IST-2003-511598 (NoE)

COGAIN

Communication by Gaze Interaction

Network of Excellence

Information Society Technologies

D5.5 Silicon retina and other novel approaches for eye tracking

Due date of deliverable: 29.2.2008
Actual submission date: 29.2.2008

Start date of project: 1.9.2004 Duration: 60 months

University of Zürich (UNIZH)

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	x
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Delbrück, T., Grover, D., Gisler, D., Lichtsteiner, P., Ersbøll, B., and Tureczek, A. (2008) **D5.5 Silicon retina and other novel approaches for eye tracking**. Communication by Gaze Interaction (COGAIN), IST-2003-511598: Deliverable 5.5. Available at http://www.cogain.org/results/reports/COGAIN-D5.5.pdf

Contributors: Tobi Delbrück (contact person), Inst. of Neuroinformatics, UNI-ETH Zurich (UNIZH)
Dale Grover, Dept of Electrical Engineering, University of Michigan (external partner)
Damian Gisler, Dept. of Physics, ETH Zurich (UNIZH)
Patrick Lichtsteiner, Inst. of Neuroinformatics, UNI-ETH Zurich (UNIZH)
Bjarne Ersbøll, Image Analysis & Computer Graphics, Informatics and Mathematical Modelling, Technical University of Denmark (TUD)
Alex Tureczek, Technical University of Denmark (TUD)

Table of Contents

Executive Summary

Conventional gaze tracking is carried out using conventional frame-based video cameras. The frames captured from the cameras are processed to extract gaze direction using machine vision techniques based on the sequence of image frames. These methods are so expensive computationally that they tax even contemporary PCs to their limit. A state-of-the-art eye tracker requires 100% of a PC and an expensive, high frame rate, high resolution video camera and frame grabber. One of the objectives of WP5 was to facilitate exploration of non-conventional approaches for eye, gaze, or head tracking that are based on novel hardware which can potentially reduce the computational cost of eye tracking to enable embedded portable eye tracking. These studies fell into two categories:

- The first set of studies used a newly developed silicon retina Dynamic Vision Sensor (DVS) for pupil tracking in a scenario where the sensor was head-mounted above one eye. These studies were carried out by partner UNIZH.
- The second category developed the non-imaging Owl gaze sensor. This development occurred by external collaborator Dale Grover from the University of Michigan.

Both approaches are still research projects that require further development. They offer the possibility for greatly reduced computational cost, at the cost of reduced precision and more suitability for feedback control by the user.

The Owl type sensor would be best suited for selection tasks where the user would select one of several (up to 8) possible directions by looking in that direction.

The silicon retina type sensor would be best suited for situations requiring more general fast tracking of eye position at low computational cost, possibly under more relaxed lighting conditions.

Both approaches investigated only the problem in pupil tracking from a head-mounted device. This is in contrast to more evolved gaze trackers that can estimate gaze direction by looking at the user from the location of the computer monitor.

Here we report some details of algorithms and show results. Detailed reports on these studies are attached as appendices to this document.

D5.5 is ongoing but is hampered by the restrictions on using COGAIN funding for paying expenses of hosting masters or summer intern students. Until internal ETH Zürich (UNIZH) or Technical University of Denmark (TUD) students can be attracted to work on this project further progress will not occur. These projects are continually advertised on the institute’s web pages.

The single instance of constructed silicon retina eye tracker hardware is with TUD. Another can be built at UNIZH if needed for project continuation. An instance of the Owl hardware was transferred to the Empirical Inference group of Bernhard Schoelkopf at the Max Planck Inst. of Biological Cybernetics in Tuebingen in Oct. 2005 and they have developed a software interface to it with updated firmware so that it is ready for further development.

1 Novel approaches to eye tracking

Conventional video-based eye trackers have made significant advances over the past 5 years. It is now possible to buy from several manufacturers (e.g. Tobii Technology, SR Research) video-based eye trackers that can track gaze direction either with unfixed head (Tobii tracker) or at high sample rate with chin rest (EyeLink). Still, these systems require very high performance high frame rate high resolution cameras that are expensive and the eye tracking computation taxes state-of-the-art multiprocessor PCs to their limits, making such approaches unfeasible for mobile applications.

Workpackage 5 of COGAIN (WP5) aimed at opening new technologies for eye, head or gaze tracking based on novel technologies that avoid the bottleneck of video-rate processing of high frame rate, high resolution camera data by using new types of sensors that more directly output usable data for gaze tracking. In particular, WP5 encouraged work with two novel methods for eye tracking, one based on using a new type of silicon retina dynamic vision sensor (DVS) [1-4], and the other based on non-imaging gaze tracking based on active eye illumination and gradual reflectance modulation (the Owl sensor) [5, 6]. This deliverable summarizes the progress on these approaches.

1.1 Recent research activity

Eye tracking is of interest to industry and the 2008 IEEE International Solid State Circuits meeting had a paper on a single-chip eye tracker solution [7]. This extremely competitive meeting only accepts papers when they show convincing results and novel approaches, suggesting that there may have been a breakthrough in this area. The results are interesting but not conclusive. This paper reports an 80x60 pixel vision sensor chip with a power consumption of 100 mW built in 0.35 um technology which can do focal-plane processing to localize the center of the of the pupil. Like many trackers, it relies on IR illumination to create a high contrast between pupil and iris. The pixel circuits then do two operations in sequence: glint removal by an expand operation on black pixels and dark iris center localization by a shrink operation after the glint is removed. After these operations, the center of the pupil is localized by circuits on the periphery of the chip that do winner-take-all operations to output the winning “center pixel” X and Y address. The paper claims a sample rate of 5 kHz, but this rate does not include the normal pixel exposure time, which is usually on the order of 10’s of ms. Moreover, this sensor may require relatively uniform illumination because the pixel operations start from a frame-based linear transduction from light to voltage. This paper is useful for it’s references to recent work on gaze trackers, integrated eye trackers, and smart pixel sensors [8-13], including an earlier work by some of the same authors from the group of Gunhee Han at the Yonsei Univ. of Technology in Korea.

1.2 Silicon retina eye tracking

The new sensor that enables dynamic vision at greatly reduced computational cost and shorter latency is called the dynamic vision sensor (DVS) [1, 2, 14-16]. The DVS is an electronic camera, but it differs from conventional cameras by outputting pixel spike event addresses instead of frames of data. These spike events model part of what the human eye does in its motion-sensitive transient pathway. Each time a pixel outputs a spike, it means that the brightness (log intensity) at that pixel changed by at least a threshold amount. That means that most of the pixels at any instant output nothing and require no processing. Pixels that output spikes have seen a brightness change, and the chip reports this brightness change almost instantaneously as the pixel addresses. This device was developed outside COGAIN and COGAIN stimulated efforts to use it for eye tracking, and most recently, head tracking with active appearance based vision. The DVS is also being used for fast robotics [17-19], vehicle and person tracking [20-26], and several other application areas that are presently ongoing.

1.3 Characteristics of the silicon retina

Conventional image sensors see the world as a sequence of frames, each consisting of many pixels. In contrast, the DVS outputs digital address spike events in response of temporal contrast at the moments that pixels see changing intensity [3, 27] (Fig. 1 and Table 1). This neuromorphic abstraction of the transient pathway seen in biology turns out to be useful for a number of reasons. Each address-event that is output from the DVS is the digital address k of a pixel and means that the log intensity at pixel k changed by an amount T since the last event from that pixel. T is a global event threshold which we typically set to about 15% contrast. One bit of the address encodes the sign of the change (ON or OFF) and the rest of the bits encode the row and column addresses. This representation of “change in log intensity” generally encodes scene reflectance change. Because this computation is based on a compressive logarithmic transformation in each pixel, it also allows for wide dynamic range operation (120 dB, or 6 decades, compared with 60 dB for a high quality traditional image sensor). This wide dynamic range means that the sensor can be used with uncontrolled natural lighting, even when the scene illumination is very non-uniform. The asynchronous response property also means that the events have the timing precision of the pixel response rather than being quantized to the traditional frame rate. Thus the “effective frame rate” is typically several kHz and is set by the available illumination which determines the pixel bandwidth. The temporal redundancy reduction greatly reduces the output data rate for scenes in which most pixels are not changing. The design of the pixel also allows for unprecedented uniformity of response: the mismatch between pixel contrast thresholds is 2.1% contrast and the event threshold can be set to 10% contrast, allowing the device to sense real-world contrast signals rather than only artificial high contrast stimuli. The vision sensor also has integrated digitally-controlled biases that minimize chip-to-chip variation in parameters and temperature sensitivity. And finally, the vision sensor has a USB2.0 interface that delivers time-stamped address-events to a host PC with timestamp resolution of 1 microsecond.. This combination of features has meant that we have had the possibility of developing algorithms for using the sensor output and testing them easily in a range of real-world scenarios.

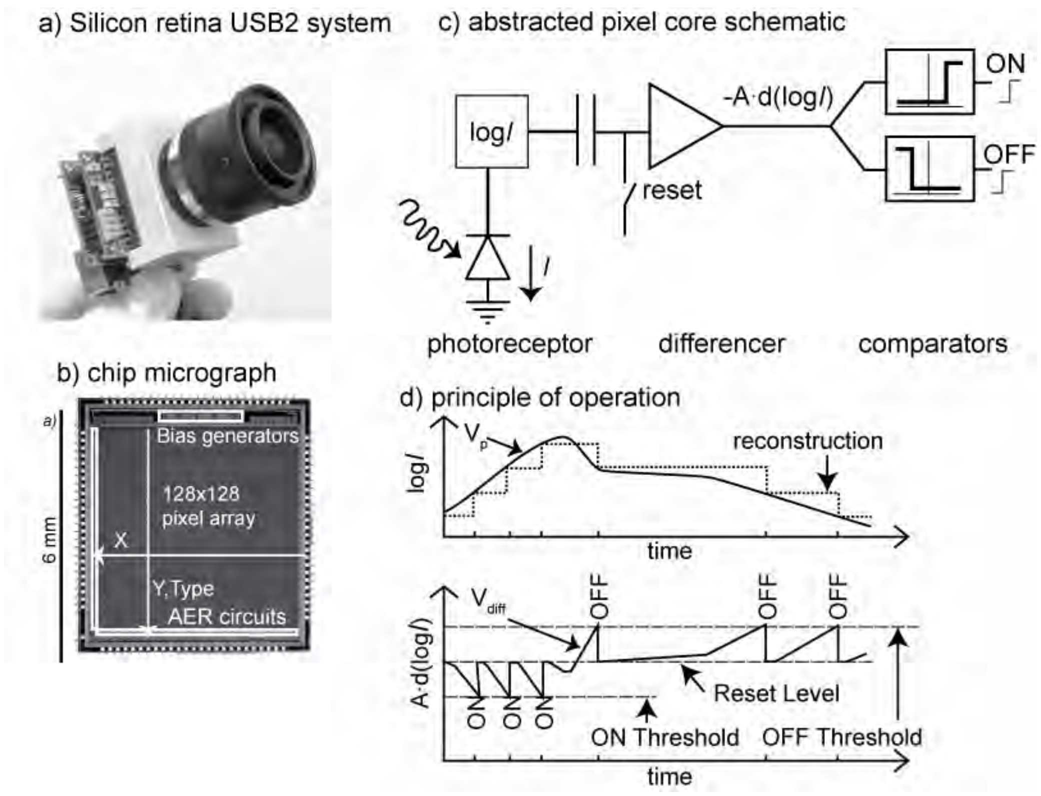


Fig. 1 Characteristics of the Tmpdiff128 dynamic vision sensor. **a)** the vision sensor with its lens and USB2.0 interface; **b)** a die photograph labeled with the row and column from a pixel which generates an event with x,y,type output (type=ON or OFF); **c)** an abstracted schematic of the pixel, which responds with events to fixed-size changes of log intensity; **d)** how the ON and OFF events are internally represented and output in response to an input signal. Figure adapted from [3].

Table 1 Tmpdiff128 dynamic vision sensor specifications

Functionality	Asynchronous temporal contrast
Pixel size um (lambda) Fill factor (%)	40x40 (200x200) 9.4% (photodiode area 151µm^2)
Fabrication process	4 metal 2 poly 0.35um
Pixel complexity	26 transistors (14 analog), 3 capacitors
Array size	128x128
Interface	15-bit word-parallel AER; USB 2.0 time-stamped address-event interface
Power consumption	Chip : 24mW @ 3.3V. USB system : 80 mA.
Dynamic range	120dB ; 2 lux to > 100 klux scene illumination with f/1.2 lens
Response latency	15µs @ 700mW/m^2
Max Events/sec	1M events/sec
Event threshold matching	2.1% scene contrast

1.4 Silicon retina eye tracker hardware

For COGAIN eye-tracker development, the hardware in Fig. 2 was assembled by Damian Gisler at UNIZH and sold to the Ersbøll group at TUD, paid by COGAIN funds. The eyeglasses hold the silicon retina over the user's right eye and the USB cable comes out over the ear. A single infrared LED provides illumination when necessary. A head strap holds the eyeglasses in place. The device is tolerable but not especially comfortable and is only suitable for experiments of <1 hour duration. Making the tracker more comfortable will require a smaller camera implementation. Although the retina is already very highly integrated for a prototype development, further miniaturization will probably not be achieved during the course of COGAIN.

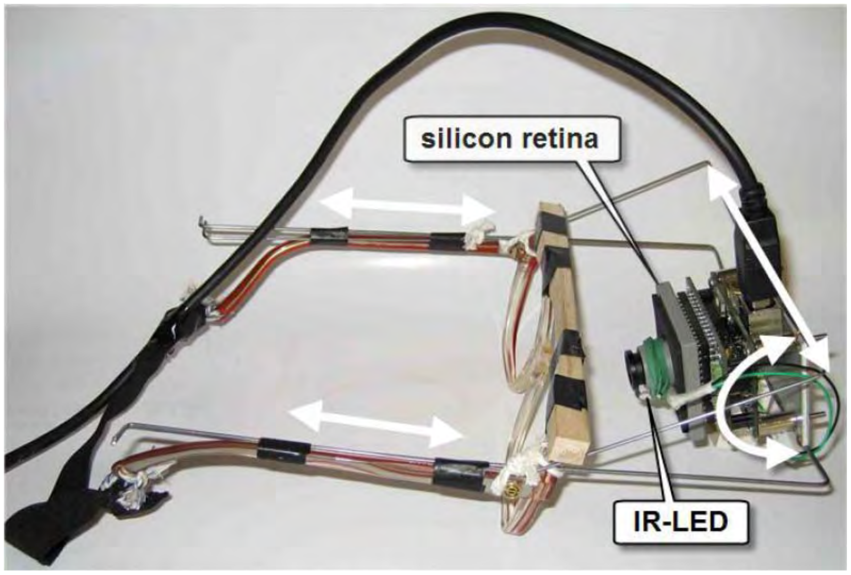


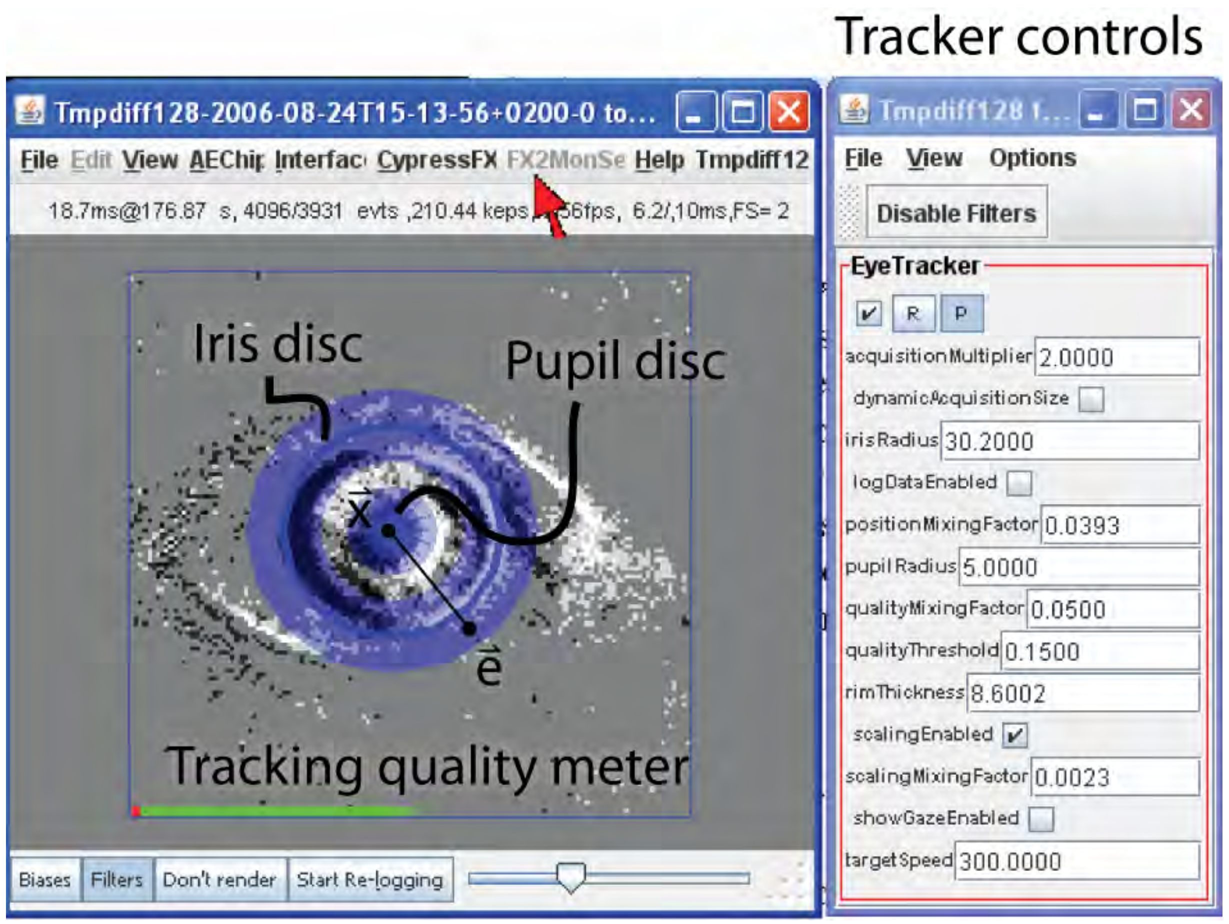
Fig. 2 Silicon retina eye tracker hardware. The white arrow show how the camera distance can be adjusted.

1.5 Event-driven tracking algorithms

Events from the DVS are processed inside jAER, a substantial open-source Java software infrastructure for real-time address-event representation (AER) processing [28] that has >300 classes and >30k non-comment lines of code (NCLOC) [28]. The methods for pupil tracking reported here are each about 2k NCLOC. The first eye tracking method reported here is a variant of a "cluster tracker" which tracks multiple moving objects by using the events to directly move the object models. The second eye tracking method reported here is an event-driven variant of a Hough transform feature detector.

1.5.1 EyeTracker: Model-based event driven pupil and iris tracking

This pupil/iris tracker class *EyeTracker* is based on the event-driven cluster tracker described briefly in [3, 29] and further enhanced in this project as the class *ch.unizh.ini.caviar.eventprocessing.tracking.EyeTracker* in the open-source jAER project [28]. The developer view of this tracker is shown in Fig. 3. This algorithm is inspired by the mean-shift approach used in frame-based vision [30, 31]. Each "cluster" models a moving object as a source of events. Here there is only one object, which is the location of the center of the pupil in the view of the vision sensor. Events that fall within the cluster move the cluster position, and a cluster is only considered supported ("visible") when it has received a threshold number of events. When the cluster loses support for a threshold period it is pruned and a new cluster is allocated when new data arrives. In this eye tracker, the lifetime of the cluster is set quite long so that the eye location is maintained even when the eye is not moving. The cluster position is updated by using a mixing factor that mixes the old value with the new measurement using fixed factors. Thus the time constant governing cluster position is inversely proportional to the evidence (event rate).


Tracker controls
Tmpdiff128-2006-08-24T15-13-56+0200-0 to...
File Edit View AEChip Interface CypressFX FX2MonSe Help Tmpdiff12
Iris disc
Pupil disc
$\vec{x}$
$\vec{e}$
Tracking quality meter
Biases Filters Don't render Start Re-logging
Tmpdiff128 f...
File View Options
Disable Filters
EyeTracker
R P

```
18.7ms@176.87  s, 4096/3931  evts ,210.44 keps, 56fps,  6.2/,10ms,FS= 2
```

acquisitionMultiplier	2.0000
dynamicAcquisitionSize	☐
irisRadius	30.2000
logDataEnabled	☐
positionMixingFactor	0.0393
pupilRadius	5.0000
qualityMixingFactor	0.0500
qualityThreshold	0.1500
rimThickness	8.6002
scalingEnabled	✓
scalingMixingFactor	0.0023
showGazeEnabled	☐
targetSpeed	300.0000

Fig. 3 EyeTracker class, showing a tracked eye with the iris and pupil disc regions, the tracking quality meter, and the tracker parameters.

The key advantages of the cluster tracker are:

- There is no frame correspondence problem because the events update the cluster location during the processing of each packet of events.
- Only pixels that generate events need to be processed. The cost of this processing is dominated by the cost of computing distance of each event to the cluster support regions.
- Memory cost is low because there is no frame memory, only cluster memory, and the cluster requires only a few hundred bytes of memory.

In the application described here the pupil and iris have a size which hardly varies, although it is initially unknown. The eye tracker dynamically computes the pupil and iris size during bootstrapping, which at present is either manually controlled or can be left free to run continuously (parameter *scalingEnabled*, Fig. 3).

More detailed steps for the eye tracker are outlined as follows. The algorithm runs on each packet of events, typically 128 or fewer events:

For each event, determine the radial distance to the center of the eye model. If the event is within the cluster, add the event to the cluster by pushing the cluster a bit towards the event and updating the last event time of the cluster. The size of the eye model is specified in a GUI interface as two radii, the radius of the pupil and the radius of the iris. An event is considered to fall within the model if its radial location falls within either of the two discs specifying the pupil or iris. The new cluster location $\vec{x}_{n+1}$ is given by mixing the old location $\vec{x}_n$ with the event location $\vec{e}$ using an adjustable mixing factor (parameter *mixingFactor*) $\alpha \approx 0.01$:

$$\vec{x}_{n+1} = (1-\alpha)\vec{x}_n + \alpha\vec{e} \qquad (0)$$

Events falling on the outer side of either ring pull the eye location toward the outer side, while events falling on the inner side pull the cluster toward the inner edge. In all cases the movement of the model is in the radial direction from the center of the model to or from the event location Fig. 3).

In addition, a tracking quality detector runs continuously and measures the fraction of events from the retina that fall inside the eye model, normalized by the area of the model's iris and pupil regions. If this fraction—after lowpass filtering—falls below a specified threshold, then the acquisition size can be dynamically increased to increase the probability of recapturing the location of the eye more quickly.

The eye model radius can be dynamically scaled. Initial pupil and iris radius are specified by the user, and then each event not only moves the model but also contributes to a rescaling of the model radius. The amount that the eye model radius is scaled is specified by the parameter *scalingMixingFactor*.

1.5.2 Results from the EyeTracker pupil/iris tracker

Fig. 4 shows tracking samples from the EyeTracker class tracking a moving eye during an experiment where the subject viewed a smoothly moving target moved by experimenter in front of the subject face. The target was moved left and right at different elevations. As can be seen from the data, the eye tracker output seems to mimic the movement of the eyes. However, this experiment has no ground truth data for comparison and would need to be repeated under more controlled conditions.

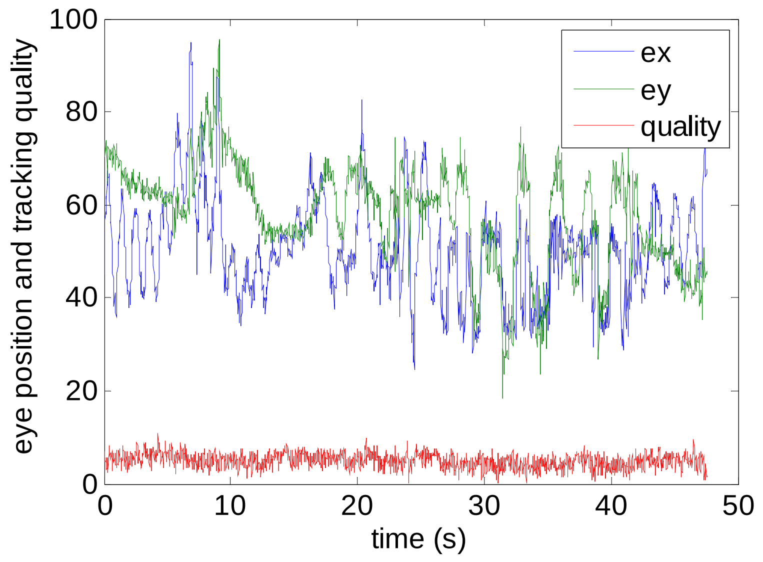


Fig. 4 Example eye tracking output data from EyeTracker class. Subject was tracking an object moved around in front of his face, left and right quickly at different elevations. Quality metric should measure tracking quality but does not do a very good job in this instance. Units for position are in 128x128 silicon retina pixels.

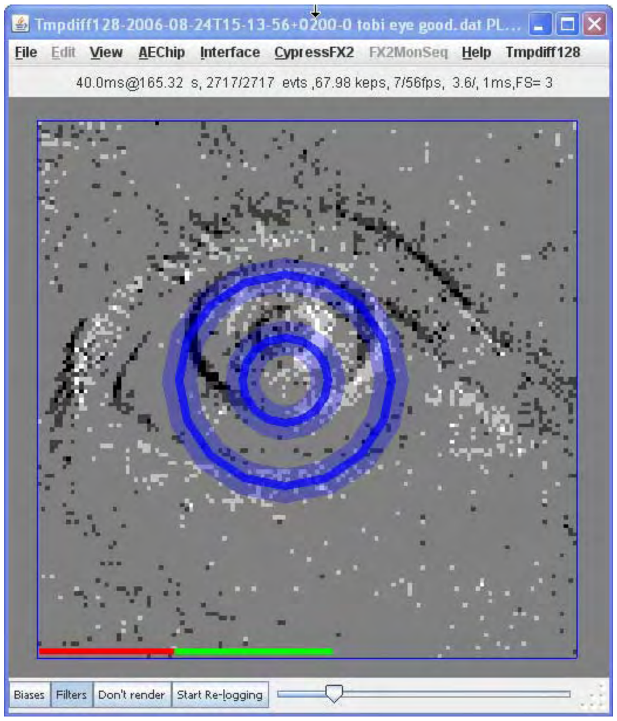
1.5.3 Advantages and drawbacks of the EyeTracker class

The EyeTracker has the undeniable advantage that is it very cheap to compute. Events can be processed on a year 2005 Pentium M laptop (2.13GHz processor) at a rate of about 1.4 million events per second (**Meps**). A moving eye produces an average of about 50 kilo events per second (**keps**). Therefore the tracker runs at about 28 times real time, or at a load of 1/28=4%, placing it in the range of capabilities of a low power embedded system.

The main disadvantage of the EyeTracker class is a tendency to become locked onto false positions (Fig. 5). Here the false position is maintained by events from the upper and lower eyelid. The size of the eye is wrong. These false positions and sizes usually occur after a blink. The underlying cause of these false locations is that EyeTracker has only a single, albeit fuzzy, hypothesis about eye position. Other potential eye positions, even if they are obvious to a human's glance, are not considered until the eye movements cause the model to fall into a more favorable position that locks the correct eye position.

An additional disadvantage of EyeTracker—shared by all the methods reported here—is the lack of a blink detector, making it difficult to maintain eye position during blinks and to recover tracking after a blink.

Another basic disadvantage of EyeTracker is the lack of computation of any reference position. It is likely that nearly all eye tracking algorithms have the capability of finding fiducial markers, such as glints or eye corners, that are essential for determining eye position relative to the head. These relative measurements are essential for long term tracking stability in open loop situations when the head mounted eye tracker can easily shift about. Still, an eye tracker that lacks a reference position measurement could still be useful in scenarios where online calibration is used to constantly update the eye tracker calibration.


Tmpdiff128-2006-08-24T15-13-56+0200-0 tobi eye good.dat PL...
File Edit View AEChip Interface CypressFX2 FX2MonSeq Help Tmpdiff128
Biases Filters Don't render Start Re-logging

40.0ms@165.32 s, 2717/2717 evts ,67.98 keps, 7/56fps, 3.6/, 1ms,FS= 3

Fig. 5 False tracking. EyeTracker has locked onto the wrong position and size for the pupil/iris of the eye because it the eyelids and the wrong pupil-iris edges support an incorrect localization.

1.6 HoughEyeTracker: Pupil tracking using an extended Hough transform algorithm

The drawback of EyeTracker led to a new study of silicon retina eye tracking based on a method borrowed from machine vision—the Hough transform. The *HoughEyeTracker* class tracks based on data from all events rather than just those in the disc regions used in EyeTracker. The HoughEyeTracker is described in detail in an appended report. The HoughEyeTracker also implements an elliptic model of the pupil appearance which more accurately models the pupil appearance, especially when the front of the eye is viewed obliquely.

1.6.1 HoughEyeTracker algorithm

The reader is referred to the appendix for details of the algorithm [32]. In the simplest case, a Hough-based detector accumulates evidence for a single hypothesis of the detected location (Fig. 6). Evidence is accumulated over a constant number of events (rather than for a constant time). Each new event contributes to the accumulator and each time there is a new event, the oldest event is erased from the accumulator. In addition, heuristic rules about eye movement limit the movement of the model location. The developer view of HoughEyeTracker is shown in Fig. 7.

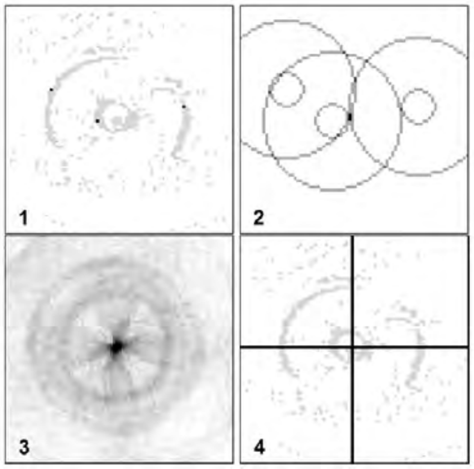
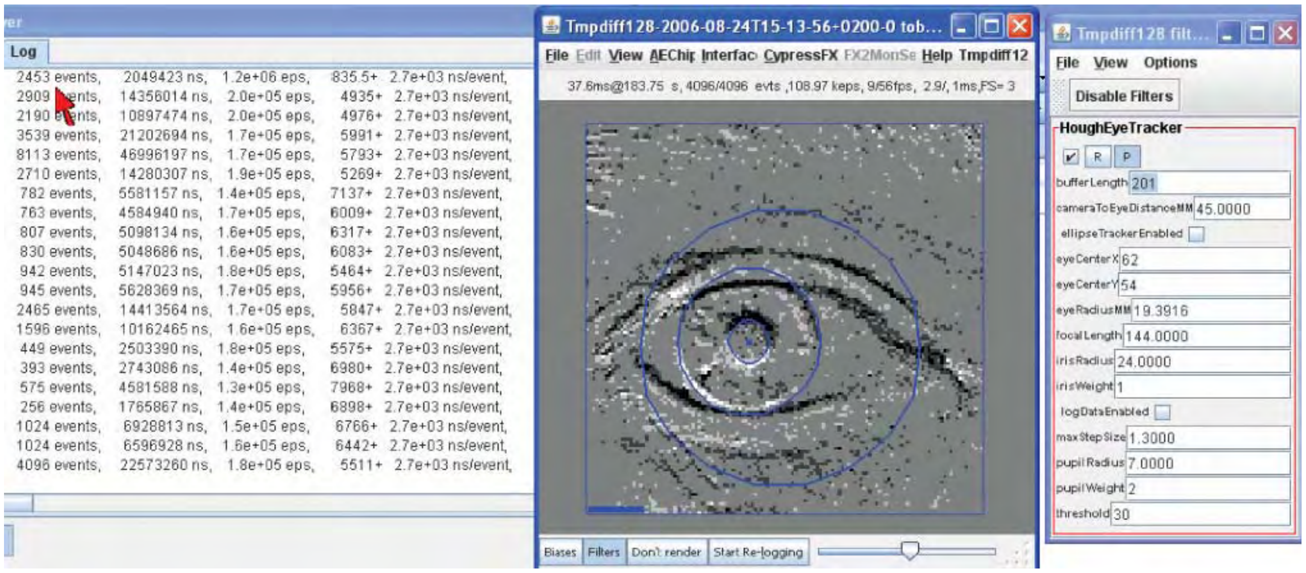


Fig. 6 Illustration of the basic concept of *HoughEyeTracker*. (1) Events from the moving eye captured by the silicon retina. (2) Accumulator array A with the patterns for the three black events of image (1). (3) Array A with accumulation of the patterns for all events of image (1) represented in grayscale. (4) Eye center defined by the maximum of array A (adapted from [32]).


Log
Tmpdiff128-2006-08-24T15-13-56+0200-0 tob...
File Edit View AEChip Interfac CypressFX FX2MonSe Help Tmpdiff12
Biases Filters Don't render Start Re-logging
Tmpdiff128 filt...
File View Options
Disable Filters
HoughEyeTracker
R P

```
2453 events,   2049423 ns,  1.2e+06 eps,   835.5+  2.7e+03 ns/event,
2909 events,  14356014 ns,  2.0e+05 eps,    4935+  2.7e+03 ns/event,
2190 events,  10897474 ns,  2.0e+05 eps,    4976+  2.7e+03 ns/event,
3539 events,  21202694 ns,  1.7e+05 eps,    5991+  2.7e+03 ns/event,
8113 events,  46996197 ns,  1.7e+05 eps,    5793+  2.7e+03 ns/event,
2710 events,  14280307 ns,  1.9e+05 eps,    5269+  2.7e+03 ns/event,
 782 events,   5581157 ns,  1.4e+05 eps,   7137+  2.7e+03 ns/event,
 763 events,   4584940 ns,  1.7e+05 eps,   6009+  2.7e+03 ns/event,
 807 events,   5098134 ns,  1.6e+05 eps,   6317+  2.7e+03 ns/event,
 830 events,   5048686 ns,  1.6e+05 eps,   6083+  2.7e+03 ns/event,
 942 events,   5147023 ns,  1.8e+05 eps,   5464+  2.7e+03 ns/event,
 945 events,   5628369 ns,  1.7e+05 eps,   5956+  2.7e+03 ns/event,
2465 events,  14413564 ns,  1.7e+05 eps,    5847+  2.7e+03 ns/event,
1596 events,  10162465 ns,  1.6e+05 eps,    6367+  2.7e+03 ns/event,
 449 events,   2503390 ns,  1.8e+05 eps,   5575+  2.7e+03 ns/event,
 393 events,   2743086 ns,  1.4e+05 eps,   6980+  2.7e+03 ns/event,
 575 events,   4581588 ns,  1.3e+05 eps,   7968+  2.7e+03 ns/event,
 256 events,   1765867 ns,  1.4e+05 eps,   6898+  2.7e+03 ns/event,
1024 events,   6928813 ns,  1.5e+05 eps,    6766+  2.7e+03 ns/event,
1024 events,   6596928 ns,  1.6e+05 eps,    6442+  2.7e+03 ns/event,
4096 events,  22573260 ns,  1.8e+05 eps,    5511+  2.7e+03 ns/event,
```

```
37.6ms@183.75 s, 4096/4096 evts ,108.97 keps, 9/56fps, 2.9/, 1ms,FS= 3
```

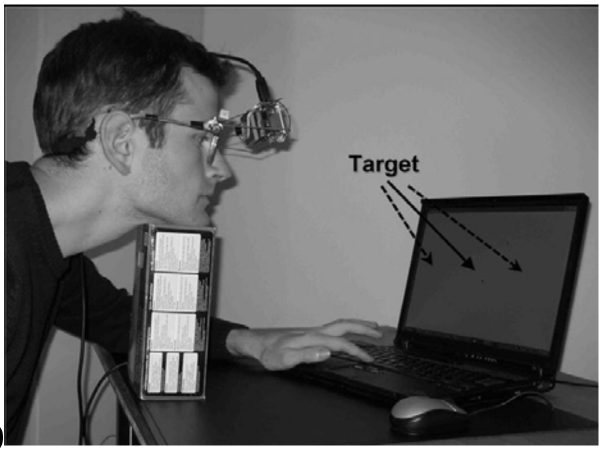
✓	
bufferLength	201
cameraToEyeDistanceMM	45.0000
ellipseTrackerEnabled	☐
eyeCenterX	62
eyeCenterY	54
eyeRadiusMM	19.3916
focalLength	144.0000
irisRadius	24.0000
irisWeight	1
logDataEnabled	☐
maxStepSize	1.3000
pupilRadius	7.0000
pupilWeight	2
threshold	30

Fig. 7 HoughEyeTracker developer view. Left panel shows performance measures. Middle panel shows tracked eye. Right panel shows tracker parameters.

1.6.2 HoughEyeTracker results

The results of an experiment using HoughEyeTracker are shown in Fig. 8. The subject viewed a screen with 3 targets (Fig. 8a). After each target fixation, a key click captured the tracked location. These samples are shown in Fig. 8b. Clearly there is considerable room for improvement, but the results are promising.

a)



b)

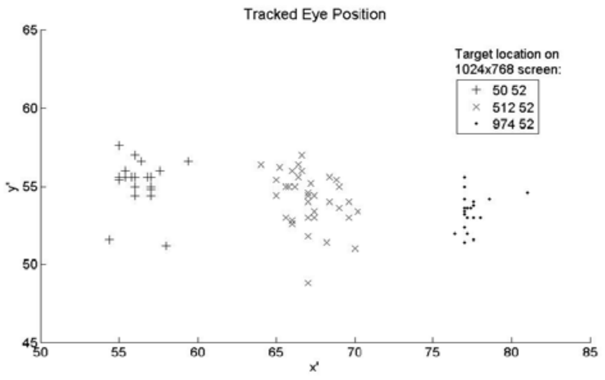


Fig. 8 Setup and result of HoughEyeTracker experiment with three targets; samples are collected on user key press after fixating each target. Units for position are in 128x128 silicon retina pixels.

1.6.3 HoughEyeTracker advantage and disadvantages

At present HoughEyeTracker requires a specified region of interest and a specified eye size. These parameters will need to be estimated from online data for a realistic tracker. HoughEyeTracker is also more expensive computationally than EyeTracker: events are processed at a rate of about 150 keps, about 10 times slower than EyeTracker and at about 3 times real time rate.

1.7 Summary of silicon retina eye tracking activity

Fig. 9 compares the EyeTracker and HoughEyeTracker tracking measurements for a retina eye movement recording with about 60 seconds of data. The measurements are largely in agreement, but there are places where they disagree. These correspond to the mistracking by EyeTracker after a blink. This comparison points out a course of further development. By using a mixture of experts, tracking can be greatly improved. A disagreement between two trackers, running in parallel, indicates a problem with tracking.

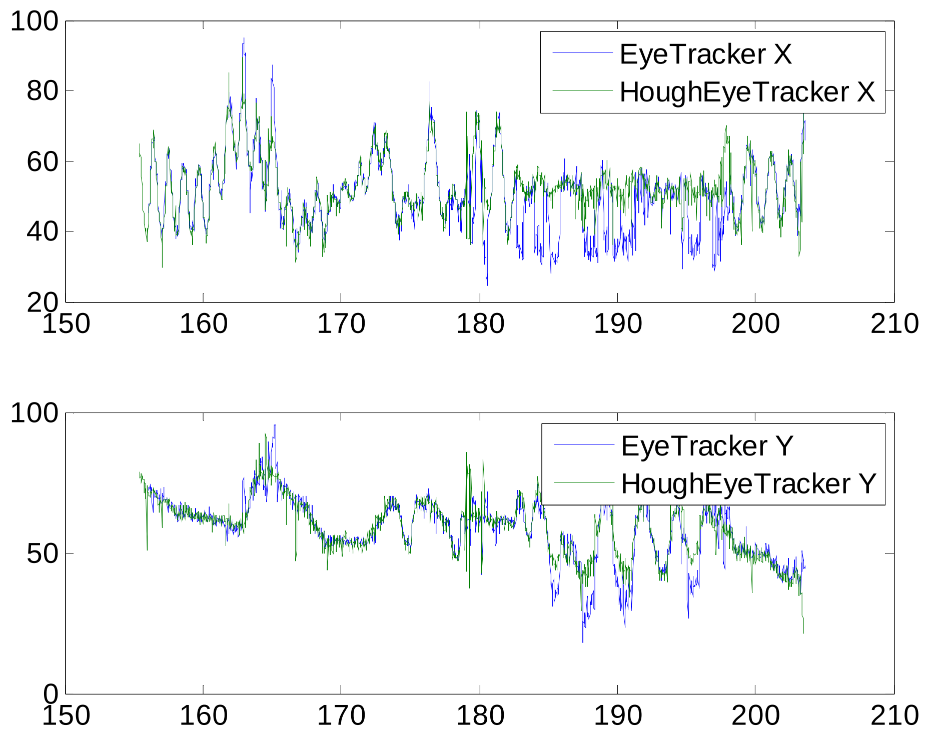


Fig. 9 Comparison of tracking output for EyeTracker and HoughEyeTracker. Units for position are in 128x128 silicon retina pixels.

1.7.1 Outlook for silicon retina eye tracking

Clearly more development is warranted and awaits semester or diploma students who show interest in this project. It would be much easier to find such students if COGAIN funds could support expenses of hosting visiting master or summer intern students.

The main opportunities include the following:

- A combined EyeTracker plus HoughEyeTracker algorithm would allow EyeTracker to bootstrap HoughEyeTracker eye location and size.
- An application scenario is essential for allowing extended testing. One possibility is to capture natural data in the form of eye locations measured during typing and on mouse click activity during normal use of a PC. This data could form a rich set of training data for online calibration. At the same time a hot key could activate eye control of the mouse pointer. Such application would entail hooking into the operating system keyboard and mouse interface at a low level to enable key and mouse click capture and mouse control, no matter which application has the foreground.
- Head tracking will be essential for full gaze tracking. One possibility is to mount active LED sources on the head-mounted tracker and use these to locate the head using a second silicon retina mounted on the computer monitor and looking towards the user. A study in this direction is presently ongoing; see Sec. 1.8.

1.8 Head tracking using active appearance modeling

A new investigation that has just been started recently in collaboration between UNIZH and TUD will use the silicon retina for head tracking. The goal of this work is to provide head gaze direction at low computational load on a host PC, with the target application being control of avatars and to provide necessary head direction input for full gaze tracker. Results on this work will be reported later.

2 Owl eye tracker

The other device that was significantly evolved during COGAIN is the Owl [5, 33], originally developed by Martin King in the 1980's and updated with a new hardware interface around 2005 by Dale Grover at the University of Michigan (Fig. 10). Partner UNIZH contributed a Matlab toolbox for interfacing to the Owl hardware. A paper from the COGAIN 2006 meeting summarizing progress on the Owl is appended in Appendix A [6].

The Owl is a non-imaging sensor. It consists of a ring of LED emitters which are stimulated in sequence and photodiode sensors which are measured for each LED stimulation. The signals measured on the photodiodes are generally monotonic or smooth functions of eye position. By a relatively simple principle component analysis, a coarse measure of eye gaze direction can be extracted from the measured photodiode signals at low computational cost, approximately equivalent to inverting a small matrix of measurements. Additional LEDs mounted on the ring are used as user feedback signals. In fact, the feedback signals are what makes the Owl particularly promising as a gaze tracker for multiple-choice selection. The simplicity of the Owl's construction and the low dimensionality of its signals means that it could be built as a wireless device on the users eyeglasses, although this has not yet been realized.

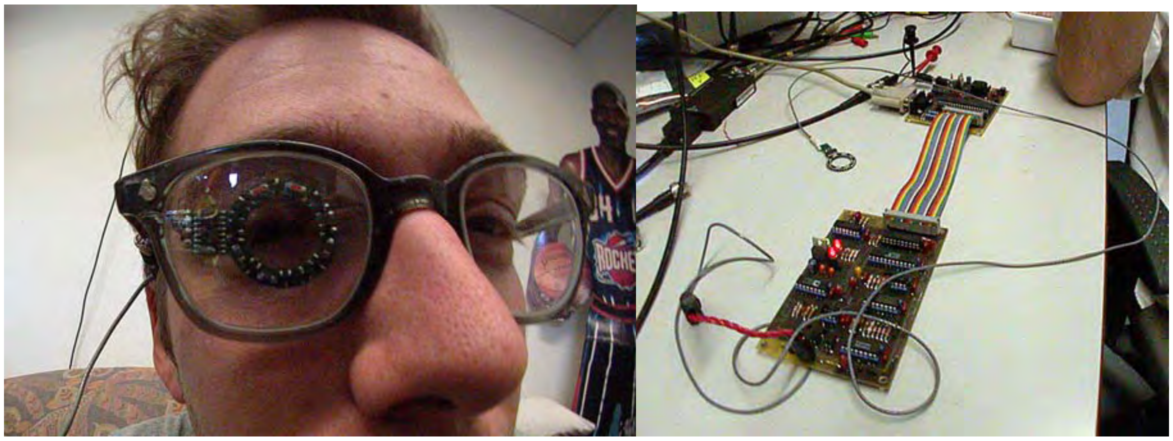


Fig. 10 Owl sensor mounted on eyeglasses, and showing the analog and digital interface boards connected by serial port to a host PC.

The disadvantages of the Owl are that it is spectacle mounted directly on the eyepiece and partially occludes the user's peripheral vision in that eye. As Grover points out in his online FAQ [33], the Owl is not suited for general gaze tracking, but would be better targeted as a selection-driven tracker. The Owl has been demonstrated in unpublished work to be usable as an input device for the gaze writing system Dasher.

An instance of the Owl hardware in Fig. 10 presently resides at the Max Planck Institute for Biological Cybernetics in Tuebingen with the Empirical Inference group of Bernhard Schoelkof. In 2006-2007 PhD student Wolf Keinzle supervised three intern students (Aditya Mandayam–hardware, David Delgado and

Thomas Schreiner–C# interface) who developed a new interface to the Owl, based on new firmware sent by Dale Grover. The technical work of developing a usable hardware–software interface is now accomplished and this device is now ready for use in experiments.

An earlier matlab and octave toolbox for interfacing to the Owl hardware has been uploaded to the COGAIN "internal pages > wp areas > wp5 > shared files" area.

3 References

[1] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE J. Solid State Circuits*, vol. 43, pp. scheduled, 2008.

[2] T. Delbruck, "Freeing vision from frames " in *The Neuromorphic Engineer*, vol. 3, 2006.

[3] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change," Visuals Supplement to ISSCC Dig. of Tech. Papers, San Francisco, pp. 508-509 (27.9), 2006.

[4] Available: http://siliconretina.ini.uzh.ch

[5] D. Grover, T. Delbruck, and M. King, "An eye tracking system using multiple near-infrared channels with special application to efficient eye-based communication," Thirteenth European Conference on Eye Movements ECEM13, Bern, pp. 36-39, 2005.

[6] D. Grover, "Progress on an Eye Tracking System Using Multiple Near-Infrared Emitter/Detector Pairs with Special Application to Efficient Eye-Gaze Communication," The 2nd Conference on Communication by Gaze Interaction - COGAIN 2006: Gazing into the future, Turin, pp. 21-25, 2006.

[7] D. Kim, J. Cho, S. Lim, D. Lee, and G. Han, "A 5000S/s single-chip smart eye-tracking sensor," ISSCC Digest of Technical Papers, San Francisco, pp. Paper 2.2, 2008.

[8] D. Kim, S. Lim, and G. Han, "Single-chip eye tracker using smart CMOS image sensor pixels," *Analog Int. Circuits and Sig. Proc.*, pp. 131-141, 2005.

[9] Y. Muramatsu, S. Kurosawa, M. Furumiya, H. Ohkubo, and Y. Nakashiba, "A signal-processing CMOS image sensor using a simple analog operation," *IEEE J. Solid State Circuits*, pp. 101-106, 2003.

[10] A. Graupner, J. Schreiter, S. Getzlaff, and e. al., "CMOS image sensor with mixed-signal processor array," *IEEE J. Solid State Circuits*, pp. 948-957, 2003.

[11] T. Miyoshi and A. Murata, "Input device using eye tracker in human-computer interactions," Proc. 10th IEEE Int. Workshop Robot and Human Interactive Communication, pp. 580-585, 2001.

[12] K. Park, "A real-time gaze position estimation method based on a 3-D eye model," *IEEE T. Systems, Man and Cybernetics, Part B*, pp. 199-212, 2007.

[13] Z. Zhiwei, J. Qiang, K. Fujimira, and e. al., "Combining Kalman filtering and mean shift for real time eye tracking under active illumination," Proc. 16th Int. Conf. Pattern Recognition, pp. 318-321, 2002.

[14] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change," ISSCC Dig. of Tech. Papers, San Francisco, pp. 508-509 (27.9), 2006.

[15] P. Lichtsteiner, "An AER Temporal Contrast Vision Sensor," in *Inst. of Neuroinformatics*, vol. PhD. Zurich: UNI-ETH Zurich, 2006, pp. 143.

[16] Available: http://siliconretina.ini.uzh.ch

[17] A. Linares-Barranco, F. Gómez-Rodríguez, A. Jiménez, T. Delbruck, and P. Lichtsteiner, "Using FPGA for visuo-motor control with a silicon retina and a humanoid robot," ISCAS 2007, pp. 1192-1195, 2007.

[18] T. Delbruck and P. Lichtsteiner, "Fast sensory motor control based on event-based hybrid neuromorphic-procedural system," ISCAS 2007, New Orleans, pp. 845-848, 2007.

[19] Available: http://siliconretina.ini.uzh.ch/wiki/doku.php?id=robogoalie

[20] M. Litzenberger, C. Posch, D. Bauer, A. N. Belbachir, P. Schon, B. Kohn, and H. Garn, "Embedded Vision System for Real-Time Object Tracking using an Asynchronous Transient Vision Sensor," Digital Signal Processing Workshop, 12th-Signal Processing Education Workshop, 4th, pp. 173-178, 2006.

[21] A. N. Belbachir, K. Reisinger, G. Gritsch, P. Schon, and H. Garn, "Automated Vehicle Velocity Estimation Using a Dual-Line Asynchronous Sensor," Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE, pp. 552-557, 2007.

[22] D. Bauer, A. N. Belbachir, N. Donath, G. Gritsch, B. Kohn, M. Litzenberger, C. Posch, P. Schön, and S. Schraml, "Embedded Vehicle Speed Estimation System Using an Asynchronous Temporal Contrast Vision Sensor," *EURASIP Journal on Embedded Systems*, vol. 2007, 2007.

[23] A. N. Belbachir, M. Hofstatter, K. Reisinger, N. Donath, and P. Schon, "Object Velocity Estimation Based on Asynchronous Data from a Dual-Line Sensor System," Industrial Informatics, 2007 5th IEEE International Conference on, pp. 347-352, 2007.
[24] M. Litzenberger, B. Kohn, G. Gritsch, N. Donath, C. Posch, N. A. Belbachir, and H. Garn, "Vehicle Counting with an Embedded Traffic Data System using an Optical Transient Sensor," Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE, pp. 36-40, 2007.
[25] M. Litzenberger, A. N. Belbachir, P. Schon, and C. Posch, "Embedded Smart Camera for High Speed Vision," Distributed Smart Cameras, 2007. ICDSC'07. First ACM/IEEE International Conference on, pp. 81-86, 2007.
[26] A. N. Belbachir, M. Litzenberger, C. Posch, and P. Schon, "Real-Time Vision Using a Smart Sensor System," Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on, pp. 1968-1973, 2007.
[27] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE J. Solid State Circuits*, vol. accepted, 2007.
[28] Available: http://jaer.wiki.sourceforge.net
[29] M. Litzenberger, C. Posch, D. Bauer, P. Schön, B. Kohn, H. Garn, and A. Belbachir, "Embedded Vision System for Real-Time Object Tracking using an Asynchronous Transient Vision Sensor," IEEE Digital Signal Processing Workshop 2006, Grand Teton, Wyoming, pp. 173-178, 2006.
[30] D. Comaniciu and V. Ramesh, "Mean shift and optimal prediction for efficient object tracking," IEEE Intl. Conf on Image Processing (ICIP), Vancouver, pp. 70-73, 2000.
[31] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE Trans. Pattern Analysis Machine Intelligence*, vol. 17, pp. 790-799, 1995.
[32] D. Gisler, "Eye tracking using event-based silicon retina (Semester project report)," ETH Zurich, Zurich 2007.
[33] Available: http://www.redcedar.com/owlQandA.html

Appendix A: External reports on silicon retina and Owl work.

The following two reports are appended here:

1. Semester project report 2007, Damian Gisler, Eye tracking using event-based silicon retina, Inst. of Neuroinformatics, UNI-ETH Zurich.
2. D. Grover, "Progress on an Eye Tracking System Using Multiple Near-Infrared Emitter/Detector Pairs with Special Application to Efficient Eye-Gaze Communication," The 2nd Conference on Communication by Gaze Interaction - COGAIN 2006: Gazing into the future, Turin, pp. 21-25, 2006.

Institute of Neuroinformatics
UNI-ETH Zürich
WS 2006/2007

Semesterarbeit

Eye Tracking using Event-Based Silicon Retina

Author
Gisler Damian
(dagisler@student.ethz.ch)

Supervisor:
Tobi Delbruck
Patrick Lichtsteiner

Abstract

The goal of this *Semesterarbeit* is to write an algorithm for eye tracking by using a silicon retina developed by the INI. Unlike conventional video cameras, the pixels of the silicon retina asynchronously respond to events that represent relative changes in intensity.

This report describes the implementation of a tracking algorithm which provides online eye tracking in real time. The tracking algorithm is based on a kind of *extended Hough transformation*, an algorithm which is able to find the location of a given pattern in pixel space. In our case, the pattern is given by the mapped shape of the pupil and the iris to the silicon retina.

The tests of the tracking algorithm show that the tracker is able to estimate the gaze direction by using a conventional 1.6 GHz Laptop PC in real time. By applying the tracker while following a target displayed on a screen, the tracker estimates the gaze direction with a good repeatability and with a resolution which is better than a third of the screen size.

Contents

Chapter 1

Introduction

1.1 Eye Tracking

Eye tracking is a general term for using techniques for measuring either the point of gaze (where we are looking) or the motion of the eye itself. Eye tracking can be used for behavior research or for human-computer interaction, for example for disabled persons. Today, several different eye tracking systems are available. One possible concept is shown in Fig. 1.1.

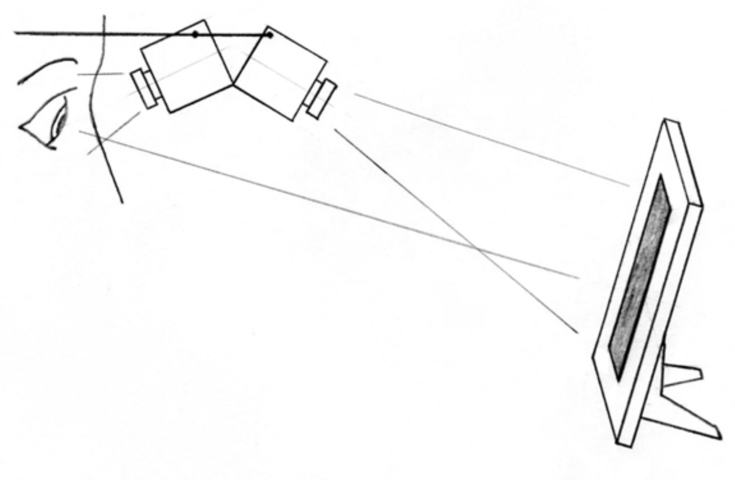


Figure 1.1: *Concept of a possible eye tracking system for man computer interaction with head mounted vision sensors. The system consists of an eye monitoring sensor which estimates the gaze relative to the head position and a sensor which monitors the head position relative to the screen.*

1.2 Silicon Retina

The most important part of the hole setup is the silicon retina. Unlike conventional video cameras, the pixels of the silicon retina asynchronously respond to events that represent relative changes of intensity [1]. Through a USB2 interface the events are sent to a Java framework where they can be processed [2].

Based on these attributes the retina is qualified for motion detection and also for object tracking. Compared to conventional motion detection we avoid reading pixels without changes and we also avoid the task of differential imaging which is often used for motion detection in image processing. This means, that the processing system needs less performance what enables us to do fast image processing by a conventional computer in realtime.

1.3 Semesterarbeit Description

The idea is to use the silicon retina for real-time eye tracking, which estimates the actual gaze direction with high accuracy and low latency. This Semesterarbeit is focused on developing a tracking algorithm for the eye monitoring, head fixed silicon retina (Fig.1.1) and could be separated in three main parts:

- As Fig.1.1 indicates, the eye monitoring silicon retina has to be fixed at the head.
- Based on an extensive existing software infrastructure the tracking algorithm has to be developed and implemented.
- Finally, the algorithm has to be tested to give an evaluation of its performance.

Chapter 2

Hardware Setup

2.1 Head Mounted Silicon Retina

According to the concept shown in figure 1.1 the relative position of the eye monitoring sensor should be independent of head motions. Therefore the silicon retina is fixed on a glasses frame. To build a stiff, light and adjustable support, it is made of steel wire. The wooden stick fixed on the glasses frame is just to get the frame more stable. To improve the contrast, a IR-LED is fixed at the lens. Because of the USB connection, the retina chip is mounted upside down.

It is possible to shift the retina to the right and left (also to change the monitored eye). To change the angel of the retina, or the distance to the eye it is possible to shift the steel wire fixed at the earpiece according to figure 2.1. By changing the distance to the eye, we have to consider that the eye is still in focus by adjusting the lens of the silicon retina.

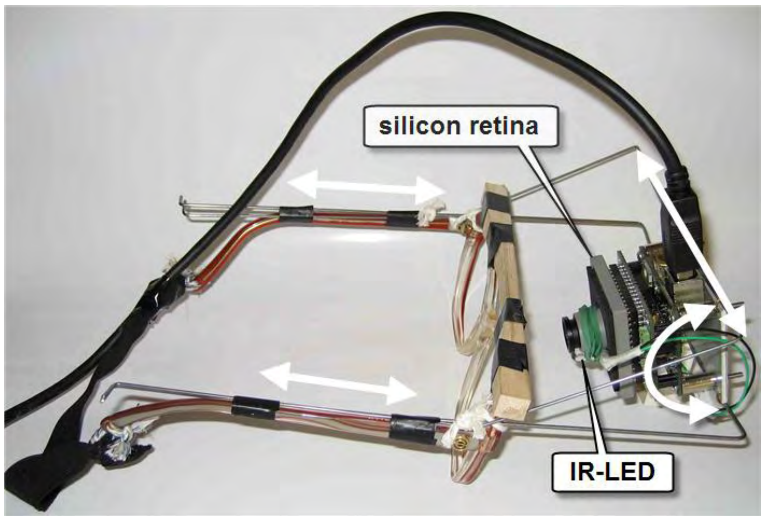


Figure 2.1: *The silicon retina fixed on a glasses frame. The white arrows display the possible adjustments.*

Chapter 3

Tracking Algorithm

The tracking algorithm is based on extended Hough Transform. The classical transform identifies just lines in an image, but it has been extended to identifying positions of arbitrary patterns. In our case, the pattern is given by the mapped shape of the pupil and the iris to the silicon retina.

3.1 Basic Idea

Let's assume we are trying to find the location an arbitrary given pattern P in a continuous two dimensional space while the shape of the pattern is just depending on a pattern-reference point (x_r, y_r) as shown in Fig.(3.1). We also assume, it is possible to describe the elements of the Pattern P as an implicit function $f(x, y, x_r, y_r) = 0$. So we can describe the pattern as:

$$P_{(x_r,y_r)} := \{(x, y)|f(x, y, x_r, y_r) = 0,\ \ x, y, x_r, y_r \in \mathbb{R}\}. \tag{3.1}$$

So, if the reference point (x_r, y_r) is given, we are able to calculate the pattern with (3.1). But in our case, the pattern itself - or at least a part of it - is given and we would like to find the reference point. So we have to calculate the *inverse* of P as a function of

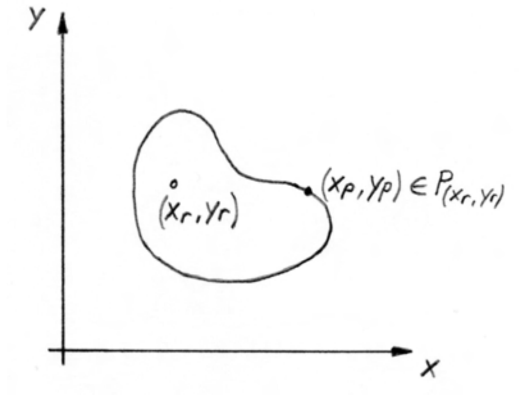


Figure 3.1: *Pattern P with pattern-reference point* (x_r, y_r)

the given pattern elements $(x_p, y_p)_k \in P$ where k indicates the different given pattern elements:

$$P_k^{-1} := \{(x_r, y_r) | f((x_p, y_p)_k, x_r, y_r) = 0\} \tag{3.2}$$

If we can calculate P^{-1} exactly, it is possible to show that the pattern-reference point is part of the intersection of all P_k^{-1}.

$$(x_r, y_r) \in \bigcap_{k=1}^{n} P_k^{-1}. \tag{3.3}$$

In case of a *static pattern*, which means that the shape of the pattern is not depending on the location of the pattern ($\Rightarrow f(x, y, x_r, y_r) = f(x - x_r, y - y_r) = 0$) the inverse pattern is simply the point reflection of the pattern at the pattern-reference point. For a pattern that changes its shape, the inverse pattern is more complicated to describe.

If we try to apply the idea from above to the tracking problem, we have to consider the following differences:

- the pattern is located in a discrete, not in a continuous space
- the mapping of the pattern isn't correct and disturbed by noise
- we do not know the pattern exactly

So it makes no sense to calculate an exact inverse pattern P^{-1} and we have to approximate a practical pattern and map it into the discrete space for each event. We will do this approximation by minimizing f.

$$\widehat{P}_k^{-1} := \{(x_r, y_r) | f(x_k, y_k, x_r, y_r) = min,\ x_k, y_k, x_r, y_r \in \{0, 1, 2..127\}\} \tag{3.4}$$

Now we can write $\widehat{P}_k^{-1}$ into an *accumulator array* $\boldsymbol{A}$. This means we add a determined value w to $\boldsymbol{A}[x_r, y_r]$ for all $(x_r, y_r) \in \widehat{P}_k^{-1}$, where w is the *weight* for the actual pattern part. It would also not be realistic to find a point of intersection of all $\widehat{P}_k^{-1}$. But in the region of the reference point we can expect a higher density of the mapped $\{(x_r, y_r)\}_k$ so we can assume the pattern-reference point as the maximum of the accumulator array. The basic concept of this Idea is illustrated in Fig.3.2.

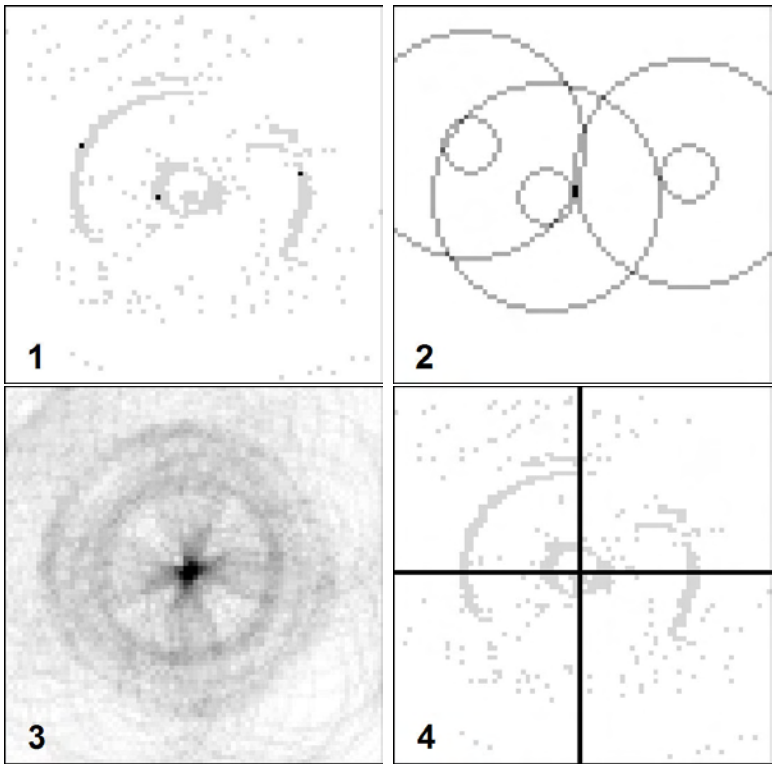


Figure 3.2: *Illustration of the basic concept. (1) Image of the moving eye captured by the silicon retina. (2) Accumulator array* ***A*** *with the patterns for the three black events of image (1). (3) Array* ***A*** *with accumulation of the patterns for all events of image (1) represented in grayscale. (4) Eye center defined by the maximum of array* ***A***

3.2 Inverse Pattern

To apply the general approach described at section 3.1 we first have to find a pattern P that corresponds to the image of the eye captured by the silicon retina. As pattern-reference point, it is obvious to define the center point of the iris respectively of the pupil.

3.2.1 Circular Pattern

In the simplest case, we can assume the inverse pattern as two concentric circles located at the event position with the radius of the pupil R_p and the iris R_i. With this assumption the tracker performance is good, if the angle θ is small ($\Rightarrow \cos\theta \approx 1$) (Fig.3.3).

3.2.2 Elliptic Pattern

If θ becomes bigger, which means the gaze is pointing for example to the side, the projection of the iris and pupill to the chip plane becomes elliptic. If we would like to respect this effect, we have to deal with a pattern which is now depending on the event location. To deviate the pattern we assume the following eye respectively camera model shown in figure 3.3. For this case we assume a simple pinhole camera model which maps a point of the three dimensional space (x, y, z) to the two dimensional space of the silicon retina (x', y') according to

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{f}{z} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.5}$$

where f is the focal length in number of pixels. If the moving eye is generating an event at the silicon retina (e'_x, e'_y), we can calculate the source point in space by calculating the intersection of the event beam

$$\overrightarrow{e}(z) = \begin{pmatrix} e'_x/f \\ e'_y/f \\ 1 \end{pmatrix} z \tag{3.6}$$

and the eye ball sphere, which we describe in spherical coordinates as a function of θ and φ with the radius R_{eye}

$$\overrightarrow{S}_{eye}(\theta, \varphi) = \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \end{pmatrix} + \begin{pmatrix} \cos\varphi \sin\theta \\ \sin\varphi \sin\theta \\ \cos\theta \end{pmatrix} R_{eye}. \tag{3.7}$$

Now we can solve the equation $\overrightarrow{e} = \overrightarrow{S}_{eye}$ to θ and φ. To get a simpler solution we set $x_{eye} = 0$ and $y_{eye} = 0$. This can be achieved by an appropriate adjustment of the

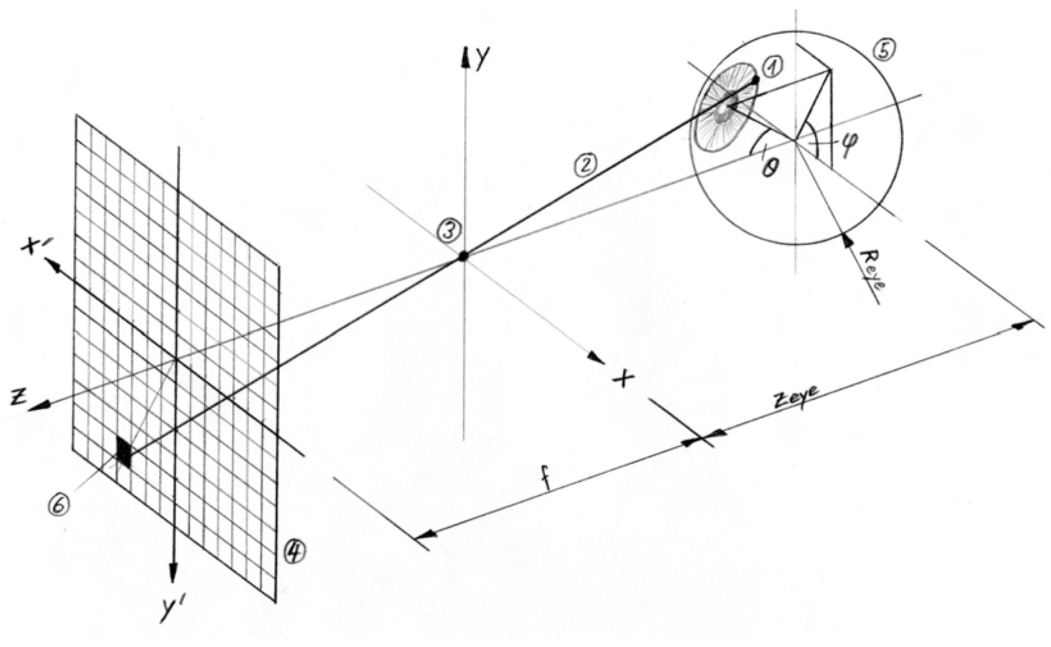


Figure 3.3: *Model of the eye/pinhole camera system. (1) event source; (2) event beam; (3) pinhole; (4) silicon retina; (5) eye ball; (6) event in pixel space* (e'_x, e'_y)

camera position (section A.2). In this case we get the following solution for the event source

$$\varphi = \arctan\left(\frac{e'_x}{e'_y}\right) \tag{3.8}$$

$$\theta = \arccos\left(\frac{-\left(e'^2_x + e'^2_y\right) z_{eye} \pm f\sqrt{\left(e'^2_x + e'^2_y + f^2\right) R^2_{eye} - \left(e'^2_x + e'^2_y\right) z^2_{eye}}}{\left(e'^2_x + e'^2_y + f^2\right) R_{eye}}\right). \tag{3.9}$$

Because we are not interested in solutions behind the eye center, we just consider the solution with the positive squareroot of (3.9).

Now we are looking for all possible eye positions, which can produce such an event. For the simple case with $\theta = 0$, the possible eye positions are located on two circles (one for the iris and one for the pupil) with center point at the event source and the circle plane perpendicular to the z-axis (Fig.3.4). The radius and the distance to the eye center of this circles for the iris and for the pupil are given by

$$r_{iris} = \frac{R_{iris}}{R_{eye}}\sqrt{R^2_{eye} - R^2_{iris}} \tag{3.10}$$

$$d_{iris} = \frac{R^2_{eye} - R^2_{iris}}{R_{eye}} \tag{3.11}$$

$$r_{pupil} = R_{pupil}\sqrt{\frac{R^2_{eye} - R^2_{iris}}{R^2_{eye} - R^2_{iris} + R^2_{pupil}}} \tag{3.12}$$

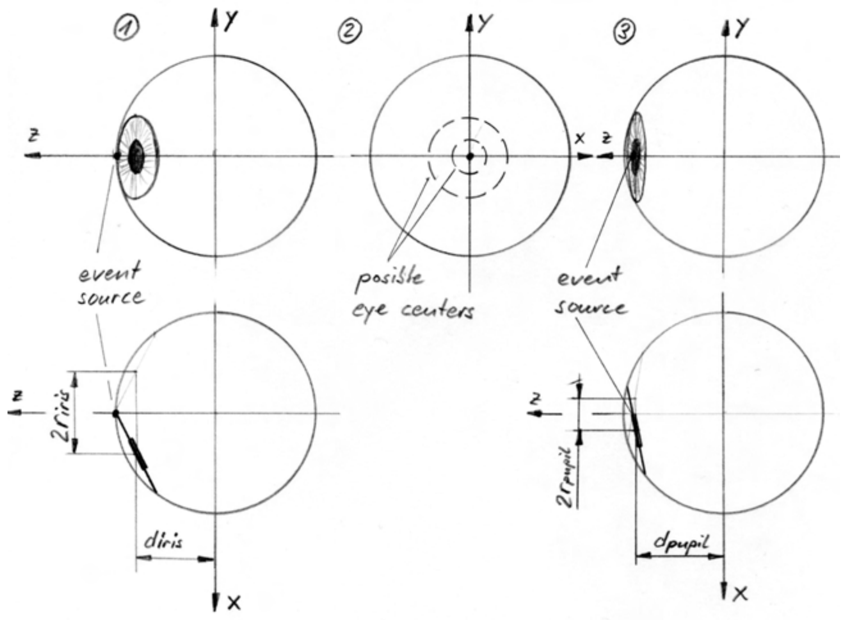


Figure 3.4: *Event source at $\theta = 0$. (1) iris event; (2) possible eye-center locations to iris or pupil event; (3) pupil event*

$$d_{pupil} = \frac{R_{eye}^2 - R_{iris}^2}{\sqrt{R_{eye}^2 - R_{iris}^2 + R_{pupil}^2}} \tag{3.13}$$

For $\theta \neq 0$ we can calculate the location of the center points just by rotate the circles in space by θ and φ where we use the parameter τ to describe the circle.

$$\begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix}_i = \begin{pmatrix} 0 \\ 0 \\ z_{eye} \end{pmatrix} + \begin{pmatrix} \cos\varphi\cos\theta & -\sin\varphi & \cos\varphi\sin\theta \\ \sin\varphi\cos\theta & \cos\varphi & \sin\varphi\sin\theta \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} r_i\cos\tau \\ r_i\sin\tau \\ d_i \end{pmatrix} \tag{3.14}$$

where $\tau \in [0, 2\pi)$ and $i \in \{pupil, iris\}$.

Now we are able to calculate P^{-1} by mapping the circles for the pupil and the iris to silicon retina with (3.5). Because this will lead to complicated equations we'll look for an other way to derive P^{-1}. First of all, we recognize that φ just rotates the pattern but does not change the shape of it. So we can set $\varphi = 0$. Next, we consider the shape of the projection in space and we find, that the projection can be described by an oblique cone. Based on the conic section theory, we know the shape of P^{-1} is an ellipse. To calculate the ellipse parameters we choose the following approximation:

$$a_i = \frac{f\ \cos\theta R_i}{(z_{eye} - d_i\cos\theta)} \tag{3.15}$$

$$b_i = \frac{f\ R_i}{(z_{eye} - d_i\cos\theta)} \tag{3.16}$$

The center point of the ellipse is given by

$$\begin{pmatrix} x'_c \\ y'_c \end{pmatrix}_i = \frac{f\ d_i\sin\theta}{z_{eye} - d_i\cos\theta} \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix} + \begin{pmatrix} x'_{eye} \\ y'_{eye} \end{pmatrix} \tag{3.17}$$

and the rotation by φ.

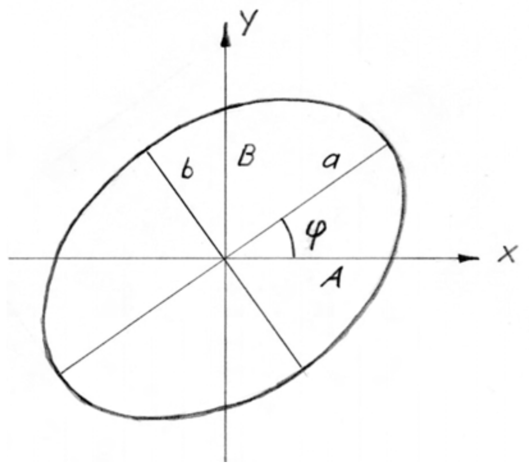


Figure 3.5: *Ellipse parameters*

3.3 Implementation

3.3.1 Parameter Calculation

The algorithm for writing the inverse pattern P^{-1} into the accumulator array $\boldsymbol{A}$ is designed for rotated ellipses (section 3.4), where the ellipses are described relative to the center (x'_c, y'_c) point by the equation

$$B_i^2 x^2 + 2C_i xy + A_i^2 y^2 - A_i^2 B_i^2 = 0 \tag{3.18}$$

where $i \in \{pupil, iris\}$ indicates the pattern for iris or the pupil.

For the circular pattern which is just a simple case of the rotated ellipse the parameters of (3.18) are given by

$$A_i^2 = R_i \tag{3.19}$$
$$B_i^2 = R_i \tag{3.20}$$
$$C_i = 0 \tag{3.21}$$
$$x'_c = e'_x \tag{3.22}$$
$$y'_c = e'_y. \tag{3.23}$$

For the elliptic pattern the parameters are given by

$$A_i^2 = \frac{a_i^2 b_i^2}{a_i^2 \sin^2(\varphi) + b_i^2 \cos^2(\varphi)} \tag{3.24}$$
$$B_i^2 = \frac{a_i^2 b_i^2}{a_i^2 \cos^2(\varphi) + b_i^2 \sin^2(\varphi)} \tag{3.25}$$
$$C_i = \frac{(b_i^2 - a_i^2)\cos(\varphi)\sin(\varphi) A_i^2 B_i^2}{a_i^2 b_i^2} \tag{3.26}$$

where φ, a_i and b_i are given by (3.8) (3.15) and (3.16). The center $(x'_c, y'_c)_i$ is given by (3.17).

The calculation of these parameters for each event during processing the algorithm is too expensive. Therefore we have to calculate the parameters for each possible event (e'_x, e'_y) before running the tracker and save them into a lookup table $\boldsymbol{T}_i$. So the lookup table consists of the elements

$$T_i(x', y') = \{A_i^2, B_i^2, C_i, (x'_c, y'_c)_i, w_i\}. \tag{3.27}$$

where w_i is an additional parameter to define different weights for the iris and the pupil.

Because we are not interested in events could not possibly be caused by eye movement respectively events outside the eye range, we create a mask array $\boldsymbol{M}$ of the same size as the retina, where $\boldsymbol{M}(x', y')$ is TRUE if (e'_x, e'_y) is a possible event and FALSE if not. In the actual tracker version, the range of possible events is defined as area inside the eye ball. It is clear, that we calculate all the parameters in $\boldsymbol{T}$ just for events (e'_x, e'_y) inside this area.

3.3.2 Tracker Loop

New Event: Now, we are ready to start the tracker. For each event (e'_x, e'_y), we receive from the silicon retina we first check if it is an event of interest. If $\boldsymbol{M}[e'_x, e'_y] =$ TRUE we read the parameters from $\boldsymbol{T}[e'_x, e'_y]$ and as we will remove the inverse pattern from the array later, we write the parameters into a ring buffer $\boldsymbol{R}$, where the length of the buffer is equivalent to the amount of events accumulated in $\boldsymbol{A}$.

Inverse Pattern to Accumulator: Next, we have to calculate the elements of the inverse pattern $(x'_p, y'_p) \in P^{-1}$ based on parameters $\boldsymbol{T}[e'_x, e'_y]$ and add the weight w to the accumulator array at position $\boldsymbol{A}(x'_p, y'_p)$. As this task has to be very fast we use a special algorithm described in section 3.4 which is based on recursive calculation with just integer additions and subtractions.

Eye Location As mentioned in section 3.1, we define the predicted eye position as the position of the maximum value in the accumulator array $(x'_{max}, y'_{max}) = \max \boldsymbol{A}$. Because it is too expensive to scan the whole array after each event, we just consider the values of $\boldsymbol{A}[x'_p, y'_p]$ calculated by adding the weight w. That is just an approximation but it makes sense in the way, that a value first has to grow before it becomes the maximum.

Threshold: While the eye does not move we receive just events caused by noise or by disruption. This leads also to a maximum at varying positions in $\boldsymbol{A}$. But the level of the maximum value is lower than the maximum value caused by eye motion. So we can try to exclude these effects by accepting a new maximum location just if it is bigger then the given *threshold* value.

Filter: In the actual tracker version a simple filter is implemented, which limits the step size in x' and y' from an old to a new maximum location to a constant, adjustable value. This filter is very simple, improves the output and respects, that the eye is continuously moving. The *caviar viewer* displays the unfiltered maximum location as a white dot, the filtered location as two blue ellipses for the iris and the pupil.

Resolve Gaze-Prediction Based on the filtered eye position x'_f and y'_f we can resolve the gaze respectively the angles θ_g and φ_g analog to equation 3.8 and 3.9. We just have to replace (e'_x, e'_y) by (x'_f, y'_f) and R_{eye} by the distance between the iris center and the eye center:

$$d_{ic} = \sqrt{R_{eye}^2 + R_{iris}^2} \tag{3.28}$$

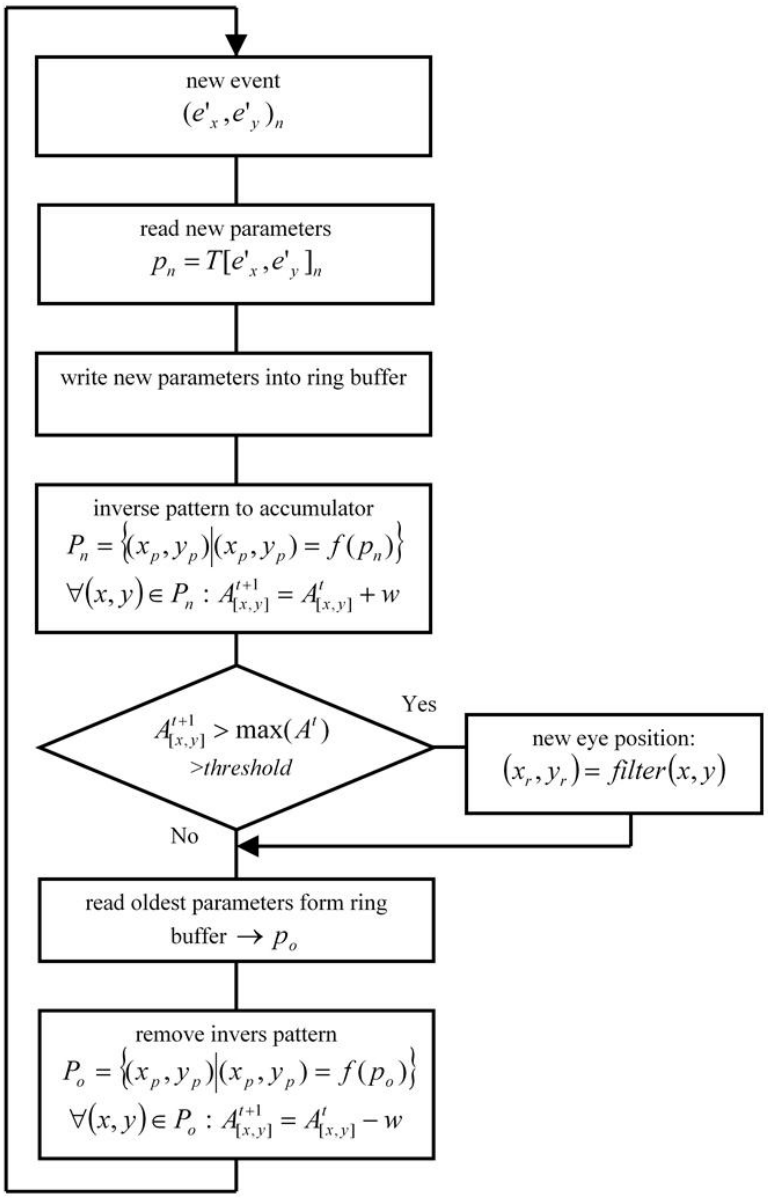
So we get:

$$\varphi_g = \arctan\left(\frac{x'_f}{y'_f}\right) \tag{3.29}$$

$$\theta_g = \arccos\left(\frac{-\left(x_f'^2 + y_f'^2\right) z_{eye} \pm f\sqrt{\left(x_f'^2 + y_f'^2 + f^2\right) d_{ic}^2 - \left(x_f'^2 + y_f'^2\right) z_{eye}^2}}{\left(x_f'^2 + y_f'^2 + f^2\right) d_{ic}}\right) \tag{3.30}$$

Remove Inverse Pattern from Accumulator: Finally we have to remove the oldest inverse pattern from $\boldsymbol{A}$. Otherwise the maximum value would grow to infinity. So we read the oldest parameters from the ring buffer and run the writing algorithm for the oldest parameter set again but this time with the negative weight $-w$.

Return: Now we are ready to handel the next event by going back to 3.3.2.



new event
$(e'_x, e'_y)_n$
read new parameters
$p_n = T[e'_x, e'_y]_n$
write new parameters into ring buffer
inverse pattern to accumulator
$P_n = \{(x_p, y_p) | (x_p, y_p) = f(p_n)\}$
$\forall (x,y) \in P_n : A^{t+1}_{[x,y]} = A^t_{[x,y]} + w$
$A^{t+1}_{[x,y]} > \max(A^t)$
>threshold
Yes
new eye position:
$(x_r, y_r) = filter(x, y)$
No
read oldest parameters form ring buffer $\rightarrow p_o$
remove invers pattern
$P_o = \{(x_p, y_p) | (x_p, y_p) = f(p_o)\}$
$\forall (x,y) \in P_o : A^{t+1}_{[x,y]} = A^t_{[x,y]} - w$

Figure 3.6: *Flow chart of the tracking algorithm.*

3.4 Fast Algorithm for creating a Rotated Ellipse

As described in the previous chapter, for each event (e'_x, e'_y) the chosen pattern has to be written into the accumulator array. If we assume that the pattern of the iris and the pupil contains about 200 pixels, it is clear that the writing algorithm has to be fast.

3.4.1 Lookup Table

One possibility to create a fast algorithm is, to write the pattern into a lookup table. So the algorithm just has to copy the pattern from the lookup table to the right position into the accumulator array (with respect to the event location (e'_x, e'_y)). This approach would be fast and suitable for static patterns. This means that just the location of the pattern but not the shape is depending on (e'_x, e'_y). If the pattern changes with the location, as it does with the elliptic eye model (section 3.2.2) it is not a good idea to use a lookup table because the table becomes huge. Instead, we look for a fast algorithm to calculate the pixels of a rotated ellipse.

3.4.2 Bresenham Algorithm

In computer graphics exists a well known, fast algorithm called *Bresenham Algorithm* for drawing lines, circles and *non* rotated ellipses on a pixel screen. The Bresenham Algorithm is based on a recursive calculation by just using integer addition and subtraction [7]. For rotated ellipses we need to extend the algorithm of the non rotated ellipses where we can use the ideas of [6] for the first part.

The common way to describe such ellipses uses trigonometric functions and floating point operations. But the numerical calculation of trigonometric functions is based on series expansion what is expensive. Instead we use a description as follows, where we use, that w.l.o.g. we can define the location of the ellipse center to the origin $(0, 0)$ because we just have to add the given ellipse location to the calculated pixel. So we can describe the rotated ellipse as:

$$f_e(x, y) = B^2x^2 + 2Cxy + A^2y^2 - A^2B^2 = 0 \tag{3.31}$$

where A^2, B^2 and $2C$ have to be rounded integer values. In a two dimensional continuous space, the rotated ellipse is a closed line where each element of this line fulfills the equation (3.31). In the discrete pixel space this definition has to be adapted, as it is impossible to fulfill for almost all pixels. Instead, we claim for each pixel of a closed line, that it reduces the equation (3.31) to a minimum. In this sense we call $f_e(x, y)$ the *ellipse error* which should be as small as possible.

Because the ellipse is a closed line, we can create it out of a given starting point (x, y), just by defining a direction of rotation and jumping from one pixel to one of the neighbors by add ± 1 to x or ± 1 to y. The idea is now to calculate the ellipse error for the neighbors and to choose the one with the smallest ellipse error f_e with respect to the chosen direction of rotation. The advantage of this approach is, that we get simple equations for $f_e(x \pm 1, y \pm 1)$. For example we calculate $f_e(x+1, y)$:

$$\begin{aligned} f_e(x+1,y) &= B^2(x+1)^2 + 2C(x+1)y + A^2y^2 - A^2B^2 \\ &= \underbrace{B^2x^2 + 2Cxy + A^2y^2 - A^2B^2}_{= f_e(x,y)} + 2B^2x + B^2 + 2Cy \\ &= f_e(x,y) + 2B^2x + B^2 + 2Cy \end{aligned} \tag{3.32}$$

We see, that the new ellipse error $f_e(x+1, y)$ is recursively depending on the former error $f_e(x, y)$ and it is also just linearly depending on x and y in the terms $2B^2x$ and $2Cy$, and this terms can also be calculated recursively. For example $2B^2(x + 1) = 2B^2x + 2B^2$. The terms for the other directions are similar (Tab. 3.1).

To avoid, that we have to calculate the ellipse error for all eight neighbors, it is a good idea to calculate also the slope of the ellipse by implicit differentiation:

$$\frac{dy}{dx} = -\frac{2B^2x + 2Cy}{2A^2y + 2Cx} \qquad \frac{dx}{dy} = -\frac{2A^2y + 2Cy}{2B^2x + 2Cy} \tag{3.33}$$

Let's define $dx := 2A^2y + 2Cx$ and $dy := -2B^2x - 2Cy$ which can also be calculated recursively. With the two variables dx, dy and the given direction of rotation, we are able to reduce the eight jump opportunities to just two, a *main* and a correction jump direction where both of them are parallel to a coordinate axis (Fig.3.7).

main direction: The differentiation of the ellipse equation into this direction with respect to the leading sign is between 0 and 1. So we always jump one pixel in this direction.

correction direction: The differentiation of the ellipse equation into this direction with respect to the leading sign is between 1 and ∞. We just jump one pixel in this direction, if the ellipse error becomes smaller then just jump in the main direction. (Note: As this jump just appears in combination with a jump in main

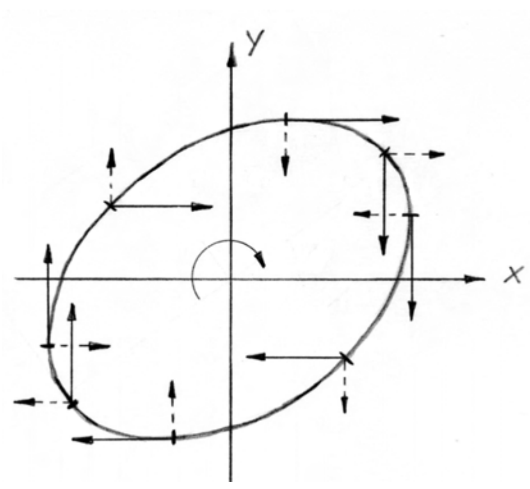


Figure 3.7: *Main direction $\rightarrow$ and correction direction $\dashrightarrow$ by clockwise rotation.*

direction, it is in fact a jump in diagonal direction. But the performance of the recursive algorithm is better, if we avoid diagonal jumps.)

If we walk along the ellipse form a given starting point in clock wise direction, the slope is continuous and monotonically decreasing except at positions, where the slope of the ellipse jumps from $-\infty$ to ∞. Therefore, we can start the algorithm at the slope range, with the biggest slope $dx/dy \in (\infty, 1]$. If we leave this range we arrive at the next smaller range $dx/dy \in (1, 0]$ and so on.
Because the ellipse is symmetric in point $(0, 0)$, we can map a calculated pixel (x, y) also to $(-x, -y)$. So we just have to calculate half of the ellipse.

3.4.3 Implementation

Now, we are ready to implement the rotated-ellipse algorithm. Because the implementation of the algorithm is not intuitive and clearly programmable, it follows a short overview, how it is done.

Initialization: To avoid multiplication during the loops and to be consistent with the code, we define new integer constants:

$$\begin{aligned} AA &:= A^2 \\ BB &:= B^2 \\ twoAA &:= 2AA \\ twoBB &:= 2BB \\ twoC &:= 2C \end{aligned} \tag{3.34}$$

To initialize the integer variables we have to choose a suitable starting point. For $x = 0$ we get $y = \pm B$. We choose $y = +B$ and initialize the following integer variables:

$$\begin{aligned} x &:= 0 \\ y &:= round(\sqrt{BB}) \\ fe &:= AA(y^2 - BB) \\ dx &:= twoAA * y + twoC * x \\ dy &:= twoBB * x + twoC * y \end{aligned} \tag{3.35}$$

where fe is now the ellipse error.

Loop: The recursive calculation of the ellipse pixels is based on a sequence of *while loops* where each while loop covers a slope range of the ellipse. The first while loop is displayed below.

```
//check slope range
while (dy > dx){
    // map weight to the Accumulator
    addWeightToAccumulator(centerX+x,centerY+y,weight);
```

```
        addWeightToAccumulator(centerX-x,centerY-y,weight);
        // jump in main direction: recursive variable calculation
        y  = y + 1;
        fe = fe + dx + AA;
        dx = dx + twoAA;
        dy = dy - twoC;
        // condition for correction step
        if (2*f > dy-BB) {
            // jump in correction direction: recursive variable calculation
            fe = fe - dy + BB ;
            dx = dx + twoC;
            dy = dy - twoBB;
            x  = x + 1;
        }
    //next slope range
    } while (dy > 0){

        ⋮

    }

    ⋮
```

The following table 3.1 shows a summary of the *while* condition with the corresponding *if* conditions and the equations for the recursive calculation for all while loops.

main step		correction step	
while	do	if	then
$dy^t > dx^t$	$fe^{t+1} = fe^t + dx^t + AA$	$2fe^{t+1} > dy^{t+1} - BB$	$fe^{t+2} = fe^{t+1} - dy^{t+1} + BB$
	$dx^{t+1} = \mathrm{d}x^t + twoAA$		$dx^{t+2} = dx^{t+1} + twoC$
	$dy^{t+1} = dy^t - twoC$		$dy^{t+2} = dy^{t+1} - twoBB$
	$y^{t+1} = y^t + 1$		$x^{t+2} = x^{t+1} + 1$
$dy^t > 0$	$fe^{t+1} = fe^t - dy^t + BB$	$2fe^{t+1} < -dx^{t+1} - AA$	$fe^{t+2} = fe^{t+1} + dx^{t+1} + AA$
	$dx^{t+1} = \mathrm{d}x^t + twoC$		$dx^{t+2} = dx^{t+1} + twoAA$
	$dy^{t+1} = dy^t - twoBB$		$dy^{t+2} = dy^{t+1} - twoC$
	$x^{t+1} = x^t + 1$		$y^{t+2} = y^{t+1} + 1$
$dy^t > -dx^t$	$fe^{t+1} = fe^t - dy^t + BB$	$2fe^{t+1} > dx^{t+1} - AA$	$fe^{t+2} = fe^{t+1} - dx^{t+1} + AA$
	$dx^{t+1} = \mathrm{d}x^t + twoC$		$dx^{t+2} = dx^{t+1} - twoAA$
	$dy^{t+1} = dy^t - twoBB$		$dy^{t+2} = dy^{t+1} + twoC$
	$x^{t+1} = x^t + 1$		$y^{t+2} = y^{t+1} - 1$
$dx^t > 0$	$fe^{t+1} = fe^t - dx^t + AA$	$2fe^{t+1} < dy^{t+1} - BB$	$fe^{t+2} = fe^{t+1} - dy^{t+1} + BB$
	$dx^{t+1} = \mathrm{d}x^t - twoAA$		$dx^{t+2} = dx^{t+1} + twoC$
	$dy^{t+1} = dy^t + twoC$		$dy^{t+2} = dy^{t+1} - twoBB$
	$y^{t+1} = y^t - 1$		$x^{t+2} = x^{t+1} + 1$
$dy^t < dx^t$	$fe^{t+1} = fe^t - dx^t + AA$	$2fe^{t+1} > -dy^{t+1} - BB$	$fe^{t+2} = fe^{t+1} + dy^{t+1} + BB$
and $x^t > 0$	$dx^{t+1} = \mathrm{d}x^t - twoAA$		$dx^{t+2} = dx^{t+1} - twoC$
	$dy^{t+1} = dy^t + twoC$		$dy^{t+2} = dy^{t+1} + twoBB$
	$y^{t+1} = y^t - 1$		$x^{t+2} = x^{t+1} - 1$
$dy^t < 0$	$fe^{t+1} = fe^t + dy^t + BB$	$2fe^{t+1} < dx^{t+1} - AA$	$fe^{t+2} = fe^{t+1} - dx^{t+1} + AA$
and $x^t > 0$	$dx^{t+1} = \mathrm{d}x^t - twoC$		$dx^{t+2} = dx^{t+1} - twoAA$
	$dy^{t+1} = dy^t + twoBB$		$dy^{t+2} = dy^{t+1} - twoCC$
	$x^{t+1} = x^t - 1$		$y^{t+2} = y^{t+1} - 1$
$dy^t < -dx^t$	$fe^{t+1} = fe^t + dy^t + BB$	$2fe^{t+1} > -dx^{t+1} - AA$	$fe^{t+2} = fe^{t+1} + dx^{t+1} + AA$
and $x^t > 0$	$dx^{t+1} = \mathrm{d}x^t - twoC$		$dx^{t+2} = dx + twoAA$
	$dy^{t+1} = dy^t + twoBB$		$dy^{t+2} = dy - twoC$
	$x^{t+1} = x^t - 1$		$y^{t+2} = y + 1$
$x^t > 0$	$fe^{t+1} = fe^t + dx^t + AA$	$2fe^{t+1} < -dy^{t+1} - BB$	$fe^{t+2} = fe^{t+1} + dy^{t+1} + BB$
	$dx^{t+1} = \mathrm{d}x^t + twoAA$		$dx^{t+2} = dx^{t+1} - twoC$
	$dy^{t+1} = dy^t - twoC$		$dy^{t+2} = dy^{t+1} + twoBB$
	$y^{t+1} = y^t + 1$		$x^{t+2} = x^{t+1} - 1$

Table 3.1: *While loop sequence for rotated ellipse.*

Chapter 4

Results

4.1 Gaze Measurement

To give an impression of the performance of the algorithm described in this report, we apply the eye tracker on a target displayed on a 1024x768 laptop display. The test setup is displayed in figure 4.1. Each time a key is typed, the actual tacker position is stored in a file and the target jumps to the next position. Figure 4.2 shows the results of this measurement for three different target locations. The estimated eye location is displayed in the (x', y') plane of the silicon retina. With equations (3.29) and (3.30) we are able to calculate the gaze direction based on the estimated eye location. Figure 4.2 shows that the tracker is able to estimate the gaze direction with a good repeatability and with a resolution which is better than a third of the screen size. We also see, that variance of the estimated eye position is in a range of the resolution of the silicon retina ($2\sigma \approx 5 pixels$) what will be a limitation for optimizing the tracker.

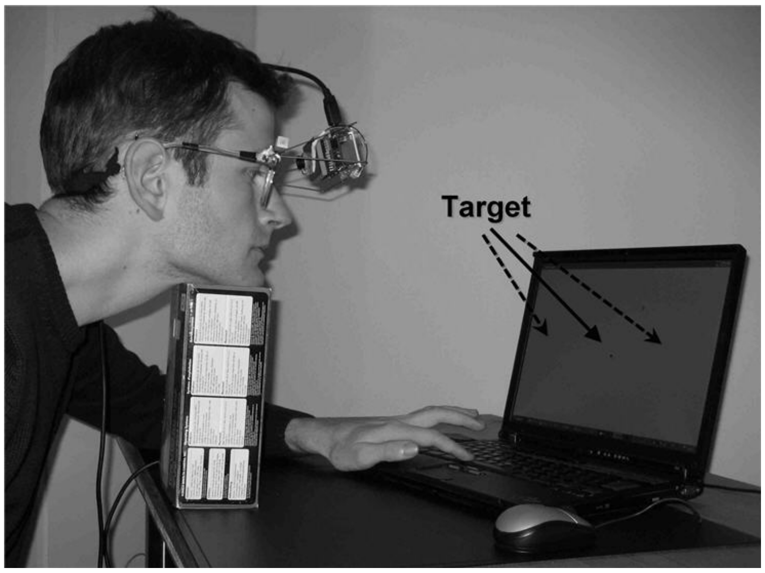


Figure 4.1: *Setup for the Gaze measurement.*

4.2 Improvement Opportunities

It follows a list with some suggestions for optimizing the eye tracker.

- As mentioned before, the resolution of the silicon retina will be a limitation for optimizing the tracker. A simple solution to retard this problem would be to use a more appropriate lens which allows to set the eye better in focus. An other solution would be to build a silicon retina with a higher resolution.
- In section 3.2 two patterns are suggested, a circular pattern and an elliptic pattern based on a spatial eye camera model. Maybe it is possible to improve the pattern by using broader borderlines or by using a more appropriate model. Especially the weight could be an important parameter to optimize the tracker.
- Actually, parameters of the filters have to be adjusted manually. Maybe it is possible to write a routine, that adjusts some of these parameters automatically. This would be desirable especially for the varying radius of the pupil.
- The filter implemented to smooth the estimated eye position is just a simple try and provides a huge range of opportunities for improvement.

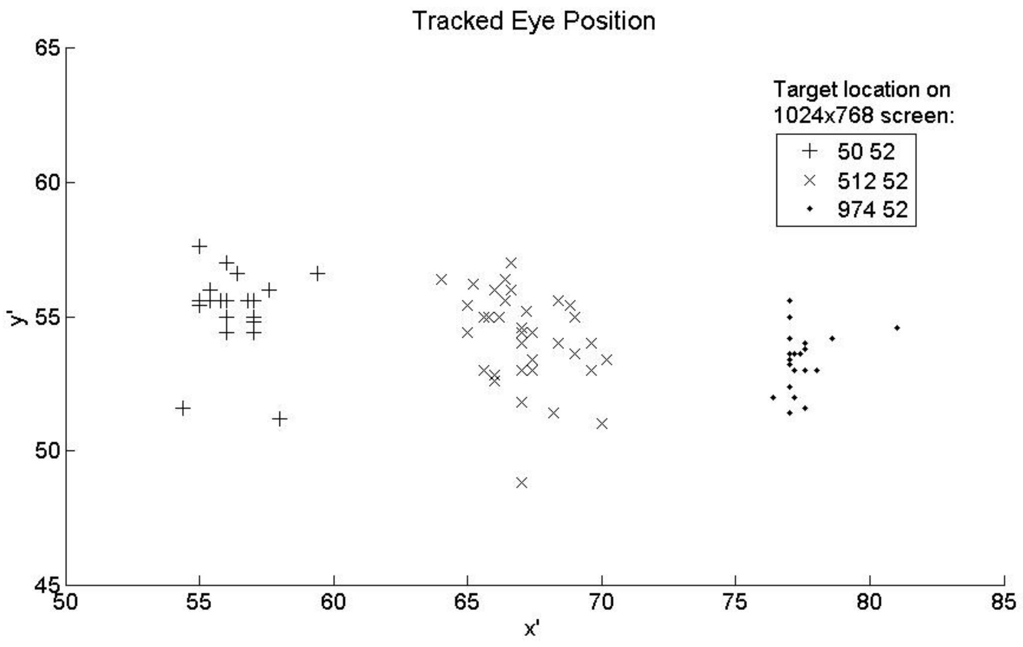


Figure 4.2: *Eye position estimated by the eye tracker for three different target locations.*

4.3 Conclusion

As shown in figure 4.2 the tracking algorithm introduced in chapter 3 is able to estimate the gaze direction by using a conventional 1.6 GHz Laptop PC. Actually, the resolution of the tracker is not good enough for commercial applications, but there exist a lot of opportunities for improvement. First of all, by increasing the resolution either by replace the lens to set the eye ball better in focus or by improving the resolution of the silicon retina. With these improvements the silicon retina combined with a well suited tracking algorithm should be able to provide a fast and accurate estimation of the gaze direction in real time.

Bibliography

[1] P. Lichtsteiner, C. Posch, and T. Delbruck: A 128x128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change. ISSCC Digest of Technical Papers 2006 508-509, 2006.

[2] T. Delbruck: CAVIAR project resources.
http://www.ini.unizh.ch/ tobi/caviar/index.php

[3] D. W. Hanson: Comitting Eye Tracking. IT University of Copenhagen, 2003.

[4] M. Seul, L. O'Gorman, M.J. Sammon: Practical Algorithms for Image Analysis. Cambridge University Press, 2000.

[5] W. Burger, M. James Burge: Digitale Bildverarbeitung : eine Einführung mit Java und ImageJ. Springer, 2005.

[6] Manfred Dürschmid: Jenseits des Hyperbolikus: Effektive Algorithmen zum Zeichnen von beliebigen Ellipsen, Hyperbeln und anderen algebraischen Kurven höherer Ordnung. http://home.arcor-online.de/manfred.duerschmid/software/elli.htm

[7] J. Kennedy: A Fast Bresenham Type Algorithm For Drawing Ellipses. Mathematics Department Santa Monica College, http://homepage.smc.edu/kennedy

Appendix A

Manual

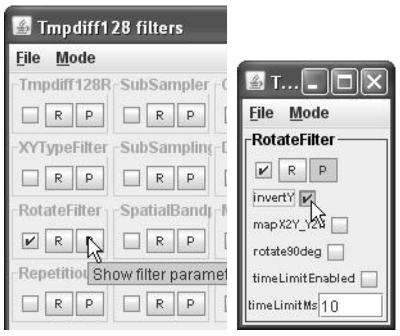
A.1 Run logged Example

To get a short impression of the eye Tracker, you can run the following logged-data file with the given filter parameters.

1. start *Caviar Viewer*

2. open the logged data file `Example1_Log.dat` located at
   `...INI-AE-Biasgen/doc/DamianSemesterarbeit/examples`

3. Because the camera is fixed upside down, we have to enable the checkboxes of *RotateFilter* and *invertY* in its parameter Window (Fig. A.1). The data are now displayed in 'mirror style'.



RotateFilter ✓
invertY ✓
mapX2Y_Y2X ☐
rotate90deg ☐
timeLimitEnabled ☐
timeLimitMs 10

Figure A.1: *Enable the* rotateFilter *and* invertY.

4. set the filter parameter of the *HoughEyeTracker* as displayed (Fig. A.2).

A.2 HoughEyeTracker Setting Up

1. Put on the glasses frame with the silicon retina and plug in the USB2 connector.

2. Start the *Caviar Viewer*. Now the data captured by the retina should be displayed on the screen. (If not, see *software setup* on [2])

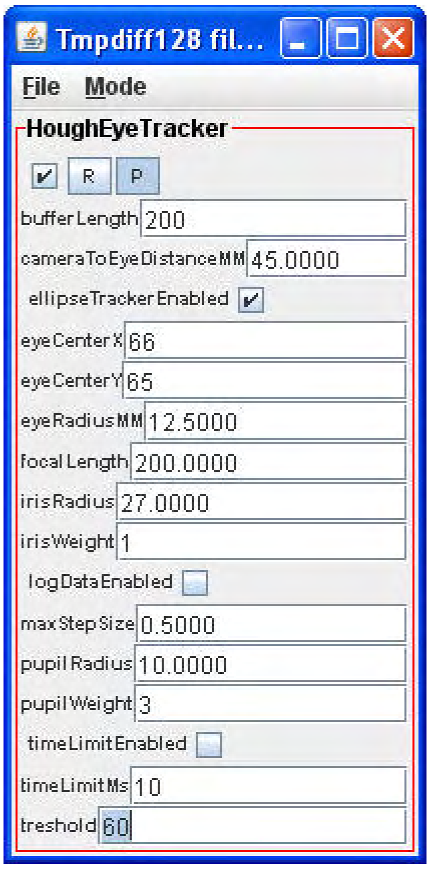
3. Because the camera is fixed upside down, we have to enable the checkboxes of *RotateFilter* and *invertY* in its parameter Window (Fig. A.1). The data are now displayed in 'mirror style'.

4. To activate the *HoughEyeTracker*, enable its checkbox and open its parameter window by clicking *P*.

5. Now you have to adjust the silicon retina according to Fig. 2.1: Set *eyeCenterX* and *eyeCenterY* to 64. By trying to look directly into the camera and observing the eye location in the screen, adjust the camera position, so that the eye is located more or less in the center of the displayed screen respectively in the center of the displayed eyeball circle. (Hints: Try to look 'through' the camera to the screen and move the head just a little bit during the eyes are fixed on the screen to make the eye visible).

6. If the location of the eye after the previous step is not exactly in the center of the eyeball circle, adjust *eyeCenterX* and *eyeCenterY*. Now the situation should look as displayed at Fig. A.3.

7. Measure the distance between the eye surface to the silicon retina lens (Fig.A.4) and enter the value in millimeter into *cameraToEyeDistanceMM*.



bufferLength 200
cameraToEyeDistanceMM 45.0000
ellipseTrackerEnabled ✓
eyeCenterX 66
eyeCenterY 65
eyeRadiusMM 12.5000
focalLength 200.0000
irisRadius 27.0000
irisWeight 1
logDataEnabled ☐
maxStepSize 0.5000
pupilRadius 10.0000
pupilWeight 3
timeLimitEnabled ☐
timeLimitMs 10
treshold 60

Figure A.2: *Parameters for the logged data file.*

8. The *eyeRadiusMM* for an adult is about 12.5*mm*.
9. Enter the *irisRadius* and the *pupilRadius* in pixels by comparing the displayed model with the events generated by iris motion.
10. Set the *irisWeight* to 1 and the *pupilWeight* to 3.
11. The threshold reduces the influence of noise and disturbance. The tracker accepts a new location just if their accumulator value is bigger than the *threshold* value. The threshold (red dot) and maximum value of the accumulator (blue bar) is displayed at the left bottom of the screen. Set *threshold* high enough, so that the blue bar reaches it just if the eye is moving. A normal threshold for the weights suggested above is about 60.
12. For the circular tracker (section 3.2.1) disable *ellipseTrackerEnabled*, for the ellipse tracker (section 3.2.2) enable *ellipseTackerEnabled*.

Now the tracker should be ready. The *caviar viewer* displays the unfiltered maximum location as a white dot, the filtered location as the two blue ellipses for the iris and the pupil.

A.3 Filter parameters

It follows a short description of all filter parameters and some remarks.

bufferLength: Defines the number of events respectively of the corresponding ellipse parameters stored in the ring buffer. A good value for the buffer length is about

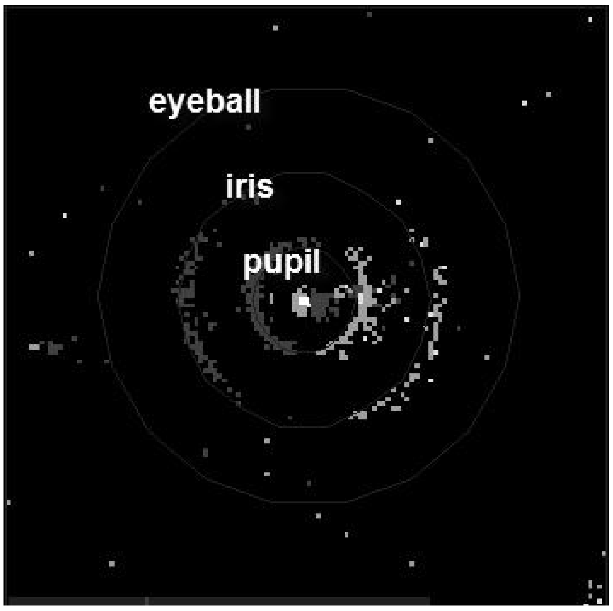


Figure A.3: *Adjusted eye position.*

200. If the buffer is too long, the reaction time is slow. If the buffer is too short, the tracker wont be able to recognize the pattern.

cameraToEyeDistanceMM: Distance from the top of the retina lens to the eye in milimiters (Fig.A.4). For measuring I used to fit two paper strips between the eye and the camera and measure the length of them.

ellipseTrackerEnabled: If the checkbox is not enabled, the tracker is fixed on two concentric circles with the radius of the iris and the pupil (section 3.2.1). If the checkbox is enabled, the tracker is based on an eye model, which respects the elliptic mapping of the iris and the pupil to the silicon retina (section 3.2.2).

eyeCenterX: Location of the mapped eye ball in x-direction.

eyeCenterY: Location of the mapped eye ball in x-direction.

eyeRadiusMM: Radius of the eye ball in millimeters. For an adult person, the eye radius is $12.5mm$.

focalLength: Parameter of the pinhole model described in section 3.2.2. For the used set up, the focalLength is about 200.

irisRadius: Radius of the iris in pixels. This parameter can be adjusted by comparing the displayed iris model with the events generated by iris motion.

irisWeight: The value added to the accumulator array for each inverse iris pattern element. As the iris is often partially hidden by the lid, it is better to choose a small iris weight (for example 1).

logDataEnabled: Opens a window for measuring the performance of the tracking algorithm. On the window a gaze target is displayed. Always if a key is pressed, the actual tracker position is written to a file and the target jumps to a new location.

maxStepSize: If the tracker detects a new location the filter reduces the step to the new location to a limit. Where the limit is defined by the *maxStepSize* parameter.

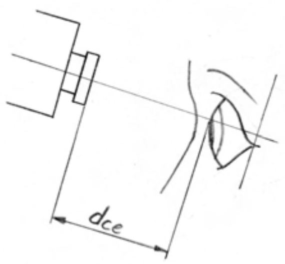


Figure A.4: d_{ce} = *cameraToEyeDistanceMM*

pupilRadius: Radius of the pupil in pixels. This parameter can be adjusted by comparing the displayed pupil model with the events generated by pupil motion.

pupilWeight: The value added to the accumulator array for each inverse pupil pattern element. Unlike the iris, the pupil shape is mostly not hidden by the lid. Therefore the pupil is more important to get a good tracker performance. So the iris weight should be higher than the iris weight (for example 3).

treshold: To reduce the influence of noise and disturbance, the tracker accepts a new location just if their accumulator value is bigger than the *threshold* value. The threshold (red dot) and maximum value of the accumulator (blue bar) is displayed at the left bottom of the screen. A normal threshold for the weights suggested above is about 60.

Progress on an Eye Tracking System Using Multiple Near-Infrared Emitter/Detector Pairs with Special Application to Efficient Eye-Gaze Communication

Dale Grover
Electrical and Computer Engineering Department
Michigan State University
2120 Engineering Building
East Lansing, MI 48824, USA
groverda@egr.msu.edu

Keywords
Eye tracking, eye-gaze communication, word-level disambiguation

Introduction

The "Owl" eye tracker uses a small sensor (Figure 1) mounted close to the user's eye, bouncing multiple channels of unfocused near-infrared light on and around the user's eye to determine direction of gaze. The hardware makes no assumption about what structures reflect light from any particular emitter/detector combination - it is thus not subject to the setup and adjustment constraints of techniques like limbus reflectometry. Specifically, specular and diffuse reflections may come from the iris/cornea, sclera, eye lids, and surrounding tissue, rather than from specific regions of the eye. Whereas conventional gaze-trackers use accurately aimed and focussed hardware along with detailed assumptions about the geometry of the eye, the Owl's approach is to use software to extract useful information from the raw unfocused data.

The Owl was developed in Ithaca, New York by Martin King in the early 1980's for augmentative and alternative communication users. Originally conceived as a direct-select device, it had a fixed display of approximately 36 hexagonal cells to allow typing English text, one letter per dwell. This arrangement required steady physical mounting due to sensor mass, good gaze detection accuracy, and resulted in a less than cosmetic appearance for the user, as well as occluding one eye. The later development of efficient word-level disambiguation (described below) allowed the display to be eliminated, leaving only eight peripheral LEDs as both targets and feedback, and producing a system that generates unlimited English text with one dwell per letter using only eight or nine cells. The reduced size of the sensor, and proximity to the eye (out of focus), means the users can still see out of both eyes, rather than having one occluded purely for communication needs.

Hardware

The present Owl sensor is an annulus 2.4 cm in outside dimension, with a 1.7 cm interior hole, and weighs less than 5 gm (<0.2 ounces). Eight LEDs and eight phototransistors are arrayed uniformly around the sensor, and a total of 64 different measurements of reflection are taken 50 or 60 times a second. The 700 nm "near-infrared" wavelength LEDs are just visible to the human eye, but also match well to the phototransistors and thus serve two purposes on the compact sensor. A lightweight, flexible cable with just three conductors suffices for signalling between the sensor and an interface board which converts the signal using a 3-bit variable gain amplifier and a 12-bit ADC with quite low sampling requirements (<100k samples/second). Bidirectional communication with the host computer is via a serial or USB connection at less than 100k bps.

Data Processing

In operation, the 8 LEDs of the sensor are turned on one at a time, and reflected light measurements from the 8 phototransistors are taken. The entire cycle is repeated 50 or 60 times a second, ideally locked to the AC line frequency since ambient interior light may have significant line frequency components. At this frequency, the user perceives the LEDs to be glowing steadily. The resulting 64 measurement channels include 8 from adjacent emitter/detector pairs, which are discarded due to excessive direct coupling. Noise is reduced using a filter that can accommodate discontinuities without excessive smoothing, such as the Savitzky-Golay filter (also known under various names such as least-squares polynomial filter). A third-order Savitzky-Golay filter of length 11 is currently used for 60 Hz frame rates.

The filtered channels are reduced by projection from 56 channels to 2 orthogonal components. The projection matrix is produced using a technique such as Principal Component Analysis (PCA). Given data from a brief calibration, the arbitrary orientation of the PCA-derived projection can be corrected via a rotation/flip matrix and the results used with parametric classifiers, though non-parametric classifiers would not need the geometric correction.

Once projected, each frame can be classified as fixation, saccade, blink. etc., with varying degrees of delay based on the method used. At present, fixations are identified using a variation of the method described by Sibert, et al (Sibert 2000), wherein eye positions within approximately 0.5 degree for 0.1 seconds are taken as the start of a fixation, further positions within a degree are taken as continuations, and 0.05 seconds of positions greater than a degree are taken as the start of a saccade. Another dwell-time fixation detection algorithm such as "N of M samples within distance D of the mean μ" (Duchowski, 2003) is implemented for future comparison.

It should also be noted that unlike image-based processing, it is possible to use alternative projections (e.g., based on Multiple Discriminant Analysis) for different classification tasks - for example, an alternative projection might produce superior results for blink classification versus PCA.

Once a group of contiguous measurements are classified as a fixation, the fixation can be classified according to target. The reduced number of targets (i.e., 8 or 9) and low processing devoted to extracting position data thus far leaves a great deal of flexibility in choosing one or more algorithms for target classification. Simple but reasonably robust non-parametric methods such as variations on the "K-Nearest-Neighbor" algorithm are useful early in a user's session when little data is available. After a short period of use, sufficient statistics will be available for a wider range of classifiers, especially parametric classifiers.

The processing requirements for the Owl are quite low. Filtering incoming data (60 Hz) requires on the order of 67,000 floating (or fixed-point) operations a second, and projection of the data to 2 components 13,000 operations a second. Thus on the order of 80,000 floating (or fixed-point) calculations per second comprise the main processing burden for generating "X/Y" eye tracking data.

During calibration, the projection matrices must be found. Calculation of the required covariance matrix for 4 seconds of raw data requires approximately 750,000 operations (once the data has been normalized and bias removed). PCA requires fewer operations depending on exact algorithms used, but a common implementation using optimized LAPACK routines on a 1 GHz PowerPC took 3 msec. For comparison, a frame at 60 Hz is 16.7 msec.

The data communication and processing requirements can thus be met without undue burden on a modern computer, reducing the cost of the hardware which needs only to make the measurements and convey them to the user's personal computer.

The Owl and Word Level Disambiguation

The dramatic reduction in price due to both high volume and advancing technology for small, easily interfaced video imaging devices (e.g., CMOS USB web cams) largely removes the historical benefits of the Owl technology as a general purpose eye tracking technology. The need to mount the Owl on the user's glasses and the fact that it provides only relative gaze information are further reasons why the Owl is not a compelling eye tracker for general purpose use.

The present research is motivated however by combining the sensor with word-level disambiguation, similar to methods now found on mobile phones. In word-level disambiguation, each of a small number of targets has multiple meanings - as for example a telephone keypad with multiple letters per key. The user selects a sequence of targets, and in the preferred implementations, the system presents the user with the most likely interpretations of those sequences. The characteristics of most languages are such that efficiencies that approach one letter generated per target selected are easily achieved - even when the letters are arranged inefficiently on keys (e.g., alphabetically). Since spelling is typically used, no special encoding method need be learned beyond spelling (though this could be an issue for some users), and vocabulary is unlimited.

Of particular importance is that visual feedback to the user can be limited to the LEDs in the sensor during target selection (target meaning is static within a sequence), and disambiguation at the end of sequences is often not required or minimal, and can be potentially replaced by auditory or other non-visual feedback. In many situations, the user thus has a cognitive load more akin to touch-typing than a feedback-intensive operation such as scanning. Since the interpretation of sequences remains static in preferred implementations, words can be generated by rote. A dynamic display, aside from the feedback offered by the Owl LEDs, can be consulted as desired, rather than being a necessity. Nor need the user maintain a relatively fixed relationship to a display, as eye position is significant relative to head position only. This provides more opportunity for eye contact with communication partners and less potential for physiological problems associated with a constrained position.

In addition, the reduction in the number of targets relaxes the accuracy requirements for the eye tracker, which should lead to more robust performance of the system.

Present Status

Offline analysis of data (see Figure 2) using the filtering and projection techniques discussed reveals good separation of targets. These techniques are being incorporated into a real time demonstration program hosted on Mac OS X, which continues to also support logging raw data for offline algorithm development (e.g., in Octave) as well.

A central issue with the Owl is how to address issues such as changes in sensor position (e.g., due to slippage of glasses down the user's nose) or ambient light. The program is written to keep time-tagged data histories to support exploration of classification algorithms that can robustly, and gracefully, handle these changes without resorting to a disruptive full calibration.

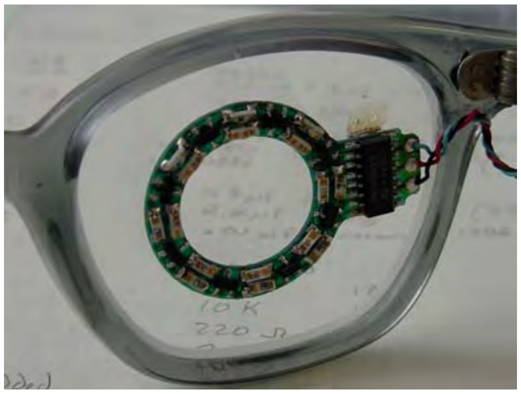


Figure 1. Owl sensor mounted on eyeglasses

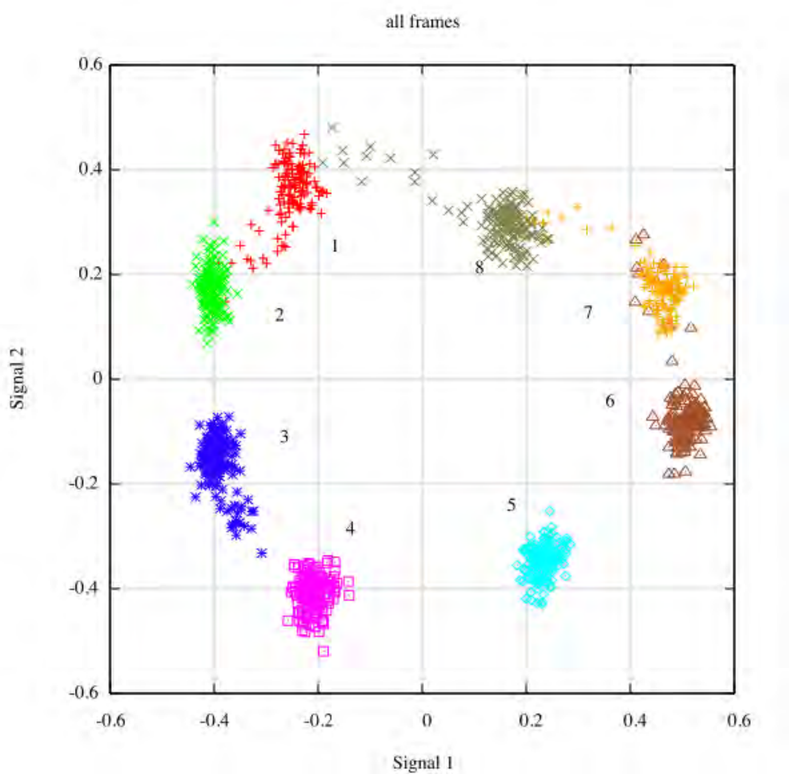


Figure 2. Frame data projected using Principal Component Analysis

References

Duchowski, A. (2003) *Eye Tracking Methodology: Theory and Practice*. Springer-Verlag, London, p. 115.

Sibert, L., Jacob, R., and Templeman, J. (2000) *Evaluation and Analysis of Eye Gaze Interaction*, NRL report. Naval Research Laboratory, Washington, D.C., p5.