls
T:/Debug/includes
T:/Debug/libs
```

The libraries have to be built in the given order. Please refer to Section B.2.1.1 for instructions on building the ImageAnalysis library.

When all libraries have been built successfully, the EyeTrackerTest project can be built.

B.2.1.1 ImageAnalysis library

The image analysis library is implemented in matlab code that is compiled into a shared dll using the Matlab compiler mcc. Once the build process finished successfully, the image analysis functionality is accessed simply by using the created header and lib files.

Unfortunately, it is not as easy as it should be to reach that point. There are quite a few problems, one of them being a mcc bug that causes the compiler to crash if the parameter list is too long. The parameter list is automatically created by the Matlab Visual Studio plugin. Unfortunately, the plugin includes the source directory many times into the parameter list, which can cause the compiler to crash.

The simple workaround is to access the ImageAnalysis project via the drive M:/ and add new files only from this directory. This way the project files can be used but the parameter list is shorter. It also works around the problem that the tool used for creating the project adds absolute pathnames to the project.

Section B.3.1 describes the steps that are necessary to create such a Visual Studio project to compile matlab code. It also describes what changes are necessary to the source files in case new matlab files are added.

Before building the project, execute the following in your matlab environment:

```
movefile([matlabroot
'/toolbox/images/images/private/check*'],[matlabroot
'/toolbox/images/images/'])
rehash toolbox
```

Otherwise the check* functions produce runtime errors and compile warnings. See This has to be done only once. Otherwise the files in the ImageAnalysis directory can be compiled as is. Simply go to M:/ and open the project file and build. Because the plugin does "things" when the project is cleaned, it is recommended to change one of the c source files, just to make the compiler rebuild the library, instead of performing a clean step.

B.2.2 Building Microcontroller Programs

The microcontroller binaries are created using the respective uVision2 project with the ending Uv2. If the libraries where installed to the required locations, the projects can be used directly.

If changes have to be made to these settings, they can be found at the interesting location "Project-¿Components, Environments, Books-¿Folders/Extensions" in the Keil uVision2 IDE.

Changing MC Memory Model. If the memory model (SMALL, LARGE, TINY) of the MC programs has to be changed, the cypress library has to be rebuilt. The file buildCypressLib.bat located in the target directory does just that. It is used to build the cypress library for different memory models (SMALL, LARGE, TINY). The file is currently set to compile SMALL model files.

Installing Cypress to a different location. If you chose to install to a different folder, the Keil .uV2 build files will not function correctly since the path is hardcoded in the .uV2 files as mentioned above. Also, you need to modify the file buildCypressLib.bat located in the target directory for the same reason. It might be necessary to take a look at L:/Cypress/Bin/setenv.bat.

B.3 Making Changes To The Source Code

B.3.1 Use Matlab functionality from a C/C++ program

The ImageAnalysis uses compiled matlab code. The matlab functionality is implemented in m files that are compiled into header files, a lib and dll file that can be used from a C/C++ program. The Matlab Visual Studio Plugin has to be installed for this purpose. The installation is described at `http://www-rohan.sdsu.edu/doc/matlab/toolbox/compiler/ch04st18.html` and a little help can be found at `http://www.codeproject.com/macro/using_matlab_add_in.asp`.

While theoretically a great idea, the whole approach offers many opportunities for problems.

The following sections proceed from the least invasive, changing the existing project, to adding new files, to creating a new project and describe the involved problems.

B.3.1.1 Making Changes to the m files

If the image analysis functionality is changed by modifying the m files, problems can occur although the m code has been successfully tested in the matlab environment. One cause for trouble is the fact that the using C++ program will just exit in many error cases. One reason might be that the changed require some function call that is not available. This is not visible at compile time, but only at runtime, "indicated" by the applications sudden shutdown.

Therefore when making changes, the according functionality should be tested using a console program to find out about missing dlls and other issues that cause the application to exit. mlfTest is intended for that purpose.

Compile Time Changes. In addition, after the m files have been modified, new versions of the c files are created and added to the project. These files have to be altered. Search for mclCExecMexFunction(...). The parameters to this function use a path that causes shutdowns. Simply perform the following replacement:

```
'images/private/imfilter_mex.dll'
 -> 'imfilter_mex.dll'
```

B.3.1.2 Adding A New m File To The Project

It is easily described: Just press the m++ button of the Matlab Visual Studio Plugin and select the new file. If it does not work and DevStudio crashes, well, just hang in there. Sometimes it helps to create a new project and add all files at once, which will take a long time.

B.3.1.3 Creating A New Matlab Project

Before creating a project it is important to execute the following statements from Matlab shell:

```
movefile([matlabroot
'/toolbox/images/images/private/check*'],[matlabroot
'/toolbox/images/images/'])
rehash toolbox
```

Otherwise the check* functions produce runtime errors and compile warnings. See `http://www.mathworks.com/support/solutions/data/1-19VC8.html?solution=1-19VC8`. This has to be done only once for all projects.

Now the project can be created. For each Dll there is a VS project created using the matlab plugin:

```
File->New project->Matlab project->Dll
```

This project creates dll, lib and h file for other programs. In this project the path L:/Matlab/bin/win32, should be added using the menu

```
Visual Studio->Tools->Options->Directories->Executable Files
```

The Matlab plugin uses Custom Build functionality, to create c files from m files. These files are then compiled by Visual Studio. In addition, it creates a "driver" c file that contains the hooks necessary for the dll to work. The relevant files that are created are for example:

```
ImageAnalysis.h
ImageAnalysis.lib
ImageAnalysis.dll
```

These have to be included in order to use the image analysis functionality from another project.

Errors. If Building brings unresolved errors because of global matlab variables, the problem can be fixed by adding a c file to the project. For example

```
// file matr.c
#include "matrix.h"
mxArray * GETPTS_AX;
mxArray * GETPTS_FIG;
mxArray * GETPTS_H1;
mxArray * GETPTS_H2;
mxArray * GETPTS_PT1;
```

The following warning can be ignored:

```
Warning: File: stem Line: 78 Column: 6
```

References to "graph2d" will produce a run-time error because it is an undefined function or variable. But stem is a plot function and unless you plot anything, this is not a problem.

Deployment. To make all matlab dlls available on a system without matlab, the matlab Add-in packager can create a zip file, that contains an installation program for the required matlab runtime dlls. Unfortunately the installed dlls are not sufficient. The following dlls have to be added to the runtime directory:

```
L:/matlab/
 toolbox/images/images/applylut.dll
 toolbox/matlab/iofun/dataread.dll
 toolbox/images/images/private/imfilter_mex.dll
 toolbox/images/images/private/imhistc.dll
 toolbox/images/images/private/imlincombc.dll
 toolbox/images/images/private/iptregistry.dll
 toolbox/images/images/imreconstruct.dll
```

B.3.2 Things To Stay Away From

Changes to EZUSB lib. In the EZUSB header files (EzRegs.h:197) are some sbit macros that are commented out. These should not be touched because using these defines crashes the MC application. The reason is explained at `http://www.keil.com/support/man/docs/c51/c51_le_sbittype.htm`

Matlab files on samba network drive. Just don't do it. Matlab misses all file changes and uses the cached functions. I tried all solutions suggested under `http://math.carleton.ca/~help/matlab/MathWorks_R13Doc/techdoc/matlab_prog/ch8_pr12.html` but none did consistently work.

B.4 Testing

Developing for the SR3300 can be difficult. The lack of debugging functionality, little available memory, complications introduced by the USB transfer, problems with the chip itself, and last but not least, the low level nature of the tracking task, pose problems. An important aid in developing the eye tracking system was the *mockup test*. Simulating the pupil on screen, this test provides a repeatable way to validate the functionality of the eye tracking system.

The test goes through all phases that are usually encountered during tracking. It calibrates the system, moves the mockup pupil along defined trajectories and simulates distortions.

To use the test, the SR3300 has to be fixed in front of the screen as it is depicted in Figure B.1. The distance used during development was 5cm, but will vary with screen size and resolution. Therefore, the initially shown rectangle should be used to adjust the SR3300 until it is centered in the image. This can be verified using the grayscale video. The class CStiResTest1 encapsulates all "stimulus response" test functionality.

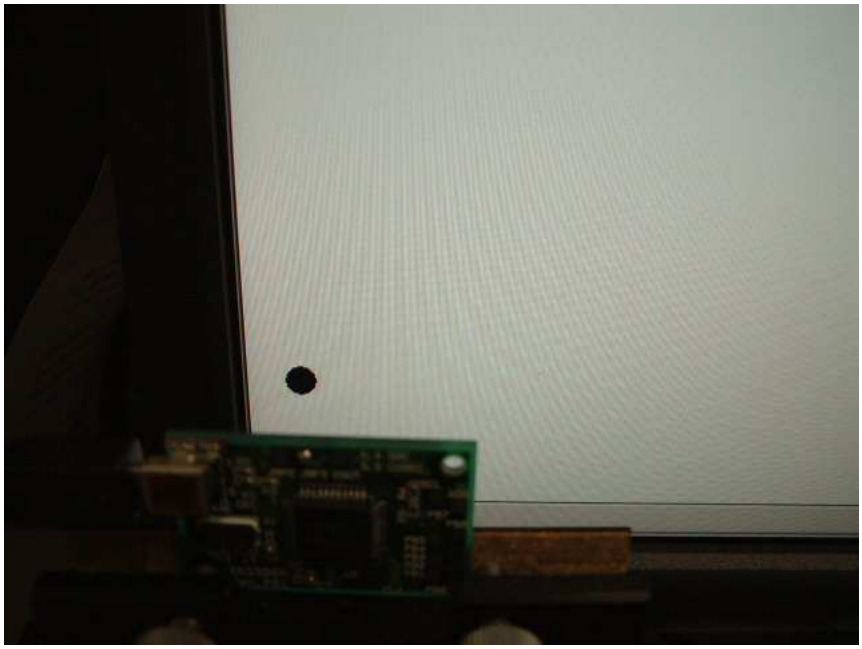


Figure B.1: Mockup Pupil captured by SR3000.

As a final note it should be mentioned that during development, a LCD display was used. It is quite possible that problems will occur with CRT displays. With these displays and the low refresh rate of 50Hz, the pupil might be lost if the Vision Chip image is captured in between refreshs.