Diploma Thesis


# A Software Based Approach to Real-Time Eye Tracking

Anders Blehr
Division of Computer Science & Telematics
The Norwegian Institute of Technology
The University of Trondheim

April 12, 1993

**Abstract**

This paper constitutes my diploma thesis at the Norwegian Institute of Technology, Division of Computer Science & Telematics. On the basis of a relatively broad discussion of general digital image processing techniques, an $O(N)$ algorithm for determining the location of the pupil in an $N \times N$ image of the eye has been developed. A prototype of the algorithm has been implemented and tested, and the results obtained were discussed in terms of a set of given requirements. Lastly, some suggestions were made as to how to employ the algorithm in a real-time eye-tracking system.

# Contents

# List of Figures

# List of Tables

# Preface

This paper constitutes my diploma thesis for the title "sivilingeniør" at the Norwegian Institute of Technology. As a participant in the EC ERASMUS exchange programme, I have had the opportunity to carry out my thesis work at the Institute of Applied Physics and Biophysics, Philipps-Universität Marburg, Germany.

My task has been to develop and implement an algorithm for determining the location of the pupil in a given image of the eye. The algorithm is at a later stage intended for form the core of a real-time eye-tracking system, and accordingly it has been of vital importance during the development process that the algorithm be capable of performing in a real-time environment. A second important requirement has been that the positions returned by the algorithm be as a accurate as possible.

The structure of the paper is as follows. Chapter 1 is intended to give an overall view of the background and motivation for my thesis work. The requirements that a future eye-tracking system will have to satisfy are presented, and in relation to these, the problem definition to which my work has been related is given. Chapter 2 supplies some general background material relating to the eye and its structure. Also, some aspects of the optical properties of the eye are discussed. Lastly, some already existing eye-tracking systems are described and evaluated with respect to the given requirements. In Chapter 3, a relatively broad presentation of different image processing techniques is given. These techniques all constitute possible approaches to the given problem and accordingly are given a relatively thorough treatment. In Chapter 4, the techniques presented are evaluated with resepct to their applicability to the problem at hand and a decision is made as which combination of techniques appears to constitute the most promising approach to designing an algorithm to solve the problem. In Chapter 5, the proposed algorithm is presented in detail whereupon it is evaluated in terms of the given requirements, based on a set of tests performed. In Section 6 some conclusions that can be drawn from my thesis work are presented and finally, Appendix A discusses some implementation issues and lists the sources constituting the current implementation of the algorithm.

I would like to thank my supervisor at Philipps-Universität Marburg, Reinhard Eckhorn, for giving me the opportunity to stay here in Marburg, and also my supervisor at the Norwegian Institute of Technology, Jan Komorowski, for consenting to me staying here. A very special thanks goes to Uwe Thomas for his continued support and help during all stages of my work, and particularly for being a friend to turn to. Thanks also to the staff here at the institute for lighting up the gray days of programming. Lastly, I would like to say a very big thanks to my family for making it all possible and to all my friends in Trondheim, presently dispersed all over Norway and Europe, for making the four years we spent together in in Trondheim a memorable time.

# Chapter 1

# Introduction

This chapter is intended to give an overall view of the background and motivation for my thesis work. In Section 1.1, a brief description of the project of which the work has been a part, the *Synchronization and Cognition* (SC) project, is given. The motivation for my assignment as related to the SC project is described in Section 1.2. In Section 1.3, a thorough definition, hereafter referred to as the *problem definition*, of what has been my task within the SC project is given. In the last section of the chapter, Section 1.4, the working environment I have been subjected to is described.

## 1.1 Synchronization and Cognition

*Synchronization and Cognition* (SC) is a cooperation project initiated by R. Eckhorn and R. Bauer of the Biophysics Group and F. Rösler of the Physiological Psychology Group at the Philipps-Universität Marburg, Germany.

The basis of the current work is observations done by R. Eckhorn and R. Bauer ([9]), as well as by the group of W. Singer (e.g., [21] and [22]), that when stimulated, distributed neural assemblies organize rhythmic activities in peripheral visual areas in the brain of anesthetized cats. The phase coupling among these assemblies, but not their frequencies, depend on the applied visual stimulus.

The goal of the SC project is, through the use of microelectrodes (monkeys; *makaba mulatta*) and EEG recordings (humans), to show that coherent neurological activations (*synchronizations*) in the visual cortex, relating to coherent visual stimuli, is a requirement for the coherent perception of these stimuli. In addition, the time-space *synchronization dynamics* in peripheral and central as well as in intermediate cortex areas are to be investigated as to how they relate to the cognitive tasks being performed.

For a further description of the SC project, see [10]. Some of the publications by R. Eckhorn *et al.* concerning the discovery and implications of synchronizations in the visual cortex of cats and monkeys are [7], [8], [9] and [11].

#### Note

During my thesis work, I have only interacted with the group of R. Eckhorn and R. Bauer. Accordingly, I will in the following refer to the *monkey* and the *subject* interchangeably. The general principles, however, when referring to the monkey, can be assumed also to apply to the work done by the group of F. Rösler.

## 1.2 Thesis Motivation

Since the direction of gaze and the orientation of the visual stimuli have to correspond as exactly as possible in order to be certain that exactly those neural assemblies are stimulated whose activity is being registered, it is of vital importance for the validity of the experimental results that the exact eye position of the subject be known at a given time. Otherwise the danger exists of registering

neurological responses whose variations are not the result of the assigned cognitive task, but of the subject's incorrect gaze direction.

In the part of the SC project carried out by the Eckhorn group, the visual stimuli are generated on a computer monitor placed in front of the monkey, sometimes referred to as the *inducing monitor* (this term stems from the fact that the neural activity in the visual cortex of the monkey is *induced* by the visual patterns on the monitor). The monkey is trained to focus on a bright point, called the reference point, on the monitor, across which various geometrical shapes move in different directions. When the monkey does not focus on the reference point, it is said to be *drifting*.

To gain absolute certainty about where the monkey at a given time focuses, it is necessary to have a system which continuously tracks the movements of the monkey's eyes. This system has to comply to the following requirements:

1. The position-analyzing method of the system may in no way interfere with recordings being made from the visual cortex of the monkey.
2. A position measuring rate of 50 Hz. Thus every 20 ms, hereafter referred to as a *pass*, a coordinate pair relative to a reference point on the inducing monitor, corresponding to the point on the monitor on which the monkey had its focus at the beginning of the pass, has to be made available. When the monkey is drifting, a warning signal has to be given with each pass as long as it continues to drift.
3. The error in the position returned by the system should be less than or equal to $\pm 0.25°$.
4. The measuring range of the system should be at least $\pm 20°$.
5. The system should be relatively easy to configure to comply with the specific needs of an experiment.
6. The price of the complete system may not exceed DM 12,000.

There exist several commercially available eye tracking systems, some of which are described in Section 2.2. For reasons elaborated upon in Section 2.2.5, it was decided to develop an entirely new eye tracking system, based on real-time image processing of video input.

## 1.3 Problem Definition and Initial Work

Originally, it was the intent that I should concern myself with the design of a complete eye-tracking system. My task was defined by the following problem definition, hereafter referred to as the *overall problem definition*:

**Overall problem definition:** Design and test a complete system for measuring the eye movements of the subject during neurophysiological recordings from the visual cortex (monkey) or EEG recordings (man), satisfying the requirements listed in Section 1.2. The different parts of the system to be designed are:

- An eye-tracking program satisfying requirements 2–6 of Section 1.2. The program shall run within the framework of an IBM-compatible PC, either in the CPU of the PC itself or on a special-purpose image processing board subjected to it.
- An interface between the eye-tracking program and the registering equipment of the setup, to enable the simultaneous and correlated registration of brain activity and gaze direction.
- A setup to ensure optimal illumination of the subject's eye. In order to minimize the induced activity in the visual system of the subject unrelated to the visual task, the illumination is to be in the infrared area.
- A setup supplying the camera with an optimal image of the eye. This is to be done with an infrared mirror placed in front of the eye, reflecting an infrared image of it to the camera. The camera is to be placed in front of and above the eye, perpendicular to the gaze direction.

In addition to the design, the following items have to be purchased:

- A highly IR sensitive camera. The camera must conform to the CCD video standard, and must be able to deliver at least 50, preferably 100, half frames per second.
- All items required to build the illumination and camera setups described above.
- A frame grabbing PC board. The video input of the grabber has to conform to the CCD video standard, and it must be able to receive and digitize as many half frames per second as the camera delivers. The digitized images are to have a resolution of $512 \times 512$ pixels with 256 gray levels. Alternatively,
- a special-purpose image processing board, in case of deciding on letting the eye-tracking program run on such a board instead of in the CPU of the host PC. The board has to fulfil the requirements listed above, and its processor(s) must be easily programmable.

The total cost of the above items may not exceed the amount given in requirement 6 in Section 1.2.

As the resolution of the system to be designed depends on the resolution of the digitized images, it would have been desirable with an image resolution of at least $1024 \times 1024 \times 256$. However, the number of pixels in an image with this resolution is too high from a real-time image processing point of view to be sustainable. Thus it was decided on a lower resolution of $512 \times 512 \times 256$, as indicated above.

**Working with the Initial Problem Definition**

During the first months of my thesis work, I spent a lot of time trying to find reasonable approaches to problems pertaining to the non-algorithmic, fundamental aspects of this definition. The most important of these can be summarized as

- Does the requirement of speed imply that the final system has to employ a special-purpose image processing board? In that case, which requirements does the image processing board have to satisfy? Which image processing boards are available, and what can they do? Do they satisfy the technical and low-cost requirements? From which vendors are they available? How long is the delivery time for the different boards? How easily programmable are they?
- In case an algorithm is found that is fast enough to allow the program to run in CPU of the host PC, which requirements does the frame grabbing board have to satisfy? Which frame grabbing boards are available, and what do they offer? Do they satisfy the technical and low-cost requirements? From which vendors are they available? How long is the delivery time for the different boards?
- Which are the requirements that the camera has to satisfy, in addition to the requirements of sensitivity, speed, and low cost? Which highly infrared sensitive cameras complying to the requirement of speed are available? And from which vendors? How do the video signals from the different cameras conform with the CCD standard, alternatively with the video input channels of the different frame grabbing and image processing boards that are available?
- How shall the problem of eye illumination be approached? Are infrared light-emitting diodes sufficient? Which is the best positioning of the diodes relative to the eye to ensure optimal illumination? How many diodes are necessary to illuminate the eye sufficiently? Does the illumination constitute a danger to the eye?

Much time was spent in order to arrive at answers to some of these questions, partly on the telephone with different companies offering various cameras, frame grabbing and image processing boards, inquiring about how well the different cameras and boards would suit the given problem, ordering information material and inquiring about other vendors offering similar products; and partly browsing through a considerable amount of information material, trying to land at a conclusion as to which vendor offered the most cost-efficient solution. Also, different approaches to the

illumination problem were investigated, diodes were acquired and tested, and simple illumination setups were built.

After some months of working with the overall problem definition, it became clear that the time available would not suffice to design a complete eye-tracking system, because neither a suitable image processing board complying to the low-cost requirement, nor a corresponding frame grabbing board or camera had been found. Particularly, as suitable image processing boards turned out to be far too costly, it was decided that an attempt should be made at developing an eye-tracking algorithm that was fast enough to run in the CPU of a host PC without assistance from an external processor. To supply the necessary test images, a small frame-grabber was purchased that received images from an ordinary video camera (cf. Section 1.4 below). Thus a revised problem definition, hereafter referred to as the *algorithmic problem definition*, or simply the *problem definition*, was found to be necessary:

**Algorithmic problem definition:** Design, implement and, as extensively as possible, test an algorithm to, on the basis of a given image of the eye, determine the position of the (centre of the) pupil, relative to some given reference point. The algorithm must to be fast enough to run in the CPU of the computer, and has to satisfy the following requirements:

- The position of the pupil is to be determined with an accuracy corresponding to 1 pixel in the digitized image.
- Whenever the pupil cannot be found (due to blinking, disturbances, etc.), an error signal is to be given.
- The total running time of the algorithm may not exceed 20 ms in the worst case.

Whether the accuracy requirement above complies with requirement 3 in Section 1.2 depends on the resolution of the image and on the fraction of the image occupied by the pupil. Preferably, the image should have a resolution of $512 \times 512$ pixels with 256 gray levels, and the eye should fill the entire image. Also, it has to be determined whether the final algorithm actually is to run in the CPU of the host PC, since data transfer rates may make it necessary to have it run in a processor near to the storage device to which the digitized images are written.

## 1.4 Thesis Environment

All computational work was carried out under MS-DOS 5.0 on an i386 IBM compatible PC running at 40 MHz. All time measures given throughout the paper were achieved on this computer. The eye-tracking algorithm OCTOPUS presented in Chapter 5 was implemented solely in Pascal using Borland Turbo-Pascal 6.0. Versions were administered using a DOS-port of the GNU versioning system RCS. The camera I had at my disposal during the work, and with which all test images were obtained, is an ordinary CCD Sony Handicam, type F550E. The circuit board for the frame-grabber used to digitize the images supplied by the camera was purchased from the German computer magazine `c't`, whereas the components had to be purchased from various vendors.

The editor used to write this paper was Demacs, a DOS-port of GNU Emacs, and the paper was prepared using emTEX, a DOS-port of LaTeX, Leslie Lamport's user-friendly interface to Donald E. Knuth's text-formatting system TEX ([14]). To convert the `.dvi`-output from TEX to printable Postscript, I used Radical Eye Software's `dvips`, and to view the Postscript files I used a DOS-port of Ghostscript, the GNU Postscript previewer. Most of the figures appearing throughout the paper were made using Corel Corporation's CorelDRAW! 3.0; other figures were made using my own tools. Lastly, I employed a row of different UNIX-like tools to make the DOS environment as bearable as possible. It is as they say: "*MS-DOS—can't live with it, can't live without it*".

# Chapter 2

# Background

This chapter consists of two sections. With the problem definition given in Section 1.3 in mind, Section 2.1 defines a number of terms relating to the eye and touches briefly upon its structure, whereas Section 2.2 presents the most frequent approaches to the problem of real-time eye tracking, drawing a conclusion as to why it was chosen to develop an entirely new system, as mentioned in Section 1.2.

## 2.1 The Eye

In this section, an introduction is given to some of the terms relating to the eye, as well as to its structure. Although the subjects may be human as well as monkeys, only the human eye is described, given that the terms in general also apply to the monkey's eye. Also, some, from an image processing point of view, useful visual properties of the eye are described.

### 2.1.1 Structure of the Human Eye

In Fig. 2.1, a horizontal cross section of the human eye is shown. The shape of the eye is nearly spherical, and its average diameter is approximately 20 mm. The eye ball (*optical globe*) is made up of three enclosing membranes: the *cornea* and *sclera*, which together form the outer cover of the eye, the middle *choroid*, and the inner *retina*.

The cornea covers the *iris* and the *pupil*, and consists of a tough, transparent tissue, allowing light to pass through. The rest of the eye is covered by the white sclera, which is continuous with the cornea.

The choroid lies directly below the sclera. This membrane contains a network of blood vessels that serves as a major source of nutrition to the eye. Its coat is heavily pigmented to reduce the amount of extraneous light entering the eye. The part of the choroid belonging to the foremost portion of the eye (its *anterior extreme*) is divided into the iris and the *ciliary body*. The *ciliary muscles* allow the iris to contract or expand to control the amount of light entering the eye through the pupil. The pupil is variable in diameter from approximately 2 mm up to 8 mm, corresponding to an area range of 3–50 $mm^2$. The front of the iris contains the visible pigment of the eye, whereas the back contains a black pigment.

The *lens*, which lies behind the iris, is made up of concentric layers of fibrous cells and is suspended by *ciliary fibres*, attaching it to the ciliary body. It contains between 60 and 70 per cent water, about 6 per cent fat, and more protein than any other tissue in the eye. The shape of the lens is controlled by the tension in the ciliary fibres. To focus on distant objects, the controlling muscles cause the lens to be relatively flattened, whereas they cause the lens to thicken to focus on objects near the eye. Both ultraviolet and infrared light are absorbed well by the lens and, in large amounts, may cause damage to the eye.

The innermost membrane of the eye is the retina, which covers the inside of the backmost portion of the eye. The surface of the retina is covered by light receptors on which an image of the outside world (an object, a scene, etc.) is focused by the lens. There are two classes of light receptors: *cones*

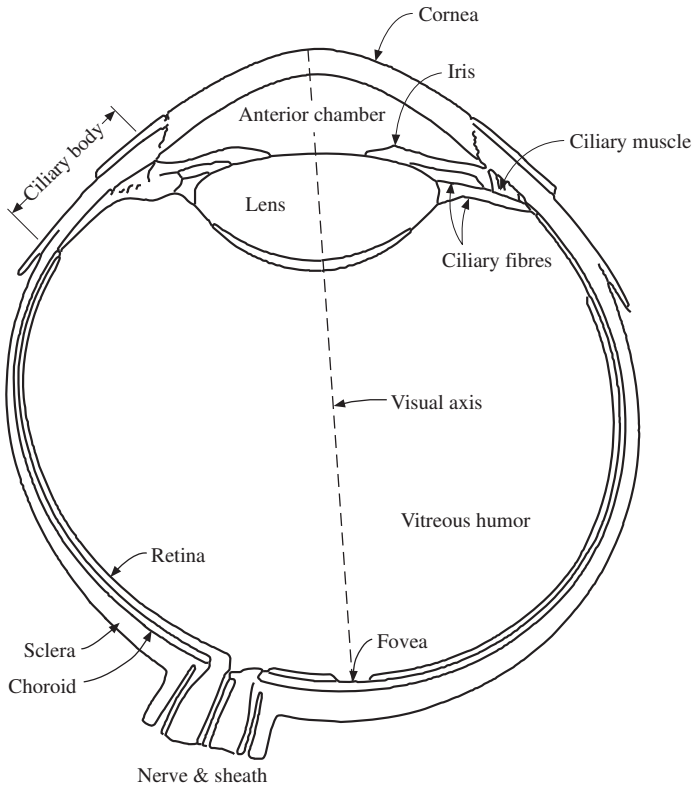



Figure 2.1: A simplified diagram of a cross section of the human eye. (From [12])

and *rods*. The cones, which number between 6 and 7 million and are highly sensitive to colour, are primarily located in the central portion of the retina, the *fovea*. The eye is rotated until the image of the object of interest falls on the fovea. Each cone is connected to its own nerve end, allowing very fine visual resolution. Cone vision is known as *photopic* or bright-light vision.

The rods, which are distributed over the entire retinal surface, number between 75 and 150 million, and serve to give a general, overall picture of the field of view. The larger area of distribution and the fact that several rods are connected to a single nerve end reduce the amount of detail discernible by them. They are insensitive to colour, but much more sensitive to low levels of illumination than the cones. Rod vision is known as *scotopic* or dim-light vision.

### 2.1.2 Optical Properties of the Eye

In Fig. 2.2 is shown a relatively high-quality image of a monkey's eye. From an eye-tracking point of view, the most interesting optical property noted is the relatively sharp transition in brightness between the dark pupil and the comparably bright iris. It is noted, however, that the iris of the monkey appears to be darker than that of man. Note also that the pupil does not constitute the only dark region in the image. In fact, in the given image there appears to be large regions outside the pupil with gray levels in the proximity of those of the pupil. Evidently, an eye-tracking system has to take this into consideration in order to be able to differentiate between the pupil and other dark regions in an image.

Another thing to be noted is that the pupil appears to approximate a circle relatively well. However, although it is known to be circular physically, the appearance of the pupil is circular only when the subject looks straight into the camera. If it looks to the side, the pupil appears as an ellipse whose major axis is vertical; if it looks up or down, the major axis of the pupil ellipse is horizontal; if it looks up or down as well as to the side, the major axis of the pupil ellipse has either a positive or negative angle with respect to the horizontal plane. The deviation in the pupil's appearance from the circular shape depends on the gaze angle relative to the camera. When the angle is relatively small, i.e., in the approximate range $\pm 15^\circ$, the deviation is less than 4%, but

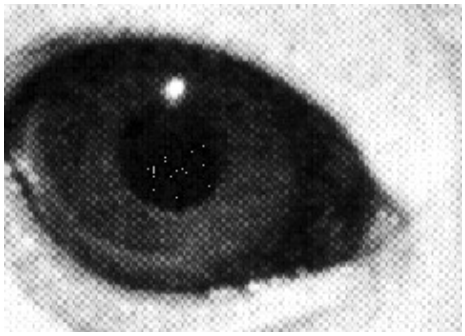


Figure 2.2: An image of a monkey's eye.

when the angle increases, the elliptic appearance becomes more dominant and is apt to cause errors in eye-tracking systems assuming that the pupil always constitutes a circle in an image.

Note also the bright spot at the upper extremity of the pupil. This is called to as the *first Purkinje image*, and constitutes the reflection of the illuminating source from the anterior surface of the cornea.

A last point of interest is that the iris appears brighter under infrared illumination than in visible light ([23]), which is an additional incentive for choosing infrared illumination of the eye for the final eye-tracking system (cf. Section 1.3).

## 2.2 Methods for Measuring Eye Movement

Methods for measuring eye movements fall into four main categories ([15]): *Magnetic induction methods*, *photoelectric methods*, *video techniques* and methods making use of the *optical properties* of the eye.

In this section, these categories are described, along with four different systems, each falling into one of the categories. Section 2.2.1 presents the *search coil* and *double magnetic induction* methods; in Section 2.2.2, a system making use of photoelectric principles is described; Section 2.2.3 introduces the video-based *eye monitor*, and in Section 2.2.4 an optical properties method employing an infrared TV-camera is described.

All systems described require the head of the subject to be fixed. With monkeys, this is less of a problem, since the head can be fixed using a crown attached to the skull. With human subjects, this requirement is harder to satisfy completely, but good results have been achieved using a bite board combined with a tightly fastened crown on the head.

### 2.2.1 Magnetic Induction Methods

The *search coil* technique was first proposed by D. A. Robinson in [18]. In Robinson's setup, the subject wears a short-circuited *induction coil* on a scleral contact lens, and is placed in two perpendicular, rapidly alternating magnetic fields, one horizontal and one vertical. The voltage induced in the coil is related to the position of the eye relative to the horizontal and vertical fields, and can be measured using phase-sensitive detection methods. To facilitate the registration of the induced voltage, the coil is equipped with a set of wires connected to the registering equipment, thus making it rather cumbersome for the subject to wear.

A development of Robinson's method, the so-called *double magnetic induction method*, first proposed by J. Allik *et al.* in [1] and further developed by J. P. H. Reulen and L. Bakker in [17], requires no physical connection between the eye coil and the registering equipment. The idea is to detect the eye position indirectly by determining the strength of the induced magnetic field of the coil through a *detection coil* placed in front of the eye.

In [3], L. J. Bour *et al.* describe a further improvement of the double magnetic induction method. In their setup, the amount of noise present is reduced through the use of a second external coil, the *compensation coil*. This coil is placed in front of the second eye, and is used to cancel out remaining

primary induction voltages, since it is the secondary, induced voltage that is of interest. This is accomplished through differential amplification of the signals from the detection and compensation coils, thus sufficiently cancelling out the primary magnetic field.

### 2.2.2 Photoelectric Methods

Photoelectric eye position measuring methods take advantage of the fact that an image of the eye, when focused on a position sensitive photodiode, induces photocurrents in the diode, which through connectors at different points can be measured, and whose values depend on the image intensities at points surrounding the connectors (i.e., on the position of the pupil within the image).

In [2], M. Bach *et al.* describe a system which measures gaze direction relative to head coordinates by analyzing the centre of gravity (COG) of the first Purkinje image as well as of the entire image of the eye viewed in infrared light. In their setup, the eye is illuminated by a number of infrared LEDs. The LEDs are arranged in an annulus around a lens which focuses the infrared image of the eye at a magnification of about two on a position sensitive photodiode. The position is then measured via the COG of the image falling on the diode. The diode has 5 connections: 2 at opposing edges $(x_1, x_2)$, 2 at the remaining edges $(y_1, y_2)$, and a central one at the backside $(s)$. The $x$- and $y$-coordinates of the COG are given by the partition of the photocurrents $I$ between the electrodes:

$$x = \frac{I_{x_1} - I_{x_2}}{I_s}$$

$$y = \frac{I_{y_1} - I_{y_2}}{I_s}.$$

Although the position of the COG depends in a complex way on gaze direction, it turns out that the resultant voltage output of the system is a linear function of gaze direction.

### 2.2.3 Video-Based Methods

Video-based eye tracking systems analyze the video signals from a camera directly in order to determine the position of the pupil within the image represented by the signals. Video-based systems are widely applicable and easy to use with untrained subjects.

In [15], G. A. Myers, K. R. Sherman and L. Stark present a system for real-time recording of eye movement, called the *eye monitor*. This system takes advantage of the ability of video-based systems to measure the pupil (e.g., area, radius) as part of the basic measurement.

The eye monitor is implemented almost entirely in hardware. It employs an infrared video camera whose pixel clock directly drives a number of counters. Each time the video signal is darker than an adjustable threshold value, one of these counters, C1, is incremented, and thus at the end of each video half frame, the contents of C1 correspond to the pupil area in pixels. This value is divided by two by shifting it one place to the right. It is then loaded into another counter, C2, and during the subsequent half frame, C2 is decremented each time C1 is incremented, so that its contents equal zero when the current video line passes through the equator of the thresholded pupil. The vertical centre of the pupil then corresponds to the number of horizontal lines processed. The horizontal centre of the pupil as well as the horizontal and vertical centres of the first Purkinje image are found in conceptually similar manners. The centre of the first Purkinje image is needed to make corrections for small head movements, as the difference between the pupil centre and that of the first Purkinje image is a first-order invariant with respect to translational head movements.

Since the edges of the video field tend to be dark, they could be mistaken to be parts of the pupil. To avoid this, a circular *tracking window* centred on the current estimate of the pupil position disables the pupil-detection hardware outside the window. The position estimate is updated every half frame, and the area (radius) of the window is also updated to correspond to that of the pupil.

All in all, the eye monitor can be said to employ an internal model of the eye, comprising most of what is known about what the eye looks like to a video camera. The shape of the tracking window (circular) corresponds to the shape of the pupil, the location and size of the window is continuously updated to conform with the last known location and size of the pupil, and the pupil is detected using *area detection*, emphasizing the low-frequency characteristics of the eye's video image, which are less sensitive to noise.

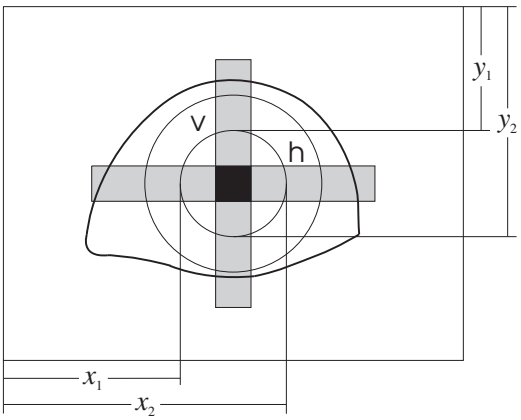



Figure 2.3: The principle of the infrared TV-camera method. v and h designate the vertical and horizontal windows, respectively, and $(x_1, x_2)$ and $(y_1, y_2)$ denote the detected margins of the pupil.

### 2.2.4 Optical Properties Methods

Although the aforementioned methods and systems all take advantage of the optical properties of the eye in determining the centre of the pupil, they have been associated with other categories of systems, corresponding to the manner in which these properties are detected.

This last category really denotes a collection of techniques which all differ from the previous ones in that they perform the analysis in two completely separate steps, of which the first is common for all. The first step consists of acquiring and possibly digitizing and storing (*grabbing*) a complete image of the eye as it appears visually. During the next step, the acquired image is analyzed and the position of the pupil is determined using methods which differ from one system to another. By far the most widely used method during the second step is *image processing*. In Chapter 3, some well-known image processing techniques are presented. Here another approach is described, reported by S. Nagao in [16].

Nagao describes a system for real-time eye-tracking making use of an infrared TV-camera. The eye is illuminated by infrared LEDs, and the camera is used to deliver an image of the eye to a high resolution TV monitor. Two narrow windows, one horizontal and one vertical, are set on the TV screen, covering the margins of the moving pupil (Fig. 2.3). Positions of the left-right and upper-lower margins of the pupil within the two windows are determined by the brightness contrast between the pupil and the iris by means of a slicer device. Due to the camera delivering only half frames, the average left and right margins ($x_1$ and $x_2$, respectively) of the pupil within the horizontal window are measured twice as often as the upper and lower margins ($y_1$ and $y_2$, respectively) within the vertical window. The centre position of the pupil within the TV screen is given by the relations

$$x = \frac{1}{2}(x_1 + x_2)$$

$$y = \frac{1}{2}(y_1 + y_2).$$

### 2.2.5 Summary

The systems discussed in this section each belong to one of the four categories of eye position measuring systems listed in the preamble. Their advantages and disadvantages correspond in large to those of the categories they represent. In this section, an overview is given of the positive and negative aspects of the different approaches in terms of the requirements specified in Section 1.2, and a conclusion is drawn as to which category seems to suit the present needs at best.

### Magnetic Induction Methods

The two magnetic induction methods described, the search coil method ([18]) and double magnetic induction ([1], [17], [3]), display approximately equal properties when it comes to accuracy and real-time abilities. L. J. Bour *et al.* conclude in [3] that the double induction method, as developed by them, is *"useful for the accurate measurement of eye movement in both man and monkey"*. The linear range of this system is $\pm 10°$, and the total measuring range is $\pm 35°$. The calculated error of the system is $\pm 0.1°$, and the eye position is measured with a rate of 500 samples/s, both of which satisfy the given requirements.

Of the two systems, the latter, especially as developed by L. J. Bour *et al.* in [3], is the least discomforting for the subject, in that there are no connecting wires between the search coil and the registering equipment. Still, the subject is required to wear a metallic ring on the eye, causing discomfort and stress. Another negative aspect, and the most important one in our case, is that the alternating magnetic fields would induce unwanted electric currents in the microelectrodes used to perform recordings from the visual cortex of the monkeys and in the EEG electrodes. A last problem with these systems is their price, being in the order of DM 50,000, which by far exceeds the available means.

### Photoelectric Methods

The resolution of the infrared oculometer of M. Bach *et al.* ([2]) is indicated to be $\pm 0.3°$ in the worst case. The highest measuring rate of the oculometer is not reported explicitly, but rates of 85 Hz are mentioned. Both resolution and measuring rate satisfy the requirements. The linear range of the system is reported to be greater than $\pm 20°$, which is twice the range of the double magnetic induction method.

An advantage of photoelectric methods as compared to magnetic induction methods is that they require no surgery, and thus do not interfere directly with the eye. The main disadvantage of the method is that it has to be implemented in hardware, thus leaving little room for adaption to specific needs. Another problem is that most photoelectric systems require time-consuming calibrating, which is not desirable in our case.

As for the magnetic induction systems, the cost of the systems employing photoelectric techniques is prohibitive, being in the order of DM 40,000.

### Video-Based Systems

The resolution of the eye monitor of G. A. Myers *et al.* ([15]) is $\pm 0.2°$ within a range of $\pm 20°$. The measuring rate of the system, as of any video system, is bound to the video refresh rate of the camera. Good cameras operate with refresh rates in the order of 50–60 half frames per second (thus reducing by a factor of 2 the vertical resolution of the system as compared to a system employing a camera delivering entire frames), the camera employed in [15] delivering approximately 60 half frames per second. Thus both resolution and measuring rate satisfy the stated requirements.

The main disadvantages with video-based systems are, as was the case for photoelectric methods, for one their price, and secondly that they are completely hardware dependent, thus preventing customization to specific needs.

### Optical Properties Methods

The measuring rate of eye tracking systems belonging to this category is limited by (1) the refresh rate of the camera used, (2) the grabbing-time of an eventual frame grabber, and (3) the computing time of the position-analyzing algorithm.

The system described by S. Nagao in [16] employs a camera delivering one half frame every 33.3 ms, needs no frame grabbing, as the image is analyzed directly as it appears on the TV screen, and the computing time of the position-analyzing device is constant in time. The measuring rate of the system is thus limited to 30 Hz, as given by the refresh rate of the camera. The spatial resolution of the system depends on the power of the TV-camera employed, and is reported to be $\pm 0.025°$ in the given setup, which adheres to the specified requirements. The measuring rate of the setup,

on the other hand, is too slow for our purpose, but constitutes no real problem, in that it can be improved by choosing a camera with higher refresh rate.

The primary advantage of this system is the superfluousness of time consuming frame grabbing, making the entire time interval between half frames available for the position analyzing device. Thus its only time limiting aspect is the refresh rate of the camera, which can be chosen more or less freely. The need to perform the position-analysis directly on the TV screen, on the other hand, does not give an impression of elegance, and the rigidness of the position-analyzing device constitutes another drawback of the system, making it unsuitable for our purpose.

**Conclusion**

As is evident from the above paragraphs, none of the presented systems fulfil all the stated requirements, their costs being the most prohibitive factor. Magnetic induction methods are unsuited because the magnetic fields they employ induce unwanted currents in the experimental registering equipment. Photoelectric methods as well as video-based methods were excluded from the set of possible candidates because of their strict hardware nature, violating the requirement of configurability. The only category which has not been not completely excluded, although the one system belonging to it was, is the category of optical properties methods.

The big advantage of image processing based optical properties systems as opposed to the approach used by Nagao in his system system is that they, being implemented mainly in software, offer easy customization and adaption to specific needs. In fact, the more that is implemented in software, the easier the customization of the system. One disadvantage of this kind of system, is that the video frames have to be grabbed before they can be analyzed, this constituting a competitive factor on valuable processor time. Having two or more processors running in parallel (e.g., Transputers), one of which being responsible for grabbing the frames, would pretty much eliminate this problem.

There already exists at least one commercially available and good eye tracking system based on a video camera delivering video frames to a computer performing the position-analysis ([10]). Its price, though, is prohibitive (∼DM 150,000), due to old and therefore expensive technology being employed. The simplicity of this system, on the other hand, suggests that it ought to be possible to develop a similar system for only a fraction of the costs, employing new and cheaper technology.

Based on this, it was decided that an entirely new image processing and software based eye tracking system should be developed, adhering as closely as possible to the requirements listed in Section 1.2.

# Chapter 3

# Digital Image Processing

## 3.1 Introduction

### 3.1.1 Intent

The intent of this chapter is to give a relatively broad presentation of different techniques that may be applied to solve the problem defined by the algorithmic problem definition in Section 1.3. The presentation is intended to indicate some possible approaches and thus to form a theoretical foundation on which to base the development of a solution as complete and satisfactory as possible.

Because of the requirement of speed, the techniques presented have been selected on the basis of being relatively cheap computationally. Most of them are also well-established, in that they are thoroughly described in the literature. Many reliable, but, due to relatively complex mathematics, computational expensive methods, such as the Canny/Deriche approach to edge detection ([5], [6]), are not discussed.

In order to maintain an overall view, the presentation is kept on a relatively fundamental level. Still, it has been aimed at making it thorough enough to suffice as a basis on which to draw a conclusion as to a suitable approach to the actual problem.

### 3.1.2 Categories in Image Processing

The general field of automated image processing may be divided into four main categories. These categories are ([12]):

**Image digitization:** In order to be in a form suitable for processing in a computer, an image has to be discretized (digitized) both spatially and in amplitude. *Image digitization* is the process of converting continuous (real) images into digitized form.

**Image enhancement and restoration:** Often an image is unsuited for machine and/or human perception, due to its being blurred, noisy, or in some other way degraded. The field of improving degraded images for machine or human perception is generally referred to as *image enhancement and restoration*.

**Image encoding:** Techniques for representing an image or the information contained in the image with fewer bits than required for the "raw" digitized image are categorized as *image encoding* techniques.

**Image segmentation, representation, and description:** An image is normally composed of contiguous regions, each region being characterized by a set of properties that the pixels belonging to the region share. These regions correspond to the objects that make up the image. Often, one is interested in segmenting out one or more of the objects that are present in an image. This can be done by applying techniques classified as *image segmentation* techniques. In addition, one may be interested in representing the constituent parts of an image in forms suitable for further computer processing, and to describe them in terms of parts and properties. This is referred to as *image representation* and *description*, respectively.

The field of *digital* image processing is concerned with the latter three of these categories, in that already digitized images are required as input to a computer. In this chapter, the focus is on the fields of *noise reduction*, and *image segmentation*, as described in the next section.

### 3.1.3 Structure of the Chapter

In this chapter, both spatial and frequency domain techniques are presented. A theoretical introduction to the spatial domain and terms pertaining to it is given in Section 3.2. The theoretical foundations of frequency domain techniques, the Fourier transform and the convolution theorem, are given a relatively extensive presentation in Section 3.3, along with a basic description of their use in image processing.

A usual problem when processing digital images, is that they often have been subjected to some sort of noise. In Section 3.4, some approaches to the problem of reducing the amount of noise present in a given image are presented.

Since the problem at hand is one of recognizing and locating a given object in an image (the pupil), the problem of decomposing an image into constituent regions representing different objects is elaborated upon in Section 3.5.

A branch of the field of image segmentation is the field that is concerned with locating discontinuities in an image, such as lines and edges. Since an obvious approach to locating the pupil in an image is to try to locate the transition in brightness between the pupil and the iris, the field of edge detection is given separate treatment in Section 3.6.

#### References

Most of the material in this chapter is from [12]. Another main source is [19]. The material in the section on template matching (Section 3.5.3) is mainly from [20] and [23], the former having supplied additional material to other sections of the chapter as well. A last source for additional material has been [13].

## 3.2 The Spatial Domain

The term *spatial domain* refers to the entire aggregate of pixels making up an image. Spatial domain methods are methods that operate directly on these pixels.

### 3.2.1 General Overview

An image in the spatial domain can be described by the two-dimensional light-intensity function $f(x, y)$ whose value at location $(x, y)$ corresponds to the intensity (or *gray level*) of the image at that point. Image-processing functions in the spatial domain may be expressed as

$$g(x, y) = T[f(x, y)], \tag{3.1}$$

where $g(x, y)$ is the processed image and $T$ is an operator on the input image $f$, defined over some neighbourhood of $(x, y)$. In other words, the value of $g$ at $(x, y)$ is determined by the values of $f$ in this neighbourhood.

The principle approach used in defining a neighbourhood about $(x, y)$ is to use an $n \times n$ subimage of $f$ centred at $(x, y)$, as shown in Fig. 3.1. Other shapes are also possible, but square arrays are by far the most predominant, due to their ease of implementation. The centre of the subimage is moved from pixel to pixel, applying the operator at each location $(x, y)$ in $f$. Thus it is seen that applying a spatial operator to an $N \times N$ image generally requires $O(N^2)$ operations.

When the size $n$ of the square neighbourhood is 1, $T$ reduces to a gray-level *transformation function* where the value of $g$ at $(x, y)$ only depends on the value of $f$ at this location. One widely used gray-level transformation function is

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) \in Z \\ 0 & \text{otherwise,} \end{cases} \tag{3.2}$$

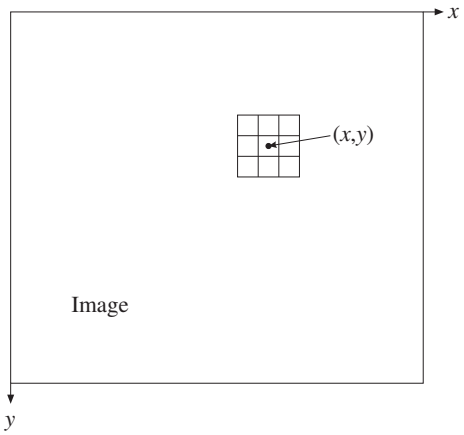



Figure 3.1: A $3 \times 3$ neighbourhood about a point $(x, y)$ in an image.

where $Z$ is a defined set of distinct gray levels and 0 and 1 denote black and white, respectively (typically represented by gray levels 0 and 255). This is a general formulation of what is called *thresholding*, elaborated further upon in Section 3.5.1.

### 3.2.2 Spatial Masks

One of the principle approaches to the formulation in Eq. (3.1) is the use of so-called *spatial masks*. A spatial mask, or just *mask*, is a small two-dimensional array ($m \times n$ or, more commonly, $n \times n$) whose elements $w_i$ are coefficients used to compute $g$ at $(x, y)$. In Fig. 3.2, a $3 \times 3$ sub-image centred at $(x, y)$ is shown, together with a $3 \times 3$ mask with general coefficients $w_i$. When *convolving* the original image $f$ with this mask (cf. Section 3.3.2, p. 22), the value at location $(x, y)$ of the resulting image $g$ is given by the relation

$$g(x, y) = T[f(x)] = w_1 f(x-1, y-1) + w_2 f(x, y-1) + \cdots + w_9 f(x+1, y+1). \qquad (3.3)$$

To compute the transformed image $g$, the mask is moved from pixel to pixel in the same manner as for the neighbourhood operator shown in Fig. 3.1. As pointed out above, if $f$ is of size $N \times N$, the number of operations required to compute $g$ is $O(N^2)$.

The coefficients $w_i$ of the mask are chosen to detect given properties in an image. For example, if all $w_i$ are chosen to be $1/9$, the gray level of each pixel in the resulting image $g$ would be the average of the the gray levels of the corresponding pixel in $f$ and its 8 immediate neighbours (*8-neighbours*). This is known as *neighbourhood averaging* (cf. Section 3.4.1), and the effect is one of *blurring* the original image.

Neighbourhood averaging is only one of several possible mask operations. By properly selecting the coefficients, it is possible to perform a variety of useful image operations, among which are *noise reduction* (Section 3.4) and *edge detection* (Section 3.6). It is worth noting, however, that applying a mask at each pixel location in an image is a computationally expensive task. From Eq. (3.3), it is seen that for $3 \times 3$ masks, nine multiplications and eight additions are required to compute $g$ at a given location $(x, y)$. Thus, convolving a $512 \times 512$ image with a $3 \times 3$ mask like the one in Fig. 3.2(b) requires a total of 2,359,296 multiplications and 2,097,152 additions.

## 3.3 The Frequency Domain

Another approach to digital image processing is to manipulate the *Fourier transform* of the image instead of the image itself. This is commonly referred to as analyzing the image in the *frequency domain*. Below, a short introduction is given to the Fourier transforms of one- and two-dimensional functions, starting with the continuous case and then extending the formulation to include the discrete case, which, in the two-dimensional case, corresponds to digitized images.

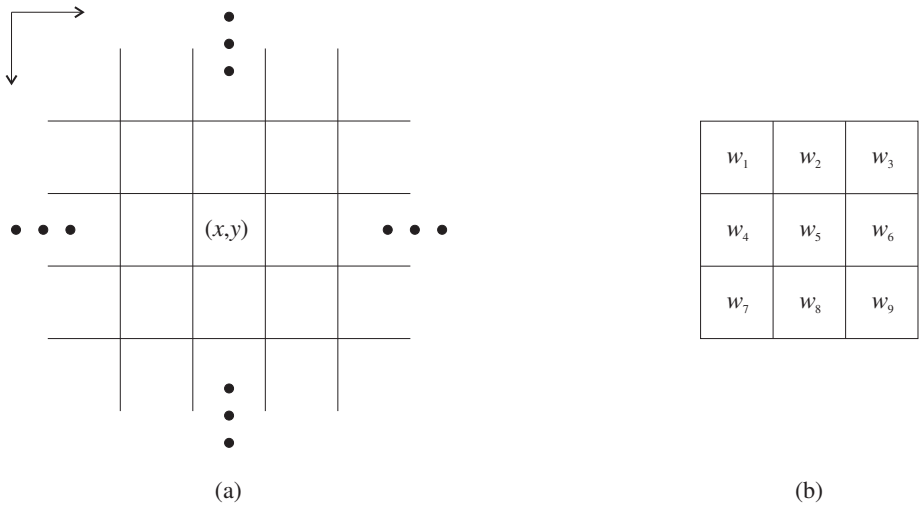



Figure 3.2: (a) Sub-area of image. (b) A $3 \times 3$ mask with general coefficients. (From [12])

### 3.3.1 The Fourier Transform

The formulation of the Fourier transform of a function $f$ depends on whether the function is one- or two-dimensional, and on whether it is a continuous or discrete function.

#### The Continuous Fourier Transform

The general definition of the Fourier transform is as follows. Let $f(x)$ be a continuous function of a real (as opposed to complex) variable $x$. The Fourier transform of $f(x)$, denoted $\mathcal{F}\{f(x)\}$, is then defined by the equation

$$\mathcal{F}\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x)e^{-j2\pi ux}dx, \tag{3.4}$$

where $j = \sqrt{-1}$. When $F(u)$ is known, $f(x)$ can be obtained using the *inverse Fourier transform*, denoted $\mathcal{F}^{-1}\{F(u)\}$, and defined by the equation

$$\mathcal{F}^{-1}\{F(u)\} = f(x) = \int_{-\infty}^{\infty} F(u)e^{j2\pi ux}du. \tag{3.5}$$

Eqs. (3.4) and (3.5) are called a *Fourier transform pair*. The Fourier transform and its inverse can be shown to exist if $f(x)$ is continuous and integrable and $F(u)$ is integrable.

The Fourier transform of a real function is generally complex:

$$F(u) = R(u) + jI(u), \tag{3.6}$$

where $R(u)$ and $I(u)$ are the real and imaginary parts of $F(u)$, respectively. It is often convenient to express Eq. (3.6) in exponential form:

$$F(u) = |F(u)|e^{j\phi(u)}, \tag{3.7}$$

where

$$|F(u)| = \sqrt{R^2(u) + I^2(u)} \tag{3.8}$$

and

$$\phi(u) = \tan^{-1}\left[\frac{I(u)}{R(u)}\right]. \tag{3.9}$$

The magnitude function $|F(u)|$ is called the *Fourier spectrum* of $f(x)$, and $\phi(u)$ its *phase angle*. The square of the Fourier spectrum,

$$P(u) = |F(u)|^2 = R^2(u) + I^2(u) \tag{3.10}$$

is called the *power spectrum* of $f(x)$.

The term frequency domain stems from the fact that the exponential term of Eq. (3.4) can be expressed in the form

$$e^{-j\pi ux} = \cos 2\pi ux - j \sin 2\pi ux, \tag{3.11}$$

thus indicating that $F(u)$ is composed of an infinite sum of sine and cosine terms, the frequency of which being determined by the *frequency variable* $u$.

The general two-dimensional Fourier transform pair is analogously defined:

$$\mathcal{F}\{f(x,y)\} = F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)}dxdy \tag{3.12}$$

and

$$\mathcal{F}^{-1}\{F(u,v)\} = f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{j2\pi(ux+vy)}dudv, \tag{3.13}$$

$u$ and $v$ being the frequency variables.

The Fourier spectrum, phase and power spectrum in the two-dimensional case are defined analogously to Eqs. (3.8), (3.9), and (3.10):

$$|F(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)} \tag{3.14}$$

$$\phi(u,v) = \tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right] \tag{3.15}$$

and

$$P(u,v) = |F(u,v)|^2 = R^2(u,v) + I^2(u,v). \tag{3.16}$$

**The Discrete Fourier Transform**

Since we are dealing with digitized images consisting of $M \times N$ pixels having discrete gray levels, it is necessary to reformulate the Fourier transform for the discrete case. The principle is first introduced for the one-dimensional case, and thereafter extended to include the two-dimensional case.

When a continuous function $f(x)$ is discretized into a sequence $\{f(x_0), f(x_0 + \Delta x), f(x_0 + 2\Delta x), \ldots, f(x_0 + (N-1)\Delta x)\}$ by taking $N$ samples $\Delta x$ units apart, its *discrete* Fourier transform is given by the relation[1]

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)e^{-\frac{j2\pi ux}{N}} \tag{3.17}$$

for $u = 0, 1, 2, \ldots, N-1$, where $f(x)$ is given by

$$f(x) = f(x_0 + x\Delta x). \tag{3.18}$$

Analogously, the inverse discrete Fourier transform is given by

$$f(x) = \sum_{u=0}^{N-1} F(u)e^{\frac{j2\pi ux}{N}} \tag{3.19}$$

for $x = 0, 1, 2, \ldots, N-1$.

The discrete Fourier transform pair of a two-dimensional function $f(x,y)$ whose one variable $x$ has been discretized by taking $M$ samples $\Delta x$ units apart and whose second variable $y$ has been discretized taking $N$ samples $\Delta y$ units apart is defined analogously to the one-dimensional case above:

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{3.20}$$

[1] For a proof of these results, see for instance [4].

for $u = 0, 1, 2, \ldots, M-1$, $v = 0, 1, 2, \ldots, N-1$, and

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi(\frac{ux}{M} + \frac{vy}{N})} \tag{3.21}$$

for $x = 0, 1, 2, \ldots, M-1$ and $y = 0, 1, 2, \ldots, N-1$.

The Fourier spectrum, phase and power spectrum of the two-dimensional discrete function are given by Eqs. (3.14), (3.15) and (3.16), respectively. The only difference is that the independent variables are discrete.

**The Fast Fourier Transform**

When implementing the discrete Fourier transform, it is important that the number of operations be kept as low as possible in order to reduce computing time. It can be shown that the formulation in Eq. (3.17) requires $O(N^2)$ operations, $N$ being the number of samples used to discretize the original continuous function. The *fast Fourier transform* (FFT) is an algorithm which, by properly decomposing Eq. (3.17), reduces the number of operations needed to $O(N \log_2 N)$. The formulation given here is from [12], but is by no means a unique formulation. For a comprehensive discussion of a number of formulations for the FFT, including this one, see [4].

It will be convenient in the following discussion to express Eq. (3.17) in the form

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) W_N^{ux}, \tag{3.22}$$

where

$$W_N = e^{-\frac{j2\pi}{N}}, \tag{3.23}$$

and $N = 2^n$, $n > 0$. Based on this, $N$ can be expressed as

$$N = 2M. \tag{3.24}$$

Substitution of Eq. (3.24) into Eq. (3.22) yields

$$\begin{aligned} F(u) &= \frac{1}{2M} \sum_{x=0}^{2M-1} f(x) W_{2M}^{ux} \\ &= \frac{1}{2} \left\{ \frac{1}{M} \sum_{x=0}^{M-1} f(2x) W_{2M}^{u(2x)} + \frac{1}{M} \sum_{x=0}^{M-1} f(2x+1) W_{2M}^{u(2x+1)} \right\}. \end{aligned} \tag{3.25}$$

From Eq. (3.23) we have that $W_{2M}^{2ux} = W_M^{ux}$. Eq. (3.25) may then be expressed as

$$F(u) = \frac{1}{2} \left\{ \frac{1}{M} \sum_{x=0}^{M-1} f(2x) W_M^{ux} + \frac{1}{M} \sum_{x=0}^{M-1} f(2x+1) W_M^{ux} W_{2M}^{u} \right\}. \tag{3.26}$$

If we define

$$F_{\text{even}}(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(2x) W_M^{ux} \tag{3.27}$$

$$F_{\text{odd}}(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(2x+1) W_M^{ux} \tag{3.28}$$

for $u = 0, 1, 2, \ldots, M-1$, Eq. (3.26) becomes

$$F(u) = \frac{1}{2} \{ F_{\text{even}}(u) + F_{\text{odd}}(u) W_{2M}^{u} \}. \tag{3.29}$$

Also, since $W_M^{u+M} = W_M^u$ and $W_{2M}^{u+M} = -W_{2M}^u$, it follows from Eqs. (3.27) through (3.29) that

$$F(u+M) = \frac{1}{2} \{ F_{\text{even}}(u) - F_{\text{odd}}(u) W_{2M}^{u} \}. \tag{3.30}$$

The essence of the FFT algorithm lies in the observation that the transform of an $N$-point function can be computed by dividing the original expression into two parts, as done in Eqs. (3.25) and (3.26). Note that the interval of $u$ for which these equations are defined, $[0, M-1]$, corresponds to the first half of the original interval $[0, N-1]$. Computation of $F(u)$ within this interval is thus performed by computing the two $N/2$-point transforms $F_{\text{even}}(u)$ and $F_{\text{odd}}(u)$, as defined in Eqs. (3.27) and (3.28), and then applying Eq. (3.29) to the results to form $F(u)$ for $u$ in $[0, M-1]$. The values of $F(u)$ for $u$ in $[M, N-1]$ follow directly from Eq. (3.30). Computation of $F_{\text{even}}(u)$ and $F_{\text{odd}}(u)$ is carried out in the exact same manner, by dividing each of them into two and proceeding as described above. This process is carried on recursively downwards until the transforms to be computed are 1-point transforms, which correspond to the samples themselves. These are then promoted upwards to form intermediate results on all levels, until finally, on the top level, the overall transform has been computed. This process is referred to as *successive doubling*. The name arises from the fact that a two-point transform is computed from two one-point transforms, a four-point transform from two two-point transforms, and so on.

The inverse Fourier transform is easily implemented using the FFT algorithm described above. To see this, consider Eqs. (3.17) and (3.19), which are repeated below:

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-\frac{j2\pi ux}{N}} \tag{3.31}$$

$$f(x) = \sum_{u=0}^{N-1} F(u) e^{\frac{j2\pi ux}{N}}. \tag{3.32}$$

Taking the complex conjugate of Eq. (3.32) and dividing both sides by $N$ yields

$$\frac{1}{N} f^*(x) = \frac{1}{N} \sum_{u=0}^{N-1} F^*(u, v) e^{-\frac{j2\pi ux}{N}}. \tag{3.33}$$

The right side of Eq. (3.33) is in the form of the forward Fourier transform in Eq. (3.31). Thus, if $F^*(u)$ is input into an algorithm computing the forward transform, the result will be the quantity $f^*(x)/N$. Taking the complex conjugate and multiplying by $N$ yields the inverse Fourier transform $f(x)$.

As mentioned above, this algorithm performs in $O(N \log_2 N)$ time, the number of complex multiplications and additions being $\frac{1}{2} N \log_2 N$ and $N \log_2 N$, respectively. For a proof of these results, see for instance [4] or [12].

Note that this formulation is for the one-dimensional case. The FFT in the two-dimensional case is computed by applying the above formulation on a row-to-row or column-to-column basis. The number of complex operations for applying the FFT on an $N \times N$ image is thus $O(N^2 \log_2 N)$. Since each operation involves the complex exponential function, which is normally implemented using the relation $e^{x+jy} = e^x \cos y + j e^x \sin y$, it can be concluded that the FFT, although cheaper than the direct implementation of Eq. (3.17), still is comparably expensive.

### 3.3.2 The Convolution Theorem

The foundation of frequency domain image processing techniques is the *convolution theorem*. As was the case for the Fourier transform, the theorem is presented in two steps. First, the general formulation of the theorem is stated for one- and two-dimensional continuous functions, then this formulation is extended to include the discrete case.

#### Convolution in the Continuous Case

The *convolution* of two continuous one-dimensional functions $f(x)$ and $g(x)$, denoted $f(x) * g(x)$, is defined by the integral

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\alpha) g(x - \alpha) d\alpha, \tag{3.34}$$

where $\alpha$ is a dummy variable of integration. The mechanics of the convolution integral is not easy to visualize, so the best way is to illustrate it graphically with a simple example (from [12]):

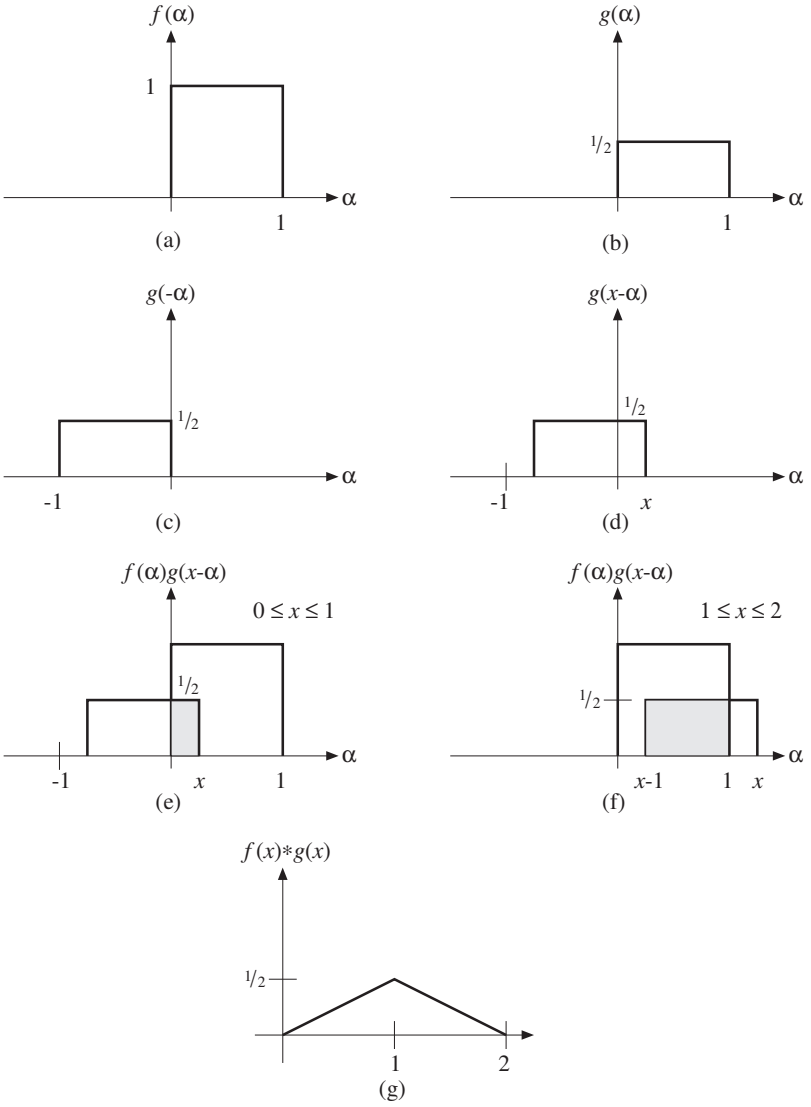



Figure 3.3: Graphical illustration of convolution.

**Example:** Consider Eq. (3.34) as applied to the two functions $f(x)$ and $g(x)$ shown in Figs. 3.3(a) and (b), respectively. Before the integration can be carried out, it is necessary to form the function $g(x-\alpha)$. This is accomplished by mirroring $g(\alpha)$ about the vertical axis to give $g(-\alpha)$ and then displacing this function by $x$, as shown in Figs. 3.3(c) and (d). Then, for all values of $x$, $f(x)$ is multiplied with $g(x-\alpha)$, and the result (the shaded portion of Fig. 3.3(e)) is integrated from $-\infty$ to $\infty$. Since this product is 0 for values of $\alpha$ outside the interval $[0, x]$, $f(x) * g(x)$ is found to be $x/2$ for $0 \leq x \leq 1$. This corresponds to the shaded region of Fig. 3.3(e). Fig. 3.3(f) illustrates the same process for $1 \leq x \leq 2$. In this case, $f(x) * g(x) = (1 - x/2)$. As $f(\alpha)g(x-\alpha)$ is zero for $x < 0$ and $x > 2$, we have that the convolution of $f(x)$ with $g(x)$ is given by

$$f(x) * g(x) = \begin{cases} x/2 & 0 \leq x \leq 1 \\ 1 - x/2 & 1 \leq x \leq 2 \\ 0 & \text{elsewhere,} \end{cases} \tag{3.35}$$

as shown in Fig. 3.3(g). □

The importance of convolution in frequency domain analysis stems from the fact that $f(x) * g(x)$ and $F(u)G(u)$ form a Fourier transform pair. In other words, if $f(x)$ has the Fourier transform

$F(u)$ and $g(x)$ has the Fourier transform $G(u)$, then the following relations hold:

$$\mathcal{F}\{f(x) * g(x)\} = F(u)G(u) \tag{3.36}$$

$$\mathcal{F}^{-1}\{F(u)G(u)\} = f(x) * g(x). \tag{3.37}$$

An analogous result is that convolution in the frequency domain reduces to multiplication in the spatial domain:

$$\mathcal{F}\{f(x)g(x)\} = F(u) * G(u) \tag{3.38}$$

$$\mathcal{F}^{-1}\{F(u) * G(u)\} = f(x)g(x). \tag{3.39}$$

These two results, formally stated as

$$f(x) * g(x) \Leftrightarrow F(u)G(u) \tag{3.40}$$

$$f(x)g(x) \Leftrightarrow F(u) * G(u), \tag{3.41}$$

are commonly referred to as the *convolution theorem*.

The convolution of two two-dimensional functions $f(x, y)$ and $g(x, y)$ is defined analogously to Eq. (3.34):

$$f(x, y) * g(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha, \beta)g(x - \alpha, y - \beta)d\alpha d\beta. \tag{3.42}$$

The two-dimensional convolution theorem is given by the relations

$$f(x, y) * g(x, y) \Leftrightarrow F(u, v)G(u, v) \tag{3.43}$$

$$f(x, y)g(x, y) \Leftrightarrow F(u, v) * G(u, v). \tag{3.44}$$

For a graphical illustration of two-dimensional convolution, see [12], p. 89.

### Convolution in the Discrete Case

The two-dimensional *discrete* convolution is formulated by letting $f(x, y)$ and $g(x, y)$ be discrete arrays of size $A \times B$ and $C \times D$. For reasons elaborated upon in [4], these arrays must be assumed to be periodic with some periods $M$ and $N$ in the $x$- and $y$-directions, respectively. It can be shown ([4]) that unless

$$M \geq A + C - 1 \tag{3.45}$$

$$N \geq B + D - 1, \tag{3.46}$$

the individual periods of the convolution will overlap. This overlap is commonly referred to as *wraparound error*. The periodic sequences are formed by extending $f(x, y)$ and $g(x, y)$ as follows:

$$f_e(x, y) = \begin{cases} f(x, y) & 0 \leq x \leq A - 1 \quad \text{and} \quad 0 \leq y \leq B - 1 \\ 0 & A \leq x \leq M - 1 \quad \text{or} \quad B \leq y \leq N - 1 \end{cases} \tag{3.47}$$

$$g_e(x, y) = \begin{cases} g(x, y) & 0 \leq x \leq C - 1 \quad \text{and} \quad 0 \leq y \leq D - 1 \\ 0 & C \leq x \leq M - 1 \quad \text{or} \quad D \leq y \leq N - 1. \end{cases} \tag{3.48}$$

With these definitions in mind, the discrete two-dimensional convolution of $f(x, y)$ and $g(x, y)$ is expressed in terms of $f_e(x, y)$ and $g_e(x, y)$:

$$f_e(x, y) * g_e(x, y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f_e(m, n)g_e(x - m, y - n) \tag{3.49}$$

for $x = 0, 1, 2, \ldots, M - 1$ and $y = 0, 1, 2, \ldots, N - 1$.

The two-dimensional convolution theorem stated in Eqs. (3.43) and (3.44) also applies to the discrete case with $u = 0, 1, 2, \ldots, M - 1$ and $v = 0, 1, 2, \ldots, n - 1$.

All computations involve, as indicated above, the extended functions $f_e(x, y)$ and $h_e(x, y)$. It is worth noting, however, that in practice, the energy of an image tends to be concentrated around the origin of the frequency plane. This reduces the wraparound error due to overlap to a non-discernible minimum, thus obliterating the need to use $f_e$ and $g_e$ in most practical applications.

#### Application of Convolution in Digital Image Processing

As mentioned above, the convolution theorem forms the basis of image processing in the frequency domain. Let $g(x, y)$ be an image which is the result of convolving an image $f(x, y)$ with a position-invariant operator $h(x, y)$[2]:

$$g(x, y) = h(x, y) * f(x, y). \tag{3.50}$$

From the convolution theorem (Eq. (3.43)) we then have that the following relation holds in the frequency domain:

$$G(u, v) = H(u, v)F(u, v), \tag{3.51}$$

where $G$, $H$ and $F$ are the Fourier transforms of $g$, $h$ and $f$, respectively. The function $H(u, v)$ is often referred to as the *transfer function* of the process.

A typical image-processing problem can be stated as Fourier-transforming the original image $f$, yielding $F$, and then trying to choose a transfer function $H$ such that the resulting image

$$g(x, y) = \mathcal{F}^{-1}\{H(u, v)F(u, v)\} \tag{3.52}$$

exhibits some desired property.

#### Spatial Masks as Convolution Masks

From a practical point of view it is often more convenient to compute the discrete convolution of an image $f$ with an operator $h$ in the frequency domain instead of using Eq. (3.49) directly. This is done by computing the Fourier transforms $F$ and $H$ using an FFT algorithm, multiplying the Fourier transforms to obtain $G$, and then applying the inverse Fourier transform to obtain the desired image $g$. However, in [4] Brigham shows that for one-dimensional arrays, the FFT approach is faster only if the number of points contributing to the operator is greater than 32. Thus, when applying small operators, it would be convenient to be able to perform the convolution in the spatial domain.

A useful observation in that respect is that the convolution of an image $f$ with an operator $h$, as expressed by Eq. (3.50), describes a spatial process analogous to the use of masks described in Section 3.2. In fact, the discrete convolution as expressed by Eq. (3.49) represents a mathematical description of the mask-shifting process illustrated in Fig. 3.1. On pp. 186-190 in [12], a method is described for obtaining small spatial masks that approximate a given $H(u, v)$ in a least-square-error sense. For this reason, spatial masks are often referred to as *spatial convolution masks*, and the process of applying a mask to an image is described as *convolving* the image with the mask.

### 3.3.3 Image Processing in the Frequency Domain

In the Fourier transform $F$ of an image $f$ the high frequency component is to a large extent caused by edges and other sharp transitions in the gray levels of the image (such as noise). Correspondingly, the low frequency component of $F$ corresponds to continuous regions in $f$ with little or no gray level transitions. Often, one is interested in only the high-frequency or low-frequency components of a given image. This can be accomplished in both cases by attenuating the opposite end of the frequency spectrum. This is known as *highpass* and *lowpass filtering*, respectively.

#### Lowpass Filtering

Lowpass filtering is achieved by convolving the original image $f$ with an operator $h$ whose Fourier transform $H$ has the property of either completely (*ideal* filter) or partly (*Butterworth* filter) filtering out the low frequency component of $F$. The resulting image, given by

$$g(x, y) = \mathcal{F}^{-1}\{G(u, v)\} = \mathcal{F}^{-1}\{H(u, v)F(u, v)\}, \tag{3.53}$$

then exhibits the desired properties. Cross sections of an ideal lowpass filter and a lowpass Butterworth filter are shown in Fig. 3.4, $D(u, v)$ being the distance from the point $(u, v)$ to the origin of the frequency plane and $D_0$ being the *cut-off frequency*.

[2] A position-invariant operator is an operator whose result depends only on the value of $f(x, y)$ at a given point in the image, not on the location of the point.

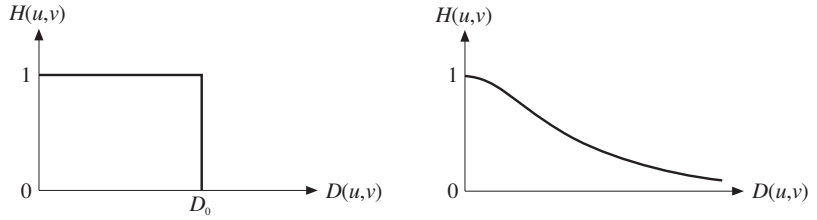



Figure 3.4: Cross section of an ideal lowpass filter (a) and a lowpass Butterworth filter (b).

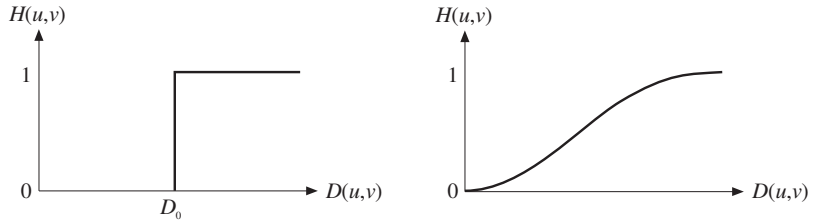



Figure 3.5: Cross section of an ideal highpass filter (a) and a highpass Butterworth filter (b).

Since the high-frequency component of an image corresponds to edges and other abrupt gray level transitions, lowpass filtering has the effect of blurring the original image. On the other hand, if the image has been distorted by noise, which is also high frequency in content, an additional effect is one of noise reduction (cf. Section 3.4.3).

### Highpass Filtering

Highpass filtering is carried out analogously to lowpass filtering, using either an ideal highpass filter or a highpass Butterworth filter, as shown in Fig. 3.5.

If the low-frequency content of an image is severely attenuated, the resulting image will be one in which edges between areas of different gray levels are predominant. If the intent is to sharpen the edges without loss of low-frequency information, a constant $c$ is often added to the highpass filter transfer function, as shown in Fig. 3.6, by choosing $\gamma_L = c$ and $\gamma_H = 1 + c$. This is known as *high frequency emphasis*.

High frequency emphasis is actually a special case of what is known as *homomorphic filtering*. Homomorphic filtering is based on an image model saying that an image $f(x, y)$ is composed of an *illumination component* $i(x, y)$ and a *reflection component* $r(x, y)$ according to the equation

$$f(x, y) = i(x, y)r(x, y). \tag{3.54}$$

The illumination component of an image is generally characterized by slow spatial variation, and is thus low frequency in content. The reflection, on the other hand, depends on the nature of

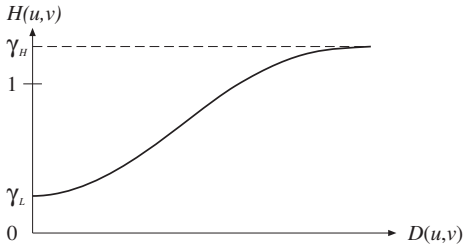



Figure 3.6: A homomorphic filter transfer function.

the objects in image and tends to vary abruptly, thus being high frequency in content. Assuming that $\gamma_L$ and $\gamma_H$ are chosen so that $\gamma_L < 1$ and $\gamma_H > 1$, the filter function in Fig. 3.6 affects the low and high frequency components of an image differently, attenuating the low frequencies and amplifying the high frequencies. The net result is simultaneous dynamic range compression and contrast enhancement. On pp. 192-193 in [19] is shown the result of applying the filter in Fig. 3.6 with $\gamma_L = 0.5$ and $\gamma_H = 2.0$ to an image with little contrast.

## 3.4 Noise Reduction

Noise reduction, or *smoothing*, is the process of reducing or, if possible, removing unwanted spurious effects (noise) that may be present in an image. Noise in a digital image may stem from several sources, such as a bad transmition channel or a poor sampling system. In this section, noise reduction techniques both in the spatial and frequency domains are considered.

Whether noise reduction measures have to be taken in the final implementation of the complete eye-tracking system is not clear at present. By ensuring optimal light conditions and using "clean" transmission channels, relatively noiseless images ought to be obtainable. This section is meant as an indicator towards some possible noise reduction schemes if it at a later stage of the project should turn out that the noise present in the supplied images cannot be ignored.

### 3.4.1 Averaging

*Averaging* is a general technique with which the value at a given pixel location in an image is replaced by the average value of a set of pixels, either from the same image, or from a set of images.

#### Neighbourhood Averaging

In *neighbourhood averaging*, the value at a given pixel location is replaced by the average value of the pixel in a defined neighbourhood of the location. The spatial mask proposed in Section 3.2.2 corresponds to neighbourhood averaging where the neighbourhood is defined to be the 8-neighbours of the given location plus the location itself. The general formulation of neighbourhood averaging of an $N \times N$ image $f$ is

$$g(x,y) = \frac{1}{M} \sum_{(n,m)\in S} f(n,m) \tag{3.55}$$

for $x, y = 0, 1, 2, \ldots, N-1$, where $S$ is the set of coordinates defining the neighbourhood of $(x, y)$ and $M$ is the number of points in the neighbourhood.

Assuming that the noise values at each point in the image are independent samples from a distribution whose mean is 0, it is clear that the standard deviation of the noise present is reduced by neighbourhood averaging (for a formal proof of this, see [19]), thus indicating an improved signal-to-noise (S/N) ratio.

An obvious side-effect of neighbourhood averaging is one of blurring the image. The degree of blurring is proportional to the size of the averaging neighbourhood used. If the noise is finer grained than the smallest details of interest in the image, the blurring can be made negligible by choosing a correspondingly small averaging neighbourhood. If this is not the case, or if it for some reason is desirable to use a larger averaging neighbourhood, the averaging can be "switched off" in regions with large variations in gray level (lines, edges, etc.) by using the following criterion instead of Eq. (3.55):

$$g(x,y) = \begin{cases} \frac{1}{M}\sum_{(m,n)\in S} f(m,n) & \text{if } \left| f(x,y) - \frac{1}{M}\sum_{(m,n)\in S} f(m,n) \right| < T \\ f(x,y) & \text{otherwise,} \end{cases} \tag{3.56}$$

where $T$ is a specified nonnegative threshold.

#### Averaging of Multiple Images

When several images $g_i$ which have been formed by adding a noise function $\eta$ to an original image $f$, and the noise $\eta$ at each location $(x, y)$ is uncorrelated with mean 0, an image $\overline{g}$ with reduced noise level can be computed by assigning to each pixel the average value of the gray levels of the corresponding pixels in all the noisy images at hand:

$$\overline{g}(x, y) = \frac{1}{M} \sum_{i=1}^{M} g_i(x, y), \tag{3.57}$$

$M$ being the number of noisy images. Since the mean of the noise function $\eta(x, y)$ is 0, the expected value of $\overline{g}$ at any point is given by

$$E\{\overline{g}(x, y)\} = f(x, y). \tag{3.58}$$

If $\sigma_{\eta(x,y)}$ denotes the standard deviation of the noise function $\eta(x, y)$, the standard deviation $\sigma_{\overline{g}(x,y)}$ at any point in the averaged image is given by the relation

$$\sigma_{\overline{g}(x,y)} = \frac{1}{\sqrt{M}} \sigma_{\eta(x,y)}, \tag{3.59}$$

thus indicating that as $M$ increases, $\overline{g}$ approaches the original image $f$.

The obvious drawback of this method is the requirement of having $M$ images of the same scene at hand. This may be the case with grainy photographs or "snowy" TV-frames, but in the case of noisy images in a real-time image application, this would hardly be the most cost-efficient approach.

### 3.4.2 Median Filtering

*Median filtering* is a technique with which the gray level of each pixel in an image is replaced by the *median* of the gray levels in a neighbourhood of that pixel instead of by the average, as was the case in the previous section. The median of a set of values is the value that is such that half of the values in the set is smaller than or equal to it, and the other half is greater than or equal to it. For example, suppose that a $3 \times 3$ neighbourhood at location $(x, y)$ (Fig. 3.1) has the values $(51, 49, 47, 53, 98, 50, 51, 52, 49)$, where 98 is the value at $(x, y)$. This value corresponds to a single-pixel spike, and can be assumed to represent a noise point. A *median filter* placed at $(x, y)$ sorts the pixel values in this neighbourhood and assigns to the pixel at $(x, y)$ the median of this sorted set of values. In the example, the values are sorted as $(47, 49, 49, 50, 50, 51, 52, 53, 98)$. The median is seen to be 50, and thus the pixel corresponding to the spike is assigned the value 50 by the filter.

The effect of median filtering is one of forcing points with very distinct gray levels to be more like their neighbours, thus eliminating intensity spikes like the one in the example above. The superiority of median filtering over e.g. neighbourhood averaging in noise removal is clearly demonstrated by the example on p. 163 in [12].

### 3.4.3 Frequency Domain Methods

By observing that noise in an image mostly is high-frequency in content, it is obvious that by properly manipulating the Fourier transform of the image, noise reduction can be achieved.

#### Lowpass Filtering

The most general and also most obvious approach to noise reduction in the frequency domain is lowpass filtering, as touched upon in Section 3.3.3. Since not only the noise, but also gray level transitions in the image are high-frequency in content, lowpass filtering, as was the case also for neighbourhood averaging, has the side-effect of blurring the image. By using a lowpass Butterworth filter (Fig. 3.4(b)) instead of an ideal lowpass filter (Fig. 3.4(a)), the degree of blurring is reduced. This is because the "tail" of the Butterworth filter lets a fairly high amount of high-frequency information through. In addition, an ideal lowpass filter tends to introduce what is commonly

referred to as *ringing* into the processed image, thus further reducing its quality (for an explanation of ringing, see for instance [12]).

An approach equivalent to lowpass filtering is *low frequency emphasis*, emphasizing low frequencies without filtering out the high frequency end of the Fourier spectrum completely (compare *high frequency emphasis*, Section 3.3.3). This technique also tends to blur the image, but the blurring is not so predominant as for pure lowpass filtering.

#### Selective Filtering

If the noise constitutes a regular pattern which has been superimposed on the image, this pattern tends to have most or all of its energy concentrated at small spots in the frequency plane. Thus, if the Fourier transform of this superimposed pattern is known, it can be removed entirely simply by frequency-domain subtraction of its Fourier transform from the Fourier transform of the noisy image. The loss of information in the processed image will be negligible.

If the nature of the superimposed noise is only partly known, but one is able to locate the spots in the frequency plane where the energy of the noise seems to be concentrated, one can reduce the amount of noise in the image without serious loss of information by suppressing these spots. This is called *selective filtering*.

## 3.5 Image Segmentation

Image segmentation is the process of dividing an image into its constituent parts or objects. One may be interested just in dividing the image into contiguous regions, without paying attention to what sort of objects the individual regions represent, or one may want to be able to identify and classify one or several parts of the image as belonging to specific classes of objects. This section is only concerned with the problem of recognizing parts of images as belonging to continuous regions with certain characteristics. A branch of image segmentation which is not touched upon here is *discontinuity detection*, which aims at detecting points, lines and edges in an image. However, in Section 3.6, the problem of *edge detection* is given separate treatment.

### 3.5.1 Thresholding

The foundation of the segmentation technique known as *thresholding* is the presumption that objects in an image can be segmented out on the basis of their gray levels. For example, if the image consists of a bright object on a dark background, the foreground object may be segmented out by classifying all pixels in the image whose gray levels are above a certain *threshold value* $T$ as belonging to it, and all other pixels (i.e., those whose gray levels are below the threshold) as belonging to the background. The threshold is said to divide the gray scale into two *bands*.

Generally, thresholding may be viewed as an image operation that involves tests against a *thresholding function* $T$ of the form

$$T = T[x, y, p(x, y), f(x, y)], \tag{3.60}$$

where $f(x, y)$ is the gray level at point $(x, y)$, and $p(x, y)$ denotes some local property of this point. A thresholded image $g(x, y)$ is then created by defining (compare Eq. (3.2), p. 14)

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T. \end{cases} \tag{3.61}$$

When $T$ depends only on $f(x, y)$, the threshold is called *global*; if it depends on both $f(x, y)$ and $p(x, y)$, it is called *local*; and if it, in addition, depends on the spatial coordinates $x$ and $y$, it is called *dynamic*. In this section, only global thresholding is considered. That is, Eq. (3.60) is in the form

$$T = T[f(x, y)]. \tag{3.62}$$

If a given image $f$ consists of objects that are, say, darker than their background, global thresholding is a natural approach to extracting them. In this case, the gray-level histogram of the image

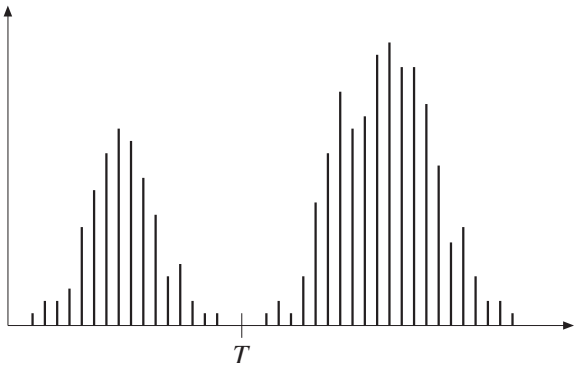



Figure 3.7: Gray-level histogram of an image containing a dark object on a bright background.

is similar to the one given in Fig. 3.7. In the figure, the object and background pixels are grouped into two dominant modes, one at the dark end of the spectrum (the object) and one on the bright end of the spectrum (the background). in the figure, the object of interest is easily separated from its background.

By choosing the threshold value $T$ as indicated

By applying Eq. (3.61) on the image, the resulting image will be a binary one, the object being black and its background white (assuming that 0 in the equation is chosen to correspond to black and 1 to correspond to white).

The problem with thresholding is to choose a value for $T$ which separates the object of interest from other parts of the image well enough; only in ideal cases are the histogram characteristics of the image similar to those of Fig. 3.7. Normally, the gray levels of the image assemble in lumps whose dual characteristics are hardly discernible. Consequently, care has to be taken when choosing the threshold level $T$.

### 3.5.2 Region Growing

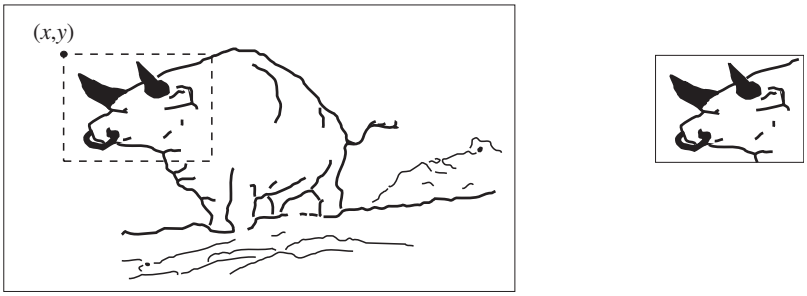
*Region growing* is an algorithm which groups pixels or subregions into larger regions according to criteria of *similarity* and *connectivity*. The criterion of similarity says that only pixels or subregions having a specific set of properties can be added to the growing region, and the criterion of connectivity says that only pixels or subregions that constitute immediate neighbours of the growing region can be added to it.

The simplest approach to region growing is called *pixel aggregation*, where the starting point is a so-called *seed point* from which a region is grown by appending to the seed point neighbouring pixels having similar properties to it. It is of course also possible to apply more than one seed point, each at a different location in the image, thus having several regions grow simultaneously. This is referred to as *multiple* region growing, as opposed to *single* region growing, applying only one seed point. In the following, only single region growing is discussed. The principles presented are, however, easily extendible to multiple region growing.

Two immediate problems with region growing are the selection of a seed point which properly represents the region of interest and the selection of a suitable set of criteria for including new points during the growing process.

If one has *a priori* knowledge of the properties of the region of interest, one approach in selecting the seed point is to apply a search strategy on the image, searching for the first pixel which satisfies these properties, and then using this pixel as seed point. With this approach there is of course the danger that the pixel found does not belong to the region of interest, since single pixels outside the region of interest may share properties with it. Thus it is important to equip the search strategy with a very tight stop criterion in order to minimize the chance of erroneously choosing a non-region seed point (cf. Section 4.1.2).

A usual similarity criterion for the inclusion of new points during the growing process is that the gray level of a pixel to be included not differ from that of the seed point by more than a specified




(a) (b)

Figure 3.8: An image (a), and a template designed to locate the bull's head (b). (From [23])

amount. The obvious stop criterion for the region growing procedure is to stop when no more pixels satisfy the criteria for inclusion.

The advantage of region growing is its degree of automation. It can be compared to filling an unevenly shaped pool with water: Once you start pouring water into it, you can be certain that the water will protrude into every far off corner of the pool, thus defining a continuous surface entirely covering the "region of interest", the floor of the pool. The problem is that you have to know where to start pouring: The seed point.

### 3.5.3 Template Matching

So far, the focus has been on methods for segmenting out objects in an image based on a set of characteristics that differentiates the objects from their background. These characteristics, however, include neither geometrical (e.g., shape) nor textural properties (e.g., dark and bright regions) of the objects to be segmented out. Thus, if one wants to be able to recognize an object in an image having an arbitrary pattern of gray levels, other techniques have to be found.

*Template matching* is perhaps one of the most straightforward methods for locating the presence of an object in an image based on specific geometrical and textural properties. The idea is that an object with given properties can be detected in an image by matching the image at different locations against a *template* having the sought properties, as illustrated in Fig. 3.8. This must be done at every pixel location in the image, and, if the sought object is allowed to rotate, at every possible orientation.

Matching of this kind would be a trivial task if an exact copy of the template could be expected to be present in the image. A match would then be detected when each pixel in the template has the same gray level as the one it is matched against. In practice, however, the image is normally noisy, and thus the sought object has to be found by maximizing some measure of the degree of match between the template and the image. Some standard measures of the degree of match between an image $f(x, y)$ and a template $T$ whose gray level at $(x, y)$ is $t(x, y)$ are[3]:

$$\max_{(x,y)\in T} |f(x, y) - t(x, y)| \tag{3.63}$$

$$\int\int_T |f(x, y) - t(x, y)| dxdy \tag{3.64}$$

$$\int\int_T [f(x, y) - t(x, y)]^2 dxdy. \tag{3.65}$$

These measures are all zero for a perfect match and have high values for poor matches. They differ, however, in the type of errors to which they are sensitive (to see how they differ, see [20]).

A popular match measure which can be derived from Eq. (3.65) is the *correlation coefficient* $\rho$, defined by

$$\rho(x, y) = \frac{\int\int_T f(x, y)t(x, y)dxdy}{\sqrt{\int\int_T f^2(x, y)dxdy \int\int_T t^2(x, y)dxdy}}. \tag{3.66}$$

[3]For the sake of generality, integrals are used; in the digital case these are replaced by summations.

It can be shown that $0 \leq \rho(x, y) \leq 1$ for all $(x, y)$ in the image, with value 1 being achieved if and only if $f(x, y) \equiv ct(x, y)$ for some positive constant $c$. Also, $\rho$ is unaffected if the gray scale of $f$ is multiplied by a constant, unlike the measures in Eqs. (3.63) through (3.65).

As is evident from the above, template matching is rather expensive computationally. Correlation may be carried out more efficiently in the frequency domain, but only if the image and the template are of the same size, which is seldom the case. The results obtained also vary depending on the similarity measure employed. In [23], three measures of match are evaluated as to their performance in determining eye position. The measures evaluated are the correlation coefficient (Eq. (3.66)), the *sum of absolute valued differences* (SAVD, Eq. (3.64)), and a relatively new technique called the *stochastic sign change criterion* (SSC). In this paper it is concluded that, due to "gross misregistrations", the correlation coefficient $\rho$ is "inapplicable as a similarity measure". For the SAVD and SSC measures, the results as to their accuracy are satisfactory.

## 3.6 Edge Detection

As mentioned in the previous section, *discontinuity detection* is a branch of image segmentation. To the area of discontinuity detection belong the fields of *point detection*, *line detection*, and *edge detection*. In this section, the focus is on edge detection, as it constitutes by far the most common approach for detecting discontinuities in gray level. An *edge* is defined as the boundary between two regions characterized by having relatively distinct gray levels.

A difficulty with edge detection as an approach to image segmentation is that the detected edges often have gaps in them, due to places where transitions between regions are not abrupt enough that an edge be detected. In addition, if the image is noisy, edges may be detected at points which are not parts of region boundaries, as discussed below. Thus the detected edges will not necessarily form closed, connected curves which surround closed, connected regions.

### 3.6.1 Thresholding

The thresholding technique introduced in Section 3.5.1 can, with slight modifications, be applied to detect edges in an image, as described in [12]. The idea is to choose a threshold $T$, based on some criteria, dividing the gray scale of the image into two distinct bands. Then the image is scanned in the $x$ and $y$ directions separately. Each time a change in gray level from one band to the other occurs, this indicates the presence of a boundary point. The procedure performs in two passes as follows:

**Pass 1:** For each row in $f(x, y)$, create a corresponding row in an intermediate image $g_1(x, y)$, using the following relation:

$$g_1(x,y) = \begin{cases} L_E & \text{if the levels of } f(x,y) \text{ and } f(x,y-1) \text{ are in} \\ & \text{different bands of the gray scale,} \\ L_B & \text{otherwise,} \end{cases} \tag{3.67}$$

where $L_E$ and $L_B$ are specified boundary and background levels, respectively.

**Pass 2:** For each column in $f(x, y)$, create a corresponding column in an intermediate image $g_2(x, y)$ using the following relation:

$$g_2(x,y) = \begin{cases} L_E & \text{if the levels of } f(x,y) \text{ and } f(x-1,y) \text{ are in} \\ & \text{different bands of the gray scale,} \\ L_B & \text{otherwise.} \end{cases} \tag{3.68}$$

The desired image, consisting of the boundary points of the object of interest on a uniform background, is then given by the relation:

$$g(x,y) = \begin{cases} L_E & \text{if } (g_1(x,y) = L_E) \vee (g_2(x,y) = L_E), \\ L_B & \text{otherwise.} \end{cases} \tag{3.69}$$

As is the case with thresholding in general, care has to be taken when choosing the threshold value $T$.

As is evident from its formulation, this procedure assumes that the edges in the given image are characterized by monotonously increasing or decreasing functions. This is seldom the case in real applications. If the image has been subjected to additive noise, so that in regions with gray levels in the proximity of $T$ pixel values are randomly distributed on both sides of $T$, which is normally the case, this method will detect an edge point each time a transition from one band to another occurs, thus making it difficult to distinguish between an actual edge, and edges detected due to noise. In the worst case an edge will be depicted as a "bulb" of layered edge fragments. If the edges of interest are relatively sharp, however, that is, they approximate a step edge rather than a ramp edge, this problem is less dominant.

### 3.6.2 Gradient Operators

As edge points in an image are characterized as points where the gray level of the image changes rapidly, an obvious approach to edge detection is to apply some kind of differentiation on the image. The most common approach to differentiation in image processing applications is the *gradient*. The gradient of a two-dimensional function $f(x, y)$ is defined as the vector

$$\vec{G}[f(x,y)] = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \\ \dfrac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} G_x \\ \\ G_y \end{bmatrix}. \tag{3.70}$$

$\vec{G}[f(x, y)]$ always points in the direction of the maximum rate of increase of $f$ at $(x, y)$, and the rate of increase per unit distance in this direction, denoted $G[f(x, y)]$, is given by

$$G[f(x,y)] = |\vec{G}| = \sqrt{G_x^2 + G_y^2}. \tag{3.71}$$

The magnitude function $G[f(x, y)]$ is generally referred to as the *gradient* of $f$, to avoid constantly having to refer to it as "the magnitude of the gradient". To reduce computational costs, the gradient is commonly approximated using absolute values:

$$G[f(x,y)] \approx |G_x| + |G_y|. \tag{3.72}$$

For digital images, $G_x$ and $G_y$ are approximated using differences. One approach is to use first-order differences in a $2 \times 2$ neighbourhood. A typical approximation in this respect is

$$G_x \simeq f(x,y) - f(x+1,y) \tag{3.73}$$

$$G_y \simeq f(x,y) - f(x,y+1), \tag{3.74}$$

yielding the following expression for $G[f(x, y)]$:

$$G[f(x,y)] \simeq |f(x,y) - f(x+1,y)| + |f(x,y) - f(x,y+1)|. \tag{3.75}$$

This approximation uses first-order differences in the horizontal and vertical directions about $(x, y)$ to compute $G[f(x, y)]$. Another approximation, commonly referred to as the *Roberts gradient*, uses differences symmetrical about the imaginary interpolation point $(x + \frac{1}{2}, y + \frac{1}{2})$:

$$G[f(x,y)] \simeq |f(x,y) - f(x+1,y+1)| + |f(x+1,y) - f(x,y+1)|. \tag{3.76}$$

The approximations in Eqs. (3.75) and (3.76) use the differences between two pairs of adjacent pixels to compute the gradient. Thus, for about the same reasons given in Section 3.6.1, these approximations tend to be sensitive to noise.

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

(a)

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

(b)

Figure 3.9: Spatial masks used to compute $G_x$ (a) and $G_y$ (b), as defined by Eqs. (3.77) and (3.78).

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

Figure 3.10: Spatial mask used to compute the Laplacian as defined by Eq. (3.80)

### The Sobel Operators

Using a $3 \times 3$ neighbourhood to compute the gradient has the advantage of increased smoothing over $2 \times 2$ operators. Consider the following definitions of $G_x$ and $G_y$:

$$\begin{aligned} G_x &= [f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1)] - \\ &\quad [f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1)] \end{aligned} \tag{3.77}$$

$$\begin{aligned} G_y &= [f(x+1,y-1) + 2f(x+1,y) + f(x+1,y+1)] - \\ &\quad [f(x-1,y-1) + 2f(x-1,y) + f(x-1,y+1)]. \end{aligned} \tag{3.78}$$

The spatial masks shown in Figs. 3.9(a) and (b) implement $G_x$ and $G_y$, respectively. These two masks are commonly referred to as the *Sobel operators*. The responses of these two operators at any point $(x, y)$ in an image are combined using Eq. (3.71) or Eq. (3.72) to obtain the gradient at that point. Convolving these masks with an image yields the gradient at all points $(x, y)$ in the image, the result often being referred to as a *gradient image*.

### The Laplacian

Higher order derivative operators can also be used to detect edges in an image. The *Laplacian* operator $L[f(x, y)]$ is a second-order derivative operator which is direction insensitive, as opposed to the Sobel-operators. In the continuous case, the Laplacian is defined by the equation

$$L[f(x,y)] = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{3.79}$$

In the digital case, it is approximated by

$$L[f(x,y)] \simeq f(x,y-1) + f(x-1,y) + f(x+1,y) + f(x,y+1) - 4f(x,y). \tag{3.80}$$

A mask implementing Eq. (3.80) is shown in Fig. 3.10. Note that Eq. (3.80) implies that $L[f(x, y)]$ takes on both positive and negative values. If only positive values are desired, the absolute value can be used. Note also that the Laplacian is zero in constant areas and on the ramp section of edges, as expected of a second-order derivative operator.

As a consequence of its being a second-order derivative operator, however, the Laplacian tends to be unacceptably sensitive to noise. It also does not respond as strongly to edges as do the other operators described thus far. On the other hand, it responds far more strongly to single points, lines and line ends than these do. It can also be applied to detect whether a given pixel is on the dark or bright side of an edge.

### 3.6.3 Highpass Filtering

Convolving an image with any of the gradient operators described above is equivalent to highpass filtering in the frequency domain. Thus it ought to be possible to detect edges in an image by highpass filtering the Fourier transform of the image directly. Highpass filtering the Fourier transform of a given image indeed has effects similar to those of applying the above operators. The usefulness of direct highpass filtering as an edge detection technique is, however, questionable, in that the responses returned are relatively weak as compared to the results obtained using the above techniques. This is clearly demonstrated in Figs. 9 and 10 on pp. 281–283 in [19].

### 3.6.4 The Hough Transform

The last approach to edge detection discussed here, is the so-called *Hough transform*. The discussion below is directed towards applying the Hough transform to detect straight lines and edges in an image, but the principle can be extended to detecting any curve or edge that can be described by the equation $g(\vec{x}, \vec{c}) = 0$, where $\vec{x}$ is a vector of coordinates and $\vec{c}$ is a vector of coefficients.

Consider a point $(x_i, y_i)$ in an image. An infinite number of lines satisfying the equation $y_i = ax_i + b$ for varying values of $a$ and $b$ pass through this point. However, the equation $b = -x_i a + y_i$, which is equivalent to the previous one, describes a single line in the $ab$ plane (often referred to as *parameter space*) for a fixed pair $(x_i, y_i)$. Analogously, a second image point $(x_j, y_j)$ also has a line in parameter space associated with it. This line intercepts the line associated with $(x_i, y_i)$ at a point $(a', b')$, where $a'$ is the slope and $b'$ the intercept of the line in the $xy$ plane (the image) containing both $(x_i, y_i)$ and $(x_j, y_j)$. All points on this line will have lines in parameter space which intercept at $(a', b')$.

By associating with each point $(a_i, b_i)$ in parameter space an *accumulator cell*, it is possible to keep track of how many points in the $xy$ plane lie on a given line. This is done by letting, for every point $(x_k, y_k)$ in the image, $a$ vary along the entire $a$ axis in parameter space, and for each $a_l$ solve for $b_l$ using the relation $b_l = -x_k a_l + y_k$. The value of the accumulator cell associated with $(a_l, b_l)$ is then incremented by one, and at the end of the procedure, a value of $M$ in the accumulator cell associated with $(a_i, b_j)$ corresponds to $M$ points in the image lying on the line $y = a_i x + b_j$.

A problem with describing a line with the equation $y = ax + b$ is that both slope and intercept approach infinity as the line approaches a vertical position. This problem is circumvented by using the *normal* description $x \cos\theta + y \sin\theta = \rho$ of the line instead. The only difference in using this representation when constructing the set of accumulator cells is that the intercepts in parameter space ($\rho\theta$ plane) are between sinusoidal curves instead of straight lines.

As mentioned above, these principles can be applied to detect any curve or edge which can be described by the relation $g(\vec{x}, \vec{c}) = 0$. The equation $(x - c_1)^2 + (y - c_2)^2 = c_3^2$, for example, describes a circle and can thus be used analogously to the above procedure to detect circles in an image. One possible application would be to detect the (circular) pupil in an image of the eye (cf. Section 4.1.3).

If the number of points in the image equals $N^2$ (i.e., the image is of size $N \times N$) and the $a$ axis is divided into $K$ equal increments, the number of computations needed for the above procedure is $KN^2$, since for each image point $(x_i, y_i)$, $a$ is varied from $a_1$ to $a_K$ to obtain the corresponding values for $b$. Thus the Hough transform performs in $O(N^2)$ time, assuming that $K \ll N$.

# Chapter 4

# Evaluation and Chosen Approach

In the first section of this chapter, Section 4.1, the different techniques presented in Chapter 3 are evaluated as to their individual applicability to the problem at hand. In Section 4.2, on the basis of the foregoing evaluation, a conclusion is drawn as to which approach to take in developing an algorithm to solve the problem. The proposed algorithm itself is is presented in Chapter 5.

## 4.1 Evaluation

As mentioned in Section 3.1.2, the techniques presented in Chapter 3 can be divided into two main categories: *Noise reduction* schemes and *image segmentation* schemes. With regard to the noise reduction schemes presented in Section 3.4, it was pointed out there that the presentation primarily was intended as an indicator towards some possible techniques for removing noise from an image if it at a later stage of the project should turn out that this would be necessary. With that in mind, the noise reduction schemes are evaluated in Section 4.1.1.

For the sake of clarity, the presentation of general image segmentation schemes was divided in two: Image segmentation as a means of recognizing parts of an image as belonging to a continuous region with certain characteristics, elaborated upon in Section 3.5; and edge detection, which was given separate treatment in Section 3.6. In Section 4.1.2 below, the image segmentation schemes are evaluated, and in Section 4.1.3, the edge detection schemes. Note that the schemes are evaluated with respect to their applicability fro detecting the pupil in an image, and that, when evaluating them, it is assumed that the pupil is circular (cf. Section 2.1).

The key criterion for applicability is low computational cost. In the following, when comparing the computational cost of the different techniques, it is assumed that the given image is of size $N \times N$, and that the technique in question is carried out on the entire image. The cost is measured in terms of $N$. Obviously, the cost of any technique applied on an entire image cannot be lower than $O(N^2)$. Thus, techniques are deemed too costly that are more expensive than $O(N^2)$. Another obvious criterion is that the technique in question serves the purpose.

### 4.1.1 Noise Reduction Schemes

The noise reduction schemes presented in this chapter belong to two different categories which are treated separately below: Spatial domain and frequency domain schemes.

#### Spatial Domain Approaches

As pointed out in Section 3.2, applying spatial operators on an image generally requires $O(N^2)$ operations. Thus the spatial noise reduction schemes presented in Sections 3.4.1 and 3.4.2, neighbourhood averaging, averaging of multiple images and median filtering, perform in $O(N^2)$ time.

Of the two averaging techniques, averaging of multiple images can be discarded without delay, owing to its requiring several input images of the same scene, a requirement which is unsustainable in our case. Neighbourhood averaging has the drawback of attenuating the edges in the image, thus making location of the pupil based on edge detecting schemes difficult.

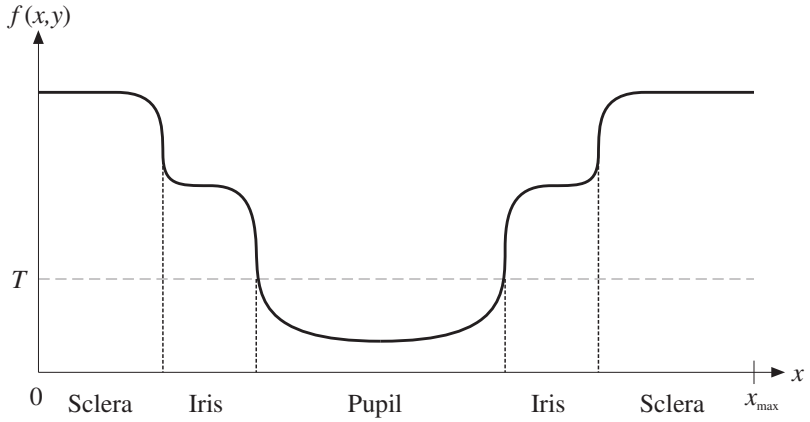



Figure 4.1: An ideal cross section along the $x$ axis of an image of the eye.

As pointed out in Section 3.4.2, median filtering turns out to be clearly superior to averaging techniques, both with respect to removing noise from an image and to maintaining edges in the image. In addition, median filtering is, when carefully implemented, computationally cheaper than averaging schemes, in that it requires mostly comparisons and no multiplications.

#### Frequency Domain Approaches

From Section 3.3.1, p. 19, we have that performing an FFT algorithm on an image requires $O(N^2 \log_2 N)$ complex operations. Thus image processing techniques operating in the frequency plane generally perform in minimum $O(N^2 \log_2 N)$ time, assuming that the image first has to be Fourier transformed. This applies to both simple highpass filtering and selective filtering.

Simple highpass filtering has a similar effect to neighbourhood averaging, namely blurring of the image, and consequently attenuating of edges. The selective filtering scheme would be better suited, but then it is required to know the Fourier transform of the superimposed noise function aforehand. This is seldom the case in real-time applications, where the noise normally has a random distribution.

The prohibitive aspect of these schemes (and of any frequency domain scheme), however, is their computational cost, elaborated upon in Section 3.3.1, p. 19. Thus frequency domain noise reduction schemes implemented in software can be discarded as an alternative in real-time applications (hardware implementations may be fast enough to be sustainable).

### 4.1.2 Pupil Detection Based on Image Segmentation

The term *image segmentation* refers in this section, as in Section 3.5, to the problem of recognizing parts of an image as belonging to a continuous region with certain characteristics. Edge detection schemes are treated separately in Section 4.1.3.

#### Thresholding

As pointed out in Section 2.1.2, the pupil constitutes the darkest portion of the eye, and can thus be assumed to correspond to the darkest region in an image of the eye. Ideally, the characteristics of a cross section of an image of the eye are as depicted in Fig 4.1. From this figure it is evident that pixels belonging to the pupil by and large can be recognized as such by thresholding the image with respect to a threshold value $T$.

However, as pointed out in Section 3.5.1, the problem with thresholding is, when applied to real images, to decide on a value for $T$ which unambiguously differentiates between actual pupil points and points not belonging to the pupil. The probability that all pixels in the pupil have values below $T$ and all pixels outside of it have values above $T$ is for real images practically zero. Thus, in the most cases, applying a pure pixel-by-pixel thresholding technique to obtain a binary image will on

the one hand result in an image where the pupil forms the largest recognizable object; on the other hand, this object may have holes in it, and its boundary will hardly form a perfect circle.

A better approach would be to apply some averaging technique in the neighbourhood about the pixel to be thresholded, and then compare the value returned to the threshold value $T$. By carefully choosing the averaging technique and equally carefully choosing $T$, it is possible to design a *pixel filter* which only lets points through that belong to the pupil. Note, however, that it would only be possible to design this filter so that no non-pupil points are let through; to design it so that *all* actual pupil points are let through, is for all practical purposes impossible.

From the above it can be concluded that thresholding is a technique which enables the unambiguous recognition of a pupil point, although as a means of reliably determining the *centre* of the pupil, it is not particularly suited. Lastly, since thresholding is a spatial technique, it performs in $O(N^2)$ time.

**Region Growing**

Region growing is a technique which would seem well suited for detecting the area and the extent of the pupil. One problem, however, is the requirement of a positive pupil point as seed point. A search algorithm operating on the image and employing a carefully designed pixel filter of the kind depicted above as stop criterion could be expected to return such a point.

Another problem is to chose the similarity criteria such that as many actual pupil points as possible and as few non-pupil points as possible are included. A possible similarity criterion would be to include those pixels whose values are below a given threshold value $T$, assuming that the value of the seed point also is below $T$. This would, however, result in exactly the same unsuitable pupil region as the one supplied by the pixel-by-pixel thresholding scheme above, given that the image and $T$ are the same. A usual, and better, similarity criterion is that the gray level of a pixel to be included not differ from that of the seed point by more than a specified amount. This is, however, also a form of thresholding (cf. Eq. (3.2), p. 14), and puts a rigid limit on the interval of gray levels accepted.

Since the pupil is assumed to constitute the darkest region in the image, having a relatively well defined edge to the surrounding iris, a third approach would be to employ some sort of edge detecting scheme (cf. Section 3.6 as well as Section 4.1.3 below) as a criterion for inclusion. That is, to use thresholding not on the image itself, but on some sort of operator aimed at detecting edges. With this scheme, during the growing process, all pixels not recognized as edge points are included.

The problems with this approach are to choose an optimal edge detecting operator and, again, to choose an appropriate threshold value $T$. Also, the danger exists of the pupil having a boundary with "holes" in it; that is, actual edge points that are not recognized as such by the edge detecting operator. In that case, the region being grown will "overflow", i.e., it will cross the actual boundaries of the pupil and possibly extend to include the entire image. By ensuring optimal illumination of the eye, in addition to the care taken when choosing the edge detecting operator and the threshold value $T$, it ought to be possible to reduce the probability of this happening to a minimum.

After a region has been grown that corresponds as closely to the pupil as possible, the problem remains of determining the position of the *centre* of the pupil. This can be done by, by means of some data structure, registering the coordinates of the end points of the rows and columns of the region being grown. When the region is complete, the $x$-position of its centre (and thus of the pupil's, assuming that the correspondence between the region and the pupil is approximately 1:1) is computed by taking the average of the centre points of all rows (using the relation $x_{\text{centre}} = \frac{1}{2}(x_{\text{left}} + x_{\text{right}})$). The $y$-position is computed analogously.

As mentioned above, region growing seems to suggest a reasonable approach to the given problem. It remains to be seen, however, if it is computationally affordable in real-time applications. In the general case, that is, when the entire image is considered to consist of contiguous regions that all are to be grown, region growing performs in $O(N^2)$ time with proportionality constant 1. In our case, however, we are only interested in growing *one* region, namely the pupil, and thus the proportionality constant depends on the fraction of the image occupied by the pupil. In our case, that is, when the image covers the entire eye and no more, this fraction is between 10% and 20%, thus reducing the proportionality constant to between 0.1 and 0.2. The number of operations required by the algorithm searching for the seed point depends on the search strategy employed.

#### Template Matching

As mentioned in Section 3.5.3, [23] offers an evaluation of three template matching algorithms for detecting and locating the pupil in an image of the eye. Of the three match measures employed, the SAVD and the SSC are characterized as satisfactory, whereas the correlation coefficient $\rho$ is deemed to be "inapplicable as a similarity measure".

At any rate, due to its computational cost, software implemented template matching can be discarded as an algorithm on which to base the needed eye-tracking system. To see why template matching is so computationally expensive, consider once again Fig. 3.8 on p. 28. It is evident that the template has to be moved to every pixel location in the image in order to obtain the best match. There are $N^2$ pixels in the image. Assuming that the template is of size $M \times M$, the number of operations required to compute the match at any location is $O(M^2)$. Thus the total number of operations is $O(M^2N^2)$. The diameter of the pupil in a typical image of the eye is $\sim 0.3N$, yielding $M \approx 0.3N$. This yields a total number of operations of $\sim 0.1N^4$, which corresponds to template matching being $O(N^4)$ in time!

### 4.1.3 Pupil Detection Based on Edge Detection

All edge detection techniques presented in this chapter, except the Hough transform, deliver images where the edges stand out on a relatively uniform background. The first technique described, thresholding, returns a binary image where the edges detected are emphasized. For the latter two, gradient operators and highpass filtering, a binary image can be obtained by appropriately thresholding the returned image. Thresholding and gradient operators perform in $O(N^2)$ time, whereas highpass filtering, as pointed out in Section 4.1.1, performs in $O(N^2 \log_2 N)$ time. The performance of the Hough transform is discussed below.

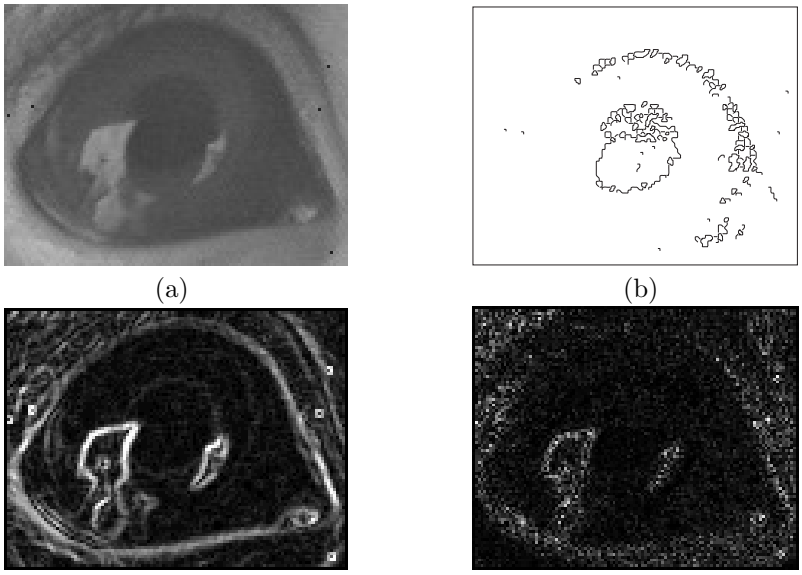
The main problem with determining the position of the pupil based on edge detection is that it has to be known whether or not a given edge corresponds to the contour between the pupil and the iris. There is nothing in the nature of this edge that uniquely separates it from the other edges in the image. There is, however, only one other circular edge in the image, namely the iris-sclera edge. By taking advantage of the fact that the radius of the pupil only can take on values within a certain interval, and that it always is smaller than the radius of the iris, it would be possible to detect the pupil by looking for an edge forming an circle whose radius lies inside this interval. This can be done by applying the Hough transform on the image, as described below and in Section 3.6.4.

Another way of unambiguously recognizing the pupil-iris edge as such, is to make it the first edge that the edge detecting algorithm encounters. To ensure this, the edge detection procedure has to start from a point positively classified as a pupil point, the *origin of operation*, and then carry on outwards in all directions until, in each direction, an edge is detected. All edges thus detected can together be assumed to represent the pupil-iris contour. By registering the coordinates of opposite edge points, it is possible to compute the position of the centre of the pupil, as described in the previous section. As a matter of fact, the last proposed inclusion criterion for the region growing procedure—that only those pixels that are *not* edge points be included—constitutes no more than a concrete implementation of the algorithm described here. To supply the necessary pupil point, some search strategy can be applied on the image, employing, for instance, a pixel filter, as described in the previous section.

The first three edge detection algorithms presented in Section 3.6—thresholding, gradient operators and highpass filtering—are evaluated with respect to being employed in this algorithm. The main evaluation criterion, next to computational cost, is thus how strongly they respond to the pupil-iris contour. The image in Fig. 4.2(a) is used to illustrate the individual responses of the techniques in question, except for highpass filtering, for which no implementation has been made. The Hough transform is treated separately in the last part of the section.

#### Thresholding

Ideal step edges are seldom found in real applications, and neither are edges that can be characterized by monotonously increasing or decreasing functions. Normally, an edge is characterized by a function with spurious increases and decreases, but with an overall tendency either to increase



(a) (b)

(c) (d)

Figure 4.2: (a) An image of a monkey's eye. (b) The result of applying the thresholding procedure depicted in Section 3.6.1 on this image. (c) The result of applying the Sobel operators on the image. (d) The result of applying the Laplacian on the image.

or decrease. Thus the main problem with the procedure described in Section 3.6.1 is its implicit requirement of having monotonous edges. The "bulb"-effect of the procedure is clearly illustrated in Fig. 4.2(b), which can be viewed as a worst-case example; note in particular the uppermost part of the pupil, where the actual edge has been "drowned" by spurious variations in gray level that lie on both sides of the threshold value $T$. This illustrates how critical the choice of a value for $T$ is.

By choosing another value for $T$, it is clear that the result would have looked different. However, since the average gray level of the images delivered by the camera tend to vary over time, it is not safe to depend on a fixed threshold value in order to detect the pupil-iris contour. Moreover, this simplest form of thresholding looks only at individual pixels, and accordingly induces no smoothing whatsoever, leaving it too sensitive to noise. All in all, thresholding can be discarded as an edge detecting scheme applicable in the algorithm described above.

### Gradient Operators

As can be seen from the image in Fig. 4.2(c), the Sobel operators described in Section 3.6.2 deliver an image in which edges are depicted as relatively thick curves whose intensities depend on the rate of change in gray level, as expected of derivative operators. The contour of the pupil is clearly seen, although it is not very prominent. The definitely strongest response in the image is induced by the reflection to the left in the image, where the largest transition in gray level takes place. By carefully choosing a threshold value for determining whether the response corresponds to an edge point, it ought to be possible to employ the Sobel operators in the above algorithm. Note in particular that gradient operators are insensitive to changes in the overall gray level of an image.

In Fig. 4.2(d), the result of applying the Laplacian on the image in Fig. 4.2(a) is shown. As pointed out in Section 3.6.2, the Laplacian does not respond as strongly to edges as do other gradient operators. Moreover, being a second-order derivative operator, it tends to be unacceptably sensitive to noise; note for instance the "rippling" in the upper left corner of the image in the figure. Most importantly, however, and what excludes the Laplacian as a means of detecting the pupil, is that the pupil is not discernible in the image at all.

#### Highpass Filtering

One prohibitive factor with respect to employing highpass filtering as a means of detecting the edges of the pupil in our case, is the computational cost of frequency domain methods in general, elaborated upon in Section 3.3.1, p. 19. Moreover, as pointed out in Section 3.6.3, the response given by highpass filtering schemes to edges in an image are notably weaker than the response given by, for instance, first-order derivative operators. Lastly, it should be noted that, since noise generally is high frequency in content, highpass filtering offers no smoothing effect whatsoever, when applied to a noisy image.

#### The Hough Transform

As described in Section 3.6.4, the Hough transform can be employed to detect any curve or edge that can be described by the equation $g(\vec{x}, \vec{c}) = 0$, where $\vec{x}$ is a vector of coordinates, and $\vec{c}$ is a vector of coefficients. Thus, since a circle is described by the equation

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2, \tag{4.1}$$

the pupil can be detected by applying this equation in the Hough transform. The difference between the algorithm designed for detecting lines, as depicted in Section 3.6.4, and a similar approach to detecting circles, is that the number of parameters now is a triplet, $(c_1, c_2, c_3)$, which results in a three-dimensional parameter space, each coordinate triplet having an accumulator cell associated with it. Thus, for each accumulator cell to contain the number of points in the image that lie on the circle corresponding to it, both $c_1$ and $c_2$ are varied along their respective axes, and and for each pair $(c_1, c_2)$, Eq. (4.1) is solved with respect to $c_3$, the accumulator cell associated with the triplet $(c_1, c_2, c_3)$ being incremented for each solution.

In addition to having to deal with a three-dimensional parameter space, it is evident from the relative complexity of Eq. (4.1) that the application of the Hough transform to detect circles in an image is rather expensive. In Section 3.6.4, it was shown that the Hough transform as applied to detecting lines in an image performs in $O(N^2)$ time, assuming that the proportionality constant $K$, corresponding to the resolution along the $a$ and $b$ axes in parameter space, satisfies the relation $K \ll N$. In the three-dimensional case, however, the proportionality constant is $K^2$, which makes it no longer sound to assume $K^2 \ll N$. Moreover, due to high resolution in parameter space being a prerequisite to high resolution in the $(x, y)$ plane, it is more than probable that in our case $K$ would have to be chosen so that $K^2 > N$. Consequently, the Hough transform as applied to detecting the pupil in an image of the eye performs in $O(N^2)$ time, but with a proportionality constant presumably larger than $N$. Thus, also the Hough transform can be discarded as a suitable approach to the problem at hand, on grounds on being too expensive computationally.

## 4.2 Chosen Approach

In this section it is assumed that the images supplied by the camera are relatively free from noise. In other words, they will not be subjected to a noise removing scheme prior to being input to the algorithm. Still, it will be aimed at making the algorithm as insensitive as possible to the noise present in the images, mainly by using operators on the image that introduce some kind of smoothing in addition to performing their designated tasks. Should it, however, at a later stage of the project, turn out that noise removing preprocessing of the images is necessary, it follows from the previous section that median filtering is the computationally cheapest and also most promising of the presented schemes.

### 4.2.1 Rejected Approaches

In the previous sections, a number of techniques were discarded as not applicable to the given problem. Most of the techniques were discarded due to being to computationally expensive. This applies to frequency domain methods in general, being at least $O(N^2 \log_2 N)$ in time; to template matching, being $O(N^4)$ in time; and lastly, to the Hough transform, being $O(N^2)$ in time, but with

a proportionality constant presumably larger than $N$. The other incentive to rejection was failure to serve the purpose. The techniques thus rejected were simple thresholding, both as a means of segmenting out the pupil, as a criterion for inclusion in region growing, and as a means of detecting the pupil-iris contour; and the Laplacian operator, responding to weakly to the edges of interest.

### 4.2.2 Promising Aspects

Many of the presented techniques displayed promising aspects which in some way may be combined to yield a satisfactory foundation for the algorithm to be developed. The most promising of these aspects are listed below:

- Thresholding was, as elaborated upon in Section 4.1, discarded, both as as a means of segmenting out the pupil, as a criterion for inclusion in region growing, and as a means of detecting the pupil-iris contour. However, as was pointed out in Section 4.1.2, it can be applied to implement a *pixel filter* which can be used to unambiguously classify a given pixel as either a pixel point or a non-pixel point.
- The region growing technique, supplied with a seed point from a search algorithm employing some sort of pixel filter as stop criterion, would be able to detect the extent of the pupil, and thus also its centre.
- The most promising inclusion criterion for the region growing approach depicted in Section 4.1.2 was deemed to be one accepting for inclusion only those pixels *not* recognized as edge points.
- By letting an edge detection procedure start from a point positively classified as a pupil point and then proceed in all directions until, in each direction, an edge is detected, the extent of the pupil and thus its centre can be computed.
- Of the presented edge detecting schemes, gradient operators, exemplified by the Sobel operators, turned out to give the strongest response to the relatively weak pupil contour in the test image in Fig. 4.2(a).

### 4.2.3 Satisfying the Requirement of Speed

As the minimum error in an eye-tracking system based on digital image analysis is proportional to the pixel size in the digitized image, it is desirable to employ images with as high a resolution as possible. As pointed out in Section 1.3, a resolution of $1024 \times 1024$ pixels with 256 gray levels would have been preferred. For reasons elaborated upon there, however, a lower resolution of $512 \times 512 \times 256$ was decided upon. Still, the total number of pixels that the algorithm has to cope with is fairly high: 262.144. In contrast to this number is the maximum time of 20 ms for the algorithm to detect and locate the pupil in a given image. From these more or less incompatible requirements it is evident that every effort has to be put into making the algorithm as fast as possible without compromising the requirement of accuracy.

To solve this problem it would be beneficial if it were possible to detect and locate the pupil without having to consider every single pixel in the image. The smaller the fraction of the total number of pixels that have to be taken into consideration, the faster the algorithm.

#### Region Growing?

From the above, it is clear that one approach to reducing the number of pixels having to be considered is to use region growing and only include non-edge points during the growing process. This, however, would have to be implemented recursively, which by nature is rather expensive. In addition, to keep track of the pixels that have been visited and those that have not, it would be necessary to maintain a large data structure of size similar to that of the image. Each time a pixel is to be tested for inclusion, this structure has to be consulted as to whether it already has been tested, which implies a large number of unnecessary comparisons. And, perhaps most importantly, this approach requires that all pixels in the pupil be tested as to whether they are to be included

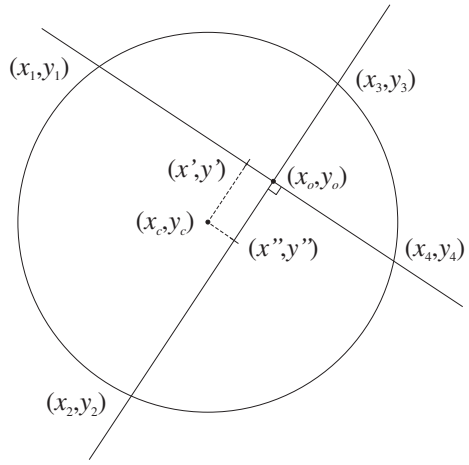



Figure 4.3: Illustration of how the centre of the pupil can be computed. The circle designates the pupil.

in the growing region or not. The number of pixels in the pupil is not large compared to the total number of pixels in the image, but an even better solution would be one with which only a subset of the pixels in the pupil has to be considered.

### 4.2.4 Line Oriented Edge Detection

An algorithm of which the above region growing procedure is but an implementation is the edge detection based algorithm referred to in the list in Section 4.2.2 and described in Section 4.1.3. The basis of this algorithm is that the pupil-iris edge can be recognized as such by letting it be the first edge that an edge detecting algorithm encounters. The algorithm assumes the pupil to form a circle in the image and operates by proceeding, from a point positively classified as a pupil point (the *origin of operation*), in different directions, applying some edge detecting operator successively on every pixel encountered in each direction, and, if an edge point is recognized, registering the coordinates of this point. Since the pupil-iris edge can be assumed to be the first edge encountered when moving in any direction from a point within the pupil, the points thus detected represent the extent of the pupil. A line passing through the origin of operation and traversing the entire extent of the pupil in some direction will hereafter be referred to as a *detection line*. To compute the position of the centre of the pupil, it is necessary to know the coordinates of the oppositely positioned edge points along two perpendicular detection lines. The procedure is illustrated in Fig. 4.3. $(x_o, y_o)$ designates the origin of operation and $(x_i, y_i)$, $1 \leq i \leq 4$, the edge points detected along the two detection lines. The centre points $(x', y')$ and $(x'', y'')$ of the two lines with respect to the extent of the pupil are given by the relations $(x', y') = (\frac{1}{2}\{x_1+x_4\}, \frac{1}{2}\{y_1+y_4\})$ and $(x'', y'') = (\frac{1}{2}\{x_2+x_3\}, \frac{1}{2}\{y_2+y_3\})$, thus yielding the following expressions for the centre coordinates $x_c$ and $y_c$ of the pupil:

$$x_c = x' + x'' - x_o \tag{4.2}$$

$$y_c = y' + y'' - y_o. \tag{4.3}$$

As mentioned above, this formulation assumes the pupil to form a circle in the image. As pointed out in Section 2.1.2, this is only the case when the subject looks straight into the camera. However, since the primary need of the Eckhorn-Bauer group is to be able to determine whether or not the subject is drifting, the approach is deemed satisfactory, assuming that the setup ensures the subject's focusing on the reference point corresponds to its looking straight into the camera.

With this in mind, it is clear that to compute an estimate of the position of the pupil on the basis of an already known pupil point—the origin of operation having coordinates $x_o$ and $y_o$—one needs to (1) detect the extent of the pupil along two perpendicular detection lines relative to this point, and (2) combine the coordinates of the interception points of the lines with the pupil-iris

contour to compute $x_c$ and $y_c$, using Eqs. (4.2) and (4.3). This method of detecting the extent of the pupil along detection lines will in the following be referred to as *line oriented edge detection*.

Obviously, the estimate obtained by performing the above procedure on only one pair of lines is not very accurate with respect to the actual centre of the pupil. However, by performing the procedure on several pairs of lines and estimating the centre position to be the average of the individual estimates thus obtained, the accuracy increases. The higher the number of line pairs, the higher the accuracy.

### 4.2.5 The Algorithm: Basic Formulation

From the above, it is clear that of the presented algorithms and techniques, line oriented edge detection constitutes the most promising approach to satisfying both the requirement of speed and the requirement of accuracy. Accordingly, I decided to develop the eye-tracking algorithm on this basis. With this in mind, it is evident that an algorithm to detect and locate the pupil in a given image of the eye consists of two separate steps:

**Step 1:** Apply a search strategy employing some sort of pixel filter to locate a point which unambiguously can be classified as a pupil point.

**Step 2:** Assign the pupil point returned by the search strategy to the origin of operation, and use line oriented edge detection iteratively on different pairs of perpendicular detection lines to compute new estimates of the pupil position. Repeat this process until the overall position estimate, given as the average of the individual estimates, no longer changes from one iteration to the next.

In order to minimize the probability of returning a non-pupil point as input to Step 2, care has to be taken when choosing the search strategy in Step 1. Also, the fraction of the total time for the algorithm spent searching for a pupil point has to be minimized in order to have as much time as possible available for Step 2, since it is not known at the outset of the procedure how many iterations will be needed to arrive at a final estimate. In addition, a reasonable choice has to be made as to the edge detection operator(s) to be employed in Step 2. From the list of promising aspects given in Section 4.2.2, it is clear that some sort of first-order gradient operator would be best suited for this task, considering the superior response of the Sobel-operators to the relatively weak pupil-iris contour of the test image in Fig. 4.2(a). It would be desirable, however, to modify these operators in order to maximize their response.

In the next chapter, both the search strategy and the design of a suitable edge detection operator are given thorough treatment.

# Chapter 5

# OCTOPUS—An Eye Tracking Algorithm

## 5.1 Introduction

In this chapter, the proposed eye-tracking algorithm is given a thorough presentation. The name of the algorithm, OCTOPUS, arises from the fact that the search part of the algorithm can be compared to a four-armed octopus looking for the sea. Also, the line-oriented edge detection, as described in the previous chapter, is for each iteration performed in a pattern similar to an eight-armed octopus.

### 5.1.1 Problem Definition and Clarification

The problem definition to which my thesis work has been related was given as the *algorithmic* problem definition in Section 1.3. For the sake of easy reference, it is repeated here:

**Problem definition:** Design, implement and, as extensively as possible, test an algorithm to, on the basis of a given image of the eye, determine the position of the (centre of the) pupil, relative to some given reference point. The algorithm must to be fast enough to run in the CPU of the computer, and has to satisfy the following requirements:

- The position of the pupil is to be determined with an accuracy corresponding to 1 pixel in the digitized image.
- Whenever the pupil cannot be found (due to blinking, disturbances, etc.), an error signal is to be given.
- The total running time of the algorithm may not exceed 20 ms in the worst case.

Whether the accuracy requirement above complies with requirement 3 in Section 1.2 depends on the resolution of the image and on the fraction of the image occupied by the pupil. Preferably, the image should have a resolution of $512 \times 512$ pixels with 256 gray levels, and the eye should fill the entire image.

Some vague points in this definition have to be clarified before continuing the presentation of the algorithm. These are:

**Reference point:** "*... determine the position... relative to some reference point...*" The upper left corner of the image has been used as reference point, having coordinates $(0, 0)$.

**Error signal:** "*... an error signal is to be returned...*" The procedure implementing Step 1 of the basic algorithm formulation given in Section 4.2.5 has been implemented as a boolean function, returning *false* if a pupil point cannot be found in an image.

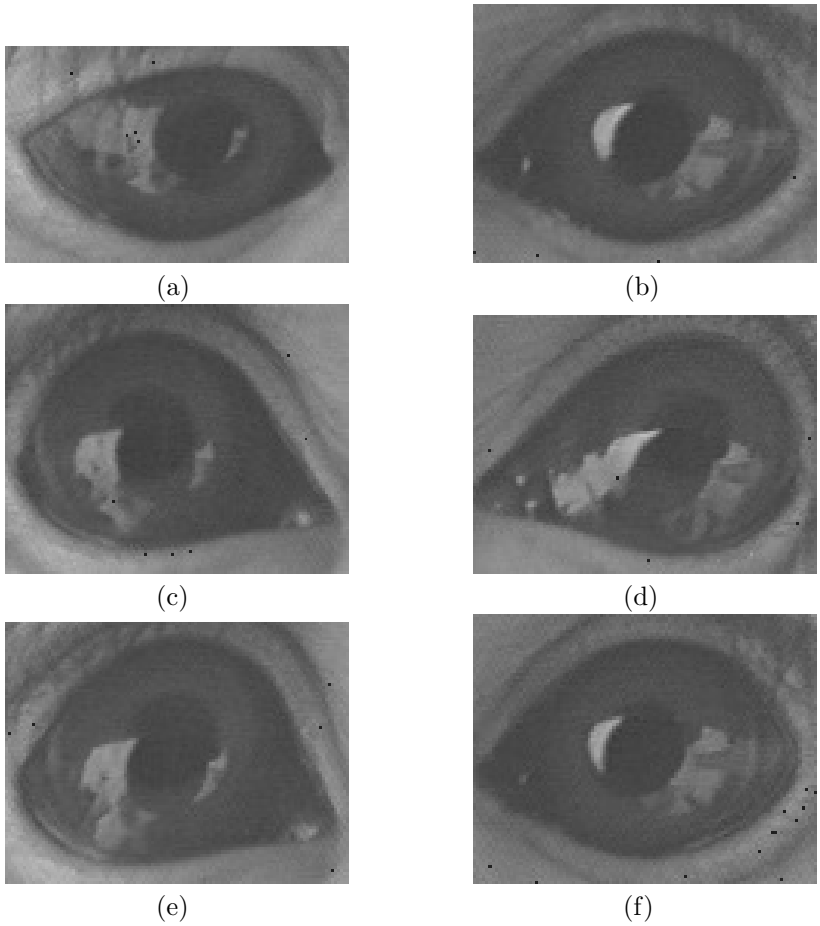



Figure 5.1: The test images used while developing and testing OCTOPUS.

**Resolution:** "*Preferably, the image should have a resolution of 512×512 pixels with 256 gray levels, and the eye should fill the entire image.*" The frame-grabber that was purchased supplies images that have a resolution of 320 × 284 pixels with 64 gray levels. However, due to limitations of the video camera, it was not possible to get close-ups of the monkey's eye that filled the entire field of view. Thus, these close-ups had to be made by shaving off the uninteresting portions of the images, leaving test images having an even lower resolution of approximately 100 × 100 pixels. Lastly, the darkest gray level in the images supplied by the frame-grabber is, for some unknown reason, 15, thus reducing the actual number of gray levels to 48.

Particularly the relatively low resolution of the test images has put limitations on the relevance of the tests that the algorithm has been subjected to. Details about the test images are given in the next section.

### 5.1.2 The Test Images

Throughout this chapter, a number of images of the monkey's eye will be used to illustrate principles, examples and results. These images, which constitute a subset of the images that were used while developing and testing OCTOPUS, are shown in Fig. 5.1. The image in Fig. 4.2(a) is identical to image (e) above. The slight variation in size/resolution of the images is due to the fact that they have been acquired by manually clipping out of larger images the portion containing the eye, as pointed out in the previous section. Clearly, the image quality is not very good. Note in particular the strong reflections present in the images, as well as their "flat" appearance. Three factors contribute to the bad quality:

**Light conditions:** The images are from a non-IR video recording of the monkey, made under "normal" light conditions in the so-called *monkey laboratory*; that is, where the neurophysiological experiments of the Eckhorn-Bauer group take place. "Normal" light conditions mean non-experimental; i.e., the light stems from ordinary fluorescent lamps in the ceiling of the laboratory. The reflections in the images are of illuminated artifacts in the near surroundings of the monkey. Note that in all images the reflections cover part of the pupil, thus making the pupil-contour non-elliptic. Since reflections in an image are of a non-additive nature, and thus completely remove information about the regions they cover, they cannot be filtered out.

**The frame-grabber:** The "flat" appearance of the images is due to relatively low contrast. This is a consequence of a bug in the frame-grabber, where increasing the contrast induces spurious dark pixels—"spots"—into the digitized frames. The number of induced spots increases with the contrast. Accordingly, a balance between high contrast and few induced spots had to be found. Actually, the induced spots have gray levels that are darker than the darkest gray level (15) in the "uncorrupted" image. Note that a small number of spots are still present in the images. Another factor which contributes to lowering the visual contrast is the fact that the darkest gray level in the images is not "black", but 15.

**Incorrect synch:** As can be seen, some of the images appear to be distorted along the vertical axis. This applies particularly to image (e) and was caused by the synch of the frame-grabber not being calibrated to handle frozen frames, which were necessary in order to be able to control which frames were going to be digitized.

However, although the images can be characterized as bad, they are good *test* images, apart from their low resolution, in that their bad quality is apt to expose the weaknesses of the eye tracking algorithm. In the following, when referring to "the given test images", the entire set of test images with which I have been working is meant, not only those of Fig. 5.1.

### 5.1.3 Structure of the Chapter

As was pointed out in the basic algorithm formulation in Section 4.2.5, the algorithm decided upon can be seen as consisting of two main steps:

**Step 1:** Apply a search strategy employing some sort of pixel filter to locate a point which unambiguously can be classified as a pupil point.

**Step 2:** Assign the pupil point returned by the search strategy to the origin of operation, and use line oriented edge detection on as high a number of line pairs as allowed by the time available in order to obtain as accurate an estimate of the centre position of the pupil as possible.

Section 5.2 presents the *swimming octopus* search algorithm as a realization of the search strategy and pixel filter required by Step 1, and Section 5.3 describes how Step 2 is realized in OCTOPUS; in particular, the gradient operators employed are elaborated upon. In Section 5.4, OCTOPUS is run on the given test test images, and the results obtained are evaluated with respect to accuracy and time consumption; and in Section 5.5, the most important aspects pertaining to the incorporation of OCTOPUS in a complete eye-tracking system are discussed.

## 5.2 Finding a Pupil Point—The Swimming Octopus

In this section, the *swimming octopus* search algorithm is described. As pointed out in the previous section, the aim of the algorithm is to locate a point in a given image that unambiguously can be classified as a pupil point. The reason for the name is that the performance of the algorithm can be compared to an octopus looking for the sea, as will become clear below. The basic idea behind the algorithm is described in Section 5.2.1; Section 5.2.2 introduces the *octopus* pixel filter; in Section 5.2.3, the "intelligent" octopus is presented; in Section 5.2.4, the overall *pick strategy* is described; and in Section 5.2.5, the overall number of operations required to detect the pupil is derived. In the following, *the algorithm* is to be understood as the swimming octopus search algorithm.

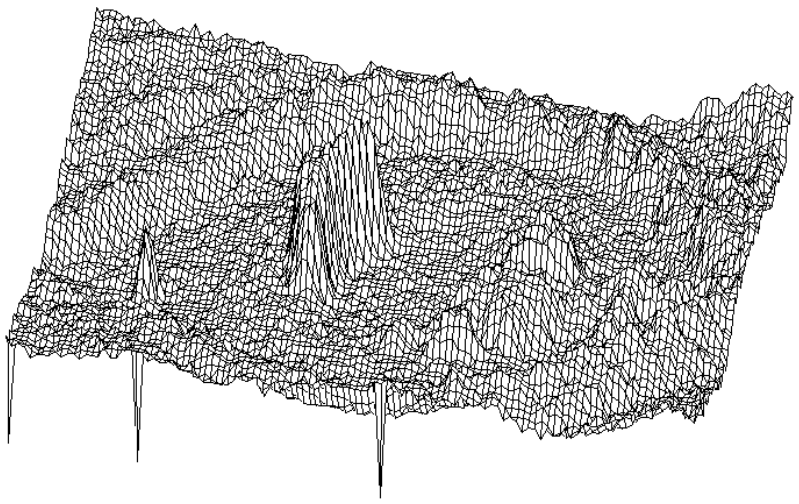


Figure 5.2: An image of the eye depicted as a three-dimensional landscape.

### 5.2.1 The Basic Idea

The foundation of the algorithm is a variant of thresholding in which all pixels with values (gray levels) below or equal to the threshold value $T_l$ have their old values replaced by $T_l$ and all pixels with values above $T_l$ retain their old values. The effect of applying thresholding of this sort to an image can be illustrated by picturing the image as a three-dimensional landscape, as depicted in Fig. 5.2. The negative peaks correspond to the aforementioned spots, whereas the positive peak to the left in the image represents an actual bright point (compare Fig. 5.1(b)). When the image is thresholded, $T_l$ can be viewed as representing the surface level of a mostly subterranean lake underneath this landscape. Wherever the landscape descends below $T_l$, this lake comes to the surface, forming "ponds" whose sizes depend on the extent of the corresponding sub-lake portion of the landscape. This is illustrated in Fig. 5.3. Note that, although the pupil is hardly discernible in Fig 5.2, it is clearly visible as forming the largest pond, or *lake*, after thresholding. Note also that a number of relatively large lakes are formed by non-pupil portions of the eye. This form of thresholding will hereafter be referred to as *lake-thresholding*.

The principle idea behind the search algorithm is that, bearing the above notation in mind, every pixel in a given image can be classified as being either *wet* or *dry*, wet pixels having values below or equal to $T_l$, and dry pixels having values above $T_l$. The advantage of this approach is that the darkest regions in the image, of which the pupil is assumed to constitute the largest, are assigned a unique value, $T_l$, which becomes *the* darkest gray level of the thresholded image, whereas non-lake regions in the image are left untouched. Thus, sub-lake landscape features, which characterize regions of which one is primarily interested in knowing that they belong to the set of the darkest regions, are eliminated, whereas the features of non-lake regions are preserved. Using a wet pixel as origin of operation (see Section 4.2.4), for the moment not heeding whether or not this pixel is a pupil point, it is clear that, in all directions, the first edge which is encountered corresponds to either the *shore* of the lake of which the pixel is a part, or to the shore of an *island* in this lake. Assuming that $T_l$ has been chosen so that the shores of the *pupil lake* in an image correspond as closely as possible to the actual pupil contour, and that the contrast of the image is such that the pupil lake is relatively *deep*, thus minimizing the probability of having islands in it, this approach ensures minimum (zero) image noise between the origin of operation and the pupil contour.

With images having as low contrast as the given test images, particular care has to be taken when choosing $T_l$. For the images I have used during my thesis work, I found that $T_l = 18$ ensures "optimal" pupil lakes and sufficiently small non-pupil lakes. However, due to the low contrast, the probability of encountering islands in the pupil lake is by no means ignorable. $T_l = 17$ tends to make the image completely dry, whereas $T_l = 19$ tends to "drown" large portions of it. It is evident that, if the final system were to employ equally low-contrasted images, only a small change

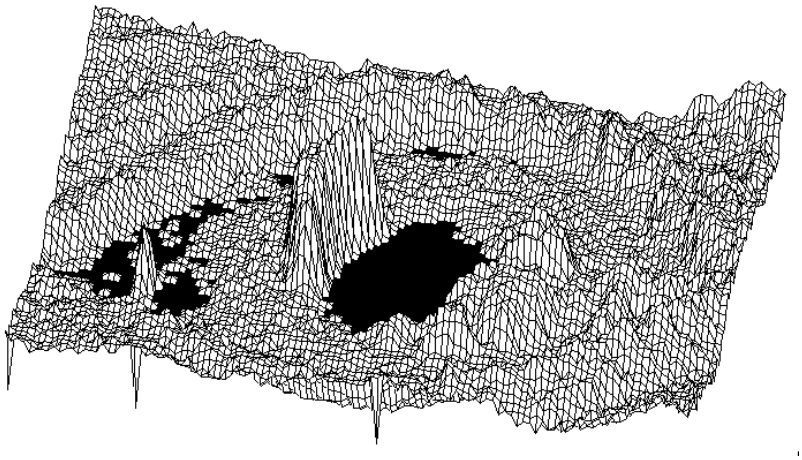


Figure 5.3: The image in Fig. 5.2 after thresholding with $T_l = 18$.

in light conditions would destroy this extremely frail balance. However, employing a frame grabber supplying images whose gray levels occupy a large portion of the desired gray level range (256) would by and large eliminate this problem.

### 5.2.2 Introducing the Octopus

From the above, it is clear that the first requirement that an actual pupil point has to satisfy in order to be recognized as such, is that it be wet. However, as is seen from Fig. 5.3, the number of wet non-pupil points in an image cannot be assumed to be zero. Thus, to avoid classifying a non-pupil point as a pupil point, some sort of neighbourhood filtering has to be done. In Section 4.1.2 it was stated that, by means of carefully thresholding the value resulting from some sort of neighbourhood averaging, it is possible to design a pixel filter which only lets points through that are actual pupil points. In the following, the pixel filter employed by the search algorithm is described.

#### The Octopus Pixel Filter

The pupil lake is assumed to constitute the largest lake in an image. Thus a point can safely be classified as a pupil point if it is known to be part of a lake whose size is in the order of the average pupil size. More precisely, the neighbourhood about the candidate pixel over which the filtering is made must have a radius slightly smaller than the minimum radius of the pupil.

The pixel filter, hereafter referred to as the *octopus* (not to be confused with OCTOPUS, the name of the entire eye-tracking algorithm), has the shape of a four-armed octopus as seen from above, with its *arms* stretched in the *north*, *east*, *south* and *west* directions, respectively. North is defined to be upwards in the image. See Fig. 5.4. The radius $r_o$ of the octopus (actually the length of its arms) is in the current implementation set to be 80% of the minimum pupil radius, which, in the case of the given test images, is estimated to be approximately 10% of the horizontal image size. The arms of the octopus are viewed as consisting of *sensors*. When applied at a given location $(x, y)$ in an image, each sensor registers and reports whether or not the pixel it covers is wet. The *first swim-criterion* that has to be satisfied for the octopus to report that it is *swimming*[1], i.e., that it has found the pupil lake, is that its central sensor, covering the pixel at $(x, y)$, as well as its *hands*, i.e., the outermost sensors of each arm, report wetness. If this requirement is satisfied, the octopus starts to count how many of its sensors that report wetness. If this number is higher than a given fraction $T_s$ of the total number of sensors, the octopus is swimming and consequently reports that the pixel at location $(x, y)$ represents a pupil point. This last requirement will be referred to as the *second swim-criterion*.

[1] Hence the name of the algorithm.

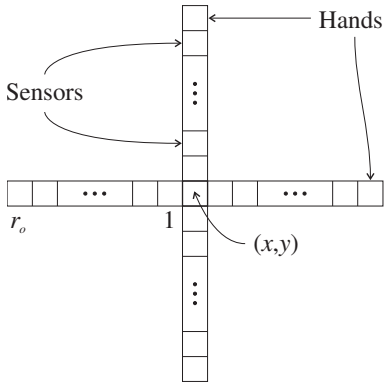



Figure 5.4: The *octopus* pixel filter. $r_o$ is its radius i pixels.

**Satisfying the Second Swim-Criterion**

The probability that the point which the octopus reports to be a pupil point actually is a non-pupil point depends on the threshold value $T_s$. With low contrast images like the given test images, it is often impossible to choose a value for $T_l$ which eliminates all non-pupil lakes in the images. Also, some of the non-pupil lakes may be relatively large in comparison with the pupil lake, and have small "ponds" dispersed around them. Thus there is a relatively large probability of having the first swim-criterion satisfied outside the pupil lake. As a consequence, $T_s$ has to be chosen as high as possible in order to minimize the probability of recognizing the corresponding point as a pupil point. Preferably, $T_s$ should be chosen near 1. However, in low-contrasted images the number of islands in the lakes, including the pupil lake, tends to be relatively high, and thus choosing $T_s = 1$ may cause no actual pupil points to be recognized as such. Accordingly, a balance has to be found. With the given test images, I found that $T_s = 0.8$ caused no non-pupil points to be returned, whereas lower values for some images caused the recognition of non-pupil points as pupil points. Values of 0.9 or higher caused no pupil points to be recognized in some of the images.

A point which requires some elaboration is that, with the given test images, lower values for $T_s$ made the search algorithm run faster. This can be attributed to the poor image quality, and in particular to the low contrast. The average gray level of the pupil lakes in the images was found to be 17, making the average depth of any pupil lake 1 (given that $T_l = 18$). Thus local variations in the pupil with positive amplitudes of 2 and more are apt to cause islands to appear in the pupil lake. Hence, if the octopus actually *is* in the pupil sea, the probability that one or more of its arms cross at least one island, thus reducing the number of sensors reporting wetness, is relatively high. Accordingly, the probability $p$ that the second swim-criterion will not be satisfied at a given location is higher than it would be if there were none or few islands in the pupil lake. Evidently, $p$ is inversely proportional to $T_s$. Since, in addition, the number $n$ of points in the image that have to be filtered in order to land at a point satisfying the swim-criteria is inversely proportional to $p$, it follows that $n$ is directly proportional to $T_s$. Thus, when $T_s$ is reduced, the time required to find a point satisfying the swim-criteria is also reduced. If the image quality were better, however, there would seldom be islands in the pupil lake, and accordingly an actual pupil point would be recognized as such more or less independently of $T_s$, thus making the search time independent of $T_s$ as well. Consequently, when working with "good" images, $T_s$ should be chosen as high as possible, preferably 1, since it would be preferable to have the octopus return pupil points as near to the centre of the pupil as possible, and low values of $T_s$ imposes the danger of the octopus settling with one arm on the "beach" of the pupil lake, its hand in a small pond[2].

[2]The danger of having small ponds on the shores of the pupil lake decreases with increased image quality. With the given test images, however, this was a common phenomenon.

### 5.2.3 The "Intelligent" Octopus

One approach to finding a pupil point would be to apply the octopus as described above to every pixel in the image starting, say, in the upper left corner (the reference point), until a pupil point had been detected. This would, however, not be a very time-efficient solution. Another approach would be to define a grid covering the image and only apply the octopus to the grid points, using some strategy as to the order in which the points are subjected to filtering. With this approach, the time needed to detect a pupil point would depend on two factors: The grid spacing and the application strategy. An even better approach would be to incorporate into the octopus some sort of mechanism which, from its current location, iteratively moves it in the most probable direction with respect to finding the pupil lake. When this direction is found to be the $\vec{0}$ vector, the swim-criteria are applied to determine whether or not the octopus swims. This is the approach taken in OCTOPUS, and allegorically it can be described as the octopus looking for the sea.

#### Selecting a Starting Pixel

In order for the procedure coarsely outlined above to proceed, an initial point, or *starting pixel*, from which the octopus starts looking for the sea, has to be chosen. The approach taken in OCTOPUS to this problem, is to choose the starting pixel semi-randomly from a predefined set **S** of *candidate pixels*. Each time a pixel has been chosen as starting pixel, it is removed from **S**, thus avoiding that the same candidate pixel is chosen multiple times. The pixels initially in **S** are evenly distributed over the entire image, and their number depends only on the fraction of the image occupied by the pupil and not on the image resolution. Starting pixels continue to be selected from **S** until either the lastly selected starting pixel caused the octopus to report detection of a pupil point or **S** is empty. In the latter case, the pupil could not be found in the given image, and an error signal is given. In Section 5.2.4, a detailed description is given of the semi-random manner in which starting pixels are selected from **S**.

#### Determining the Most Probable Direction to the Pupil Lake

From a given pixel in an image, the most probable direction to the pupil lake is determined by letting the sensors of each arm successively register whether or not the pixel it covers is wet, starting at the hand and moving towards the centre (or *body*) of the octopus, stopping if a wet pixel is detected or if the body is reached without having detected any wet pixels. When a sensor sees a wet pixel, a *pulling force* is associated with the arm to which it belongs, equal to the distance from the sensor to the body of the octopus. E.g., if the hand of the north arm covers a wet pixel, a northward pulling force equal to $r_o$ is associated with it. Otherwise, the sensor next to the hand is made current, and the procedure is repeated. Thus, if sensor number 3 from the hand detects a wet pixel, the pulling force associated with the arm is set to $r_o - 3$. If the body is reached without having detected any wet pixels, the pulling force associated with the arm is set to zero. After each arm has had a pulling force associated with it, the octopus is moved to a new location whose offset relative to the old position is given by the vector sum of the individual pulling forces. By using the pixel of the new location as starting pixel, the above procedure is repeated until the octopus remains in the same position, that is, the vector sum of the individual pulling forces is zero. When this is the case, the swim-criteria are applied to determine whether the current location corresponds to a pupil point.

The principle is illustrated in Fig. 5.5. Each square corresponds to a pixel in an image, the shaded region in the lower part of the figure represents a lake, and the solid cross represents the octopus in its initial location, with radius $r_o = 6$. The north and west arms are seen to be completely dry, and thus have pulling forces of zero associated with them. The south arm has its hand in the lake, causing a force of 6 ($r_o$) to be associated with it, whereas the east arm becomes a force of 3 associated with it. The arrows indicate the strength of the pulling forces. By forming the vector sum of the individual forces, the new location is found, and the octopus is moved, as indicated by the dotted cross. In the next iteration all pulling forces are 6, except the west force, which is 5, causing the octopus to retain its vertical position and moving one pixel eastwards, as indicated by the first small arrow; thereafter it retains its horizontal position and moves one pixel southwards, as indicated by the second small arrow. And after that, since the pulling forces here are the same in all directions, it remains in the same position, thus causing the swim-criteria to be applied. As

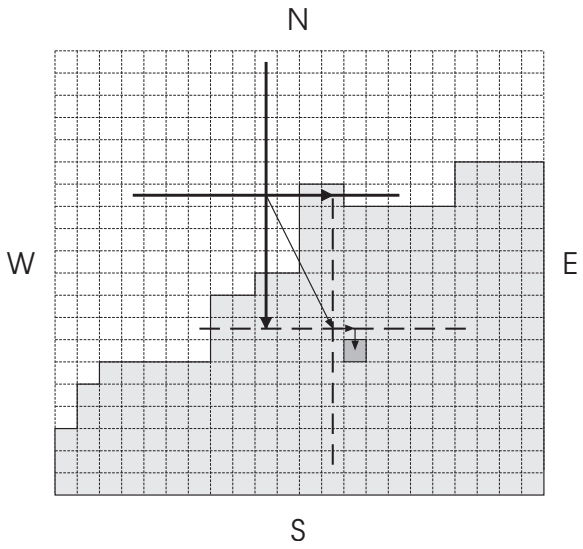



Figure 5.5: The principle behind the "intelligent" octopus.

is evident, they will both be satisfied, since all the sensors detect wetness, and thus a pupil point has been found.

One thing that has to be considered is the danger of the octopus starting to jump back and forth between two positions in the image. This would for instance happen if the lake into which it has jumped is one pixel to narrow for it to fit, say, vertically. In this case, it would alternately have its north and south hand on land, whereas the other would be in the lake, causing it to enter an infinite loop of jumping back and forth. Generally, the danger of this happening is largest whenever it jumps into a lake that is too small for it to fit in either direction; it is bound to happen if the east-west or north-south shore-to-shore distance with respect to its current location in the lake is smaller than $2r_o + 1$ and is an even number of pixels. To avoid OCTOPUS entering an infinite loop like this, there is an upper limit to the number of jumps the octopus can do without settling. In the current implementation, this limit is set to 10.

#### Settlement Categories

There are four categories of locations at which the octopus can settle. The first corresponds to the octopus being entirely on dry land. That is, none of its sensors detect wetness. Evidently, this can only happen if the initial starting position is completely dry, since once it has detected a lake, it does not fall out of it. The second category corresponds to a location where the octopus actually is on dry land, but has all of its arms crossing small ponds at equal distances from its body. A special case of this scenario is when it has its hands as well as its body in small ponds, but the major part of its other sensors are dry. In this case, the first swim-criterion will be satisfied, thus invoking the second, which causes the location to be discarded as a pupil point (that is, if $T_s$ has been assigned a reasonable value). The third category is when it settles in a lake that is too small for it to fit. That is, its body is in the lake, but either two or all of its hands are on land. Note also that it is in this case that the danger of entering an infinite loop as described above is largest. Evidently, locations belonging to the first and third categories will not satisfy the first swim-criterion, obliterating the need to apply the second. The last category is when the octopus jumps into a lake which is large enough to contain it. This is the scenario depicted in Fig. 5.5. In this case, the first swim-criterion will always be satisfied, and the second has to be applied to determine whether or not the given location corresponds to a pupil point.

### 5.2.4 The Pick Strategy

In the previous section it was said that the pixels in the set **S** of candidate pixels were evenly distributed over the entire image, and that their number was determined by the fraction of the

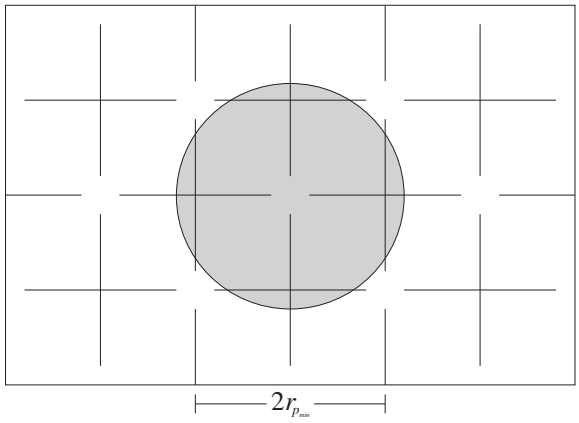



Figure 5.6: The distribution of the candidate pixels of **S** in the image. $r_{p_{min}}$ is the minimum pupil radius in pixels.

image occupied by the pupil, and not by the image resolution. It was also mentioned that the points supplied as starting pixels to the "intelligent" octopus are selected in a semi-random manner from **S**. Below, the distribution and selection aspects of the *pick strategy* are given separate treatment.

### The Distribution of the Candidate Pixels in the Image

Basically, the pixels in **S** are grid points of a grid superimposed on the image, whose spacing is equal to the estimated minimum pupil radius. However, not all grid points of this grid are candidate pixels, as shown in Fig. 5.6. For the sake of clarity, crosses of equal size to the octopus have been superimposed at each location of a candidate pixel. As is seen, the spacing between candidate pixels in each row and column of the grid is twice the estimated minimum pupil radius $r_{p_{min}}$, and their horizontal and vertical alignment alternates with an offset of $r_{p_{min}}$ every second row and column, respectively. Evidently, since the grid spacing is equal to the estimated minimum pupil size, the number of candidate pixels in **S** depends only only on the fraction of the image occupied by the pupil.

Since the neurophysiological experiments into whose setup OCTOPUS is going to be incorporated take place in relative darkness, the only source of visible light being the inducing monitor (cf. Section 1.2), it is clear that the actual radius of the pupil in the images supplied to OCTOPUS will be considerably larger than the estimated minimum size. If assuming that the actual pupil radius will be at least 20% larger than the estimated minimum[3], careful examination of Fig. 5.6 reveals that the minimum number of starting pixels that cause the octopus to jump into the pupil lake is 4. If the image quality is relatively good and the lake level $T_l$ is chosen so that the number of islands in the pupil lake is minimal the probability of the octopus settling in the pupil lake without recognizing its location as a pupil point is negligible (cf. Section 5.2.1).

### Selecting Starting Pixels

One approach to selecting starting pixel from **S** would be to select them at complete random. However, taking into account that in most cases the pupil occupies the central portion of the image, this would cause an unnecessary waste of time, in that, on the average, a relatively large number of peripheral pixels would be supplied as starting pixels before supplying a central pixel. Thus, in OCTOPUS, a *semi-random* selection procedure is employed. The basis of the procedure is the division of the image plane into *plausibility areas*, each of which is assigned a *priority level* according to the probability of its containing the pupil. The idea is illustrated in Fig. 5.7.

When the search procedure is invoked on an image, the priority level is set to 1, which corresponds to plausibility area 1 in the figure. Thus, as long as the priority level is 1 and the

[3]In the given test images, made under relatively bright light conditions (cf. Section 5.1.2), the actual pupil radius is $\sim 30\%$ larger than the estimated minumum.

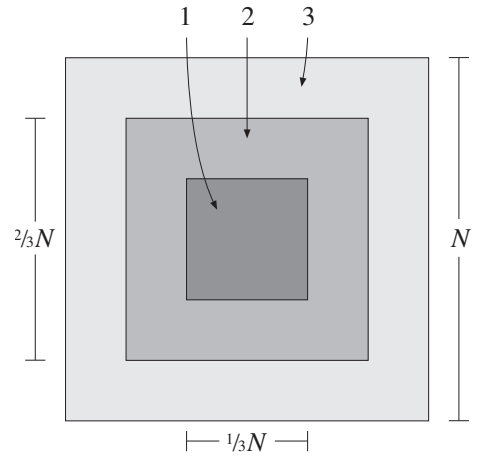



Figure 5.7: The division of the image plane into plausible areas. The gray levels of the areas correspond to the probabilities of their containing the pupil.

corresponding subset of **S** is not empty, starting pixels are selected randomly from this subset. If the pupil could not be found in plausibility area 1, the priority level is incremented, and starting pixels are selected randomly from plausibility area 2. Note that plausibility area 1 is a proper subset of plausibility area 2. However, since plausibility area 1 is empty when the priority level is incremented, selections are only made from the enclosing portion of plausibility area 2. The number of plausibility areas can in principle be chosen freely, but is in the current implementation set to 3, as shown Fig. 5.7. Each time the subset of **S** is empty that corresponds to the plausibility area given by the current priority level, the priority level is incremented, and selections are made from the enclosing plausibility area. If the subset of **S** corresponding to the outermost plausibility area—**S** itself, that is, corresponding to the entire image—has been emptied without finding the pupil, the pupil is assumed not to be present in the image, and an error signal is given.

With the given test images it turned out that, in nearly all cases, the pupil was found without having to leave plausibility area 1. In terms of time consumption, the search algorithm was found to run 4–5 times as fast with plausibility areas as it did when starting pixels were chosen from **S** at complete random. The effect of the plausibility areas is that the average image area that has to be searched in order to detect the pupil is drastically reduced without imposing the danger of not detecting the pupil if it should happen to be located outside this area. Moreover, the average time needed to detect the pupil when located peripherally in the image is only slightly more than the average time[4] needed for general pupil detection without plausibility areas.

### 5.2.5 Number of Operations

Assuming that the fraction of the image occupied by the pupil is constant, it is clear that the number of candidate pixels in **S** is $O(1)$ (cf. Section 5.2.4). The number of operations needed to determine the strength of the pulling forces exerted by each arm at a given location depends on the radius of the octopus in pixels. Assuming that the fraction of the image occupied by the pupil is constant and that the image is of size $N \times N$, $r_{p_{min}}$ is $O(N)$. Since $r_o$ is given in terms of $r_{p_{min}}$, the number of operations needed to determine the pulling forces is $O(N)$. The number of jumps that the octopus can do from a given starting pixel is limited by a constant, and is thus also $O(1)$. The number of operations needed to apply the first swim-criterion at a location is 5 and accordingly $O(1)$. For the second swim-criterion, the number of operations is proportional to $r_o$ in pixels and is thus $O(N)$. It should be noted, however, that each of these operations consists of looking up a boolean value in an array, and thus is very cheap computationally. Nevertheless, the overall number of operations for the swimming octopus search algorithm as presented above is $O(N)$.

[4] 1–2 ms with the given test images.

## 5.3 Determining the Position of the Pupil

The principle behind the line oriented edge detection algorithm employed in OCTOPUS was given a fairly thorough presentation in Section 4.2.4. Thus the focus in this section is on the different application specific aspects of the general algorithm. In Section 5.3.1, an operational description of how line oriented edge detection has been tailored to suit the specific needs of OCTOPUS is given, and in Section 5.3.2, the gradient operators employed to detect the pupil contour are presented. In Section 5.3.3, the number of operations required to determine the position of the pupil is derived.

### 5.3.1 Operational Description

The original formulation of line oriented edge detection, as stated in Section 4.2.4, says that the algorithm operates by proceeding, from the origin of operation, in different directions, applying some edge detecting operator successively on every pixel encountered in each direction, and, if an edge point is recognized, registering the coordinates of this point. By registering the coordinates of the oppositely positioned edge points along two perpendicular detection lines, an estimate for the position of the pupil can be computed using the procedure illustrated in Fig. 4.3. By taking the average of several estimates thus computed, an improved estimate of the position is obtained. Accordingly, as stated in the basic formulation of the OCTOPUS algorithm in Section 4.2.5, line oriented edge detection must be applied to as high a number of line pairs as needed to arrive at an overall estimate that does not change from one iteration to the next.

#### The Direct Approach

The normal approach to tracing a line in the discrete $xy$-plane is to choose one parameter, say $x$, as the *running parameter*, and then solve for the other parameter, in this case $y$, by using the general slope-intercept equation $y = ax + b$. Which parameter to choose as the running parameter depends on the slope $a$ of the line to be traced; if $-1 \leq a \leq 1$, $x$ is used, otherwise $y$ is used; this is to ensure the connectivity of the traced line. Thus, in the general case, performing line oriented edge detection along a pair of perpendicular detection lines involves one addition and one multiplication, respectively one subtraction and one division for each value taken on by the running parameters $x$ and $y$, in order to compute the location $(x_i, y_i)$ of the pixel to which to apply the edge detection operator.

If $(x_o, y_o)$ designates the origin of operation, an approach to implementing the general formulation above would be to displace the origin of the image plane by a vector $[x_o, y_o]^T$, thus making it coincide with the origin of operation and making all detection lines intercept at $(0, 0)$ in the transposed image coordinate system. This would make it possible to represent each perpendicular pair of detection lines by a single constant, namely the slope $a$ of one of them. The slope $a'$ of the other would be given by $a' = 1/a$, and the intercepts of both detection lines with the $y$ axis would be 0. On this basis, a set of detection line pairs can be maintained by defining an array of slopes varying from 0 to $\infty$, which corresponds to rotating the detection lines 90 degrees, thus in principle covering all possible directions. The actual resolution of the directions contained in the array depends directly on its size. An estimate of the pupil position can be computed by picking a slope $a$ from the array, using its magnitude to determine whether $x$ or $y$ is to be the running parameter, and then performing line oriented edge detection along the pair of detection lines that corresponds to $a$. Since this procedure should be repeated until the overall estimate no longer changes from one iteration to the next, it is important that the number of elements in the slope array be high enough to allow this.

There are a number of objections to this approach, despite its being a direct implementation of Step 2 of the basic algorithm formulation. For one, the computations necessary to obtain the coordinates of each point along the detection lines impose a computational overhead which, if possible, should be avoided. Secondly, and most important, the slope-intercept representation of straight lines is unable to express vertical lines, and near-vertical lines have slopes with very high magnitudes, which, in addition to being cumbersome to handle, decrease the computational accuracy. Thus it would be desirable with a technique having all the advantages of the above approach, but without the drawbacks.

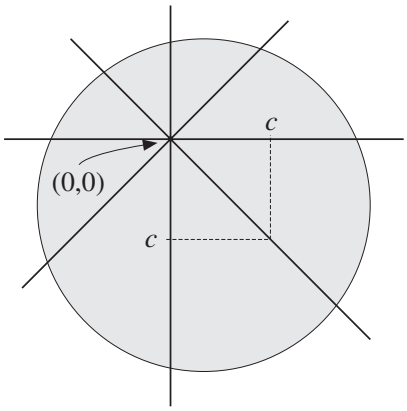



Figure 5.8: Detection lines along which positions do not need be computed.

**Repositioning the Origin of Operation**

The only way to avoid the computational overhead described above, is to reduce the number of detection lines to four. The only four detection lines along which no computations are necessary to obtain the positions are the ones occupying the horizontal, vertical and two diagonal directions, as shown in Fig. 5.8. For the horizontal line only the $x$ parameter is varied, for the vertical only the $y$ parameter, and for the two diagonal lines both parameters are varied simultaneously so that either $x = y$ or $x = -y$.

As is seen, these lines form only two pairs of perpendicular detection lines, thus reducing the number of estimates for the pupil position to two. Evidently, an overall estimate computed from only two iterations of line oriented edge detection will not suffice to satisfy the requirement of accuracy—particularly since the procedure should continue to iterate until the overall estimate no longer changes from one iteration to the next. Apparently, however, with the given origin of operation, no more iterations are possible without returning to the direct approach which was deemed not satisfactory above. There is, however, an obvious solution to this problem: Perform the two first iterations, obtain an overall position estimate and move the origin of operation to this estimated position. Thus another two iterations can be made, resulting in an improved overall estimate. This procedure is, as for the above procedure, repeated until a final overall estimate is obtained. In fact, the only principal difference in operation between the above technique and this, assuming that the "quality" of an estimate is constant with respect to the location of the origin of operation in the image, is that with the latter, the origin of operation has to be updated every second iteration.

The process of iterating the above procedure until an overall position estimate has been found may, if the intermediate estimates for some reason are distributed over a relatively large portion of the pupil, cause the available time interval of 20 ms to be exceeded without having arrived at a final overall position estimate. To avoid this happening, an upper limit to the number of iterations can be determined. This number has to determined on the basis of the time required to perform one iteration[5]. If it should happen that the allowed number of iterations has been carried out without having arrived at a final overall estimate, the current estimate has to be returned. Clearly, the estimate thus returned would hardly satisfy the requirement of accuracy. However, assuming a relatively good image quality and thus minimizing the spatial distribution of the intermediate estimates, the probability of this happening will practically be zero.

Note that, when combining an intermediate position estimate with the current overall estimate to obtain an improved overall estimate, the current overall estimate has to be weighted according to the number of intermediate estimates that contribute to it. Otherwise, if the "improved" estimate were computed as the average of two equally weighted image positions, one being the overall and the other being an intermediate estimate, all the contributions made to the overall estimate by

[5]In the current implementation, this number is set to 55, given that each iterations requires $\sim$ 0.35 ms (cf. Section 5.4.2). The lowest number of iterations required to arrive at a final overall estimate for all test images is 5.

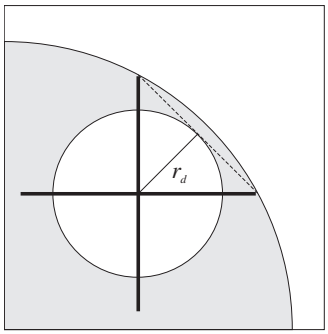



Figure 5.9: The shaded region designates a portion of the pupil lake, and the white circle designates the "dead region" about the origin of operation, inside which the sought pupil contour will not be found.

previous intermediate estimates would be lost, and a situation could occur in which the origin of operation is moved around in the pupil without ever settling at a final position. This also applies to the direct approach above.

**Skipping the "Dead Region" about the Origin of Operation**

If the image quality is relatively good, that is, when choosing a reasonable value for $T_l$, the extent of the pupil lake corresponds fairly close to the actual extent of the pupil and does not extend beyond the actual pupil contour, there is a way of reducing the time needed for one iteration of the above procedure. Recall how the initial origin of operation is supplied by the swimming octopus search algorithm. Evidently, if $T_s \approx 1$ (cf. Section 5.2.2, p. 48) and the above requirement of the pupil lake not extending beyond the actual pupil contour is satisfied, an origin of operation cannot be supplied that lies closer to the pupil contour than $r_d = r_o/\sqrt{2}$. This is illustrated in Fig. 5.9. The shaded region designates a fraction of the (idealized) pupil lake, and the white circle designates a "dead region" about the origin of operation inside which the probability of locating the pupil contour is zero. Thus, when moving outwards along the detection lines, looking for the pupil contour, the initial $r_d$ pixels of the horizontal and vertical detection lines and the initial $r_d/\sqrt{2}$ pixels along the diagonal detection lines can be skipped, reducing the overall number of locations at which to apply the edge detection operators.

With low-contrast images, however, like the given test images, the "dead region" may not be entirely "dead". In other words, the initial origin of operation may lie closer to the actual pupil contour than $r_d$. This can be attributed to the fact that $T_s$ has to be chosen relatively low to ensure that some actual pupil point are recognized as such. Since the pupil lakes in low-contrasted images tend to have ponds dispersed around them, some of which may lie on the outside of the pupil boundary (cf. Section 5.2.2), there is a probability that the octopus will settle at a location at which it has one or even two of its arms crossing the pupil boundary. This happens when it jumps to a location at which the first swim-criterion is satisfied by its having one (or two) of its hands in a non-pupil pond. Consequently, by skipping the assumed "dead region", one may happen to start looking for the pupil boundary after actually having crossed it, thus never finding it. One solution to this problem would be to try to choose a smaller value for $r_d$. It is, however, not clear which criteria to use when choosing this smaller value, since nothing is known about the distribution of the non-pupil ponds surrounding the pupil lake. Consequently, the recommended solution, and the one I chose, since the probability of this happening was relatively high with the given test images, is to set $r_d = 0$.

**Skipping Lake Pixels**

A way of reducing the time required for one iteration of the above procedure even further, assuming that the pupil lake does not extend beyond the actual pupil contour, is, during the process of moving along the detection line, to investigate whether or not the pixel *in front* of the current pixel is wet.

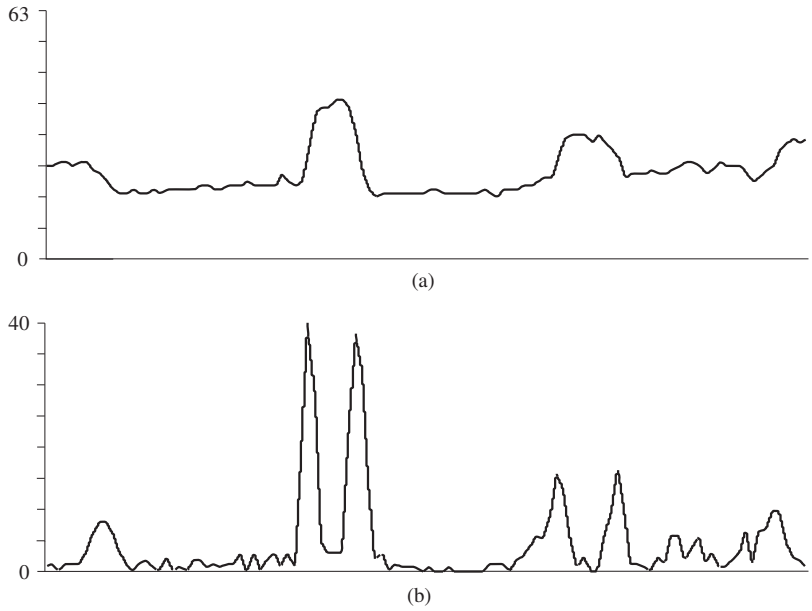



Figure 5.10: (a) The profile of a pixel row in an image. (b) The result of applying the mask in Fig. 3.9(b) to this profile.

If it is dry, the edge detection operator is applied at the corresponding location. However, if it is wet, there is no need to apply the edge detection operator at this location, since the pupil contour will not be detected at a location where both the pixel at the location and the pixel in front of it in the direction of movement are wet. The pixel at the given location is known to be wet since, with this technique, every pixel in the direction of movement is tested for wetness, so that when testing a given pixel for wetness, it is known that all pixels preceding it are wet. After a dry pixel has been detected, the edge detection operator is applied to every pixel along the remaining portion of the detection line until the pupil contour is detected.

Evidently, the number of locations at which the relatively costly edge detection operator has to be applied is drastically reduced by employing this technique. The largest impact of the technique is obtained when the number of islands encountered along the detection lines is minimal, so that the first dry pixel detected is in the proximity of the actual pupil contour. This corresponds to having relatively high-contrasted images with correspondingly deep pupil lakes. However, also with the low-quality test images, the effect of applying this technique was notable, and for the few images without islands in the pupil lake (cf. Fig. 5.3), the effect can be characterized as dramatic. Assuming relatively high image quality, this effect can be attributed to the fact that the number of locations at which the edge detection operator has to be applied is more or less independent of the resolution $N$ of the image, whereas the number of locations whose pixels are tested for wetness along the detection line, a computationally cheap operation, is $O(N)$.

### 5.3.2 Edge Detection Operators

In the basic algorithm formulation in Section 4.2.5, it was said that, based on the comparison of different spatial edge detection schemes in Fig. 4.2, some sort of first-order derivative operator would be best suited for the task of detecting the pupil contour. Two obvious candidates are the Sobel operators described in Section 3.6.2 and shown in Fig. 3.9. The result of applying these to Fig. 5.1(e) is shown in Fig. 4.2(c). Although the pupil contour is hardly discernible in the original image, it is clearly visible in the processed image. Thus, by moving along the detection lines originating at the current origin of operation and at each pixel appropriately thresholding the output from the Sobel operators, the pupil contour can be detected.

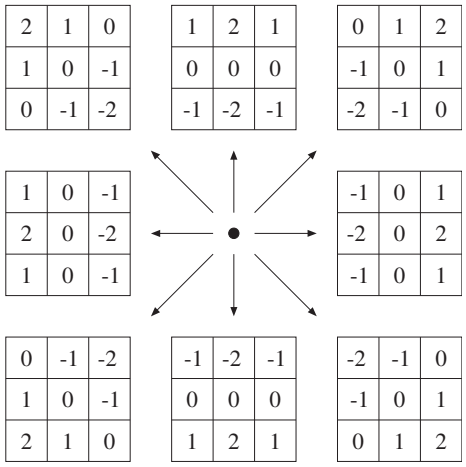



Figure 5.11: Eight masks that can be used to detect the pupil contour along the four detection lines in Fig. 5.8. The central point represents the origin of operation.

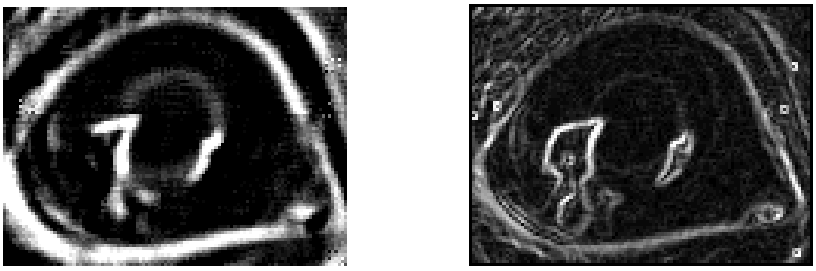
### Characteristics of the Sobel Operators

Before deciding to employ the Sobel operators directly as they are depicted in Fig. 3.9, it is necessary to examine their characteristics and to determine if and how they can be tailored to suit the task they are to perform.

Normally, assuming the image origin to be the upper left corner of the image, the Sobel operators are applied to an image in two passes, one vertical, applying the mask Fig. 3.9(a) column by column to every pixel in a top-down manner, and one horizontal, applying the mask in Fig. 3.9(b) row by row to every pixel in a left-to-right manner. In Fig. 5.10(b), the result of applying the horizontal Sobel operator to a single row in an image is shown[6]. The profile of the row itself is depicted in Fig. 5.10(a). Note that the processed profile in (b) also contains information about the neighbouring rows to the row in (a). As is seen from the figure, the operator responds to positive as well as negative edges. Note in particular the high peaks caused by the two edges of the reflection occupying the left portion of the pupil. The lack of differentiation between positive and negative edges is due to the returned response being computed as the magnitude of the actual response (corresponding to the partial gradients $G_x$ and $G_y$, cf. Eq. (3.72), p. 30). It should be noted that the pupil by no means constitutes a flat surface in the image landscape. Moreover, the irregularities present in the pupil are interpreted as edges, however small, by the Sobel operators, as is evident from the figure. Thus one cannot assume the pupil contour to be the first edge encountered along the detection lines. Instead it has to be assumed that the pupil contour constitutes the first edge encountered which causes a response higher than or equal to a threshold value $T_e$. The magnitude of $T_e$ is discussed below.

By examining the two masks in Fig. 3.9 and keeping in mind that the mask in (a) proceeds in a top-down manner and the one in (b) in a left-to-right manner, it is seen that a positive edge causes a positive actual response, whereas a negative edge causes a negative actual response. Lastly, two points should be noted about the coefficients employed in the masks. For one, the 2's cause intensity transitions in the column/row along which the mask moves to contribute more to the overall response than transitions in the neighbouring columns/rows. This is clearly desirable from a line oriented edge detection viewpoint. Secondly, the "empty" (i.e., filled with 0's) row/column of the masks lessen their sensitivity to spurious noise from neighbouring rows/columns which intercept the column/row along which the mask moves.

[6]The image in question is the one depicted in Fig. 5.1(b). The selected row passes horizontally through the approximate middle of the pupil (cf. Fig. 5.2). Note that the darkest gray level in the unprocessed profile is 15, as pointed out earlier. Note also that the contrast is relatively good in the image. It was a general trend with the given test images that the horizontal contrast was much better than the vertical (cf. Fig. 5.2).



(a) (b)

Figure 5.12: (a) The result of applying the extended versions of the diagonal masks in Fig. 5.11 to the image in Fig. 5.1(e). (b) The result of applying the "normal" Sobel masks to the same image.

### Tailoring the Sobel Masks

The primary difference between the usual manner in which the Sobel operators are applied, as described above, and the manner in which they would be applied in line oriented edge detection, is the direction of movement. They would no longer move in only two directions, top-down and left-to-right, but in eight different directions corresponding to the four detection lines depicted in Fig. 5.8. Evidently, the vertical mask would be well suited to detect the northern and southern pupil edges, and the horizontal mask would be equally well suited to detect the western and eastern edges. They would both be equally but not particularly well suited to detect the diagonal edges, and thus at least two new masks have to be designed that are better suited to this task.

Before proceeding to design the new masks, it should be noted that, when moving from a point inside the pupil, the pupil contour will always constitute a positive edge. Accordingly, approximately half of the edges encountered along the detection lines can be discarded, namely those that are negative. It was pointed out above that the lack of differentiation between the responses to positive and negative edges was due to the returned response being computed as the magnitude of the actual response. Thus, by not taking the magnitude of the actual response and discarding all negative responses, only positive edges will be detected.

By discarding all negative responses, the two masks in Fig. 3.9(a) and (b) can only be applied to detect the southern and eastern pupil edge, respectively, since the northern and western edges would cause negative responses and thus be discarded. However, by mirroring the mask in Fig 3.9(a) about the horizontal axis and the one in (b) about the vertical axis, two masks are obtained that would return positive responses to the northern and western pupil edges, respectively. The diagonal masks are obtained analogously, and the entire set of eight masks, two for each detection line, is shown in Fig. 5.11.

### Reducing the Sensitivity to Noise

As has been stated earlier, the given test images have low contrast and are also infected with additive, spurious noise. Particularly should the dark "spots" dispersed throughout the images be noted (cf. Section 5.1.2)[7]. Since the combination of low contrast and spurious noise may cause edges to be detected where no edge should be detected, it is important to reduce the spurious noise in the image as much as possible, at least inside the pupil, since this is where the edge detection takes place. In particular must the dark "spots" be more or less eliminated, since these would cause the Sobel operators to detect veritable "walls" if they should happen to be located along or next to the detection lines.

An obvious approach to more or less eliminating the irregularities inside the pupil, as well as the spots, is to process the lake-thresholded image instead of the raw image itself. As stated earlier (cf. Section 5.2.1), lake-thresholding an image causes sub-lake landscape features to be eliminated, whereas the features of non-lake regions are preserved. By appropriately choosing the lake-threshold value $T_l$, the probability of having the pupil lake extend beyond the actual

[7]These "spots" can be seen as negative peaks in the image landscape in Fig. 5.2, and are also visible in the images themselves, cf. Fig. 5.1.

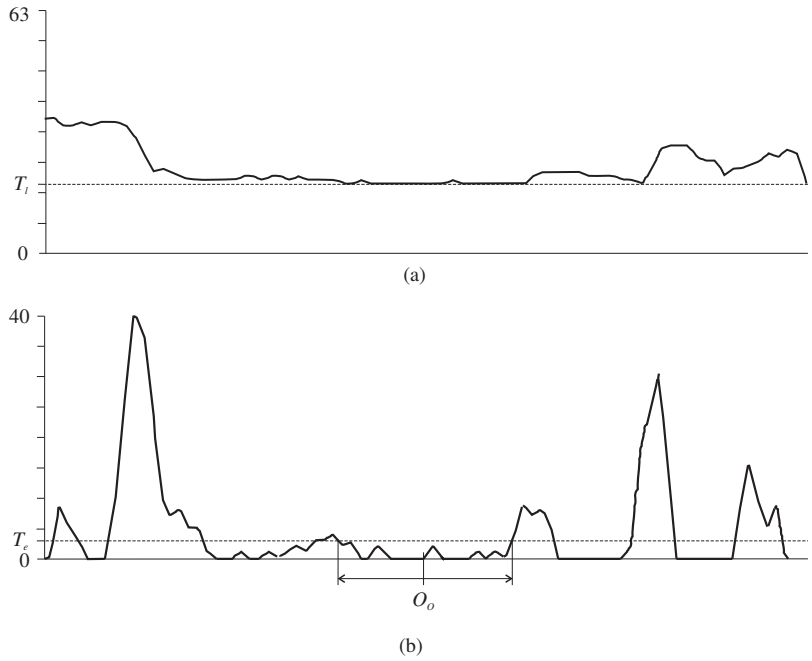



Figure 5.13: (a) The profile of the image in Fig. 5.1(d) along a north-eastwards detection line passing through the pupil. (b) The result of applying the SW and NE Sobel mask of Fig. 5.11 to this profile. $O_O$ designates the origin of operation.

pupil contour can be reduced to a minimum, albeit with low-quality images it cannot entirely be eliminated. Consequently, all sub-lake "edges" will be ignored by the edge detection operators and the first edge detected will correspond to the shore of the pupil lake, alternatively to the shores of an island in the lake. By appropriately thresholding the output from the operators, it is possible to differentiate between these edges and the actual pupil edge, assuming that the pupil edge is the most prominent of these. Most importantly, the dark "spots" will also be "filled", and thus either form small, one-pixel ponds of their own, or be part of larger lakes.

With the given test images, the number of islands in the pupil lake was on the average relatively high, and on occasions fooled the edge detection operators into believing that the pupil edge had been detected. A way of reducing the operators' sensitivity to this kind of regional noise is to enlarge the neighbourhood over which they compute the gradient. That is, the $3 \times 3$ neighbourhoods in Fig. 5.11 are extended to (e.g.) $5 \times 5$ masks by inserting two rows and two columns containing only 0's between the three existing rows and columns. Since the inserted 0's do not contribute to the computation, doing this does not increase the number of operations needed to compute the gradient at a given location, but, since pixel values are gathered from a larger neighbourhood, it reduces the sensitivity of the masks to islands in the pupil lake and to spurious noise in general. $5 \times 5$ versions of $3 \times 3$ masks will in the following be referred to as *extended*. The result of applying the extended versions of the diagonal masks in Fig. 5.11 to the image in Fig. 5.1(e), applying each mask according to which quadrant of the image relative to the origin of operation the pixel in questions belongs to, is shown in Fig. 5.12(a), together with the result of applying the standard Sobel masks of Fig. 3.9 to the same image (b). Note that the pupil contour is more predominant in (a) than in (b), and also that negative edges as seen from the origin of operation are not visible (note in particular the far side of the reflection).

### Thresholding the Output from the Sobel Operators

Evidently, the ability to differentiate between the pupil edge and spurious intensity variations in the proximity of the pupil edge depends on how the output from the employed Sobel operators is thresholded. With the given test images, the S/N ratio is so low that it often is difficult on the basis of an image profile visually to determine where the actual pupil contour is. This is clearly

demonstrated in Fig. 5.13(a), which shows the profile of the image in Fig. 5.1(d) along a north-eastwards detection line passing through the pupil. Although the pupil lake is clearly identifiable, this is not the case for the southwestern pupil contour (left of the pupil lake). Note the two islands in the pupil lake, and that the southwestern shore constitutes a *plain* whose height above the lake level is only 1. Note also that this plain, apart from three *hills* of height 2, extends almost to the margin of the eye. In Fig. 5.13(b) is shown the result of applying the extended southwest and northeast masks of Fig. 5.11 to this profile. $O_O$ designates the origin of operation. Note that the processed profile also contains information about neighbouring lines to the actual detection line, as can be seen from the hill "ranges" on both sides of the pupil lake, as well as from the two "extra" islands by the northeastern shore.

Clearly, the detected extent of the pupil along this detection line depends heavily on the threshold value $T_e$. Particularly, choosing $T_e$ too high would cause one or both of the actual pupil edges along the given detection line to remain undetected. Moreover, the edges detected and assumed to represent to the pupil contour would actually represent the more prominent edges at the margins of the eye. Furthermore, choosing $T_e$ too low would cause the edges of the islands in the pupil lake to be recognized as representing the pupil contour. As is seen from the figure, the interval of appropriate values for $T_e$ can be very narrow, depending on the image quality[8]. However, if the image quality and in particular the contrast is relatively high, so that the number of islands in the pupil lake is reduced to a minimum and the slope of the pupil contour is relatively steep, it ought to be possible to choose an overall satisfactory value for $T_e$.

Note that the standard deviation of the error in the obtained position estimate caused by the given value for $T_e$ apparently being inappropriate for the profile along a detection line in a given image will approach zero as the number of iterations increase. When the estimate no longer changes from one iteration to the next, which is the stop criterion for iteration process, the error is less than one pixel with respect to the pupil contour as seen from the current origin of operation. Thus it is of vital importance to ensure the highest possible image quality in the images fed to the OCTOPUS eye-tracking algorithm. Some requirements the images ought to satisfy in order to increase the probability of having OCTOPUS return accurate position estimates are listed in Section 5.4.3.

### 5.3.3 Number of Operations

The number of operations required to obtain one intermediate position estimate, that is, to perform line oriented edge detection four times along two perpendicular detection lines, is determined by the number of pixels from the origin of operation to the pupil contour along the given detection line. On the average, this number equals the radius of the pupil in pixels, which in turn depends on the horizontal and vertical resolution $N$ of the $N \times N$ image. Since the number $n$ of iterations necessary to arrive at a final overall position estimate, in addition to being limited upwards, generally satisfies the relation $n \ll N$, it can be concluded that the position determining part of the OCTOPUS eye-tracking algorithm performs in $O(N)$ time. It should be noted, however, that, by applying the technique of skipping lake pixels, described in Section 5.3.1, p. 55, the number of locations in the image at which the edge detection operators have to applied is reduced to $O(1)$.

## 5.4 Evaluation

In this section, the OCTOPUS eye-tracking algorithm described in the previous sections is evaluated with respect to the algorithmic problem definition given in Section 1.3. First, the overall number of operations for the algorithm is derived in Section 5.4.1. In Section 5.4.2, an implementation (previously referred to as the *current* implementation) is run on the set of test images presented in Fig. 5.1 and the results returned are discussed with respect to both accuracy and time consumption. Lastly in Section 5.4.3, some suggestions as to how to improve the performance and accuracy of the algorithm are presented.

[8]With the given test images, it was impossible to choose a value for $T_e$ which was appropriate for all images. $T_e = 5$ was found to be the best overall choice. However, as is seen from Fig. 5.13(b), this value caused unwanted results in some images.

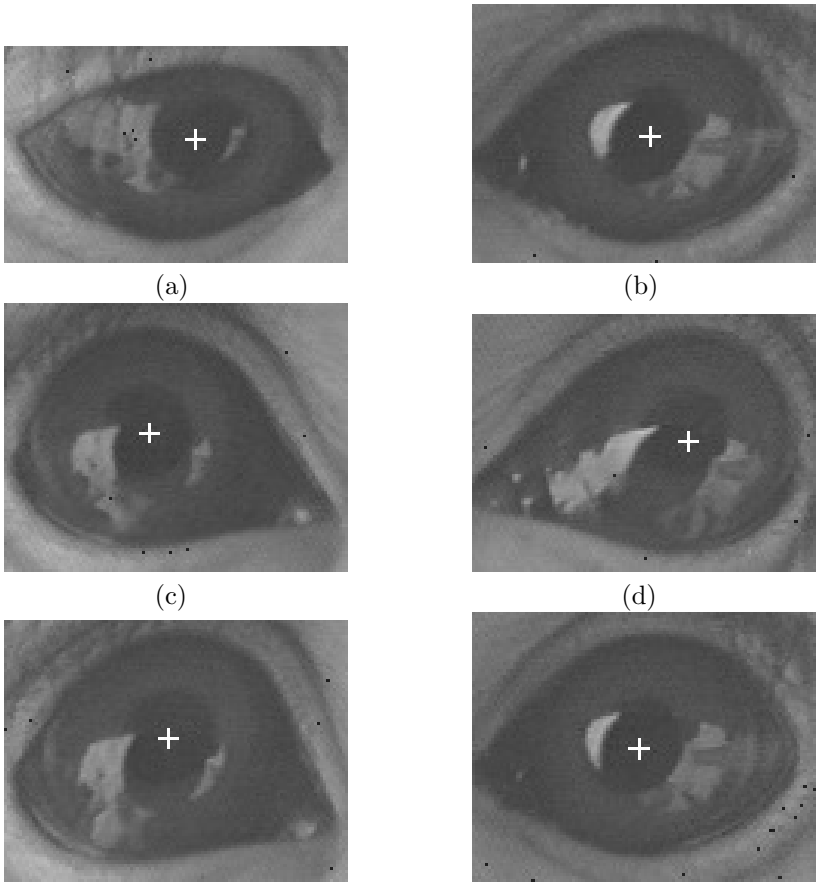



Figure 5.14: Returned estimates of the pupil position in the images in Fig. 5.1. The coordinates of the estimates relative to the upper left image corner are: (a) $x = 65$, $y = 32$; (b) $x = 59$, $y = 44$; (c) $x = 49$, $y = 44$; (d) $x = 71$, $y = 42$; (e) $x = 54$, $y = 39$; (f) $x = 54$, $y = 44$.

### 5.4.1 Overall Number of Operations

In Section 5.2.5, it was shown that the swimming octopus search algorithm, described in Section 5.2 and implementing Step 1 of the basic algorithm formulation, performs in $O(N)$ time when applied to an $N \times N$ image. In Section 5.3.3 it was shown that the position determining algorithm described in Section 5.3, implementing Step 2 of the basic algorithm formulation, performs in $O(N)$ time. Thus it can be concluded that the OCTOPUS eye-tracking algorithm, being composed of the given implementations of Steps 1 and 2 of the basic algorithm formulation, performs in $O(N)$ time. Taking into account that the number of locations in the image at which the edge detection operators of Fig. 5.11 have to employed, assuming relatively good image quality, is $O(1)$ (cf. Section 5.3.3), the advantage of OCTOPUS over a "traditional" approach in terms of computational cost is evident.

### 5.4.2 Test Results

The major amount of testing was done while developing and implementing OCTOPUS. Six of the images with which I worked during that time are given in Fig. 5.1; in addition I had another four images at my disposal. How the images were acquired is described in Section 5.1.2. The tests performed at this stage were a vital part of the development process, pointing at weaknesses in the current algorithm formulation, suggesting solutions to difficult problems and in general guiding the development process towards sounder and more time-efficient algorithm formulations. Figs. 4.2, 5.2, 5.3, 5.10, 5.12 and 5.13 were all obtained using software primarily designed to sustain this process. In the following, the focus will be on the final formulation of OCTOPUS as given in the

| | |
|---|---|
| (a) | (65,31), (65,32) |
| (b) | (59,44), (60,44), (60,43) |
| (c) | (49,43), (49,44), (50,44) |
| (d) | (71,41), (72,41) |
| (e) | (54,39) |
| (f) | (55,44) |

Table 5.1: Position estimates obtained from the figures in Fig. 5.14.

previous sections, and on its ability to satisfy the requirements listed in the algorithmic problem definition given in Section 1.3. Three points will be addressed, accuracy, distortion of the pupil's appearance and time consumption. With respect to the requirement that OCTOPUS give an error signal whenever the pupil cannot be found in a given image, refer to Section 5.1.1. The relatively low resolution of the test images ($\sim 100 \times 100$) should be kept in mind when evaluating the results, particularly with respect to the time measures obtained.

**Accuracy**

In Fig. 5.14, the position estimates returned by OCTOPUS when supplied with the images in Fig. 5.1 are shown. As is seen, the actual extent of the pupil in the images can hardly be determined visually, which imply relatively weak pupil edges. Moreover, the edges of the bright reflections, which are seen to cover portions of the pupil in all images, evidently constitute the sharpest and most clearly defined edges present. As pointed out in Section 5.1.2, these reflections are of a non-additive nature, i.e., the information contained in the regions they cover is lost, and accordingly they cannot be removed. Hence it is unavoidable that the position estimates returned by OCTOPUS are influenced by these reflections. Taking this and the inaccuracies due to the chosen value for $T_e$ not being appropriate for all images into account (cf. Section 5.3.2, p. 60), it is evident that the difference between the actual pupil centre and the position estimate returned by OCTOPUS cannot be used as an accuracy measure. Actually, due to the poor image quality, it would be practically impossible with some of the images to manually determine the actual pupil centre.

Consequently, another accuracy measure has to be employed. Since the stop criterion of the iterating position determining algorithm is that the overall position estimate not change from one iteration to the next, a natural choice is to use the variation in the estimates returned for one image as an accuracy measure. Ideally and in compliance with the requirement of accuracy given in the problem definition, this variation should be zero both horizontally and vertically. In other words, OCTOPUS should always return the same position estimate when supplied with the same image. This is, however, not the case with all the given images. The estimates obtained with the different images are given in Table 5.1. The letters refer to Fig. 5.14. As is seen, only for images (e) and (f) does OCTOPUS satisfy the requirement of accuracy. For images (a) and (d), two estimates are returned, and for images (b), (c) and (d), three different estimates are returned. It is noted, however, that the maximum deviation from one estimate to another is not more than 1 pixel, either horizontally, vertically or both. Still, with the low resolution of the given images, 1 pixel corresponds to approximately 5% of the pupil diameter, an error which hardly can be called tolerable, even without the given accuracy requirement.

This error can be ascribed to two factors. For one, the low resolution of the images limits the number of different intermediate estimates that can be obtained during the iteration process. Consequently, comparably few iterations are necessary for a final estimate to be obtained, correspondingly reducing the "weight" of this estimate. Accordingly, depending on the location of the initial origin of operation, different final estimates may be obtained which would have converged if the number of iterations were higher. Secondly, the technique of moving the origin of operation to the current overall estimate every two iterations may cause the overall estimate to converge at a location constituting the centre of the pupil as seen along the detection lines intercepting at this location, but which may not constitute the centre as seen from another location. Thus, once the overall estimate has landed at this possibly erroneous location, it will remain there no matter

| | (a) | (b) | (c) | (d) | (e) | (f) |
|---|---|---|---|---|---|---|
| $t_s$ | 0.28 | 0.22 | 0.24 | 0.36 | 0.22 | 0.19 |
| $t_p$ | 2.46 | 0.88 | 1.51 | 1.29 | 2.58 | 1.30 |
| $t$ | 2.74 | 1.10 | 1.75 | 1.65 | 2.80 | 1.49 |

Table 5.2: Time consumption of the OCTOPUS eye-tracking algorithm. All times are in ms.

| | (a) | (b) | (c) | (d) | (e) | (f) |
|---|---|---|---|---|---|---|
| $\overline{n}_i$ | 6.5 | 3.4 | 4.6 | 3.0 | 5.8 | 4.2 |
| $\overline{t}_i$ | 0.38 | 0.26 | 0.33 | 0.43 | 0.44 | 0.31 |

Table 5.3: $\overline{n}_i$: Average number of iterations required to obtain a final position estimate for the images in Fig. 5.14. $\overline{t}_i$: The average time consumption per estimate for the same images, in ms.

how many consequent iterations are performed. In Section 5.4.3 below, a possible solution to this problem is suggested.

**Time Consumption**

The time measures presented here were obtained by measuring the time needed for OCTOPUS to succeedingly obtain a position estimate for the same image 2000 times. The times needed for the two steps of the algorithm were measured separately, and the times thus obtained were divided by 2000 to obtain estimates of the times required to perform each step once. The times measured are presented in Table 5.2. $t_s$ denotes the search time, corresponding to Step 1 of the basic algorithm formulation, and $t_p$ the position determining time, corresponding to Step 2 of the basic algorithm formulation. $t$ denotes the overall time for the algorithm, computed as the sum of $t_s$ and $t_p$. All times are in ms. The letters refer to Fig. 5.14. Clearly, the requirement of determining the position of the eye within 20 ms is satisfied.

From the table, rather large variations in the measured times are observed. The variations in $t_s$ can be ascribed to variations in the number of times the octopus jumps into the pupil lake without recognizing the location where it settles as a pupil point. For image (e), the particularly long time needed to locate a pupil point is attributed to its having a relatively small pupil lake[9], actually forcing the octopus to settle at a location where it has one of its hands in a pond on the shore of the pupil lake (cf. Section 5.2.2). For $t_p$, the variations can be ascribed to the varying number of iterations necessary to arrive at a final estimate. The fact that the times constitute the averages of 2000 performances of the algorithm implies that for some of the images more iterations are generally required for a final estimate to be arrived at than for others.

In Section 5.3.1, p. 54, it was mentioned that the average time for one iteration with the current implementation was $\sim 0.35$ ms, and that the minimum number of iterations required to arrive at a final overall estimate for all test images was 5. It would, however, be interesting to know the average number of iterations required to arrive at a final estimate for each image, as well as the average time needed to perform one iteration on the given image. These numbers are presented in Table 5.3. A obvious correlation between $t_p$ in Table 5.2 and $\overline{n}_i$ is observed. Note also that for images (a) and (e), $\overline{n}_i > 5$, which above was given as the minimum number of iterations required to arrive at a final estimate for all images. This apparent inconsistence implies nothing more than that it is *possible* with these images to arrive at a final position estimate with no more than 5 iterations, but that the average number of iterations required is higher.

A last point to notice is the variation from image to image in the average time $\overline{t}_i$ required to perform one iteration. As indicated above, this time depends on the nature of the pupil lake of the given image. If the number of islands in the pupil lake is low and its extent roughly corresponds to the actual pupil extent, the technique of skipping pupil pixels when looking for the pupil edge (cf. Section 5.3.1, p. 55) increases the velocity with which the algorithm moves along the detection lines

[9] With lake level $T_l = 18$, that is, cf. Section 5.2.1.

in comparison with the case when islands often are encountered or the shore of the pupil lake lies some distance from the actual pupil contour, in which case the edge detection operators have to be applied at a comparatively high number of locations. As is evident from Fig. 5.3, p. 47, image (b), having the lowest value for $\overline{t}_i$, displays a more or less "ideal" pupil lake in this respect.

#### Summary

The validity of the presented test results with respect to the ability of OCTOPUS to function satisfactorily within the framework of the final eye-tracking system is somewhat limited. The main reason for this is the low quality of the test images I have had at my disposal. Admittedly, as pointed out in Section 5.1.2, the low contrast of the images has helped in exposing the weaknesses of the algorithm, weaknesses of which some are still present (cf. the footnote on p. 60), but the presence of the strong reflections occupying partly large portions of the pupil area has made it more or less impossible for OCTOPUS to arrive at position estimates corresponding to the actual pupil centre. One thing which has become increasingly clear during my work with OCTOPUS is that, if it is to work satisfactorily, the quality of the images supplied to it has to be better. In particular, strong reflections of the kind present in the given test images must be avoided, and the contrast has to be improved.

Another problem with the test images with respect to the validity of the test results is their low resolution, being in the order of $\sim 100 \times 100$, whereas the images with which OCTOPUS is to work when incorporated in the complete eye-tracking system are to have a resolution of $512 \times 512$. All time measures presented above relate to these relatively low-resolution images. However, none of the time measures presented above exceed 3 ms, which leaves another 17 ms available for computations without exceeding the maximum time of 20 ms. Also, letting OCTOPUS run on a faster computer than the one with which I have worked (cf. Section 1.4) would leave yet more time available. Moreover, as pointed out in Section 5.3.3, assuming a relatively high image quality, the number of locations in the image at which the relatively costly Sobel operators of Fig. 5.11 have to be applied is more or less independent of the image resolution. Accordingly, the time required to perform one iteration should not increase proportionally with the increase in resolution. All in all, the time efficiency of OCTOPUS should obliterate the danger of the 20 ms available not sufficing.

The main weakness of OCTOPUS is thus its inability to univocally arrive at a final position estimate (cf. Table 5.1). As was mentioned above, this can partly be ascribed to the low resolution of the test images, but nevertheless the problem needs to be addressed further. In the next section a possible solution to the problem is presented.

### 5.4.3 Improving the Algorithm

From the above, it is evident that OCTOPUS needs some refinement before being suitable for incorporation into a complete eye-tracking system. This pertains in particular to the accuracy of the current algorithm formulation. A problem which thus far has not been touched upon is that the appearance of the pupil as seen from the camera rarely constitutes a perfect circle. This will be given a short treatment below, along with a suggestion of how to lower the overall number of operations needed to perform one iteration of line oriented edge detection even further.

#### Accuracy

As pointed out in Section 5.4.2, the technique of moving the origin of operation to the current overall estimate every two iterations may cause the overall estimate to converge at a location constituting the centre of the pupil as seen along the detection lines from this location, but which may not constitute the centre as seen from another location. Thus, once the overall estimate has landed at this possibly erroneous location, it will remain there no matter how many consequent iterations are performed.

A way of avoiding this happening is to, instead of assigning the current overall estimate to the origin of operation, choose the new location of the origin of operation randomly from a predefined neighbourhood of the overall estimate. When deciding the extent of the neighbourhood, care has to be taken that it not extend beyond the actual pupil contour, since this would impose a danger

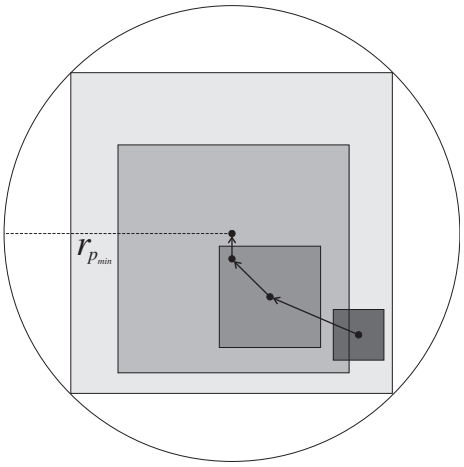



Figure 5.15: Letting the radius of the neighbourhood from which new values for the origin of operation are chosen increase dynamically with increased "weight" of the current overall position estimate.

of assigning to the origin of operation a pixel located outside the actual pupil. A possible approach to deciding the extent of this neighbourhood would be to let its radius increase dynamically with the number of iterations. In other words, as the number of intermediate estimates contributing to the overall estimate increases, thus making the overall estimate converge towards the apparent pupil centre (as opposed to the actual pupil centre, of which nothing is known), the radius of the neighbourhood approaches $r_{p_{min}}/\sqrt{2}$, as depicted in Fig. 5.15. By choosing the radius of the initial neighbourhood sufficiently small, the chance of picking a location outside the pupil is minimized. Note that the first time a location is picked, the overall position estimate already has a "weight" of 2, corresponding to the two intermediate estimates that contribute to it.

The advantage of this approach is that the locations from which new intermediate estimates are computed are dispersed over a relatively large portion of the pupil, thus avoiding viewing the pupil contour repeatedly from only one location, as may happen with the implemented approach.

Finally, the "weight" of the final overall estimate can be increased by not having as stop criterion for the iteration process that the overall estimate not change form one iteration to the next, but rather that the number of intermediate estimates contributing to the overall estimate be higher than a predetermined number. By combining this with the above approach, it ought to be possible for OCTOPUS to univocally land at a position estimate for a given image.

### Distortion in the Pupil's Appearance

An accuracy aspect which has not been touched upon thus far is the fact that the shape of the pupil equals a circle only when the subject looks straight into the camera (cf. Section 2.1.2). If it looks to the side, the pupil forms an ellipse whose major axis is vertical; if it looks up or down, the major axis of the pupil ellipse is horizontal; if it looks up or down as well as to the side, the major axis of the pupil ellipse has either a positive or negative angle with respect to the horizontal plane. The relations used to compute an intermediate position estimate, given in Eqs. (4.2) and (4.3), apply only to the case when the pupil is circular. However, since the vital requirement the complete eye-tracking system has to satisfy is that it be able to determine whether or not the subject is drifting (cf. Section 1.1), the error thus introduced constitutes no major problem. Moreover, correcting for this error would introduce a large amount of overhead computing into the algorithm, since the nature of the pupil's distortion depends on the gaze angle in a computationally expensive manner. Still, if OCTOPUS is to be viewed as interesting from a general point of view, this problem has to be addressed.

**Time Consumption**

By carefully examining the Sobel masks in Fig. 5.11, it becomes clear that by employing dynamic programming when moving along the horizontal and vertical detection lines, the number of multiplications and additions needed to compute an intermediate estimate can be reduced. Consider for instance the western mask and keep in mind that it is applied in a right-to-left manner. Evidently, during the first two steps along the detection line, all coefficients of the mask have to be applied to their respective pixels. However, at the third step, the sum to be computed by the three rightmost coefficients is the negation of the sum that was obtained by the leftmost coefficients during the first step. Analogously, at the fourth step, the sum to be computed by the three rightmost coefficients is the negation of the sum that was obtained by the leftmost coefficients during the second step. Thus, by maintaining a two-element FIFO queue and at each step along the detection line, from and including the third, first popping from the queue the sum computed by the leftmost coefficients two steps ago, then push onto the queue the sum computed by the same coefficients at the current location, and finally subtract the former from the latter to obtain the gradient at the current location, the number of multiplications and additions needed to obtain the gradient at a given location can be halved (not counting the extra subtraction). Analogue procedures can be applied for the eastern, northern and southern masks. For the diagonal masks, the entire gradient would still have to be computed at each step.

**Image Quality**

As pointed out above, the performance of OCTOPUS somewhat depends on the quality of the images it is supplied with. Some requirements the images ought to satisfy are:

- The eye should occupy as large a fraction of the image as possible without leaving out possible locations of the pupil.
- The pupil should constitute the largest dark region in the image and should be connected to no other dark regions in the image. With dark region is to be understood a member of the set of darkest regions in the image (i.e., those regions that constitute lakes by the introduced notation).
- No bright reflections protruding into the pupil. As described above, reflections are of a non-additive nature and thus cannot be removed.
- The contrast of the image should be as high as possible, in order to have a pupil lake with as few islands as possible and also to make the shores of the pupil lake as steep as possible and thus having the extent of the pupil lake correspond as closely as possible to the actual extent of the pupil. Having a gray scale of 256 gray levels ought to supply the desired contrast, given that the camera has the needed sensitivity and that the frame-grabber introduces a minimum of noise and errors into the digitized image.

## 5.5 OCTOPUS and the Complete System

In this section the modifications and additions that have to be made to OCTOPUS in order to have it work in cooperation with the complete eye-tracking system are discussed. Problems pertaining to non-algorithmic parts of the system, such as the camera and illumination setups mentioned in Section 1.3, are not discussed.

### 5.5.1 Major Line of Operation

The major line of operation for OCTOPUS when integrated in the complete eye-tracking system will be as follows:

1. OCTOPUS declares itself ready to start processing a new image by sending a ready-signal to the frame-grabber.

2. It waits until the frame-grabber signals that a new image is available for processing. At the reception of this signal, the present time is logged in order to be able to relate the returned gaze direction with the registered brain activity.
3. When a dedicated timer signals that the previous 20 ms period is ended, processing of the new image starts and the timer is reset to signal the end of the next 20 ms period.
4. As soon as a final position estimate has been arrived at, the position of the corresponding point of focus is sent to a given port address of the host PC. A signal is given that the position is available at the port. The relation between position estimate and point of focus in given in Section 5.5.2 below.
5. OCTOPUS returns to step 1 and the process is repeated.

Evidently, the above procedure assumes that OCTOPUS always will be able to arrive at a final position estimate in less than 20 ms. With the time measures presented in Section 5.4.2, this ought to be a valid assumption. However, if measures are taken to correct for errors due to distortions of the pupil's appearance whenever the subject does not look straight into the camera (cf. Section 5.4.3), the computational overhead introduced may invalidate this assumption.

### 5.5.2 Desired Output from OCTOPUS

As is clear from the preceding sections, the value returned by the current formulation of OCTOPUS is a coordinate pair corresponding to the estimated position of the pupil centre relative to the upper left corner of the image. However, this is not the coordinate pair that is of primary interest from an experimental point of view. Moreover, the actual point of interest is the point on the inducing monitor (cf. Section 1.2) at which the subject focuses, hereafter referred to as the *point of focus*. Consequently, the position of the pupil in an image has to be mapped to the point on the inducing monitor corresponding to it.

An admittedly memory consuming but time-efficient approach to implementing this mapping is to create for the $N \times N$ image a $N \times N$ array containing, for each possible (discrete) location of the pupil in the image, the corresponding point of focus on the inducing monitor. Accordingly, when OCTOPUS has arrived at a final position estimate for a given image, the $x$ and $y$ values of this estimate are used to index the array, and the coordinate pair thus obtained is made available to the outside world.

An array of this type would, assuming an image resolution of $512 \times 512$, require 1 MB of memory when each coordinate pair contained in it requires 32 byte. This would, however, constitute no major problem, considering today's low cost and availability of RAM expanders. The advantage of this approach is that, assuming an invariant experimental setup (cf. the next section), the values contained in it can be computed once and then stored in a file which is read anew each time OCTOPUS is invoked. Moreover, the inherit non-linearity between the pupil position in the image and the actual gaze angle can be compensated for without computational overhead by incorporating the needed correction in the values contained in the array.

### 5.5.3 Calibrating the Experimental Setup

Evidently, OCTOPUS will need some calibrating before being able to run satisfactorily within the framework of a complete eye-tracking system. Here, two calibration issues are addressed; parameter values and initializing the mapping array described above.

#### Parameter Values

As is clear from the preceding chapters, the position estimates returned by OCTOPUS depend heavily on the values assigned to the different parameters it employs. These are:

- $T_l$: The threshold value for the lake-thresholding technique described in Section 5.2.1. Accordingly, also the surface level of the lakes in an image are given by $T_l$. For the given test images it was found that $T_l = 18$ ensures "optimal" pupil lakes and sufficiently small non-pupil lakes. For a thorough discussion on the choice of a value for $T_l$, see Section 5.2.1.

- $T_s$: The fraction of the total number of sensors possessed by the octopus pixel filter that have to detect wetness for the octopus to report that it is swimming. With the given test images, $T_s = 0.8$ was found to be the best choice. A discussion on how to decide on a value for $T_s$ is found in Section 5.2.2.

- $T_e$: The threshold value used to determine whether or not the responses returned by the Sobel operators in Fig. 5.11 correspond to the pupil edge. The choice made for the given test images was $T_e = 0.5$. See Section 5.3.2 for a discussion of how to choose an appropriate value for $T_e$, and in particular p. 60 for a discussion of problems related to this choice.

- $r_{p_{min}}$: The fraction of the image resolution $N$ to which the minimum pupil radius corresponds. With the given test images, $r_{p_{min}}$ was estimated to be 10% of the images' size in the $x$ direction[10].

- $r_o$: The fraction of the image resolution $N$ to which the radius of the octopus pixel filter corresponds. In the current implementation, this fraction was set to $0.8r_{p_{min}}$. Two general rules for choosing a value for $r_o$ are (1) $r_o < r_{p_{min}}$ and (2) the better the image quality, the smaller $r_o$ can be chosen without running the risk of recognizing a non-pupil point as pupil point. Evidently, the number of operations for OCTOPUS decreases with lower values for $r_o$ (cf. Section 5.2.5).

- $r_d$: The radius of the "dead" region about the origin of operation inside which the probability of locating the pupil edge is zero. In the algorithm formulation, it is automatically set to $r_o/\sqrt{2}$ along the horizontal and vertical detection lines and to $r_o/2$ along the diagonal detection lines, whereas in the current implementation it is set to 0 along all detection lines. See Section 5.3.1, p. 55, for a discussion of problems related to defining a "dead" region about the origin of operation.

Clearly, the appropriateness of a given set of values for these parameters depends on the image quality and on the illumination conditions under which the images are made. Assuming that these are the same from one experiment to another, the same set of values can be used each time. Otherwise, a somewhat cumbersome process of determining an apparently optimal set of values has to be carried out. How cumbersome this process would be depends on its implementation[11].

### Initializing the Mapping Array

When employing the above array to map the location of the pupil in an image to the corresponding point of focus on the inducing monitor, it is of vital importance that the coordinate pairs contained in the array actually represent the correct points of focus in the given setup. If this is not the case, the estimates supplied by OCTOPUS are of no value whatsoever. Accordingly, either care has to be taken when calibrating the experimental setup or the values contained in the array have to be computed anew for each experiment.

An approach to initially compute the values to be contained in the array is to make five illuminated points appear on the inducing monitor, and as the subject focuses on each of the points, register the position estimates returned by OCTOPUS. Since the coordinates on the inducing monitor of the five points are known, the relations thus obtained can be used to compute the entire set of values for the array.

If it can be made certain that the position of the camera relative to the eye of the subject is exactly the same from one experiment to another, and that this also applies to the infrared mirror between the eye and the camera, the array thus obtained can be stored in a file. Next time an experiment is to be made, this file is read and the array initialized with the values contained in it. However, if this requirement cannot be satisfied, which is most probable to be the case, the values contained in the array have to be computed each time an experiment is made, and the above procedure will constitute a part of the calibration process of the experimental setup.

[10] Recall that the given test images are not square, i.e., they are of size $M \times N$, $M \neq N$.

[11] In the current implementation, the parameters are assigned values through typed constants which may be overridden by values defined in the initialization file `_search.rc`, which in turn may be overridden by command line parameters (cf. Section A.1.2).

### 5.5.4 Cooperation and Correlation

Evidently, the proper functioning of OCTOPUS in a complete eye-tracking system depends on its ability to work in cooperation and correlation with other integral parts of the system. In addition, as pointed out in the overall problem definition in Section 1.3, an interface between the eye-tracking system and the registering equipment of the setup has to be designed, in order to enable the simultaneous and correlated registration of brain activity and gaze direction.

#### Cooperating with the Frame-Grabber

From the above, it is evident that some sort of communication between OCTOPUS and the employed frame-grabber has to take place. The line of operation depicted indicates that this has to be a two-way communication, having OCTOPUS tell the frame-grabber that it is ready to start processing a new image as well as having the frame-grabber tell OCTOPUS that a new image is available for processing. In a strict sense, only the latter of these two is necessary, since the former can be implemented by having OCTOPUS ignore all signals that a new image is available until it is ready to start processing. Still, it is important that the frame-grabber signal that a new image is available, since otherwise OCTOPUS might start reading pixel values from memory locations still being written. Also, this signal is used to log the time when the gaze direction was recorded, and is thus of vital importance for the ability to relate it to the registered brain activity.

Another important issue is the location of the memory to which the digitized image is written. If OCTOPUS is to run in the CPU of the host PC, it would be preferable to have this be the RAM of the host PC. However, since most commercially available frame-grabbers have their own storage devices, this will hardly be possible. Moreover, it would require transferring the entire digitized image to the main memory of the host PC during the digitization process. If this transfer were to be carried out over the PC bus, its comparably low transfer rate would make little—if any—time available for processing the image. An alternative solution would be to use a dedicated parallel bus for this purpose, using DMA to access the main memory.

The easiest and most cost-efficient solution, though, would be the straightforward approach; to let the frame-grabber write the digitized images to its own storage device(s). This, however, imposes another problem on the setup. If the digitized image is to be stored on the frame-grabber board and the processing is to take place in the CPU of the host PC, the pixel values of the image have to be accessed over the PC bus. Since OCTOPUS operates by accessing individual pixels of the image, this would imply initiating and closing a dedicated transfer session for each pixel value needed, thus imposing an unproportionally high transfer overhead in addition to the question of the transfer rate of the PC bus at all being able to sustain this communication. Again, a possible solution would be to employ a dedicated bus for this purpose, but the transfer overhead involved suggests that another solution be sought.

The apparently most promising approach to the above problem would be to purchase a frame-grabber also possessing an on-board processor capable of running an implementation of OCTOPUS within the given time limits. Ideally, one processor should be dedicated to digitizing frames received from the camera and another processor should perform the image analysis. In addition, the purchased board should possess two memory banks to enable OCTOPUS to access one image while the next is being digitized. Accordingly, the digitized frames have to be written to the two banks in an interleaved manner.

Presently, a Transputer-based DSP board has kindly been placed at the group's disposal, possessing one T800 and one T200. Attempts are being made at having the T200 grab the frames, which at a later stage would make it possible to have the T800 perform the image analysis. However, the board will remain at our disposal only for a limited time period, and its cost ($\sim$ DM 18,000) suggests that another board would have to be sought for the final implementation. Still, the experiences obtained with it may prove valuable when a cheaper board has been chosen and is about to be programmed.

#### Correlation Issues

As pointed out in Section 1.1, it is of vital importance for the proper functioning of the complete eye-tracking system within the framework of the Synchronization and Cognition-project that it be

possible to relate registered brain activity to a specific gaze angle. In other words, OCTOPUS has to work in correlation with the registering equipment of the experimental setup. As indicated above, the interface between OCTOPUS and the general setup will consist of a port address at which OCTOPUS makes the returned estimate of the point of focus available. Also, each time a new frame has been digitized by the frame-grabber, it gives a signal indicating this. Since the time the frame is made available can be approximated to equal the time when the visual snapshot of the subject represented by the frame was taken, this signals triggers the current time to be logged. When OCTOPUS signals that an estimate of the point of focus is available at the given port address, these two values—the time of the snapshot and the estimated point of focus resulting from the snapshot—are coupled to form a tuple. This tuple can be made available to the program managing the output from the registering equipment, so that data concerning the registered brain activity at a given time can be compared to the estimated point of focus at that time.

# Chapter 6

# Conclusions

As pointed out in Section 1.3, the original aim of my thesis work was to develop a complete eye-tracking system, comprising the non-algorithmic as well as the algorithmic aspects of the overall problem definition. However, given the limited time available, this turned out not to be feasible. In fact, during the last six months, neither a frame-grabbing board nor a camera satisfying both the technical and the low-cost requirements has been found, and consequently it has become more or less clear that the DM 12,000 at the group's disposal for developing the eye-tracking system may not suffice.

## 6.1 Algorithmic Evaluation

In Section 5.4, the proposed eye-tracking algorithm OCTOPUS was given a relatively thorough evaluation. Evidently, the accuracy of the algorithm has to be improved to comply with the accuracy requirement given in the algorithmic problem definition if OCTOPUS is to be employed as the core of a complete eye-tracking system satisfying the requirements given in Section 1.2. The technique suggested in Section 5.4.3 ought to increase the accuracy, but since it has not been implemented and consequently not tested, nothing can be said about whether or not an implementation incorporating this technique would be sufficiently accurate.

As pointed out in Section 4.2.4, OCTOPUS assumes the pupil to form a circle in a given image. However, as stated in Section 2.1.2, this is only the case when the subject looks straight into the camera. Still, this assumption was chosen as a basis on which to formulate the algorithm. The incentive for doing this was that the primary point of interest as seen by the Eckhorn-Bauer group is to be able to determine whether or not the monkey is focusing on the reference point on the inducing monitor (cf. Section 1.2). Since the experimental setup is to be designed so that the subject focusing on the reference point corresponds to its looking straight into the camera, the assumption of having a circular pupil is sound. However, if OCTOPUS is to be of interest as a general-purpose eye-tracking algorithm, this problem has to be addressed.

The most promising aspect of OCTOPUS is doubtlessly its computational efficiency. The fact is that the current implementation of OCTOPUS running on a 40 MHz i386 PC needs approximately 2 ms to return an estimate of the pupil's location in the supplied image. Incorporating the suggested technique for increasing the accuracy of the returned estimates should have minimal influence on the time consumption. However, if measures are to be taken to correct for the error introduced by the pupil not forming a circle when the subject is drifting, the computational overhead introduced may, depending on their nature and implementation, cause the time needed for one estimate to surpass the limit of 20 ms. Apparently there is a trade-off between accuracy and speed.

## 6.2 Summary

All in all, I am very pleased that I got the opportunity of doing my thesis work here in Marburg. Some of the experiences I have obtained during my stay will doubtlessly be of value later. In particular I believe that my experiences during the first couple of months of my thesis work (cf.

Section 1.3) will come in handy, as they were of a very "this is the real world"-nature; having to phone one vendor after the other in order to obtain information material, going to fairs in Wiesbaden and Cologne, discussing with various "experts" how their solution would fit the needs of the group, going through immeasurable numbers of brochures, folders and technical notes to try to extract the essential information, etc. I also had to work with tasks not related to my professional field, such as soldering and debugging the frame-grabber used to digitize the test images, all this contributing to a varied and challenging sojourn in Germany.

On a more personal note, I consider it of immense value to have had the opportunity of living and working in a "foreign" country for almost a year (including my pre-thesis stay in Marburg during the summer of 1992). It is indisputably the best way to get to know the culture, atmosphere, people and language of a country to stay and live there for a relatively long period of time. In fact, a pleasant "side-effect" of my stay in Germany is that I now consider myself more or less fluent in German!

# Appendix A

# The Current Implementation

In this appendix, Section A.1 presents a couple of implementation issues demanding further elaboration and Section A.2 lists the source code of the current implementation.

## A.1 Implementation Details

As pointed out in Section 1.4, OCTOPUS has been implemented solely in Pascal, using Borland's Turbo-Pascal 6.0. Three issues demand further elaboration, these being the format of the images that are fed to the algorithm, presented in Section A.1.1; parameter initialization, discussed in Section A.1.2; and lastly, implementation of the plausibility areas presented in Section 5.2.4, discussed in Section A.1.3.

### A.1.1 Image Format

So far, the format of the images supplied to the current implementation of OCTOPUS has been given no attention. In the final implementation, they are to be accessed from the memory device(s) of the frame-grabber to be employed by the complete eye-tracking system. Temporarily, however, they have to be stored in files conforming to a specific format. The name of the file which is to be supplied to OCTOPUS is given as a command line parameter[1]. Obviously, the current implementation of OCTOPUS is only capable of handling one image at the time.

The format of the image file is very simple. The two first bytes of the file contain the horizontal resolution $M$ of the $M \times N$ image (cf. footnote 10 on p. 68), and the two next bytes contain the vertical resolution $N$ of the image. All consequent bytes are read as gray levels for pixels in the image. Note that the highest gray level in the current implementation is set to be 63. Note also that the image files are not compressed in any way.

### A.1.2 Parameter Initialization

As was indicated in Section 5.5.3, there are three different methods of assigning values to the different parameters employed by OCTOPUS. Firstly, they are assigned default values through typed constants. The default values can be overridden by defining new default values in an initialization file. The name of the initialization file is `_search.rc`. Lastly, the values assigned by either of these two methods are overridden by command line parameters. Here the focus will be on the format of the initialization file. See the comments in the source listing in Section A.2.2 for a description of valid command line parameters.

A sample initialization file is shown in Fig. A.1. As is seen, each line of the file has the form

$$\langle \textit{Variable name} \rangle \ \langle \textit{Assigned value} \rangle.$$

*Variable name* has to be the exact same name which is used in the source, and *Assigned value* is interpreted as a `real` number. Evidently, the variable names used in the implementation do not

[1] The file name has to be the last command line parameter supplied; cf. Section A.2.2.

```
% _search.rc - initialization file for OCTOPUS.

lakeLevel                   18    % T_{l}
sobelThreshold               5    % T_{e}
wetFraction                  0.8  % T_{s}

minPupilRadiusFraction      10    % r_{p_{min}}
octopusRadiusFraction        8    % r_{d}

nIterations                  1
```

Figure A.1: A sample initialization file.

correspond very well to the notation introduced and used in this paper. See the comments in the source listings in Section A.2 for the correspondence between source variable names and parameter names used in the text. Note that a random number of empty lines can occur in the initialization file and that end-of-line comments start with a `%`-character.

As mentioned, the name of the initialization file is `_search.rc`. The current implementation looks for a file with this name in the directory returned by the auxiliary function `homeDirectory`[2].

### A.1.3 Implementing the Plausibility Areas

The plausibility areas described in Section 5.2.4 are basically implemented using a three-element array indexing a second array containing the candidate pixels. Element $i$, $1 \leq i \leq 3$, of the three-element array contains an integer denoting the number $n_i$ of candidate pixels contained in plausibility area $i$. The second array contains all candidate pixels in $\mathbf{S}$. Recall that plausibility area 1 is a proper subset of plausibility area 2 and that plausibility area 2 is a proper subset of plausibility area 3, the latter corresponding to $\mathbf{S}$ itself[3]. The array is ordered so that the members of plausibility area 1 are contained in the $n_1$ first elements, the members of plausibility area 2 in the $n_2$ first elements and the members of plausibility area 3 in the $n_3$ first elements. Each element of the array contains the coordinate pair of a candidate pixel and a boolean flag *visited* telling whether or not the candidate pixel already has been used as starting pixel.

At the outset of the search procedure, the priority level $p$ is assigned value 1, which is used to index the three-element array. The value thus obtained is $n_1$, the number of elements in plausibility area 1, and the Turbo-Pascal function `random` is called with $n_1$ as parameter, returning a value between 0 and $n_1 - 1$. This value is used to index the second array, and since the $n_1$ first elements of the array are the members of plausibility area 1, the point thus obtained belongs to plausibility area 1. If the point is already marked visited, a new random number is generated, and this process is repeated until a point is found which has not been visited. It is then marked visited and used as starting pixel. A counter is incremented each time a starting pixel has been selected. If this counter reaches $n_1$, the priority level is incremented, thus enabling points belonging to plausibility area 2 to be selected. Note that when the priority level is incremented, all candidate pixels belonging to plausibility area 1 have been marked visited, so the only members of plausibility area 2 that can be selected are those not belonging to plausibility area 1. This process is repeated until a pupil point is found or there are no more unvisited elements in the array, which corresponds to every candidate pixel of $\mathbf{S}$ having been used as starting pixel without success. In the latter case, the pupil could not be found in the image, and the value `false` is returned. The procedure is illustrated in Fig. A.2.

[2] `homeDirectory` is currently implemented in `auxil.pas` (not listed) and the value returned is `/user/blehr/`.
[3] Given that the number of plausibility areas is 3, cf. Section 5.2.4.

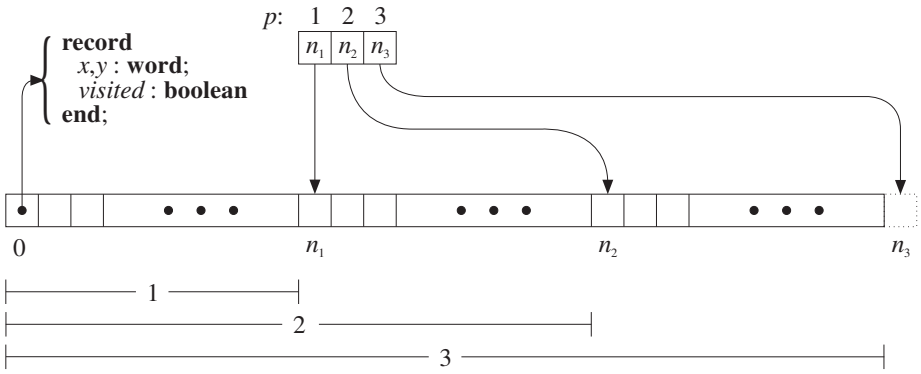



Figure A.2: The principle behind the implementation of the plausibility areas described in Section 5.2.4. $p$ denotes the priority level and $n_i$, $1 \leq i \leq 3$, the number of candidate pixels contained in plausibility area $i$.

## A.2 Sources

Two source files are listed in this section. The first is the Turbo-Pascal unit `search.pas`, implementing the OCTOPUS eye-tracking algorithm, and the second is `eyepos.pas`, which is an example of how to use `search.pas`. Note that the variable names used in the implementations do not correspond particularly well with the notation used throughout his paper.

### A.2.1 `search.pas`

This Turbo-Pascal unit implements the OCTOPUS eye-tracking algorithm. The two main procedures are `findPupilPoint` and `findPosition`, implementing Steps 1 and 2, respectively, of the basic algorithm formulation. See in-line comments for additional information.

```
(***
 ***  search.pas, version 2.3
 ***
 ***  Turbo-Pascal unit implementing the OCTOPUS eye-tracking algorithm.
 ***  The main procedures are:
 ***
 ***     findPupilPoint
 ***     findPosition,
 ***
 ***  implementing Steps 1 and 2, respectively, of the basic algorithm
 ***  formulation.
 ***)

unit search;


INTERFACE

(*
 *  auxil is a unit containing auxiliary procedures and functions.
 *)

uses graph,auxil;

(**
```

```
 **  Global constants
 **)

const

(*
 *  maxSize corresponds to the maximum resolution N of the NxN image.
 *)
      maxSize                   = 512;

(*
 *  Number of gray levels in the images supplied.
 *)
      nGrayLevels               = 64;

(*
 *  Maximum number of jumps the octopus can make.
 *)
      nLeapsMax                 = 10;

(*
 *  Maximum number of intermediate position estimates that can be made.
 *)
      nEstimatesMax             = 55;

(*
 *  Defining directions.  null means no direction (i.e., stationary).
 *)
      null                      = 0;
      west                      = -1;
      east                      = 1;
      north                     = -1;
      south                     = 1;

(*
 *  Constants needed by applications calling markPosition.
 *)
      asPoint                   = 0;
      asStar                    = 1;
      asCircle                  = 2;

(**
 **  Global type definitions.
 **)

type

(*
 *  Array of byte, used by blockRead and blockWrite.
 *)
      bufferType                = array[1..maxSize] of byte;

(*
 *  Two-dimensional dynamic array, used to contain the image being
 *  processed.
```

```
 *)
      pixelLinePtr              = ^pixelLine;
      pixelLine                 = array[1..maxSize] of byte;
      pixelMatrixPtr            = ^pixelMatrix;
      pixelMatrix               = array[1..maxSize] of pixelLinePtr;

(*
 *  Array used to map between actaul gray levels and gray levels of the
 *  lake-thresholded image.
 *)
      mapType                   = array[0..nGrayLevels-1] of byte;

(*
 *  Boolean array mapping between gray levels and wet/not wet.
 *)
      wetType                   = array[0..nGrayLevels-1] of boolean;

(*
 *  Type of the candidate pixels in S.  visited=true corresponds to the
 *  pixel's having been removed from S.
 *)
      point                     = record
        x                         : word;
        y                         : word;
        visited                   : boolean
      end;

(*
 *  Types employed to implement the set S of candidate pixels.
 *)
      pointArrayPtr             = ^pointArray;
      pointArray                = array[0..maxSize-1] of point;

(*
 *  Array indexed by the current priority level and returning a pointer
 *  into S corresponding to the number of candidate pixels contained in
 *  the plausibility area corresponding to the priority level.
 *)
      priAreasType              = array[1..3] of word;

(**
 **  Typed constants (initialized variables).  They are overridden by:
 **  (1) initializations made in the init-file _searc.rc and (2) command
 **  line parameters.  (2) overrides (1).
 **)

const

(*
 *  T_{l} [-l option]
 *)
      lakeLevel                 : byte = 1;

(*
 *  T_{e} [-s option]
```

```
 *)
      sobelThreshold            : byte = 5;

(*
 *  r_{p_{min}}, per cent.
 *)
      minPupilRadiusFraction    : real = 10;

(*
 *  r_{o} [-o option], per cent.
 *)
      octopusRadiusFraction     : real = 8;

(*
 *  r_{s} [-w option]
 *)
      wetFraction               : real = 0.8;

(*
 *  Used by eyepos.pas; included here so that it can be initialized in
 *  _search.rc.  Denotes how many times an overall estimate is to be
 *  found and is used to obtain performance measures.  [-n option]
 *)
      nIterations               : word = 1;

(*
 *  Used by the application program; denotes where on the physical
 *  to display the image.
 *)
      startX                    : word = 0;
      startY                    : word = 0;

(*
 *  If true, each point at which the Sobel operators are applied are
 *  marked with a white pixel in the image.  [-d option]
 *)
      showSearch                : boolean = false;

(**
 **  Global variables.
 **)

var

(*
 *  The two-dimensional array containing the image.
 *)
      image                     : pixelMatrixPtr;

(*
 *  Horizontal and vertical size of the image.
 *)
      xSize                     : word;
      ySize                     : word;
```

```
(*
 *  Instances of mapType and wetType explained above.
 *)
      map                       : mapType;
      wet                       : wetType;

(*
 *  r_{p_{min}} in pixels.
 *)
      minPupilRadius            : word;

(*
 *  r_{o} in pixels
 *)
      octopusRadius             : word;

(*
 *  r_{d}; radius of the "dead" region about the origin of opration,
 *  in pixels.
 *)
      nonSearchRadius           : word;

(*
 *  r_{d}/sqrt(2)
 *)
      diagonalNonSearchRadius   : word;

(*
 *  Number of the octopus' sensors that have to report wetness for the
 *  octopus to be swimming.  Determined by T_{s}.
 *)
      inSeaThreshold            : word;

(*
 *  The set S of candidate pixels.
 *)
      startingPoints            : pointArrayPtr;

(*
 *  Instance of priAreasType explained above.
 *)
      priAreas                  : priAreasType;


(**
 **  Exported procedures and functions.
 **)


(*
 *  findPupilPoint
 *
 *  Implements the swimming octopus search algorithm.  Returns true if
 *  a pupil point is found, and in that case pupilPoint contains the
 *  coordinates of the point.
```

```
 *)

function findPupilPoint(var pupilPoint : pointType) : boolean;


(*
 *  findPosition
 *
 *  Implements Step 2 of the basic algorithm formulation.  Returns true
 *  if the maximum number of iterations allowed (determined by
 *  nEstimatesMax) sufficed to arrive at a final overall estimate.
 *  pupilPoint contains the initial origin of operation and is typically
 *  the point returned by findPupilPoint.  position contains the overall
 *  estimate arrived at, also when the returned value is false.
 *)

function findPosition(pupilPoint : pointType;
                  var position   : pointType) : boolean;


(*
 *  applySobelMask
 *
 *  Applies to location (x,y) in the image the Sobel mask determined by
 *  northOrSouth and eastOrWest.  E.g., northOrSouth=north and
 *  eastOrWest=east means the northeast mask, and northOrSouth=south
 *  and eastOrWest=null means the south mask.
 *)

function applySobelMask(x,y                       : word;
                        northOrSouth,eastOrWest : shortInt) : integer;


(*
 *  initGlobals
 *
 *  Initializes global variables and reads _search.rc.  Among the
 *  variables that are initialized is the set S of candidate pixels.
 *)

procedure initGlobals;


(*
 *  markPosition
 *
 *  Marks the location (x,y) in the image as a point, circle or star,
 *  depending on the value of asWhat (see the constant declarations
 *  above).
 *)

procedure markPosition(x,y : word; asWhat : byte);


(*
```

```
 *  closeSearch
 *
 *  Cleans up.
 *)

procedure closeSearch;


IMPLEMENTATION

uses crt,math;

const whiteSpace                = [#9,#32];

type  variableRPtr              = ^variableR;
      variableR                 = record
        name                      : string;
        value                     : real
      end;


(*
 *  readInitFile
 *
 *  Internal procedure called by initGlobals which reads the init-file
 *  _search.rc.
 *)

procedure readInitFile;

var   initFile                  : text;
      currentLine               : string;
      variable                  : variableRPtr;


  function nextVariable : variableRPtr;

  var   variableName              : string;
        variableValue             : real;


    function nextWord : string;

    var   theWord                   : string;


      function nextLine : string;

      var   aLine                     : string;

      begin
        if (not EOF(initFile)) then
          readLn(initFile,aLine)
        else
          aLine:='EOF';
```

```
    nextLine:=aLine
  end;


  procedure removeWhiteSpace;

  begin
    while ((currentLine<>'') and (currentLine[1] in whiteSpace)) do
      delete(currentLine,1,1)
  end;


begin
  theWord:='';
  removeWhiteSpace;
  while ((currentLine='') or (currentLine[1]='%')) do
    begin
      currentLine:=nextLine;
      removeWhiteSpace
    end;
  if (currentLine<>'EOF') then
    begin
      while ((currentLine<>'') and (not (currentLine[1] in whiteSpace))) do
        begin
          theWord:=theWord+currentLine[1];
          delete(currentLine,1,1)
        end
    end;
  nextWord:=theWord
end;


function getValue : real;

var   valueAsString             : string;
      value                     : real;
      result                    : integer;

begin
  valueAsString:=nextWord;
  val(valueAsString,value,result);
  if (result=0) then
    getValue:=value
  else
    begin
      variableName:=valueAsString;
      getValue:=maxInt
    end
end;


function makeVariable(name : string; value : real) : variableRPtr;

var   theVariable               : variableRPtr;
```

```
    begin
      new(theVariable);
      theVariable^.name:=name;
      theVariable^.value:=value;
      makeVariable:=theVariable
    end;


  begin
    variableName:=nextWord;
    variableValue:=maxInt;
    while ((variableName<>'') and (variableValue=maxInt)) do
      variableValue:=getValue;
    if (variableName='') then
      nextVariable:=nil
    else
      nextVariable:=makeVariable(variableName,variableValue)
  end;


begin
  if fileExists(homePath('_search.rc')) then
    begin
      assign(initFile,homePath('_search.rc'));
      reset(initFile);
      readLn(initFile,currentLine);
      variable:=nextVariable;
      while (variable<>nil) do
        begin
          if (variable^.name='lakeLevel') then
            lakeLevel:=trunc(variable^.value)
          else if (variable^.name='sobelThreshold') then
            sobelThreshold:=trunc(variable^.value)
          else if (variable^.name='minPupilRadiusFraction') then
            minPupilRadiusFraction:=variable^.value
          else if (variable^.name='octopusRadiusFraction') then
            octopusRadiusFraction:=variable^.value
          else if (variable^.name='wetFraction') then
            wetFraction:=trunc(variable^.value)
          else if (variable^.name='nIterations') then
            nIterations:=trunc(variable^.value);
          variable:=nextVariable
        end;
      close(initFile)
    end
end;


(**
 **  Implementation of the exported proedures and functions.
 **)


function findPupilPoint(var pupilPoint : pointType) : boolean;
```

```
var   pupilFound               : boolean;
      priLevel                 : byte;
      nInvestigated            : word;
      startingPoint            : pointType;
      seaPointer               : pointType;
      previousPos              : pointType;
      lastPos                  : pointType;
      i                        : word;


  function octopusSwims(pos : pointType; depth : byte) : boolean;


    procedure locateSea(pos : pointType; var seaPointer : pointType);


      function feelNArm(offset : word) : word;

      begin
        while ((not wet[image^[pos.y-offset]^[pos.x]]) and (offset>0)) do
          offset:=offset-1;
        feelNArm:=offset
      end;


      function feelSArm(offset : word) : word;

      begin
        while ((not wet[image^[pos.y+offset]^[pos.x]]) and (offset>0)) do
          offset:=offset-1;
        feelSArm:=offset
      end;


      function feelEArm(offset : word) : word;

      begin
        while ((not wet[image^[pos.y]^[pos.x+offset]]) and (offset>0)) do
          offset:=offset-1;
        feelEArm:=offset
      end;


      function feelWArm(offset : word) : word;

      begin
        while ((not wet[image^[pos.y]^[pos.x-offset]]) and (offset>0)) do
          offset:=offset-1;
        feelWArm:=offset
      end;


    begin
      seaPointer.x:=feelEArm(octopusRadius)-feelWArm(octopusRadius);
      seaPointer.y:=feelSArm(octopusRadius)-feelNArm(octopusRadius)
```

```
    end;


    function octopusInSea(pos : pointType) : boolean;

    var   pixelsInSea               : word;
          probablyInSea             : boolean;
          i                         : integer;

    begin
      probablyInSea:=(wet[image^[pos.y]^[pos.x-octopusRadius]] and
                      wet[image^[pos.y-octopusRadius]^[pos.x]]);
      if probablyInSea then
        begin
          pixelsInSea:=0;
          for i:=(1-octopusRadius) to (octopusRadius-1) do
            begin
              if wet[image^[pos.y]^[pos.x+i]] then
                pixelsInSea:=pixelsInSea+1;
              if wet[image^[pos.y+i]^[pos.x]] then
                pixelsInSea:=pixelsInSea+1
            end;
          if (pixelsInSea>=inSeaThreshold) then
            begin
              pupilPoint:=pos;
              octopusInSea:=true
            end
          else
            octopusInSea:=false
        end
      else
        octopusInSea:=false
    end;


  begin
    previousPos:=lastPos;
    lastPos:=pos;
    locateSea(pos,seaPointer);
    if ((seaPointer.x=0) and (seaPointer.y=0)) then
      octopusSwims:=octopusInSea(pos)
    else
      begin
        pos.x:=pos.x+seaPointer.x;
        pos.y:=pos.y+seaPointer.y;
        if (((pos.x=previousPos.x) and (pos.y=previousPos.y)) or
            (depth>=nLeapsMax)) then
          octopusSwims:=false
        else
          octopusSwims:=octopusSwims(pos,depth+1)
      end
  end;


begin
```

```
  priLevel:=1;
  nInvestigated:=0;
  lastPos.x:=maxInt;
  lastPos.y:=maxInt;
  repeat
    repeat
      repeat
        i:=random(priAreas[priLevel])
      until (not startingPoints^[i].visited);
      startingPoint.x:=startingPoints^[i].x;
      startingPoint.y:=startingPoints^[i].y;
      startingPoints^[i].visited:=true;
      pupilFound:=octopusSwims(startingPoint,0);
      nInvestigated:=nInvestigated+1
    until (pupilFound or (nInvestigated=priAreas[priLevel]));
    priLevel:=priLevel+1
  until (pupilFound or (priLevel=4));
  for i:=0 to (priAreas[priLevel-1]-1) do
    startingPoints^[i].visited:=false;
  findPupilPoint:=pupilFound
end;


function findPosition(pupilPoint : pointType;
                  var position   : pointType) : boolean;

var   nEstimates                : word;
      previousEstimate          : pointType;
      break                     : boolean;


(*
 *  Many of the subprocedures of iterateOnce and iterateTwice could
 *  have been combined to form fewer procedures.  However, for the
 *  sake of computational efficiency, they have been implemented
 *  separately in order to minimize the number of operations needed.
 *)

  procedure iterateOnce(origin   : pointType;
                    var estimate : pointType);

  var   NSCentre                  : word;
        WECentre                  : word;


    procedure NS_getMiddlePoint(x,y : word);

    var   NPoint                    : word;
          SPoint                    : word;


      procedure N_searchEdge(x,y : word);

      begin
        y:=y-nonSearchRadius;
```

```
    while wet[image^[y-2]^[x]] do
      y:=y-1;
    repeat
      y:=y-1;
      if showSearch then
        markPosition(x,y,asPoint)
    until (applySobelMask(x,y,north,null)>=sobelThreshold);
    NPoint:=y
  end;


  procedure S_searchEdge(x,y : word);

  begin
    y:=y+nonSearchRadius;
    while wet[image^[y+2]^[x]] do
      y:=y+1;
    repeat
      y:=y+1;
      if showSearch then
        markPosition(x,y,asPoint)
    until (applySobelMask(x,y,south,null)>=sobelThreshold);
    SPoint:=y
  end;


begin
  N_searchEdge(x,y);
  S_searchEdge(x,y);
  NSCentre:=NPoint+((SPoint-NPoint) div 2)
end;


procedure WE_getMiddlePoint(x,y : word);

var   WPoint                    : word;
      EPoint                    : word;


  procedure W_searchEdge(x,y : word);

  begin
    x:=x-nonSearchRadius;
    while wet[image^[y]^[x-2]] do
      x:=x-1;
    repeat
      x:=x-1;
      if showSearch then
        markPosition(x,y,asPoint)
    until (applySobelMask(x,y,null,west)>=sobelThreshold);
    WPoint:=x
  end;


  procedure E_searchEdge(x,y : word);
```

```
    begin
      x:=x+nonSearchRadius;
      while wet[image^[y]^[x+2]] do
        x:=x+1;
      repeat
        x:=x+1;
        if showSearch then
          markPosition(x,y,asPoint)
      until (applySobelMask(x,y,null,east)>=sobelThreshold);
      EPoint:=x
    end;


  begin
    W_searchEdge(x,y);
    E_searchEdge(x,y);
    WECentre:=WPoint+((EPoint-WPoint) div 2);
  end;


begin
  with origin do
    begin
      NS_getMiddlePoint(x,y);
      WE_getMiddlePoint(x,y);
      estimate.x:=(nEstimates*estimate.x+WECentre) div (nEstimates+1);
      estimate.y:=(nEstimates*estimate.y+NSCentre) div (nEstimates+1);
      nEstimates:=nEstimates+1
    end
end;


procedure iterateTwice(origin   : pointType;
                   var estimate : pointType);

var   NWSECentre               : pointType;
      SWNECentre               : pointType;


  procedure NWSE_getMiddlePoint(x,y : word);

  var   NWPoint                  : pointType;
        SEPoint                  : pointType;


    procedure NW_searchEdge(x,y : word);

    begin
      x:=x-diagonalNonSearchRadius;
      y:=y-diagonalNonSearchRadius;
      while wet[image^[y-2]^[x-2]] do
        begin
          x:=x-1;
          y:=y-1
```

```
      end;
    repeat
      x:=x-1;
      y:=y-1;
      if showSearch then
        markPosition(x,y,asPoint)
    until (applySobelMask(x,y,north,west)>=sobelThreshold);
    NWPoint.x:=x;
    NWPoint.y:=y
  end;


  procedure SE_searchEdge(x,y : word);

  begin
    x:=x+diagonalNonSearchRadius;
    y:=y+diagonalNonSearchRadius;
    while wet[image^[y+2]^[x+2]] do
      begin
        x:=x+1;
        y:=y+1
      end;
    repeat
      x:=x+1;
      y:=y+1;
      if showSearch then
        markPosition(x,y,asPoint)
    until (applySobelMask(x,y,south,east)>=sobelThreshold);
    SEPoint.x:=x;
    SEPoint.y:=y
  end;


begin
  NW_searchEdge(x,y);
  SE_searchEdge(x,y);
  NWSECentre.x:=NWPoint.x+((SEPoint.x-NWPoint.x) div 2);
  NWSECentre.y:=NWPoint.y+((SEPoint.y-NWPoint.y) div 2)
end;


procedure SWNE_getMiddlePoint(x,y : word);

var   SWPoint                   : pointType;
      NEPoint                   : pointType;


  procedure SW_searchEdge(x,y : word);

  begin
    x:=x-diagonalNonSearchRadius;
    y:=y+diagonalNonSearchRadius;
    while wet[image^[y+2]^[x-2]] do
      begin
        x:=x-1;
```

```
          y:=y+1
        end;
      repeat
        x:=x-1;
        y:=y+1;
        if showSearch then
          markPosition(x,y,asPoint)
      until (applySobelMask(x,y,south,west)>=sobelThreshold);
      SWPoint.x:=x;
      SWPoint.y:=y
    end;


    procedure NE_searchEdge(x,y : word);

    begin
      x:=x+diagonalNonSearchRadius;
      y:=y-diagonalNonSearchRadius;
      while wet[image^[y-2]^[x+2]] do
        begin
          x:=x+1;
          y:=y-1
        end;
      repeat
        x:=x+1;
        y:=y-1;
        if showSearch then
          markPosition(x,y,asPoint)
      until (applySobelMask(x,y,north,east)>=sobelThreshold);
      NEPoint.x:=x;
      NEPoint.y:=y
    end;


  begin
    SW_searchEdge(x,y);
    NE_searchEdge(x,y);
    SWNECentre.x:=SWPoint.x+((NEPoint.x-SWPoint.x) div 2);
    SWNECentre.y:=SWPoint.y+((NEPoint.y-SWPoint.y) div 2)
  end;


begin
  with origin do
    begin
      NWSE_getMiddlePoint(x,y);
      SWNE_getMiddlePoint(x,y);
      estimate.x:=(nEstimates*estimate.x+
                   (NWSECentre.x+SWNECentre.x-x)) div (nEstimates+1);
      estimate.y:=(nEstimates*estimate.y+
                   (NWSECentre.y+SWNECentre.y-y)) div (nEstimates+1);
      nEstimates:=nEstimates+1
    end
end;
```

```
begin
  nEstimates:=0;
  position.x:=0;
  position.y:=0;
  repeat
    previousEstimate:=position;
    iterateOnce(pupilPoint,position);
    break:=((nEstimates>nEstimatesMax) or
            ((position.x=previousEstimate.x) and
             (position.y=previousEstimate.y)));
    if (not break) then
      begin
        iterateTwice(pupilPoint,position);
        break:=((nEstimates>nEstimatesMax) or
                ((position.x=previousEstimate.x) and
                 (position.y=previousEstimate.y)));
        pupilPoint:=position
      end
  until break;
  findPosition:=(nEstimates<=nEstimatesMax)
end;


function applySobelMask(x,y                      : word;
                        northOrSouth,eastOrWest : shortInt) : integer;

begin
  case northOrSouth of
    null :
      case eastOrWest of
        null : applySobelMask:=0;
        west : applySobelMask:=   map[image^[y-2]^[x-2]]-
                                  map[image^[y-2]^[x+2]]+
                                2*map[image^[y  ]^[x-2]]-
                                2*map[image^[y  ]^[x+2]]+
                                  map[image^[y+2]^[x-2]]-
                                  map[image^[y+2]^[x+2]];
        east : applySobelMask:=  -map[image^[y-2]^[x-2]]+
                                  map[image^[y-2]^[x+2]]-
                                2*map[image^[y  ]^[x-2]]+
                                2*map[image^[y  ]^[x+2]]-
                                  map[image^[y+2]^[x-2]]+
                                  map[image^[y+2]^[x+2]]
      end;
    north :
      case eastOrWest of
        null : applySobelMask:=   map[image^[y-2]^[x-2]]+
                                2*map[image^[y-2]^[x  ]]+
                                  map[image^[y-2]^[x+2]]-
                                  map[image^[y+2]^[x-2]]-
                                2*map[image^[y+2]^[x  ]]-
                                  map[image^[y+2]^[x+2]];
        west : applySobelMask:= 2*map[image^[y-2]^[x-2]]+
                                  map[image^[y-2]^[x  ]]+
```

```
                                  map[image^[y  ]^[x-2]]-
                                  map[image^[y  ]^[x+2]]-
                                  map[image^[y+2]^[x  ]]-
                                2*map[image^[y+2]^[x+2]];
        east : applySobelMask:=   map[image^[y-2]^[x  ]]+
                                2*map[image^[y-2]^[x+2]]-
                                  map[image^[y  ]^[x-2]]+
                                  map[image^[y  ]^[x+2]]-
                                2*map[image^[y+2]^[x-2]]-
                                  map[image^[y+2]^[x  ]]
      end;
    south :
      case eastOrWest of
        null : applySobelMask:=  -map[image^[y-2]^[x-2]]-
                                2*map[image^[y-2]^[x  ]]-
                                  map[image^[y-2]^[x+2]]+
                                  map[image^[y+2]^[x-2]]+
                                2*map[image^[y+2]^[x  ]]+
                                  map[image^[y+2]^[x+2]];
        west : applySobelMask:=  -map[image^[y-2]^[x  ]]-
                                2*map[image^[y-2]^[x+2]]+
                                  map[image^[y  ]^[x-2]]-
                                  map[image^[y  ]^[x+2]]+
                                2*map[image^[y+2]^[x-2]]+
                                  map[image^[y+2]^[x  ]];
        east : applySobelMask:=-2*map[image^[y-2]^[x-2]]-
                                  map[image^[y-2]^[x  ]]-
                                  map[image^[y  ]^[x-2]]+
                                  map[image^[y  ]^[x+2]]+
                                  map[image^[y+2]^[x  ]]+
                                2*map[image^[y+2]^[x+2]]
      end
  end
end;


procedure initGlobals;

var   i                         : byte;


(*
 *  initPointMatrix
 *
 *  Initializes the array implementing the set S of candidate pixels
 *  so that the candidate pixels belonging to plausibility area 1
 *  occupy the first elements and the pixels belonging to plausibility
 *  area 2 occupy the following elements.  The last portion of the
 *  array contains the candidate pixels belonging exclusively to
 *  plausibility area 3.
 *)

  procedure initPointMatrix;

  var   xCentre                   : word;
```

```
    yCentre                  : word;
    xRadius                  : word;
    yRadius                  : word;
    nHorizontal              : word;
    nVertical                : word;
    xStart                   : word;
    yStart                   : word;
    leftOffset               : word;
    priArea1xRadius          : word;
    priArea1yRadius          : word;
    priArea2xRadius          : word;
    priArea2yRadius          : word;
    priCount                 : priAreasType;
    pri                      : byte;
    priOffset                : word;
    currentX                 : word;
    currentY                 : word;
    i,j                      : word;


procedure initVariables;

var   i                        : byte;


  function nPointsInRectangle(x,y : word) : word;

  begin
    if even(trunc(x/2)+trunc(y/2)) then
      nPointsInRectangle:=high(0.5*x*y)
    else
      nPointsInRectangle:=trunc(0.5*x*y)
  end;


begin
  xCentre:=xSize div 2;
  yCentre:=ySize div 2;
  xRadius:=trunc((xCentre-minPupilRadius)/minPupilRadius);
  yRadius:=trunc((yCentre-minPupilRadius)/minPupilRadius);
  nHorizontal:=2*xRadius+1;
  nVertical:=2*yRadius+1;
  xStart:=xCentre-xRadius*minPupilRadius;
  yStart:=yCentre-yRadius*minPupilRadius;
  currentX:=xStart;
  currentY:=yStart;
  if ((even(xRadius) and even(yRadius)) or
      (odd(xRadius)  and odd(yRadius) )) then
    leftOffset:=0
  else
    leftOffset:=minPupilRadius;
  priArea1xRadius:=xRadius div 3;
  priArea1yRadius:=yRadius div 3;
  priArea2xRadius:=trunc((2/3)*xRadius);
  priArea2yRadius:=trunc((2/3)*yRadius);
```

```
    priAreas[1]:=nPointsInRectangle(2*priArea1xRadius+1,2*priArea1yRadius+1);
    priAreas[2]:=nPointsInRectangle(2*priArea2xRadius+1,2*priArea2yRadius+1);
    priAreas[3]:=nPointsInRectangle(nHorizontal,nVertical);
    for i:=1 to 3 do
      priCount[i]:=0;
    getMem(startingPoints,priAreas[3]*sizeOf(point));
  end;


  function getPriority(x,y : word) : byte;


    function inRectangle(x,y,rX,rY : word) : boolean;

    begin
      inRectangle:=((x>=xCentre-rX*minPupilRadius) and
                    (x<=xCentre+rX*minPupilRadius) and
                    (y>=yCentre-rY*minPupilRadius) and
                    (y<=yCentre+rY*minPupilRadius))
    end;


  begin
    if inRectangle(x,y,priArea1xRadius,priArea1yRadius) then
      getPriority:=1
    else if inRectangle(x,y,priArea2xRadius,priArea2yRadius) then
      getPriority:=2
    else
      getPriority:=3
  end;


  function getPriOffset(i : byte) : word;

  begin
    case (i=1) of
      true  : getPriOffset:=0;
      false : getPriOffset:=priAreas[i-1]
    end
  end;


  procedure toggleLeftOffset;

  begin
    case (leftOffset=minPupilRadius) of
      true  : leftOffset:=0;
      false : leftOffset:=minPupilRadius
    end
  end;


begin
  initVariables;
  repeat
```

```
      pri:=getPriority(currentX+leftOffset,currentY);
      priOffset:=getPriOffset(pri);
      startingPoints^[priOffset+priCount[pri]].x:=currentX+leftOffset;
      startingPoints^[priOffset+priCount[pri]].y:=currentY;
      startingPoints^[priOffset+priCount[pri]].visited:=false;
      inc(priCount[pri]);
      inc(currentX,2*minPupilRadius);
      if (currentX+leftOffset>xCentre+xRadius*minPupilRadius) then
        begin
          inc(currentY,minPupilRadius);
          currentX:=xStart;
          toggleLeftOffset
        end
    until (currentY>yStart+2*yRadius*minPupilRadius)
  end;


begin
  randomize;
  minPupilRadius:=round(xSize*minPupilRadiusFraction/100);
  octopusRadius:=round(xSize*octopusRadiusFraction/100);

(*
 *  These have been commented out for reasons elaborated upon in
 *  Section 5.3.1:
 *
 *     nonSearchRadius:=trunc(octopusRadius/sqrt(2));
 *     diagonalNonSearchRadius:=trunc(nonSearchRadius/sqrt(2));
 *
 *  The following two assigments replace them:
 *)

  nonSearchRadius:=0;
  diagonalNonSearchRadius:=0;

  inSeaThreshold:=round(4*wetFraction*pred(octopusRadius))+2;
  initPointMatrix;
  for i:=0 to lakeLevel do
    begin
      map[i]:=lakeLevel;
      wet[i]:=true
    end;
  for i:=(lakeLevel+1) to (nGrayLevels-1) do
    begin
      map[i]:=i;
      wet[i]:=false
    end
end;


procedure markPosition(x,y : word; asWhat : byte);

begin
  setColor(63);
  case asWhat of
```

```
    asPoint :
      putPixel(startX+x,startY+y,63);
    asStar :
      begin
        line(startX+x-3,startY+y,startX+x+3,startY+y);
        line(startX+x,startY+y-3,startX+x,startY+y+3)
      end;
    asCircle :
      circle(startX+x,startY+y,2)
  end
end;


procedure closeSearch;

var   i                          : word;

begin
  freeMem(startingPoints,priAreas[3]*sizeOf(point));
  for i:=1 to ySize do
    freeMem(image^[i],xSize);
  freeMem(image,4*ySize)
end;


begin
  readInitFile
end.
```

### A.2.2 eyepos.pas

This file contains an example of how to use `search.pas` to locate the pupil in an image. It also illustrates the principle used to obtain the time measures presented in Section 5.4.2. Lastly, the currently available command line options are shown.

```
(***
 ***  eyepos.pas, version 2.3
 ***
 ***  Example of how to employ search.pas, version 2.2 or later.
 ***  The image is supplied as command line parameter, is displayed
 ***  on the screen, and the pupil position found is marked with a
 ***  small star in the image.
 ***
 ***  Usage:
 ***
 ***     eyepos [-d][-l<T_{l}>][-o<r_{o}>][-s<T_{e}>]
 ***            [-w<T_{s}>][-n<nIterations>] file[.pic]
 ***
 ***  The -d option causes showSearch to be true and consequently each
 ***  location in the image to which the Sobel operators are applied
 ***  will be marked with a white point on the screen.
 ***)

program eyePos;

(**
 **  SVGA.pas contains routines for displaying high-resolution black &
 **  white images on the screen.  Requires an ET4000 SVGA graphic card.
 **)

uses crt,dos,graph,auxil,search,SVGA;

(**
 **  Variables
 **)

var

(*
 *  Variables used to measure the time.
 *)
      startHours               : word;
      startMinutes             : word;
      startSeconds             : word;
      start100thSeconds        : word;
      endHours                 : word;
      endMinutes               : word;
      endSeconds               : word;
      end100thSeconds          : word;

(*
 *  A pupil point, supplied by findPupilPoint.
 *)
      pupilPoint               : pointType;
```

```
(*
 *  The position of the pupil in the given image.
 *)
      pupilPosition             : pointType;

(*
 *  Estimate of the time needed to locate the pupil.  Accuracy
 *  increases when nIterations increases.
 *)
      searchTime                : real;


procedure init;

var   fileName                  : string;
      inputFile                 : file;
      buffer                    : bufferType;
      i,j                       : word;


(*
 *  getParameters
 *
 *  Reads and interprets the command line parameters.
 *)

  procedure getParameters(numberOfParameters : word);

  var   i                         : byte;
        thisParameter             : string;


    function getParameterValue(parameterString : string) : real;

    var   value                     : real;
          result                    : integer;

    begin
      val(copy(parameterString,3,length(parameterString)-2),value,result);
      if (result=0) then
        getParameterValue:=value
      else
        fatalError('Illegal parameter value, must be a number!')
    end;


  begin
    for i:=1 to numberOfParameters-1 do
      begin
        thisParameter:=paramStr(i);
        if (thisParameter[1]='-') then
          case thisParameter[2] of
            'd' : showSearch:=true;
            'l' : lakeLevel:=trunc(getParameterValue(thisParameter));
```

```
            'o' : octopusRadiusFraction:=getParameterValue(thisParameter);
            's' : sobelThreshold:=trunc(getParameterValue(thisParameter));
            'w' : wetFraction:=getParameterValue(thisParameter);
            'n' : nIterations:=trunc(getParameterValue(thisParameter))
          else
            fatalError('Illegal option: -'+thisParameter[2])
          end
        else
          fatalError('Cannot specify more than one file: '+thisParameter)
      end;
    fileName:=picturePath(paramStr(numberOfParameters))
  end;


  procedure initVariables;

  begin
    blockRead(inputFile,buffer,4);
    xSize:=(word(buffer[1]) shl 8)+buffer[2];
    ySize:=(word(buffer[3]) shl 8)+buffer[4];
    startX:=round(getMaxX/2-xSize/2);
    startY:=round(getMaxY/2-ySize/2);
    initGlobals
  end;


begin
  if (paramCount<>0) then
    begin
      getParameters(paramCount);
      if fileExists(fileName) then
        begin
          SVGA_initGraphics(mode800x600);
          SVGA_setBlackAndWhitePalette;
          assign(inputFile,fileName);
          reset(inputFile,1);
          initVariables;
          getMem(image,4*ySize);
          for i:=1 to ySize do
            begin
              blockRead(inputFile,buffer,xSize);
              getMem(image^[i],xSize);
(*
 *  Initializes the in-memory image and displays it on the screen.
 *)
              for j:=1 to xSize do
                begin
                  image^[i]^[j]:=buffer[j];
                  putPixel(startX+j-1,startY+i-1,buffer[j])
                end
            end;
          close(inputFile);
          rectangle(startX,startY,startX+xSize,startY+ySize)
        end
      else
```

```
        fatalError('Image file not found: '+picturePath(fileName))
    end
  else
    fatalError('Usage: eyepos [-dlonsw] file[.pic]')
end;


(*
 *  findPupilPosition
 *
 *  Finds the pupil and computes the time needed.
 *)

procedure findPupilPosition;

var   i                          : word;


  function sec100ths : real;

  begin
    sec100ths:=(6000*(endMinutes     -startMinutes)+
                 100*(endSeconds     -startSeconds)+
                     (end100thSeconds-start100thSeconds))/
               (nIterations/10)
  end;


begin
  if findPupilPoint(pupilPoint) then
    begin
      getTime(startHours,startMinutes,startSeconds,start100thSeconds);
      for i:=1 to nIterations do
        begin
          findPupilPoint(pupilPoint);
          findPosition(pupilPoint,pupilPosition)
        end;
      getTime(endHours,endMinutes,endSeconds,end100thSeconds);
      searchTime:=sec100ths
    end
  else
    fatalError('No pupil was found!');
  markPosition(pupilPosition.x,pupilPosition.y,asStar)
end;


procedure writeData;

begin
  setTextStyle(defaultFont,horizDir,1);
  setTextJustify(leftText,centerText);
  setColor(60);
  outTextXY(startX,startY+ySize+30,'Position         : ('+
            intString(pupilPosition.x)+','+intString(pupilPosition.y)+')');
  outTextXY(startX,startY+ySize+40,'Search time (ms) : '+
```

```
            realString(searchTime))
end;


procedure finishOff;

begin
  closeSearch;
  readKey;
  closeGraph
end;


begin
  init;
  findPupilPosition;
  writeData;
  finishOff
end.
```

# Bibliography


[1] J. Allik, M. Rauk, and A. Luuk. Control and sense of eye movement behind closed eye lids. *Perception*, 10:39–51, 1981.

[2] M. Bach, D. Buois, and B. Fischer. An accurate and linear infrared oculometer. *Journal of Neuroscience Methods*, 9:9–14, 1983.

[3] L. J. Bour, J. A. M. van Gisbergen, J. Bruijns, and F. P. Ottes. The double magnetic induction method for measuring eye movement—results in monkey and man. *IEEE Transactions on Biomedical Engineering*, BME-31(5):419–427, May 1984.

[4] E. O. Brigham. *The Fast Fourier Transform*. Prentice-Hall, 1974.

[5] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986.

[6] R. Deriche. Using Canny's criteria to derive a recursively implemented optimal edge detector. *International Journal of Computer Vision*, pages 167–187, 1987.

[7] R. Eckhorn. Stimulus-evoked synchronizations in the visual cortex: Linking of local features into global figures? In J. Krüger, editor, *Springer Series in Synergetics*, pages 184–224. Springer Verlag, 1991.

[8] R. Eckhorn, R. Bauer, A. Frien, T. Wölbern, and H. Kehr. High frequency (60-90 hz) oscillations in primary visual cortex of awake monkey. *NeuroReport*, 1993. In press.

[9] R. Eckhorn, R. Bauer, W. Jordan, M. Brosch, W. Kruse, M. Munk, and H. J. Reitböck. Coherent oscillations: A mechanism of feature linking in the visual cortex? Multiple electrode and correlation analysis in the cat. *Biologic Cybernetics*, 60:121–130, 1988.

[10] R. Eckhorn, R. Bauer, and F. Rösler. Synchronisation und Kognition. DFG grant application, April 1992.

[11] R. Eckhorn and A. Obermüller. Single neurons are differently involved in stimulus-specific oscillations in cat visual cortex. *Experimental Brain Research*, 1993. In press.

[12] R. C. Gonzalez and P. Wintz. *Digital Image Processing*. Addison-Wesley Publishing Company, Inc., 1987.

[13] P. Haberäcker. *Digitale Bildverarbeitung*. Carl Hanser Verlag, 1991.

[14] L. Lamport. *LaTeX—A Document Preparation System. User's Guide & Reference Manual*, 1986.

[15] G. A. Myers, K. R. Sherman, and L. Stark. Eye monitor: Microcomputer-based instrument uses an internal model to track the eye. *Computer*, pages 14–21, March 1991.

[16] S. Nagao. A non-invasive method for real-time eye position recording with an infrared TV-camera. *Neuroscience Research*, (8):210–213, 1990.

[17] J. P. H. Reulen and L. Bakker. The measurement of eye movement using double magnetic induction. *IEEE Transactions on Biomedical Engineering*, BME-29:740–744, November 1982.

[18] D. A. Robinson. A method of measuring eye movement using a scleral search coil in a magnetic field. *IEEE Transactions on Biomedical Engineering*, BME-10:137–145, October 1963.

[19] A. Rosenfeld and A. C. Kak. *Digital Picture Processing*. Academic Press, 1976.

[20] A. Rosenfeld and J. S. Weszka. Picture recognition. In K. S. Fu, editor, *Digital Pattern Recognition*, volume 10 of *Communication and Cybernetics*, chapter 5. Springer Verlag, 1980.

[21] W. Singer and C. M. Gray. Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. *PNAS USA*, 86:1698, 1989.

[22] W. Singer, C. M. Gray, P. Koenig, and A. Engel. Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. *Nature*, 338:334–337, 1989.

[23] R. Wagner and H. L. Galiana. Evaluation of three template matching algorithms for registering images of the eye. *IEEE Transactions on Biomedical Engineering*, 39(12):1313–1319, December 1992.