Computer Science Tripos Part III

Gaze Tracking for Commodity Portable Devices

Erroll Wood
eww23@cam.ac.uk

Gonville & Caius College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for Computer Science Tripos Part III

June 21, 2013

Proforma

Name:	Erroll Wood
College:	Gonville & Caius College
Project Title:	Gaze Tracking for Commodity Portable Devices
Examination:	Computer Science Tripos, June 2013
Word Count:	11,901
Project Originator:	Dr Andreas Bulling
Supervisor:	Prof. Peter Robinson

Declaration

I, Erroll Wood of Gonville & Caius College, being a candidate for Part III of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date **June 21, 2013**

Abstract

I present the design of a novel gaze tracking system for commodity portable devices, taking their limited sensor hardware and typical use conditions into account. Current gaze trackers require specialist hardware so are confined to laboratories. If gaze could be tracked pervasively, visual attention studies could be easily deployed over a large population, and we could augment traditional interaction with attentive interfaces or multi-modal input.

We follow Wang et al.'s [1] general approach – fitting an ellipse to the user's iris in the image, and back-projecting it to a 3-D circle, thus determining eye orientation. Our system adapts and extends a range of gaze tracking techniques and algorithms to ensure robustness and reliability under lighting changes. Unlike typical systems, our model-based approach can be used instantly without user-specific calibration.

I evaluate the system in two ways: a technical analysis where we demonstrate tracking accuracy comparable to other non-specialist systems (up to 2.6°), and a user study in which we partially recreated Kumar et al.'s [2] EyePassword experiment, replacing expensive specialist equipment with a consumer tablet. These inital results are promising, and suggest our approach has potential for cheap portable devices of the near-future.

Contents

Chapter 1

Introduction

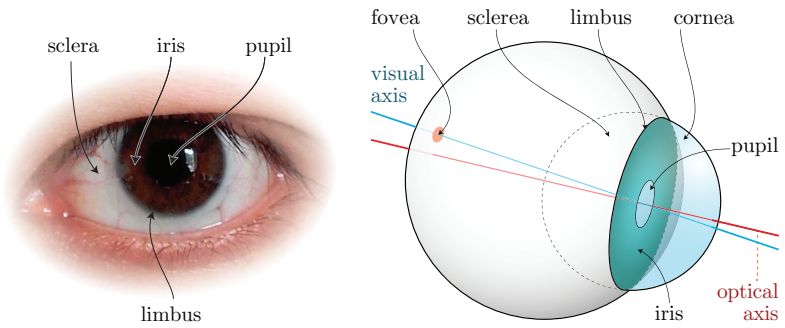
Humans are very visual beings. Our eyes and where we look convey our attention, interests, emotions, and relationships [3]. A *gaze tracking* system measures where someone is looking, either as the direction they are looking in, or point in space (or on a screen) they are looking at.

These are widely used to measure *visial attention* in both academic and commercial studies, including marketing, interface design, and psychology [4]. By identifying and analysing gaze patterns, we can speculate on what someone is thinking about. For example, if a user's gaze lingers on a particular photograph or piece of text, it might indicate they are interested in that subject.

Despite being a rich source of information, gaze is not used much in the public domain. This is because most gaze trackers use expensive specialist equipment (Figure 1.2), and require tedious and frequent calibration sessions [3]. This confines them to laboratory settings, preventing them being used *pervasively* – where gaze could be spontaneously tracked in everyday life without preparation or effort. If this were possible, marketing studies could be deployed directly into the consumer domain, enhancing mobile advertising – a popular revenue stream for app publishers [5]. Traditional interaction could be augmented with multimodal techniques that combine gaze with other inputs, or attentive interfaces that adapt to the user's attention [6].

In this dissertation I present a gaze tracking system developed for pervasive use with commodity portable devices[1]. I first describe the limitations of typical gaze tracking systems and explain design considerations. I then present description of our system's three main modules: eye localisation, limbal ellipse fitting, and gaze geometry. Finally I discuss a feasibility study and show it can track gaze on a device screen to within 7.7mm from a distance of 20cm (2.6°), comparable to other non-specialist hardware systems [3, 7].

[1] *Commodity* hardware refers to cheap, readily-available, off-the-shelf products (e.g. tablets and phones), as opposed to expensive specialist equipment.



(a) External eye anatomy **(b)** A geometric eye model

Figure 1.1: Internal and external eye structure. (b) shows the difference between visual and optical axes.

1.1 Human eye anatomy

Before discussing gaze tracking techniques, I briefly introduce the eye's anatomy. The eye itself, or *eyeball*, is a hollow spheroid. Its outermost layer is composed of two distinct regions: the *sclera*, commonly known as "the white of the eye"; and the *cornea* – a transparent protective window into the eye (Figure 1.1). The circular sclera/cornea boundary is known as the *limbus*.

The middle section of the eyeball contains the *iris* – the visible coloured portion of the eye. The circular aperture at its center is the *pupil*. The iris is shaped like a flattened doughnut and formed of smooth muscle that allows it to adjust the pupil's size, thus regulating the amount of incident light [8]. Iris colour varies between people, ranging from light (blue) to dark (brown)[2].

The inner layer of the eyeball is the *retina*, housing a neural layer of *photoreceptors* which produce signals in response to light. A pinhead-sized densely packed area of photoreceptors known as the *fovea* is located near the back of the eyeball. This provides a small area of critically-focused vision – about a thousandth of the entire visual field [8].

1.1.1 Approximate geometric model

In gaze tracking, the human eye is often approximated as two spheroid surfaces with different curvatures, representing the cornea and eyeball (Figure 1.1b). The *optical axis* is the line intersecting the centres of the pupil, cornea, and eyeball; and the *visual axis* is the line intersecting the fovea and pupil. The visual axis better represents the true direction of gaze.

[2]The iris is named after the Greek goddess *Iris* – the personification of the rainbow [8].

(a) Tobii X120 gaze tracker **(b)** head-mounted system

Figure 1.2: Typical gaze tracking systems. (a) shows a tabletop active IR system used in a portable device attention study [9].

1.2 Typical gaze tracking limitations

Image-based gaze trackers generally contain two subsystems – *eye localization* and *gaze estimation* [3]. Eye localisation detects the existence of eyes in the image and determines their image coordinates. Then gaze can be estimated by analysing image features related to the eye's structure.

1.2.1 Active infrared techniques

The majority of gaze tracking systems illuminate the user with *active infrared* [IR] light. Active IR is popular as it is invisible to humans, does not cause pupillary contraction, and provides stable lighting independent of external changes. Eye localization is trivial due to reflective properties of the eye, as the pupil is made to appear bright or dark (example: Figure 2.7a). Gaze is then estimated using corneal reflections, either with a simple mapping or a geometric model.

A practical limitation of active IR is its unreliability in the presence of sunlight. This is because natural light from the sun floods IR cameras, so users must wear wide-brimmed hats [10] or remain indoors. Other issues originate from the complicated and expensive equipment. Multiple IR emitters and cameras require tedious calibration sessions which must be repeated if the setup is moved, and the high-quality cameras remain too pricey for consumer use ($\geq$£10,000 [11]).

1.2.2 Head-mounted systems

A recent trend in gaze tracking is the use of head-mounted cameras to address some of these issues. These acquire IR images of the eye at a close range, avoiding issues that would arise from lighting changes or head movement. The extreme close range allows the use of cheaper, lower resolution cameras which can still detect physically small eye features such as the pupil or eyelashes.

These systems are known as *mobile* gaze trackers, as opposed to *remote* ones which sit on a desk.

The limitation of this approach is the awkward equipment and its proximity to the face. Until these shrink significantly, the discomfort and distraction they invoke makes them inappropriate for pervasive use. These systems are also not readily available to the public, or require significant effort to build.

1.3 This project

The goal of this project was to explore approaches suited to cheap commodity portable devices, taking their limited sensor hardware and typical use conditions into account. We investigate gaze tracking's feasibility using the minimal information available from a portable device.

My main contribution is the design of a model-based[3] gaze tracker for pervasive use with unmodified tablets – the first of its kind. I show that it satisfies requirements of pervasive use:

- **Low quality camera support**: Though devices may be equipped with high quality rear-facing cameras, the front-facing[4] counterpart often takes low resolution, noisy, out-of-focus images. Our system is capable of analysing eye-features despite this.
- **Reliability in different lighting conditions**: Commodity device cameras operate in the visible light spectrum, so cannot use IR to normalize scene lighting. Our system is robust under a range of realistic lighting conditions.
- **Calibration-free operation**: Typical systems require tedious calibration procedures which are unsuitable for devices which may be used spontaneously and in short bursts. With our calibration-free system, a user can have their gaze tracked instantly.

In an evaluation over eleven participants we show the system can track gaze on a screen to within 7.7mm (2.6°), sufficient for many applications. This result is promising, and suggests our approach has potential for cheap consumer devices of the near-future. We also recreate a gaze-interaction study [2], replacing an expensive specialist system with a consumer tablet. This shows our approach and implementation framework is feasible for gaze studies.

1.3.1 System overview

Figure 1.3 presents the overall system architecture. It comprises three main subsystems that this dissertation describes in turn:

[3] Gaze is estimated using 3-D geometry.
[4] Users face the device screen, so our system uses its *front-facing* camera.

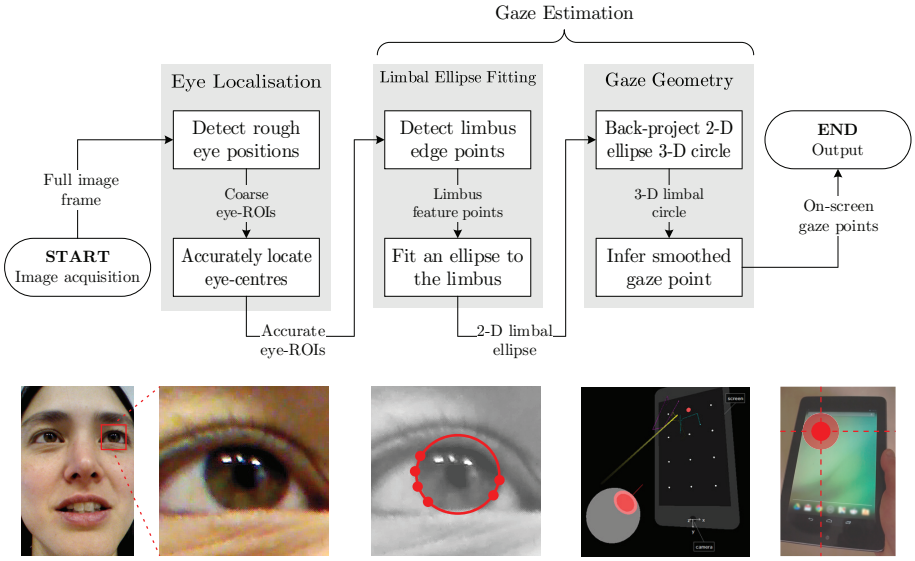


Figure 1.3: System architecture diagram presenting how a screen-space gaze point is computed from a camera image frame.

1. **Eye localisation** (chapter 2)
   We first robustly and accurately track eyes in the camera video feed, extracting eye-ROIs – minimal sub-images of the eye.

2. **Limbus ellipse fitting** (chapter 3)
   We then fit an ellipse to the limbus in the image's eye regions.

3. **Gaze geometry** (chapter 4)
   Finally we reverse the projective image formation process, using the 2-D ellipse to determine the real-world 3-D limbal circle, and thus estimate the gaze direction.

1.3.2 Related work

Wang et al. [1] first introduced single-camera visible light gaze tracking using the iris-contour (limbus) with their "one-circle" algorithm, which has since been extended by Wu et al. [12]. This project adapts their general approach – we calculate the normal to the 3-D limbal circle by back-projecting a 2-D limbal ellipse.

However, they used high quality DLSR cameras with zoom lenses, and only tested under optimum lighting conditions with unrepresentative participants (dark irises). This allows use of simpler techniques which are not compatible with our design considerations. They also operate on single images rather than video. This project adapts and extends on their general approach.

Our system borrows techniques from webcam gaze trackers which support low quality images. Valenti et al. [13] describe a system for accurately tracking head-pose and eye positions using a low resolution webcam. However they determine a point of gaze using a simple regression mapping of eye-features, requiring user calibration.

Recent research has explored gaze tracking for portable devices. Holland and Komogortsev [7] developed an integrated neural-network gaze tracker for a tablet. Though their accuracy was comparable, frame rate was not close to real-time (< 1Hz) and they physically constrained devices in their evaluation. Nagamatsu et al. [14] constructed a phone-mounted gaze tracker, but used additional IR cameras and emitters. Others explored portable device gaze-interaction using tabletop [15] or head-mounted systems [16] – these do not consider the sensor limitations.

As a feasibility study we partially replicated Kumar et al's EyePassword [2] system for gaze-based authentication. They compared password entry speed and accuracy between traditional PC keyboard input and gaze-interaction. While it was implemented using a Tobii 1750 gaze tracking display [2], we adapted it for a tablet.

Chapter 2

Eye localization

Following image acquisition, the initial goal is to extract the eyes' *regions of interest* [ROI] – minimal sub-images containing the object's boundaries. This reduces the search-space for following stages.

Detecting eyes in an image is an example of the difficult *object recognition problem* faced in computer vision. It is challenging as it must account for high within-class variability expressed by the same eye under different conditions, and high between-class variability expressed by eyes belonging to different people. It must also be fast enough to run in real-time[1].

Precisely localizing the ROIs is important as future stages require sub-images to exhibit eye features predictably. By ensuring ROIs are positioned around the eye-centre (or pupil), as opposed to around an eye corner or eye-lid, we can make assumptions that simplify processing and improve efficiency.

We therefore employ a multi-stage approach described in this chapter (Figure 2.1). First we use fast and reliable cascade classifiers to extract coarse eye-ROIs from the full image frame. Then we combine two shape-based methods in a novel hybrid approach to re-position coarse ROIs as precisely as possible on the eye-centre.

2.1 Coarse eye localisation

We use *cascade classifiers* – special-purpose object detectors, to rapidly identify eyes in an image. I describe their use in the system, and explain domain-specific assumptions made to improve efficiency.

[1] We define *real-time* as being fast enough to cope with the camera frame rate, $\sim$15 fps for current mobile devices.

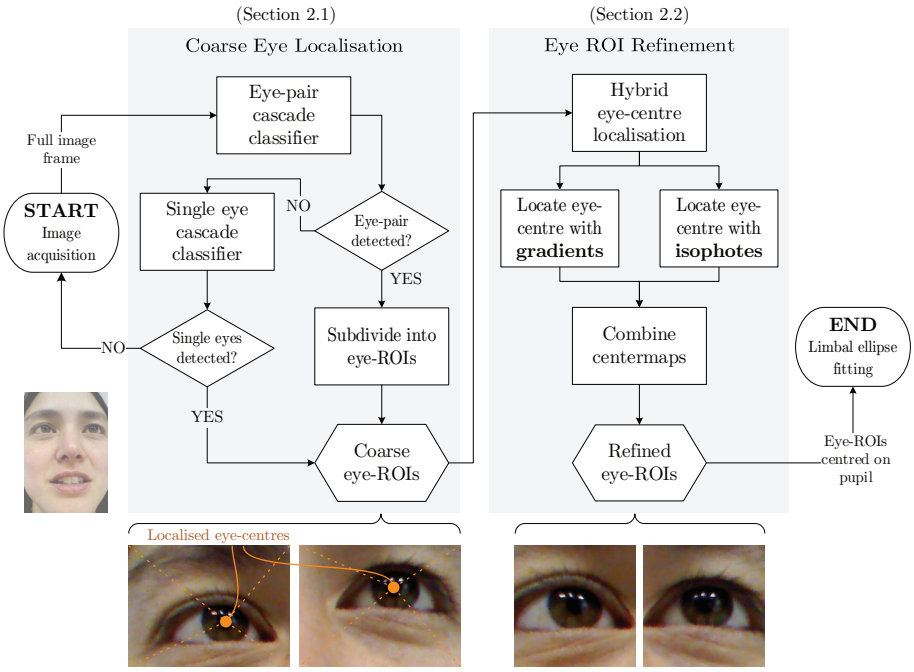


Figure 2.1: Eye localisation subsystem pipeline, showing typical coarse and refined eye-ROIs following eye-centre estimation.

2.1.1 Eye detection using cascade classifiers

Of all the face (or facial element) detectors currently in use, the *cascade of boosted classifiers*[2] introduced by Viola and Jones [17] is probably the most popular [18]. As shown in Figure 2.2, we slide a multi-scale window across the full-frame image and test with a cascade classifier if windowed candidate regions contain eyes.

The classifier is a *cascade* in the sense that it consists of several increasingly discriminating sub-classifier stages which are applied in sequence to a candidate region. In this way, non-object candidates are rapidly discarded, so computation time is efficiently distributed over more promising regions.

Each sub-classifier $h(\mathbf{x})$ is *boosted* in the sense that it is a weighted sum of *weak-learners* $h_j(\mathbf{x})$ – simple image functions that do not reveal much image information in isolation,

$$h(\mathbf{x}) = \sum_j \alpha_j h_j(\mathbf{x}) \quad \text{and} \quad h_j(\mathbf{x}) = \begin{cases} +1 & \text{if } f_j < k_j \\ -1 & \text{otherwise} \end{cases} \tag{2.1}$$

The weak learners $h_j(\mathbf{x})$ are simple threshold functions concering some feature response f_j and threshold k_j.

[2]Referred to as a *cascade classifier* in this project.

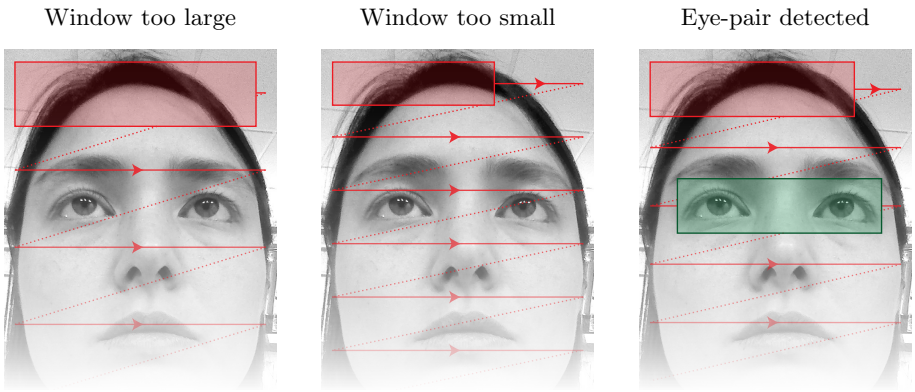
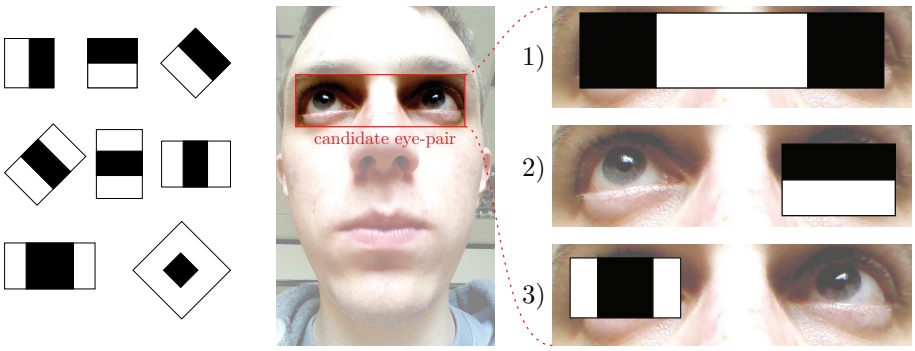


Figure 2.2: An eye-pair is detected in the top half of the image frame using a *multi-scale sliding window*. The window's path is shown in red.



(a) Haar wavelets **(b)** Haar-like features applied to a candidate region

Figure 2.3: Figure (b) shows Haar-like features being used to detect differences between the darker eyes and lighter nose-bridge or cheek skin areas

Figure 2.3 shows how features f_j measure intensity differences between rectangular regions in the candidate ROI, and are reminiscent of 2D Haar wavelets. They respond to edges, lines and center-surround features. These are very fast to compute once a summed area table has been generated [18], so are well-suited for real-time processing.

Choices of weights α_j and features f_j are made by training the classifier over thousands of positive and negative examples of the object – this can take days or weeks so publicly available implementations were used. As the classifier is trained to detect eyes' appearance in general, this step requires no user-calibration, and works over different lighting conditions.

Domain-specific assumptions

In this project, we make several assumptions about how the device is held to simplify eye detection and improve computational efficiency.

1. The user holds the device at close range, so images contain a large face.
2. The device is in *reverse-portrait* orientation – it is upright and the front-facing camera is at the bottom.
3. Yaw[3] between face and camera is low.
4. The face is approximately laterally centred in the image.

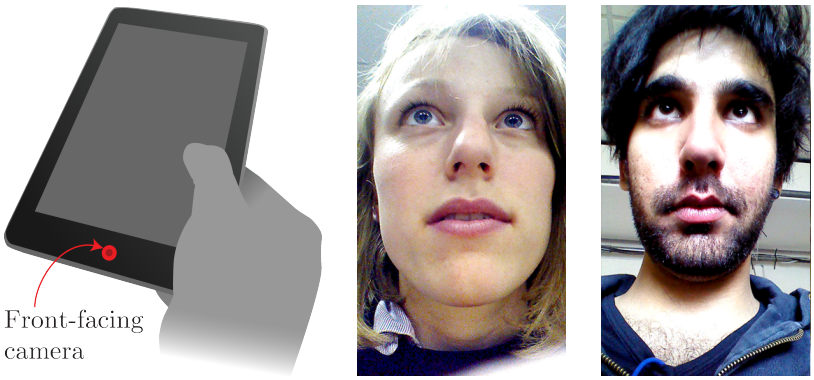


Figure 2.4: A device held in *reverse portrait* orientation, with some example front-facing camera images.

In this system, we first try to detect eyes by using an *eye-pair* cascade classifier, rotated at different angles to account for low in-plane face rotation (roll). This works in most situations, as both of the user's eyes (an eye-pair) are visible. The classifier is robust against some out-of-plane face motion (pitch), but we assume there is little yaw.

The eye-pair classifier is especially fast as the minimum size candidate regions are large. As we do not handle small eye-pairs (image fidelity is too low for them to be useful) the classifier is not invoked at small scales, saving computational cost.

Eye-pair detection occasionally fails in the presence of strong shadows across the face, or if the eye-pair extends beyond frame bounds. In this scenario we back-off to using single-eye detectors – a right-eye classifier on the left half of the image, and a left-eye classifier on the right half of the image[4].

2.1.2 Post-classifier candidate region rejection

The eye detectors may return multiple candidate regions, e.g. a pair of eyes and a pair of eyebrows (false-positive). In this situation we assume candidate regions with eyes have more apparent edges, e.g. eyelids and eye-features, while incorrect regions appear relatively uniform and feature-less. We measure edge strength between regions, and select the one with the most edges.

[3] *Yaw* is turning the head left/right, *pitch* is nodding up/down, *roll* is tilting left/right.
[4] Classifier eye-type has opposite handed-ness to the user's face as the projective image formation process flips the image.

Linear spatial filtering

Before discussing how edge strength is measured, this off-system section describes *linear spatial filtering* – a fundemental image processing operation used throughout the project.

Let our greyscale[5] image be described by a discrete two-dimensional spatial intensity function f, indexed by pixel coordinates (i, j). A *linear filter* is a neighbourhood operator, which determines its output $g(i, j)$ as a weighted sum of pixel values in the vicinity of a given pixel $f(i, j)$ [18].

$$g(i,j) = \sum_{k,l} f(i-k, j-l)\ h(k,l) \tag{2.2}$$

$$\equiv \sum_{k,l} f(k,l)\ h(i-k, j-l) \tag{2.3}$$

Weights are entries in a 2-D $k \times l$ *kernel* h, also known as the *point spread function*. The filtered image result is g. This is commonly epxressed as *convolution*,

$$g = f * h \tag{2.4}$$

Approximating relative blurriness using the LoG

An measure of relative edginess is found by convolving candidate regions with a *Laplacian of a Gaussian* kernel [LoG] $\nabla^2 G_\sigma(x, y)$, and comparing the responses (Figure 2.5).

The *Laplace operator* ∇^2 gives an undirected measure of the image's second spatial derivative, emphasising sharp edges and suppressing uniform areas. The Laplacian of an image $f(x, y)$ is given by

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.5}$$

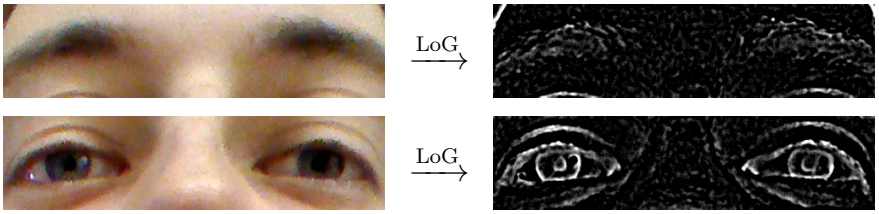
This is calculated by filtering with Laplacian kernel $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$, or similar.

However, the Laplace operator responds to image noise as well as edges, so we first pre-process the sub-image to suppress high-frequency noise. This is done by filtering with a Gaussian of width σ,

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.6}$$

which is discretely approximated as a finite-sized kernel. Convolution is associative so smoothing the image before taking its Laplacian is equivalent to

[5] Images with multiple colour channels (e.g. `[red, green, blue]`) are first transformed to greyscale before many image processing operations.



(a) Two candidate regions **(b)** Their LoG filtered images

Figure 2.5: Filtering candidate regions with a LoG gives a simple measure of relative blurriness. The feature-less eyebrows return lower response.

filtering once with a LoG kernel.

$$\nabla^2 \left[G_\sigma(x,y) * f(x,y) \right] = \underbrace{\left[\nabla^2 G_\sigma(x,y) \right]}_{\text{LoG}} * f(x,y) \tag{2.7}$$

A metric is calculated for each candidate region by taking the average intensity of the top pixels in the LoG filtered image. Uniform regions containing eyebrows have few strong edges, resulting in low filter response, while regions containing eyes have more strong edges, returning higher filter response. The sharpest edged candidate is determined to contain eyes.

2.1.3 Segmenting ROIs from eye regions

Once an eye-pair is detected, its rectangular sub-image is horizontally segmented into five disjoint regions, representing the sides of the head, the eyes, and the nose-bridge skin area (Figure 2.6). The non-eye regions are ignored, and the eye-regions are returned along with their image coordinates as the coarse eye-ROIs.

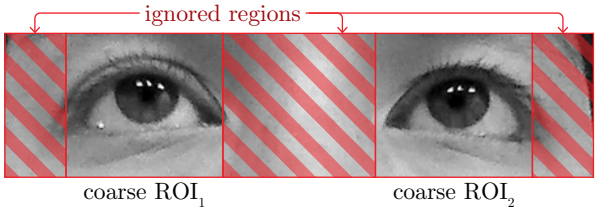


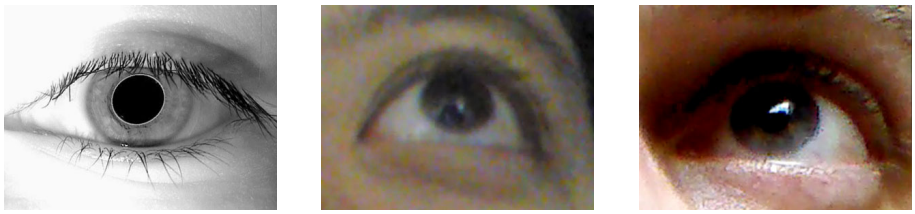
Figure 2.6: Coarse eye-ROI segmentation

The width ratios of these separate regions are constant. This is acceptable as higher precision eye localization is performed in section 2.2, with techniques that only require initialization near the eye. In the back-off scenario where two single eyes are separately detected, we return their individual sub-images as the eye-ROIs.

2.2 Eye-ROI refinement

Coarse ROI extraction using cascade classifiers is fast, reliable, and user-independent, but it is not precise. The next step is to refine eye-ROIs so they are correctly centred on the eye-centre (or pupil).

A common approach is to localise the pupil as the darkest patch in the image [4, 19, 20]. However, this is not always the case in visible light images as certain shadows may be darker, or the pupil may not be discernible.



(a) IR image from [20] **(b)** pupil not aparent **(c)** dark shadow

Figure 2.7: Active-IR can exploit high pupil contrast which may not be apparent in visible light images, as shown.

We precisely localise the eye-centre using a novel hybrid approach, combining the outputs of two *shape-based* algorithms to provide high precision and robustness in pathological lighting conditions. These exploit the predicted circular shape of eye-features, like the iris.

1. Timm and Barth's [21] *gradients* method infers the eye-centre at the intersection of image gradients.
2. Valenti and Gevers's [22] *isophotes* method infers the eye-centre at the middle of radially symmetric brightness patterns.

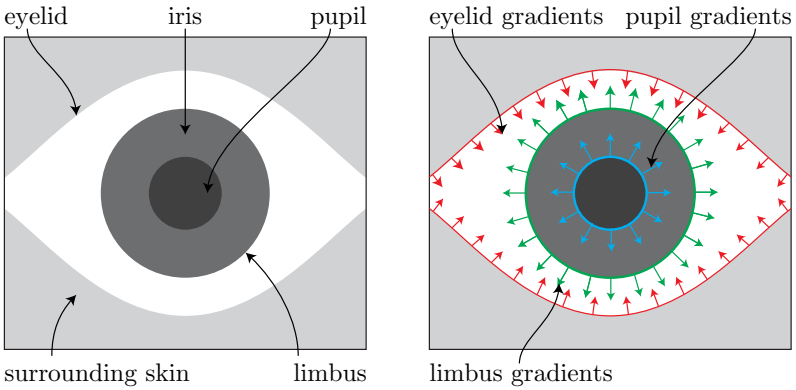
2.2.1 Eye-centre localisation using gradients

In images, the eye appears as a dark circular pattern. Timm and Barth [21] locate its centre in an image (spatial intensity function f), by analysing the *vector field of image gradients* ∇f. As shown in Figure 2.8b, the gradient vector at a pixel represents the 2-D direction in which the image function increases most rapidly.

We define a vector function of image gradients $\mathbf{g}$,

$$\begin{aligned} \mathbf{g}(x,y) = \nabla f(x,y) &= \frac{\partial f}{\partial x}\hat{\mathbf{x}} + \frac{\partial f}{\partial y}\hat{\mathbf{y}} \\ &= f_x\,\hat{\mathbf{x}} + f_y\,\hat{\mathbf{y}} \end{aligned} \tag{2.8}$$

Where $f_x = \frac{\partial f}{\partial x}$ and $f_y = \frac{\partial f}{\partial y}$ are the image gradients in the x and y dimensions, and $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are orthonormal basis vectors. We compute f_x and f_y by filtering with Sobel kernels [23].



(a) Toy coarse eye-ROI image **(b)** Gradient vectors at edges

Figure 2.8: Gradient vectors at image edges point from dark to light regions.

For simplicity, we write $\mathbf{g}_i$ for the gradient vector at point $\mathbf{x}_i$ in the image, where $\mathbf{g}_i = \mathbf{g}(x, y)_i$ and $\mathbf{x}_i = (x_i, y_i)$. Let $i \in [1, \ldots, N]$ for an image with N pixels. The centre of a singular dominant circular pattern is the point where most of these gradients intersect. This is because gradients at a (dark) circular boundary point towards that circle's centre.

Primary objective function

We infer circular pattern's centre $\mathbf{c}^*$ at the maximum of an objective function relating all possible centers $\mathbf{c}$, with normalized image gradients $\hat{\mathbf{g}}_i$ and displacement vectors $\mathbf{d}_i$.

For a good center estimate, the normalized displacement vector $\mathbf{d}_i$ from possible centre $\mathbf{c}$ to point $\mathbf{x}_i$ should have the same orientation as the gradient vector $\hat{\mathbf{g}}_i$ at point $\mathbf{x}_i$. We can measure this similarity by computing $\mathbf{d}_i \cdot \hat{\mathbf{g}}_i$, which is be greatest when the angle between $\mathbf{d}_i$ and $\hat{\mathbf{g}}_i$ is 0. Figure 2.9 demonstrates this using a simpler version of Figure 2.8b, showing gradients pointing away from dark regions.

We can then estimate optimal eye-centre $\mathbf{c}^*$ with,

$$\mathbf{c}^* = \arg\max_{\mathbf{c}} \left\{ \frac{1}{N} \sum_{i=1}^{N} (\mathbf{d}_i \cdot \hat{\mathbf{g}}_i) \right\} \tag{2.9}$$

$$\text{where} \quad \mathbf{d}_i = \frac{\mathbf{x}_i - \mathbf{c}}{|\mathbf{x}_i - \mathbf{c}|} \quad \text{and} \quad \hat{\mathbf{g}}_i = \frac{\mathbf{g}_i}{|\mathbf{g}_i|}$$

Dark patch weighting

In some conditions, $\mathbf{c}^*$ may be incorrectly chosen at a non-eye-centre local maximum corresponding to wrinkles, eye-corners, or eyelids. We therefore introduce a weight $w_{\mathbf{c}}$ for each possible centre $\mathbf{c}$ based on prior knowledge of

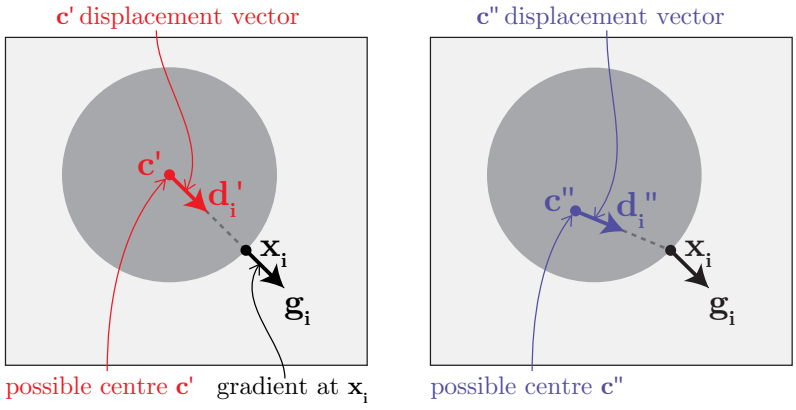
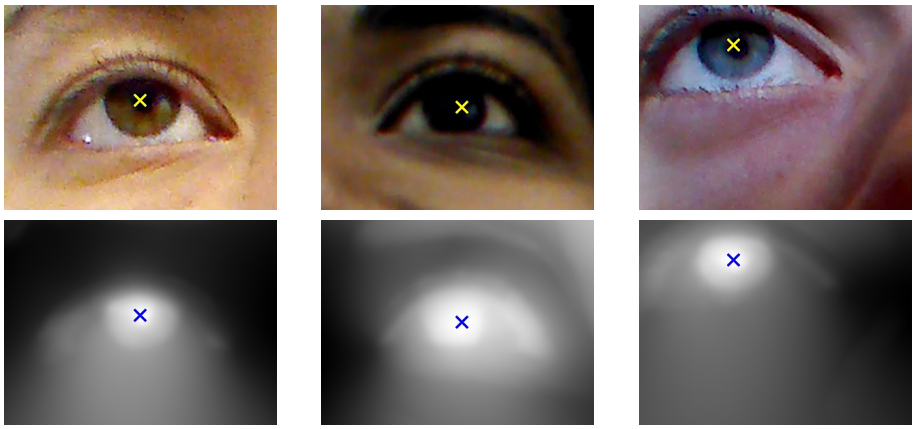


Figure 2.9: A minimal example of a dark eye surrounded by lighter sclera. Possible eye-centre $\mathbf{c}'$ scores higher than $\mathbf{c}''$ as $(\mathbf{d}_i' \cdot \mathbf{g}_i) > (\mathbf{d}_i'' \cdot \mathbf{g}_i)$.



(a) Optimum example **(b)** Pupil not visible **(c)** Coarse ROI offset

Figure 2.10: Three gradients method centre-maps showing estimated eye-centres in coarse eye-ROIs. (b) and (c) represent difficult conditions – zero pupil contrast and poor coarse ROI positioning.

the eye – the eye-centre (pupil) often appears darker than incorrect possible centres. So $w_{\mathbf{c}} = 1 - f(c_x, c_y)$ – the intensity in the inverted image[6] $(1 - f)$ at possible centre $\mathbf{c}$.

Therefore we adjust our objective function (Figure 2.9), to give higher weight to darker possible centres.

$$\mathbf{c}^* = \arg\max_{\mathbf{c}} \left\{ \frac{1}{N} w_{\mathbf{c}} \sum_{i=1}^{N} (\mathbf{d}_i \cdot \hat{\mathbf{g}}_i) \right\} \tag{2.10}$$

As shown in Figure 2.10, this objective function can be accumulated into a *centre-map*, showing the possible centre likelihood of each pixel.

[6]Intensity is 1 in white areas, and 0 in black areas.

Computational cost

We compute the centre-map by looping over every possible centre, calculating objective function (Equation 2.10) for the entire image at each pixel. This is computationally intensive – $\mathcal{O}(N^2)$ for an N pixel image.

We alleviate this by thresholding the image gradients so any with magnitude below a minimum value are ignored. We also resize the eye-ROI sub-image to a smaller fixed size F ($F < N$), scaling it by $\frac{F}{N}$. Precision is sacrificed for speed, but future stages can handle slight innacuracies in refined eye-ROIs.

2.2.2 Eye-centre localisation using isophotes

The eye can appear as radially symmetric brightness patterns in the image. Valenti & Gevers [22] use *isophotes* to infer the centre of these patterns. Isophotes are contours (or curves) in an image that connect pixels of equal intensity.

The general approach is that each pixel votes with the point at the centre of the circle that best approximates it. The most voted for point in the image is the eye-centre.

Isophote curvature

The *curvature* κ of a curve C at point P measures how fast its tangent vector rotates. We can reason about curvature geometrically. The curvature of of a straight line is zero, and the curvature of a circle of radius R is low when R is large – it locally appears like a line, and high when R is small – the tangent quickly deviates. So $\kappa = \frac{1}{R}$, the reciprocal of the radius.

For point P on any non-straight curve C, there is a unique circle which most closely approximates the curve at that point – the *osculating circle*. Similarly to how the tangent is the best approximating line at P, the osculating circle is the best approximating circle.

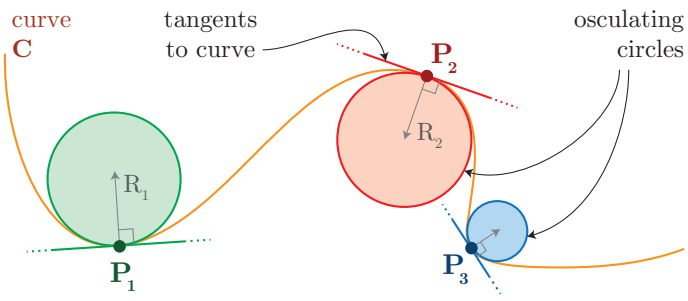
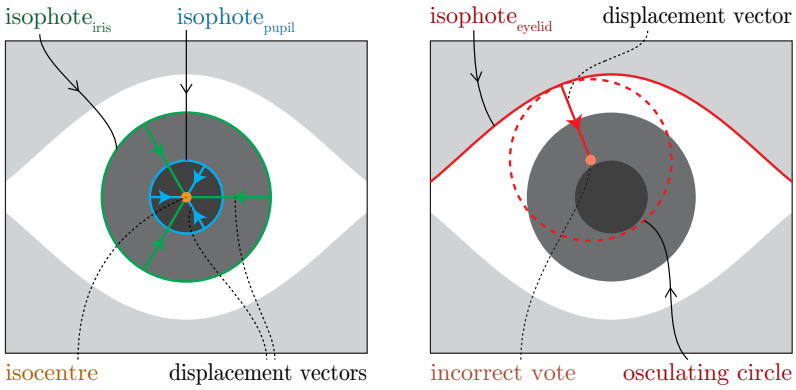


Figure 2.11: Osculating circles at several points along curve C.



(a) Correct isocentre votes **(b)** An incorrect vote

Figure 2.12: Isophote contours corresponding to iris and pupil edges vote on the correct eye-centre. However, the eyelid's isocentre is incorrect.

So the curvature κ at a point on an isophote contour is the curvature of its osculating circle at that point. This is calculated as[7],

$$\kappa = -\frac{f_y^2\, f_{xx} - 2 f_x\, f_{xy}\, f_y + f_x^2\, f_{yy}}{\left(f_x^2 + f_y^2\right)^{\frac{3}{2}}} \tag{2.11}$$

Where f_x and f_y are first-order image derivatives in the x and y dimensions, f_{xy} is the first-order derivative in x and then y, and f_{xx} and f_{yy} are second-order derivatives in x and y. These are approximated by filtering with an extended Sobel operator [23].

Isocenters

For each pixel in an isophote contour, we vote with the centre of its osculating circle – its *isocentre*. For this we calculate the displacement vector $\mathbf{d}(x, y)$ from that pixel at point (x, y) to its isocentre, as shown in Figure 2.12.

The magnitude of $\mathbf{d}(x, y)$ is the osculating circle radius $\frac{1}{\kappa}$, and its orientation matches the normalized image gradient at that point $\hat{\mathbf{g}}(x, y)$. So,

$$\mathbf{d}(x, y) = \frac{1}{\kappa}\, \hat{\mathbf{g}}(x, y) = \frac{1}{\kappa}\, \frac{\mathbf{g}(x, y)}{|\mathbf{g}(x, y)|} \tag{2.12}$$

[7] See original work by Valenti and Gevers [22] for a full derivation.

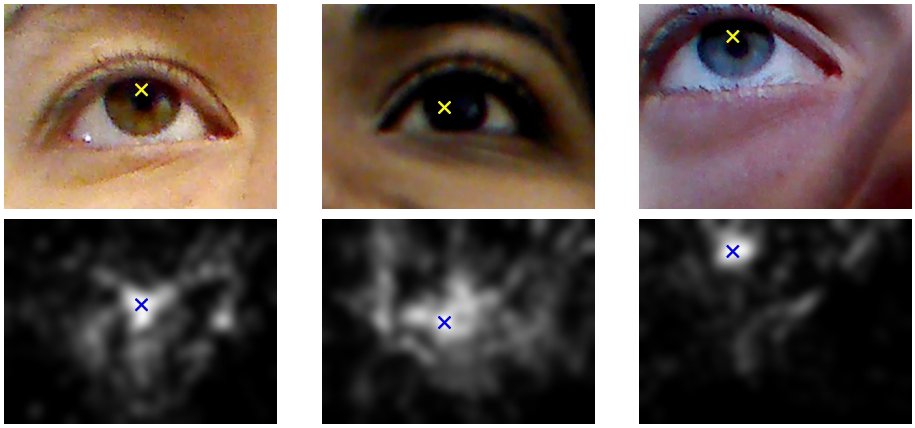


Figure 2.13: Corresonding isophote centre-maps for Figure 2.10. Noisy patches of outliers are apparent, particularly for the dark skin example.

Subsituting Equations 2.8 and 2.11 into Equation 2.12 lets us determine displacement vectors as

$$\begin{aligned}\mathbf{d}(x,y) &= (f_x\,\hat{\mathbf{x}} + f_y\,\hat{\mathbf{y}})\,\frac{1}{\kappa\,\left(f_x^2+f_y^2\right)^{\frac{1}{2}}} \\ &= (f_x\,\hat{\mathbf{x}} + f_y\,\hat{\mathbf{y}})\,\frac{f_x^2+f_y^2}{f_y^2\,f_{xx} - 2f_x\,f_{xy}\,f_y + f_x^2\,f_{yy}}\end{aligned} \tag{2.13}$$

Each pixel $f(x,y)$ votes for its isocentre at $(x,y)+\mathbf{d}(x,y)$. Similar to the gradients method, we accumulate isocentre votes in a centre-map (Figure 2.13).

Curvedness weighting

A limitation of this approach is that if we allow each pixel to vote equally, our results will be meaningless. This is because edges in real images are not always isophotes, and many isophotes correspond to non-edges, e.g. flat surfaces shaded by lighting conditions.

To prevent non-edge isocentres corrupting our centre-map, we weight each pixel's vote with its *curvedness* c, which measures of how much a region deviates from flatness [22].

$$c = \sqrt{f_{xx}^2 + f_{xy}^2 + f_{xx}^2} \tag{2.14}$$

Figure 2.14 shows this operator responds highly to curved edges, but little for flat regions. By weighting each isocentre with its generating isophote curvedness, we achieve a better estimate of the centre of a strong-edged circular pattern.

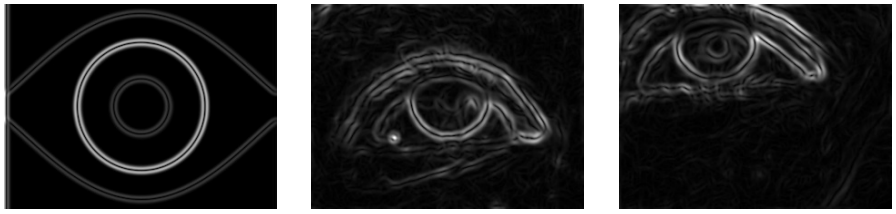


Figure 2.14: Curvedness of the toy image and real examples.

Computational cost

While the gradients method loops over image pixels as a possible eye-centres, this approach loops over pixels in isophotes and lets each one vote once for a possible eye-centre. This is less computationally intensive – $\mathcal{O}(N)$ for an N pixel image.

2.2.3 Center-map combination

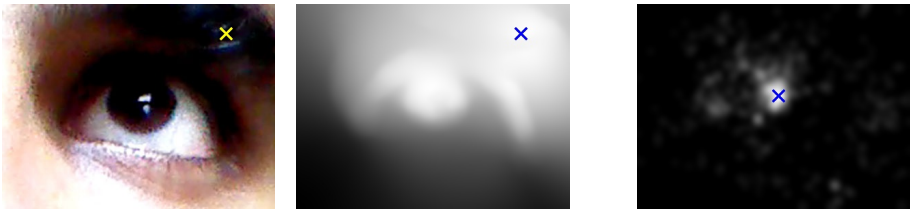
Relative performance

Though faster, we found the isophotes approach less precise than the gradients method. This is partly because each pixel votes only once, so if noise locally perturbs an isophote's curvature it will vote poorly. This is alleviated by smoothing the isophote centre-map, so each vote is spread amongst its neighbours.

An inherent limitation of the isophote approach is that pixels vote at the centre of osculating *circles*. This gaze tracking system relies on *elliptical* limbuses, and osculating circles at limbus isophotes do not coincide with its elliptical centre. Though this suggests that we might always prefer the gradients method, there are situations where it fails.

Pathological conditions for the gradients method

In certain lighting conditions and for eyes of a particular shape, shadows may fall over the eye-area which cause the gradients method to continuously fail. As shown in Figure 2.15, this occurs when the face is illuminated from above, and the eyes are *deep set*, positioned further into the skull making the brow line more prominent.



(a) Incorrect eye-centre **(b)** Grads. centre-map **(c)** Isophotes centre-map

Figure 2.15: The gradients method incorrectly chooses the dark shadowed area under the brow.

$\alpha\,\text{CMAP}_{grad}$ $\beta\,\text{CMAP}_{iso}$ CMAP_{joint}

Figure 2.16: The incorrect eye-centre choice from Figure 2.15 has been rectified.

Large, dark shadows can appear around the eyelids or brow. These occlude image features, and score highly after darkness weighting. The correct eye-centre also scores highly, but a *tie-breaker* is required to improve reliability. As the isophotes method is unaffected by linear lighting changes, we use it to assist the gradients method in choosing the correct eye-centre. Timm and Barth [21] simply discard centre-estimates near the ROI edges. This is unsuitable, as we must correctly localise offset eye-centres like Figure 2.10c.

Weighted sum of centre-maps

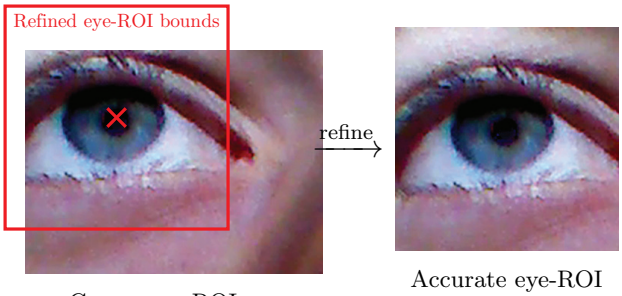
We weight and sum the centre-maps for a joint eye-centre estimate. For gradients centre-map CMAP_{grad} and isophotes centre-map CMAP_{iso},

$$\text{CMAP}_{joint} = \alpha\,\text{CMAP}_{grad} + \beta\,\text{CMAP}_{iso} \quad [\alpha + \beta = 1] \tag{2.15}$$

The eye-centre is localised at $\arg\max\{\text{CMAP}_{joint}\}$ (Figure 2.16). As the gradients method is more precise, $\alpha > \beta$; $(\alpha, \beta) = (0.8, 0.2)$ in this project.

2.2.4 Eye-ROI repositioning

We finally refine the eye-ROI around the eye-centre by redefining its bounding rectangle. The limbus is somewhat circular and fits in a square ROI, so we equalise eye-ROI dimensions.



Coarse eye-ROI

Accurate eye-ROI

The eye-ROI now exhibits eye-features at predictable positions and scales, so is ready for limbus extraction.

Chapter 3

Limbal ellipse fitting

After an eye-ROI has been extracted from the image frame, the next step is to analyse its sub-image and determine an ellipse that fits the limbal boundary. This requires *feature detection* and *model fitting*. Feature detection localises image *feature points* – pixels that carry important semantic associations [18]. Model fitting then simplifies the data for higher level processing.

Feature-detection is particularly challenging for low quality, visible light cameras. Poor lighting can mean features that we would like to use will not be visible, e.g. the pupil. Precise model fitting is difficult when low-resolution images are perturbed by noise, as few pixels belong to the limbus. Many approaches [1, 24] extract limbal boundaries with the Canny edge detector [25], but this is ineffective for the soft boundaries of light coloured irises.

In this chapter I present the limbal ellipse fitting subsystem (Figure 3.1). I first describe how a set of potential limbus feature points are collected using radial image gradients, and explain specular reflection preprocessing. I then describe how we remove erroneous points using eyelid pose, and finally present the algorithm for robustly fitting an ellipse to the remaining data.

3.1 Identifying potential limbus points

Following eye-ROI refinement, we assume the centre of the ROI is a good estimate of the pupil, and the ratio of eye-feature sizes to ROI size is roughly constant. This allows the application of a *search domain*, within which we expect to find limbus edge points. Like the iris, the search domain is doughnut-shaped with minimum radius s_{min} and maximum radius s_{max}.

We detect the limbal boundary within this domain by analysing the image's *radial derivative*. This can be used to detect radial edges, like the limbus, in the same way a directional derivative can detect horizontal or vertical edges. We can calculate this using *directional derivatives* $\nabla_{\hat{\mathbf{u}}} = \frac{\partial}{\partial \hat{\mathbf{u}}}$.

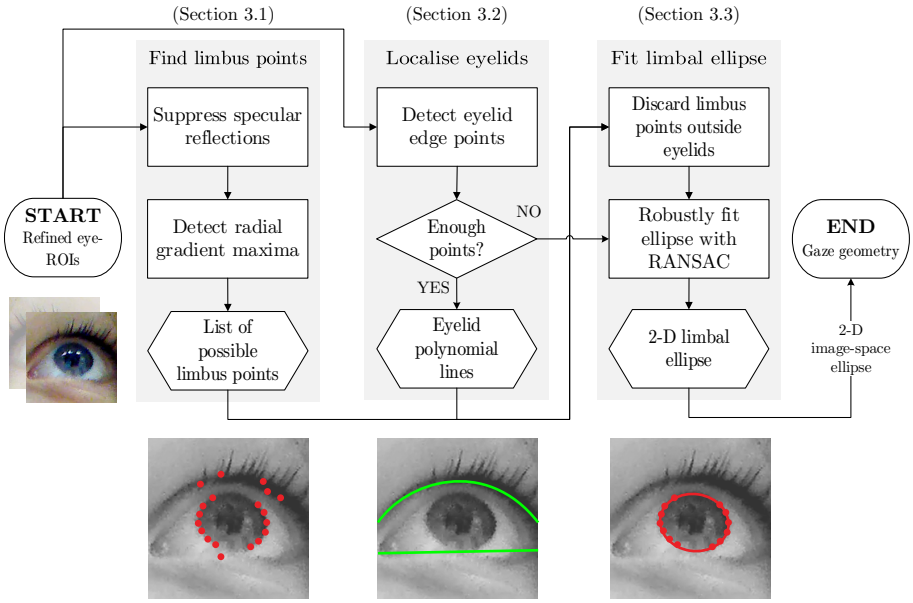


Figure 3.1: Limbal ellipse fitting subsystem using one eye-ROI as an example.

Unlike Canny-based techniques [1, 24], our approach does not use pre-defined thresholds. This makes it robust under a range of eye-appearances, both between users and lighting conditions. It also caters for out-of-focus images, as it does not require a strong edge.

3.1.1 Image derivatives along rays

This is equivalent to sampling the image along rays from the eye-centre $\mathbf{e_c}$, with directions $\hat{\mathbf{u}}(\theta) = (\cos\theta, \sin\theta)$, and then finding the maximum of that sample's 1-D derivative (Figure 3.2).

The sample row of pixels hit by each ray at θ, $\theta \in [0, 2\pi]$ is given by

$$\text{row}_\theta(r) = f\left[\,\mathbf{e_c} + r\,\hat{\mathbf{u}}(\theta)\,\right] \quad \forall\, s_{min} < r < s_{max} \tag{3.1}$$

The distance r_θ^{limb} along a ray of the possible limbus edge point is given at the maximum of its derivative, so we can determine the 2-D position of a possible limbus point $\mathbf{l}(\theta)$.

$$r_\theta^{\text{limb}} = \arg\max_r \left\{ \frac{d}{dr}\,\text{row}_\theta(r) \right\} \tag{3.2}$$

$$\mathbf{l}(\theta) = \mathbf{e_c} + r_\theta^{\text{limb}}\,\hat{\mathbf{u}}(\theta) \quad \forall\,\theta \in [0, 2\pi] \tag{3.3}$$

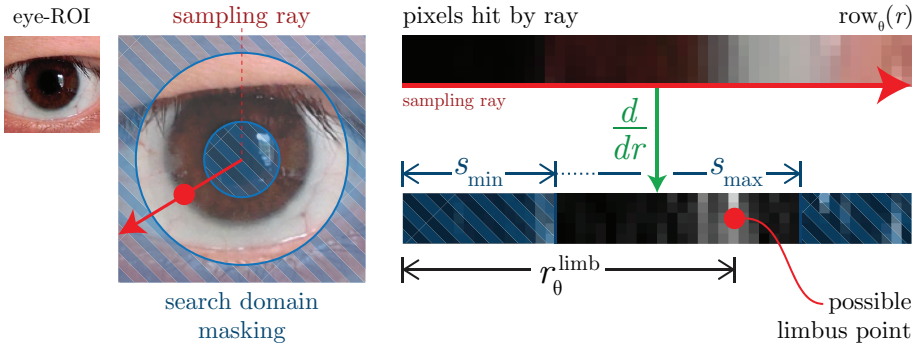


Figure 3.2: Calculating distance r_θ^{limb} , and thereby Determing a possible limbus point along a sampling ray.

3.1.2 Limbus detection in the polar domain

Sampling and filtering an image along arbitrarily oriented rays is computationally expensive, so we improve efficiency by taking the image's *polar transform* and filtering once.

This transform maps computes the equivalent polar coordinates (r, θ) for each Cartesian coordinate pixel at (x, y). Figure 3.3 shows these polar coordinate pixels plotted in an image.

$$(x,\ y) \xmapsto{\text{polar}} (r\ \cos\theta,\ r\ \sin\theta) \tag{3.4}$$

$$r = \sqrt{x^2 + y^2}, \quad \theta = \tan^{-1}(y, x) \tag{3.5}$$

This unwraps the eye-ROI image about its centre, so circular image features appear horizontally straight. We assume the radial limbus edge is close enough to circular so it appears approximately horizontal in the polar image.

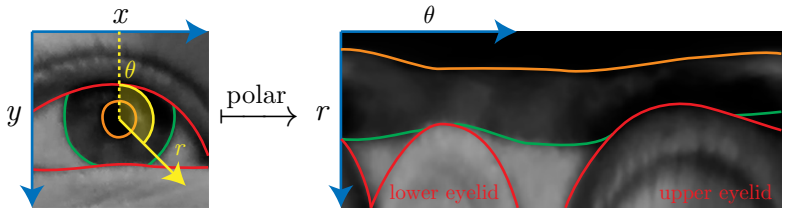
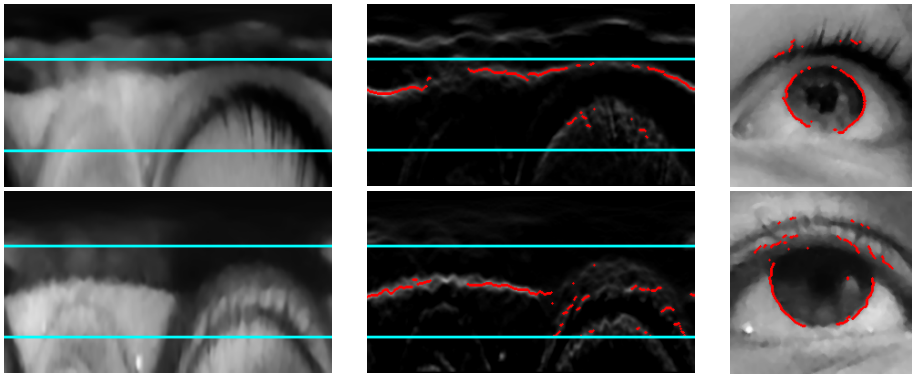


Figure 3.3: Cartesian/polar edge correspondence

Figure 3.4 shows how we then filter once to obtain the polar image's first-order vertical derivative[1] $\frac{\partial}{\partial r}$. To detect the now-horizontal limbus edge we take the maximum of each filtered column of pixels as a possible limbus point. Masking out the search domain is trivial as the top s_{min} and bottom s_{max} pixels are ignored.

[1] This is the Cartesian image's radial derivative



(a) polar image f_{pol} **(b)** limbus points in $\frac{\partial}{\partial r} f_{pol}$ **(c)** remapped

Figure 3.4: Detecting potential limbus points (red) within a search domain (blue).

The polar coordinates of the possible limbus points undergo a reverse transfrom to calculate their Cartesian coordinates.

Fixed size polar transform

Computing a Cartesian to polar coordinates mapping is computationally intensive as it involves $\mathcal{O}(N^2)$ trigonometric operations for an image of size N. To improve efficiency, this mapping is calculated only once for a fixed size P, and we scale the eye-ROI image by $\frac{P}{N}$. This avoids recomputing a mapping for every frame, and replaces it by a cheaper scaling operation.

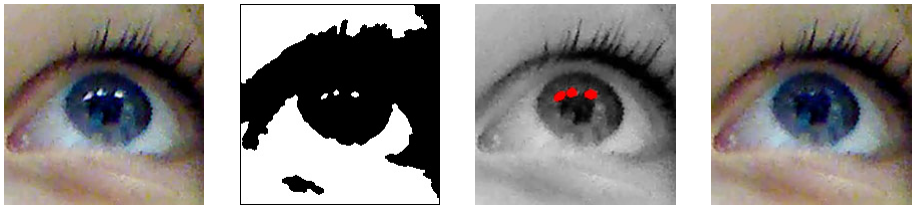
Early eyelid avoidance

As might be expected, the probability of an eyelid overlapping the limbus is greatest at angles of $\frac{\pi}{2}$ (upper eyelid) and $\frac{3\pi}{2}$ (lower eyelid) [26]. To avoid mis-classifiying eyelid edge points as limbus edge points, we ignore maxima between the ranges $\left[\frac{\pi}{2} \pm \psi\right]$ and $\left[\frac{3\pi}{2} \pm \psi\right]$[2]. This is apparent as gaps between points in Figure 3.4.

3.1.3 Suppression of specular reflections

The outside surface of the cornea is smooth and coated with an thin film of tear fluid which makes it reflective [27]. Nearby light sources like computer monitors or windows can cause specular reflections (*specularities*) which confound limbus edge detection. This happens as a iris/reflection boundary can locally appears like an iris/sclera boundary. Therefore, an important pre-processing step is to suppress these reflections.

[2] ψ is chosen experimentally – 20° in this project.



(a) input eye-ROI **(b)** thresholded **(c)** specularities **(d)** inpainted

Figure 3.5: Removing multiple specularities from a refined eye-ROI.

Specularity detection

We threshold out the darkest half[3] of the image to obtain a set of possible specularities (Figure 3.5b). These appear in the thresholded image as *connected components* of pixels. We identify specularities as small connected components that do not touch ROI edges (Figure 3.5c). Unfortunately, if one invades the limbal boundary it will remain undetected.

Specularity inpainting

Once specularities are localised, we must ensure they are not misclassified as the iris boundary. We could pass a mask through future processing steps, but this increases system complexity.

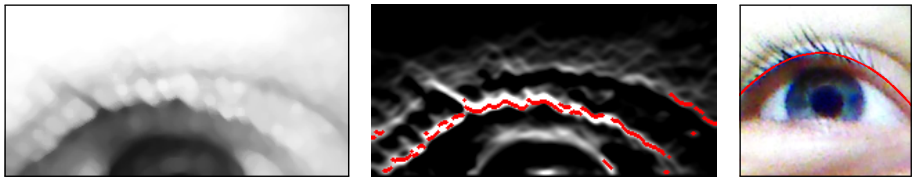
Instead we modify the ROI image itself using *inpainting* – a technique for reconstructing small image regions. This re-colours a group of pixels by propagating image information from around its boundaries in a perceptually plausible manner (Figure 3.5d). A public implementation of Telea's technique [28] is used (subsection 5.1.1). As we only ever inpaint small regions this is not computationally expensive.

3.2 Eyelid localisation

Unless the user opens their eyes extremely wide, it is likely that the set of possible limbus points contains some that belong to eyelids or surrounding skin, as seen in Figure 3.4. Before we fit the ellipse, we try to remove as many erroneous data points as possible by localising the eyelids, and removing exterior points.

Eyelid localisation is challenging due to their irregular shape. A person's eyelids appear different when they look up or down, and eye-features present in one pose may not be in another. Similarly to limbus extraction, our approach finds eyelid edge feature points and fits a model to them.

[3] In other approaches the threshold is usually higher, but this is unreliable for visible light images as skin or sclera may appear brighter than specularities.



(a) flattened eyelashes **(b)** upper eyelid edge points **(c)** parabola

Figure 3.6: (a) and (b) represent the upper eyelid sub-ROI

3.2.1 Eyelid feature point detection

Eyelid edge points are detected in a similar manner to limbus edge points in the polar domain: we apply sub-ROIs and filter areas in which we expect to find eyelid features. We must first avoid eyelashes which can perturb the true eyelid edge. This is because sharp eyelash edges can locally appear as good eyelid edge contenders when analysing the image derivative.

We suppress these using a *morphological close* operation (Figure 3.6a). This operation is a *dilation* – each eyelash is thickened in the image, followed by an *eriosion* – image elements are thinned or shrunk. This closes gaps between eyelashes, flattening their appearance.

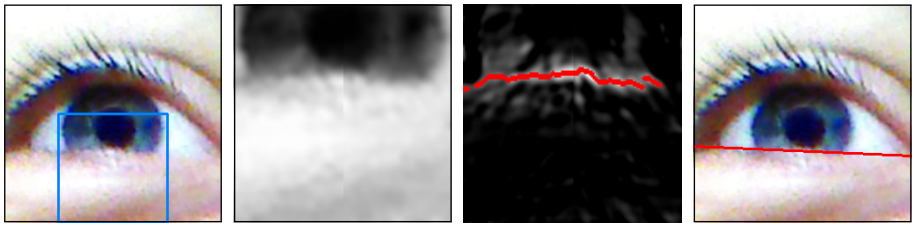
Upper eyelid point detection

The upper eyelid can be described in general as a shallow parabola. We search in the top half of the eye-ROI for where the lighter upper eyelid skin meets darker pixels belonging to shadows or the iris. As the eyelid parabola is shallow, upper eyelid edge points can be detected using a horizontal edge filter $\frac{\partial}{\partial y}$, and selecting the maxima.

Depending on user-specific eyelid anatomy, the upper eyelid may exhibit *eye-crease* features, also known as *double eyelids*. To avoid mis-classifying a crease as the eyelid/eye-ball boundary, we check the neighbourhood below each maxima for a similarly strong local maxima. If one exists, this is taken instead as the eyelid point.

Lower eyelid point detection

When the bottom eyelid occludes part of the iris, a strong horizontal edge is visible below the pupil. This is where the darker iris meets the lighter skin of the bottom eyelid.



(a) sub-ROI **(b)** window **(c)** edge points **(d)** horizontal line

Figure 3.7: Fitting a horizontal line to detected lower eyelid points.

We search in a small rectangular sub-ROI below the pupil (Figure 3.7a) for lower eyelid edge points. To avoid mis-classifying the bottom edge of the limbus as the iris/eyelid boundary, we filter with two opposite diagonal derivatives in the left and right halves of the sub-ROI. These respond to the near-horizontal iris/eyelid boundary, but are tailored to ignore limbal edges they are aligned with. When the bottom eyelid does not occlude the iris, few points are found and we do not fit a lower eyelid line.

3.2.2 Fitting a polynomial to eyelid points

Once possible edge points have been found for the upper and lower eyelids, a polynomial is robustly fit to them using least squares minimization (similar to subsection 3.3.1) and RANSAC (subsection 3.3.2). The upper eyelid is approximated as a parabola, and the lower eyelid as a straight line.

Any possible limbus points outside the enclosed region are then discarded. Figure 3.11 shows how erroneous limbus points are harmful to model fitting.

3.3 Robust ellipse fitting

Unfortunately, even after eyelid detection, our set of potential limbus points may contain outliers. This can happen if the ROI is mis-centred so part of the pupil/iris boundary lies within the search domain, or if eyelids are poorly localised. An ellipse can be fit to the entire set of points using a least-squares method, but will be adversely affected by these outliers.

I first describe the standard least-sqaures approach commonly used in gaze tracking systems, and then the generic Random Sample Consensus [RANSAC] algorithm [29] that leverages it. I finally explain some domain-specific modifications used in this project.

3.3.1 Direct least squares ellipse fitting

Ellipse fitting is an important task in computer vision, as an ellipse is the perspective projection of a circle. The most popular algorithm is Fitzgibbon's algebraic least squares approach which chooses ellipse parameters that minimizes the distance between data points and the ellipse [30].

A general conic can be represented in quadratic form

$$Q(x, y) = ax^2 + bxy + cy^2 + dx + ey + f = 0 \tag{3.6}$$

which can be re-expressed as a product of vectors

$$\mathbf{d}^\intercal \mathbf{z} = \left[x^2, xy, y^2, x, y, 1\right]^\intercal [a, b, c, d, e, f] = 0 \tag{3.7}$$

With a set of N points (x_n, y_n) to fit an ellipse to, vector $\mathbf{d}$ becomes the $N \times 6$ *design matrix* of variables $\mathbf{D}$,

$$\mathbf{D} = \begin{bmatrix} x_1^2 & x_1 y_1 & y_1^2 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N^2 & x_N y_N & y_N^2 & x_N & y_N & 1 \end{bmatrix} \tag{3.8}$$

We then choose parameters $\mathbf{z}$ such that the sum of the squared distances between each point and the conic is minimized[4],

$$\arg\min_{\mathbf{z}} \left\{ |\mathbf{D}\mathbf{z}|^2 \right\} \quad \text{and} \quad \mathbf{z} \neq \mathbf{0}_6 \tag{3.9}$$

Fitzgibbon reposes this problem as an ellipse-specific one by constraining the parameters of $\mathbf{z}$, so the conic it represents is an ellipse [30]. This is done by forcing an equality constraint on the *discrimiant*, so $4ac - b^2 = 1$.

[4] We ignore the trivial null solution $\mathbf{z} = \mathbf{0}_6$

This constraint can be expressed as matrix $\mathbf{C}$

$$\mathbf{z}^\intercal \mathbf{C} \mathbf{z} = 1 \quad \text{where} \quad \mathbf{C} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.10}$$

And the problem can now be reformulated as

$$\arg\min_{\mathbf{z}} \{\mathbf{z}^\intercal \mathbf{S} \mathbf{z}\} \quad \text{subject to} \quad \mathbf{z}^\intercal \mathbf{C} \mathbf{z} = 1 \tag{3.11}$$

Where $\mathbf{S} = \mathbf{D}^\intercal \mathbf{D}$ is the *scatter matrix*. We then apply Lagrange multiplier λ and differentiate to give the system

$$\mathbf{S}\mathbf{z} - \lambda \mathbf{C}\mathbf{z} = 0 \quad \text{and} \quad \mathbf{z}^\intercal \mathbf{C} \mathbf{z} = 1 \tag{3.12}$$

which is solved by considering the generalized eigenvectors of Equation 3.12.

Geometric parameters of an ellipse

The coefficients of $\mathbf{z}$ do not intuitively describe an ellipse's shape. It can also be expressed with geometric parameters, centre (e_x, e_y), orientation ϕ, and major and minor semiaxes $r_{\min}$ and $r_{\max}$ (Figure 3.8). There is a straight-forward mapping from the coefficient vector $\mathbf{z}$ to these parameters [4].

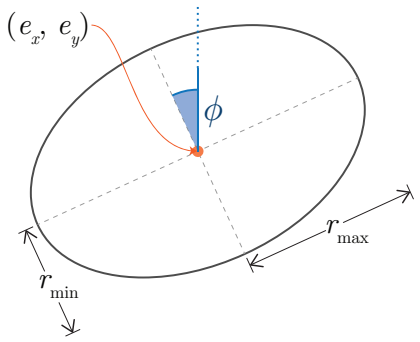


Figure 3.8: An ellipse's geometric parameters

3.3.2 Generic RANSAC model fitting

The main methods for robustly fitting ellipses to data are either *voting-based* or *searching-based*. Voting methods (e.g. Hough transform) are exhaustive, but computationally expensive [19]; while faster searching methods (e.g. RANSAC) sample data randomly so may be less accurate.

Algorithm 1: Generic RANSAC

input : data points
output: best-model

best-model, best-support $\leftarrow$ (NONE, $-\infty$)

repeat N **times** // N RANSAC iters
    initial inliers $\leftarrow$ `randomMinimalSample(data points)`
    model $\leftarrow$ `fitModel(initial inliers)`
    inliers $\leftarrow$ {point $\in$ data points | `error(point, model)` $< \epsilon$}
    support $\leftarrow$ `calculateSupport(model, inliers)`
    if support $>$ best-support **then**
        best-model $\leftarrow$ model
        best-support $\leftarrow$ support
    end if
end

return best-model

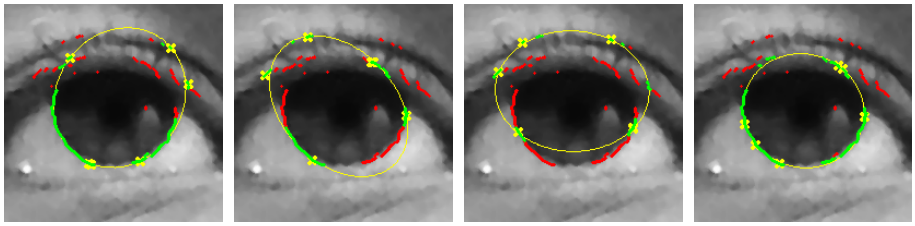


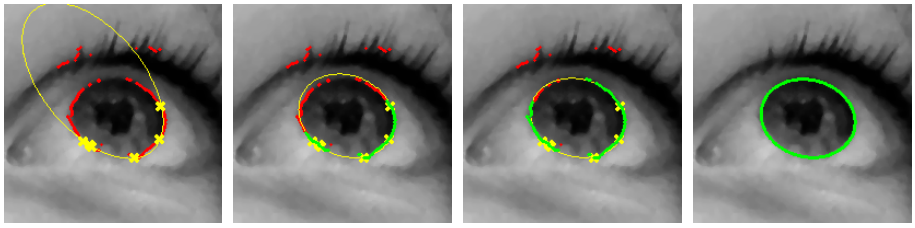
Figure 3.9: After four RANSAC iterations we choose the best model (the rightmost one). Yellow crosses: initial sample; green dots: inliers.

RANSAC is an iterative approach. It starts each iteration by *randomly* selecting a *minimal* sample from the data – these are the *initial inliers*. For a straight line, we would pick two random points from the dataset. It then fits a model to these initial inliers, often using a least squares approach, and calculates that model's *support*. This measures how good a fit it is to the entire dataset. A simple support function may return the number of *inliers* – data points sufficiently close to the model. Having completed several iterations, RANSAC selects the model with the highest support (Figure 3.9).

Its goal is to randomly select true inliers and fit the model, thus avoiding outlying data. It is non-deterministic as it only succeeds with a certain probability – this increases with the number of RANSAC iterations. Though the algorithm uses a least squares method internally, it is only applied to model inliers, so the overall fit is still robust. The generic RANSAC is described by algorithm 1.

3.3.3 Domain-specific modifications

For fitting limbal ellipses, this project emplpoys Świrski et al's [19] four modifications to the RANSAC algorithm. These were specifically chosen for gaze



(a) initial sample **(b)** refined once **(c)** refined twice **(d)** best ellipse

Figure 3.10: Model refinement can reach a good fit from a poor inital random sample. Yellow crosses: sample; green dots: inliers.

tracking ellipse fitting. Algorithm 2 describes our version of RANSAC, with modifications labelled inline.

1. **Early sample rejection** (ESR)
   If the image gradients and ellipse gradients do not agree for the initial samples, we quickly reject that candidate model.

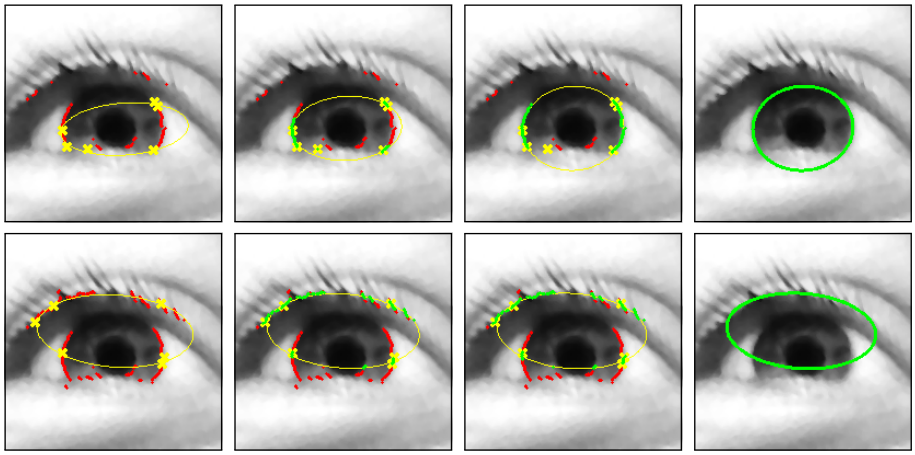
2. **Iterative Refinement** (ITR)
   The initial model is iteratively re-fit to newly detected inliers, improving fit precision and allowing fewer RANSAC iterations (Figure 3.10)

3. **Image-aware support function** (IAS)
   We wish to fit ellipses on strong image edges moving from a dark regions (iris) to a lighter one (sclera). Therefore we take image and ellipse gradients into account,

   $$\texttt{imageAwareSupport}(Q, f, \mathbf{r}) = \sum_{i}^{R} \frac{\nabla Q(r_i)}{|\nabla Q(r_i)|} \cdot \nabla f(r_i) \tag{3.13}$$

   where Q is an ellipse, f is our image function, and $\mathbf{r}$ are the R inliers. Large sets of inliers score highly as usual, but inliers are now weighted by how well their corresponding ellipse and image gradients agree.

4. **Early termination** [ET]
   As is commonly for RANSAC implementations [19], a well-fit model is returned early if it accounts for $\geq 95\%$ of the data points.



(a) initial sample **(b)** refined once **(c)** refined twice **(d)** best ellipse

Figure 3.11: The benefit of eyelid removal for RANSAC – the top row without eyelid points has a better ellipse fit than the bottom.

A fifth modification, **Coverage** [C], is used to avoid further processing of images where little of the limbus is visible. Before returning, we measure the best ellipse's coverage C – the percentage of its edge covered by inliers. If its coverage is low, this suggests that though it may be the best ellipse fit for the points given, it will not accurately represent the real world limbus.

$$\mathrm{C} = \frac{1}{360} \sum_{\theta=0^\circ}^{360^\circ} c(\theta)$$

$$\text{where} \quad c(\theta) = \begin{cases} 1 & \text{if } \exists\,(x,y) \in \text{inliers} \mid \tan(\theta) = \frac{e_y - y}{e_x - x} \\ 0 & \text{otherwise} \end{cases}$$

We now have robustly fit an ellipse representing the image-space limbus which is ready for back-projection.

Algorithm 2: Modified RANSAC

```
inputs : limbus points, image
output: best-ellipse
best-ellipse, best-support ← (NONE, −∞)
repeat N times                                                  // N RANSAC iters
    initial inliers ← randomSample(limbus points, 5)
    ellipse ← fitEllipse(initial inliers)
ESR if ∃ point ∈ limbus points | ∇ellipse(point) · ∇image(point) < 0 then
        continue
    end if
ITR repeat M times                                              // M refine iters
        ellipse ← fitEllipse(initial inliers)
        inliers ← {point ∈ limbus points | error(point, ellipse) < ϵ}
    end
IAS support ← imageAwareSupport(ellipse, image, inliers)
    if support > best-support then
        best-ellipse ← ellipse
        best-inliers ← inliers
        best-support ← support
    end if
 ET if |inliers| ≥ 0.95 |limbus points| then break
end
coverage ← calculateCoverage(best-ellipse, best-inliers)
if coverage < 30% or best-ellipse = NONE then
 C  raise NoGoodEllipseFit
else
    return best-ellipse
end if
```

Chapter 4

Gaze geometry

When the limbic circle is projected onto an image plane using perspective projection, it is drawn as an ellipse. In this chapter I describe how the 2-D limbal ellipse fit with RANSAC is *back-projected* to its generating 3-D circle, thus determining the gaze vector. Back-projection is the reversed process of image formation, where a 3-D scene is projected onto a 2-D image plane.

I first introduce the fundemental concepts of image projection, and then explain the two back-projection processes of

1. Localising the limbus position $\mathbf{c}$ in 3-D space
2. Calculating its 3-D circle's normal vector $\mathbf{n}$ – the optical axis

I finally describe how this data is smoothed over time to estimate the device on-screen gaze point. Figure 4.1 displays the subsystem pipeline.

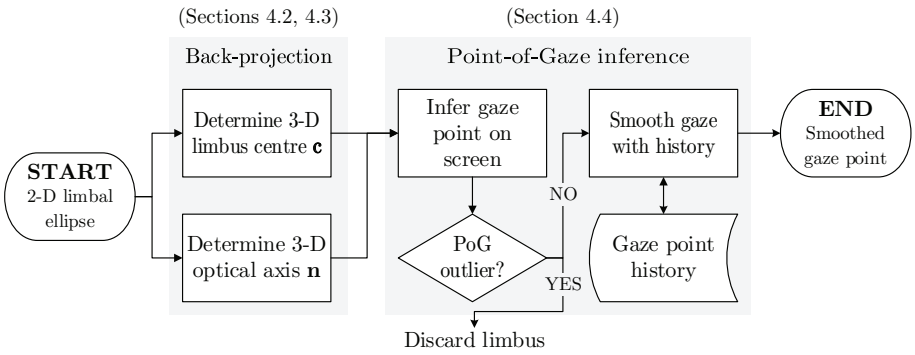


Figure 4.1: Gaze geometry subsystem architecture.

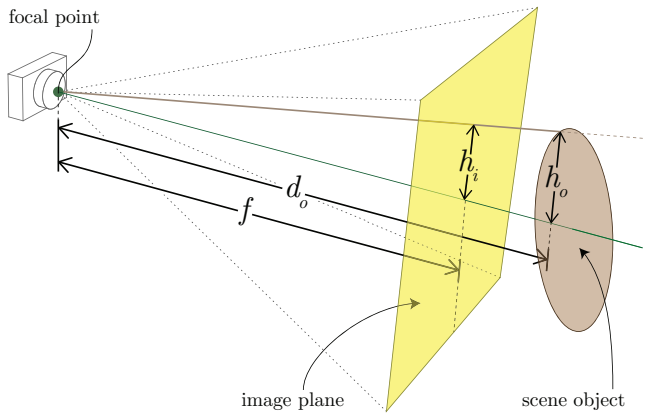


Figure 4.2: Basic pinhole camera model

4.1 The pinhole camera model

The *pinhole camera model* is an approximation often used in graphics and computer vision which considers a camera with an infinitesimal aperture – an ideal pinhole.

Images are formed when light reflecting off a 3-D scene passes through the pinhole aperture (also known as the *focal point*) and hits the image plane. For a portable device this image plane is the digital camera's image sensor, and for our eyes it is the retina.

In our model (Figure 4.2) we reposition the image plane in front of the aperture to obtain an identical, upright image. While physically impossible, it is mathematically equivalent to a post-aperture image plane model [23].

Using this model and the law of similar triangles, we derive a basic form of the projective equation:

$$h_i = f \frac{h_o}{d_o} \tag{4.1}$$

h_i	=	size of the object's projection in the image	(px)
h_o	=	size of the object in the 3-D scene	(mm)
d_o	=	focal point to object distance	(mm)
f	=	focal point to image plane distance – the *focal length*	(mm)

4.1.1 Extended projective equation

This basic projective equation can be extended to relate a 3-D point (X, Y, Z) in world coordinates (mm) with its projection (x, y) in image coordinates (px). For this we must consider the *principal point* (u_0, v_0) – the point at the intersection of the image plane with a line through the focal point orthogonal to the image plane; see Figure 4.3. For an ideal camera the principal point would be at the centre of the image plane, but in practice it is often misaligned by a few

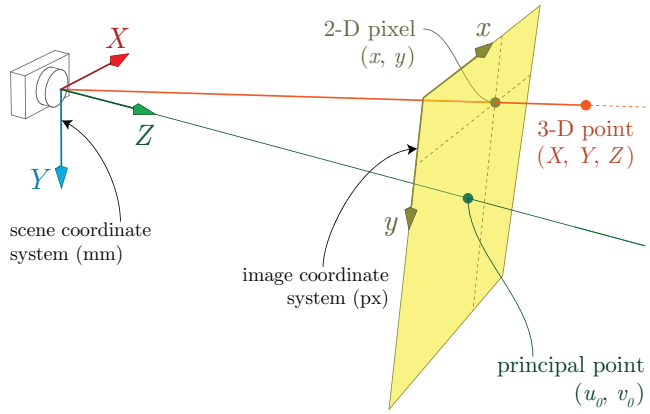


Figure 4.3: Extended camera model, relating a 3-D point with its 2-D projection.

pixels, depending on manufacturing precision [23].

Equation (4.1) projects 3-D point (X, Y, Z) onto $(f\frac{X}{Z}, f\frac{Y}{Z})$ in the 2-D image plane. To translate this point into pixels, we divide the 2-D point by the pixel width p_x and height p_y[1], so:

$$x = f\frac{X}{Z\,p_x} + u_0 \quad \text{and} \quad y = f\frac{Y}{Z\,p_y} + v_0 \tag{4.2}$$

By dividing the focal length f by the pixel dimensions, we can also express the focal length in terms of horizontal pixels $f_x = \frac{f}{p_x}$, or vertical pixels $f_y = \frac{f}{p_y}$. Thus we can write the complete projective equation:

$$x = f_x\frac{X}{Z} + u_0 \quad \text{and} \quad y = f_y\frac{Y}{Z} + v_0 \tag{4.3}$$

The projective equation (4.3) can be rewritten in matrix form using homogeneous coordinates:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\substack{\text{intrinsic} \\ \text{parameters } \mathbf{K}}} \underbrace{\begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix}}_{\substack{\text{extrinsic} \\ \text{parameters } [\mathbf{R}|\mathbf{T}]}} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.4}$$

The first and second matricies describe the *intrinsic* and *extrinsic* camera parameters respectively. The intrinsic parameters $\mathbf{K}$ describe the physical properties of the camera itself, and remain constant for a given system. These are determined using *camera calibration*, described in subsection 4.1.2.

The extrinsic parameters $[\mathbf{R}|\mathbf{T}]$ describe the camera's orientation in space, and is comprised of a 3-D rotation $\mathbf{R}$ and translation $\mathbf{T}$. This transformation

[1] In many cases, the pixels will be square-shaped, so $px = py$, but this depends on the image sensor design and camera manufacturing precision.

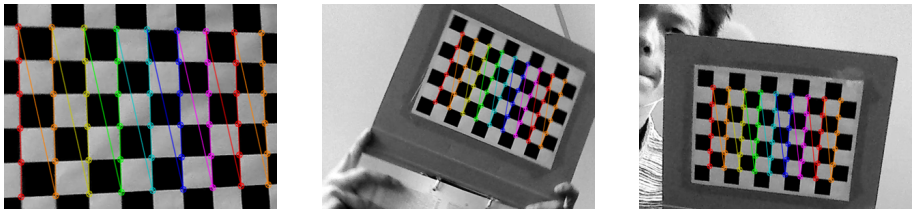


Figure 4.4: Example chessboard camera calibration images

maps 3-D scene points into the camera's reference frame. However, front-facing cameras in commodity portable devices generally have a permanently fixed position and orientation, so we can simplify the projection by equating the scene and camera reference frames:

$$[\mathbf{R}|\mathbf{T}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.5}$$

4.1.2 Camera calibration

Intrinsic parameters $\mathbf{K}$ are estimated using *camera calibration*. This process analyses a sequence of images containing sets of scene-points with known 3-D positions. By comparing the scene-points' projection in the 2-D image and their known 3-D positions, an optimization process can determine the optimal intrinsic parameters that explain these observations [23].

A pre-defined chessboard pattern (Figure 4.4) was used to generate 3-D scene points at the corners of each square. As the pattern is flat, we can assume it lies at the origin with $Z = 0$, and the X and Y axes aligned with the chessboard grid. Calibration is then carried out by recording a sequence of pattern images at different positions and angles.

4.2 Localising the 3-D limbus centre

We now use the projective equations (4.3) and (4.1) to determine the center $\mathbf{c} = [c_X, c_Y, c_Z]^\intercal$ of the 3-D limbal circle from its 2-D projection as an ellipse. Recall: an image-space ellipse can be desribed by its centre (e_x, e_y), orientation ϕ, and major and minor semiaxes $r_{\min}$ and $r_{\max}$ (subsection 3.3.1).

The first step is to calculate the z-axis distance c_Z between focal point and limbus center $\mathbf{c}$. We assume approximately-square pixel dimensions, $p_x \approx p_y$, and thus let $f_z = \frac{f_x + f_y}{2}$, where f_z is the focal length in pixels. As we know the real-world size of the limbus in millimetres[2] r_{limbus}, and its image projection

[2]Limbus size varies little [1] so we assume a constant $r_{\text{limbus}} = 6\text{mm}$

size in pixels $r_{\max}$, we can adapt equation (4.1),

$$r_{\max} = f_z \frac{r_{\text{limbus}}}{c_Z} \tag{4.6}$$

$$\text{so,} \quad c_Z = f_z \frac{r_{\text{limbus}}}{r_{\max}} \tag{4.7}$$

We now use equation (4.3), and substitute (e_x, e_y) for (x, y),

$$(e_x,\, e_y) = \left(f_x \frac{c_X}{c_Z} + u_0 \,,\; f_y \frac{c_Y}{c_Z} + v_0 \right) \tag{4.8}$$

$$\text{so,} \quad (c_X,\, c_Y) = \left(c_Z \frac{e_x - u_0}{f_x} \,,\; c_Z \frac{e_y - v_0}{f_y} \right) \tag{4.9}$$

Thus the complete back-projection from an image-space 2-D ellipse to real-world 3-D circle centre $\mathbf{c}$ in millimetres is:

$$\mathbf{c} = \left[c_Z \frac{e_x - u_0}{f_x} \,,\; c_Z \frac{e_y - v_0}{f_y}, c_Z \right]^\intercal \,, \quad c_Z = f_z \frac{r_{\text{limbus}}}{r_{\max}} \tag{4.10}$$

With this we calculate the 3-D position of the limbus $\mathbf{c}$, so we now need its optical axis $\mathbf{n}$ to determine gaze direction.

4.3 Determining the optical axis

Donder's law states that gaze direction uniquely determines eye oreintation, and this orientation is independent of the eye's previous positions [3]. Therefore if we can calculate the orientation of the eye, we can estimate the gaze direction. We use a simplified geometric eye model (subsection 1.1.1), equating the 3-D limbus plane and eye orientation. Though true gaze is better described by the visual axis we approximate it as the optical axis, thereby avoiding user-specific calibration procedures.

4.3.1 Circular cross-section of an elliptical cone

When the limbal circle is projected onto an image plane using perspective projection, it appears as an ellipse. Our goal to estimate the normal vector $\mathbf{n}$ to the limbal circle's supporting plane using its elliptical projection on the image plane.[3] Figure 4.5 describes the geometry behind our approach.

The ellipse Q fit using RANSAC (subsection 3.3.2) can be expressed as

$$Q(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F \tag{4.11}$$

where the ellipse is the isocontour at $Q(x, y) = 0$. This can be rewritten in

[3] See Chen et al. [31] for further description of this process.

matrix form with 3×3 symmetric matrix $\mathbf{Q}$ and column vector $\mathbf{x}$.

$$\mathbf{x}^\intercal \, \mathbf{Q} \, \mathbf{x} = \mathbf{0} \tag{4.12}$$

$$\text{where} \quad \mathbf{Q} = \begin{bmatrix} A & \frac{B}{2} & \frac{D}{2} \\ \frac{B}{2} & C & \frac{E}{2} \\ \frac{D}{2} & \frac{E}{2} & F \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

For simplicity we consider a camera coodinate system where the origin is the optical center, and the negative z-axis is the camera axis. The camera's z-axis focal length is f_z so the image plane is given by $z = -f_z$. A point on the ellipse in this image plane can be expressed by $\mathbf{x_e} = [x,\ y,\ -f_z]^\intercal$.

We can write:

$$\mathbf{x} = \mathbf{K} \, \mathbf{x_e} \quad \text{where} \quad \mathbf{K} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -f_z \end{bmatrix} \tag{4.13}$$

Then, by substituting equation (4.13) for $\mathbf{x}$ in (4.12):

$$(\mathbf{K} \, \mathbf{x_e})^\intercal \, \mathbf{Q} \, (\mathbf{K} \, \mathbf{x_e}) = \mathbf{x_e}^\intercal \, \mathbf{Q_e} \, \mathbf{x_e} = \mathbf{0} \tag{4.14}$$

$$\text{where} \quad \mathbf{Q_e} = \mathbf{K} \, \mathbf{Q} \, \mathbf{K} = \begin{bmatrix} A & \frac{B}{2} & \frac{-D}{2f_z} \\ \frac{B}{2} & C & \frac{-E}{2f_z} \\ \frac{-D}{2f_z} & \frac{-E}{2f_z} & \frac{F}{f_z^2} \end{bmatrix}$$

And $\mathbf{Q_e}$ represents the 3-D ellipse in the $z = -f_z$ image plane. We can form an *oblique* elliptical cone by projecting a bundle of straight line rays through the camera origin and boundary of ellipse $\mathbf{Q_e}$. An oblique cone's axis (line joining its vertex and base) is not perpendicular to its base plane.

This cone can also be intersected by another plane with which these rays intersect to form a circle in 3-D space. This circle's supporting plane is the base plane for an oblique circular cone.

Let $\mathbf{p}$ represent a point on the oblique elliptical cone, so

$$\mathbf{p} = h \, \mathbf{x_e} \tag{4.15}$$

By substituting equation (4.15) for $\mathbf{x_e}$ in (4.14) we can obtain the matrix representation of the cone using $\mathbf{Q_e}$

$$\mathbf{p}^\intercal \, \mathbf{Q_e} \, \mathbf{p} = h^2 (\mathbf{x_e}^\intercal \, \mathbf{Q_e} \, \mathbf{x_e}) = \mathbf{0} \tag{4.16}$$

As $\mathbf{Q_e}$ is a real symmetric matrix, it is diagonalizable, so can be expressed in terms of its normalized eigenvalues $\lambda_1, \lambda_2, \lambda_3$ and associated eigenvectors $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$ as

$$\mathbf{Q_e} = \mathbf{V} \, \mathbf{\Lambda} \, \mathbf{V}^\intercal \tag{4.17}$$

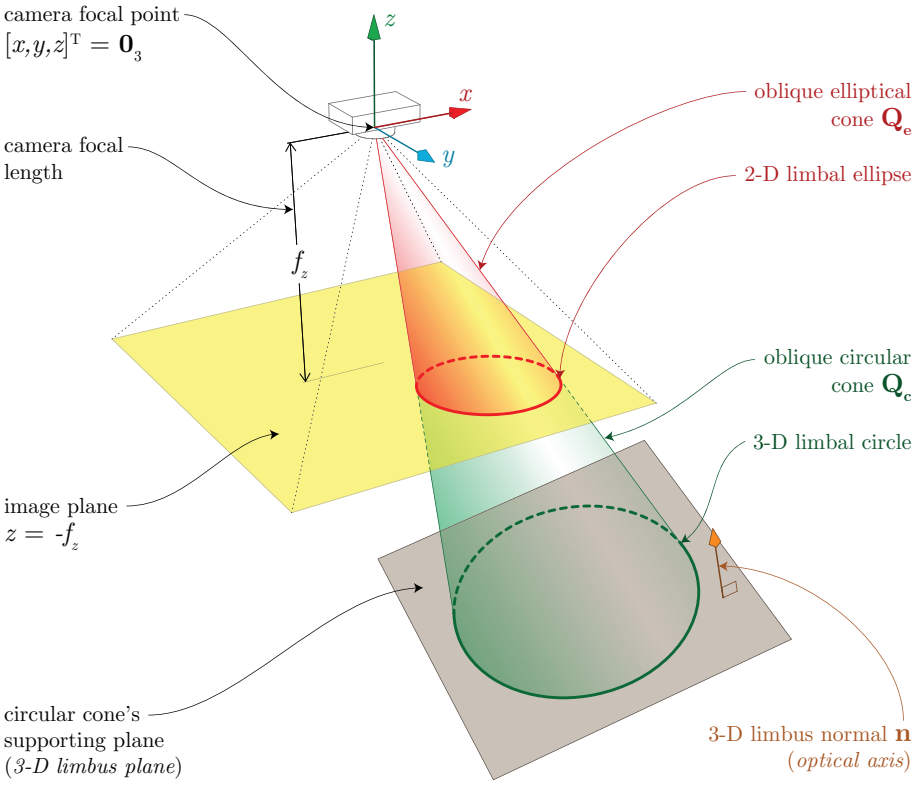


Figure 4.5: The geometric constructs used to calculate optical axis $\mathbf{n}$.

$$\text{where} \quad \mathbf{\Lambda} = diag(\lambda_1, \lambda_2, \lambda_3) = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

$$\text{and} \quad \mathbf{V} = [\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}] = \begin{bmatrix} v_{11} & v_{21} & v_{31} \\ v_{12} & v_{22} & v_{32} \\ v_{13} & v_{23} & v_{33} \end{bmatrix}$$

So $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues, and the columns of $\mathbf{V}$ are the eigenvectors.

For Sylvester's Law of Inertia to hold, the signature of $\mathbf{\Lambda}$ must match that of $\mathbf{Q_e}$, $(-,+,+)$ [4]. Therefore, one eigenvalue is negative, and the other two are positive. We enforce $\lambda_3 < 0$ and $\lambda_1 > \lambda_2 > 0$ by re-ordering our eigenvalues and corresponding eigenvectors accordingly.

Let us now consider the oblique circular cone incident with the oblique elliptical cone. We rotate the camera coordinate system to equate the z-axis with the normal to the 3-D circle's supporting plane. Let $\mathbf{p_c}$ be a point on the oblique circular cone. In this coordinate system, the oblique circular cone is characterized by

$$\mathbf{p_c}^{\mathsf{T}} \mathbf{Q_c} \mathbf{p_c} = \mathbf{0} \tag{4.18}$$

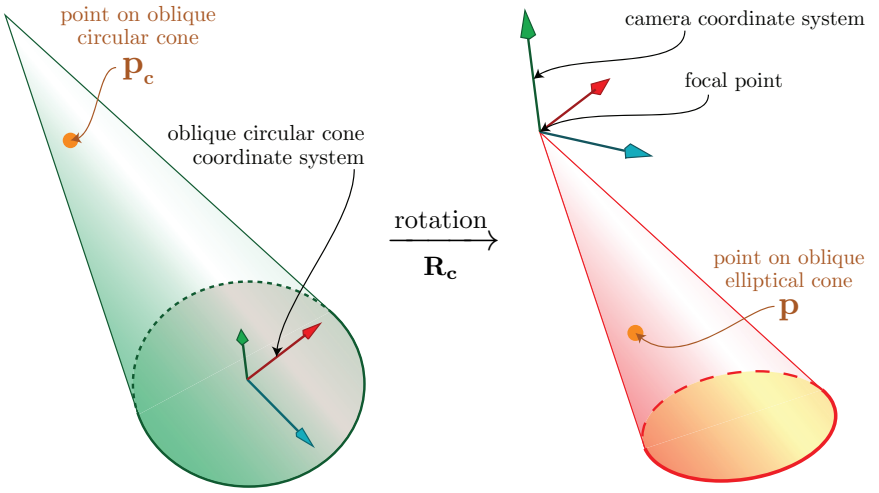


Figure 4.6: $\mathbf{p} = \mathbf{R_c}\,\mathbf{p_c}$ – rotating from the oblique cone's base plane coordinate system to the original camera coordinate system.

$$\text{where} \quad \mathbf{Q_c} = \begin{bmatrix} 1 & 0 & \frac{-x_0}{z_0} \\ 0 & 1 & \frac{-y_0}{z_0} \\ \frac{-x_0}{z_0} & \frac{-x_0}{z_0} & \frac{x_0^2+y_0^2-r^2}{z_0^2} \end{bmatrix} \quad \begin{cases} \text{circle centre } (x_0, y_0, z_0) \\ \text{circle radius } r \end{cases}$$

Rotation between coordinate systems

The matrices $\mathbf{Q_c}$ and $\mathbf{Q_e}$ describe the same cone's surface, so there exists a rotation matrix $\mathbf{R_c}$ to transform points $\mathbf{p_c}$ on the rotated coordinate system's oblique circular cone to points $\mathbf{p}$ on the original coordinate system's oblique elliptical cone (Figure 4.6).

$$\mathbf{p} = \mathbf{R_c}\,\mathbf{p_c} \tag{4.19}$$

By substituting equation (4.19) for $\mathbf{p}$ in (4.16), we can write

$$\mathbf{p}^\intercal\,\mathbf{Q_e}\,\mathbf{p} = (\mathbf{R_c}\,\mathbf{p_c})^\intercal\,\mathbf{Q_e}\,(\mathbf{R_c}\,\mathbf{p_c}) \tag{4.20}$$

Then, we can substitute equation (4.17) for $\mathbf{Q_e}$ in equation (4.20) and equate with (4.18) to obtain

$$(\mathbf{V}^\intercal\,\mathbf{R_c})^\intercal\,\mathbf{\Lambda}\,(\mathbf{V}^\intercal\,\mathbf{R_c}) = k\,\mathbf{Q_c} \tag{4.21}$$

As $k\,\mathbf{Q_c}$ represents the same cone as $\mathbf{Q_c}$ for $k \neq 0$.

$\mathbf{Q_e}$ is real symmetric so its eigenvector columns $\mathbf{V}$ will form an *orthogonal matrix*: $\mathbf{V}\,\mathbf{V}^\intercal = \mathbf{I}$. As $\mathbf{R_c}$ represents a rotation in 3-D Euclidian space, it too will be orthogonal: $\mathbf{R_c}\,\mathbf{R_c}^\intercal = \mathbf{I}$. We can then write

$$(\mathbf{V}^\intercal\,\mathbf{R_c})\,(\mathbf{V}^\intercal\,\mathbf{R_c})^\intercal = \mathbf{V}^\intercal\,(\mathbf{R_c}\,\mathbf{R_c}^\intercal)\,\mathbf{V} = \mathbf{I} \tag{4.22}$$

Solving equations (4.21) and (4.22) yields [31]

$$\mathbf{V}^\intercal \mathbf{R_c} = \begin{bmatrix} g\cos(\alpha) & S_1 g \sin(\alpha) & S_2 h \\ \sin(\alpha) & -S_1 g \cos(\alpha) & 0 \\ S_1 S_2 h \cos(\alpha) & S_2 h \sin(\alpha) & -S_1 g \end{bmatrix} \tag{4.23}$$

where α is a free variable, S_1 and S_2 are undetermined signs, and

$$g = \sqrt{\frac{\lambda_2 - \lambda_3}{\lambda_1 - \lambda_3}} \qquad h = \sqrt{\frac{\lambda_1 - \lambda_2}{\lambda_1 - \lambda_3}}$$

Normal to the circular cone supporting plane

The normal vector $\mathbf{n}$ to the limbal circle's supporting plane can be calculated by transforming $[0,\ 0,\ 1]^\intercal$ from the rotated coordinate system of the oblique circular cone to the original coordinate system using $\mathbf{R_c}$:

$$\mathbf{n} = \mathbf{R_c} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.24}$$

Equation (4.22) can be rearranged as $\mathbf{R_c} = \mathbf{V}\,(\mathbf{V}^\intercal \mathbf{R_c})$ and combined with equations (4.23) and (4.24) to give:

$$\mathbf{n} = \mathbf{V}\,(\mathbf{V}^\intercal \mathbf{R_c}) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \mathbf{V} \begin{bmatrix} S_2 h \\ 0 \\ -S_1 g \end{bmatrix} \tag{4.25}$$

Since S_1 and S_2 are undetermined signs, we are left with four possible solutions for $\mathbf{n}$: $\mathbf{n_a}$, $\mathbf{n_b}$, $\mathbf{n_c}$, $\mathbf{n_d}$

$$\mathbf{n_a} = \mathbf{V} \begin{bmatrix} h \\ 0 \\ -g \end{bmatrix} \quad \mathbf{n_b} = \mathbf{V} \begin{bmatrix} h \\ 0 \\ g \end{bmatrix} \quad \mathbf{n_b} = \mathbf{V} \begin{bmatrix} -h \\ 0 \\ g \end{bmatrix} \quad \mathbf{n_b} = \mathbf{V} \begin{bmatrix} -h \\ 0 \\ -g \end{bmatrix}$$

Domain-specific assumptions

We reduce these solutions down to one $\mathbf{n}$ by exploiting two assumptions:

1. $\mathbf{n}$ points towards the camera, rather than away from it. This gives the constraint $\mathbf{n} \cdot [0, 0, 1]^\intercal > 0$.
2. $\mathbf{n}$ points upwards, rather than downwards, so $\mathbf{n} \cdot [0, 1, 0]^\intercal < 0$.

Assumption 2 is specific for reverse-portrait device use, where the camera is positioned below the screen and the user's eyes are near the centre of the frame. Other methods exist for disambiguation but require greater complexity e.g. considering a 3-D geometric head model [1, 12]

Each assumption halves the number of solutions so we are left with one solution $\mathbf{n}$ – the normal to the limbal circle's supporting plane. We take this as being the direction of gaze.

4.4 Inferring the Point-of-Gaze

Once we have calculated the 3-D position and orientation of each eye, we estimate a point-of-gaze [PoG] for each eye as the intersection between that eye's direction-of-gaze [DoG] vector (optical axis) and the 3-D screen.

The DoG is the ray $\mathbf{c} + m\mathbf{n}$ extending from the 3-D limbus centre along the optical axis. We assume the screen lies in-plane with the camera axes ($z = 0$), and calculate the PoG in millimetres at this planar intersection,

$$\text{PoG}_{\text{mm}} = \begin{bmatrix} x_{\text{mm}} \\ y_{\text{mm}} \\ 0 \end{bmatrix} = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} + m \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \tag{4.26}$$

$$\text{so} \quad m = -\frac{c_z}{n_z} \quad \text{and} \quad \begin{bmatrix} x_{\text{mm}} \\ y_{\text{mm}} \end{bmatrix} = \begin{bmatrix} c_x + m \\ c_y + m\, n_y \end{bmatrix} \tag{4.27}$$

We consolidate two-eye data $\text{PoG}^{\text{left}}_{\text{mm}}$ and $\text{PoG}^{\text{right}}_{\text{mm}}$ by using their average.

There is a one-to-one mapping from 3-D points (mm) to screen-space coordinates (px) on the portable device. Assuming the camera is laterally centred and below the screen,

$$x_{\text{px}} = \frac{W_{\text{mm}}}{W_{\text{px}}} \left(x_{\text{mm}} + \frac{W_{\text{mm}}}{2} \right) \tag{4.28}$$

$$y_{\text{px}} = \frac{H_{\text{mm}}}{H_{\text{px}}} \left(y_{\text{mm}} - (H_{\text{mm}} + C_{\text{mm}}) \right) \tag{4.29}$$

Where W_{mm} and H_{mm} are the screen dimensions in millimetres, W_{px} and H_{px} are the screen dimensions in pixels, and C_{mm} is the distance from the camera to the bottom of the screen.

This requires extrinsic knowledge of the hardware setup – the position and orientation of the camera and screen. For typical gaze tracking systems, this would be recalculated every time the equipment is moved. An advantage of commodity portable devices is that these extrinsic parameters remain unchanged over a device's lifetime, and should vary little between devices of the same model type.

4.4.1 Gaze smoothing

The raw output of gaze tracking systems always contains some noise due to imperfections in algorithms and technology [32]. Though users perceive their gaze as stable, fixations are affected by eye micro-movements and tremor. These undesirable signals should be smoothed during output. This section describes how this is done using a buffer of recent gaze points, and how outliers are discarded using anatomical constants and fixation detection.

History buffer filtering

For each eye, we store a history of the i^{th} most recent detected gaze points $\mathbf{k}_i$ in a buffer of size H. To smooth gaze output, we filter over this buffer with a kernel with weights $w_i,\ i \in [1, H]$ returning smoothed point $\mathbf{k}^*$. For weights we use Kumar et al.'s [33] *triangular* kernel, which linearly assigns minimum weight to the oldest stored gaze point ($w_H = 1$), and maxiumum weight to the most recent ($w_0 = H$).

$$\mathbf{k}^* = \sum_{i=1}^{H} w_i \, \mathbf{k}_i, \quad \text{and} \quad w_i = 1 + (H - i) \tag{4.30}$$

With a high enough frame rate, this stabilises gaze during fixations without significantly increasing latency.

Outlier detection

When gaze estimation uses a non-limbus (e.g. fitting an ellipse to a nostril) or when a limbal ellipse is poorly fit, the gaze is inaccurate. As well as affecting the gaze for that image frame, incorrect gaze points contaminate the buffer, spoiling accuracy for the next H frames.

We discard an erroneously localised limbus by measuring the 3-D distance between the two eyes. If this is beyond an anatomical limit (80mm), we know at least one eye is incorrect, so discard the limbus that deviates the most from its previous position.

Though a limbus' 3-D coordinates seem plausible, it may have a poorly estimated optical axis. We can discard single-limbus errors of this kind during a gaze fixation – if we know one eye's gaze is stable, then we assume irregularities

for the other eye originate from errors. This is determined by calculating the weighted gaze distance d for each eye,

$$d = \sum_{i=1}^{H} w_i \left| \mathbf{k}_i - \mathbf{k}_0 \right| \tag{4.31}$$

If this is below some threshold $d_{\min}$, it suggests that the most recent point $\mathbf{k}_0$ has deviated little gaze history – that eye's gaze is fixed. In this situation we discard the other limbus if its weighted distance is above a maximum allowed value $d_{\max}$. This allows slightly inaccurate gaze points to contribute to an average, while detecting and discarding more extreme outliers.

Chapter 5

Implementation details

Chapters 2, 3, and 4 describe our approach for gaze tracking with portable devices. In this short chapter I detail how it was implemented, first on a desktop computer, and then as a distributed system for real-time gaze tracking on a tablet.

5.1 Python implementation

We decided to initally target a PC using Python [34] as it allowed rapid prototyping and supports a diverse range of useful libraries. Python bytecode is interpreted so it is not as fast as native machine-code compiled languages. But as the system design should ultimately run on a portable device, its PC-bound performance should not be an issue. The machine used features an Intel Xeon E3 3.3 GHz and 12 GB RAM, and samples at $\sim$18 fps[1]. Figure 5.1 is a screenshot of gaze tracking being visualized[2].

5.1.1 OpenCV

The Open Source Computer Vision Library [OpenCV] [35] is a popular collection of computer vision tools and functions. In this project we used its implementations of

- Fundemental image processing operations including linear spatial filtering, Gaussian blurring, and morphology.
- Optimized eye and eye-pair cascade classifiers trained by Castrillón-Santana et al. [36] (section 2.1).
- Direct least-squares ellipse fitting [30] (subsection 3.3.1).

[1] Further possible optimizations were not undertaken as this was sufficient.
[2] Video available online: `https://www.youtube.com/watch?v=lPcjQdSzKX4`

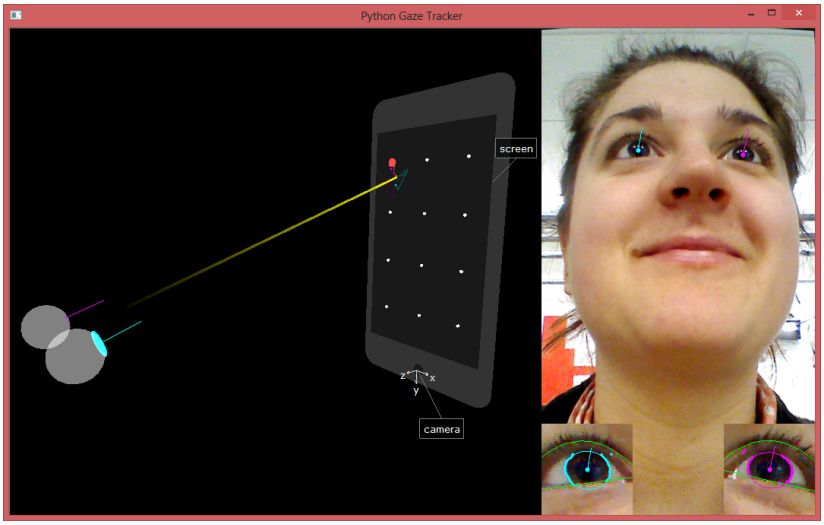


Figure 5.1: A screenshot from our VPython-based [37] 3-D visualizer. The yellow line is the smoothed gaze direction; cyan/magenta components represent raw limbus output.

5.1.2 NumPy

The NumPy package provides fast multi-dimensional arrays suitable for image operations, and in-built higher-level mathematical functions like linear algebra and polynomial fitting (subsection 3.3.1). As Python is interpreted it can be slow for mathematical algorithms like eye-centre-localisation, so efficient implementations using NumPy were required.

5.1.3 Parallelism

The system contains thread-level parallelism that we would like to exploit. Individual eyes are processed independently, as are RANSAC iterations – these could be executed simultaneously. Each of the subsystems could also be pipelined over sequential image frames with few concurrency issues.

Unfortunately traditional multi-threaded parallelism is barred in Python by the *Global Interpreter Lock* which prevents programs taking advantage of a multiprocessor system. Trials were conducted with `multiprocessing`, Python's process-based "threading" interface [38], but it isn't suited for real-time applications and frequently spawning new interpreters was detrimental to performance.

5.2 Android client

We originally planned to re-implement the system for Android tablets using OpenCV4Android [39] – Android Java bindings for OpenCV functions. Though porting the first stage of the system from OpenCV Python to Java was relatively straight-forward, it became clear that performance would not be acceptable without significant software engineering effort. On the development device, OpenCV4Android provides video up to ∼30fps, but this drops to ∼15fps once eyes are coarsely localised. This suggests real-time performance using Java on Android is unlikely, as coarse eye localisation only accounts for $\sim 2\%$ of processing time on PC.

This is because each OpenCV function in Java invokes native code using the *Java Native Interface* [JNI]. The JNI framework allows Android Java code to call pre-compiled OpenCV functions, but these calls can be slow if they require copying large matrices (e.g. images). Better performance can be achieved by using the JNI as little as possible, but this would require restructuring the entire project and compiling it to native code.

5.2.1 Distributed system

To explore gaze tracking on an unmodified tablet, we instead chose to implement a lightweight Android client to work in conjunction with the Python system over a WiFi network. This proof-of-concept system operates by streaming tablet camera frames to a PC which returns PoG coordinates.

We chose the *Motion JPEG* [MJPEG] standard as it provided the lowest latency ($\sim$ 500ms). MJPEG video frames are individually compressed JPEG images, so the frame rate was bounded by the tablet's JPEG compression rate (∼8fps). While an H.264 stream over Real-Time Streaming Protocol achieved a better frame rate, it's latency was higher and thus unacceptable for gaze-interaction. The Peepers open source Android streaming framework [40] was adapted for this project.

Though not pervasively usable itself, this distributed framework can be deployed for research in gaze-interaction (as described in chapter 6) or could provide cheap, unintrusive gaze tracking for attention studies without funds for expensive commercial systems.

Chapter 6

Evaluation

The evaluation of the project was carried out in two parts.

1. A technical analysis to characterize system accuracy over varied participants and realistic conditions.
2. A feasibility study of gaze-based password entry using the system.

This chapter presents the design and results of each experiment in turn. We show that the system is capable of tracking gaze with root-mean-squared error of 2.6° (or 7.7mm), and was succesfully deployed in a gaze-interaction study where users entered gaze passwords correctly 49% of the time.

6.1 Technical analysis

For tracked gaze to be useful in analysis or interaction, it should be accurate. For pervasive use, this accuracy should be stable over realistic portable device use conditions. Therefore we conducted an experiment to measure our system's accuracy over varied participants and realistic lighting conditions.

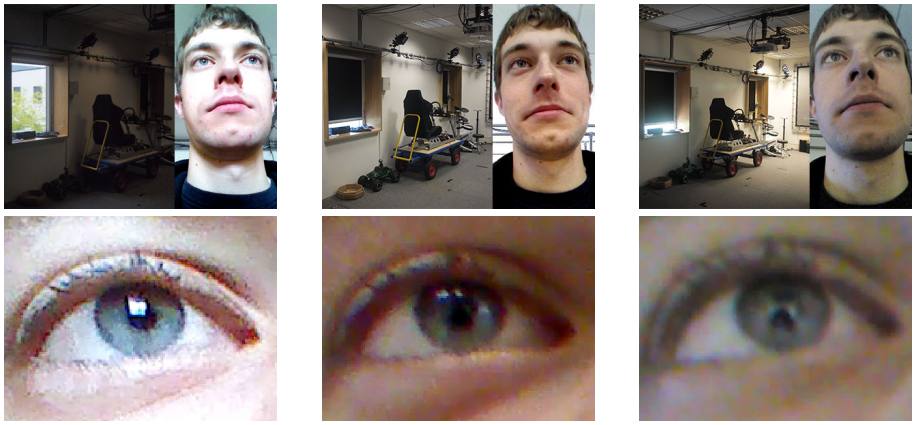
As is typical for gaze tracking evaluation, we measured the *root-mean-squared error* [RMSE] between true and estimated gaze points. For a set of N points,

$$\text{Acc}_{\text{mm}} = \text{RMSE}_{\text{mm}} = \sqrt{\frac{1}{N}\sum_i^N (\mathbf{x}_i - \hat{\mathbf{x}}_i)^2} \tag{6.1}$$

where $\hat{\mathbf{x}}_i$ and $\mathbf{x}_i$ are the true and estimated gaze point coordinates.

Accuracy is most commonly presented in angular degrees, calculated using some average distance between camera and eyes Z_{mm}. We use the average camera/eye-pair distance over all users – 201mm.

$$\text{Acc}_{\text{deg}} = \tan^{-1}\left(\frac{\text{Acc}_{\text{mm}}}{Z_{\text{mm}}}\right) \tag{6.2}$$



Sunlight (L1) Indoors, bright (L2) Indoors, dark (L3)

Figure 6.1: Photos of the environment under the three lighting conditions, with corresponding face and example eye-ROI images.

The system does not run on-device so *gaze videos* were recorded and analysed offline. These videos were taken with the front-facing camera and recorded participants' faces as they stared at a pattern of *gaze-markers* – on-screen widgets representing true gaze points.

6.1.1 Experiment design

Independent variables

We conducted a 3×2 repeated measures design experiment, varying the order conditions were presented in over participants for counterbalancing. Accuracy was measured under three lighting conditions which represented realistic typical use environments (Figure 6.1).

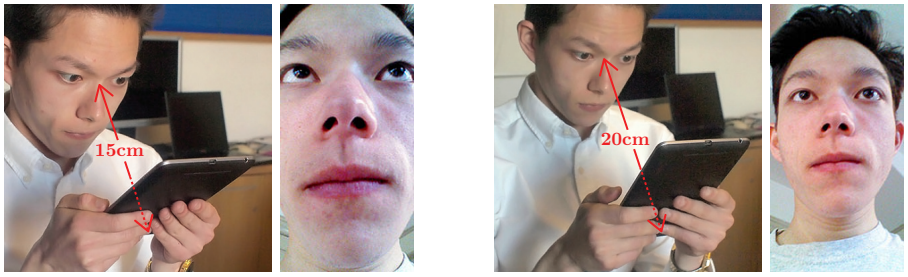
- **Sunlight (L1)**: Participants sat $\sim$3 metres from a large window.
- **Indoors, bright (L2)**: Window blinds were shut, and the room was illuminated by 3 strips of overhead halogen lights.
- **Indoors, dark (L3)**: As L2, but with only one strip of lights.

For the system to be robust to lighting changes, accuracy should not differ significantly between L1, L2 and L3.

Participants were instructed to hold the device in two ways (Figure 6.2):

- **Near distance (D1)**: $\sim$15cm between camera and eyes.
- **Far distance (D2)**: $\sim$20cm between camera and eyes.

This was to measure the effect of smaller eye-ROIs on system accuracy, as we hypothesised that fewer limbus pixels means less precise model fitting. It was



Near distance (D2) Far distance (D2)

Figure 6.2: Excerpts from our instruction booklet showing suggested holding positions for D1 and D2, with example camera frames.

also to ensure that data collected was useful in analysis – users must cooperate and hold the tablet so the eyes are visible in image frames.

Dependent variables

In this study we took measurements for:

- **Full-screen accuracy (FA)**: RMSE over all video frames.
- **Half-screen accuracy (HA)**: RMSE over video frames corresponding to gaze in the top half of the screen.

HA was measured as we hypothesized that accuracy degrades as the users look towards the bottom of the screen[1]. This is because eyelid occlusion becomes severe, with both top and bottom eyelids covering the iris. Also, when the PoG approaches the camera focal point the limbal ellipse appears more like a circle, and small errors in ellipse fitting become exaggerated when used in rotation calculations.

Experiment Apparatus & Software

An Asus Nexus 7 tablet was used for recording gaze videos (Figure 6.3). This 7-inch tablet represents cheaper commodity portable devices[2] (∼£160), so does not feature cutting-edge components. Its camera records at 0.9 megapixels (720 × 1280px) at a variable frame rate ($7.5 \leq \text{fps} \leq 29$) depending on scene exposure. Gaze-markers were arranged in a 3 × 4 grid on its 800 × 1280px screen. To simulate pervasive use, the tablet was not physically mounted and particpants held it themselves.

[1] This is domain-specific for reverse-portrait orientated cameras.
[2] Millions of Nexus 7s are in circulation [41], so it is representative of popular devices.

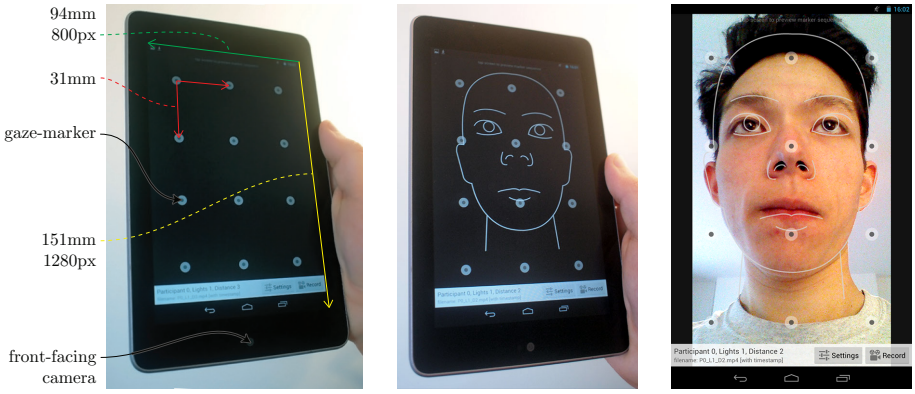


Figure 6.3: The Nexus 7 tablet running our video recording app, displaying face outlines and gaze-markers.

An Android app was developed for recording the gaze vidoes. It featured animated gaze-markers to guide the participants and face-part outlines to help the participants position their face for D1 and D2. Following data collection, gaze videos were analysed offline using an evaluation harness which automatically processed gaze videos and measured accuracy. The PC described in section 5.1 was used.

Participants & Procedure

11 paid participants (`P01` to `P11`, aged 21-48, 7 female) were selected via mailing lists, online adverts, and the Computer Lab. No glasses-wearers were chosen as a pilot study revealed our system does not handle them reliably.

Participants sat in each of the three lighting conditions sequentially, recording gaze videos at each of the different distance conditions. Each recording was repeated for a total of three videos. We then presented participants with a post-study questionnaire asking how familiar, natural, and comfortable the conditions were.

6.1.2 Results

Average full-screen accuracy was poor $(13.8 \pm 5.48°)$[3], and signifcantly worse than half-screen accuracy $(9.48 \pm 4.3°)$, as confirmed by a paired-samples t test ($t(65) = 9.51$, $p < 0.0001$). This confirms our hypothesis that accuracy is worse at the bottom of the screen.

Repeated measures ANOVA tests showed a significant effect of the distance on FA ($F(1, 10) = 11.3$, $p < 0.001$) and HA ($F(1, 10) = 8.37$, $p < 0.01$), confirming our hypothesis that accuracy degrades with smaller eye-ROIs.

[3]Results formatted as (Mean ± Standard Deviation)

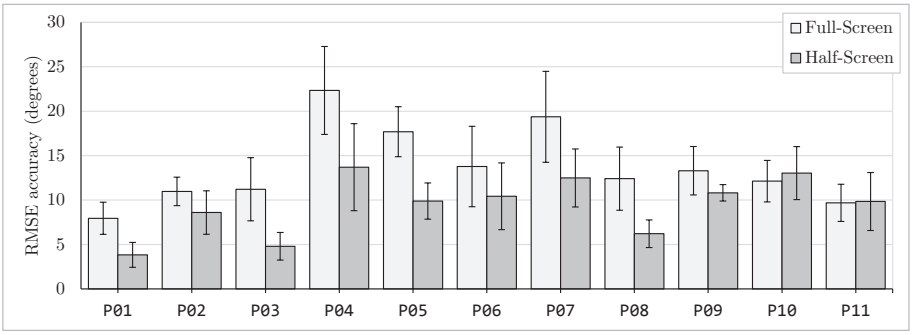
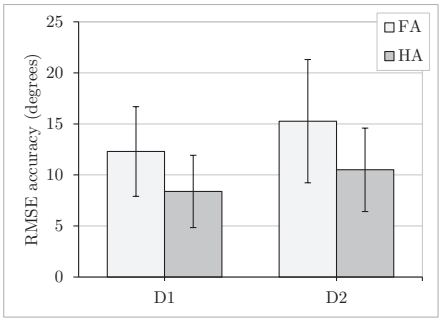
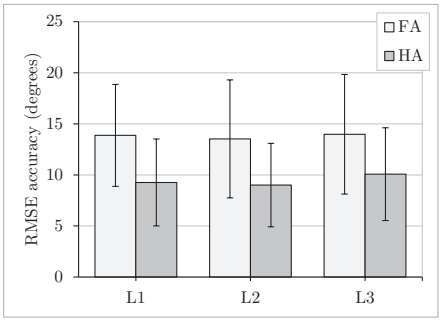


Figure 6.4: Average accuracy for all participants P01-P11 (M ± SD).



(a) D accuracy (M ± SD)



(b) L accuracy (M ± SD)

Figure 6.5: While distance had a significant effect on accuracy, lighting did not..

There was no significant effect of lighting on FA ($F(2, 20) = 0.222$, $p > 0.1$) or HA ($F(2, 20) = 1.01$, $p > 0.1$), and our questionnaire revealed that all participants used devices in similar lighting to L1, and most in L2 and L3 also. This confirms that our system design is robust in a range of realistic lighting conditions. No participants found D1 natural and only a few were happy to use it for a short time (∼2 mins), but most found D2 natural and were willing to use it for a long time (∼15 mins).

One-way ANOVA tests showed there were significant differences between participants for FA ($F(10, 55) = 7.44$, $p < 0.001$) and HA ($F(10, 55) = 7.44$, $p < 0.001$). As shown in Figure 6.4, average FA ranged from 8° to 22°, and HA from 3.7° to 14°.

There was no interaction effect between independent variables for FA ($F(2, 20) = 0.291$, $p > 0.1$) or HA ($F(2, 20) = 0.888$, $p > 0.1$).

6.1.3 Discussion

Average FA is poor, and can only guarantee knowledge of which vertical half of the screen a user looks at. HA is a little better, and could be used to determine if a user has looked at a relatively large mobile ad.

Accuracy for some participants was quite good. P01 and P03's half-screen gaze was tracked to under 5°, with P01's gaze being tracked 2.6° in the best case. This accuracy is closer to typical gaze tracking systems and is sufficient for more applications such as attention analysis. P01 was blue-eyed and P03 was brown-eyed, so this accuracy confirms our system handles limbus extraction well despite lower iris/sclera contrast. Our approach is limited by its oversimplfied eye-model, so we cannot expect accuracy $< 1°$.

We argue that decent accuracy in the top half of the screen is sufficient for a range of applications. Mobile ads can be positioned there and dismissed once viewed to reduce user frustration. During one-handed use, the top of the screen may be difficult to reach, so users could interact with it using gaze. Mobile OS notifications often appear at the top of the screen[4], so a an attentive system could automatically dismiss them once noticed.

Though accuracy was better when held at D1 rather than D2, no one found D1 comfortable. This suggests applications using our approach should either rely on lower accuracy for longer sessions, or only require high accuracy in short bursts. Examples might be acknowledging short video adverts in lieu of app payments, or confirming that all of an app's permissions have been inspected before installation to protect against opportunistic malware.

Study Limitations

Though participants were instructed to use the distance-aligned face outlines, they occasionally drifted, making imaging the eye more difficult. This is apparent as the overall average camera/eye-pair distance was 201mm, where it should have been $\sim$ 175mm for correct face outline use. This affected the distance condition internal validity.

Analysis was limited by variable frame rates. We planned to treat each video frame as a repetition, but while some L1 videos contained $\sim$1400 frames (29fps), the L3 counterparts only had $\sim$360 (7.5fps)[5]. Some videos switched between framerates within a file, so were incompatible with our evaluation harness and were discarded.

6.2 Gaze-based password entry

Gaze-based authentication is a popular form of gaze-interaction because it provides resistance to *shoulder-surfing*. This is where an attacker steals credentials or other sensitive information via direct-observation. Shoulder-surfing is difficult to protect against and undermines effort involved with encryption and secure authentication protocols. Commodity portable devices are particularly succeptible to this, as they are used often in public places.

[4]For example: Android and Windows Phone *toast* widgets [42, 43]

[5]Frame rate could be artificially degraded during evaluation, but this would no longer represent the system running at full possible frame rate.

In this section I describe how we explored gaze-based password entry with this system, using an adapted form of Kumar et al's EyePassword study [2] (subsection 1.3.2). In it, participants were shown a *target password* and asked to enter it using different input methods.

6.2.1 Experiment design

Two input methods (indepdendent variable) were compared:

- **Touch (T)**: Participants tapped on-screen buttons, similar to typical touch-screen keyboard interaction.
- **Gaze+Trigger (G)**: Participants stared at a desired key and pressed a *trigger* button to activate it [2].

We used a repeated-measures design where each participant entered five passwords with each input method. The order of input methods was varied over participants to counterbalance against learning effects.

The dependent variables were time taken to input each password, and *success rate* – the percentage of passwords entered correctly. On an incorrect entry attempt we stored the target password and input password so we could analyse the *edit distance*[6] – the number of single-digit replacements required to match the input and target passwords.

Experiment apparatus & software

The distributed gaze tracking system consisted of the Nexus 7 from section 6.1, and a PC powered by an Intel i5 3.4 GHz. An Android app was built using the distributed gaze tracking framework described in chapter 5. The interface included a camera-preview, password fields, and a keypad for password entry[7]. Metrics were recorded in app logs.

Keypad layout

EyePassword used familiar keyboard layouts to leverage the user's memory, thus reducing *visual search time* [2] – the time they spend looking for a desired key. As shown in Figure 6.6, we used a modified ATM-style keypad, featuring digits one to nine in a 3×3 layout. Zero was omitted for screen-space efficiency. The ATM-style keypad also included buttons for clearing or confirming the current input, and a button for triggering gaze. Gaze target zones were 28mm×28mm.

Target passwords were four digits long and chosen randomly, e.g. `6846`.

[6] Also known as the *Levenshtein distance*.
[7] Video available online: `https://www.youtube.com/watch?v=xAFqg-IXlCQ`

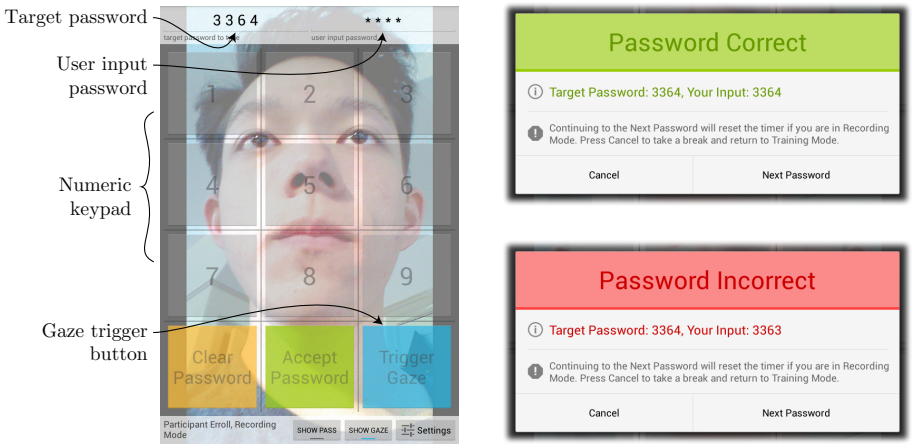


Figure 6.6: App interface and dialogs for password entry success or failure.

Input feedback

Though gaze-based typing techniqes commonly present PoG feedback to assist input [44], authentication applications should hide this information from attackers. However, as the distributed system's latency is high, it was decided to include a visible PoG-marker. The screen also flashed to signify gaze being triggered.

The user-input password was displayed in asterisks (e.g. ****), so they could tell how many digits they'd entered.

Participants & procedure

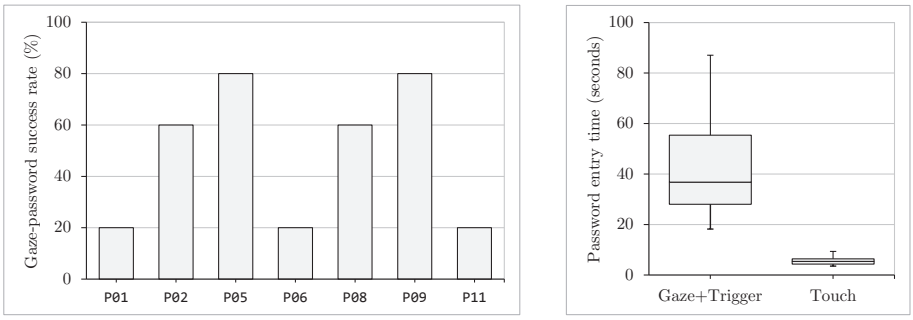
This experiment was carried out directly after gaze video recording, so used the same participants. 4 of these found the system to be too inaccurate during the training period, and chose to withdraw, leaving 7 (aged 21-33, 5 female).

Participants sat in L1 lighting conditions (section 6.1). They were allowed a training period to acquaint themselves with Gaze+Trigger's operation, and get a feel for system performance. One they were comfortable with the system, they were asked to enter five passwords using each input method. If they felt they'd made a mistake, they could clear their input and try again.

Finally we presented them with a questionnaire to collect their views on gaze-based password entry in general, and the system in this experiment.

6.2.2 Results

A paired-samples t test showed a significant effect of the input method on password entry time. ($t(34) = 11.5$, $p < 0.0001$). G password entry took 42.5±18.6 seconds compared to 5.61±1.61 seconds with T. Participants entered



(a) Success rate **(b)** Entry times

Figure 6.7: Gaze-password performance. Bars in (b) are max and min values, boxes represent Q2 and Q3.

all T passwords correctly, but only 49 ± 26 correctly % with G. For incorrect G passwords, the average edit distance was 1.28 ± 0.56 digits, with 78% of incorrect passwords only being off by one.

Our subjective analysis showed that participants unanimously felt "in control" using T and were not concerned at all about the T input time. None felt they "in control" when using G, though half the participants were not sure. Apart from one, they were all concerned about the the time taken for G input. Most described the system's accuracy as "somewhat accurate".

All participants reported that they compensated for the system's inaccuracy. Most "overlooked" – they stared at extreme corners of the on-screen buttons, especially towards the bottom half of the keypad.

All participants felt it was important to protect oneself against shoulder-surfing, but were not enthusiastic about gaze-passwords – just over half would prefer to enter their password using gaze.

6.2.3 Discussion

Gaze-password success rate was low. This may be because of tracking inaccuracy, high latency, and low frame rate. Poor accuracy meant the PoG deviated from the desired digits. High latency and low frame rate meant participants had to wait for the system to register shifts in gaze, and caused gaze estimation to jerk around the screen. However, when the participants entered an incorrect gaze password, they were normally only off by one digit. This suggests that with better accuracy, the success rate might be acceptable. Though our gaze success rates were lower than EyePassword's (15%) [2], it was comparable to other gaze-authentication methods (Forget et al.'s [45] CGP-1: 50%).

Curiously, the participants with the best G success rates (`P05`, `P09`) did not correspond to having their gaze tracked accurately. This suggests that they adjusted their eye-shape to better suit the system during use, encouraged by real time feedback.

We hypothesise that poor system performance led participants to use it more as a *gaze-joystick* rather than a *gaze-mouse* – thinking of the PoG-marker as a cursor which they had to "nudge into place"[8] using their gaze. This is supported by how they "overlooked" to exaggerate their gaze direction.

Inputting a password with gaze took participants about eight times longer than touch, and all participants but one found this concerning. This may be because of poor latency and frame rate, so participants stared for a long time to smooth the visble PoG-marker. The keypad did not include a delete key, so noticing an input error meant re-entering the entire password.

This study was limited by the randomized passwords. Pre-defined training and test sets would have allowed more insight into errors. The participant sample size was low, and not gender or age-balanced.

Though these results suggest that our system is unsuitable for reliable gaze authentication, this study demonstrated the feasibility of using the system in a gaze-interaction study.

[8]Participants' phrases taken from a free-form questionnaire section.

Chapter 7

Conclusion

I have presented a model-based gaze tracker for commodity tablets – the first of its kind. I have shown that it satisfies requirements of pervasive use:

- **Low quality camera support**
  We can precisely fit limbal ellipses despite low resolution, noisy, and out-of-focus images using feature detection and RANSAC model fitting.
- **Reliability in different lighting conditions**
  Eye localisation and limbus extraction is fast and reliable under a range of realistic lighting conditions, as demonstrated in our technical analysis.
- **Calibration-free operation**
  As eye localisation is user-independent and our geometric model is simple, users can have their gaze tracked instantly, as demonstrated in our gaze-interaction study.

We demonstrated the effectiveness of our approach in an evaluation over eleven participants where the system tracked gaze at 9° (half-screen accuracy), and up to 2.6° (7.7mm) for some users – adequate for many gaze tracking applications. These initial results are promising, and suggest our approach has potential for cheap consumer devices of the near-future.

Though the system was not implemented on-device, we developed a distributed framework which was used to partially recreate Kumar et al.'s [2] EyePassword study. This study showed the system's feasibility for gaze-interaction studies without instrusive or specialist equipment.

7.1 Future work

The next natual step is to implement an integrated single-device system using our design, fulfilling the project's original goal. Using software engineering approaches described in chapter 5, I believe real time gaze tracking is feasible

without PC-tier processing power.

Our disjoint subsystem architecture provides several tracks for future work. A limitation of eye localisation is its unreliability for glasses-wearers. This could be solved using additional classifiers. Our hybrid eye-centre localisation approach could be extended by considering temporal information or with a scale-space framework to improve accuracy and efficiency. As eyelid fitting is important for iris recognition as well as gaze tracking, a wealth of advanced techniques for it exist. Our simple approach could be replaced with active contours or graph-cut segmentation to improve limbus point outlier removal before fitting.

The most interesting future research would explore how gaze might be used pervasively. Our approach could be used in interfaces that adapt to the user's visual attention, e.g. automatically dismissing a notification if the user has read it. Fusion of gaze and touch input could assist interaction with difficult-to-reach parts of the screen during one-handed use [14]. An adaptation of our system could be deployed to study these in natural environments without needing expensive or awkward hardware setups.

Bibliography

[1] J-G Wang, Eric Sung, and Ronda Venkateswarlu. Eye gaze estimation from a single image of one eye. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 136–143. IEEE, 2003.

[2] Manu Kumar, Tal Garfinkel, Dan Boneh, and Terry Winograd. Reducing shoulder-surfing by using gaze-based password entry. In *Proceedings of the 3rd symposium on Usable privacy and security*, pages 13–19. ACM, 2007.

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

[4] Andrea Canessa, Agostino Gibaldi, Manuela Chessa, Silvio Paolo Sabatini, and Fabio Solari. The perspective geometry of the eye: Toward image-based eye-tracking. *Human-Centric Machine Vision, InTech*, 2012.

[5] Ilias Leontiadis, Christos Efstratiou, Marco Picone, and Cecilia Mascolo. Don't kill my ads!: balancing privacy in an ad-supported mobile application market. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, HotMobile '12, pages 2:1–2:6, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1207-3. doi: 10.1145/2162081.2162084. URL `http://doi.acm.org/10.1145/2162081.2162084`.

[6] Andreas Bulling and Hans Gellersen. Toward mobile eye-based human-computer interaction. *Pervasive Computing, IEEE*, 9(4):8–12, 2010.

[7] Corey Holland and Oleg Komogortsev. Eye tracking on unmodified common tablets: Challenges and solutions. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 277–280. ACM, 2012.

[8] Elaine Nicpon Marieb and Katja Hoehn. *Human anatomy & physiology*. Pearson Education, 2007.

[9] Tobii Eye Tracking Research. Flexible eye tracking - tobii x60 & x120, 2013. URL `www.tobii.com/en/eye-tracking-research/global/products/hardware/tobii-x60x120-eye-tracker/`.

[10] Karen M Evans, Robert A Jacobs, John A Tarduno, and Jeff B Pelz. Collecting and analyzing eye-tracking data in outdoor environments.

[11] Arrington Research. Turnkey integrated headtracking & eyetracking solutions, 2013. URL `www.arringtonresearch.com/prices.html`.

[12] Haiyuan Wu, Qian Chen, and Toshikazu Wada. Conic-based algorithm for visual line estimation from one image. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 260–265. IEEE, 2004.

[13] Roberto Valenti, Nicu Sebe, and Theo Gevers. Combining head pose and eye location information for gaze estimation. *Image Processing, IEEE Transactions on*, 21(2):802–815, 2012.

[14] Takashi Nagamatsu, Michiya Yamamoto, and Hiroshi Sato. Mobigaze: Development of a gaze interface for handheld mobile devices. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, pages 3349–3354. ACM, 2010.

[15] Heiko Drewes, Alexander De Luca, and Albrecht Schmidt. Eye-gaze interaction for mobile phones. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, pages 364–371. ACM, 2007.

[16] Kristian Lukander. A system for tracking gaze on handheld devices. *Behavior research methods*, 38(4):660–666, 2006.

[17] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[18] Richard Szeliski. *Computer vision: algorithms and applications*. Springer, 2010.

[19] Lech Świrski, Andreas Bulling, and Neil Dodgson. Robust real-time pupil tracking in highly off-axis images. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 173–176. ACM, 2012.

[20] Wayne J Ryan, Damon L Woodard, Andrew T Duchowski, and Stan T Birchfield. Adapting starburst for elliptical iris segmentation. In *Biometrics: Theory, Applications and Systems, 2008. BTAS 2008. 2nd IEEE International Conference on*, pages 1–7. IEEE, 2008.

[21] Fabian Timm and Erhardt Barth. Accurate eye centre localisation by means of gradients. In *Proceedings of the Int. Conference on Computer Theory and Applications (VISAPP), Algarve, Portugal*, pages 125–130, 2011.

[22] Roberto Valenti and Theo Gevers. Accurate eye center location and tracking using isophote curvature. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[23] Robert Laganière. *OpenCV 2 computer vision application programming cookbook*. Packt Publishing Ltd, 2011.

[24] Jose Sigut and S-A Sidha. Iris center corneal reflection method for gaze tracking using visible light. *Biomedical Engineering, IEEE Transactions on*, 58(2):411–419, 2011.

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

[26] Carles Fernández, Dídac Pérez, Carlos Segura, and Javier Hernando. A novel method for low-constrained iris boundary localization. In *Biometrics (ICB), 2012 5th IAPR International Conference on*, pages 291–296. IEEE, 2012.

[27] Dirk Schnieders, Xingdou Fu, and K-YK Wong. Reconstruction of display and eyes from a single image. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1442–1449. IEEE, 2010.

[28] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004.

[29] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[30] Andrew Fitzgibbon, Maurizio Pilu, and Robert B Fisher. Direct least square fitting of ellipses. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(5): 476–480, 1999.

[31] Qian Chen, Haiyuan Wu, and Toshikazu Wada. Camera calibration with two arbitrary coplanar circles. In *Proc. European Conf. Computer Vision*, pages 521–532. Inc, 2004.

[32] Oleg Špakov. Comparison of eye movement filters used in hci. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 281–284. ACM, 2012.

[33] Manu Kumar, Jeff Klingner, Rohan Puranik, Terry Winograd, and Andreas Paepcke. Improving the accuracy of gaze input for interaction. In *Proceedings of the 2008 symposium on Eye tracking research & applications*, pages 65–68. ACM, 2008.

[34] Guido Van Rossum and Fred L Drake. *The Python Language Reference Manual*. Network Theory Ltd., 2011.

[35] Itseez OpenCV Foundation. Opencv open source computer vision, 2013. URL `http://opencv.org/`.

[36] Modesto Castrillón-Santana, O Déniz-Suárez, L Antón-Canalís, and J Lorenzo-Navarro. Face and facial feature detection evaluation performance evaluation of public domain haar detectors for face and facial feature detection. 2008.

[37] Bruce Sherwood. Vpython, 2013. URL `http://www.vpython.org/`.

[38] The Python Standard Library. multiprocessing process-based threading interface, 2013. URL `http://docs.python.org/2/library/multiprocessing.html`.

[39] Itseez OpenCV Foundation. Opencv4android, 2013. URL `http://opencv.org/platforms/android.html`.

[40] Foxdog Studios. Peepers: realtime video streaming from android, 2013. URL `http://foxdogstudios.com/peepers.html`.

[41] Joshua Sherman. Analyst estimates nexus 7 sales as high as 4.8 million, 2013. URL `http://www.digitaltrends.com/mobile/nexus-7-sales-estimated-not-bad/`.

[42] Google Android. Toasts, 2013. URL `http://developer.android.com/guide/topics/ui/notifiers/toasts.html`.

[43] Windows Phone Dev Centre. Toasts for windows phone, 2013. URL `http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj662938(v=vs.105).aspx`.

[44] Paivi Majaranta, I Scott MacKenzie, Anne Aula, and Kari-Jouko Räihä. Auditory and visual feedback during eye typing. In *CHI'03 Extended Abstracts on Human Factors in Computing Systems*, pages 766–767. ACM, 2003.

[45] Alain Forget, Sonia Chiasson, and Robert Biddle. Shoulder-surfing resistance with eye-gaze entry in cued-recall graphical passwords. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 1107–1110. ACM, 2010.